

Training Human-AI Teams

by

Hussein Mozannar

Submitted to the Institute for Data, Systems, and Society
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOCIAL AND ENGINEERING SYSTEMS AND STATISTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© 2024 Hussein Mozannar. This work is licensed under a [CC BY-NC-ND 4.0](#) license.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Hussein Mozannar
Institute for Data, Systems, and Society
May 20, 20224

Certified by: David Sontag
Professor of Computer Science, Thesis Supervisor

Certified by: Arvind Satyanarayan
Associate Professor of Computer Science, Thesis Supervisor

Certified by: Elena Glassman
Assistant Professor of Computer Science, Thesis Supervisor

Certified by: Eric Horvitz
Chief Scientific Officer, Microsoft, Thesis Supervisor

Accepted by: Fotini Christia
Program Chair, Social and Engineering Systems

Training Human-AI Teams

by

Hussein Mozannar

Submitted to the Institute for Data, Systems, and Society
on May 20, 20224 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOCIAL AND ENGINEERING SYSTEMS AND STATISTICS

ABSTRACT

AI systems are augmenting humans' capabilities in settings such as healthcare and programming, forming human-AI teams. To enable more accurate and timely decisions, we need to optimize the performance of the human-AI team directly. In this thesis, we utilize a mathematical framing of the human-AI team and propose a set of methods that optimize the AI, the human, and the interface in which they communicate to enable better team performance. We first show how to provably train AI classifiers that complement humans and can defer the decision to humans when it is best to do so. However, in specific settings, AI cannot autonomously make decisions and thus only provides advice to humans. In that case, we build onboarding procedures that train humans to have an accurate mental model of the AI to enable appropriate reliance. Finally, we study how humans interact with large language models (LLMs) to write code. To understand current inefficiencies, we developed a taxonomy to categorize programmers' interactions with the LLM. Motivated by insight from the taxonomy, we leverage human feedback to know when to best display LLM suggestions.

Thesis supervisor: David Sontag
Title: Professor of Computer Science

Thesis supervisor: Arvind Satyanarayan
Title: Associate Professor of Computer Science

Thesis supervisor: Elena Glassman
Title: Assistant Professor of Computer Science

Thesis supervisor: Eric Horvitz
Title: Chief Scientific Officer, Microsoft

Acknowledgments

I first want to thank the Institute for Data, Systems, and Society at MIT for admitting me as a PhD student and funding my first year. Notably, I want to thank Ali Jadbabaie, Munther Dahleh, and Beth Milnes. I am incredibly grateful for the supervision of my advisor David Sontag. David and I started working together informally at the end of my first semester at MIT, and it didn't take long for me to figure out that David was not only brilliant but incredibly supportive on a personal basis. It was incredibly fun to work with David, from long walks around Boston to whiteboard sessions in E25, which led to us doing creative and impactful research. I am also incredibly thankful to my thesis committee: Arvind Satyanarayana, Eric Horvitz, and Elena Glassman. Arvind taught me most of what I know about HCI, and helped me develop a more critical and thoughtful research method. By chance of luck, I got to work with Eric during my internship at Microsoft, and I got to witness an incredibly creative researcher who has incredible joy in his work, which I hope to carry on. Elena has been a great sounding board for my research ideas on programming and helped me find my rigor in HCI.

I have been very lucky to have only amazing collaborators during my PhD. Mesrob Ohannessian and Nati Srebro, who supported me as an undergrad, helped me get a head start on my first semester in the PhD. Mohammad-Amin Charusaie and Samira Samadi invited me to work with them on learning to defer, and I have great memories of late nights and early mornings working on proofs with Amin. I lured Hunter Lang to also work with me on learning to defer and had a lot of fun proving computational hardness results on the whiteboards at MIT. I was very fortunate to intern in Saleema Amershi's group with Gagan Bansal and Adam Fournery at Microsoft Research and had the most productive summer of my career working on Copilot. I am also lucky that they decided they want more of me, so I am looking forward to our future work. I have been very lucky to mentor incredibly talented students. Working with Yuria Utsumi and Irene Chen on pregnancy risk management with IBC was one of the most impactful things of my PhD. It was a lot of fun working with Jimin J Lee on the onboarding project. Notably, I want to acknowledge the support and funding of MIT-IBM Watson AI Lab for most funding the last 3 years of my PhD. For the past three years, every Thursday at 1 pm ET, I was lucky to work with Dennis Wei, Subhro Das, and Prassana Sattigieri at MIT-IBM. This collaboration has been incredibly fruitful, leading us to co-author four papers together. Finally, I want to thank my most recent collaborators, Valerie Chen and Ameet Talwalkar; while the work we did together did not make it to this thesis, it was very helpful for my overall research.

I want to thank the numerous friends and lab mates at MIT and Boston who have I have been lucky to have: Andy, Arnab, Hamaad, Erin, Sarah, Eric, Manon, Siri, Bernardo, Tony, Rabih, Alaa, Abbas, Mohammed(s), Monica, Hunter, Irene, Ilker, Chandler, Sharron, Zeshan, Christina, Rebecca, Mike. I also want to thank my friends from back home: Nadeem, Bassem, Rawad, Firas, Wassim,

Roula, Elie, Karl, Carla, Anthony, and Rami. I want to especially thank Pratibha for her unwavering support and encouragement during my PhD which I am forever grateful for. Most important of all, I want to thank my siblings and parents from the bottom of my heart for their support and encouragement.

Contents

Title page	1
Abstract	3
Acknowledgments	5
List of Figures	13
List of Tables	21
1 Introduction	25
1.1 Overview	25
1.2 Deferral Systems	27
1.3 AI-Assisted Decision Making	29
1.4 Interactive Human-AI Collaboration	31
I Conditional Delegation with AI: Learning to Defer	33
2 Sample Efficient Learning to Defer	35
2.1 Introduction	35
2.2 Related Work	36
2.3 Problem Setting	37
2.4 Staged Learning of Classifier and Rejector	39
2.4.1 Model Complexity Gap	39
2.4.2 Data Trade-offs	40
2.5 Surrogate Losses For Joint Learning	42
2.5.1 Family of Surrogates	42
2.5.2 Theoretical Properties of Surrogate	44
2.6 Active Learning for Expert Predictions	45
2.6.1 Theoretical Understanding	45
2.6.2 Disagreement on Disagreements	47
2.7 Experimental Illustration	48

3	Exact Algorithms for Learning to Defer	53
3.1	Introduction	53
3.2	Related Work	55
3.3	Learning with Deferral: Problem Setup	56
3.4	Computational Complexity of Learning with Deferral	57
3.5	Mixed Integer Linear Program Formulation	59
3.6	Realizable Consistent Surrogate	61
3.6.1	Consistency vs Realizable Consistency	61
3.6.2	Derivation of Surrogate	63
3.6.3	Theoretical Guarantees	64
3.6.4	Underfitting The Target	65
3.7	Experiments	66
3.7.1	Human-AI Deferral Benchmark	68
3.7.2	Synthetic and Semi-Synthetic Data	69
3.7.3	Realistic Data	70
II	AI-Assisted Decision Making: Onboarding	73
4	Teaching Humans When To Defer to a Classifier via Exemplars	75
4.1	Introduction	75
4.2	Related Work	76
4.2.1	Relation to Learning to Defer	76
4.2.2	Further Related Work	77
4.3	Problem Setup	79
4.4	Human Mental Model	81
4.5	Teaching a Student Learner	83
4.6	Experimental User Study	86
4.6.1	Experimental Preliminaries	86
4.6.2	Simulated Users	87
4.6.3	Crowdsourced Experiments Details	90
4.6.4	User Study Observations and Results	92
4.7	Additional Synthetic Experiments	94
4.8	Discussion	95
5	Effective Human-AI Teams via Learned Natural Language Rules and Onboarding	97
5.1	Introduction	97
5.2	Related Work	99
5.3	AI Assisted Decision Making	100
5.4	Learning Rules for Human-AI Cooperation: IntegrAI	103
5.4.1	Region Discovery Algorithm	104
5.4.2	Region Description Algorithm	105
5.5	Onboarding and Recommendations to Promote Rules	108
5.6	Method Evaluation	111
5.7	User Studies to Evaluate Onboarding Effect	113

III Interactive Human-AI Collaboration: Case Study in LLM-Assisted Programming 119

6	Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming	121
6.1	Introduction	121
6.2	Background and Related Work	124
6.3	Copilot System Description	126
6.3.1	Influences of CodeRec on Programmer’s Activities	127
6.3.2	Programmer Activities in Telemetry Segments	127
6.4	A Taxonomy for Understanding Programmer-CodeRec Interaction: CUPS	128
6.4.1	Creating the Taxonomy	128
6.4.2	Taxonomy of Telemetry Segments	131
6.5	CUPS Data Collection Study	132
6.5.1	Procedure	132
6.5.2	Participants	133
6.6	Understanding Programmer Behavior with CUPS: Main Results	134
6.6.1	Aggregated Time Spent in Various CUPSs	136
6.6.2	Patterns in Behavior as Transitions Between CUPS States	137
6.6.3	Programmers Often Defer Thought About Suggestions	140
6.6.4	CUPS Attributes Significantly More Time Verifying Suggestions than Simpler Metrics	141
6.6.5	Insights About Prompt Crafting	142
6.6.6	Post-Study Survey Answers	144
6.7	Limitations	145
7	When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming	147
7.1	Introduction	147
7.2	Related Work	149
7.3	Problem Setting	149
7.4	Theoretical Formulation of Suggestion Utility	151
7.5	Conditional Suggestion Display From Human Feedback	154
7.6	Experiments	155
7.6.1	Dataset and Feature Engineering.	156
7.6.2	Model Evaluation	157
7.6.3	Retrospective Evaluation of CDHF	158
7.7	Which Suggestion to Show?	160
8	Conclusion	163
A	Additional Information for Chapter 2	169
A.1	Proof of Theorem 1	169
A.2	Proof of Proposition 1	179
A.3	Proof of Proposition 2	185

A.4	Proof of Theorem 2	186
A.5	Proof of Theorem 3	190
A.6	Proof of Proposition 3	196
A.7	An example on which CAL algorithm fails	199
A.8	Proof of Theorem 4	200
A.9	Experimental Details	201
B	Additional Information for Chapter 3	203
B.1	Practitioner’s guide to our approach	203
B.1.1	MILP	203
B.1.2	Realizable Surrogate	206
B.2	MILP	207
B.2.1	Verification	207
B.3	Experimental Details and Results	208
B.3.1	Baseline Implementation	208
B.3.2	Training Details	209
B.3.3	Synthetic Data	209
B.3.4	NIH Chest X-ray	211
B.3.5	CIFAR-10H	212
B.4	Deferred Proofs and Derivations	212
B.4.1	Related Work	212
B.4.2	Section 3.4 (Hardness)	212
B.4.3	Section 3.5 (MILP)	219
B.4.4	Section 3.6 (Realizable Surrogate)	221
C	Additional Information for Chapter 4	227
C.1	Extended Related Work	227
C.2	Theoretical Results and Proofs	230
C.2.1	Further Derivations	230
C.2.2	Proofs	231
C.2.3	Hardness result	233
C.2.4	Efficient Implementation of Greedy Selection	234
C.3	SAE Model Error Analysis	235
C.3.1	Factors of difference	235
C.3.2	Embedding clustering	238
C.4	Synthetic Experiments Details and Results	241
C.4.1	Misspecification results	241
C.5	Additional Synthetic Experiments	242
C.5.1	CIFAR-10	242
C.5.2	Gaussian Data Illustration	245
C.6	Crowdsourced Experiments Details and Results	247
C.6.1	Experiment Details	247
C.7	Extended Discussion	248
C.8	User Interface Screenshots	251

D	Additional Information for Chapter 5	265
D.1	Extended Related Work	265
D.2	Region Finding Algorithm - Details	267
D.3	Region Description Algorithm - Details	269
D.4	Onboarding and Recommendations to Promote Rules - Details	271
D.5	Method Evaluation - Details	272
D.6	User Studies - Details	276
D.6.1	BDD Study	276
D.6.2	MMLU Study	277
D.6.3	Screenshots of User Study Interface for BDD	279
D.6.4	Screenshots of User Study Interface for MMLU	286
E	Additional Information for Chapter 6	289
E.1	Programmer Behavior by Task	289
E.2	Predicting CUPS from Telemetry	289
E.3	Details User Study	291
E.3.1	Interfaces	291
E.3.2	Task Instructions	292
E.3.3	Survey Questions Results	297
E.3.4	Full User Timelines	301
E.3.5	Full CUPS Graph	303
F	Additional Information for Chapter 7	305
F.1	Extended Related Work	305
F.2	Derivation of \mathbb{P}^*	306
F.3	Model Evaluation and Analysis	307
F.4	Which Suggestion to Show: Plots	310
	References	313

List of Figures

1.1	The Human-AI Team: an overview of the three thesis parts showcasing opportunities for training the human-AI team: the human, the AI, and the interface.	26
1.2	Human-AI interaction settings studied in this thesis. In subfigure (a), the AI-assisted decision-making setting where onboarding helps humans build better mental models of AI (part II). In (b), an AI rejector dictates who should predict between the human and AI (part I). Finally, in (c) is my work on programmers interacting with LLMs and interventions to selectively display suggestions (part III).	27
2.1	Illustration of our active learning algorithm Disagreement on Disagreements (DoD) (1). At each round, we compute the disagreement set for our predictors of the human label disagreement, we then query the human for their prediction on these points. After we learn the expert error boundary, we then learn a consistent classifier-rejector pair.	45
2.2	Difference of accuracy between joint learning and staged learning of the classifier-rejector pair (y-axis is log scale of number of parameters).	49
2.3	Performance of joint learning and staged learning as we increase the ratio of the data labeled by the expert $\frac{n_l}{n_u+n_l}$	50
2.4	Error of the DoD algorithm compared to staged and joint learning for increasing number of training data that are labeled by human.	51
3.1	The learning to defer setting with the RealizableSurrogate illustrated in the application of making predictions for chest X-rays.	53
3.2	The realizable LWD-H setting illustrated. The task is binary classification with labels $\{o, +\}$; the human is perfect on the green-shaded region, and the data outside the green region is linearly separable. As a result, the optimal classifier and rejector obtain zero error. Assumption 2 is illustrated graphically as well as the MILP variables of equations (3.9)-(3.14).	57
3.3	Accuracy vs coverage (fraction of points where classifier predicts) plots across the real world datasets showcasing the behavior of our method and the baselines. On each plot, we showcase the test accuracy of each method with a large marker, with the curve representing varying the rejector threshold on the test set. To achieve different levels of coverage, we sort the rejection score for each method on the test set and vary the threshold used, for RealizableSurrogate the rejector is defined as $r(x) = \mathbb{I}_{g_{\perp}(x) - \max_y g_y(x) \geq c}$ where the optimal solution is at $c = 0$ and we vary $c \in \mathbb{R}$ to obtain the curve.	67

3.4	(a) Test performance of the different methods on synthetic data as we increase the training data size and repeat the randomization over 10 trials to get standard errors. (b) Test performance on the semi-synthetic CIFAR-K dataset vs. the number of classes K for which the expert is perfect.	68
3.5	Sensitivity of the RealizableSurrogate to the hyperparameter α . We vary the hyperparameter α in the RealizableSurrogate surrogate loss and show the different metrics including overall accuracy, accuracy when we defer, accuracy when we don't defer, and finally coverage.	70
4.1	The AI assisted decision making pipeline. The AI first sends to the human a message A , then the human decides with their rejector $r(Z, A)$ if they should follow the AI's advice and predict $\pi_Y(X)$ or they should predict on their own using $h(Z, A)$	79
4.2	Illustration of human rejector on toy example. The task is classification with labels $\{0, +\}$, the human prediction h is the blue line and the prior g_0 is the shaded orange region surrounding the boundary. Points in red is where the human is incorrect, in blue correct and in black point deferred to the AI. The AI is assumed to be correct on examples far from the human boundary. The human receives a teaching example z_1 (in green) with radius γ_1 . Also shown are the two contrasting examples z_{j1} and z_{jk} (in pink) that define the region.	83
4.3	Teaching set size versus the negative difference between the human's learner test accuracy under the different methods compared to ORACLE. We plot the average result across 10 trials and standard deviation as error bars.	88
4.4	On the left in subfigure (a) is the testing interface shown for an example. This is the same interface that is also shown at the beginning of each teaching example. After the human predicts and we are in the teaching phase, we show them the correct answer and transition to the interface in subfigure (b) that shows the two supporting examples for the example in (a), the top weighted words in the region and asks the user to write down their rule for the example.	90
4.5	Synthetic experiment on CIFAR-10, showing difference between the performance of the methods and ORACLE (defined as taking the optimal decision at test time) for expert $k = 6$	95
5.1	The proposed onboarding approach with the IntegrAI algorithm.	98
5.2	The setting of AI-assisted decision making studied in this work. We show an example of an AI system providing assistance to a human driver to inform them about traffic lights in a low visibility situation. The AI provides advice to the human who then incorporates it to make a final decision.	101
5.3	The IntegrAI-Describe algorithm illustrated. To obtain a description for a region of points, the algorithm first samples a set of points inside and outside the region and gets a description from an LLM that contrasts inside versus outside (Step 0). We then embed the obtained description in our cross-modal embedding space (Step 1) and find counterexamples to that description, both points outside the region with high similarity to the description and points inside the region with low similarity to the description (Step 3). The process is repeated for as many rounds as necessary (Step 4).	108

5.4	Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators as we vary the number of regions maximally allowed for each integrator on the BDD dataset.	111
5.5	(a) Interface for humans to detect a traffic light in images from BDD dataset in the presence of AI’s prediction, confidence score, and bounding box and (b) interface for humans to answer multiple choice questions from MMLU dataset with AI’s prediction and explanation.	114
6.1	Profiling a coding session with the CodeRec User Programming States (CUPS). In (a) we show the operating mode of CodeRec inside Visual Studio Code. In (b) we show the CUPS taxonomy used to describe CodeRec related programmer activities. A coding session can be summarized as a timeline in (c) where the programmer transitions between states.	123
6.2	Schematic of interaction telemetry with Copilot as a timeline. For a given coding session, the telemetry contains a sequence of timestamps and actions with associated prompt and suggestion features (not shown).	126
6.3	Taxonomy of programmer’s activities when interacting with CodeRec– CUPS.	128
6.4	Screenshot of retrospective labeling tool for coding sessions. Left: Navigation panel for telemetry segments. Right: Video player for reviewing video of a coding session. Bottom: Buttons and text box for labeling states.	130
6.5	Visualization of CUPS labels from our study as timelines, a histogram, and a state machine.	135
6.6	Myriad of CUPS patterns observed in our study.	139
6.7	Illustration of a coding scenario with Copilot where the programmer may choose to defer verifying a suggestion (‘Deferring Thought’). Here, Copilot suggests an implementation for the class Logistic Regression line-by-line (illustrated from left to right). And the programmer may need to defer verifying intermediate suggestion of <code>self.cost</code> (middle screenshot) because the method that implemented it is suggested later (right screenshot).	140
6.8	Illustration of one of the adjustments required for measuring the total time a programmer spends to verify a suggestion. Here, when a programmer defers thought for a suggestion, they spend time verifying it after accepting it and may also have to wait beforehand for the suggestion to be shown.	143
7.1	Operating mode of CodeRec inside Visual Studio Code showing how CDHF influences the interaction by selectively hiding certain suggestions. Data collected by the interaction is stored in telemetry and used to train CDHF to create a feedback loop.	147
7.2	Schematic of telemetry with CodeRec as a timeline. For a given coding session, the telemetry contains a sequence of timestamps and actions with associated prompts and suggestions.	150
7.3	Graphical depiction of analysis of Proposition 6 when the latency is zero. The y-axis shows total time and the x-axis is the programmer’s probability of accepting $\mathbb{P}(A = \text{accept} X, S, \phi)$. At probability \mathbb{P}^* , showing and not showing the suggestion have equal time cost.	153

7.4	Features used to build action prediction model in Experiments, including from the suggestion, prompt, and session.	156
7.5	Evaluation of CDHF for selectively hiding suggestions. For a given constraint on FNR (accuracy when a suggestion is hidden) on the x-axis, we show on the y-axis the fraction of the total suggestions we can hide while guaranteeing the desired FNR. We plot these curves while varying how often the decision is made generating suggestions ($R:=\mathbb{E}[r(x)]$, when $R=0$, we generate the suggestion then decide to hide or not, when $R=1$, we decide to hide without knowing the suggestion).	159
B.1	(Test performance of the different methods on realizable synthetic data as we increase the training data size and repeat the randomization over 10 trials to get standard errors on uniform data.	210
B.2	(Test performance of the different methods on unrealizable ($d = 10, p_m = 0.1, p_{h0} = 0.4, p_{h1} = 0.1$, Gaussian distribution with 20 clusters) synthetic data as we increase the training data size.	210
B.3	Runtime of the MILP on the realizable synthetic data with uniform data distribution. Note that the test accuracy of the MILP is demonstrated in Figure 3.4a and the MILP always reaches 0 training error across the different data dimensions and training set sizes.	211
B.4	NIH Chest X-ray results on the two remaining tasks with the baselines and our method and red with circle markers. We see that all methods aren't able to obtain a performance of a human-AI team with better performance than the human, our method on both tasks defers to the human.	211
B.5	On CIFAR-10H, classifier accuracy on non-deferred set and human accuracy when deferred vs coverage (fraction of points where classifier predicts).	212
B.6	Data Distribution for our example: the data consists of four regions R_0, R_1, R_2 and R_3 . Each region respectively has mass $1/4 + \alpha, 1/4, 1/4 - \alpha, 1/4$. Each region respectively has label $Y = 0, Y = 1, Y = 0, Y = 2$. The Human is only accurate on Region 0.	224
C.1	Performance across lengths of passages in terms of words. First bin contains very little samples to be significant.	236
C.2	Performance across number of supporting sentences. Black bars indicate 95% confidence interval around the mean, the x axis is: (number of sentences, number of examples with that many sentences)	237
C.3	Performance across LDA topics	237
C.4	Performance (left EM, right F1) across question words	238
C.5	Performance (left EM, right F1) across model embeddings clusters.	238
C.6	Performance across passage embeddings clusters. No differences emerge significantly.	239
C.7	Performance across question embeddings clusters.	239
C.8	Performance across answer embeddings clusters.	240
C.9	Difference in Oracle accuracy at teaching size @ $T=30$ for the DOUBLE-GREEDY method assuming an error in h by δ in setting B.	241

C.10	Comparing a 1-nearest neighbor rejector model to the radius nearest neighbor model introduced in Assumption 4 for expert $k = 6$. The "1-NN" line is obtained by first obtaining T points using K-medoids and then running a 1-NN rejector on these points with the label assigned to each point being the optimal deferral decision r_i . We can see that 1-NN struggles with less than 6 examples, but then reaches a steady state that has the same error as the radius nearest neighbor model. The effectiveness of the radius nearest neighbor model when the teaching set is very small is due to the local nature of each update with the addition of a teaching example.	242
C.11	Performance of the AI-Behavior baseline as we vary the parameter K : the AI-Behavior baseline uses a K -nearest neighbor rejector and at each teaching step selects the point that best reduces the error of the rejector at detecting the AI's errors. We show results for the human expert $k = 6$ with the consistent radius strategy $\alpha = 1$. We can see that the parameter K has little effect and thus we use a natural choice of $K = 6$	242
C.12	Extended legend: Varying the human parameter k (number of classes human can classify) and plotting the difference to oracle accuracy for all the baselines when using the consistent radius strategy including the surrogate-loss learning to defer method of [26] at 3 different teaching set sizes.	243
C.13	Extended legend: Varying the human parameter k (number of classes human can classify) and plotting the difference to oracle accuracy for all the baselines when using DOUBLE-GREEDY including the surrogate-loss learning to defer method of [26] at 3 different teaching set sizes.	244
C.14	Teaching complexity plot for synthetic Gaussian data setup. The x-axis shows the difference in test human accuracy between our method and the baselines. Plotted are the averages over the 100 trials along with 95% confidence interval error bars for the average.	245
C.15	Extended legend: blue dots indicate a correct decision while red dots indicate mistakes. Points with an "x" are labels 1 while points with an "o" are labels 0 (in the Y space). The lines labeled human and machine are the respective classifiers.	246
C.16	Consent form to be confirmed before entering experiment	251
C.17	Information collected about workers prior to experiment. MTurk worker ID was only saved for cross-checking and then deleted.	252
C.18	First step of the tutorial introducing the task	253
C.19	Second step of the tutorial solving without AI help	254
C.20	Third step of the tutorial solving with AI help	255
C.21	Teaching instructions	256
C.22	Teaching initial example to be solved by the human.	257
C.23	Feedback shown after human solves the example along with supporting examples.	258
C.24	Top words for the teaching example along with instructions for lesson writing	259
C.25	The LIME-Teaching user teaching introduction	259
C.26	The LIME-Teaching feedback after answering teaching question.	260
C.27	The LIME-Teaching teaching introduction to second part of the teaching phase	260
C.28	The LIME-Teaching user interface of the second part of the teaching phase where users observe examples and the AI answers.	261
C.29	Interface during testing.	262

C.30	Questions collected after workers complete experiment for the Teaching condition.	263
D.1	Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the MS-COCO dataset.	274
D.2	Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the MMLU dataset.	274
D.3	Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the Dyanasent dataset.	275
D.4	Consent Form	280
D.5	User Information Collection	280
D.6	Practice Task instructions	281
D.7	Prediction without AI interface	281
D.8	Instructions for the onboarding phase	282
D.9	Model card information shown during onboarding.	282
D.10	Prediction with AI interface (AI predicts no traffic light)	282
D.11	Feedback shown during onboarding phase after human predicts.	283
D.12	Feedback shown during onboarding phase after human predicts (sets of examples from region)	283
D.13	Prediction with AI interface (AI predicts there is a traffic light)	283
D.14	Feedback shown during onboarding phase after human predicts (correct feedback)	284
D.15	Testing phase instructions	284
D.16	Testing phase instructions that include AI-integration recommendation	284
D.17	Prediction interface with AI and with AI-integration recommendations.	285
D.18	Model Card for MMLU study	286
D.19	Prediction Interface for MMLU study	286
D.20	Feedback shown to user during teaching phase	287
D.21	Prediction Interface for MMLU study with AI-integration recommendations.	287
E.1	Screenshot of Labeling Tool represented in Figure 6.4	291
E.2	Screenshot of Virtual Machine interface with VS Code	292
E.3	Data Manipulation Task.	292
E.4	Algorithmic Problem Task.	293
E.5	Data Analysis Task.	293
E.6	Classes and Boilerplate Code Task.	294
E.7	Logistic Regression Task	294
E.8	Editing Code Task	295
E.9	Machine Learning Task	295
E.10	Writing Tests Task	296
E.11	User Study Survey results (1)	297
E.12	User Study Survey results (2)	298
E.13	User Study Survey results (3)	299
E.14	User Study Survey results (4)	300

E.15	Participants timelines for the first 10 minutes of their sessions (P1 to P10)	301
E.16	Participants timelines for the first 10 minutes of their sessions (P11 to P21)	302
E.17	CUPS diagram with all transitions shown that occur with probability higher than 0.05	303
F.1	Calibration curve for the XGBoost stage 2 model.	308
F.2	Sample complexity analysis of the XGBoost stage 2 model when trained on a fraction of the training data and plotting the AU-ROC on the full test set.	309
F.3	Feature importance for the seven highest-rated unique features of the model for predicting the likelihood of accepting a suggestion. The feature importance is in terms of the F score which counts how often a feature was split on in the tree ensemble.	309
F.4	Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 0$ (docstring only)	310
F.5	Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 1$ (docstring + first line of solution)	311
F.6	Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 2$ (docstring + first two lines of solution)	311
F.7	Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 3$ (docstring + first three lines of solution)	312

List of Tables

3.1	Datasets used for our benchmark for learning with deferral to humans. We note the total number of samples n , the target set size $ \mathcal{Y} $, the number of tasks in each dataset (a task is a set of human and target labels), the human expert where 'random annotator' means that for each point we have multiple human annotations and we let the target be a consensus and the human label be a random sample while 'separate human annotation' means that the human label is completely separate from the label annotations and finally the model class for both the classifier and rejector.	68
3.2	Test accuracy of the operating point of the different methods on the datasets tested on. The baselines are L_{CE} [26], Ψ_{OVA} [62], Selective Prediction (SP), Compare Confidence (CP) [25], DIFT [27] and MoE [28].	71
4.1	Comparison on different dimensions between the teaching to defer framework in this paper (TTD) and the learning to defer framework (LTD) from [26], [28], [87].	78
4.2	Test Accuracy gap between DOUBLE-GREEDY and ORACLE at teaching set of size 30 under various conditions. This is performed under setting B.	89
4.3	Comparison of the metrics between our teaching condition (split into all participants, those who gave accurate lessons (acc) and those who didn't (inacc), see description below), the No-teaching+AI-prediction condition and LIME teaching. Shown are averages across all participants with 95% confidence interval error bars. The F1 of the AI alone in this setting is 46.7%; we did not separately measure the F1 of the human in isolation.	91
4.4	Comparison of the metrics on clusters that were seen during teaching with our method (ID for in distribution) compared to performance on clusters that were not seen during teaching (OOD for out of distribution). We also show the performance of the no-teaching baselines on the two cluster sets as a reference point. The errors on the OOD estimates are much higher as there are much fewer samples in the not-seen clusters.	92
4.5	Synthetic experiment on CIFAR-10, showing the test Accuracy for our method DOUBLE-GREEDY at different teaching set sizes and learning to defer baselines.	95
5.1	Human-AI Card presented to the human as part of onboarding	109
5.2	The exact Human-AI card used in the user study for BDD.	110
5.3	Error (\downarrow) on the test set (in %) of the human-AI system when following integrators resulting from different region discovery methods with 10 regions on the different non-synthetic datasets.	112

5.4	Clustering metrics (Adjusted Rand index [161] ↑, Fowlkes–Mallows index [162] ↑) of the regions (10 regions) found by the different methods on the synthetic dataset setup.	112
5.5	Evaluation of our region description algorithm (Algorithm 4) on selected subsets of MS-COCO where the different algorithms try to describe a set of images that all contain a given object. For example, a region may be defined by images containing the object “apple”. Then we compare the descriptions resulting from the different algorithms to the description “apple”.	113
5.6	Results from our user studies for BDD. For accuracy, time per example, and AI-reliance we report mean and standard error across participants. The "Test vs H-AI" row reports the adjusted p-value and t-test statistic for a two-sample t-test between the human+AI condition and the other conditions (columns).	116
5.7	Results from our user studies for MMLU.	116
6.1	Description of each state in CodeRec User Programming States (CUPS).	129
6.2	Description of the coding tasks given to user study participants and task assignment. Participants were randomly allocated to tasks for tasks which they had familiarity with.	133
6.3	We compute the percentage of suggestions accepted given the programmer was in the CUPS state while the suggestion is being shown (% Ss accepted while shown). We compute the percentage of suggestions accepted given the programmer was in the CUPS state before the suggestion is shown, the state just before the one where the suggestion is shown (% Ss accepted before S is shown). We compute the standard error for the acceptance rate (%).	141
B.1	Training details for each dataset, we use the Adam optimizer [227] and AdamW [226]	209
C.1	Performance on the dev set without yes/no questions.	235
C.2	Performance based on question types.	235
C.3	Cluster main theme (manually obtained) and top 3 Wikipedia categories of examples in clusters for the AI used in the MTurk study.	248
C.4	Example of lessons that users in the Ours-Teaching condition wrote during the teaching phase. We show examples of the lessons on the first 3 examples in the teaching phase and separate the participant lessons into 4 categories: participants who wrote accurate lessons, participants who wrote irrelevant lessons (not relevant to the question or required no effort to write), participants who wrote complex lessons that don’t pertain to the example topic and finally participants who wrote narrow lessons that are on topic but only apply to the example and not the neighborhood of the example.	250

D.1	Datasets for "Learning Accurate Integrators (Aim 1)". We note the total number of samples n , the target set size $ \mathcal{Y} $, the human expert finally the model of the AI. When we note human "XX% accurate", this indicates a synthetic human model that is accurate uniformly at random with probability XX%. For DynaSent, the AI model is a a pre-trained sentiment analysis roBERTa-base model [160] on Twitter data and achieves 75% accuracy. For both BDD and MS-COCO we blur the images using a Guassian Blur with scale 21 and variance 5.	273
D.2	Region descriptions found by IntegrAI for the BDD user study.	277
D.3	Region descriptions found by IntegrAI for the MMLU user study.	279
E.1	Acceptance rate and the top three CUPS states in terms of time spent as a fraction of session time for each of the tasks. We include standard errors of the acceptance rate aggregated across participants.	304
F.1	Comparison of different classifiers on the test set based on AU-ROC, accuracy, and macro f1 for the full model to predict programmer suggestion acceptance. FCNN refers to a fully connected neural network.	308

Chapter 1

Introduction

1.1 Overview

As artificial intelligence (AI) becomes more ubiquitous and sophisticated, humans are increasingly working alongside AI systems to accomplish various tasks, ranging from medical diagnosis [1], [2], content moderation [3] to writing [4] and programming [5]. The promise of AI is to enhance human performance and efficiency by providing fast and accurate solutions. Furthermore, the emergence of assistants based on large language models (LLMs), such as ChatGPT [6] and Copilot [7], holds the potential to enhance productivity in various industries. What enables these applications is work that bridges the interface between humans and AI, which this thesis focuses on. For such human-AI teams to function effectively, the human must have an accurate mental model of the AI to understand when to trust its advice. Moreover, the AI also has to model the human's expertise and behavior to be an effective partner. To make progress, I utilize a mathematical framing of the problem setting informed by a deep understanding of practice. This framing allows me to develop provable methods for human-AI teams leveraging techniques from machine learning (ML) and human-computer interaction (HCI).

To start, a **human-AI team** is an abstraction of a system that consists of both a human and an artificially intelligent agent (the AI) that aims to solve a concrete task. This task could be defined as making a classification decision (medical diagnosis), producing an artifact that meets certain specifications (code that passes unit tests), or providing a free-form answer to a question. We assume that given an attempted task solution, we can measure the quality of the solution as well as measure metrics around the effort in generating the solution, for instance, time. The human and the AI interact in this system according to a certain interaction scheme. For instance, the AI may provide a suggestion to the human, who then incorporates the suggestion to solve the task; this is illustrated in Figure 1.1. As an example, suppose a radiologist wishes to diagnose a patient given their chest X-ray. The AI provides a candidate diagnosis and assigns a confidence score to their

certainty about the diagnosis. The radiologist can observe this suggestion and then make a final diagnosis. This type of interaction is denoted as AI-assisted decision-making; we will explore this further in this thesis alongside other interaction schemes.

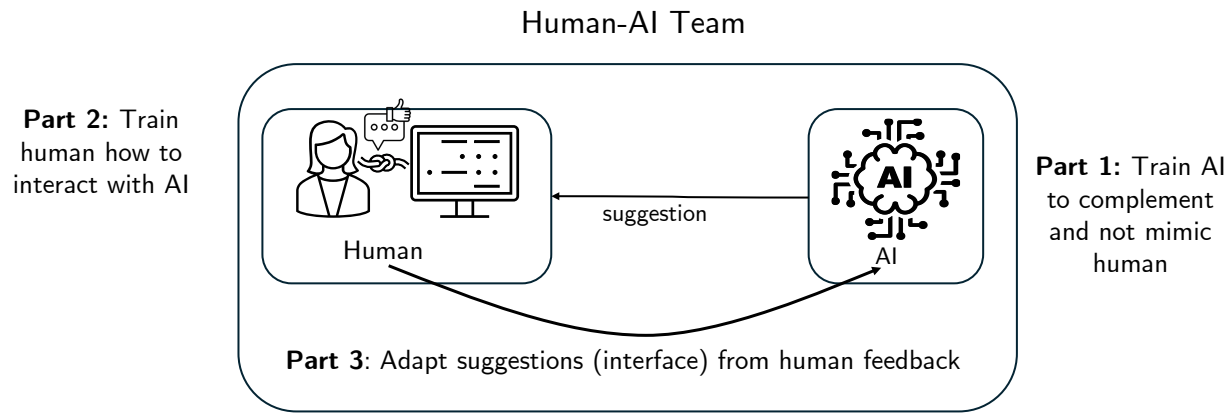


Figure 1.1: The Human-AI Team: an overview of the three thesis parts showcasing opportunities for training the human-AI team: the human, the AI, and the interface.

The human-AI team consists of four components: the interaction scheme, the human, the AI, and the interface in which they communicate. In our radiology example, the four components are: the interaction scheme is having the AI provide their suggestion to the human first, the human is the radiologist, the AI we assumed is a model that is trained to make predictions and provide a confidence score, and finally, the interface is how the AI's suggestion is communicated, e.g., is the confidence score displayed as a percentage or communicated as being "low" or "high". These four components present targeted opportunities to optimize the performance of the human-AI team, which we will explore in this thesis.

The first challenge is that current AI models are often trained without considering the human in the loop. Since the AI is part of a team, we should optimize the AI model to increase team performance rather than creating the best-performing AI independently. In the first part of this thesis [I](#), we build a set of methods to allow AI classifiers to complement their human counterparts and to defer when it is best to do so in [Chapters 2 and 3](#). This enables the AI to be the best teammate possible instead of the best model in isolation. The second challenge is that the human prior mental model of the AI is usually incorrect, leading to inefficient collaboration. In settings where AI provides advice to a human, such as providing a radiologist with a candidate diagnosis, we propose in [part II](#) of this thesis a set of procedures to explicitly teach humans how to collaborate with the AI: when to trust, ignore, or modify the AI advice. Moreover, the AI has to improve from the feedback of the human continuously interacting with it to allow for alignment with human needs. For AI models that provide code suggestions to programmers, we show in [part III](#) of this thesis, how to

improve the interface and the suggestions based on feedback from programmer actions. We will now discuss our contributions and the problem settings for each part of the thesis.

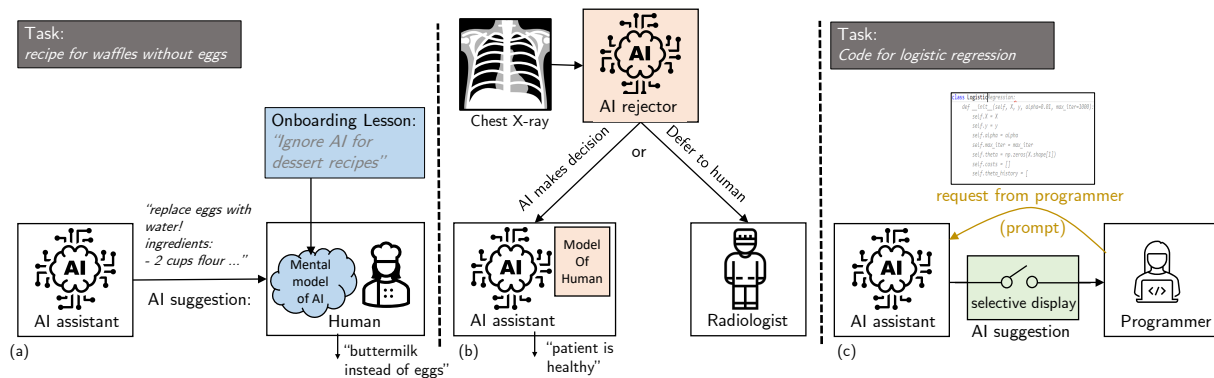


Figure 1.2: Human-AI interaction settings studied in this thesis. In subfigure (a), the AI-assisted decision-making setting where onboarding helps humans build better mental models of AI (part II). In (b), an AI rejector dictates who should predict between the human and AI (part I). Finally, in (c) is my work on programmers interacting with LLMs and interventions to selectively display suggestions (part III).

1.2 Deferral Systems

How do we combine AI systems and human decision makers to both reduce error and alleviate the burden on the human? AI systems are starting to be frequently used in combination with human decision makers, including in high-stakes settings like healthcare [1] and content moderation [3]. A possible way to combine the human and the AI is to learn a 'rejector' that queries either the human or the AI to predict on each input illustrated in Figure 1.2(b). This allows us to route examples to the AI model, where it outperforms the human, so as to simultaneously reduce error and human effort. Moreover, this formulation allows us to jointly optimize the AI so as to complement the human's weaknesses, and to optimize the rejector to allow the AI to defer when it is unable to predict well. This type of interaction is typically referred to as a *deferral system* and the learning problem is that of jointly learning the AI classifier and the rejector. Empirically, this approach has been shown to outperform either the human or the AI when predicting by their own [8], [9]. A motivating application arises in health care settings; for example, deep neural networks can outperform radiologists in detecting pneumonia from chest X-rays [10], however, many obstacles are limiting complete automation.

Formalization. We frame the learning with deferral setting as the task of predicting a target $Y \in \mathcal{Y} = \{1, \dots, C\}$. The classifier has access to features $X \in \mathcal{X} = \mathbb{R}^d$, while the human (also

referred to as the expert) has access to a potentially different set of features $Z \in \mathcal{Z}$ which may include side-information beyond X . The human is modeled as a fixed predictor $h : \mathcal{Z} \rightarrow \mathcal{Y}$. The AI system consists of a classifier $m : \mathcal{X} \rightarrow \mathcal{Y}$ and a rejector $r : \mathcal{X} \rightarrow \{0, 1\}$. When $r(x) = 1$, the prediction is deferred to the human and we incur a cost if the human makes an error, plus an additional, optional penalty term: $\ell_{\text{HUM}}(x, y, h) = \mathbb{I}_{h \neq y} + c_{\text{HUM}}(x, y, h)$. When $r(x) = 0$, then the classifier makes the final decision and incurs a cost with a different optional penalty term: $\ell_{\text{AI}}(x, y, m) = \mathbb{I}_{m \neq y} + c_{\text{AI}}(x, y, m)$. We can put this together into a loss function for the classifier and rejector:

$$L_{\text{def}}(m, r) = \mathbb{E}_{X, Y, Z} [\ell_{\text{AI}}(X, Y, m(X)) \cdot \mathbb{I}_{r(X)=0} + \ell_{\text{HUM}}(X, Y, h(Z)) \cdot \mathbb{I}_{r(X)=1}].$$

The joint learning optimization problem in the deferral system is:

$$\underset{m, r}{\text{minimize}} L_{\text{def}}(m, r) \tag{1.1}$$

Existing deployments tend to ignore that this system has two components and typically, the AI is trained without taking into account the human—and deferral is done by routing low-confidence examples to the human. By learning the AI jointly with the rejector as above, the aim is for the AI to *complement* the radiologist so that the human-AI team performance is higher. We formulate a natural loss function for the combined human-AI system and showed a reduction from this setting to cost-sensitive learning. We then prove that minimizing the human-AI system loss is computationally difficult. Therefore, we proposed a family of novel convex surrogate losses that consistently estimates the human-AI system loss and resolves an open problem in cost-sensitive learning [11].

Furthermore, a main limitation of complementing the human is the availability of samples of human predictions. For example, suppose we wish to deploy a system for diagnosing pneumonia from chest X-rays in a new hospital. To know when to defer to the new radiologists, we need to understand their specific strengths and weaknesses. We design a provable active learning scheme that first understands the human error boundary and then learns an AI classifier-rejector pair that adapts to it. We evaluate these approaches compared to baselines on tasks of chest X-ray diagnoses, content moderation, image classification, and income prediction. Specifically the contributions of part I of this thesis are:

- Chapter 2: we prove bounds on the gap between joint learning and staged learning, we propose a novel family of consistent surrogates that generalizes prior work and analyze asymptotic and sample properties, and finally, we provide an algorithm that is able to learn a classifier-rejector pair by minimally querying the human on selected points.

- Chapter 3: we prove the computational hardness of PAC-learning with deferral in the linear setting, we show how to formulate learning to defer with halfspaces as a MILP and provide a novel surrogate loss and we showcase the performance of our algorithms on a wide array of datasets and compare them to several existing baselines. We contribute a publicly available repository with implementations of existing baselines and datasets.

1.3 AI-Assisted Decision Making

The mode of interaction whereby the automated agent serves only to provide a recommendation to the human decision maker, a setting typically named *AI assisted decision making* illustrated in Figure 1.2(a), is the focus of our study in part II of this thesis. A key question is how does the human expert know when to rely on the AI for advice. The literature on human-AI collaboration has often revealed that humans often underperform expectations when working with AI systems [12]–[15]. The negative results of human-AI performance may be attributed to a few possible reasons. First, humans can have miscalibrated expectations about AI’s ability, which leads to over-reliance [16]. Second, the cost of verifying the AI’s answer with explanations might be too high thus providing a bad cost-benefit tradeoff for the human and leading to either over-reliance [17] or under-reliance on the AI [18]. Finally, the AI explanations do not able the human to verify the correctness of the AI’s answer and thus are not as useful for human-AI collaboration [19].

We make the case for the need to initially onboard the human decision maker on when and when not to rely on the automated agent. We propose that before an AI agent is deployed to assist a human decision maker, the human is taught through a tailored onboarding phase how to make decisions with the help of the AI. The purpose of the onboarding is to help the human understand when to trust the AI and how the AI can complement their abilities. This allows the human to have an accurate mental model of the AI agent, and this mental model helps in setting expectations about the performance of the AI on different examples.

Formalization. We consider a setting where a human is making decisions with the help of an AI agent who provides advice to complete a task. Formally, the human has to make a decision $Y \in \mathcal{Y}$ given access to information about the context as $X \in \mathcal{X}$ and the AI’s advice $A \in \mathcal{A}$. We denote the human as a potentially randomized function $H(X, A; \theta_h)$ with parameters θ_h which are unobservable. On the other hand, the AI agent provides advice based on its viewpoint of the context X according to $M(X; \theta_m) := A \in \mathcal{A}$. The advice always includes a candidate decision \hat{A} and possibly an explanation of the decision. We assume that the observed tasks are drawn from an underlying distribution, $\mathbb{P}_{X,Y}$, over the contexts of AI and human, and the ground truth. The setting is illustrated in Figure 5.2.

The human wants to make a decision that optimizes various metrics of interest. Given a ground truth decision Y and a chosen decision \hat{Y} , the loss is given by $l(Y, \hat{Y}) : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$. In our example, this could be the 0-1 loss $\mathbb{I}_{Y=\hat{y}}$. We denote the loss $L(H, M)$ of the Human-AI team over the entire domain as:

$$L(H, M) := \mathbb{E}_{x, y \sim \mathbb{P}} [l(y, H(x, M(x; \theta_m); \theta_h))] \quad (1.2)$$

We propose a two-stage framework for the human to cooperate with the AI: the human first decides whether to ignore the AI advice, use the AI’s decision or integrate the AI advice to make a decision with explicit cooperation. Each of these three cases provides a clear path to the second stage of making the final output decision. The AI-integration function $R(X, A; \theta_r)$, also referred to as integrator, formalizes a framework for the human to cooperate with the AI:

$$R(X, A; \theta_r) = \begin{cases} 0 \rightarrow H(X; \theta_h) & \text{(ignore AI)} \\ 1 \rightarrow \hat{A} & \text{(use AI decision as is)} \\ 2 \rightarrow H(X, A; \theta_h) & \text{(collaborate with AI)} \end{cases} \quad (1.3)$$

The integration function can be thought of as a specific formalization of the human mental model of the AI [16], [20].

Our onboarding phase consists of letting the human predict on a series of specially selected teaching examples in a setting that mimics the deployment use case. The examples are chosen to give an overview of AI’s strengths and weaknesses, especially when they complement human abilities. After predicting on each example, the human agent then receives feedback on their performance and that of the AI. To allow the human to generalize from each example, we display features of the region surrounding the example including a description. Our approach is inspired by research in the education literature that highlights the importance of feedback and lesson retention for learning [21], [22].

To select the teaching examples, we need to have a mathematical framework of how the human mental model evolves after we give them feedback. We model the human thought process as first deciding whether to rely on the AI’s prediction or not using an internal integrator. This rejector is what we refer to as the human mental model of the AI. We propose to model the human integrator as consisting of a prior rejector and a nearest neighbor rule that only applies in local regions surrounding each teaching example. Assuming this mental model, we give a near-optimal greedy strategy for selecting a set of representative teaching examples that allows us to control the examples and the region surrounding them. We conducted user studies on question answering and object detection with AI models to evaluate the onboarding method. The contributions are as follows:

- Chapter 4: we propose a mathematical formulation of human’s mental model of AI based on a local neighbor rule, based on this formulation, we create an example selection procedure that picks examples that best improve On a user study for question answering, our onboarding method outperforms baseline onboarding methods and allows us to reveal participants’ mental models
- Chapter 5: we propose a region description algorithm to automatically create natural language algorithms of regions found for onboarding to extend our prior work. We evaluate an improved version of our onboarding procedure on a questions answering task and an object detection showcasing that onboarding helps improve humans performance.

1.4 Interactive Human-AI Collaboration

The previous setting of AI-assisted decision-making consisted of one-sided interaction between humans and AI. The advent of LLMs allows the human to indicate their intent for support through a textual prompt to the AI, who can respond back through text. We focus on the code suggestion system, [GitHub Copilot](#), powered by the Codex LLM, which is being used by millions of programmers daily illustrated in Figure 1.2(c). Given the nascent nature of these systems, numerous questions exist regarding the behavior of their users, e.g., what activities do users undertake in anticipation of or to trigger a suggestion? How *costly* for users are these various new tasks, and which take the most time and effort? To answer these and related questions systematically, we developed a taxonomy called CUPS, shown in Figure 6.3 for the different activities programmers perform while coding with Copilot.

Given the initial taxonomy, we conducted a user study with 21 developers who were asked to retrospectively review videos of their coding sessions and explicitly label their intents and actions. The study discovered that participants spend **22.4% of their time verifying suggestions from the LLM** and that more than half of total coding time is spent interacting with the LLM: prompt crafting, and editing suggestions, among other activities. This signals a major shift in how programmers write code. Moreover, we uncovered that existing metrics that measure the time to verify suggestions underestimate actual verification time by a factor of two.

From this analysis, programmers spend the most time verifying Copilot suggestions and reject more than 80% of them; how can we reduce that verification time? The suggestions programmers accept and reject can provide a window into their preferences of which suggestion they benefit from and which they do not. We formalized the interaction between programmers and Copilot in a utility theory framework that assigns a utility value of showing a given suggestion. Using data from 535 programmers at Microsoft, we materialized the framework to learn a model that assigns a value

to each suggestion. This model can be leveraged to conditionally hide suggestions that have low utility. The specific contributions are as follows:

- Chapter 6: we propose a novel taxonomy of common activities of programmers (called CUPS) when interacting with code recommendation systems. We collect data in a user study where programmers retrospectively label their activities with CUPS resulting in a new dataset of coding sessions annotated with user actions, CUPS, and video recordings available publicly. We further propose a new instrument for measuring and analyzing patterns in user behavior, time spent in different states, and adjustments required for better estimating the time impact of code-generation tools.
- Chapter 7: we propose a mathematical framework for AI-Assisted programming named CDHF, that decides when to show code suggestions to save programmers time. We train a prediction model of the programmer's actions learned from hundreds of users to operationalize CDHF. We further conduct an experimental analysis of the consequences of ranking suggestions by probability of acceptance.

Part I

Conditional Delegation with AI: Learning to Defer

Chapter 2

Sample Efficient Learning to Defer

Acknowledgements of Co-authors. This chapter is based on the published work in [23]. I would like to thank my co-authors, Mohammad-Amin Charusaie and Samira Samadi, for their help in this chapter. This work was an equal contribution between Mohammad-Amin and myself and his contributions were essential to proving the theorems in this chapter.

2.1 Introduction

How do we combine AI systems and human decision makers to both reduce error and alleviate the burden on the human? AI systems are starting to be frequently used in combination with human decision makers, including in high-stakes settings like healthcare [1] and content moderation [3]. A possible way to combine the human and the AI is to learn a 'rejector' that queries either the human or the AI to predict on each input. This allows us to route examples to the AI model, where it outperforms the human, so as to simultaneously reduce error and human effort. Moreover, this formulation allows us to jointly optimize the AI so as to complement the human's weaknesses, and to optimize the rejector to allow the AI to defer when it is unable to predict well. This type of interaction is typically referred to as *expert deferral*, and the learning problem is that of jointly learning the AI classifier and the rejector. Empirically this approach has been shown to outperform either the human or the AI when predicting by their own [8], [9]. One hypothesis is that humans and machines make different kinds of errors. For example, humans may have bias on certain features [24] while AI systems may have bounded expressive power or limited training data. On the other hand, humans may outperform AI systems as they may have side information that is not available to the AI, for example, due to privacy constraints.

Existing deployments tend to ignore the fact that the system has two components: the AI classifier (henceforth, the classifier) and the human. Typically, the AI is trained without taking into account the human—, and deferral is done using post-hoc approaches like model confidence

[25]. The main problem of this approach which we refer to as *staged learning*, is that it ignores the possibility of learning a better-combined system by accounting for the human (and its mistakes) during training. More recent work has developed joint training strategies for the AI and the rejector based on surrogate losses and alternating minimization [26], [27]. However, we lack a theoretical understanding of the fundamental merits of joint learning compared to the staged approach. In this work, we study three main challenges in expert deferral from a theoretical viewpoint: 1) *model capacity* constraints, 2) lack of *data of human expert’s prediction*, and 3) optimization using *surrogate losses*.

When learning a predictor and rejector in a limited hypothesis class, it becomes more valuable to allocate model capacity to complement the human. We prove a bound on the gap between the approach that learns a predictor that complements humans and the approach that learns the predictor ignoring the presence of the human in Section 2.4. To practically learn to complement the human, the literature has shown that surrogate loss functions are successful [26], [28]. We propose a family of surrogate loss functions that generalizes existing surrogates such as the surrogate in [26], and we further prove surrogate excess risk bounds and generalization properties of these surrogates in Section 2.5. Finally, a main limitation of being able to complement the human is the availability of samples of human predictions. For example, suppose we wish to deploy a system for diagnosing pneumonia from chest X-rays in a new hospital. To be able to know when to defer to the new radiologists, we need to understand their specific strengths and weaknesses. We design a provable active learning scheme that is able to first understand the human expert error boundary and learn a classifier-rejector pair that adapts to it in Section 2.6. To summarize, the contributions of this chapter are the following:

- **Understanding the gap between joint and staged learning:** we prove bounds on the gap when learning in bounded capacity hypothesis classes and with missing human data.
- **Theoretical analysis of Surrogate losses:** we propose a novel family of consistent surrogates that generalizes prior work and analyze asymptotic and sample properties.
- **Actively learning to defer:** we provide an algorithm that is able to learn a classifier-rejector pair by minimally querying the human on selected points.

2.2 Related Work

A growing literature has focused on building models that can effectively defer predictions to human experts. Initial work posed the problem as that of a mixture of experts [28]. However, their approach does not allow the model to adapt to the expert. A different natural baseline that is

proposed in [25] learns a predictor that best classifies the target and then compares its confidence to that of the expert. This is what we refer to as *staged learning*, and in our work, we provide the first theoretical results on the limitations of this approach. [29] and [30] jointly learn a classifier and rejector based on the mixture of experts loss, but the method lacks a theoretical understanding and requires heuristic adjustments. [26] proposes the first consistent surrogate loss function for the expert deferral setting, which leads to an effective joint learning approach with subsequent work building on their approach [31], [32]. In this chapter, we generalize the surrogate presented in [26] and present generalization guarantees that enable us to effectively bound performance when learning with this surrogate. [33] proposes a surrogate loss which is the sum of the loss of learning the classifier and rejector separately but which is not a consistent surrogate. [27] proposes an iterative method that alternates between optimizing the predictor and the rejector and shows that it converges to a local minimum and empirically matches the performance of the surrogate in [26]. Multiple works have used the learning-to-defer paradigm in other settings [34]–[37].

In our work, we derive an active learning scheme that enables us to understand the human expert error boundary with the least number of examples. This bears similarity to work on onboarding humans on AI models where the objective is reversed: teaching the human about the AI models error boundary [38]–[40] and work on machine teaching [41], [42]. However, our setting requires a distinct methodology as we have no restrictions on the parameterization of our rejector, which the previous line of work assumes. Works on Human-AI interaction usually keep the AI model fixed and optimize for other aspects of the interaction, while in our work we optimize the AI to complement the human [20], [43].

The setting when the cost of deferral is constant has a long history in machine learning and goes by the name of rejection learning [44]–[47] or selective classification (only predict on $x\%$ of data) [48]–[51]. [52] explored an online active learning scheme for rejection learning; however, their scheme was tailored to a surrogate for rejection learning that is not easily extendable to expert deferral. Our work also bears resemblance to active learning with weak (the expert) and strong labelers (the ground truth) [53]

2.3 Problem Setting

We study classification problems where the goal is to predict a target $Y \in \{1, \dots, K\}$ based on a set of features $X \in \mathcal{X}$, or via querying a human expert opinion $M \sim \mu_{M|XY}$ that has access to a domain \mathcal{Z} . Upon viewing the input X , we decide first via a rejector function $r : \mathcal{X} \rightarrow \{0, 1\}$ whether to defer to the expert, where $r(\mathbf{x}) = 1$ means deferral and $r(\mathbf{x}) = 0$ means predicting using a classifier $h : \mathcal{X} \rightarrow [K]$. The expert domain may contain side information beyond X to classify instances. For example, when diagnosing diseases from chest X-rays the human may have access to

the patient’s medical records while the AI only has access to the X-ray. We assume that X, Y, M have a joint probability measure μ_{XYM} .

We let deferring the decision to the expert incur a cost equal to the expert’s error and an additional penalty term: $\ell_{\text{exp}}(\mathbf{x}, y, m) = \mathbb{I}_{m \neq y} + c_{\text{exp}}(\mathbf{x}, y, m)$ that depends on the features \mathbf{x} , the value of target $Y = y$, and the expert’s prediction $M = m$. Moreover, we assume that predicting without querying the expert incurs a different cost equal to the classifier error and an additional penalty: $\ell_{\text{AI}}(\mathbf{x}, y, m) = \mathbb{I}_{h(\mathbf{x}) \neq y} + c_{\text{AI}}(\mathbf{x}, y, m)$ where $h(\mathbf{x})$ is the prediction of the classifier. With the above in hand, we write the true risk as

$$L_{\text{def}}(h, r) = \mathbb{E}_{X, Y, M} [\ell_{\text{AI}}(X, Y, h(X)) \cdot \mathbb{I}_{r(X)=0} + \ell_{\text{exp}}(X, Y, M) \cdot \mathbb{I}_{r(X)=1}] \quad (2.1)$$

In the setting when we only care about misclassification costs with no additional penalties, the deferral loss becomes a 0 – 1 loss as follows:

$$L_{\text{def}}^{0-1}(h, r) = \mathbb{E} [\mathbb{I}_{h(X) \neq Y} \mathbb{I}_{r(X)=0} + \mathbb{I}_{M \neq Y} \mathbb{I}_{r(X)=1}] \quad (2.2)$$

We focus primarily on the 0 – 1 loss for our analysis; it is also possible to extend parts of the analysis to handle additional cost penalties. We restrict our search to classifiers within a hypothesis class \mathcal{H} and a rejector function within a hypothesis class \mathcal{R} . The optimal joint classifier and rejector pair is the one that minimizes (3.1):

$$h^*, r^* = \underset{h \in \mathcal{H}, r \in \mathcal{R}}{\operatorname{argmin}} L_{\text{def}}^{0-1}(h, r) \quad (2.3)$$

To approximate the optimal classifier-rejector pair, we have to handle two main obstacles: (i) *optimization* of the non-convex and discontinuous loss function and (ii) availability of the *data* on human’s predictions and the true label.

In the following section 2.4 and in section 2.6, we restrict the analysis to binary labels $Y = \{0, 1\}$ for a clearer exposition. The theoretical results in the following section are shown to apply further for the multiclass setting in a set of experimental results. However, in section 2.5, where we discuss practical algorithms, we switch back to the multiclass setting for full generality. In the following section, we compare two strategies for expert deferral across these two dimensions.

2.4 Staged Learning of Classifier and Rejector

2.4.1 Model Complexity Gap

Staged learning. The optimization problem framed in (2.3) requires joint learning of the classifier and rejector. Alternatively, a popular approach comprises of first learning a classifier that minimizes average misclassification error on the distribution and then learning a rejector that defers each point to either classifier or the expert, depending on who has a lower estimated error [25], [29].

Formally, we first learn h to minimize the average misclassification error:

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} \mathbb{E}_{X,Y} [\mathbb{I}_{h(X) \neq Y}] \quad (2.4)$$

and in the second step we learn the rejector r to minimize the joint loss (3.1) with the now fixed classifier \hat{h} :

$$\hat{r} = \operatorname{argmin}_{r \in \mathcal{R}} L_{\text{def}}^{0-1}(\hat{h}, r) \quad (2.5)$$

This procedure is particularly attractive as the two steps (2.4) and (2.5) could be cast as classification problems, and approached by powerful known tools that are devised for such problems. Despite its convenience, this method is not guaranteed to achieve the optimal loss (as in (2.3)), since it decouples the joint learning problem. Assuming that we are able to optimally solve both problems on the true distribution, let (h^*, r^*) denote the solution of joint learning and (\hat{h}, \hat{r}) the solution of staged learning. To estimate the extent to which staged learning is sub-optimal, we define the following minimax measure $\Delta(d_1, d_2)$ for the binary label setting:

$$\Delta(d_1, d_2) = \inf_{\mathcal{H}, \mathcal{R} \in \mathfrak{H}_{d_1, d_2}} \sup_{\mu_{XYM}} L_{\text{def}}^{0-1}(\hat{h}, \hat{r}) - L_{\text{def}}^{0-1}(h^*, r^*)$$

To disentangle the above measure, the supremum $\sup_{\mu_{XYM}}$ is a worst-case over the data distribution and expert pair, while the infimum $\inf_{\mathcal{H}, \mathcal{R} \in \mathfrak{H}_{d_1, d_2}}$ is the best-case classifier-rejector model classes with specified complexity d_1 and d_2 where $\mathfrak{H}_{d_1, d_2} = \{(\mathcal{H}, \mathcal{R}) : d(\mathcal{H}) = d_1, d(\mathcal{R}) = d_2\}$ and $d(\cdot)$ denotes the VC dimension of a hypothesis class. As a result, this measure expresses the worst-case gap between joint and staged learning when learning from the optimal model class, given the complexity of the predictor and rejector model classes. The following theorem provides a lower- and upper-bound on $\Delta(d_1, d_2)$.

Theorem 1. *For every set of hypothesis classes \mathcal{H}, \mathcal{R} where $d(\cdot)$ denotes the VC-dimension of a hypothesis class, the minimax difference measure between joint and staged learning is bounded*

between:

$$\frac{1}{d(\mathcal{H}) + 1} \leq \Delta(d(\mathcal{H}), d(\mathcal{R})) \leq \frac{d(\mathcal{R})}{d(\mathcal{H})} \quad (2.6)$$

Proof of the theorem can be found in Appendix A.1. The theorem implies that for any classifier and rejector hypothesis classes, we can find a distribution and an expert such that the gap between staged learning and joint learning is at least 1 over the VC dimension of the classifier hypothesis class. This means that the more complex our classifier hypothesis class is, the smaller the gap between joint and staged learning is. On the other hand, the gap is no larger than the ratio between the rejector complexity and the classifier complexity. This again implies if our hypothesis class is comparatively much richer than the rejector class, the gap between the joint and staged learning reduces. What this does not mean is that deferring to the human is not required for optimal error when the classifier model class is very large, but that training the classifier may not require knowledge of human performance.

2.4.2 Data Trade-offs

Current datasets in machine learning are growing in size and are usually in the form of feature X and target Y pairs. It is unrealistic to assume that the human expert is able to individually provide their predictions for all of the data. In fact, the collection of datasets in machine learning often relies on crowd-sourcing where the label can either be a majority vote of multiple human experts, e.g., in hate-speech moderation [54], or due to an objective measurement, e.g., a lab test result for a patient medical data. In the expert deferral problem, we are interested in the predictions of a particular human expert, and thus, it is infeasible for that human to label all the data and perhaps unnecessary.

In the following analysis, we assume access to fully labeled data $S_l = \{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^{n_l}$ and data without expert labels $S_u = \{(\mathbf{x}_i, y_i)\}_{i=n_l+1}^{n_l+n_u}$. This is a realistic form of the data we have available in practice. We now try to understand how we can learn a classifier and rejector from these two datasets. This is where we expect the staged learning procedure can become attractive as it can naturally exploit the two distinct datasets to learn.

Joint Learning. Learning jointly requires access to the dataset with the entirety of expert labels, thus we can only use S_l to learn

$$\tilde{h}, \tilde{r} = \operatorname{argmin}_{h,r} \sum_{i \in S_l} \mathbb{I}_{h(\mathbf{x}_i) \neq y_i} \mathbb{I}_{r(\mathbf{x}_i) = 0} + \mathbb{I}_{y_i \neq m_i} \mathbb{I}_{r(\mathbf{x}_i) = 1}$$

Staged learning. On the other hand, for staged learning we can exploit our expert unlabeled data to first learn h :

$$\hat{h} = \operatorname{argmin}_h \sum_{i \in S_u} \mathbb{I}_{h(\mathbf{x}_i) \neq y_i}$$

and in the second step we learn \hat{r} to minimize the joint loss with the fixed \hat{h} but only on S_l .

Generalization. Given that staged learning exploits both datasets, we expect that if we have much more expert unlabeled data than labeled data, i.e., $n_u \gg n_l$, then it may be possible to obtain better generalization guarantees from staged learning. The following proposition shows that when the Bayes optimal classifier is in the hypothesis class, then staged learning can possibly improve sample complexity over joint learning.

Proposition 1. *Let $S_l = \{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^{n_l}$ and $S_u = \{(\mathbf{x}_{i+n_l}, y_{i+n_l})\}_{i=1}^{n_u}$ be two iid sample sets that are drawn from the distribution μ_{XYM} and are labeled and not labeled by the human, respectively. Assume that the optimal classifier $\bar{h} = \operatorname{argmin}_h \mathbb{E}_{X,Y \sim \mu_{XY}} [\mathbb{I}_{h(X) \neq Y}]$ is a member of \mathcal{H} (i.e., realizability).*

Let (\hat{h}, \hat{r}) be the staged solution and let (\tilde{h}, \tilde{r}) be the joint solution obtained by learning only on S_l . Then, with probability at least $1 - \delta$ we have for staged learning

$$\begin{aligned} L_{\text{def}}^{0-1}(\hat{r}, \hat{h}) &\leq L_{\text{def}}^{0-1}(h^*, r^*) + \mathfrak{R}_{n_u}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{R}) + 2 \min \{P(M \neq Y), \mathfrak{R}_{n_l \mathbb{P}(M \neq Y)/2}(\mathcal{R})\} \\ &\quad + C \sqrt{\frac{\log 1/\delta}{\min(n_l, n_u)}} + P(M \neq Y) e^{-n_l \frac{\mathbb{P}(M \neq Y)^2}{2}} \end{aligned} \quad (2.7)$$

while for joint learning we have:

$$\begin{aligned} L_{\text{def}}^{0-1}(\tilde{r}, \tilde{h}) &\leq L_{\text{def}}^{0-1}(h^*, r^*) + \mathfrak{R}_{n_l}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{R}) + 2\mathfrak{R}_{n_l \mathbb{P}(M \neq Y)/2}(\mathcal{R}) + C \sqrt{\frac{\log 1/\delta}{n_l}} \\ &\quad + P(M \neq Y) e^{-n_l \frac{\mathbb{P}(M \neq Y)^2}{2}} \end{aligned} \quad (2.8)$$

Proof of the proposition can be found in Appendix A.2. From the above proposition, when the Bayes classifier is in the hypothesis class, the upper bound for the sample complexity required to learn the classifier and rejector is reduced by only paying the Rademacher complexity of the hypothesis class on the unlabeled data compared to on the potentially smaller labeled dataset. The Rademacher complexity is a measure of model class complexity on the data and can be related to the VC dimension.

While in this case study, staged learning may improve the generalization error bound compared to that of joint learning, the number of labeled samples for both to achieve ϵ -upper-bound on the true

risk is of order $O(\frac{\log 1/\epsilon}{\epsilon^2})$. As we can see, there exist computational and statistical trade-offs between joint and staged learning. While joint learning leads to more accurate systems, it is computationally harder to optimize than staged learning. In the next section, we investigate whether it is possible to solve the joint learning problem more efficiently while still retaining its favorable guarantees in the multiclass setting.

2.5 Surrogate Losses For Joint Learning

2.5.1 Family of Surrogates

A common practice in machine learning is to propose surrogate loss functions, which often are continuous and convex, that approximate the original loss function we care about [55]. The hope is that these surrogates are more readily optimized and minimizing them leads to predictors that also minimize the original loss. In their work on expert deferral, [26] reduces the learning to defer problem to cost-sensitive learning which enables them to use surrogates for cost-sensitive learning in the expert deferral setting. We follow the same route in deriving our novel family of surrogate losses. We now recall the reduction in [26]: define the random costs $\mathbf{c} \in \mathbb{R}_+^{K+1}$ where $c(i)$ is the i 'th component of \mathbf{c} and represents the cost of predicting label $i \in [K + 1]$. The goal of cost sensitive learning is to build a predictor $\tilde{h} : \mathcal{X} \rightarrow [K + 1]$ that minimizes the cost-sensitive loss $\mathbb{E}[c(\tilde{h}(X))]$. The reduction is accomplished by setting $c(i) = \ell_{\text{AI}}(X, Y, i)$ for $i \in [K]$ while $c(K + 1)$ represents the cost of deferring to the expert with $c(K + 1) = \ell_{\text{exp}}(X, Y, M)$. Thus, the predictor \tilde{h} learned in cost-sensitive learning implicitly defines a classifier-rejector pair (h, r) with the following encoding:

$$h(\mathbf{x}), r(\mathbf{x}) = \begin{cases} h(\mathbf{x}) = i, r(\mathbf{x}) = 0, & \text{if } \tilde{h}(\mathbf{x}) = i \in [K] \\ h(\mathbf{x}) = 1, r(\mathbf{x}) = 1 & \text{if } \tilde{h}(\mathbf{x}) = K + 1 \end{cases} \quad (2.9)$$

Note that when $\tilde{h}(\mathbf{x}) = K + 1$, the classifier h is left unspecified, and thus, we assign it a dummy value of 1. Cost-sensitive learning is a non-continuous and non-convex optimization problem that makes it computationally hard to solve in practice. In order to approximate it, we propose a novel family of cost-sensitive learning loss functions that extend any multi-class loss function to the cost-sensitive setting.

First we parameterize our predictor \tilde{h} with $K + 1$ functions $f_i : \mathcal{X} \rightarrow \mathbb{R}$ and define the predictor to be the max of these $K + 1$ functions: $\tilde{h}_{\mathbf{f}}(\mathbf{x}) = \arg \max_i f_i(\mathbf{x})$. Note that $\tilde{h}_{\mathbf{f}}$ gives rise to the classifier-rejector pair $(h_{\mathbf{f}}, r_{\mathbf{f}})$ according to the decoding rule (2.9).

Formally, let $\ell_{\phi}(y, \mathbf{f}(\mathbf{x})) : [K + 1] \times \mathbb{R}^{K+1} \rightarrow \mathbb{R}$ be a surrogate loss function of the zero-one loss for multiclass classification. We define the extension of this surrogate to the cost-sensitive

setting as:

$$\tilde{\ell}_\phi(\mathbf{c}, \mathbf{f}(\mathbf{x})) = \sum_{i=1}^{K+1} \left[\max_{j \in k+1} c(j) - c(i) \right] \ell_\phi(i, \mathbf{f}(\mathbf{x})) \quad (2.10)$$

Note that if ℓ_ϕ is continuous or convex, then because $\ell_{\mathbf{c},\phi}$ is a finite positively-weighted sum of ℓ_ϕ 's, then $\ell_{\mathbf{c},\phi}$ is also continuous or convex, respectively. We show in the following proposition, that if ℓ_ϕ is a consistent surrogate for multi-class classification, then $\tilde{\ell}_\phi$ is consistent for cost-sensitive learning and by the reduction above is also consistent for learning to defer.

Proposition 2. *Suppose $\ell_\phi(y, \mathbf{f}(\mathbf{x}))$ is a consistent surrogate for multi-class classification, meaning if the surrogate is minimized over all functions, then it also minimizes the misclassification loss:*

let $\mathbf{f}^ = \arg \inf_{\mathbf{f}} \mathbb{E} [\ell_\phi(Y, \mathbf{f}(\mathbf{x}))]$, then: $\tilde{\mathbf{h}}_{f^*} = \arg \inf_{\mathbf{h}} \mathbb{E} [\mathbb{I}_{Y \neq \mathbf{h}(X)}]$, where $\tilde{\mathbf{h}}_{f^*}$ is defined as above.*

Then, our surrogate $\tilde{\ell}_\phi(\mathbf{c}, \mathbf{g}(\mathbf{x}))$ defined in (2.10) is a consistent surrogate for cost-sensitive learning and thus for learning to defer:

let $\tilde{\mathbf{f}}^ = \arg \inf_{\mathbf{g}} \mathbb{E} [\tilde{\ell}_\phi(\mathbf{c}, \mathbf{f}(\mathbf{x}))]$, then: $h_{\tilde{\mathbf{f}}^*}^*, r_{\tilde{\mathbf{f}}^*}^* = \arg \inf_{h,r} L_{\text{def}}^{0-1}(h, r)$, with $(h_{\tilde{\mathbf{f}}^*}^*, r_{\tilde{\mathbf{f}}^*}^*)$ defined in (2.9)*

Proof of the proposition can be found in Appendix A.3. To illustrate the family of surrogates implied by Proposition 2, we first start by recalling a family of surrogates for multi-class classification. Theorem 4 of [56] shows that there is a family of consistent surrogates for 0 – 1 loss in multi-class classification parameterized by three functions (u, s, t) and takes the form $\ell_\phi(y, \mathbf{f}(\mathbf{x})) = u(f_y(\mathbf{x})) + s\left(\sum_{j=1}^{K+1} t(f_j(x))\right)$. This family is consistent under certain conditions of the aforementioned functions.

Now we show with a few examples that this family encompasses some popular surrogates used in cost sensitive learning:

Examples. (1) *If we set $u(x) = -2x$, $s(x) = x$, and $t(x) = x^2$, then we can obtain a weighted quadratic loss:*

$$\tilde{L}_2 = \mathbb{E} \left[\pi \sum_{i=1}^{K+1} |f_i - q(i)|^2 \right], \quad (2.11)$$

where $q(i)$ is the normalized expected value of $\max_{j \in [K+1]} c(j) - c(i)$ given $X = x$, and π represents the normalization term.

(2) *If we set $u(x) = -x$, and $s(x) = \log(x)$ and $t(x) = e^x$, then we have $a\psi'(x) + t'(x) = -a + e^x = 0$, and as a result $x = \log a$, which is an increasing function of a . As a result, the surrogate loss*

$$\tilde{L}_{CE}(\mathbf{f}) = \mathbb{E} \left[- \sum_{i=1}^{K+1} \left(\max_j c(j) - c(i) \right) \log \frac{e^{f_i(X)}}{\sum_{k=1}^{K+1} e^{f_k(X)}} \right] \quad (2.12)$$

which is the loss defined in [26] and used for learning to defer.

2.5.2 Theoretical Properties of Surrogate

Goodness of a Surrogate. Given a surrogate, how can we quantify how well it approximates our original loss? One avenue is through the surrogate excess-risk bound as follows. Let \tilde{L} be a surrogate for the loss function L , and let \tilde{h}^* be the minimizer of the surrogate and h^* the minimizer of L . We call the *excess surrogate risk* [55] the following quantity if we can find a *calibration function* ψ such that for any h we have:

$$\psi(L(h) - L(h^*)) \leq \tilde{L}(h) - \tilde{L}(\tilde{h}^*) \quad (2.13)$$

The excess surrogate risk bound tells us if we are ϵ -close to the minimizer of the surrogate, then we are $\psi^{-1}(\epsilon)$ -close to the minimizer of the original loss.

We now show that the family of surrogates defined in (2.10) has a polynomial excess-risk bound and furthermore prove an excess-risk bound for the surrogate loss function \tilde{L}_{CE} defined in [26].

Theorem 2. *Suppose that $\psi(x) = Cx^\epsilon$, for $\epsilon \in [1, \infty)$ is a calibration function for the multiclass surrogate ℓ_ϕ and if $|c(i)| \leq M$ for all i , then $\psi'(x) = \frac{C}{M^{\epsilon-1}}x^\epsilon$ is a calibration function of $\tilde{\ell}_\phi(\mathbf{c}, \cdot)$.*

As an example, for the surrogate \tilde{L}_{CE} (2.12) the calibration function is $\psi(x) = \frac{1}{16MK}x^2$.

Proof of the theorem can be found in Appendix A.4. Note that [57] proved that for the cross-entropy loss, the calibration function ϕ is of order $\Theta(\epsilon^2)$, which is in accordance with our results.

Generalization Error. Equipped with the excess surrogate risk bound, we can now study the sample complexity properties of minimizing the surrogates proposed. For concreteness, we focus on the surrogate \tilde{L}_{CE} of [26] when reduced to the learning to defer setting. The following theorem proves a generalization error bound when minimizing the surrogate \tilde{L}_{CE} for learning to defer.

Theorem 3. *Let K denote the number of classes, and let \mathcal{F} be a hypothesis class of functions $f_i : \mathcal{X} \rightarrow \mathbb{R}$ with bounded infinity norm $\|f_i\|_\infty < C$. Given $\hat{\mathbf{f}} \in \mathcal{F}^{k+1}$ the empirical minimizer of the surrogate loss \tilde{L}_{CE} , then we have with probability at least $1 - \delta$, we have*

$$\begin{aligned} \psi(L_{\text{def}}^{0-1}(h_{\hat{\mathbf{f}}}, r_{\hat{\mathbf{f}}}) - \min_{\mathbf{f}} L_{\text{def}}^{0-1}(h_{\mathbf{f}}, r_{\mathbf{f}})) &\leq 2(K+1)\mathfrak{R}_n(\mathcal{F}) + \sqrt{\frac{(8C - 4 \log(K+1)) \log 2/\delta}{n}} \\ &\quad + 2(K+2) \min\{\mathbb{P}(M \neq Y), \mathfrak{R}_{n\mathbb{P}(M \neq Y)/2}(\mathcal{F})\} \\ &\quad + C\mathbb{P}(M \neq Y)(k+2)e^{-n\mathbb{P}^2(M \neq Y)/2} + e_{\phi\text{-appr}}, \end{aligned} \quad (2.14)$$

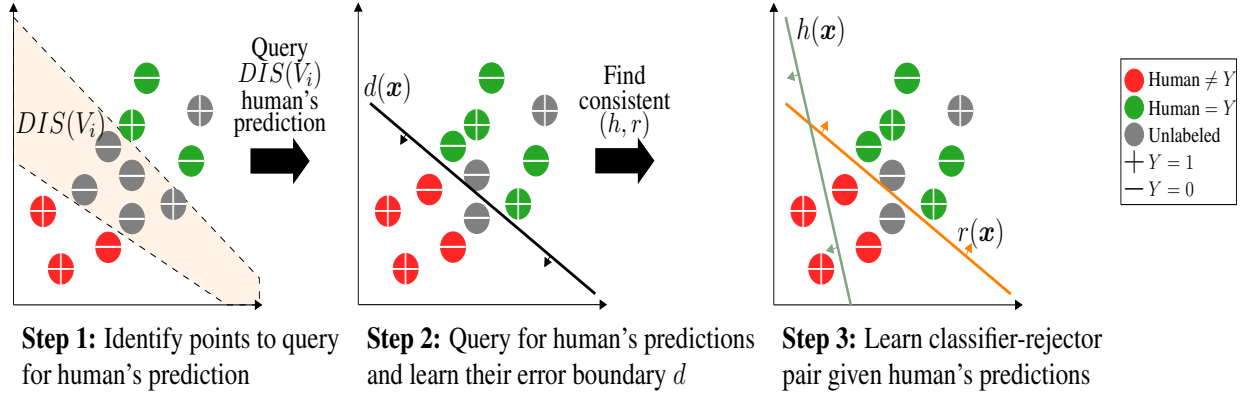


Figure 2.1: Illustration of our active learning algorithm Disagreement on Disagreements (DoD) (1). At each round, we compute the disagreement set for our predictors of the human label disagreement, we then query the human for their prediction on these points. After we learn the expert error boundary, we then learn a consistent classifier-rejector pair.

where $e_{\phi\text{-appr}}$ is the approximation error for the ϕ -surrogate, which is defined as

$$e_{\phi\text{-appr}} = \min_{\mathbf{f} \in \mathcal{F}^{k+1}} \tilde{L}_{CE}(\mathbf{f}) - \min_{\mathbf{f}} \tilde{L}_{CE}(\mathbf{f}). \quad (2.15)$$

Proof of the theorem can be found in Appendix A.5. Comparing the sample complexity estimate for minimizing the surrogate to that of minimizing the 0-1 loss as computed by [26], we find that we pay an additional penalty for the complexity of the hypothesis class in addition to the higher sample complexity that scales with $O(\frac{\log \epsilon}{\epsilon^d})$ due to the calibration function. To compensate for such increase in sample complexity, in the next section we seek to design active learning schemes that reduce the required amount of human labels for learning.

2.6 Active Learning for Expert Predictions

2.6.1 Theoretical Understanding

In Section 2.4, we assumed that we have a randomly selected subset of data that is labeled by the human expert. In a practical setting, we may assume that we have the ability to choose which points we would like the human expert to predict. For example, when we deploy an X-ray diagnostic algorithm to a new hospital, we can interact with each radiologist for a few rounds to build tailored classifier-rejector pairs according to their individual abilities.

Therefore, we assume that we have access to the distribution of instances \mathbf{x} and their labels, and we could query for the expert's prediction on each instance. The goal is to query the human expert on as few instances as possible while being able to learn an approximately optimal classifier-rejector

pair. To make progress in theoretical understanding, we assume that we can achieve zero loss with an optimal classifier-rejector pair:

Assumption 1 (Realizability). *We assume that the data is realizable by our joint class $(\mathcal{H}, \mathcal{R})$: there exists $h^*, r^* \in \mathcal{H} \times \mathcal{R}$ that have zero error $L(h^*, r^*) = 0$.*

In this section, the algorithms we develop apply to the multiclass setting, but we restrict the theoretical analysis to binary labels. The fundamental algorithm in active learning in the realizable case for classification is the CAL algorithm [58]. The algorithm keeps track of a version class of the hypothesis space that is consistent with the data so far and then, at each step, computes the disagreement set: the points on which there exists two classifiers in the hypothesis class with different predictions and then picks at random a set of points from this disagreement set. We start by initializing our version space by taking advantage of Assumption 1:

$$V_0 = \{h, r \in \mathcal{H} \times \mathcal{R} : \forall \mathbf{x}, r(\mathbf{x}) = 0 \rightarrow h(\mathbf{x}) = y\} \quad (2.16)$$

The above initialization of the version space assumes we know the label of all instances in our support. Alternatively, one could collect at most $O(\frac{1}{\epsilon}(d(\mathcal{H}) \log \frac{1}{\epsilon} + \log \frac{1}{\delta}))$ labels of instances and that would be sufficient to test for realizability of our classifier with error ϵ (see Lemma 3.2 of [58]).

The main difference with active learning for classification is that we are not able to compute the disagreement set for the overall prediction of the deferral system as it requires knowing the expert predictions. However, we know that a necessary condition for disagreement is that a feasible pair of classifiers-rejectors exists where the rejectors disagree. Suppose (h_1, r_1) and (h_2, r_2) are in our current version space. These two pairs can only disagree when on an instance \mathbf{x} : $r_1(\mathbf{x}) \neq r_2(\mathbf{x})$, since otherwise, when both defer, the expert makes the same prediction, and when both do not defer, both classify the label correctly by the realizability assumption. Thus, we define the disagreement set in terms of only the rejectors that are in the version space at each round j :

$$DIS(V_{j-1}) = \{x \in \mathcal{X} \mid \exists (h_1, r_1), (h_2, r_2) \in V_{j-1} \text{ s.t. } r_1(x) \neq r_2(x)\} \quad (2.17)$$

Then we ask for the labels of k instances in $DIS(V_{j-1})$ to form $\mathcal{S}_j = \{(\mathbf{x}_i, y_i, m_i) : \mathbf{x}_i \in DIS(V_{j-1})\}$ and we update the version space as

$$V_j = \{(h, r) \in V_{j-1} \mid \forall (\mathbf{x}, y, m) \in \mathcal{S}_j, \text{ if } r(\mathbf{x}) = 1 \rightarrow y = m\} \quad (2.18)$$

Now, we prove that the above rejector-disagreement algorithm will converge if the optimal unique classifier-rejector pair is unique:

Proposition 3. Assume that there exists a unique pair $(h^*, r^*) \in \mathcal{H} \times \mathcal{R}$ that have zero error $L(h^*, r^*) = 0$. Let Θ be defined as:

$$\Theta = \sup_{t>0} \frac{\mathbb{P}(X \in DIS(B((h^*, r^*), t)))}{t} \quad (2.19)$$

where $B((h, r), t) = \{(h', r') \in \mathcal{H} \times \mathcal{R} : \mathbb{P}(r(X)M + (1 - r(X))h(X) \neq r'(X)M + (1 - r'(X))h'(X)) \leq t\}$.

Then, running the rejector-disagreement algorithm with $k = O(\Theta^2((d(\mathcal{H})+d(\mathcal{R}) \log \Theta + \log \frac{1}{\delta} + \log \log \frac{1}{\epsilon})))$ for $\log(1/\epsilon)$ iterations will return classifier-rejector with ϵ error and with probability at least $1 - \delta$.

Proof of the proposition can be found in Appendix A.6.

2.6.2 Disagreement on Disagreements

If we remove the uniqueness assumption for the rejector-disagreement algorithm in the previous subsection, we show in Appendix A.7 with an example that the algorithm no longer converges as $DIS(V)$ can remain constant. We expect that the uniqueness assumption may not hold in practice, so we now hope to design algorithms that do not require it. Instead, we now make a new assumption that we can learn the error boundary of the human expert via a function $f \in \mathcal{D}$, that is given any sample (x, y, m) we have $f(x) = \mathbb{I}_{y \neq m}$. This assumption is identical to those made in active learning for cost-sensitive classification [59]. This assumption is formalized as follows:

Assumption 2. We assume that there exists $f^* \in \mathcal{D}$ such that $\mathbb{P}(\mathbb{I}_{M \neq Y} \neq f^*(X)) = 0$.

Our new active learning will now seek to take advantage of Assumption directly 2. The algorithm consists of two stages: the first stage runs a standard active learning algorithm, namely CAL, on the hypothesis space \mathcal{D} to learn the expert disagreement with the label with error at most ϵ . In the second stage, we label our data with the predictor \hat{f} that is the output of the first stage, and then learn a classifier-rejector pair from this pseudo-labeled data. Key to this two-stage process is to show that the error from the first stage is not too amplified by the second stage. The algorithm is named Disagreement on Disagreements (DoD) and is described in detail in Algorithm box 1.

In the following, we prove a label complexity guarantee for Algorithm 1.

Theorem 4. Let us define Θ_2 as

$$\Theta_2 = \sup_{t>0} \frac{\mathbb{P}(X \in DIS_2(B_2(f^*, t)))}{r}, \quad (2.20)$$

Algorithm 1: Active Learning algorithm DoD (Disagreement on disagreements)

Input: parameter n_u, T, k , class \mathcal{D}, \mathcal{H} , and \mathcal{R}

1. $V \leftarrow \mathcal{D}$

2. **for** $i \in \{1, \dots, T\}$ **do**

 Sample from μ_X until you have k samples $\{\mathbf{x}_i\}_{i=1}^k$ within $DIS_2(V)$

 Query for $\{(y_i, m_i)\}_{i=1}^k$ for the samples $\{\mathbf{x}_i\}_{i=1}^k$

 Update $V \leftarrow \{d \in V : \forall j d(\mathbf{x}_j) = \mathbb{1}_{m_j \neq y_j}\}$

end

4. Collect n_u samples $\{(\mathbf{x}'_i, y'_i)\}_{i=1}^{n_u}$ from μ_{XY}

Return: $(h, r) \in \mathcal{H} \times \mathcal{R}$ such that $\sum_{(\mathbf{x}'_i, y'_i)} \mathbb{1}_{h(\mathbf{x}'_i) \neq y'_i} (1 - r(\mathbf{x}'_i)) + r(\mathbf{x}'_i) d(\mathbf{x}'_i) = 0$, for some $d \in V$

where $B_2(f, t) = \{f' \in \mathcal{D} \mid \mathbb{P}(f'(X) \neq f(X)) \leq t\}$, and $DIS_2(V) = \{\mathbf{x} \in \mathcal{X} \mid \exists f_1, f_2 \in V, f_1(\mathbf{x}) \neq f_2(\mathbf{x})\}$.

Assume we have $\mathcal{H}, \mathcal{R}, \mathcal{D}$ that satisfy assumption 1 and 2. Then for $n_u = O(\frac{\log 1/\delta + \max\{d(\mathcal{H}), d(\mathcal{R})\} \log 1/\epsilon}{\epsilon^2})$, and $k = d(\mathcal{D})\Theta_2 \log(\frac{\Theta_2}{\delta} \log(\frac{1}{\epsilon}))$, then Algorithm 1 takes $T = O(\log \frac{1}{\epsilon})$ iterations to output a solution with ϵ -upper-bound on deferral loss with probability at least $1 - \delta$. As a result, the sample complexity of labeled data n_l is $O(d(\mathcal{D})\Theta_2 \log(\frac{\Theta_2}{\delta} \log(\frac{1}{\epsilon})) \log(\frac{1}{\epsilon}))$.

Proof of the proposition can be found in Appendix A.8. Recall that in Proposition 1, where the labeled data was chosen at random, the sample complexity n_u is in order $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$. As we see in Theorem 4, the proposed active learning algorithm reduces sample complexity to $O(\log \frac{1}{\epsilon})$, with the caveat that realizability is assumed for active learning. Further, note that for this algorithm, in contrast to the previous subsection, the uniqueness of the consistent pair (h, r) is not needed anymore. However, this algorithm ignores the classifier and rejector classes when querying for points, which makes the sample complexity n_l dependent only on the complexity of \mathcal{D} instead of \mathcal{H}, \mathcal{R} . In the next section, we try to understand how to use surrogate loss functions to practically optimize for our classifier-rejector pair.

2.7 Experimental Illustration

Code for our experiments is found in https://github.com/clinicalml/active_learn_to_defer.

Dataset. We use the CIFAR-10 image classification dataset [60] consisting of 32×32 color images drawn from 10 classes. We consider the human expert models considered in [26]: if the image is in the first 5 classes the human expert is perfect, otherwise the expert predicts randomly. Further experimental details are in Appendix A.9.

Model and Optimization. We parameterize the classifier and rejector by a convolutional neural network consisting of two convolutional layers followed by two feedforward layers. For staged learning, we train the classifier on the training data optimizing for performance on a validation set, and for the rejector we train a network to predict the expert error and defer at test time by comparing the confidence of the classifier and the expert as in [25]. For joint learning, we use the loss L_{CE}^α , a simple extension of the loss (2.12) in [26], optimizing the parameter α on a validation set.

Model Complexity Gap. In Figure 2.2, we plot the difference of accuracy between joint learning and staged learning as we increase the complexity of the classifier class by increasing the filter size of the convolutional layers and the number of units in the feedforward layers. Model complexity is captured by the number of parameters in the classifier which serves only as a rough proxy of the VC dimension that varies in the same direction. The difference is decreasing as predicted by Theorem 1 as we increase the classifier class complexity as we fix the complexity of the rejector.

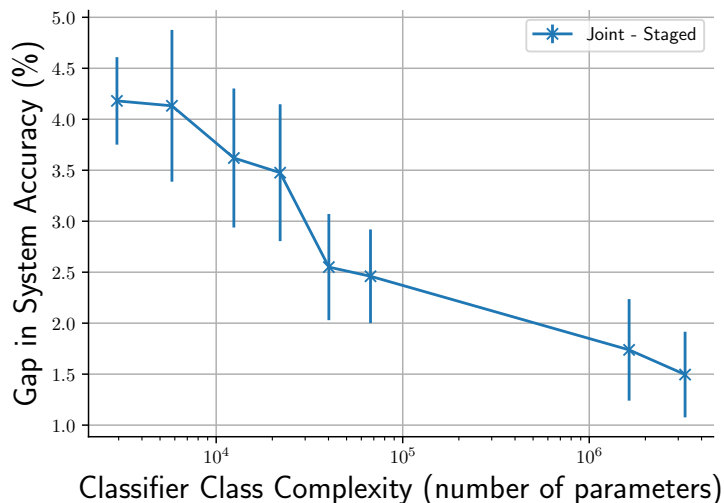


Figure 2.2: Difference of accuracy between joint learning and staged learning of the classifier-rejector pair (y-axis is log scale of number of parameters).

Data Trade-Offs. In Figure 2.3, we plot the of accuracy between joint learning and staged learning when only a subset of the data is labeled by the expert as in Section 2.4.2. We plot the average difference across 10 trials, and error bars denote the standard deviation. We only plot the performance of joint learning when initialized first on the unlabeled data to predict the target and then trained on the labeled expert data to defer, we denote this approach as 'Joint-SemiSupervised'. For staged learning, the classifier is trained on all of the data $S_l \cup S_u$, while for joint learning we only train on S_l . We can see that when there is more unlabeled data than labeled, staged learning outperforms joint learning in accordance with Proposition 1. The heuristic method 'Joint-

SemiSupervised’ improves on the sample complexity of ‘Joint’ but still lags behind the Staged approach in low data regimes.

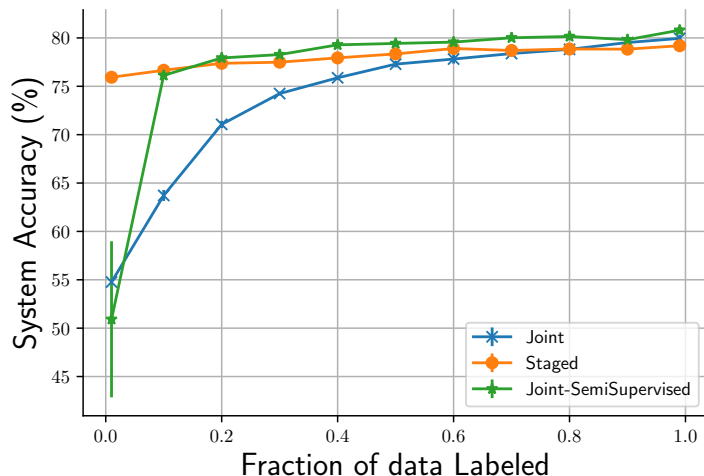


Figure 2.3: Performance of joint learning and staged learning as we increase the ratio of the data labeled by the expert $\frac{n_l}{n_u+n_l}$.

DoD algorithm. In Figure 2.4, we plot corresponding errors of the DoD algorithm, and we compare them to the staged and joint learning. The features \mathbf{x} of the synthetic data in here is generated from a uniform distribution on interval $[0, 1]$, and the labels y are equal to 1 where $\mathbf{x} > 0.3$ (full-information region) and are equal to random outcomes of a *Bernoulli*(0.5) distribution otherwise (no-information region). The human’s decision is inaccurate ($M \neq Y$) for $X > 0.3$ and accurate ($M = Y$) otherwise. We further assume each hypothesis class of rejectors and classifiers be 100 samples of half-spaces over the interval $[0, 1]$. The error plotted in Figure 2.4 is an average of 1000 random generations of training data. The test set is formed by $N_{test} = 1000$ samples that are generated from the same distribution as training data. Here, we note that the number of unlabeled data in staged learning is set $N_u = 100$. The result of this experiment shows that in the DoD algorithm, one needs less number of samples that are labeled by human to reach a similar level of error.

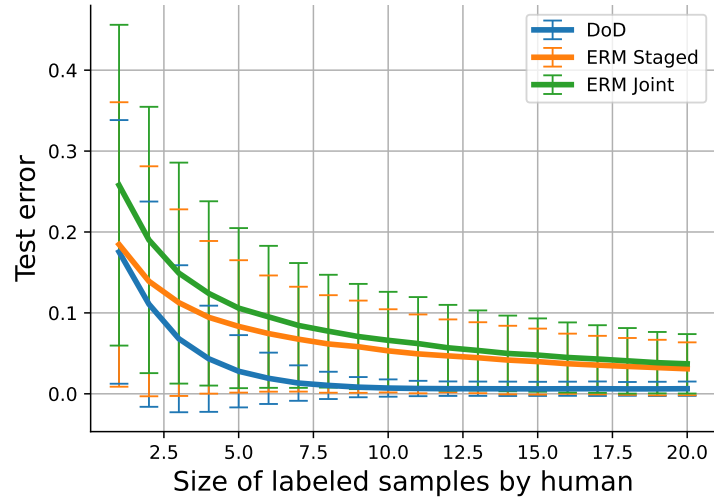


Figure 2.4: Error of the DoD algorithm compared to staged and joint learning for increasing number of training data that are labeled by human.

Chapter 3

Exact Algorithms for Learning to Defer

Acknowledgements of Co-authors. This chapter is based on the published work in [61]. I would like to thank my co-authors, Hunter Lang, Dennis Wei, Prasanna Sattigeri, and Subhro Das, for their help. Hunter Lang was essential in helping prove some of the results in this work especially on computational complexity.

3.1 Introduction

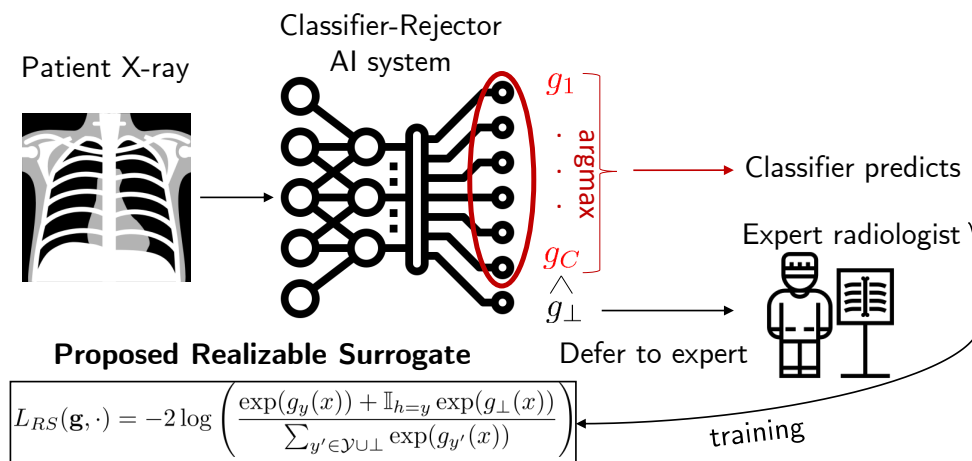


Figure 3.1: The learning to defer setting with the Realizable Surrogate illustrated in the application of making predictions for chest X-rays.

The goal of this work is to jointly learn a classifier that can predict pneumonia and a *rejector*, which decides on each data point whether the classifier or the human should predict illustrated in Figure 3.1.

Failure of Prior Approaches. Existing literature has focused on *surrogate loss functions* for

deferral [26], [28], [62] including our work in Chapter 2 and confidence-based approaches [25], [27]. We give a simple synthetic setting where all of these approaches fail to find a classifier/rejector pair with a low system error. In this setting, there exists a halfspace classifier and halfspace rejector that have zero system error (illustrated in Figure 3.2), but our experiments in Section 3.7.2 demonstrate that all prior approaches fail to find a good classifier/rejector pair in this setting.

To understand possible reasons for this failure, we first study the **computational complexity** of learning with deferral using halfspaces for the rejector and the classifier, which we call LWD-H. The computational complexity of learning with deferral has received little attention in the literature. We prove that even in our simple setting where the data is realizable (i.e., there exists a halfspace classifier and halfspace rejector achieving zero system error), there is no polynomial-time algorithm that finds an approximately optimal pair of halfspaces unless $NP = RP$. We also extend our hardness result to approximation algorithms and when the data is not realizable by halfspaces. In contrast, training a linear classifier in the realizable linear setting can be solved in polynomial time with linear programming [63].

Learning with deferral using halfspaces is also of significant *practical* importance. Sample efficiency is critical in learning with deferral since the training data is expensive to collect—it requires both human outputs *and* ground-truth labels. This motivates restricting to smaller model classes, and in particular to linear classifiers and rejectors. Linear models have the benefit of being interpretable with respect to the underlying features, which can be crucial for a human-AI deferral system. Additionally, the *head tuning* or *linear probing* paradigm, where only the final (linear) layer of a pretrained deep neural network is fine-tuned on different tasks, has become increasingly common as pretrained representations improve in quality, and it can be more robust than full fine-tuning [64]. However, as previously mentioned, existing surrogate approaches fail to find a good linear classifier and rejector even when one is guaranteed to exist. This motivates the need for an algorithm for **exact minimization** of the system training error.

We show that exact minimization of the system error can be formulated as a **mixed integer linear program** (MILP). This derivation overcomes a naive quadratic formulation of the problem. In addition to exactly minimizing the training loss, the MILP formulation allows us to easily incorporate constraints to govern the human-AI system. We show that modern commercial solvers such as Gurobi [65] are capable of solving fairly large instances of this MILP, making it a practical algorithm for the LWD-H problem. To obtain similar gains over prior approaches, but with a more scalable algorithm, we develop a new differentiable surrogate loss function L_{RS} , dubbed **RealizableSurrogate**, that can solve the LWD-H problem in the realizable setting by virtue of being *realizable-consistent* [66] for a large class of hypothesis sets that includes halfspace classifier/rejector pairs. We also show empirically that L_{RS} is competitive with prior work in the non-linear setting.

To summarize, the contributions of this chapter are the following:

- **Computational Complexity of Deferral:** We prove the computational hardness of PAC-learning with deferral in the linear setting.
- **Mixed Integer Linear Program Formulation and Realizable Surrogate :** We show how to formulate learning to defer with halfspaces as a MILP and provide a novel surrogate loss.
- **Experimental Evaluation:** We showcase the performance of our algorithms on a wide array of datasets and compare them to several existing baselines. We contribute a publicly available repository with implementations of existing baselines and datasets: https://github.com/clinicalml/human_ai_deferral

3.2 Related Work

A natural baseline for the learning to defer problem is to first learn a classifier that minimizes average misclassification error, then learn a model that predicts the probability that the human makes an error on a given example, and finally defer if the probability that the classifier makes an error is higher than that of the human. This is the approach adapted by [25]. However, this does not allow the classifier to adapt to the human. Another natural approach is to model this problem as a mixture of experts: the human and the AI. This is the approach introduced by [28] and adapted by [29], [30] by introducing a mixture of experts surrogates. However, this approach has been shown to fail empirically as the loss is not easily amenable to optimization. Subsequent work [26] introduced consistent surrogate loss functions for the learning with deferral objective, with follow-up approaches addressing limitations including better calibration [31], [32]. Another consistent convex surrogate was proposed by [62] via a one-vs-all approach. [23] provides a family of convex surrogates for learning with deferral that generalizes prior approaches, however, our proposed surrogate does not belong to that family. [33] proposes a surrogate loss which is the sum of the loss of learning the classifier and rejector separately but that is not a consistent surrogate. [67] proved the hardness of linear regression (not classification) where some training points are allocated to the human (not deferral but subset selection of the training data). Finally, [27] proposes a method that iteratively optimizes the classifier on points where it outperforms the human on the training sample, and then learns a post-hoc rejector to predict who between the human and the AI has higher error on each point. The setting when the cost of deferral is constant has a long history in machine learning and goes by the name of rejection learning [44]–[47] or selective classification (only predict on $x\%$ of data) [48]–[51]. Our MILP formulation is inspired by work in binary linear classification that optimizes the 0-1 loss exactly [68], [69].

3.3 Learning with Deferral: Problem Setup

We frame the learning with deferral setting as the task of predicting a target $Y \in \mathcal{Y} = \{1, \dots, C\}$. The classifier has access to features $X \in \mathcal{X} = \mathbb{R}^d$, while the human (also referred to as the expert) has access to a potentially different set of features $Z \in \mathcal{Z}$ which may include side-information beyond X . The human is modeled as a fixed predictor $h : \mathcal{Z} \rightarrow \mathcal{Y}$. The AI system consists of a classifier $m : \mathcal{X} \rightarrow \mathcal{Y}$ and a rejector $r : \mathcal{X} \rightarrow \{0, 1\}$. When $r(x) = 1$, the prediction is deferred to the human and we incur a cost if the human makes an error, plus an additional, optional penalty term: $\ell_{\text{HUM}}(x, y, h) = \mathbb{I}_{h \neq y} + c_{\text{HUM}}(x, y, h)$. When $r(x) = 0$, then the classifier makes the final decision and incurs a cost with a different optional penalty term: $\ell_{\text{AI}}(x, y, m) = \mathbb{I}_{m \neq y} + c_{\text{AI}}(x, y, m)$. We can put this together into a loss function for the classifier and rejector:

$$L_{\text{def}}(m, r) = \mathbb{E}_{X, Y, Z} [\ell_{\text{AI}}(X, Y, m(X)) \cdot \mathbb{I}_{r(X)=0} + \ell_{\text{HUM}}(X, Y, h(Z)) \cdot \mathbb{I}_{r(X)=1}].$$

In this chapter we focus mostly on the cost of misclassification with no additional penalties, the deferral loss becomes a misclassification loss $L_{\text{def}}^{0-1}(m, r)$ for the human-AI system, and the optimization problem is:

$$\underset{m, r}{\text{minimize}} L_{\text{def}}^{0-1}(m, r) := \mathbf{P} [((1 - r(X))m(X) + r(X)h(Z)) \neq Y]. \quad (3.1)$$

Constraints. We may wish to constrain the behavior of the human-AI team when learning the classifier and rejector pair. For example, we may have a limit on the percentage of times the AI can defer to the human, due to the limited time the human may have. We express this as a **coverage** constraint:

$$\mathbb{P}(r(X) = 1) \leq \beta. \quad (3.2)$$

Finally, it is desirable that our system does not perform differently across different demographic groups. Let $A \in \{1, \dots, |A|\}$ denote the demographic identity of an individual. Then if we wish to equalize the error rate across demographic groups, we impose the **fairness** constraint $\forall a \in A$:

$$\begin{aligned} & \mathbb{P}((1 - r(X))m(X) + r(X)h(Z) \neq Y | A \neq a) \\ & = \mathbb{P}((1 - r(X))m(X) + r(X)h(Z) \neq Y | A = a) \end{aligned} \quad (3.3)$$

Data. We assume access to samples $S = \{(x_i, h(z_i), y_i)\}_{i=1}^n$ where $h(z_i)$ is the human's prediction on the example, but note that we do not observe z_i , the information used by the human. We emphasize that the label y_i and human prediction $h(z_i)$ are different, even though y_i could also

come from humans. For example in our chest X-ray classification example, y_i could come from a consensus of 3 or more radiologists, while $h(z_i)$ is the prediction of a single radiologist not involved with the label. Given the dataset S the system *training loss* is given by:

$$\hat{L}_{\text{def}}^{0-1}(m, r) := \frac{1}{n} \sum_{i=1}^n \mathbb{I}_{m(x_i) \neq y_i} \mathbb{I}_{r(x_i)=0} + \mathbb{I}_{h(z_i) \neq y_i} \mathbb{I}_{r(x_i)=1} \quad (3.4)$$

In the following section, we study the computational complexity of learning with deferral using halfspaces, which reduces to studying the optimization problem (3.4) when m and r are constrained to be halfspaces.

3.4 Computational Complexity of Learning with Deferral

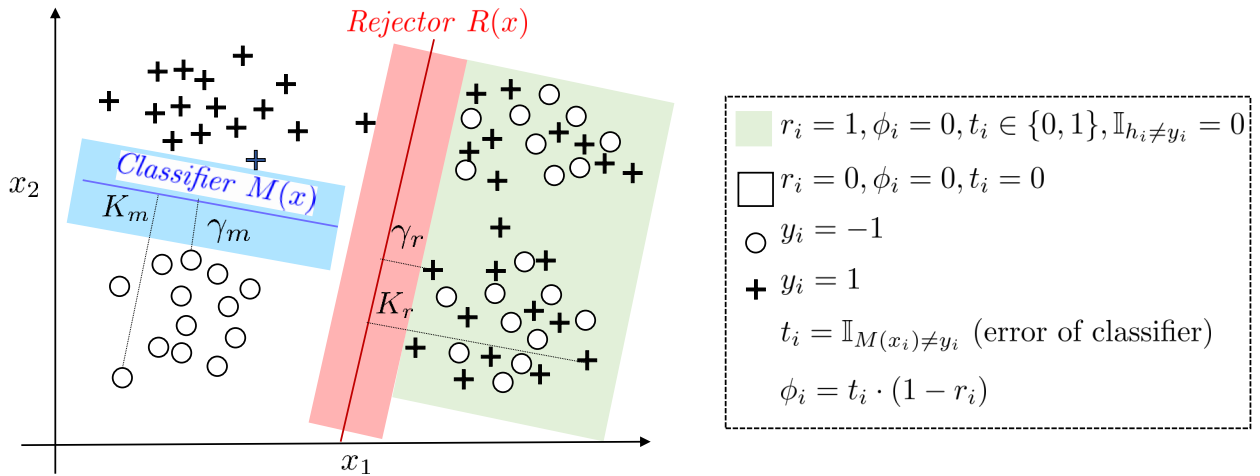


Figure 3.2: The realizable LWD-H setting illustrated. The task is binary classification with labels $\{o, +\}$; the human is perfect on the green-shaded region, and the data outside the green region is linearly separable. As a result, the optimal classifier and rejector obtain zero error. Assumption 2 is illustrated graphically as well as the MILP variables of equations (3.9)-(3.14).

The misclassification error of the human-AI team in equation (3.1) is challenging to optimize as it requires searching over a joint set of functions for the classifier and rejector, in addition to dealing with the nonconvex 0-1 aspect. To study the computational complexity of minimizing the loss, we restrict our attention to a foundational setting: linear classifiers and linear rejectors in the binary label scenario.

We begin with the realizable case when there exists a halfspace classifier and rejector that can achieve zero loss:

Assumption 1 (Realizable Linear Setting). Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$. We assume that for the given expert h there exists a linear classifier $m^*(x) = \mathbb{I}_{M^\top x > 0}$ and a linear rejector $r^*(x) = \mathbb{I}_{R^\top x > 0}$ that achieve 0 error:

$$\mathbb{E}_{(x,y,z) \sim \mathbf{P}} [\mathbb{I}_{m^*(x) \neq y} \mathbb{I}_{r^*(x)=0} + \mathbb{I}_{h(z) \neq y} \mathbb{I}_{r^*(x)=1}] = 0.$$

This setting is illustrated in Figure 3.2. Since the decision regions of m and r are halfspaces, we also use the term “halfspace” interchangeably. Note that while the classifier is assumed to be linear, the human can have a non-linear decision boundary. The analog of this assumption in the binary classification without deferral setting is to assume that there exists a halfspace that can correctly classify all the data points. In that case, we can formulate the optimization problem as a linear program to efficiently find the optimal solution [63].

Hardness. In contrast to learning without deferral, we will prove that in general, it is computationally hard to learn a linear m and r under Assumption 1. Define the *learning with deferral using halfspaces* (LWD-H) problem as that of finding halfspace m and halfspace r such that the system error in (3.1) is approximately minimized.

Theorem 5. Let $\epsilon > 0$ be an arbitrarily small constant. Under a guarantee that there exist halfspaces m^*, r^* with zero system loss (Assumption 1), there is no polynomial-time algorithm to find a pair of classifier-rejector halfspaces with error $1/2 - \epsilon$ unless $NP = RP$.

This shows that even in the realizable setting (i.e., there exists a pair of halfspaces with zero system loss), it is hard to find a pair of halfspaces that even gets system error $1/2 - \epsilon$.

Corollary 6. There is no efficient proper PAC-learner for LWD-H unless $NP = RP$.

Proof Sketch. First, because the true distribution of points could be supported on a finite set, the LWD-H problem boils down to approximately minimizing the training loss (3.4). Our proof relies on a reduction from the problem of learning an intersection of two halfspaces in the realizable setting. Let $D = \{x_i, y_i\}_{i=1}^n$ and suppose there exists an intersection of two half-spaces g_1, g_2 that achieve 0 error for D . This is an instance of learning an intersection of two halfspaces in the realizable setting, which is hard to even *weakly* learn [70]. We show that this is an instance of the realizable LWD-H problem by setting $m = g_1$ and $r = \bar{g}_2$ and the human H to always predict 0. Hence, an algorithm for efficiently finding a classifier/rejector pair with error $\frac{1}{2} - \epsilon$ would also find an intersection of halfspaces with error $\frac{1}{2} - \epsilon$, which is hard unless $NP = RP$. \square

All proofs can be found in the Appendix. This hardness result holds in the realizable setting, with proper learning, and with no further assumptions on the data distribution.

Extensions. Even if the problem is not realizable and the goal is to find an approximation algorithm, this is still computationally hard as presented in the following corollary.

Corollary 1. *When the data is not realizable (i.e., Assumption 1 is violated), there is no polytime algorithm for finding a pair of halfspaces with error $\frac{1}{2} - \epsilon$ unless $NP = RP$.*

Exact Solution. These hardness results motivate the need for new approaches to solving the LWD-H problem. In the next section, we derive a scheme to exactly minimize the misclassification error of the human/AI system using mixed-integer linear programming (MILP).

3.5 Mixed Integer Linear Program Formulation

In the previous section, we saw that in the linear setting it is computationally hard to learn an optimal classifier and rejector pair. As discussed in the introduction, we are interested in the linear setting due to the cost of labeling large datasets for learning with deferral. Linear predictors can perform similar to non-linear predictors in applications involving high-dimensional medical data [71]. Moreover, we can rely on pre-trained representations, which can allow linear predictors on top of embedded representations to attain performance comparable to non-linear predictors [72].

A First Formulation. As a first step, we write down a mixed integer *nonlinear* program for the optimization of the training loss $\hat{L}_{\text{def}}^{0-1}$ in (3.4) over linear classifiers and linear rejectors with binary labels. For simplicity, let $\mathcal{Y} = \{-1, +1\}$. A direct translation of (3.4) with halfspace classifiers and rejectors yields the following:

$$M^*, R^*, \cdot = \arg \min_{M, R, m_i, r_i} \sum_{i=1}^n (1 - r_i) \mathbb{I}_{m_i \neq y_i} + r_i \mathbb{I}_{h_i \neq y_i} \quad (3.5)$$

$$\begin{aligned} \text{s.t. } m_i &= \text{sign}(M^\top x_i), \quad r_i = \mathbb{I}_{R^\top x_i \geq 0} \quad \forall i \in [n], \\ M &\in \mathbb{R}^d, R \in \mathbb{R}^d. \end{aligned} \quad (3.6)$$

The variables m_i and r_i are simply the binary outputs of the classifier and rejector. We observe that the objective involves a quadratic interaction between the classifier and rejector. Furthermore, the constraints (3.6) are indicator constraints that are difficult to optimize.

Making Objective Linear. We observe that since the r_i 's are binary, the term $(1 - r_i) \mathbb{I}_{m_i \neq y_i}$ can be equivalently rewritten as $\max(0, \mathbb{I}_{m_i \neq y_i} - r_i)$. This is a crucial simplification that avoids having a mixed integer quadratic program. Below we use this to create a binary variable $t_i = \mathbb{I}_{m_i \neq y_i}$ representing the error of the classifier and a second *continuous* variable ϕ_i that upper bounds $\max(0, t_i - r_i)$ and represents the classifier error after accounting for deferral.

Making Constraints Linear. Constraints (3.6) make sure that the binary variables r_i and m_i are the predictions of half-spaces R and M respectively. As mentioned above, we will formulate the problem using the classifier error variables t_i instead of the classifier predictions m_i . To reformulate constraints (3.6) in a linear fashion, we have to make assumptions on the optimal M and R :

Assumption 2 (Margin). *The optimal solution (M, R) that minimizes the training loss (3.4) has margin and is bounded, meaning that (M, R) satisfy the following for all $i \in [n]$ in the training set and some $\gamma_m, \gamma_r, K_m, K_r > 0$:*

$$\gamma_m \leq |M^\top x_i| \leq K_m - \gamma_m, \quad \gamma_r \leq |R^\top x_i| \leq K_r - \gamma_r \quad (3.7)$$

A similar assumption is made in [68]. The upper bounds in (3.7) are often naturally satisfied as we usually deal with bounded feature sets \mathcal{X} such that we can normalize x_i to have unit norm, and the norms of M and R are constrained for regularization.

Mixed Integer Linear Program. With the above ingredients and taking inspiration from the big-M approach of [68], we can write down the resulting mixed integer linear program (MILP):

$$M^*, R^*, \dots =$$

$$\arg \min_{M, R, \{r_i\}, \{t_i\}, \{\phi_i\}} \sum_i \phi_i + r_i \mathbb{I}_{h_i \neq y_i}, \quad s.t. \quad (3.8)$$

$$\phi_i \geq t_i - r_i, \quad \phi_i \geq 0 \quad \forall i \in [n] \quad (3.9)$$

$$K_m t_i \geq \gamma_m - y_i M^\top x_i \quad \forall i \in [n] \quad (3.10)$$

$$R^\top x_i \leq K_r r_i + \gamma_r (r_i - 1), \quad (3.11)$$

$$R^\top x_i \geq K_r (r_i - 1) + \gamma_r r_i \quad \forall i \in [n] \quad (3.12)$$

$$r_i \in \{0, 1\}, t_i \in \{0, 1\}, \quad (3.13)$$

$$\phi_i \in \mathbb{R}^+ \quad \forall i \in [n], M, R \in \mathbb{R}^d \quad (3.14)$$

Please see Figure 3.2 for a graphical illustration of the variables. We show that constraints (3.12) function as intended; the rest of the constraints are verified in the Appendix. When $r_i = 0$, then we have the constraints $R^\top x_i \leq -\gamma_r$ and $R^\top x_i \geq -K_r$: this correctly forces the rejector to be negative. When $r_i = 1$, we have $R^\top x_i \geq \gamma_r$ and $R^\top x_i \leq K_r$: which means the rejector is positive. Note that we do not need to know the margin γ_r exactly, only a lower bound γ , $0 < \gamma \leq \gamma_r$; the formulation is still correct with γ in place of γ_r . However, we cannot set $\gamma = 0$ as then the trivial solution $R = 0$ is feasible and the constraint is void. The same statements apply to γ_m . This MILP has $2n$ binary variables, $n + 2d$ continuous variables and $4n$ constraints. Finally, note that the MILP minimizes the 0-1 error even when Assumption 1 is violated.

Regularization and Extension to Multiclass. We can add l_1 regularization to our model by adding the l_1 norm of both M and R to the objective. This is done by defining two sets of variables constrained to be the l_1 norm of the classifier and rejector and adding their values to the objective in (3.9). Adding regularization can help prevent the MILP solution from overfitting to the training data. The above MILP only applies to binary labels but can be generalized to the multi-class setting

where $\mathcal{Y} = \{1, \dots, C\}$ (see Appendix).

Generalization Bound. Under Assumption 2 and non-realizability, assume $\|x_i\|_1 \leq 1$ and constrain the search of the MILP to M and R with infinity norms of at most K_m and K_r respectively. We can relate the performance of MILP solution on the training set to the population 0-1 error.

Proposition 4. *For any expert h and data distribution \mathbf{P} over $\mathcal{X} \times \mathcal{Y}$ that satisfies Assumption 2, let $0 < \delta < \frac{1}{2}$. Then with probability at least $1 - \delta$, the following holds for the empirical minimizers (\hat{m}^*, \hat{r}^*) obtained by the MILP:*

$$L_{0-1}(\hat{m}^*, \hat{r}^*) \leq \hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) + \frac{(K_m + K_r)d\sqrt{2\log d} + 10\sqrt{\log(2/\delta)}}{\sqrt{n\mathbb{P}(h(Z) \neq Y)}}.$$

This bound improves on surrogate optimization since the MILP will achieve a lower training error, $\hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*)$, than the surrogate, which optimizes a different objective.

Adding Constraints. A major advantage of the MILP formulation is that it allows us to provably integrate any linear constraints on the variables with ease. For example, the constraints mentioned in the problem setting can be added to the MILP as follows in a single constraint:

- Coverage: $\sum_i r_i/n \leq \beta$
- Fairness: $\sum_{i:A=1} (\phi_i + r_i \mathbb{I}_{h_i \neq y_i}) / |\{i : A = 1\}| = \sum_{i:A=0} (\phi_i + r_i \mathbb{I}_{h_i \neq y_i}) / |\{i : A = 0\}|$.

So far, we have provided an exact solution to the linear learning to defer problem. However, the MILP requires significant computational time to find an exact solution for large datasets. Moreover, we might need a non-linear classifier or rejector to achieve good error. The remaining questions are (i) how to efficiently find a good pair of halfspaces for large datasets and (ii) how to generalize to non-linear predictors. In the following section, we give a novel surrogate loss function that is optimal in the realizable LWD-H setting, performs well with non-linear predictors, and can be efficiently minimized (to a local optimum).

3.6 Realizable Consistent Surrogate

3.6.1 Consistency vs Realizable Consistency

Most machine learning practice is based on optimizing surrogate loss functions of the true loss that one cares about. The surrogate loss functions are chosen so that optimizing them also optimizes the true loss functions, and also chosen to be differentiable so that they are readily optimized. This first property is captured by the notion of *consistency*, which was the main focus of much of the prior

work on expert deferral: [23], [26], [62]. We start by giving a formal definition of the consistency of a surrogate loss function:

Definition 1 (Consistency¹). A surrogate loss function $\tilde{L}(m, r)$ is a consistent loss function for another loss $L_{\text{def}}^{0-1}(m, r)$ if optimizing the surrogate over all measurable functions is equivalent to minimizing the original loss.

For example, the surrogates L_{CE} and Ψ_{OvA} both satisfy consistency for $L_{\text{def}}^{0-1}(m, r)$ [26], [62]. It is crucial to note that consistency only applies when optimizing over *all measurable functions*. Conversely, in LWD-H, and in the setting of Figure 3.2, when we optimize with linear functions, consistency does not provide any guarantees, which explains why these methods can fail in that setting.

Since we normally optimize over a restricted model class, we want our guarantee for the surrogate to also hold for optimization under a certain model class. The notion of realizable \mathcal{H} -consistency is such a guarantee that has proven fruitful for classification [66], [74] and was extended by [26] for learning with deferral. We recall the notion when extended for learning with deferral:

Definition 2 (realizable $(\mathcal{M}, \mathcal{R})$ -consistency). A surrogate loss function $\tilde{L}(m, r)$ is a realizable $(\mathcal{M}, \mathcal{R})$ -consistent loss function for the loss $L_{\text{def}}^{0-1}(m, r)$ if there exists a zero error solution $m^*, r^* \in \mathcal{M} \times \mathcal{R}$ with $L_{\text{def}}^{0-1}(m^*, r^*) = 0$. Then optimizing the surrogate returns such zero error solution:

$$\tilde{m}, \tilde{r} \in \arg \inf_{m, r \in \mathcal{M} \times \mathcal{R}} \tilde{L}(m, r) \implies L_{\text{def}}^{0-1}(\tilde{m}, \tilde{r}) = 0$$

Realizable $(\mathcal{M}, \mathcal{R})$ -consistency guarantees that when our data comes from some ground-truth $m^*, r^* \in \mathcal{M} \times \mathcal{R}$, then minimizing the (population) surrogate loss will find an optimal (m, r) pair. We propose a novel, differentiable, and $(\mathcal{M}, \mathcal{R})$ -consistent surrogate for learning with deferral when \mathcal{M} and \mathcal{R} are *closed under scaling*. A class \mathcal{G} of scoring functions from \mathcal{X} to \mathbb{R}^C is closed under scaling if $\mathbf{g} \in \mathcal{G} \implies \alpha \mathbf{g} \in \mathcal{G}$ for any $\alpha \in \mathbb{R}$. For example, we can let \mathcal{G} be the class of linear scoring functions $\mathbf{g}(x) = G^\top x$ and $G \in \mathbb{R}^{d \times C}$. Our results hold for arbitrary \mathcal{G} that are closed under scaling, e.g., ReLU feedforward neural networks (FNN). We parameterize the (m, r) pair with $|\mathcal{Y}| + 1$ dimensional scoring function $\mathbf{g} : (g_1, \dots, g_{|\mathcal{Y}|}, g_\perp)$. We define $m(x) = \arg \max_{y \in \mathcal{Y}} g_y(x)$ and $r(x) = \mathbb{I}_{\max_{y \in \mathcal{Y}} g_y(x) \leq g_\perp(x)}$. The joint classifier-rejector model class $(\mathcal{M}, \mathcal{R})$ is thus defined by \mathcal{G} , and we say $(\mathcal{M}, \mathcal{R})$ is closed under scaling whenever \mathcal{G} is closed under scaling. The proposed new surrogate loss L_{RS} , dubbed RealizableSurrogate, is defined at each point (x, y, h) as:

¹This is also referred to as Fisher consistency [73] and classification-calibration [55].

$$L_{RS}(\mathbf{g}, \cdot) = -2 \log \left(\frac{\exp(g_y(x)) + \mathbb{I}_{h=y} \exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right) \quad (3.15)$$

3.6.2 Derivation of Surrogate

We now derive our proposed surrogate RealizableSurrogate with a principled approach. In this chapter, our primary goal is to predict a target Y given a set of covariates X while having the ability to query a human H to predict. Our overall predictor is denoted as \hat{Y} , a function of both H and X , our goal is learning a predictor that maximizes the agreement between \hat{Y} and Y :

$$\mathbb{E}[\mathbb{I}_{\hat{Y}(X,H)=Y}] = \mathbb{E}_X \left[\mathbb{P}(\hat{Y}(x, H) = Y | X = x) \right]$$

It will be easier to maximize the logarithm of the probability and thus using Jensen's inequality we obtain an upper bound :

$$\log \left(\mathbb{E}_X \left[\mathbb{P}(\hat{Y}(x, H) = Y | X = x) \right] \right) \leq \mathbb{E}_X \left[\log \left(\mathbb{P}(\hat{Y}(x, H) = Y | X = x) \right) \right]$$

We now decompose our predictor into a classifier-rejector pair (m, r) where the rejector decides if the classifier or the human should predict. This transforms our objective to:

$$\begin{aligned} L &= \mathbb{E}_X \left[\log \left(\mathbb{P}(\hat{Y} = Y | X = x) \right) \right] = \\ &\mathbb{E}_X [\log(\mathbb{P}(\hat{Y} = Y | X = x, r(x) = 0) \mathbb{P}(r(x) = 0 | X = x) \\ &+ \mathbb{P}(\hat{Y} = Y | X, r(x) = 1) \mathbb{P}(r(x) = 1 | X = x))] \\ &= \mathbb{E}_X [\log(\mathbb{P}(m(x) = Y | X = x) \mathbb{P}(r(x) = 0 | X = x) + \mathbb{P}(H = Y | X = x) \mathbb{P}(r(x) = 1 | X = x))] \end{aligned} \quad (3.16)$$

In [28], their proposed loss splits the sum inside the log above into a sum of log-likelihoods of the classifier and expert each weighted by the probability of predicting and deferring respectively. Instead in this work, we try to optimize the above log likelihood L (3.16) directly.

Parameterization. We now try to understand how we can parameterize the classifier-rejector pair. We first parameterize the classifier with a set of scoring functions $g_y : \mathcal{X} \rightarrow \mathbb{R}$ for $y \in \mathcal{Y}$ and define the classifier as the label y that attains the maximum value among the set $\{g_{y'}\}_{y' \in \mathcal{Y}}$. To parameterize the rejector r , we define a single scoring function $g_{\perp} : \mathcal{X} \rightarrow \mathbb{R}$ and defer if

$g_{\perp} > \max_y g_y$ which induces a comparison between the function g_{\perp} and the classifier scores. One could instead parameterize the rejector r with a single function $g_{\perp} : \mathcal{X} \rightarrow \mathbb{R}$ and defer if $g_{\perp}(x)$ is positive, we find empirically that the previous parameterization has better performance ².

However, with the characterization, both our classifier and rejector are deterministic. Plugging in the parameterization of (m, r) into the loss in (3.16) would result in a loss function that is non-differentiable in the parameters g_y and g_{\perp} due to thresholding. Instead, we allow the classifier and rejector to only be probabilistic during training by defining:

$$\mathbb{P}(m(x) = Y|X = x) = \frac{\exp(g_Y(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))}, \quad \mathbb{P}(r(x) = 1|X = x) = \frac{\exp(g_{\perp}(x))}{\sum_{i \in \mathcal{Y} \cup \perp} \exp(g_i(x))}$$

This transforms the log likelihood L to:

$$\begin{aligned} & \mathbb{E}_X \left[\log \left(\frac{\exp(g_Y(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))} \cdot \frac{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))}{\sum_{i \in \mathcal{Y} \cup \perp} \exp(g_i(x))} + \mathbb{P}(H = Y|X) \cdot \frac{\exp(g_{\perp}(x))}{\sum_{i \in \mathcal{Y} \cup \perp} \exp(g_i(x))} \right) \right] \\ &= -\mathbb{E}_X \left[\log \left(\frac{\exp(g_Y(x)) + \mathbb{P}(H = Y|X) \exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))} \right) \right] \end{aligned}$$

We multiple the above likelihood by -2 so that we can instead minimize a loss and so that it becomes an upper bound of the 0 – 1 deferral loss $L_{\text{def}}^{0-1}(m, r)$. Given the dataset S , our proposed loss then becomes:

$$L_{RS}^S = -2 \sum_{i=1}^n \log \left(\frac{\exp(g_{y_i}(x_i)) + \mathbb{I}_{h(z_i)=y_i} \exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))} \right) \quad (3.17)$$

3.6.3 Theoretical Guarantees

Notice in our proposed loss L_{RS} that when the human is incorrect, i.e. $\mathbb{I}_{h=y} = 0$, the loss incentivizes the classifier to be correct, similar to cross entropy loss. However, when the human is correct, the learner has the *choice* to either fit the target or defer: there is no penalty for choosing to do one or the other. This is what enables the classifier to complement the human and differentiates L_{RS} from prior surrogates, such as L_{CE} [26], that are not realizable-consistent (see Theorem 14 in Appendix B.4.4) and penalize the learner for not fitting the target even when deferring. This

²This parameterization form can achieve a halfspace rejector and results in the following loss:

$$L_{RS2} = \mathbb{E}_X \left[\log \left(\frac{\exp(g_Y(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))} \frac{1}{1 + \exp(g_{\perp})} + \mathbb{P}(H = Y|X) \cdot \frac{\exp(g_{\perp})}{1 + \exp(g_{\perp})} \right) \right]$$

property is showcased by the fact that our surrogate is realizable $(\mathcal{M}, \mathcal{R})$ -consistent for model classes that are closed under scaling. Moreover, it is an upper bound of the true loss $L_{\text{def}}^{0-1}(m, r)$. The theorem below characterizes the properties of our novel surrogate function.

Theorem 7. *The RealizableSurrogate L_{RS} is a realizable $(\mathcal{M}, \mathcal{R})$ -consistent surrogate for L_{def}^{0-1} for model classes closed under scaling, and satisfies $L_{\text{def}}^{0-1}(m, r) \leq L_{RS}(m, r)$ for all (m, r) .*

This theorem implies that when Assumption 1 is satisfied and \mathcal{G} is the class of linear scoring functions, minimizing L_{RS} yields a classifier-rejector pair with zero system error. The resulting classifier is the halfspace $\mathbb{I}((G_1 - G_0)^\top x \geq 0)$ and the form of the rejector is $\mathbb{I}((G_\perp^\top x - \max(G_1^\top x, G_0^\top x)) \geq 0)$, which is an intersection of halfspaces. One can obtain a halfspace rejector by minimizing instead with the parameterization of L_{RS2} .

The surrogate is differentiable but *non-convex* in \mathbf{g} , though it is convex in each g_i . Indeed, a jointly convex surrogate that provably works in the realizable linear setting would contradict Theorem 5. In practice, we observe that in the linear realizable setting, the local minima reached by gradient descent obtain zero training error despite the nonconvexity. The mixture-of-experts surrogate in [28] is realizable $(\mathcal{M}, \mathcal{R})$ -consistent, non-convex and not classification consistent as shown by [26], however, [26] also showed that it leads to worse empirical results than simple baselines. We have not been able to prove or disprove that RealizableSurrogate is classification-consistent, unlike other surrogates like that of [26]. It remains an open problem to find both a consistent **and** a realizable-consistent surrogate.

3.6.4 Underfitting The Target

Minimizing the proposed loss leads to a classifier that attempts to complement the human. One consequence is that the classifier might have high error on points that are deferred to the human, resulting in possibly high error across a large subset of the data domain. We can explicitly encourage the classifier to fit the target on all points by adding an extra term to the loss:

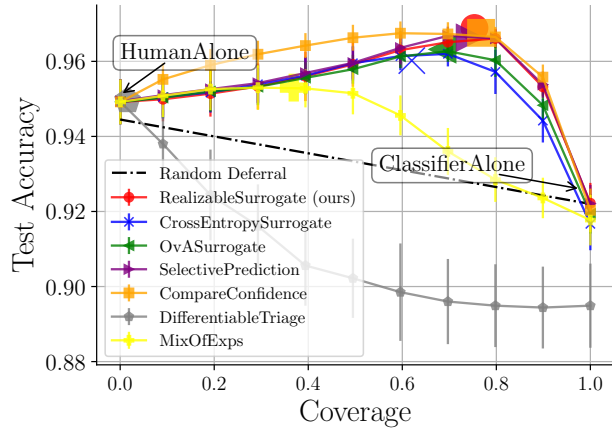
$$L_{RS}^\alpha(\mathbf{g}, x, y, h) = -\alpha \log \left(\frac{\exp(g_y(x) + \mathbb{I}_{h=y} \exp(g_\perp(x)))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right) - (1 - \alpha) \log \left(\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y}} \exp(g_{y'}(x))} \right)$$

(3.18)

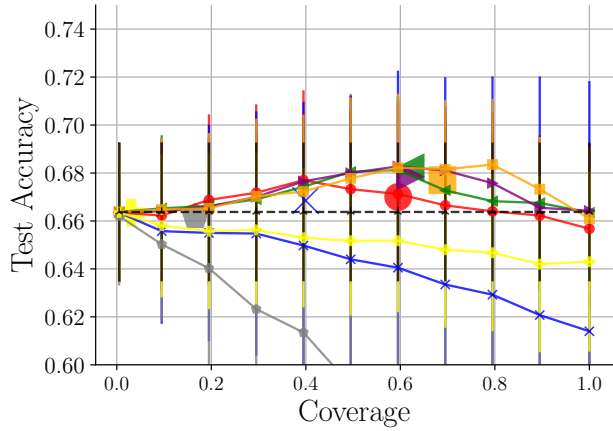
The new loss L_{RS}^α with $\alpha \in [0, 1]$ (a hyperparameter) is a convex combination of L_{RS} and the cross entropy loss for the classifier (with the softmax applied only over the functions g_y rather than including g_\perp). Empirically, this allows the points that are deferred to the human to still help provide extra training signal to the classifier, which is useful for sample-efficiency when training complex,

non-linear hypotheses. Finally, due to adding the parameter α , the loss no longer remains realizable consistent, thus we let the rejector be $r(x) = \mathbb{I}_{g_{\perp}(x) - \max_y g_y(x) \geq \tau}$ and we learn τ with a line search to maximize system accuracy on a validation set. In the next section, we evaluate our approaches with an extensive empirical benchmark.

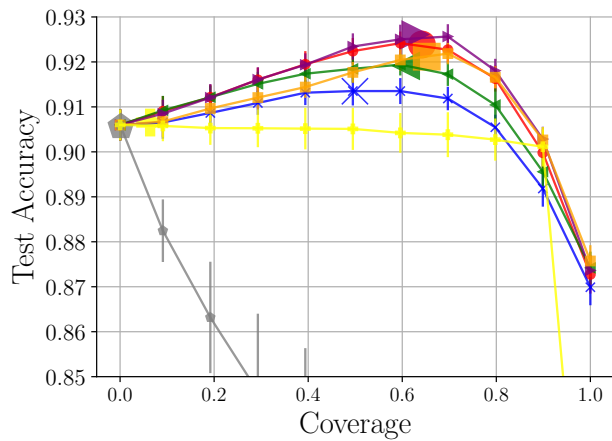
3.7 Experiments



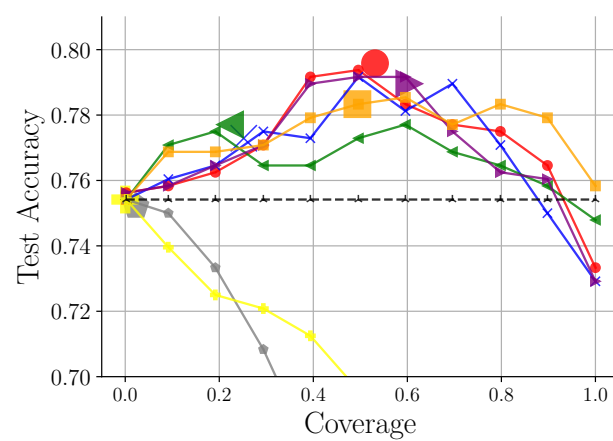
(a) CIFAR-10H



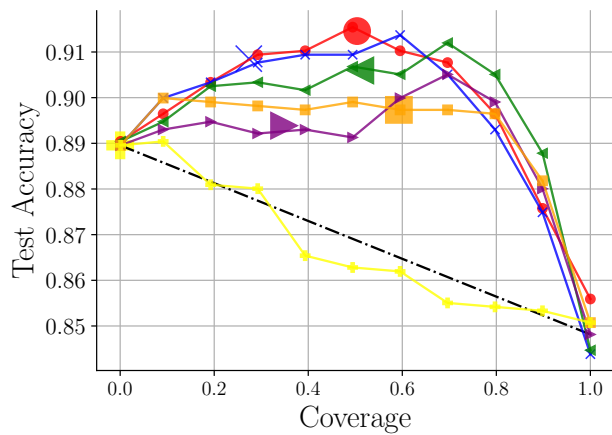
(b) COMPASS



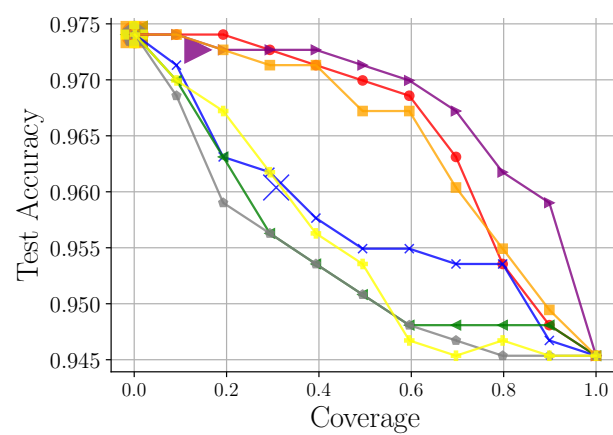
(c) HateSpeech



(d) ImageNet-16H



(e) Chest X-ray - Airspace Opacity

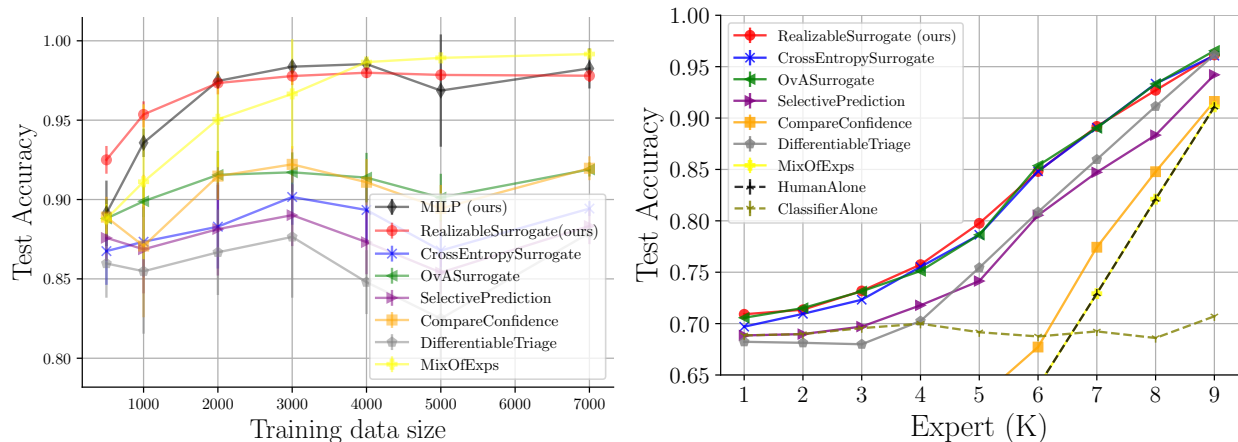


(f) Chest X-ray - Pneumothorax

Figure 3.3: Accuracy vs coverage (fraction of points where classifier predicts) plots across the real world datasets showcasing the behavior of our method and the baselines. On each plot, we showcase the test accuracy of each method with a large marker, with the curve representing varying the rejector threshold on the test set. To achieve different levels of coverage, we sort the rejection score for each method on the test set and vary the threshold used, for RealizableSurrogate the rejector is defined as $r(x) = \mathbb{I}_{g_{\perp}(x) - \max_y g_y(x) \geq c}$ where the optimal solution is at $c = 0$ and we vary $c \in \mathbb{R}$ to obtain the curve.

Table 3.1: Datasets used for our benchmark for learning with deferral to humans. We note the total number of samples n , the target set size $|\mathcal{Y}|$, the number of tasks in each dataset (a task is a set of human and target labels), the human expert where 'random annotator' means that for each point we have multiple human annotations and we let the target be a consensus and the human label be a random sample while 'separate human annotation' means that the human label is completely separate from the label annotations and finally the model class for both the classifier and rejector.

Dataset	n	$ \mathcal{Y} $	Number of Tasks	Human	Model Class
SyntheticData (ours)	arbitrary	2	1	synthetic	linear
CIFAR-K	60k	10	10 (per expert k)	synthetic (perfect on k classes)	CNN
CIFAR-10H [75]	10k	10	1	separate human annotation	pretrained WideResNet [76]
Imagenet-16H [43]	1.2k	16	4 (per noise version)	separate human annotation	pretrained DenseNet121 [77], finetuning last layer only
HateSpeech [54]	25k	3	1	random annotator	FNN on embeddings from SBERT [78]
COMPASS [79]	1k	2	1	separate human annotation	linear
NIH Chest X-ray [80], [81]	4k	2	4 (for different conditions)	random annotator	pretrained DenseNet121 on non-human labeled data



(a) Synthetic Data Sample Complexity

(b) CIFAR-K Semi-Synthetic

Figure 3.4: (a) Test performance of the different methods on synthetic data as we increase the training data size and repeat the randomization over 10 trials to get standard errors. (b) Test performance on the semi-synthetic CIFAR-K dataset vs. the number of classes K for which the expert is perfect.

3.7.1 Human-AI Deferral Benchmark

Objective. We investigate the empirical performance of our proposed approaches compared to prior baselines on a range of datasets. Specifically, we want to compare the accuracy of the human-AI team at the learned classifier-rejector pairs. We also check the accuracy of the system when we change the deferral policy by varying the threshold used for the rejector, this leads to an accuracy-coverage plot where *coverage* is defined as the fraction of the test points where the classifier predicts.

Datasets. In Table D.1 we list the datasets used in our benchmark. We start with synthetic data

described below, then semi-synthetic data with CIFAR-K [26]. We then evaluate on 5 real world datasets with three image classification domains with multiple tasks per domain, a natural language domain and a tabular domain. Each dataset is randomly split 70-10-20 for training-validation-testing respectively.

Baselines. We compare to multiple methods from the literature including: the confidence method from [25] (CompareConfidence), the surrogate L_{CE}^α from [26] (CrossEntropySurrogate), the surrogate Ψ_{OVA} from [62] (OvASurrogate), Diff-Triage from [27] (DifferentiableTriage), mixture of experts from [28] (MixOfExps) and finally a selective prediction baseline that thresholds classifier confidence for the rejector (SelectivePrediction). For all baselines and datasets, we train using Adam and use the same learning rate and the same number of training epochs to ensure an equal footing across baselines, each run is repeated for 5 trials with different dataset splits. We track the best model in terms of system accuracy on a validation set for each training epoch and return the best-performing model. For RealizableSurrogate, we perform a hyperparameter search on the validation set over $\alpha \in [0, 1]$, and do hyperparameter tuning over L_{CE}^α .

3.7.2 Synthetic and Semi-Synthetic Data

Synthetic Data. We create a set of synthetic data distributions that are realizable by linear functions (or nearly so) to benchmark our approach. For the input X , we set the dimension d , and experiment with two data distributions. (1) Uniform distribution: we draw points $X \sim \text{Unif}(0, U)^d$ where $U \in \mathbb{R}^+$; (2) Mixture-of-Gaussians: we fix some $K \in \mathbb{N}$ and generate data from K equally weighted Gaussians, each with random uniform means and variances. To obtain labels Y that satisfy Assumption 1, we generate two random halfspaces and denote one as the optimal classifier $m^*(x)$ and the other as the optimal rejector $r^*(x)$. We then set the labels Y on the side where $r^*(x) = 0$ to be consistent with $m^*(x)$ with probability $1 - p_m$ and otherwise uniform. When $r^*(x) = 1$, we sample the labels uniformly. Finally, we choose the human expert to have error p_{h0} when $r^*(x) = 0$ and have error p_{h1} when $r^*(x) = 1$. When $p_m = 0, p_{h0} \in [0, 1]$, and $p_{h1} = 0$, this process generates datasets $D = \{x_i, y_i, h_i\}_{i=1}^n$ that satisfy Assumption 1.

Sample Complexity. For realizable data with a feature distribution that is mixture of Gaussians ($d = 30, p_m = 0, p_{h0} = 0.3, p_{h1} = 0$), Figure 3.4a plots the test accuracy of the different methods on a held-out dataset of 5k points as we increase the training data size. We observe that MILP and RealizableSurrogate are able to get close to zero error, while all other methods fail at finding a near zero-error solution. We also experiment with non-realizable data. For example, when $p_m = 0.1, p_{h0} = 0.4, p_{h1} = 0.1$ with $n = 1000$, the optimal test error is $7.5 \pm 1.0\%$ for the generated data: the MILP obtains 11.2 error and RealizableSurrogate achieves 17.8 ± 1.0 error, while the best baseline CrossEntropySurrogate achieves 21.4 ± 1.1 error. In the Appendix, we show

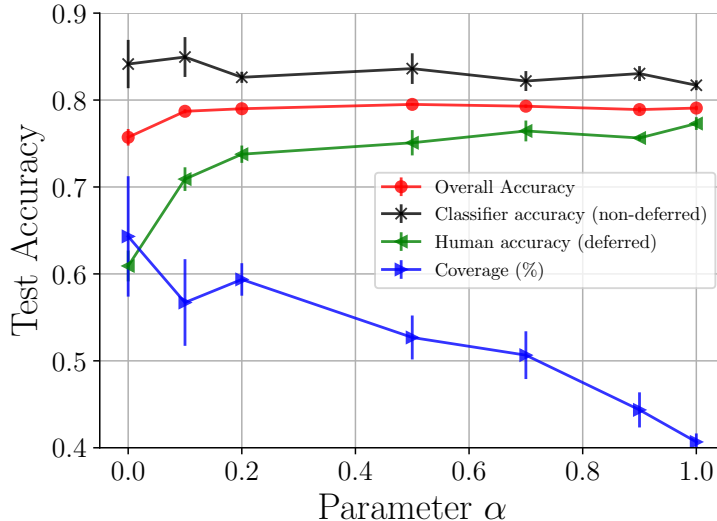


Figure 3.5: Sensitivity of the RealizableSurrogate to the hyperparameter α . We vary the hyperparameter α in the RealizableSurrogate surrogate loss and show the different metrics including overall accuracy, accuracy when we defer, accuracy when we don’t defer, and finally coverage.

results on the uniform data distribution, which shows an identical pattern, and we study the run-time and performance of the MILP as we increase the error probabilities.

CIFAR-K. We use the CIFAR-10 image classification dataset [60] and employ a simple convolution neural network (CNN) with three layers. We consider the human expert models from [26], [62]: if the image is in the first K classes the expert is perfect, otherwise the expert predicts randomly. Figure 3.4b shows the test accuracy of the different methods as we vary the expert strength K . RealizableSurrogate outperforms the second-best method by 0.8% on average and up to 2.8% maximum showcasing that the method can perform well for non-linear predictors.

3.7.3 Realistic Data

Models. In Figure 3.3, we showcase the test accuracy of the different baselines on the real datasets in Table D.1, and illustrate their behavior when we constrain our method and the baselines to achieve different levels of coverage. The test accuracy of the operating point on the different datasets is shown in Table 3.2. We can see that L_{RS}^α is competitive with the best baseline on each dataset/task. Moreover, we see that the human-AI team is often able to achieve performance that is higher than the human or classifier on their own. The methods often achieve peak performance at a coverage rate that is not at the extremes of [0,1], and on each of the six datasets we notice variability between the peak accuracy coverage rate indicating that they are finding different solutions. This demonstrates that deferral using L_{RS}^α is able to achieve complementary human-AI team performance

in practice. In summary, the new surrogate L_{RS} performs as well as the MILP on synthetic data, and as well as all the baselines (or better) on real-world data. Note that Differentiable Triage on these datasets is underperforming as we are testing it on a setting beyond the chapter as here we only have samples of expert predictions instead of probabilities from the expert.

Table 3.2: Test accuracy of the operating point of the different methods on the datasets tested on. The baselines are L_{CE} [26], Ψ_{OvA} [62], Selective Prediction (SP), Compare Confidence (CP) [25], DIFT [27] and MoE [28].

Dataset	L_{RS}^α (ours)	L_{CE}	Ψ_{OvA}	SP	CC	DIFT	MoE
Synthetic Realizable	0.979	0.891	0.918	0.882	0.918	0.870	0.992
Synthetic Non-Realizable	0.879	0.828	0.839	0.797	0.836	0.770	0.774
Cifar-K (K=5)	0.795	0.785	0.786	0.747	0.621	0.749	0.550
Compass	0.670	0.668	0.682	0.678	0.677	0.662	0.663
Cifar-10H	0.969	0.960	0.963	0.966	0.968	0.949	0.953
Hate Speech	0.924	0.913	0.919	0.926	0.921	0.906	0.907
ImageNet16H (noise 80)	0.912	0.908	0.909	0.910	0.908	0.898	0.904
ImageNet16H (noise 95)	0.865	0.872	0.872	0.875	0.868	0.856	0.861
ImageNet16H (noise 110)	0.802	0.791	0.791	0.809	0.792	0.756	0.761
ImageNet16H (noise 125)	0.755	0.707	0.732	0.756	0.743	0.655	0.604
Pneumothorax	0.976	0.963	0.978	0.972	0.978	0.978	0.978
Airspace Opacity	0.913	0.908	0.906	0.899	0.905	0.894	0.894

Hyperparameter α . We show how the behavior of the classifier and rejector system changes when we modify the hyperparameter $\alpha \in [0, 1]$ in Figure 3.5. When α is small, the behavior of the surrogate is the same as selective prediction which is why we see the lowest accuracy of the human when we defer. As α increases to 1, we can see that the system better adapts to the human.

Recommendations: Which Method to Use? Given our experimental results, the question to ask is which method should be used for a given dataset and model class. The simple and natural baseline of CompareConfidence should be the first tool one applies to their setting, it often achieves good performance, outperforming the naive baseline SelectivePrediction. However, CompareConfidence does not allow the classifier to adapt to the humans strengths and weaknesses. The surrogates CrossEntropySurrogate and OvASurrogate when applied with expressive model classes such as deep networks can find complementary classifiers. The surrogates offer other advantages, notably, CrossEntropySurrogate has been shown to have better sample complexity over the CompareConfidence baseline and can incorporate arbitrary costs of deferral and prediction [26]. However, as our synthetic experiments have shown, there is a limit of the CrossEntropySurrogate and OvASurrogate

surrogates to how much they can complement the human and defer accordingly. This is where our proposed methods MILPDefer and RealizableSurrogate come in. We recommend using the MILP in settings with limited data where linear models are suitable as it can achieve optimal performance, however, one must carefully tune regularization parameters to not overfit. If the data is realizable, then the RealizableSurrogate is also optimal and is much easier to optimize, one can apply the surrogate without knowing beforehand if the data is realizable. RealizableSurrogate works well with linear and non-linear model classes, and performs the best under model resource constraints, we recommend using it broadly when optimizing accuracy.

Part II

AI-Assisted Decision Making: Onboarding

Chapter 4

Teaching Humans When To Defer to a Classifier via Exemplars

Acknowledgements of Co-authors. This chapter is based on the published work in [82]. I would like to thank my co-author, Arvind Satyanarayan, for his help.

4.1 Introduction

Automated agents powered by machine learning are augmenting the capabilities of human decision makers in settings such as healthcare [1], [2], content moderation [83] and more routine decisions such as asking AI-enabled virtual assistants for recommendations [84]. This mode of interaction whereby the automated agent serves only to provide a recommendation to the human decision maker, a setting typically named *AI assisted decision making*, is the focus of our study here. A key question is how the human expert knows when to rely on the AI for advice. In this work, we make a case for the need to initially onboard the human decision-maker on when and when not to rely on the automated agent. We propose that before an AI agent is deployed to assist a human decision maker, the human is taught through a tailored onboarding phase how to make decisions with the help of the AI. The purpose of the onboarding is to help the human understand when to trust the AI and how the AI can complement their abilities. This allows the human to have an accurate mental model of the AI agent, which helps set expectations about the AI's performance on different examples.

Our onboarding phase consists of letting the human predict on a series of specially selected teaching examples in a setting that mimics the deployment use case. The examples are chosen to give an overview of the AI's strengths and weaknesses, especially when it complements the abilities of the human. After predicting on each example, the human agent then receives feedback on their performance and that of the AI. To allow the human to generalize from each example, we display

features of the region surrounding the example. Finally, to enable retention of the example, we let the human write down a lesson indicating whether they should trust the AI in that region and what characterizes the region. Our approach is inspired by research in the education literature that highlights the importance of feedback and lesson retention for learning [21], [22].

To select the teaching examples, we need to have a mathematical framework of how the human mental model evolves after we give them feedback. We model the human thought process as first deciding whether to rely on the AI’s prediction or not using an internal *rejector*. This rejector is what we refer to as the human mental model of the AI. We propose to model the human’s rejector as consisting of a prior rejector and a nearest neighbor rule that only applies in local regions surrounding each teaching example in section 4.4. This novel parameterization is inspired by work in cognitive science that suggests that humans make decisions by weighing similar past experiences [85]. Assuming this rejector model, we give a near-optimal greedy strategy for selecting a set of representative teaching examples that allows us to control the examples and the region surrounding them.

We first evaluate the efficacy of our algorithmic approach on a set of synthetic experiments and its robustness to the misspecification of the human model. For our main evaluation, we conducted experiments on Amazon Mechanical Turk on the task of passage-based question answering from HotpotQA [86]. Crowdworkers first performed a teaching phase and were then tested on a randomly chosen subset of examples. Our results demonstrate the importance of teaching: around half of the participants who undertook the teaching phase were able to correctly determine the AI’s region of error and had a resulting improved performance.

4.2 Related Work

4.2.1 Relation to Learning to Defer

Revised Section: Our framework of Human-AI assisted decision making, dubbed teaching to defer (TTD), and its associated framing can be considered as the analog of the learning to defer framework described in (LTD) [25], [26], [28], [29] which we discussed in Part I and Chapters 2 and 3. The main goal of LTD is to learn a rejector that determines which of the AI and the human should predict on each example. However, there are numerous legal and accountability constraints that may prohibit a machine from making final decisions in high-stakes scenarios. Additionally, the actual test-time setting may differ from that which was used during training, but since in our setting, the human makes the final decision, this allows them to adapt their decision-making and detect any unexpected model errors. As an example, in a clinical use case, factors such as times of substantially increased patient load may affect the human expert’s accuracy. The human may

also occasionally have side information that was unavailable to the AI that could improve their decision-making. Compared to LTD, deployment may be simplified because the same AI is used for all experts; as new experts arrive, our onboarding phase trains them to use the AI according to their unique abilities. Our teaching setting and LTD also use very different techniques. Although the objective that we present in Equation (4.3) is closely related to the objective used by [26], the main task in our setting is that of teaching the human when to defer. This requires us to develop a formalization of the human mental model and algorithms for selecting a subset of examples that enables accurate learning.

In our setting, the human observes the AI prediction and then makes a prediction. In LTD, the AI model first decides using a rejector whether to predict on its own or defer to the human. There is no interaction in LTD between the human and the AI as the goal is to reduce the burden on the human expert. We borrow the notion of a rejector to formalize the thought process of the human deciding whether or not to use the AI prediction. Table 4.1 highlights some of the main differences between the two frameworks.

System Objective. The objective in our framework is stated in equation (4.3), which can be compared to the system objective from LTD [26]:

$$L(h, r) = \mathbb{E}_{(x,y) \sim \mathbf{P}, m \sim M|(x,y)} [l(x, y, h(x)) \mathbb{I}_{r(X)=0} + l_{\text{exp}}(x, y, m) \mathbb{I}_{r(x)=1}] \quad (4.1)$$

Beyond the fact that in TTD, the human controls the rejector and in LTD the AI controls the rejector, a technical difference is the input to the rejector function r : in LTD it's the AI domain X , while in TTD it's the human domain Z and the AI prediction $\pi(X)$.

Human-AI Interaction. In LTD, when the AI predicts or defers, it does so without observing the human's prediction, and when the human predicts, they do so without seeing the AI prediction. On the other hand, in our framework, the human observes the AI's prediction and explanation before making their final prediction. This allows the human and AI to combine their predictions in a way that the LTD framework does not allow.

4.2.2 Further Related Work

One of the goals of explainable machine learning is to enable humans to better evaluate the correctness of the AI's prediction by providing supporting evidence [88]–[97]. However, these explanations do not inform the decision maker how to weigh their own predictions against those of the AI or how to combine the AI's evidence to make their final decision [98]. The AI explanations cannot factor in the effect of the human's side information, and thus the human has to learn what

Table 4.1: Comparison on different dimensions between the teaching to defer framework in this paper (TTD) and the learning to defer framework (LTD) from [26], [28], [87].

Dimension	LTD [26]	TTD (this paper)
Information at training	Samples from AI domain X , label Y , human prediction M	Samples from AI domain X , Human domain Z , label Y , and error distribution of AI and Human
Information at testing	AI domain X	AI domain X , Human domain Z , AI prediction π
AI training	joint training with rejector	trained without knowledge of human rejector
Knowledge about human Form of rejector	samples of prediction no constraint	error distribution radius nearest neighbor defined by Assumption 4
Interaction between Human and AI	No by design, AI doesn't see the Human prediction and Human doesn't see the AI prediction	Yes
Final decision maker	AI or Human	Human
Does Human observe each example	No, since AI might not defer	Yes
Ease of Deployment	Needs re-training for every human expert	Same deployment for any human expert

their side information reveals about the performance of the AI or themselves. Moreover, if the AI's explanations are unfaithful or become so due to a distribution shift in the data [99], then the human may then over-weigh the AI's abilities. Another direct approach for teaching is presenting the human with a set of guidelines of when to rely on the AI [100]. However, these guidelines need to be developed by a set of domain experts and no standard approach currently exists for creating such guidelines. As a byproduct of our teaching approach, each human writes a set of unorganized rules that can then be more easily turned into such guidelines.

Related work has explored how to best onboard a human to trust or replicate a model's prediction. LIME, a black-box feature importance method, was used to select examples so that crowdworkers could evaluate which of two models would perform better [38], [39]. Their selection strategy does not take into account the human predictor, nor does their approach do more than display the examples. On a task of visual question answering, [101] handpicked 7 examples to teach crowdworkers about the AI abilities and found that teaching improved the ability to detect the AI's failure. [102] on a Quizbowl question answering task highlights the importance of modeling the skill level of the human expert when designing the explanations; this further motivates our

incorporation of the human predictor into the choice of the teaching set. Through a study of 21 pathologists, [103] gathered a set of guidelines of what clinicians wanted to know about an AI prior to interacting with it. [104] study the effect of initial debriefing of stated AI accuracy compared to observed AI accuracy in deployment and find a significant effect of stated accuracy on trust, but that diminishes quickly after observing the model in practice; this reinforces our approach of building trust through examples that simulate deployment. [16] investigate the role of the human’s mental model of the AI on task accuracy, however, the mental model is formed through test time interaction rather than through an onboarding stage. [105] propose a theoretical model for AI-assisted decision making, assuming that the human has a perfect mental model of the AI and that the human has uniform error.

Finally, our work was inspired by the literature on machine teaching [42], [106]–[109] and curriculum learning [110], [111]. Our work differentiates itself from the machine teaching literature by the use of our novel radius neighbor human model and the goal of teaching how to defer to an AI rather than teaching concepts to humans. Studies have also explored the use of reinforcement learning as a tool for online education [112]–[114]. We further expand the related work in Appendix C.1.

4.3 Problem Setup

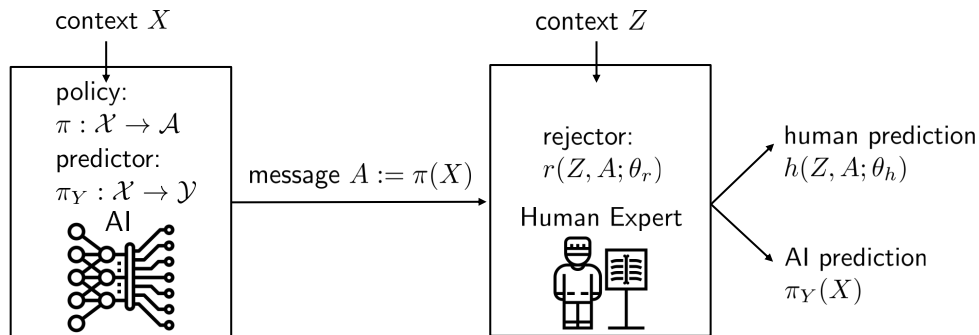


Figure 4.1: The AI assisted decision making pipeline. The AI first sends to the human a message A , then the human decides with their rejector $r(Z, A)$ if they should follow the AI’s advice and predict $\pi_Y(X)$ or they should predict on their own using $h(Z, A)$.

Our formalization is based on the interaction between two agents: the AI, an automated agent, and a human expert who both collaborate to predict a target $Y \in \mathcal{Y}$ based on a given input context. The setup is as follows: the AI perceives a view of the input $X \in \mathcal{X}$, then communicates a message $A \in \mathcal{A}$ that is perceived by the human. The human expert then integrates the AI message A and their own view of the input $Z \in \mathcal{Z}$ to make a final decision $M(Z, A)$ which can either be to predict

on their own or allow the AI agent to predict. The input space of the human Z and that of the AI X could be different since the human may have side information that the AI can't observe. This is essentially the *AI-Assisted Decision Making* setup illustrated in Figure 4.1 which is the more common mode of interaction between humans and artificially intelligent agents in high-stakes scenarios.

More formally, the AI consists of a predictor $\pi_Y : \mathcal{X} \rightarrow \mathcal{Y}$ that can solve the task on its own and a policy $\pi : \mathcal{X} \rightarrow \mathcal{A}$ which serves to communicate with the human. The message space \mathcal{A} may consist for example of the AI's prediction $\pi_Y(X)$ alongside an explanation of their decision. On the other hand, the human when seeing the AI's message consists of a **predictor** $h : \mathcal{Z} \times \mathcal{A} \rightarrow \mathcal{Y}$ parameterized by θ_h and the human decides to allow the AI to predict or not according to a **rejector** $r : \mathcal{Z} \times \mathcal{A} \rightarrow \{0, 1\}$ parameterized by θ_r , where if $r(Z, A; \theta_r) = 1$ the human uses the AI's answer for its final prediction. This implies that the final human decision M is as follows:

$$M(Z, A) = \begin{cases} \pi_Y(x) & , \text{ if } r(Z, A; \theta_r) = 1 \\ h(Z, A; \theta_h) & , \text{ otherwise} \end{cases} \quad (4.2)$$

System objective. Given the above ingredients and a performance measure on the label space $l(y, \hat{y}) : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ (e.g. 0-1 loss), the loss that we incur is the following:

$$L(\pi, \pi_Y, h, r) = \mathbb{E}_{x,z,y} \left[\underbrace{l(\pi_Y(x), y)}_{\text{AI cost}} \overbrace{\mathbb{I}_{r(x, \pi(x))=1}}^{\text{AI predicts}} + \underbrace{l(h(z, \pi(x)), y)}_{\text{Human cost}} \overbrace{\mathbb{I}_{r(x, \pi(x))=0}}^{\text{Human predicts}} \right] \quad (4.3)$$

We put ourselves in the role of a system designer who has knowledge of both the human and the AI and wishes to minimize the loss of the system L (4.3).

The central Human-AI interaction problem. Given a fixed AI policy, and human parameters (θ_h, θ_r) , the manner in which the human expert integrates the AI's message depends only on the expert context Z and the message itself A . In particular, for two different policies π_1 and π_2 that output the same message A on input Z , our framework tells us that the resulting behavior of the human expert would be identical in both cases. However, if it is known to the human that AI π_1 has very high error compared to AI π_2 , then is more likely for them to trust the message if it is coming from π_2 rather than from π_1 . Thus it is more realistic to assume that the expert has a *mental model* of the policy π that they have arrived at from either a description of the policy or from previously interacting with it; the rejector here formalizes the *mental model*. This insight forces us to now consider the parameters (θ_h, θ_r) as variables that are learned by the human as a function of the underlying AI policy π . This makes the optimization of the loss now much more challenging as

whenever the policy π changes, the human’s mental model, (θ_h, θ_r) , needs to update. Therefore, we need to first understand how the human’s mental model evolves and how we can influence it.

Teaching Humans about the AI. In this work, we focus on exemplar based strategies to allow the human to update their mental models of the AI. The question is then how do we select a minimal set of examples that teaches the human an accurate mental model of the AI. To make progress, we need to first understand the form of the human’s rejector and how it evolves, which we elaborate on in the following section. Crucially, we will keep the AI in this work as a fixed policy and not look to optimize for it. Once we understand this first step, future work can then look to close the loop which entails learning an updated AI with the knowledge of the human learner dynamics.

4.4 Human Mental Model

We now introduce our model of the human’s rejector and the elements of the teaching setup. The tasks we are interested in are where humans are *domain experts*, where we define domain experts to mean that their knowledge about the task and their predictive performance are fixed. We further extend this to how they may incorporate the AI message in their prediction, but crucially not how they decide when to use the AI. This assumption translates in our formulation as follows.

Assumption 3. *The human predictor does not vary as they interact with the AI, i.e. we assume θ_h to be fixed.*

While we have assumed θ_h is fixed and have so far spoken about a singular human, in reality, the AI might be deployed in conjunction with multiple human experts. These experts might have different parameters θ_h individually, however; for the rest of this chapter, we focus on a singular expert that we are interacting with.

We now move our attention to the human’s rejector, which represents their mental model of the AI, and learned after observing a series of labeled examples. Research on human learning from the cognitive science literature has postulated that for complex tasks humans make decisions by sampling similar experiences from memory [85], [115], [116]. Moreover, [85] makes the explicit comparison with nearest neighbor models found in machine learning. However, standard nearest neighbor models don’t allow for prior knowledge to be incorporated. For this reason, we postulate a nearest neighbor model for the human rejector that starts with a prior and updates in local regions of each shown example in the following assumption.

Assumption 4 (Form of Human’s rejector). *The human’s rejector consists of a prior rejector rule and a nearest neighbor rule learned after observing teaching examples $D_T = \{z_i, a_i, r_i\}_{i=1}^m$.*

Formally, let $g_0(Z, A) : \mathcal{Z} \times \mathcal{A} \rightarrow \{0, 1\}$ be the human's prior rejector. Figure 4.2 illustrates the scenario: the prior is the region at the boundary of the human predictor h . Let $K(., .) : \mathcal{Z} \times \mathcal{Z} \rightarrow \mathbb{R}^+$ be the similarity measure that the human employs to measure the degree of similarity between two instances.

The human's rejector uses a learned rule if they had observed an example similar with respect to $K(., .)$ during teaching, otherwise falling back on their prior:

$$r(Z, A; \theta_r) = \begin{cases} \text{vote}(B(Z)) & , \text{ if } B(Z) \neq \emptyset \\ g_0(Z, A) & , \text{ otherwise} \end{cases} \quad (4.4)$$

where $B(Z)$ is the set of all points in D_T that they observed in training sufficiently similar to Z :

$$B(Z) = \{i \in [m] \mid K(Z, z_i) > \gamma_i\} \quad (4.5)$$

The degree of similarity is measured by a scalar γ_i that the human sets for each teaching example, in figure 4.2 all the points in the shaded ball have $B(Z) = \{z_1\}$. The rule $\text{vote}(B(Z))$ defines the label for all points similar to Z based on a weighted decision:

$$\text{vote}(B(Z)) = \arg \max_{k \in \{0,1\}} \frac{\sum_{i \in B(Z)} \mathbb{I}\{r_i = k\} K(Z, z_i)}{\sum_{i \in B(Z)} K(Z, z_i)} \quad (4.6)$$

Where r_i is the deferral rule that the human has learned on example z_i .

We can possibly further assume that the prior takes a rather simple form of thresholding the predictor's error: $g_0(Z, A) = \mathbb{I}\{\mathbb{P}(h(Z, A) \neq Y \mid Z, A) \geq \epsilon\}$ for some $\epsilon > 0$. One possibility for ϵ is the error rate of the AI.

Discussion on the Assumptions. In our assumptions above, we assumed knowledge of the following parameters: the human predictor $h(Z, A)$, the prior human rejector $g_0(Z, A)$ and the human similarity measure $K(., .)$. In fact, as we will see, we only need to know the expert error distribution $\mathbb{E}[l(h(Z, A), Y) \mid Z, A]$ rather than the full expert predictor; it may be reasonable to estimate the expert's error distribution from previously collected data. The prior rejector g_0 can also be learned by testing the human prior as evidenced by prior work on capturing human priors [117], [118], otherwise, a reasonable guess is the human deferring by just thresholding their own error rate. Finally to teach the human, we need a proxy for the similarity measure $K(., .)$. This can be obtained in many ways: one can learn this metric with separate interactions with the human, see [119], [120], or rely on an AI based similarity measure e.g. from neural network embeddings [78]. This last proxy is readily available and in the framework of our study, we believe it is reasonable to use.

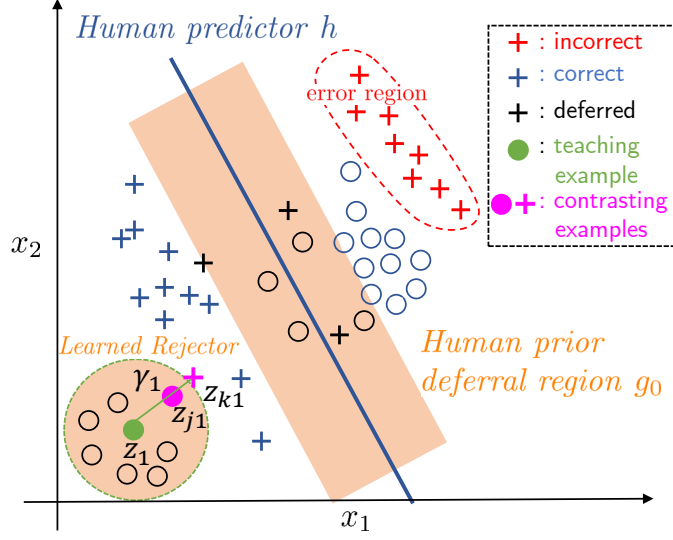


Figure 4.2: Illustration of human rejector on toy example. The task is classification with labels $\{o, +\}$, the human prediction h is the blue line and the prior g_0 is the shaded orange region surrounding the boundary. Points in red is where the human is incorrect, in blue correct and in black point deferred to the AI. The AI is assumed to be correct on examples far from the human boundary. The human receives a teaching example z_1 (in green) with radius γ_1 . Also shown are the two contrasting examples z_{j1} and z_{k1} (in pink) that define the region.

An important part of the rejector is the associated radius γ_i with each teaching example i , the radius allows the human to generalize from each teaching example to the entire domain. The human learning process leaves the setting of γ_i completely up to the human and is not observed. However, we hope to directly influence the value of γ_i that the human sets during teaching.

4.5 Teaching a Student Learner

Formulation. The previous section introduced the model of the human learner, in this section we will set out our approach to select the teaching examples for the onboarding stage. Essentially, our approach is trying to find local regions, balls with respect to $K(\cdot, \cdot)$, that best teach the human about the AI. We assume access to a labeled dataset $S = \{x_i, z_i, y_i\}_{i=1}^n$ that is independent from the training data of the AI model. For each point we can assign a deferral decision r_i that the human should undertake that minimizes the system loss. Explicitly, the optimal deferral decision r_i is defined to select who between the human and AI has lower loss on example i :

$$r_i = \mathbb{I}\{\mathbb{E}[l(h(z_i, a_i), y_i)] \geq \mathbb{E}[l(\pi_Y(x_i), y_i)]\} \quad (4.7)$$

Note that to derive r_i we only need to know the loss of the human on the teaching set and not their predictions. Define then $S^* = \{x_i, z_i, r_i\}_{i=1}^n$ as a set of examples alongside deferral decisions. As mentioned previously, the human is also learning a radius γ_i with each example. The radius γ_i should be set large enough to enable generalization to the domain, but small enough for the region to be coherent so that the human can interpret why should they follow the optimal deferral decision.

Let $D_z \subset S^*$ and let D_γ be the set of radiuses associated with each point in D_z and define $D = (D_z, D_\gamma)$. Define the loss of the human learner $M(\cdot, \cdot; D)$ now only parameterized by the teaching set D as follows:

$$L(D) = \sum_{i \in S} l(M(z_i, a_i; D), y_i) \quad (4.8)$$

Greedy Selection. Note that since the radiuses set by the human are learned only after observing the example, we try to jointly optimize for the teaching point and the radius to teach. To optimize for D , consider the following greedy algorithm (GREEDY-SELECT) which starts with an empty set D_0 , and then repeats the following step for $t = 1, \dots, m$ to select the example z and radius γ that leads to the biggest reduction of loss if added to the teaching set:

$$z, \gamma = \arg \min_{z_i \in S \setminus D_t, \gamma} L(D_t \cup \{z_i, \gamma\}), \quad (4.9)$$

$$\text{s.t. } \exists k \in [n] \text{ s.t. } \gamma = K(z_i, z_k), \quad (4.10)$$

$$\text{and } \frac{\sum_{j \in [n], K(z_i, z_j) > \gamma} \mathbb{I}_{r_j = r_i}}{|\{j \in [n], K(z_i, z_j) > \gamma\}|} \geq \alpha \quad (4.11)$$

Constraint (D.2) restricts γ to be the similarity between z and another data point and constraint (D.3) ensures that $\alpha\%$ of all points inside the ball centered at z share the same deferral decision as z . The scalar α is a hyperparameter that controls the consistency of the local region: when $\alpha = 1$, the region is perfectly consistent and we call this setting CONSISTENT-RADIUS, and when $\alpha = 0$ the constraint is void and we dub the algorithm as DOUBLE-GREEDY.

Contrasting examples. Note that the radius γ is actually defined by two points: the point z_k in equation (D.2) that defines the boundary and an interior point z_j that is the least similar point to z with similarity at least γ ; these two points are illustrated in Figure 4.2 with the color pink. These two points must actually share opposing deferral actions with $r_k \neq r_j$ and thus are contrasting points later used as a way to describe the local region.

Theoretical Guarantees. Let D_t be the solution found by the greedy algorithm and D^* the optimal solution. We now try to see how we can compare D_t to D^* . To do so, we make a further assumption on the choice of radiuses that the human sets.

Assumption 5 (Radius consistency). *We assume that if $j \in B(z_i) \cap S$ then $r_i = r_j$. This implies that if z_j is at least γ_j close to z_i , then the best deferral choice for j is the same as that for i . This assumption is an assumption on the choice of γ_i 's for each example in the teaching set.*

Assumption 5 in essence says that the human is always conservative enough such that the lesson drawn from example i is consistent on S . This translates to setting $\alpha = 1$ in our algorithm; when $\alpha < 1$ the guarantees may not hold. Given this assumption we can deduce that our objective function is now submodular and monotone. Furthermore, equipped with the fact that our problem is submodular we can derive the following guarantee on the gap of performance of our algorithm versus the optimal teaching set, as the next theorem demonstrates.

Theorem 8. *Let $F(X) = L(\emptyset) - L(X)$, when $\alpha = 1$, $F(\cdot)$ is submodular, monotone and positive. Moreover, the GREEDY-SELECT algorithm described above achieves the following performance compared to the optimal set D^* :*

$$\underbrace{L(D_m)}_{\text{loss of chosen set}} \leq \left(1 - \frac{1}{e}\right) \underbrace{L(D^*)}_{\text{loss of optimal set}} + \frac{1}{e} \underbrace{L(\emptyset)}_{\text{loss of prior rejector}}$$

All proofs can be found in Appendix C.2.

Theorem 8 gives a guarantee on the subset chosen by the greedy algorithm with an $1 - \frac{1}{e}$ approximation factor, one can ask if we can do better. We prove that a generalization of our problem is in fact NP-hard in the appendix. In what was previously discussed, the dataset that we measure performance on and that we teach from are the same. We generalize to have a separate training set S_T and a validation set S_V and define the loss of the human with respect to S_V and now define our optimization problem in terms of finding a minimal size subset D that achieves a certain loss $\delta \geq 0$:

$$D_\delta^* = \arg \min_{D \subset S_T} |D| \quad \text{s.t.} \quad \sum_{i \in S_V} l(M(z_i, a_i; D), y_i) \leq \delta \quad (4.12)$$

Proposition 5. *Problem (4.12) is NP-hard.*

The reduction is to the set cover problem and can be found in Appendix C.2.

Human Teaching Approach. After running our greedy algorithm, we obtain a teaching set D that we now need to teach to the human. We rely on a four stage approach for teaching on each example so that they are able to learn and generalize to the neighborhood around it shown in Algorithm 2. The human first predicts on the example z , then they receive feedback on their prediction and the AI's prediction. We then show them a description of the region around the example that helps them learn the radius. Specifically, we show them the two contrasting examples z_j and z_k defined by γ_i

Algorithm 2: Our Human Teaching Approach

Input: Teaching set D

- 1: **for** $i = 1, \dots, m$ **do**
 - 2: **Stage 1: Testing.** Test the human on example z_i with AI message a_i
 - 3: **Stage 2: Feedback.** Show human feedback of actual label y_i , AI prediction π_i , and recommended deferral action r_i
 - 4: **Stage 3: Lesson Generalization.** Show the two contrasting examples z_j and z_k and high level features about the region to allow generalization around z_i .
 - 5: **Stage 4: Lesson Reinforcement.** We ask the human to write a rule R_i that describes the region surrounding the example z_i and which action they should take.
 - 6: **end for**
-

and high level features about the neighborhood. Finally, we ask them to formalize in writing a rule describing the region and the action to take inside that region. This rule that they write per example helps the human in creating a set of guidelines to remember for when to rely on the AI and ensures that they reflect on the teaching material.

4.6 Experimental User Study

We provide code to reproduce our experiments ¹. Additional experimental details and results are left to Appendix C.6.

4.6.1 Experimental Preliminaries

Experimental Task and Dataset. Our focus will be on *passage-based question answering* tasks. These are akin to numerous real world applications such as customer service, virtual assistants and information retrieval. It is of interest as relying on an AI can reduce the time one needs to answer questions by not reading the entire passage and as an experimental setup it allows a greater range in the type of *sub-expertise* we can allow for compared to experimental tasks in the literature. We rely on the HotpotQA dataset [86] collected by crowdsourcing based on Wikipedia articles. We slightly modify the HotpotQA examples for our experiment by removing at random a supporting sentence from the two paragraphs. The supporting sentence removed does not contain the answer, so that each question always has an answer in the passage, however, it may not always be possible to arrive at that answer. This was done to make the task harder and create incentives for expert humans to use the AI. We further remove yes/no questions from the dataset and only consider hard multi hop questions from the train set of 14631 examples and the dev set of 6947 examples.

¹<https://github.com/clinicalml/teaching-to-understand-ai>

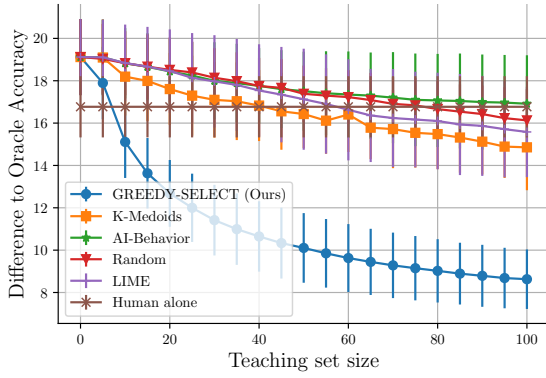
Simulated AI. One of the top performing models on HotpotQA is SAE-large: a graph neural network on top of RoBERTa embeddings [121]. We performed a detailed error analysis in Appendix C.3 of the SAE-large model predictions on the dev set. However, our analysis uncovered only few and small regions of model error. For our experimental study, we want to evaluate the effect of teaching in two ways: 1) through systematically checking the validity of the user lessons and 2) through objective task metrics. The SAE model makes it harder for us to do both especially with a limited number of responses from crowdworkers. For this reason, we decided to create a simulated AI whose error regions are more interpretable. We first cluster the dataset using K-means with k_p clusters based on only the paragraph embeddings obtained from a pre-trained SentenceBERT model [78]. The simulated AI model is parameterized by a vector $err_p \in [0, 1]^{k_p}$ where the probability of error of the AI on cluster i by $err_p[i]$. The answer of the AI when it is incorrect is manually constructed to be reasonably incorrect: for example if the answer asks for a date, we provide an incorrect date rather than a random sentence. To summarize, the AI for each cluster in the data has a specified probability of error that is constant on the cluster. To show that each cluster computed has a distinct meaningful theme, we retrieve the top 10 most common Wikipedia categories in each cluster. The full categories are shown in Appendix C.6; example cluster categories include singers/musicians, movies and soccer (but not football).

Metrics. Our aim will be to measure objective task performance and effort through the proxy of time spent on average per example. Our task performance metric is the F1 score on the token level [122]; we will measure this when considering the final predictions (Overall F1), on only when the human defers (Defer F1) and when the human does not defer (Non-Defer F1). We will also measure *AI-reliance*: this is calculated as how often they rely on the "Let AI answer for you" button in Figure 4.4a.

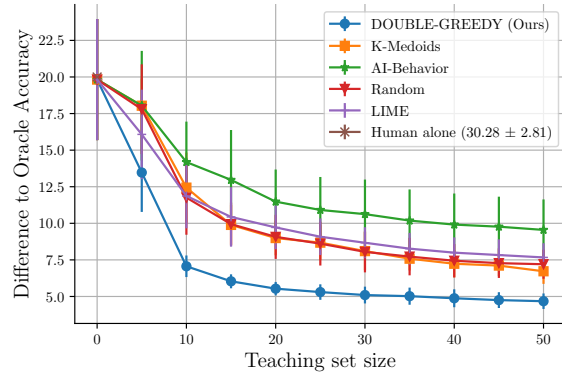
4.6.2 Simulated Users

Before we experiment with real human users, we evaluate the teaching complexity, i.e. the relation between teaching set size and human accuracy, of our teaching algorithm on simulated human learners that follow our assumptions. We further evaluate the robustness of our approach when we do not have full knowledge of the human parameters.

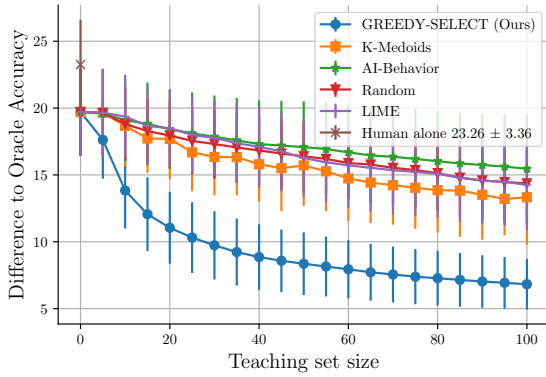
AI and Human model. We use the simulated AI model with $k_p = 15$ and a vector of errors err_p where for each i , $err_p[i]$ is drawn *i.i.d.* from $\text{Beta}(\alpha_{ai}, \beta_{ai})$. The human predictor is analogous to the AI model with a different vector of probabilities err'_p sampled from $\text{Beta}(\alpha_h, \beta_h)$. The human prior thresholds the probability error of the human to a constant ϵ . Finally, the human similarity



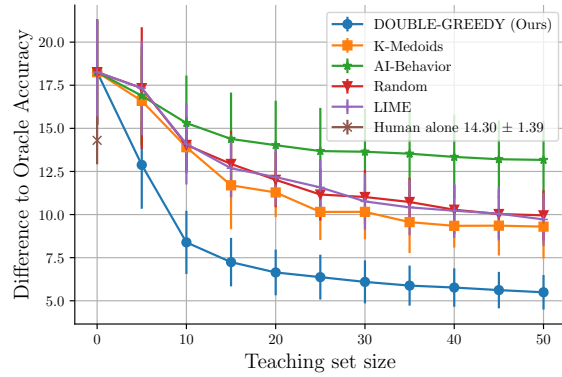
(a) Setting B and CONSISTENT-RADIUS



(b) Setting A and GREEDY-RADIUS



(c) Setting A and CONSISTENT-RADIUS



(d) Setting B and GREEDY-RADIUS

Figure 4.3: Teaching set size versus the negative difference between the human’s learner test accuracy under the different methods compared to ORACLE. We plot the average result across 10 trials and standard deviation as error bars.

measure is the RBF kernel on the passage embeddings i.e. $K(x, x') = e^{-|x-x'|^2}$. In this setup both the human and AI contexts are identical and the AI does not send any messages to the human.

Baselines. We implement a domain cover subset selection baseline in K-Medoids, the LIME selection strategy by [38] with 10 features per example following [39] (LIME), random selection baseline (RANDOM) and a baseline that greedily selects the point that helps a 1-nearest neighbors learner best predict the AI errors (AI-BEHAVIOR). Finally, we also compare to the optimal rejection rule computed with knowledge of the human and AI error rates by picking the lower one (ORACLE). The ORACLE rejector is an upper bound on achievable performance by any possible rejector regardless of the human student model.

Experimental setup. We will compare to the baselines as we vary the size of the teaching set D_T . To illustrate the effectiveness of the teaching methods, we focus on two settings: A) the Human is less accurate than the AI but their prior rejector rarely defers where we set the following and B) the

Condition	Oracle Gap @n=30
Full Information	6.38 ± 1.56
Missing g_0	6.90 ± 1.80
Noisy Radius	9.74 ± 3.0
Missing h	13.47 ± 5.07
No Information+Noise	15.12 ± 4.00
Prior only	16.72 ± 1.22
Human Alone	19.8 ± 2.80

Table 4.2: Test Accuracy gap between DOUBLE-GREEDY and ORACLE at teaching set of size 30 under various conditions. This is performed under setting B.

Human is more accurate than the AI but their prior rejector over defers to the AI. These two settings is where teaching is most beneficial as the prior is erroneous. Specifically in setting A) we set the following: $(\alpha_{ai} = 2, \beta_{ai} = 1)$ (the pdf is a straight line from the origin to $(1, 2)$), $(\alpha_h = 1, \beta_h = 1)$ (uniform distribution) and $\epsilon = 0.1$ and B) we set $(\alpha_{ai} = 1, \beta_{ai} = 1)$, $(\alpha_h = 2, \beta_h = 1)$ and $\epsilon = 0.9$. We evaluate for each setting 10 different random settings of the human and AI error probability vectors and average the results.

Results. Figure 4.3 shows the gap between Oracle and human accuracy on the dev set compared to the size of the teaching set for each of the methods. We can see that our approach is able to outperform the baselines under setting B with CONSISTENT-RADIUS. We observe a wide gap between our method and the baselines, this is because the teaching examples here must focus on only a select number of the clusters and cover them sufficiently. With the greedy radius selection, we require fewer examples to reach high accuracy and the gap between our method and the baselines narrows.

Robustness to Misspecification of Human model. We evaluate accuracy when the human is not learning the correct radius; this simulates noise in the learning process. The radius γ_i that the human learns is a noisy version of $\hat{\gamma}_i$ where we add a uniformly distributed noise $\delta \sim \mathcal{U}(-(1 - \hat{\gamma}_i)/2, (1 - \hat{\gamma}_i)/2)$ to it. We then evaluate when we have no knowledge of the prior rejector g_0 or/and no knowledge of the human predictor h . When we don't know either of these parameters, we replace them by a random binary vector Bernoulli $(1/2)^n$ on the teaching set. Results are shown in Table 4.2. We can see that even if we don't have knowledge about the prior, accuracy is not impacted. However, if we don't have knowledge about the predictor h , then performance drops significantly. To evaluate how much information about h we need to properly teach the human, we learn a teaching set assuming the human's error probability is $err'_p + \delta$ where δ has each component

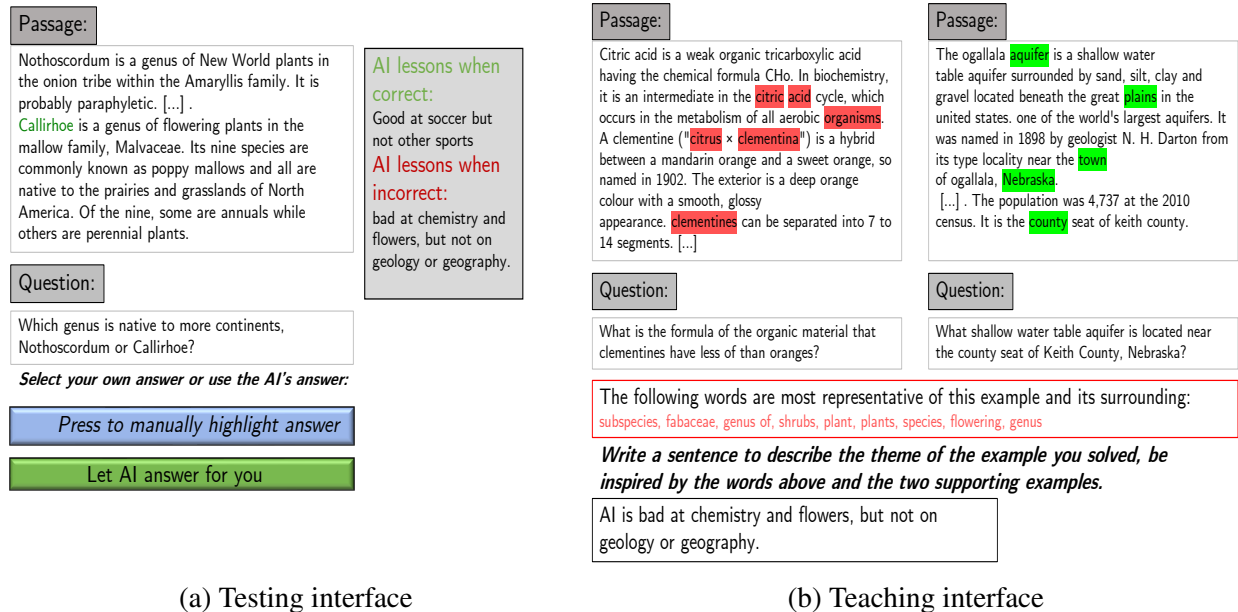


Figure 4.4: On the left in subfigure (a) is the testing interface shown for an example. This is the same interface that is also shown at the beginning of each teaching example. After the human predicts and we are in the teaching phase, we show them the correct answer and transition to the interface in subfigure (b) that shows the two supporting examples for the example in (a), the top weighted words in the region and asks the user to write down their rule for the example.

drawn from $\{-\delta, \delta\}$ uniformly where $\delta > 0$. On setting B with DOUBLE-GREEDY, we can tolerate up to 0.25 error in knowledge about cluster error probability with no noticeable drop in performance; full results are in Appendix C.4. Note that when we don't have any knowledge about the human and the learning process is noisy, teaching is impacted.

4.6.3 Crowdsourced Experiments Details

Testing user interface. Our user interface during testing is shown in Figure 4.4a which shows a paragraph and its associated question. The human can either submit their own answer or let the AI answer for them using a special button. However, the interface does not display the AI's answer or any explanation, which forces the user to rely solely on their mental model and the teaching examples to make a prediction. This was done so that we can control for the effect of teaching solely, as showing the AI prediction at test time leaks information about the AI beyond what was shown in the teaching set. Moreover, not showing the AI prediction forces the human to explicitly think about the AI performance. The right panel next to the passage shows the lessons that the user wrote down during teaching.

Metric	Ours-Teaching (all)	No-Teaching	LIME (all)	Ours (acc)	Ours (inacc)	LIME (acc)	LIME (inacc)
Overall F1	58.2 ± 3.4	57.6 ± 3.4	52.9 ± 3.4	62.8 ± 4.7	53.5 ± 4.9	56.5 ± 6.4	52.0 ± 4.2
Defer F1	50.7 ± 4.7	57.8 ± 4.9	48.1 ± 5.3	53.4 ± 6.7	50.0 ± 6.8	44.6 ± 9.0	49.9 ± 6.5
Non-Defer F1	67.6 ± 4.7	57.6 ± 4.7	56.9 ± 4.6	73.92 ± 6.2	60.6 ± 7.1	70.0 ± 8.6	53.7 ± 5.4
Time/ex (min)	0.60 ± 0.03	0.62 ± 0.03	0.68 ± 0.04	0.54 ± 0.04	0.68 ± 0.05	0.65 ± 0.08	0.69 ± 0.05
AI-Reliance (%)	55.2 ± 3.6	48.9 ± 3.6	45.4 ± 3.6	53.3 ± 4.9	58.9 ± 5.0	52.8 ± 3.6	43.6 ± 4.3

Table 4.3: Comparison of the metrics between our teaching condition (split into all participants, those who gave accurate lessons (acc) and those who didn’t (inacc), see description below), the No-teaching+AI-prediction condition and LIME teaching. Shown are averages across all participants with 95% confidence interval error bars. The F1 of the AI alone in this setting is 46.7%; we did not separately measure the F1 of the human in isolation.

Teaching user interface. Following our teaching algorithm, during teaching, the worker is first faced with the same user interface as in test time. The difference is that *after* they answer, they receive feedback on the correctness of their answer and can see the AI’s answer. We then show the human the two contrasting examples with LIME word highlights. As a high level description of the local region, we show the top 10 most weighted words obtained by LIME in the ball surrounding the original teaching example [38] (see Figure 4.4b). After they observe the two supporting examples, they are asked to write a sentence that describes the lesson of the example. These sentences are available during test-time for the workers to review as help for answering new questions.

Experimental Design and Baselines. The experimental teaching setup proceeds in three stages. The first stage (Stage 0) is a tutorial that introduces the task with two examples and where we gather the worker’s demographic information, knowledge of machine learning and how often they visit Wikipedia. Stage 1 is the teaching stage where the worker solves 9 teaching examples and stage 2 is the testing phase where the worker solves 15 questions with no feedback. After the two stages is an exit survey where users are asked about their decision process for using the AI. The two stage experimental design mimics what we believe would be a realistic deployment in practice; we don’t expect feedback to be possible during deployment, but rather only in a specialized teaching phase. We randomly assign each participant to one of three conditions.

In the first condition the participants go through the entire pipeline described above (Ours Teaching). The second is condition is called (LIME-Teaching) where LIME is first used to obtain 18 examples. During teaching, users are asked to solve the first 9 questions and are then shown: LIME highlights of the example, performance feedback and asked to write a lesson of what they learned. Then users view the 9 remaining examples with LIME highlights without needing to solve them or write lessons. The difference with our method is that workers don’t see the supporting examples or the word level description of the regions. The third is a baseline condition (No-teaching+AI-prediction) that makes the following modifications to the experimental design:

Metric	Ours-Teaching (ID)	No-Teaching (ID)	Ours (OOD)	No-Teaching (OOD)
Overall F1	56.8 ± 3.6	56.0 ± 3.6	70.9 ± 10.5	72.86 ± 10.7
Defer F1	51.42 ± 4.9	57.8 ± 5.2	42.95 ± 17.2	56.7 ± 18.8
Non-Defer F1	63.7 ± 5.2	54.4 ± 5.1	96.05 ± 5.82	85.0 ± 11.5
AI-Reliance (%)	56.1 ± 3.8	49.4 ± 3.8	47.3 ± 11.6	42.9 ± 11.8

Table 4.4: Comparison of the metrics on clusters that were seen during teaching with our method (ID for in distribution) compared to performance on clusters that were not seen during teaching (OOD for out of distribution). We also show the performance of the no-teaching baselines on the two cluster sets as a reference point. The errors on the OOD estimates are much higher as there are much fewer samples in the not-seen clusters.

the participants skip the teaching stage (Stage 1) and immediately proceed to the testing phase (Stage 2). However, during the testing phase, the participants *can see the AI prediction* before they press the use AI button which gives them an edge compared to the teaching condition.

Participants We recruited 50 US based participants from Amazon Mechanical Turk per each condition (150 total) and initial pilot studies were also conducted with graduate students in computer science at a US university. Participants in the non-teaching baseline were paid \$3 for 10 minutes of work and those in the teaching condition received \$6 for 20 minutes of work. Any demographic information we gathered in our study is kept confidential and workers were asked to consent to their use of their responses in research studies.

AI and Test Set details. The simulated AI had $k_p = 11$ and was randomly chosen to have probability of error 0 or 1 on each cluster. This means there are clusters where the AI is perfect on and other clusters where the AI is always wrong. We split the HotpotQA dev set into two parts 80:20 for the teaching and testing set respectively. To obtain the 9 teaching examples we run GREEDY-SELECT with the consistent radius strategy with no knowledge of g_0 or h . The examples in the testing phase was obtained first by filtering the data using K-medoids with $K = 200$ as a way to get diverse questions. Then each participant received 7 random questions from the filtered set on which the AI was correct and 8 on which the AI is incorrect.

Further details can be found in Appendix C.6.

4.6.4 User Study Observations and Results

Teaching enables participants to better know when to predict on their own, but not when to defer to the AI. The first three columns of Table 4.3 display the metrics measured across both conditions on all participants. We can first note that participants with teaching are able to

predict overall just as well as participants in the baseline no-teaching condition who have additional information about the AI prediction at test time. Moreover, participants who received teaching can better recognize when they are able to predict better than the AI. There is a difference significant at p -value 0.05 ($t = 2.9$, from a two sample t-test) of the F1 score when the human doesn't defer between our method and the no-teaching baseline and significant at p -value 0.001 ($t = 3.2$) compared to LIME. However, the participants in the teaching condition deferred to the AI when it was incorrect more often than those in the no-teaching baseline condition. A positive difference significant at p -value 0.05 ($t = -2.0$) in F1 when the humans defers for No-teaching+AI-prediction workers. An explanation for this is that the participants might press the use AI button on examples where their own prediction agrees with that of the AI instead of manually selecting the answer which takes more effort.

Accurate teaching lessons might predict improved task performance and our method teaches more participants than LIME. Given our knowledge about the clusters and the AI, the correct form of the teaching lesson of each example is "AI is good/bad at TOPIC" where TOPIC designates the theme of each cluster amongst a set of 11 topics which include soccer, politics, music and more. Manually inspecting the lessons of the 50 participants without seeing their test performance, we found that 25 out of 50 participants in our teaching condition were able to properly extract the right lesson from each teaching example. The remaining 25 participants were split into two camps: those who gave explanations on question/answer type or too broad or narrow of explanations e.g. "AI is good at people" rather than a specific subgroup of musicians for example (14 out of 50), and those who gave irrelevant explanations (11 out of 50, this group performed non trivially and so could not be disqualified). Table C.4 in Appendix C.6 gives examples of the actual lessons that users wrote. Results for participants who had accurate vs not accurate lessons are shown in the last four columns of Table 4.3. The participants who had accurate lessons had a 9 point average overall F1 difference significant at p -value 0.01 compared to those with inaccurate lessons. With LIME-Teaching we found that only 14 out of 50 participants were able to properly extract the right lessons. The difference between LIME and our method in enabling teaching is significant at p -value 0.02 with $t = 2.3$, however, we observe that accurate teaching has a similar effect in both conditions. Note, that even when participants have accurate lessons, they often don't always follow their own recommendations as evidenced by the low Defer F1 score. This provides evidence that the impact of onboarding is potentially bimodal, with one group having a significant increase in its performance and one group getting no benefit from onboarding.

Differences in performance on in-distribution and out-of-distribution examples. During teaching with our method we let the users solve 9 examples, each corresponding to a unique cluster.

The data domain is in fact split into 11 clusters where the AI has a different error probability in $\{0, 1\}$ on each of them. Thus, there are 2 clusters where users have not seen examples from, which we call the out-of-distribution examples (OOD), and 9 from which they have, the in-distribution examples (ID). In table 4.4 we show the different metrics split into ID and OOD distribution for teaching participants in our method and for the no-teaching participants as a reference point. LIME-Teaching participants observe all the clusters during teaching so there is no distinction between ID and OOD. We can first observe a very high F1 for OOD examples where the human predicts (Non-Defer F1) for our method. This is also the case for the non-teaching participants, thus the increase in F1 lies with the nature of the examples in the OOD clusters rather than the distinction of them being ID versus OOD. On the other hand, we observe that Defer F1 is higher by 8.36 points on average for ID examples compared to OOD with our teaching method while we do not observe a difference in Defer F1 for the baseline non-teaching group. However, the results are not significant as the 95% confidence intervals overlap.

4.7 Additional Synthetic Experiments

Dataset. To complement our NLP-based experiments, we run a study on the CIFAR-10 image classification dataset [60] consisting of 32×32 color images drawn from 10 classes. For CIFAR we use a WideResNet [76] with no data augmentation that achieves 90.46% test accuracy and the model is trained to minimize the cross entropy loss with respect to the target. We split the dataset into three distinct parts: training set for AI model (90% CIFAR train, 45k), teaching set to obtain teaching images (10% of CIFAR train set, 5k) and test set for the human learner (CIFAR test set, 10k).

Setup. We let $X = Z$ and use the respective models’ last layer encodings as the input space to the teaching algorithm. The message the AI sends is the pair $A = (\hat{y}, \hat{c})$ consisting of the AI prediction and a confidence score (softmax output of model). We assume the human is following the human rejector Assumption 2 and is perfectly learning the radius and actions. We consider the human expert models considered in [26]: let $k \in [10]$, then if the image is in the first k classes the expert is perfect, otherwise the expert predicts randomly. The human prior rejector defers if the AI’s confidence \hat{c} is less than $\epsilon = 0.5$.

Results. We show the results in Table 4.5 for various teaching set sizes for the expert $k = 6$ and a learning curve in Figure 4.5; full results are in Appendix C.4. We compare our approach to solving the problem as learning to defer with the AI deferring to the human: we compare to the surrogate loss baseline in [26], the confidence baseline in [25] and a ModelConfidence baseline which optimizes over the prior parameter ϵ . We find that with only 4 teaching examples, DOUBLE-GREEDY increases accuracy from 90.98 to 96.3 ± 0.1 on the test set.

Method	CIFAR (acc)
Prior only	90.98 \pm 0.0
DOUBLE-GREEDY @T=4	96.3 \pm 0.1
DOUBLE-GREEDY @T=8	96.4 \pm 0.1
DOUBLE-GREEDY @T=14	96.5 \pm 0.1
K-Medoids @T=4	94.58 \pm 0.3
K-Medoids @T=8	95.5 \pm 0.2
K-Medoids @T=14	96.5 \pm 0.2
Random @T=8	95.3 \pm 0.5
Oracle	97.91
Surrogate Loss [26]	97.1
Confidence [25]	95.5
ModelConfidence	93.94

Table 4.5: Synthetic experiment on CIFAR-10, showing the test Accuracy for our method DOUBLE-GREEDY at different teaching set sizes and learning to defer baselines.

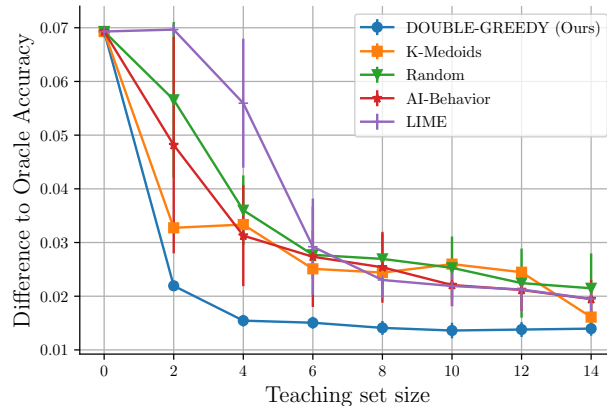


Figure 4.5: Synthetic experiment on CIFAR-10, showing difference between the performance of the methods and ORACLE (defined as taking the optimal decision at test time) for expert $k = 6$.

4.8 Discussion

One limitation of our human experiments is that we used a simulated AI that has an easier to understand error boundary. This enabled us to have a more in-depth study of the crowdworker responses than otherwise would have been possible. Having a simulated AI which we perfectly understand where its error regions are, enables us to define what the "lessons" should be and

thus evaluate if users are learning correctly. Future user studies will evaluate with non-simulated AI models. We hypothesize that the example selection algorithm presented in this work will be sufficient, however, we might require better methods to illustrate the neighborhood for each example. Another limitation is that our test-time interface did not include model explanations, which was done to eliminate additional confounding factors when comparing approaches. Future work will evaluate whether the effect of teaching remains as significant when evaluating with test-time model explanations. Other limitations include the fact that we are using a proxy task of passage based question answering and proxy tasks have been documented to be misleading for evaluating AI systems [123]. Another limitation is the use of MTurk which may not ensure high quality workers and the final limitation is that our study only focuses on the onboarding phase of AI deployment.

Teaching is used in our work to influence human’s perception of an AI model; this can be potentially used to manipulate workers into relying on AI agents in high stakes settings if the AI predictions during teaching were fabricated. While our work was conducted in a low stakes scenario and was designed to portray an accurate reflection of the AI performance, it is possible by manipulating the AI predictions during teaching to have the worker learn any desired rejector. We believe if the data used during teaching is not manipulated, then our approach can serve to give an unbiased overview of the AI.

Chapter 5

Effective Human-AI Teams via Learned Natural Language Rules and Onboarding

Acknowledgements of Co-authors. This chapter is based on the published work in [124]. I would like to thank my co-author Jimin J Lee for her help with building the interface for the user study and preliminary analysis of user study results.

5.1 Introduction

How can we collaborate better with AI models? In this chapter, we propose an intuitive framework for thinking about human-AI collaboration where the human first decides on each example whether they should rely on the AI, ignore the AI, or collaborate with the AI to solve the task extending the previous chapter. We refer to these three actions as the *AI-integration decisions*. The human-AI team will perform optimally if the human knows which action is best on a task-by-task basis. We propose IntegrAI (Figure 5.1), an algorithm that leverages data from baseline human interactions with the AI to learn near-optimal integration decisions, in the form of *natural language rules* that are easily understandable. These rules are then taught to the human through an onboarding stage, analogous to an onboarding class that humans might take before operating machines and equipment. Onboarding additionally calibrates the human’s expectations about AI performance. We further investigate surfacing the AI-integration decisions found by IntegrAI as recommendations to the human within an AI dashboard used after onboarding. The hope is that onboarding and the dashboard help the human know which action they should take, thereby leading to effective AI adoption for enhanced decision-making.

Learning AI-integration rules requires a dataset of paired examples and human predictions (Figure 5.1 Step 1). Each rule is defined as a bounded local region centered around a learned point in a potentially multi-modal embedding space spanning the task space and natural language

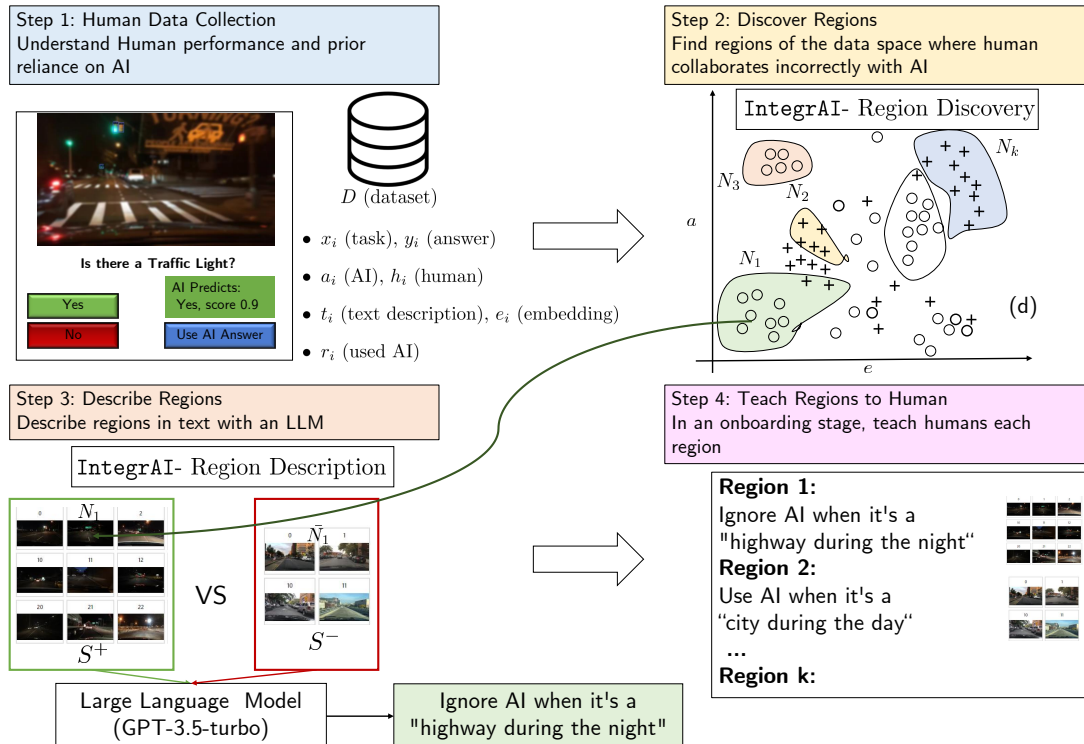


Figure 5.1: The proposed onboarding approach with the IntegrAI algorithm.

(Figure 5.1 Step 2). For example, CLIP embeddings [125] connect image and text spaces for tasks involving images, and typical text embeddings [126] are used for natural language tasks such as question answering. The regions are obtained with a novel region discovery algorithm. Then, a text description of the region is generated, resulting in a rule that indicates whether the human should ignore, rely on, or collaborate with the AI. We obtain descriptions using a novel procedure that connects the summarization ability of a large language model (LLM) [127] with the retrieval ability of embedding search to find similar and dissimilar examples. The procedure first queries the LLM to describe points inside the region (Figure 5.1 Step 3). The embedding space is then leveraged to find counterexamples inside and outside the region to refine the description.

We first evaluate the ability of our region finding and region description algorithms to find regions that will aid the human-AI team in several real-world datasets with image and text modalities. We then investigate the efficacy of both algorithms in synthetic scenarios where we know the ground truth regions. Finally, we conduct user studies on tasks with real-world AI models to evaluate our onboarding and AI-integration recommendation methodology. Our main task is detecting traffic lights in noisy images [128] from the perspective of a road car, motivated by applications to self-driving cars. The user study reveals that our methodology significantly improves the accuracy of the human-AI team by 5.2% compared to no onboarding. We investigate a second task of multiple choice question answering using the MMLU dataset [129] and find that onboarding has no effect on

performance and that only displaying AI-integration recommendations has a negative effect. To summarize, the key contributions of this chapter are as follows:

- Our region discovery algorithm finds regions that help the human know when to rely on the AI (IntegrAI-Discover Section 5.4.1).
- Our region description algorithm can describe regions using an LLM by contrasting points inside and outside the region. (IntegrAI-Describe Section 5.4.2). We evaluate the performance of our algorithms in isolation in experiments in Section 5.6.
- We demonstrate the effect of onboarding and displaying AI-integration recommendations on two real-world tasks, and find that onboarding has a significant positive effect in one task whereas the integration recommendations are not useful in the second task (Section 5.7). In all studies, we present users with information about both human and AI performance in a Human-AI card (Section 5.5).

5.2 Related Work

This work builds on the previous chapter 4 where we proposed an onboarding procedure that involved participants describing different regions of the data space where AI made significant mistakes or was significantly better than human performance. We found that only 50% of participants had accurate descriptions of the underlying regions; we could measure this percentage as the AI model was synthetic. The participants who accurately guessed the correct regions had significantly better performance than those who didn't. This motivates our work on automating the process of describing the regions and more rigorously evaluating the impact of onboarding.

A growing literature of empirical studies on AI-assisted decision making has revealed that human-AI teams do not perform better than the maximum performance of the human or AI alone even with AI explanations [13], [130], [131]. This can be summarized with the following conjecture in equation form: $\text{Human} + \text{AI} \leq \max(\text{Human}, \text{AI})$ (where accuracy is the unit). Note that equality can be achieved by an expert deferral system where a second AI model decides who between the human or the AI should predict [26].

[132] proposes a method for human-AI collaboration via conditional delegation rules that the human can write down. Our framework enables the automated learning of such conditional delegation rules for more general forms of data that can also depend on the AI output. [15] proposes to modify the confidence displayed by the AI model to encourage and discourage reliance on the AI model appropriately. However, this technique deliberately misleads the human on the AI model ability. Our methodology incorporates similar ideas by learning the human prior function of reliance

on the AI and then improving on it with the learned integration recommendations; however, we display these recommendations in a separate dashboard without modifying the AI model output. A related approach to our methodology by [133] is to adaptively display or hide the AI model prediction and display the estimated confidence level of the human and the AI on a task of predicting whether a person’s income exceeds a certain level. They show that displaying the confidence of the human and the AI to the human improves performance. Our method is able to learn the confidence level of the human and the AI, but also incorporates how the human utilizes the AI and describes the regions where AI vs human performance is different. [134] presents a similar approach to our AI recommendations, however, they use simulated and faked AI models and descriptions of behavior while we are able to obtain automated generation of these descriptions of AI behavior.

A growing literature exists on onboarding humans to work with AI models [38], [39], [82], [103], [134], [135]. Our work differs in enabling the automated creation of onboarding material without any human in the loop. We compare our approach to a representative work from a body of research on discovering regions of errors in AI models [136]–[144]. Note however that our work focuses on regions of *disparate performance* between human and AI. Learning to defer methods learn models that decide using a secondary AI model who between the human and the AI classifier should predict [23], [25], [26], [28], [61], whereas [145], [146] propose methods to ensure fairness in such selective deferral settings. This chapter, in contrast, focuses on the reverse setting where the human makes all decisions but we do utilize some of the thinking from that literature. Our AI-integration recommendations are also related to personalized policies [147]. Our MMLU experiments share similarities with recent work [148]–[151]. Further comparison to prior work can be found in Appendix D.1.

5.3 AI Assisted Decision Making

Setting. We consider a setting where a human is making decisions with the help of an AI agent who provides advice to complete a **task**. Formally, the **human** has to make a decision $Y \in \mathcal{Y}$ given access to information about the context as $X \in \mathcal{X}$ and the AI’s advice $A \in \mathcal{A}$. We denote the human as a potentially randomized function $H(X, A; \theta_h)$ with parameters θ_h which are unobservable. On the other hand, the **AI** agent provides advice based on its viewpoint of the context X according to $M(X; \theta_m) := A \in \mathcal{A}$. The advice always includes a candidate decision \hat{A} and possibly an explanation of the decision. We assume that the observed tasks are drawn from an underlying distribution, $\mathbb{P}_{X,Y}$, over the contexts of AI and human, and the ground truth. The setting is illustrated in Figure 5.2.

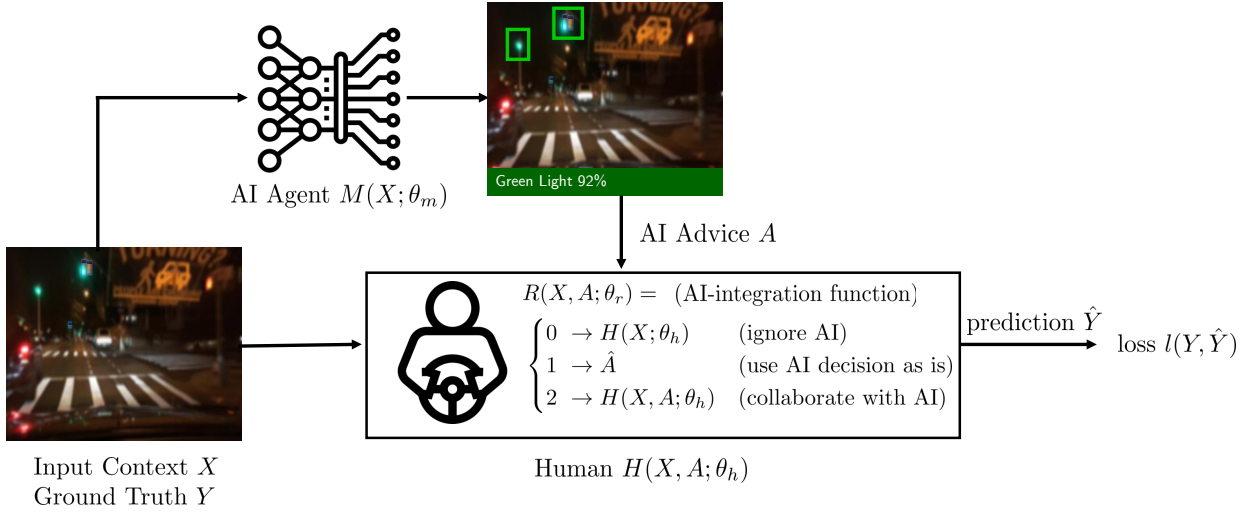


Figure 5.2: The setting of AI-assisted decision making studied in this work. We show an example of an AI system providing assistance to a human driver to inform them about traffic lights in a low visibility situation. The AI provides advice to the human who then incorporates it to make a final decision.

Task Metrics. The human wants to make a decision that optimizes various metrics of interest. Given a ground truth decision Y and a chosen decision \hat{Y} , the loss is given by $l(Y, \hat{Y}) : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$. In our example, this could be the 0-1 loss $\mathbb{1}_{Y \neq \hat{y}}$. We denote the loss $L(H, M)$ of the Human-AI team over the entire domain as:

$$L(H, M) := \mathbb{E}_{x, y \sim \mathbb{P}} [l(y, H(x, M(x; \theta_m); \theta_h))] \quad (5.1)$$

We are further interested in metrics that convey efforts undertaken by the human during the process. Particularly, we focus on *time to make a decision*, which can be measured when the human makes decisions.

Human-AI team. The decision of the Human-AI team is represented by the function $H(X, A; \theta_h)$. The human without the AI is denoted by $H(X, \emptyset; \theta_h) := H(X; \theta_h)$, which is obtained by setting the AI advice to the empty set, i.e. no advice. In theory, we expect the human with advice to perform at least as well as without advice, simply because the human can always ignore the advice. However, the literature on Human-AI teams has clearly demonstrated that this is often not the case [152]. An **effective** Human-AI team is one where the human with the AI's advice achieves a more favorable trade-off in terms of the metrics than without the advice.

Framework for Cooperation. Our insight into forming an effective team is to explicitly recommend when the human should consider the advice and how to incorporate it into their decision. We propose a two-stage framework for the human to cooperate with the AI: the human first decides whether to ignore the AI advice, use the AI’s decision or integrate the AI advice to make a decision with explicit cooperation. Each of these three cases provides a clear path to the second stage of making the final output decision.

Definition 3. The AI-integration function $R(X, A; \theta_r)$, also referred to as integrator, formalizes a framework for the human to cooperate with the AI:

$$R(X, A; \theta_r) = \begin{cases} 0 \rightarrow H(X; \theta_h) & \text{(ignore AI)} \\ 1 \rightarrow \hat{A} & \text{(use AI decision as is)} \\ 2 \rightarrow H(X, A; \theta_h) & \text{(collaborate with AI)} \end{cases} \quad (5.2)$$

In this work, we only consider the actions of ignoring or using the AI decision, $R \in \{0, 1\}$, and leave the action of $R = 2$ for future work.

The integration function can be thought of as a specific formalization of the human mental model of the AI [16], [20]. Given an integration function R and a human H , we can define a hypothetical human decision maker H_R who first computes R and then follows its recommendation to make a final decision. Similarly, for each human $H(X, A; \theta_h)$, we can associate an integration function R_H , such that $H_{R_H} = H$. By fixing $H(X; \theta_h)$, one can try to minimize the loss $L(H_R, M)$ over all possible choices of R , an optimal point of such an integration function is denoted R^* . Following the recommendations of the optimal AI-integration function leads to an **effective** Human-AI team.

Learning Rules and Onboarding. There are two problems that need to be solved to achieve this vision: how do we learn such an R^* and how can we ensure that the human follows the recommendations of this R^* ? In the next section, we outline how we approximate the optimal integration function; this is fundamentally a machine-learning problem with its own challenges that we tackle in Section 5.4. The second obstacle is that the human should know to follow the recommendations of R^* . To ensure this, we propose an **onboarding** stage where the human learns about the AI and the optimal integration function. We additionally propose displaying the recommendations as part of an AI dashboard. In the onboarding stage, we will help the human shape their internal parameters (θ_h, θ_r) to improve performance. This is a human-computer interaction (HCI) problem that we tackle in Section 5.5.

5.4 Learning Rules for Human-AI Cooperation: IntegrAI

In this section, we discuss how to learn an *integrator* function $\hat{R} : \mathcal{X} \times \mathcal{A} \rightarrow \{0, 1\}$ to approximate an optimal integrator while being understandable to the human. We first describe the ingredients for this learning (integrator as a set of regions, objective function, dataset) before detailing how we learn regions and describe them in Sections 5.4.1 and 5.4.2 respectively.

Integrator as a Set of Regions. Since the integrator \hat{R} will be used to both onboard the human and provide test-time recommendations as part of an AI dashboard, it should be easily understandable to humans. If the goal was simply to build the most accurate integrator, we could use work on learning to defer [23], [26], [61]. To address the requirement of understandability, we propose to parameterize the integrator in terms of a set of local data regions, each with its own integration decision label as well as a natural language description. More specifically, we aim to learn a variable number of regions N_1, N_2, \dots as functions of (X, A) , the observable context and the AI’s advice. Each region N_k consists of the following: 1) an indicator function $I_{N_k} : \mathcal{X} \times \mathcal{A} \rightarrow \{0, 1\}$ that indicates membership in the region, 2) a textual description of the region T_k , and 3) a takeaway for the region consisting of an integration decision $r(N_k) \in \{0, 1\}$. We additionally want these regions to satisfy a set of constraints so that they are informative for the human and suitable for onboarding.

Maximizing Human’s Performance Gain. Since we are working with human decision-makers, we have to account for the fact that the human implicitly has a **prior** integration function R_0 , which represents how the human would act without onboarding. Thus, in learning integrator regions, our goal is to maximize the human performance gain relative to their prior. The performance gain is defined as follows for points in a region N :

$$G(N, \hat{R}, R_0) = \sum_{i \in N} l(y_i, H_{R_0}(x_i, a_i)) - l(y_i, H_{\hat{R}}(x_i, a_i)), \quad (5.3)$$

where l is the loss defined in our problem setting. Note that the notion of a human’s prior mental model was also discussed by [82] but we expand on the notion and are able to learn priors as we discuss below.

Dataset with Human Decisions. We assume we have access to a dataset $D = \{x_i, y_i, h_i, r_{0i}\}_{i=1}^n$ sampled from \mathbb{P} , where x_i is the AI-observable context, y_i is the optimal decision on example i , h_i is a human-only decision as $H(x_i; \theta_h)$, and $r_{0i} \in \{0, 1\}$ is an indicator of whether the human relied on AI on example i . We thus regard the samples $\{r_{0i}\}_{i=1}^n$ as a proxy for prior human integration function R_0 . The prior integration decisions of the human r_{0i} are collected through a data collection study

where the human predicts with the AI without onboarding. For example in Figure 6.4, when the human presses on the "Use AI" button we record $r_{0i} = 1$, and 0 otherwise. The human predictions h_i are collected through a second data collection study where the human makes predictions without the AI. We also assume that we are given an AI model M , from which we obtain AI decisions $\hat{a}_i \in a_i$ from the AI advice $a_i = M(x_i; \theta_m)$ ¹. Given the dataset D , AI decisions \hat{a}_i , and loss $l(\cdot, \cdot)$, we can define optimal per-example integration decisions r_i^* by comparing human and AI losses on the example: $r_i^* = \mathbb{I}(l(y_i, h_i) > l(y_i, \hat{a}_i))$.

5.4.1 Region Discovery Algorithm

In this subsection, we describe a sequential algorithm that starts with the prior integration function R_0 and adds regions N_k one at a time.

Representation Space. The domain \mathcal{X} for the task may consist of images, natural language, or other modalities that possess an interpretable representation space. We follow a similar procedure for all domains. The first step is to map the domain onto a potentially cross-modal embedding space using a mapper $E(\cdot)$, where one of the modes is natural language. The motivation is that such an embedding space will have local regions that share similar natural language descriptions, enabling us to learn understandable rules. For example, for natural images, we use embeddings from the CLIP model [125]. The result of this step is to transform the dataset $\{x_i\}_{i=1}^n$ into a dataset of embeddings $\{e_i\}_{i=1}^n$ where $e_i \in \mathcal{E}_{\mathcal{X}} \subseteq \mathbb{R}^d$.

Region Parameterization. We define region N_k in terms of a centroid point c_k , a scaled Euclidean neighborhood around it, and an integration label $r_k \in \{0, 1\}$. The neighborhood is in turn defined by a radius $\gamma_k \in \mathbb{R}$ and element-wise scaling vector w_k . Both c_k and w_k are in $\mathcal{E}_{\mathcal{X}} \times \mathcal{A}$, the concatenation of the embedding space and the AI advice space. The indicator of belonging to the region is then $I_{N_k}(e_i, a_i) = \mathbb{I}_{\|w_k \circ ((e_i, a_i) - c_k)\|_2 < \gamma_k}$, where \circ is the Hadamard (element-wise) product.

Region Constraints. We add the following constraints on each region N_k to make them useful to the human during onboarding. First, the region size in terms of fraction of points contained must be bounded from below by β_l and above by β_u . Second, the examples in each region must have high agreement in terms of their optimal per-example integration decisions r_i^* . Specifically, at least a fraction α of the points in a region must have the same value of r_i^* . Finally, each region must have at least a gain (5.3) of δ , simply speaking adding the region to our integrator provides a gain of δ in terms of our loss l .

¹Moreover, we assume the AI was not trained on the dataset D so that we can use D to obtain an unbiased measurement of AI performance; this is crucial, as otherwise, we might overestimate its performance.

IntegrAI-Discover. Our procedure is fully described in Algorithm 3. In round k , we add a k th region to the integrator R_{k-1} from the previous round (with $k - 1$ regions) to yield R_k . After T rounds, the updated integrator R_T is defined as follows: Given a point (x, e, a) where e is the embedding of x , if it does not belong to any of the regions N_1, \dots, N_T , then we fall back on the prior. Otherwise, we take a majority vote over all regions to which (x, e, a) belongs:

$$R_T(x, e, a) = \begin{cases} R_0(x, a) & \text{if } I_{N_k}(e, a) = 0, k = 1, \dots, T \\ \text{majority}(\{r(N_k) : k \text{ s.t. } I_{N_k}(e, a) = 1\}) & \text{otherwise.} \end{cases} \quad (5.4)$$

In round k , we compute the potential performance gain for each point if we were to take decision r on the point as $g_{i,r} = l(y_i, H_{R_{k-1}}(x_i, a_i)) - l(y_i, r)$; this vector of gains is denoted by \mathbf{g} . The optimization problem to find the optimal regions is non-differentiable due to the discontinuous nature of the region indicators. To make this optimization problem differentiable, we relax the constraints to be penalties with a multiplier λ and replace the indicators with sigmoids scaled by a large constant C_1 . To find a new region R given a gain vector \mathbf{g} we need to solve the following optimization problem:

$$\max_{c, \gamma, w, r} J(c, \gamma, w, r; \mathbf{g}) := \sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) \cdot g_{i,r} \quad (\text{maximize gain}) \quad (5.5)$$

$$- \lambda \max\left(\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) \cdot \mathbb{I}_{r_i^* = r} + \alpha n, 0\right) \quad (\text{consistent takeaway}) \quad (5.6)$$

$$- \lambda \max\left(\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) - \beta_u n, 0\right) \quad (\text{region max size}) \quad (5.7)$$

$$- \lambda \max\left(-\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) + \beta_l n, 0\right) \quad (\text{region min size}) \quad (5.8)$$

The optimization variables c, γ, w , and r to find the region are all real-valued except for r which is a binary variable. Thus to optimize the objective J , we use AdamW to optimize over (c, γ, w) twice: once for $r = 0$ and once for $r = 1$ and pick the solution that has the highest objective value between the two. The hyperparameters of the algorithm are the minimum size of the region β_l , maximum size of the region β_u , consistency of the region α , and minimum gain of the region δ . Further details can be found in Appendix D.2.

5.4.2 Region Description Algorithm

We now describe our region description algorithm aimed at making the rules for integration human-understandable. Natural language descriptions are a good match for this objective. Specifi-

Algorithm 3: IntegrAI-Discover

Input: Dataset D , prior integrator R_0 , maximum number of regions T

- 1: $\mathcal{N} \leftarrow \emptyset$ (regions found so far)
- 2: $C^0 = \{c_1^0, \dots, c_{100}^0\}$: set of possible initializations from **K-medoids** on D
- 3: **for** $k = 1, \dots, 2T$ **do**
- 4: **for** $r \in \{0, 1\}$ **do**
- 5: **Compute Gain Vector:** \mathbf{g} (gain from action r on each point given R_{k-1})
- 6: **Initialization:** $c^0, \gamma^0, w^0 = \arg \max_{c \in C^0, \gamma, w} J(c, \gamma, w, r; \mathbf{g})$ (search over C^0 with 200 epochs (5.5))
- 7: **Full Optimization:** $c, \gamma, w = \arg \max_{c, \gamma, w, r} J(c, \gamma, w, r; \mathbf{g})$ (optimization starts from initialization above for 2000 epochs)
- 8: **Form candidate region:** N_k^r (from c, γ, w, r)
- 9: **end for**
- 10: **Form potential integrators:** \hat{R}_0, \hat{R}_1 by adding N_k^0 and N_k^1 respectively
- 11: $r^* = \arg \max_{r \in \{0, 1\}} G(N_k^r, \hat{R}_r, R_{k-1})$ (**find best takeaway decision from last step following (5.3)**)
- 12: **if** $G(N_k^{r^*}, \hat{R}_{r^*}, R_{k-1}) \geq \delta$ **then**
- 13: $\mathcal{N} = \mathcal{N} \cup N_k^{r^*}$ (**add region to set** - as it has high enough gain)
- 14: **Form new integrator:** R_k by adding region $N_k^{r^*}$ following (5.4).
- 15: **else**
- 16: **Form integrator:** $R_k \leftarrow R_{k-1}$ (no update)
- 17: **end if**
- 18: **if** $|\mathcal{N}| == T$ **then**
- 19: **break** (exit for loop) - we have enough regions
- 20: **end if**
- 21: **end for**

Return: Set of Regions discovered \mathcal{N}

cally, we would like to find a contrastive textual description T_k of each region N_k that describes it in a way to distinguish it from the rest of the data space. The algorithm is formally described in Algorithm 4 and illustrated in Figure 5.3.

Textual Descriptions for Regions: The first step is to have a textual description t_i for each example in our dataset D based on x_i . If textual descriptions are not available, we can obtain them by utilizing models that map from the domain \mathcal{X} to natural language, such as captioning models for images, summarization models for text, or exploiting metadata to construct a natural language description. The information that is captured in the textual description of each example is a bottleneck for the description of the region, as the region descriptions can only summarize what is described in the example textual descriptions. If the domain of the example is text, this is less of an issue as we can easily generate and tweak the textual descriptions with summarization methods. If the domain is images, commonly found image captions are not always sufficiently descriptive [153];

Algorithm 4: IntegrAI-Describe

Input: Dataset D , region N_k

- 1: $S^+ \leftarrow 15$ random examples from N_k , $S^- \leftarrow 5$ random examples outside N_k
- 2: **Initial Region Description:** $T_k^0 \leftarrow \hat{O}(S^+, S^-)$ (LLM call)
- 3: **for** $i = 1, \dots, m$ **do**
- 4: **Find Counterexample outside region.** $s^- = \arg \max_{j \notin N_k} \text{sim}(E(T_k^i), e_j)$ (most similar outside region)
- 5: **Find Counterexample inside region.** $s^+ = \arg \min_{j \in N_k} \text{sim}(E(T_k^i), e_j)$ (least similar inside region)
- 6: **Update Inside and Outside Sets.** $S^- \leftarrow S^- \cup t_{s^-}$ and $S^+ \leftarrow S^+ \cup t_{s^+}$
- 7: **Get New Region Description:** $T_k^i \leftarrow \hat{O}(S^+, S^-)$ (LLM call)
- 8: **end for**

Return: Region description T_k^m

thus, one can use image-to-text methods to describe the example; however, in our experiments, they underperform text-to-text summarization.

To obtain a region description, one idea is to ask an LLM (such as GPT-3.5) to summarize all textual descriptions of points inside the region. However, there are two issues with this approach: first, the region may contain thousands of examples so we need an effective way to select which points to include, and second, the obtained region description may not contrast with points outside the region. To resolve these issues, we propose an algorithm that iteratively refines region descriptions with repeated calls to an LLM (\hat{O}) in Algorithm 4 (IntegrAI-Describe). The algorithm starts with an initial description and then at each round finds two types of counterexamples to that description: examples outside the region (type $-$) with high cosine similarity in terms of embeddings (sim) to the region description, and examples inside the region (type $+$) with low similarity to the region description. Then we add those counterexamples to our example sets and derive a new region description. We use a specially created prompt with an exemplar to the LLM to get the region description at each round; this prompt can be found in Appendix D.3.

Illustrative Example. Suppose we want to describe a region consisting of images of highways during the night, with no cars present (see Figure 5.3 for images of BDD [128]). Our method’s initial description is “The highway during the night with clear, rainy or snowy weather,” not mentioning that the highway has no cars, particularly because the captions of examples t_i only mention the presence of cars and not their absence. In the second round, the algorithm finds the counterexample s^- with caption “city street during the night with clear weather with a lot cars” and counterexample s^+ “highway during the night with clear weather.” The new description T_k^1 becomes “clear highway during the night with various weather conditions, while outside the region are busy city street at night with clear weather.” After one more round, the description T_k^2 becomes “driving on a clear

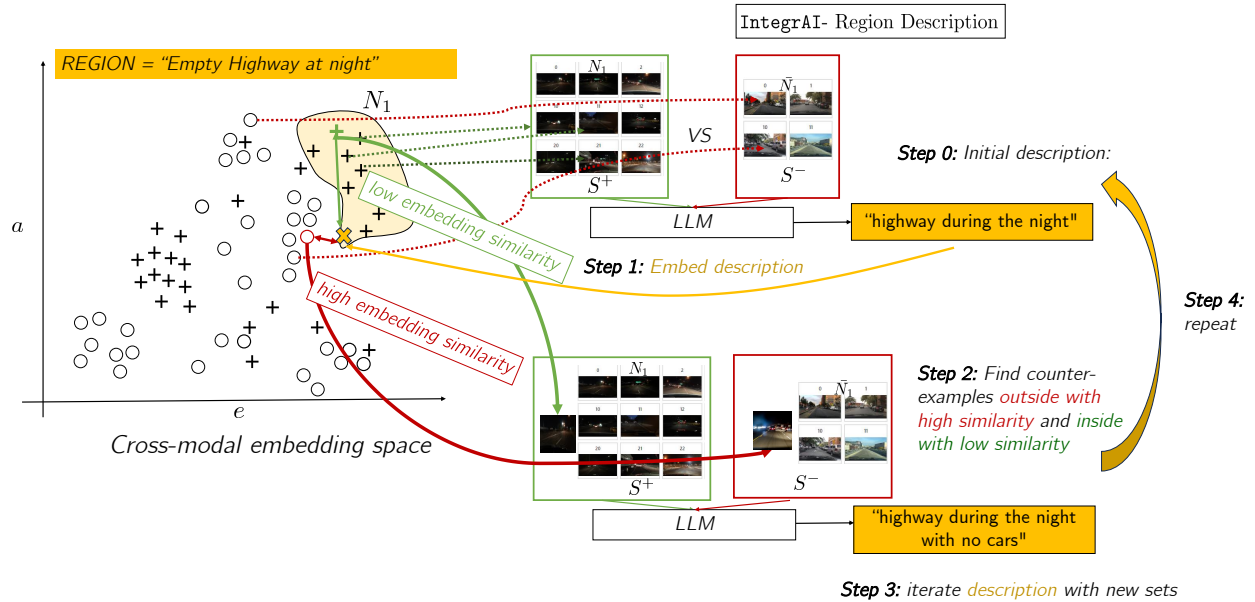


Figure 5.3: The IntegrAI-Describe algorithm illustrated. To obtain a description for a region of points, the algorithm first samples a set of points inside and outside the region and gets a description from an LLM that contrasts inside versus outside (Step 0). We then embed the obtained description in our cross-modal embedding space (Step 1) and find counterexamples to that description, both points outside the region with high similarity to the description and points inside the region with low similarity to the description (Step 3). The process is repeated for as many rounds as necessary (Step 4).

and uncongested highway during the night in various weather conditions.” A simplified version of this example is shown in Figure 5.3. We now proceed in the next section to describe how we onboard the human decision maker using the regions.

5.5 Onboarding and Recommendations to Promote Rules

Once rules for integration have been learned as described in the previous section, the task is to teach these rules to the human with an **onboarding** stage and encourage their use at test time. We accomplish this through an **onboarding** process followed by test-time **recommendations**, as described next.

Human-AI Card. The onboarding process accordingly consists of an *introductory* phase and a *teaching* phase. In the *introductory phase*, the user is first asked to complete a few practice examples of the task on their own to gain familiarity. The user is then presented with general information about the AI model in the form of a **human-AI card**, very similar to a model card [154] and inspired by [103]. The card structure showcased in Table 5.1 includes the AI model inputs and outputs,

training data, training objectives, overall human and AI performance along with AI performance on subgroups where performance deviates from average performance. This card represents the bare minimum that humans should know about the AI model before collaborating with it. The instantiation of the Human-AI card for the BDD user study is shown in Table 5.2.

Table 5.1: Human-AI Card presented to the human as part of onboarding

Information	Description
AI Input	What the AI uses to make its prediction
AI Output	What the AI provides as output (predictions, explanations, ...)
Source of Training Data for AI	Description of data used to train the AI
Source of Pre-Training Data of AI	Description of pre-training data that AI is based on
Training Objective of AI	What the AI is trying to achieve (minimize classification error, detect objects, next word prediction)
Average AI Performance	Relevant metrics of overall AI performance (accuracy, FPR, ...)
Average Human Performance	Relevant metrics of overall human performance (accuracy, FPR, ...)
AI vs Human Performance on Subgroups	

The **teaching phase** aims to provide a more detailed picture of how the human should collaborate with the AI. It is structured as a sequence of “lessons”, each corresponding to a region N_k resulting from the algorithm of Section 5.4.1. The specific steps in each lesson are as follows:

- **Step 1: Human predicts on example.** A representative from the region is selected at random that has an optimal integration decision identical to that of the region. The human is asked to perform the task for the chosen representative and is shown the AI’s output along with the option to use the AI’s response.
- **Step 2: Human receives feedback.** After submitting a response, the user is told whether the response is correct and whether the AI is correct.
- **Step 3: From example to region learning.** The user is informed that the representative belongs to a larger region N_k and is provided with the associated recommendation r_k , textual

description t_k , and AI and human performance in the region as well as the raw examples from the region in a gallery viewer.

After completing all lessons as above, a second pass is done where the user is re-shown all lessons for which their response was incorrect. This serves to reinforce these lessons and is similar to online learning applications such as Duolingo for language learning [155]. Our teaching approach is motivated by literature showing that humans learn through examples and employ a nearest neighbor type of mechanism to make decisions on future examples [85], [115], [116]. We follow this literature by showing concrete data examples in the belief that it effectively teaches humans how to interact with AI. We improve on the approach in [82] by incorporating an introductory phase with the Human-AI card, showing pre-defined region descriptions for each region, and iterating on misclassified examples in the teaching phase.

Recommendations in AI Dashboard. At test time, we can check whether an example x and its corresponding AI output a fall into one of the learned regions N_k . If they do, then our dashboard shows the associated recommendation r_k and description t_k alongside the AI output a .

Table 5.2: The exact Human-AI card used in the user study for BDD.

Attribute	Description
Average AI Accuracy	78%
Average Human Accuracy	72%
AI Model Input	Blurry Image
AI Model Output	Prediction of traffic light, bounding box on image showing its location and a score indicating its confidence
Source of Training Data	Dataset of road images from New York and San Francisco Bay Area
AI Training Objective	Detect traffic lights and other objects in image

Category	Accuracy
more than 5 traffic lights in image	90%
1 to 4 traffic lights	62%
no traffic lights	86%
no cars	83%
daytime or overcast weather	75%
few pedestrians	76%

5.6 Method Evaluation

Objective. In this experimental section², we evaluate the ability of our algorithms to achieve three aims: (Aim 1) Learn an integration function that leads to a human-AI team with low error; (Aim 2) discover regions of the data space that correspond to the underlying regions where human vs AI performance is different; and, for our region description algorithm, (Aim 3) come up with accurate descriptions of the underlying regions. Full experimental details are in Appendix D.5.

Datasets and AI Models. The experiments are performed on two image object detection datasets and two text datasets. The image datasets include Berkeley Deep Drive (BDD) [128] where the task is to detect the presence of traffic lights from blurred images of the validation dataset (10k), and the validation set of MS-COCO (5k) where the task is to detect whether a person is present in the image³ [156]. The text-based validation datasets are Massive Multi-task Language Understanding (MMLU) [129], and Dynamic Sentiment Analysis Dataset (DynaSent) [157]. The pre-trained Faster R-CNN models [158] are considered for BDD and MS-COCO. For MMLU, a pre-trained flan-t5 model [159] is utilized, whereas a pre-trained sentiment analysis model roBERTa-base is used for DynaSent [160]. Each dataset is split into a 70-30 ratio for training and testing five different times so as to obtain error bars of predictions. We obtain embeddings using a sentence transformer [126] for the text datasets and CLIP for the image datasets [125]

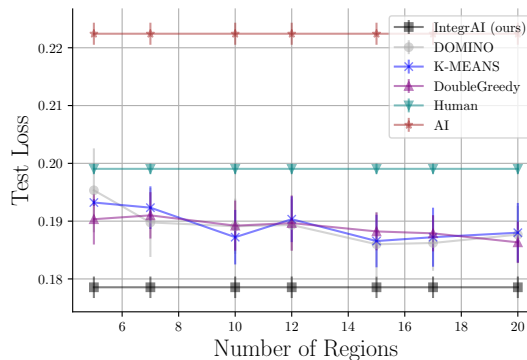


Figure 5.4: Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators as we vary the number of regions maximally allowed for each integrator on the BDD dataset.

Baselines. We benchmark our algorithm with different baseline methods that can find regions of the space. The baselines include: (a) DOMINO [136] which is a slice-discovery method for AI errors, (b) K-Means following the approach of [143], and (c) the double-greedy algorithm from [82] that finds regions for Human-AI onboarding. For the regions obtained from these baselines, we compute the optimal integration decision that results in minimal loss. For our method, we set

²Code is available in https://github.com/clinicalml/onboarding_human_ai.

³We extend this to detecting the presence of any object.

$\beta_u = 0.5, \beta_l = 0.01, \alpha = 0.0$ for Aim 1 and $\beta_u = 0.2, \beta_l = 0.01, \alpha = 0.8$ for Aim 2, random prior decisions (50-50 for 0 and 1), and $\delta = 2$. In the context of region-description algorithms, we compare to the SEAL approach [143], a simple baseline that picks the best representative description from the existing dataset (best-caption), and ablations of our method. For Aim1 and Aim2, we repeat each experiment 5 times and report the average value and standard error (standard deviation divided by $\sqrt{5}$).

Table 5.3: Error (\downarrow) on the test set (in %) of the human-AI system when following integrators resulting from different region discovery methods with 10 regions on the different non-synthetic datasets.

	BDD	MMLU	DynaSent	MS-COCO
IntegrAI (ours)	17.8 \pm 0.2	45.3 \pm 0.3	20.2 \pm 0.3	22.6 \pm 0.4
DOMINO [136]	18.9 \pm 0.4	48.1 \pm 0.2	20.0 \pm 0.2	22.7 \pm 0.4
K-MEANS [143]	19.0 \pm 0.5	45.3 \pm 0.3	20.0 \pm 0.2	23.2 \pm 0.1
DoubleGreedy [82]	18.9 \pm 0.1	46.1 \pm 0.6	20.0 \pm 0.2	23.8 \pm 0.4

Table 5.4: Clustering metrics (Adjusted Rand index [161] \uparrow , Fowlkes–Mallows index [162] \uparrow) of the regions (10 regions) found by the different methods on the synthetic dataset setup.

	BDD	MMLU	MS-COCO
IntegrAI (ours)	(0.02,0.24)	(0.17, 0.53)	(0.07,0.43)
DOMINO [136]	(0.05,0.20)	(0.11,0.45)	(0.04,0.35)
K-MEANS [143]	(0.05,0.20)	(0.09,0.35)	(0.03,0.27)
DoubleGreedy [82]	(0.01,0.21)	(0.04,0.36)	(0.03,0.29)

Learning Accurate Integrators (Aim 1). The goal is to measure the ability of our method in learning integration functions that lead to low Human-AI team error (the loss $L(H, M)$). This can be well represented by measuring the errors on the training set (discovering regions of error) and the test set (generalization ability). In Table 5.3, we show the results of our method and the baselines at learning integrators and find that our method can find regions that are more informative with respect to Human vs AI performance on the test data. Figure 5.4 shows that on BDD our method can find an integrator that leads to lower loss at test time than the baselines with a minimal number of regions.

Recovering Ground truth Regions (Aim 2). We just established that the regions discovered by our algorithm result in a Human-AI team with lower error than human or AI alone. However, it still needs to be verified if the regions are indeed meaningful and consistent regions of space. We utilize a synthetic setup by simulating the AI model and the human responses such that there exist (randomized) regions in the data space where either the human or the AI are accurate/inaccurate (though the regions may slightly overlap). These regions are defined in terms of metadata. As an example on the BDD dataset, we can define the AI to be good at daytime images and bad at images of highways, and the human to be good at nighttime images and bad at images of city streets. We employ our algorithm and the baselines to discover regions and compare them with the ground truth regions corresponding to the partition of the data, which is essentially a clustering task with ground truth clusters. Results are shown in Table 5.4 and show that we have clustering metrics mostly higher than the baselines.

Describing Regions (Aim 3). We conduct an ablation study where we evaluate the power of the contrasting and self-correcting ability of Algorithm 4 against baselines. On the MS-COCO dataset, we take regions defined in terms of the presence of a single object (e.g., ‘apple’) and try to obtain a single-word description of the region from the image captions. We use standard captioning metrics that compare descriptions from the algorithms to the object name, we include a metric called "sent-sim" that simply measures cosine similarity with respect to a sentence transformer [126]. We compare to ablations of Algorithm 4 with $m \in \{0, 5, 10\}$ (rounds of iteration) and without having examples outside the region (IntegrAI, $S^- = \emptyset$). Results are in Table 5.5 and show that including examples outside the region improves all metrics while increasing iterations (m) further improves results slightly. For the apple example, IntegrAI ($S^- = \emptyset$) finds the description to be “fruit” whereas our IntegrAI ($m = 5$) finds it to be “apple”.

Table 5.5: Evaluation of our region description algorithm (Algorithm 4) on selected subsets of MS-COCO where the different algorithms try to describe a set of images that all contain a given object. For example, a region may be defined by images containing the object “apple”. Then we compare the descriptions resulting from the different algorithms to the description “apple”.

	best-caption	SEAL	IntegrAI ($S^- = \emptyset$)	IntegrAI (m=0)	IntegrAI (m=5)	IntegrAI (m=10)
METEOR	12.9 ± 1.9	9.16 ± 1.89	24.3 ± 3.3	25.4 ± 3.2	26.1 ± 3.3	25.4 ± 3.3
sent-sim	39.8 ± 1.9	44.1 ± 2.5	65.1 ± 3.2	67.0 ± 3.1	66.0 ± 3.2	68.0 ± 3.3
ROUGE	5.81 ± 1.2	0.0 ± 0.0	25.6 ± 4.9	32.6 ± 5.4	27.9 ± 5.1	35.6 ± 5.5
SPICE	12.7 ± 1.9	7.53 ± 2.3	41.1 ± 5.8	43.8 ± 5.8	45.2 ± 5.8	45.2 ± 5.8

5.7 User Studies to Evaluate Onboarding Effect

Tasks. We perform user studies on two tasks: 1) predicting the presence of a traffic light in road images from the BDD dataset [128] and 2) answering multiple-choice questions from the MMLU [129] dataset. For BDD, we blur the images with Gaussian blur to make them difficult for humans and use the Faster R-CNN model as the AI. Participants can see the AI’s prediction, bounding box on the image as an explanation and the model’s confidence score. For MMLU, participants are shown a question, four possible answers and the prediction of GPT-3.5-turbo [6], and then have to pick the best answer. We also obtain an explanation from GPT-3.5 by using the prompt “Please explain your answer in one sentence.” Both the AI answer and the explanation are shown. GPT-3.5 obtains an accuracy of 69% during our evaluation and we restrict our attention to specific subjects within the MMLU dataset. Specifically, we sample 5 subjects (out of the 57 in MMLU) where ChatGPT has significantly better performance than average, 5 where it’s significantly worse, and 4 subjects where performance is similar to average performance. We sample 150 questions from each subject and additionally sample 150 questions from the OpenBookQA dataset [163] to use as

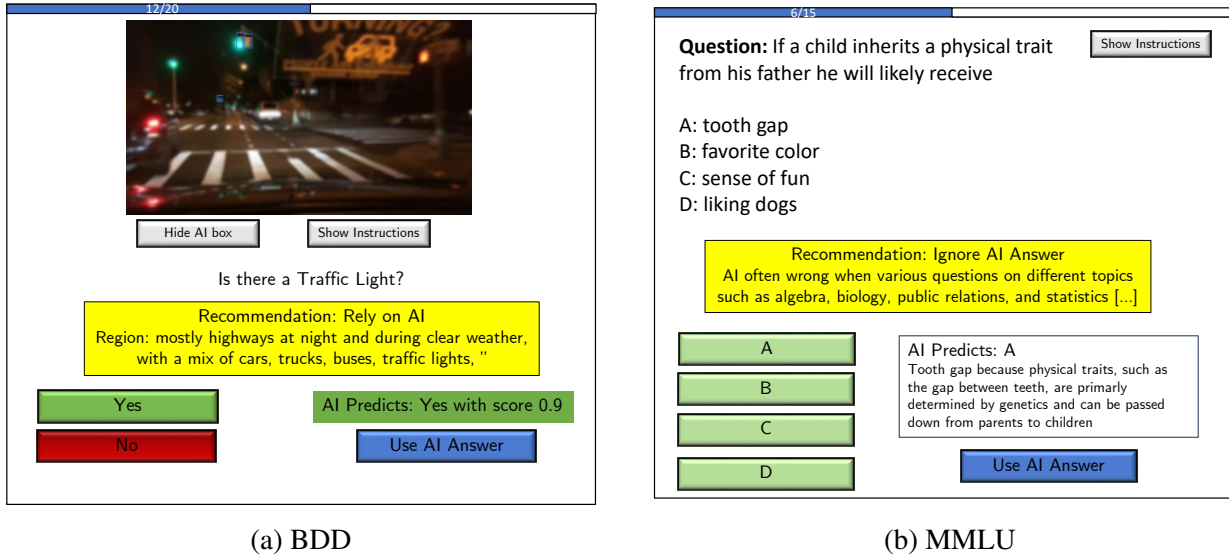


Figure 5.5: (a) Interface for humans to detect a traffic light in images from BDD dataset in the presence of AI’s prediction, confidence score, and bounding box and (b) interface for humans to answer multiple choice questions from MMLU dataset with AI’s prediction and explanation.

attention checks. We show the prediction interfaces in Figure 6.4. Details are in Appendix D.6.

Participants. We submitted an IRB application and the IRB declared it exempt as is. All participants agreed to a consent form for sharing study data. We recruited participants from the crowdsourcing website Prolific [164] from an international pool, filtering for those who are fluent in English, have above a 98% approval rating, have more than 60 previous submissions, and have not completed any of our studies before. For BDD, participants are compensated \$3 per 20 examples in the study and then some receive a bonus of \$2 for good performance. For MMLU, we pay participants \$3 for every 15 questions. We collected information about participants’ age, gender (52% identify as Female), knowledge of AI, and other task-specific questions. Participants have to correctly answer on three initial images without blur in the case of BDD (questions from OpenBookQA for MMLU). They encounter attention checks throughout the study to further filter them as we exclude participants who fail all attention checks.

Experimental Conditions. For BDD, we initially collect responses from 25 participants who predict without the AI and then predict with the help of AI (but no onboarding). We use this data as the basis of the dataset D of prior human integration decisions and human predictions to find 10 regions using IntegrAI. We then run four different experimental conditions with 50 unique participants in each where each participant predicts on 20 examples: (1) human predicts alone (H) and human predicts with the help of AI (H-AI) but no onboarding and random order between

with and without AI, (2) human receives onboarding using our method and then in a random order also receives recommendations (Onboard(ours)+Rec) or no recommendations (Onboard(ours)), (3) human goes through a modified onboarding procedure that only uses step 1 and step 2 from Section 5.5 and then uses regions from DOMINO [136] (Onboard(baseline)), and finally (4) human does not receive onboarding but receives the AI-integration recommendations (Rec). For MMLU, participants are tested on 15 examples per condition and onboarding goes through 7 regions found by our algorithm. We run IntegrAI twice: once on both the dataset embeddings and metadata (subject name), and once on an embedding of ChatGPT explanations separately. We find 10 regions based on the metadata and 2 regions based on the ChatGPT explanations, we show participants the 7 regions that have the highest gain. Due to budget constraints, we only run conditions 1-2-4 for MMLU. Note that all participants receive the introduction phase of the onboarding (AI model information) regardless of whether they go through the teaching phase or not.

Results. In Table 5.6 and Table 5.7 we display various results from the user studies for BDD and MMLU respectively across all experimental conditions. We show the average accuracy (\pm standard error) across all participants of their final predictions, AI reliance as measured by how often they pressed the “Use AI Answer” button, and the average time it took for them to make a prediction per example (we remove any time period of more than 2 minutes). Finally, we compute using a two-sample independent t-test the p-value and t-test statistic when comparing each condition (the columns) to the Human-AI condition where the human receives no onboarding (to compare the Human-only condition to Human-AI we use a paired t-test for this pair only). Since we perform multiple tests, we need to correct for multiple hypothesis testing so we rely on the Benjamini/Hochberg method [165].

Analysis. For BDD, we first observe that human and AI performance are very comparable at around 79%, which reduces slightly to 77.2% when the human collaborates with the AI without onboarding. Participants who go through onboarding have a significantly higher task accuracy compared to those who didn’t go through onboarding (corrected p-value of 0.042) with a 5.4% increase. The onboarding baseline fails to significantly increase task accuracy, showcasing that the increase is not just due to task familiarity but possibly due to insights gained from regions found by IntegrAI. Displaying recommendations in addition to onboarding (Onboard(ours)+Rec) does not improve performance but adds time to the decision-making process (7.6s compared to 5.9s without). For MMLU, we note that there is a 20% gap between human and AI performance, but human+AI with and without onboarding can obtain an accuracy of around 75% which is slightly higher than AI alone; onboarding had no additional effect. Onboard(ours) does result in slightly lower time per example than Human+AI without onboarding. Interestingly, we find a weakly significant negative

Metric	AI only	Human	Human+AI	Onboard(ours)+Rec	Onboard(ours)	Onboard(baseline)	Rec
Accuracy (%)	79.0 ± 0.7	78.5 ± 1.7	77.2 ± 1.4	79.9 ± 1.4	82.6 ± 1.3	80.4 ± 1.4	81.4 ± 1.8
Test vs H-AI	0.272, 1.211	0.455, 0.752	N/A	0.268, 1.352	0.042, 2.747	0.261, 1.525	0.206, 1.839
AI reliance (%)	N/A	N/A	16.5 ± 3.1	66.5 ± 2.4	25.5 ± 3.4	24.4 ± 4.4	21.4 ± 3.2
Time/example (s)	N/A	5.408 ± 0.289	7.78 ± 0.517	7.622 ± 0.371	5.936 ± 0.288	6.841 ± 0.543	8.717 ± 0.516

Table 5.6: Results from our user studies for BDD. For accuracy, time per example, and AI-reliance we report mean and standard error across participants. The "Test vs H-AI" row reports the adjusted p-value and t-test statistic for a two-sample t-test between the human+AI condition and the other conditions (columns).

Metric	AI only	Human	Human+AI	Onboard(ours)+Rec	Onboard(ours)	Rec
Accuracy (%)	72.9 ± 0.6	52.8 ± 2.2	75.0 ± 1.7	73.7 ± 1.8	74.4 ± 1.7	69.8 ± 1.8
Test vs H-AI	0.230, -1.488	0.0, -7.899	N/A	0.747, -0.53	0.792, -0.265	0.101, -2.08
AI reliance (%)	N/A	N/A	40.0 ± 3.6	34.5 ± 3.7	40.6 ± 3.8	34.2 ± 2.9
Time/example (s)	N/A	30.608 ± 2.109	23.623 ± 1.66	22.917 ± 1.362	20.977 ± 1.509	29.535 ± 1.883

Table 5.7: Results from our user studies for MMLU.

effect of only showing AI-integration recommendations, which decreases accuracy by 5% and adds 6 seconds of time per example.

(Informal) Qualitative Analysis. At the end of each experiment, we asked the participant the following question: “What was your decision process for relying on the AI answer and for when to ignore the AI answer?” For the BDD task, we compare the responses of participants who were in the baseline Human+AI condition versus participants in the Onboard(ours) condition. To summarize the responses, we use IntegrAI-Describe with $m = 0$,⁴ first with the region of interest corresponding to the responses from the Onboard(ours) condition (asking it to contrast with Human+AI responses). We get the following description verbatim (and add bolding for emphasis):

Points inside the region involve scenarios primarily where the individual uses or relies on AI when uncertain, particularly when visibility is poor or objects are too distant. In contrast, points lying outside the region pertain to cases where **individuals often ignore the AI** either because they feel confident in their own judgment, the picture is clear, or they believe the AI is not accurate enough.

On the other hand, when we describe the region corresponding to Human+AI responses, we get:

The region comprises descriptions wherein individuals primarily rely on their own judgement to identify traffic lights in images, resorting to the AI’s aid when the image is too blurry, unclear or when doubt exists. In stark contrast, points outside the region

⁴No iteration because we can fit all responses in the context of the LLM which is GPT-4 in this analysis. We changed the prompt to ask for 100 words instead of the usual 20-word limit.

detail instances where reliance on AI is more pronounced or where **external factors like road type and presence of cars are considered.**

One theme that emerges is using AI more in the onboarding condition, which is confirmed quantitatively as participants in the Onboard(ours) condition relied on AI 9% more (see Table 5.6). Another theme is that of participants in onboarding relying on external factors, which can potentially be attributed to lessons learned during onboarding.

Discussion. We believe that for MMLU, due to the wide gap between human and AI accuracy and the availability of GPT-3.5 explanations, onboarding did not improve performance. We note that in a lot of instances, GPT-3.5 explanations express uncertainty over the answer, as in the following examples:

- ‘Unfortunately, the options provided do not provide a clear answer to what happens after the Meyer tree’s flower petals drop. Can you please provide more information or context about the question’
- ‘The answer cannot be provided with the given information as it does not specify which president is being referred to.’
- ‘[...] the answer cannot be provided with the given information as it does not specify which president is being referred to.’

Such statements about uncertainty can already help the human more accurately know when not to trust the AI and try harder to find the correct answer. In cases where GPT-3.5 does not express uncertainty, it tries to explain its answer and often does so correctly, but it is not clear whether the explanations allow the human to easily verify the answer. Moreover, it is clear that in its current form, displaying the AI-integration recommendation is not an effective strategy and that onboarding on its own is sufficient. Finally, note that even the Human-AI baseline benefits from the human-AI card, which might explain why the team is at least as good as its components in our experiments.

Limitations. Onboarding and recommendations can significantly affect human decision making. If the recommendations are inaccurate, they could lead to drops in performance and thus require safeguarding. Onboarding and recommendations can be tailored to the specific human by leveraging their characteristics to few-shot learn their prior integrator and prediction abilities.

Part III

Interactive Human-AI Collaboration: Case Study in LLM-Assisted Programming

Chapter 6

Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming

Acknowledgements of Co-authors. This chapter is based on the published work in [166]. I would like to thank my co-authors, Gagan Bansal, Adam Fourney, and Eric Horvitz, for their help.

6.1 Introduction

Programming-assistance systems based on the adaptation of large language models (LLMs) to code recommendations have been recently introduced to the public. Popular systems, including Copilot [7], CodeWhisperer [167], and AlphaCode[168], signal a potential shift in how software is developed. Though there are differences in specific interaction mechanisms, the programming-assistance systems generally extend existing IDE code completion mechanisms (e.g., IntelliSense¹) by producing suggestions using neural models trained on billions of lines of code [169]. The LLM-based completion models can suggest sentence-level completions to entire functions and classes in a wide array of programming languages. These large neural models are deployed with the goal of accelerating the efforts of software engineers, reducing their workloads, and improving their productivity.

Early assessments suggest that programmers do feel more productive when assisted by the code recommendation models [170] and that they prefer these systems to earlier code completion engines [171]. In fact, a recent study from GitHub, found that Copilot could potentially reduce task completion time by a factor of two [172]. While these studies help us understand the benefits of

¹<https://code.visualstudio.com/docs/editor/intellisense>

code-recommendation systems, they do not allow us to identify avenues to improve and understand the nature of interaction with these systems.

In particular, the neural models introduce new tasks into a developer’s workflow, such as writing AI prompts [173] and verifying AI suggestions [171], which can be lengthy. Existing interaction metrics, such as suggestion acceptance rates, time to accept (i.e., the time a suggestion remains onscreen), and reduction of tokens typed, tell only part of this interaction story. For example, when suggestions are presented in monochrome popups (Figure 6.1), programmers may choose to accept them into their codebases so that they can be read with code highlighting enabled. Likewise, when models suggest only one line of code at a time, programmers may accept sequences before evaluating them together as a unit. In both scenarios, considerable work verifying and editing suggestions occurs *after* the programmer has accepted the recommended code. Prior interaction metrics also largely miss user effort invested in devising and refining prompts used to query the models. When code completion tools are evaluated using coarser task-level metrics such as task completion time [174], we begin to see signals of the benefits of AI-driven code completion but lack sufficient detail to understand the nature of these gains, as well as possible remaining inefficiencies. We argue that an ideal approach would be sufficiently low level to support interaction profiling while sufficiently high level to capture meaningful programmer activities.

Given the nascent nature of these systems, numerous questions exist regarding the behavior of their users:

- What activities do users undertake in anticipation for, or to trigger a suggestion?
- What mental processes occur while the suggestions are onscreen, and, do people double-check suggestions before or after acceptance?
- How *costly* for users are these various new tasks, and which take the most time?

To answer these and related questions in a systematic manner, we apply a mixed-methods approach to analyze interactions with a popular code suggestion model, GitHub Copilot² which has more than a million users. To emphasize that our analysis is not restricted to the specifics of Copilot, we use the term CodeRec to refer to any instance of code suggestion models, including Copilot. Through small-scale pilot studies and our first-hand experience using Copilot for development, we develop a novel taxonomy of common states of a programmer when interacting with CodeRec models (such as Copilot), which we refer to as CodeRec User Programming States (CUPS). The CUPS taxonomy serves as the main tool to answer our research questions.

Given the initial taxonomy, we conducted a user study with 21 developers who were asked to retrospectively review videos of their coding sessions and explicitly label their intents and actions

²<https://github.com/features/copilot>

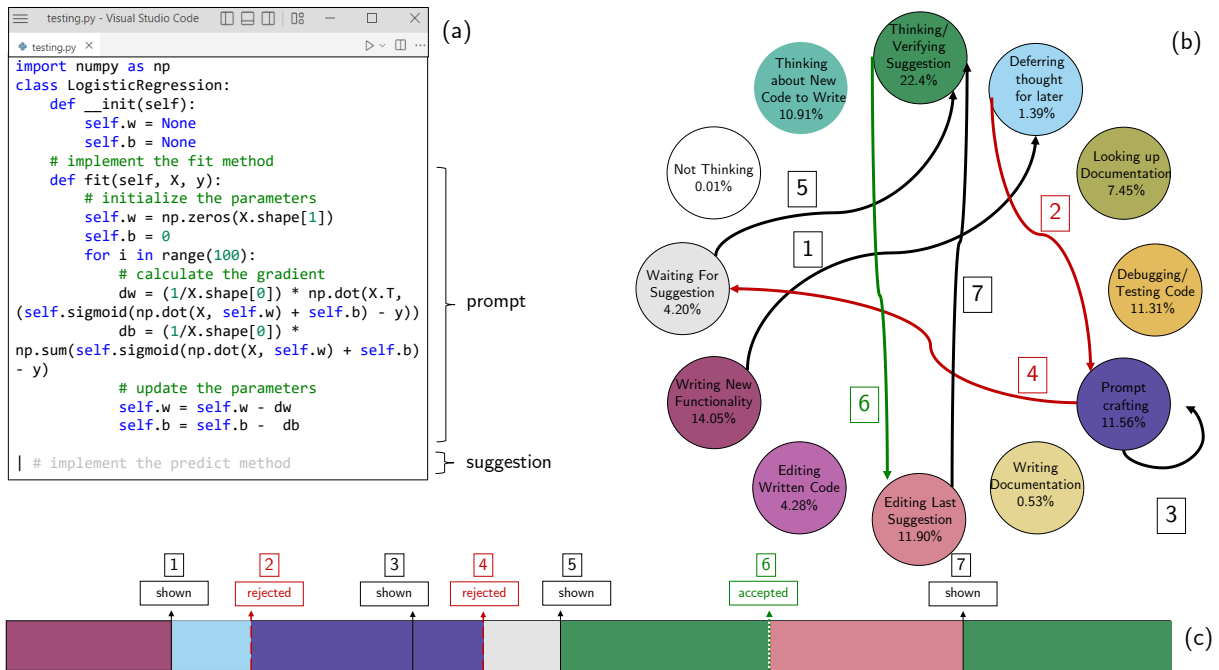


Figure 6.1: Profiling a coding session with the CodeRec User Programming States (CUPS). In (a) we show the operating mode of CodeRec inside Visual Studio Code. In (b) we show the CUPS taxonomy used to describe CodeRec related programmer activities. A coding session can be summarized as a timeline in (c) where the programmer transitions between states.

using this model, with an option to add new states if necessary. The study participants labeled a total of 3137 coding segments and interacted with 1096 suggestions. The study confirmed that the taxonomy was sufficiently expressive, and we further learned transition weights and state dwell times—something we could not do without this experimental setting. Together, these data can be assembled into various instruments, such as the CUPS diagram (Figure 6.1), to facilitate profiling interactions and identify inefficiencies. Moreover, we show that such analysis nearly doubles our estimates for how much developer time can be attributed to interactions with code suggestion systems, as compared with existing metrics. We believe that identifying the current CUPS state during a programming session can help serve programmer needs. This can be accomplished using custom keyboard macros or automated prediction of CUPS states, as discussed in our future work section and the Appendix. Overall, we leverage the CUPS diagram to identify some opportunities to address inefficiencies in the current version of Copilot.

In sum, our main contributions are the following:

- A novel taxonomy of common activities of programmers (called CUPS) when interacting with code recommendation systems (Section 6.4)
- A dataset of coding sessions annotated with user actions, CUPS, and video recordings of programmers coding with Copilot (Section 6.5).
- Analysis of which CUPS states programmers spend their time in when completing coding tasks (Subsection 6.6.1).
- An instrument to analyze programmer behavior (and patterns in behavior) based on a finite-state machine on CUPS states (Subsection 6.6.2).
- An adjustment formula to properly account for how much time do programmers spend verifying CodeRec suggestions (Subsection 6.6.4) inspired by the CUPS state of deferring thought (Subsection 6.6.3).

6.2 Background and Related Work

Large language models based on the Transformer network [175], such as GPT-3 [127], have found numerous applications in natural language processing. Codex [169], a GPT model trained on 54 million GitHub repositories, demonstrates that LLMs can very effectively solve various programming tasks. Specifically, Codex was initially tested on the HumanEval dataset containing 164 programming problems, where it is asked to write the function body from a docstring [169] and achieves 37.7% accuracy with a single generation. Various metrics and datasets have been

proposed to measure the performance of code generation models [5], [168], [176], [177]. However, in each case, these metrics test how well the model can complete code in an offline setting without developer input rather than evaluating how well such recommendations assist programmers in situ. This issue has also been noted in earlier work on non-LLM based code completion models where performance on completion benchmarks overestimates the model's utility to developers [178]. Importantly, however, these results may not hold to LLM-based approaches, which are radically different [179].

One straightforward approach to understanding the utility of neural code completion services, including their propensity to deliver incomplete or imperfect suggestions, is to simply ask developers. To this end, Weisz et al. interviewed developers and found that they did not require a perfect recommendation model for the model to be useful [180]. Likewise, Ziegler et al. surveyed over 2,000 Copilot users [170] and asked about perceived productivity gains using a survey instrument based on the SPACE framework [181] – we incorporate the same survey design for our own study. They found both that developers felt more productive using Copilot and that these self-reported perceptions were reasonably correlated with suggestion acceptance rates. [182] administered a survey to 410 programmers who use various AI programming assistants, including Copilot, and highlighted why the programmers use the AI assistants and numerous usability issues. Similarly, [183] surveyed how introductory programming students utilize Copilot.

While these self-reported measures of utility and preference are promising, we would expect gains to be reflected in objective metrics of productivity. Indeed, one ideal method would be to conduct randomized control trials where one set of participants writes code with a recommendation engine while another set codes without it. GitHub performed such an experiment where 95 participants were split into two groups and asked to write a web server. The study concluded by finding that task completion was reduced by 55.8% in the Copilot condition [172]. Likewise, a study by Google showed that an internal CodeRec model had a 6% reduction in 'coding iteration time' [184]. On the other hand, [171] showed in a study of 24 participants showed no significant improvement in task completion time – yet participants stated a clear preference for Copilot. An interesting comparison to Copilot is Human-Human pair programming, which [185] details.

A significant amount of work has tried to understand the behavior of programmers [186]–[189] using structured user studies under the name of "psychology of programming." This line of work tries to understand the effect of programming tools on the time to solve a task or ease of writing code and how programmers read and write code. Researchers often use telemetry with detailed logging on keystrokes [190], [191] to understand behavior. Moreover, eye-tracking is also used to understand how programmers read code [192], [193]. Our research uses raw telemetry alongside user-labeled states to understand behavior; future research could also utilize eye-tracking and raw video to get deeper insights into behavior.

This wide dispersion of results raises interesting questions about the nature of the utility afforded by neural code completion engines: how, and when, are such systems most helpful; and conversely, when do they add additional overhead? **This is the central question to our work.** The related work closest to answering this question is that of Barke et al. [194], who showed that interaction with Copilot falls into two broad categories: the programmer is either in “acceleration mode” where they know what they want to do, and Copilot serves to make them faster; or they are in “exploration mode”, where they are unsure what code to write and Copilot helps them explore. The taxonomy we present in this chapter, CUPS, enriches this further with granular labels for programmers’ intents. Moreover, the data collected in this work was labeled by the participants themselves rather than by the researchers interpreting their actions, allowing for more faithful intent and activity labeling and the data collected in our study can also be used to build predictive models as in [195]. The next section describes the Copilot system formally and describes the data collected when interacting with Copilot.

6.3 Copilot System Description

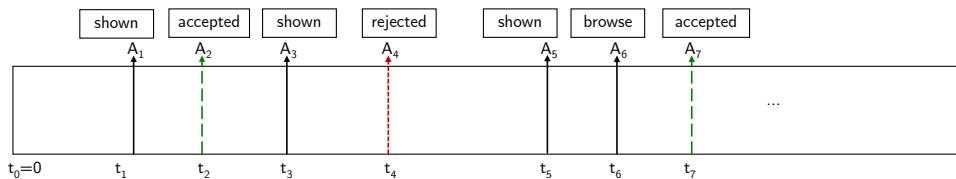


Figure 6.2: Schematic of interaction telemetry with Copilot as a timeline. For a given coding session, the telemetry contains a sequence of timestamps and actions with associated prompt and suggestion features (not shown).

To better understand how code recommendation systems influence the effort of programming, we focus on GitHub Copilot, a popular and representative example of this class of tools. Copilot³ is based on a Large Language Model (LLM) and assists programmers inside an IDE by recommending code suggestions any time the programmer pauses their typing. Figure 6.1 shows an example of Copilot recommending a code snippet as an inline, monochrome popup, which the programmer can accept using a keyboard shortcut (e.g., <tab>).

To serve suggestions, Copilot uses a portion of the code written so far as a *prompt*, P , which it passes to the underlying LLM. The model then generates a suggestion, S , which it deems to be a likely completion. In this regime, programmers can *engineer* the prompt to generate better suggestions by carefully authoring natural language comments in the code such as “# split the

³The version of Copilot that this manuscript refers to is Copilot as of August 2022.

data into train and test sets.” In response to a Copilot suggestion, the programmer can then take one of several actions A , where $A \in \{\text{browse, accept, reject}\}$. The latter of these actions, *reject*, is triggered implicitly by continuing to type something that differs from the suggestion or by pressing the escape key. The browse action enables the programmer to change the suggestion shown with a keyboard shortcut from a set of at most three suggestions. Copilot logs aspects of the interactions via *telemetry*. We leverage this telemetry in the studies described in this chapter. Specifically, whenever a suggestion is shown, accepted, rejected, or browsed, we record a tuple to the telemetry database, (t_i, A_i, P_i, S_i) , where t_i represents the within-session timestamp of the i^{th} event ($t_0 = 0$), A_i details the action taken (augmented to include ‘shown’), and P_i and S_i capture features of the prompt and suggestion, respectively. Figure 7.2 displays telemetry of a coding session, and Figure 6.1a shows Copilot implemented as a VSCode plugin. We have the ability to capture telemetry for any programmer interacting with Copilot; this is used to collect data for a user study.

6.3.1 Influences of CodeRec on Programmer’s Activities

Despite the limited changes that Copilot introduces to an IDE’s repertoire of actions, LLM-based code suggestions can significantly influence how programmers author code. Specifically, Copilot leverages LLMs to stochastically *generate* novel code to fit the arbitrary current context. As such, the suggestions may contain errors (and can appear to be unpredictable) and require that programmers double-check and edit them for correctness. Furthermore, programmers may have to refine the prompts to get the best suggestions. These novel activities associated with the AI system introduce new efforts and potential disruptions to the flow of programming. **We use time as a proxy to study the new costs of interaction introduced by the AI system.** We recognize that this approach is incomplete: the costs associated with solving programming tasks are multi-dimensional, and it can be challenging to assign a single real-valued number to cover all facets of the task [196]. Nevertheless, we argue that, like accuracy, efficiency-capturing measures of time are an important dimension of the cost that is relevant to most programmers.

6.3.2 Programmer Activities in Telemetry Segments

Copilot’s telemetry captures only instantaneous user actions (e.g., accept, reject, browser), as well as the suggestion display event. By themselves, these entries do not reveal such programmer’s activities as double-checking and prompt engineering, as such activities happen *between* two consecutive instantaneous events. **We argue that the regions between events, which we refer to as *telemetry segments*, contain important user intentions and activities unique to programmer-CodeRec interaction**, which we need to understand in order to answer how Copilot affects

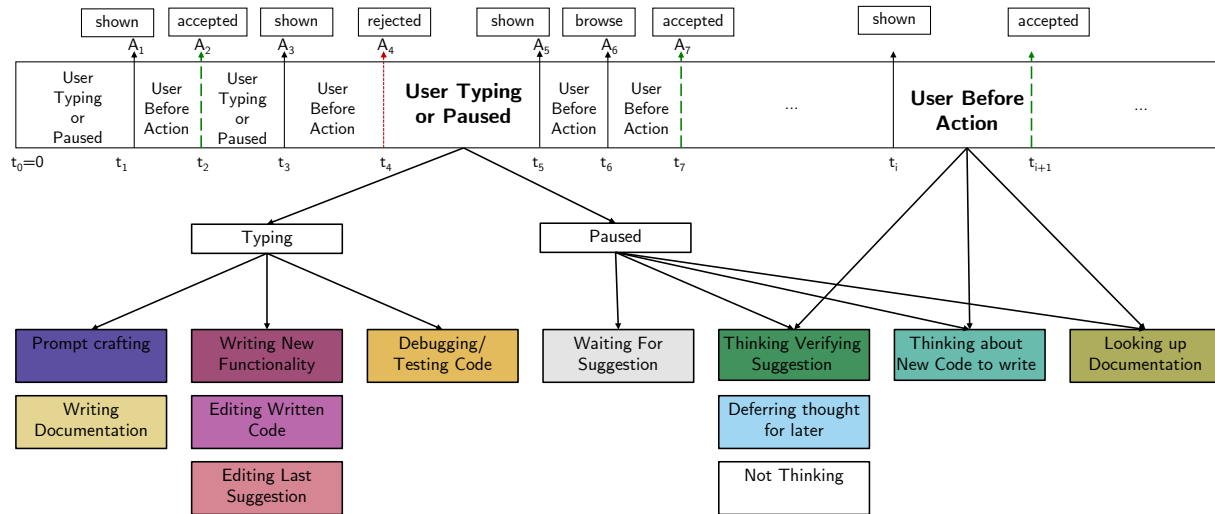


Figure 6.3: Taxonomy of programmer’s activities when interacting with CodeRec– CUPS.

programmers—and where and when Copilot suggestions are useful to programmers.

Building on this idea, telemetry segments can be split into two groups (Figure 7.2). The first group includes segments that start with a suggestion shown event and end with an action (accept, reject, or browse). Here, the programmer is paused and has yet to take action. We refer to this as ‘User Before Action’. The second group includes segments that start with an action event and end with a display event. During this period, the programmer can be either typing or paused; hence we denote it as ‘User Typing or Paused’. These two groups form the foundation of a deeper taxonomy of programmers’ activities, which we will further develop in the next section.

6.4 A Taxonomy for Understanding Programmer-CodeRec Interaction: CUPS

6.4.1 Creating the Taxonomy

Our objective is to create an extensive, but not complete, taxonomy of programmer activities when interacting with CodeRec that enables a useful study of the interaction. To refine the taxonomy of programmers’ activities, we developed a labeling tool and populated it with an initial set of activities based on our own experiences from extensive interactions with Copilot (Figure 6.4). The tool enables users to watch a recently captured screen recording of them solving a programming task with Copilot’s assistance and to *retrospectively* annotate each telemetry segment with an activity label. We use this tool to first refine our taxonomy with a small pilot study (described below) and then to collect data.

Table 6.1: Description of each state in CodeRec User Programming States (CUPS).

State	Description
Thinking/Verifying Suggestion	Actively thinking about and verifying a shown or accepted suggestion
Not Thinking	Not thinking about suggestion or code, programmer away from keyboard
Deferring Thought For Later	Programmer accepts suggestion without completely verifying it, but plans to verify it after
Thinking About New Code To Write	Thinking about what code or functionality to implement and write
Waiting For Suggestion	Waiting for CodeRec suggestion to be shown
Writing New Code	Writing code that implements new functionality
Editing Last Suggestion	Editing the last accepted suggestion
Editing (Personally) Written Code	Editing code written by a programmer that is not a CodeRec suggestion for the purpose of fixing existing functionality
Prompt Crafting	Writing prompt in the form of comment or code to obtain desired CodeRec suggestion
Writing Documentation	Writing comments or docstring for purpose of documentation
Debugging/Testing Code	Running or debugging code to check functionality may include writing tests or debugging statements
Looking up Documentation	Checking an external source for the purpose of understanding code functionality (e.g. Stack Overflow)
Accepted	Accepted a CodeRec suggestion
Rejected	Rejected a CodeRec suggestion

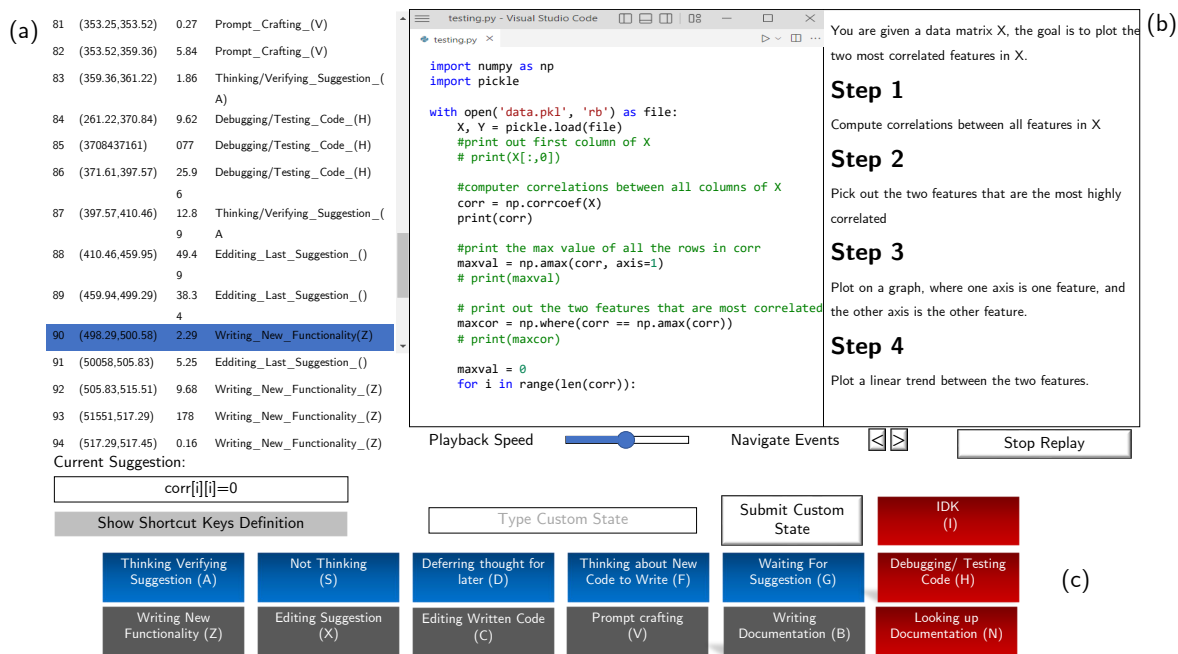


Figure 6.4: Screenshot of retrospective labeling tool for coding sessions. Left: Navigation panel for telemetry segments. Right: Video player for reviewing video of a coding session. Bottom: Buttons and text box for labeling states.

The labeling tool (Figure 6.4) contains three main sections: a) A navigation panel on the left, which displays and allows navigating between telemetry segments and highlights the current segment being labeled in blue. The mouse or arrow keys are used to navigate between segments. b) A video player on the right, which plays the corresponding video segments in a loop. The participant can watch the video segments any number of times. c) Buttons on the bottom corresponding to the CUPS taxonomy, along with an “IDK” button and a free-form text box to write custom state labels. Buttons also have associated keyboard bindings for easy annotation.

To label a particular video segment, we asked participants to consider the hierarchical structure of CUPS in Figure 6.3. The hierarchical structure first distinguishes segments by whether a typing segment occurred in that segment and then decides based on the typing or non-typing states. For example, in a segment where a participant was initially double-checking a suggestion and then wrote new code to accomplish a task, the appropriate label would be "Writing New Functionality" as the user eventually typed in the segment. In cases where there are two states that are appropriate and fall under the same hierarchy, e.g., if the participant double-checked a suggestion and then looked up documentation, they were asked to pick the state in which they spent the majority of the time. These issues arise because we collect a single state for each telemetry segment.

Pilot. Through a series of pilots involving the authors of this work, as well as three other participants drawn from our organization, we iteratively applied the tool to our own coding sessions and to the user study tasks. We then expanded and refined the taxonomy by incorporating any “custom state” (using the text field) written by the pilot participants. The states ‘Debugging/Testing Code’, ‘Looking up Documentation’, and ‘Writing Documentation’ were added through the pilots. By the last pilot participant, the code book was stable and saturated as they did not write a state that was not yet covered. We observed in our study that the custom text field was rarely used. We describe the resultant taxonomy in the sections below.

6.4.2 Taxonomy of Telemetry Segments

Figure 6.3 shows the finalized taxonomy of programmer activities for individual telemetry segments with Copilot. As noted earlier, the taxonomy is rooted in two segment types: ‘User Typing or Paused’, and ‘User Before Action’. We first detail the ‘User Typing or Paused’ segments, which precede shown events (Figure 7.2) and are distinguished by the fact that no suggestions are displayed during this time. As the name implies, users can find themselves in this state if they are either actively ‘Typing’⁴, or have ‘paused’ but have not yet been presented with a suggestion. In cases where the programmer is actively typing, they could be completing any of a number of tasks such as: ‘writing new functionality,’ ‘editing existing code,’ ‘editing prior (CodeRec) suggestions,’ ‘debugging code,’ or authoring natural language comments, including both documentation and prompts directed at CodeRec (i.e., ‘prompt crafting’). When the user pauses, they may simply be “waiting for a suggestion” or can be in any number of states common to ‘User Before Action’ segments.

In every ‘User Before Action’ segment, CodeRec is displaying a suggestion, and the programmer is paused and not typing. They could be reflecting and verifying that suggestion, or they may not be paying attention to the suggestion and thinking about other code to write instead. The programmer can also *defer* their efforts on the suggestion for a later time period by accepting it immediately, then pausing to review the code at a later time. This can occur, for example, because the programmer desires syntax highlighting rather than grey text or because the suggestion is incomplete, and the programmer wants to allow Copilot to complete its implementation before evaluating the code as a cohesive unit. The latter situation tends to arise when Copilot displays code suggestions line by line (e.g., Figure 6.7).

The leaf nodes of the finalized taxonomy represent 12 distinct states that programmers can find themselves in. These states are illustrated in Figure 6.3 and are further described in Table 6.1. While the states are meant to be distinct, siblings may share many traits. For example, “Writing New

⁴Active typing allows for brief pauses between keystrokes.

Functionality" and "Editing Written Code" are conceptually very similar. This taxonomy also bears resemblance to the keystroke level model in that it assigns a time cost to mental processes as well as typing [197], [198]. As evidenced by the user study—which we describe in the next section—these 12 states provide a language that is both *general* enough to capture most activities (at this level of abstraction), and *specific* enough to meaningfully capture activities unique to LLM-based code suggestion systems.

6.5 CUPS Data Collection Study

To study CodeRec-programmer interaction in terms of CodeRec User Programming States, we designed a user study where programmers perform a coding task, then review and label videos of their coding session using the telemetry segment-labeling tool described earlier. We describe the procedure, the participants, and the results in the sections that follow.

6.5.1 Procedure

We conducted the study over a video call and asked participants to use a remote desktop application to access a virtual machine (VM). Upon connecting, participants were greeted with the study environment consisting of Windows 10, together with Visual Studio Code (VS Code) augmented with the Copilot plugin.

Participants were then presented with a programming task drawn randomly from a set of eight pre-selected tasks (Table 6.2). If the participant was unfamiliar with the task content, we offered them a different random task. The task set was designed during the pilot phase so that individual tasks fit within a 20-minute block and so that, together, the collection of tasks surfaces a sufficient diversity of programmer activities. It is crucial that the task is of reasonable duration so that participants are able to remember all their activities since they will be required to label their session immediately afterward. Since the CUPS taxonomy includes states of thought, participants must label their session immediately after coding, and each study took approximately 60 minutes in total. To further improve diversity, task instructions were presented to participants as images to encourage participants to author their own Copilot prompts rather than copying and pasting from the problem description. The full set of tasks and instructions is provided as an Appendix.

Upon completing the task (or reaching the 20-minute mark), we loaded the participant's screen recording and telemetry into the labeling tool (previously detailed in Section 6.4.1). The researcher then briefly demonstrated the correct operation of the tool and explained the CUPS taxonomy. Participants were then asked to annotate their coding session with CUPS labels. Self-labeling allows us to easily scale such a study and enables more accurate labels for each participant, but may

cause inconsistent labeling across participants. Critically, this labeling occurred within minutes of completing the programming task so as to ensure accurate recall. We do not include a baseline condition where participants perform the coding task without Copilot, as this work focuses on understanding and modeling the interaction with the current version of Copilot.

Finally, participants completed a post-study questionnaire about their experience mimicking the one in [170]. The entire experiment was designed to last 60 minutes. The study was approved by our institutional review board (IRB), and participants received a \$50.00 gift card as remuneration for their participation.

Table 6.2: Description of the coding tasks given to user study participants and task assignment. Participants were randomly allocated to tasks for tasks which they had familiarity with.

Task Name	Participants	Description
Algorithmic Problem	P4,P17,P18	Implementation of TwoSum, ThreeSum and FourSum
Data Manipulation	P1,P2,P11,P20	Imputing data with average feature value and feature engineering for quadratic terms
Data Analysis	P5,P8	Computing data correlations in a matrix and plotting of most highly correlated features
Machine Learning	P3,P7,P12,P15	Training and Evaluation of models using sklearn on given dataset
Classes and Boilerplate Code	P6,P9	Creating different classes that build on each other
Writing Tests	P16	Writing tests for a black box function that checks if a string has valid formatting
Editing Code	P10,P14,P21	Adding functionality to an existing class that implements a nearest neighbor retriever
Logistic Regression	P13,P19	Implementing a custom Logistic Regression from scratch with weight regularization

6.5.2 Participants

To recruit participants, we posted invitations to developer-focused email distribution lists within our large organization. We recruited 21 participants with varying degrees of experience using Copilot: 7 used Copilot more than a few times a week, 3 used it once a month or less, and 11 had never used it before. For participants who had never used it before, the experimenter gave a short oral tutorial on Copilot explaining how it can be invoked and how to accept suggestions. Participants’ roles in the organization ranged from software engineers (with different levels of seniority) to researchers and graduate student interns. In terms of programming expertise, only 6 participants had less than 2 years of professional programming experience (i.e., excluding years spent learning to program), 5 had between 3 to 5 years, 7 had between 6 to 10 years, and 3 had more than 11 years of experience. Participants used a language in which they stated proficiency (defined as language in which they were comfortable designing and implementing whole programs). Here, 19 of the 21 participants used Python, one used C++, and the final participant used JavaScript.

On average, participants took 12.23 minutes (sample standard deviation, $s_N = 3.98$ minutes) to complete the coding task, with a maximum session length of 20.80 minutes. This task completion time is measured from the first line of code written for the task until the end of the allocated time. During the coding tasks, Copilot showed participants a total of 1024 suggestions, out of which

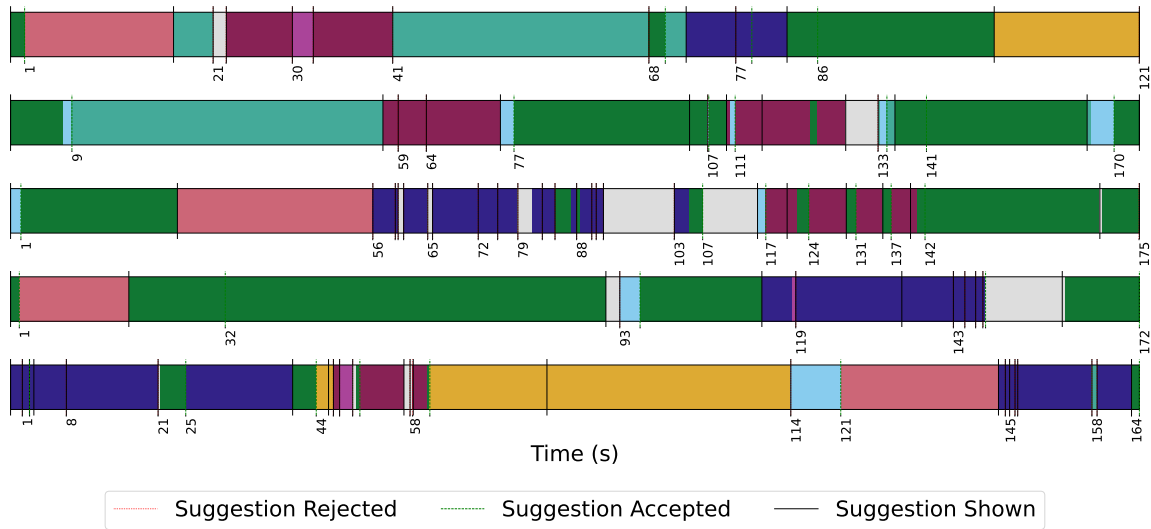
they accepted 34.0%. The average acceptance rate for participants was 36.5% (averaging over the acceptance rate of each participant), and the median was 33.8% with a standard error of 11.9%; the minimum acceptance rate was 14.3%, and the maximum was 60.7%. In the labeling phase, each participant labeled an average of 149.38 ($s_N = 57.43$) segments with CUPS, resulting in a total of 3137 CUPS labels. The participants used the ‘custom state’ text field only three times total, twice a participant wrote ‘write a few letters and expect suggestion’ which can be considered as ‘prompt crafting’ and once a participant wrote ‘I was expecting the function skeleton to show up[..]’ which was mapped to ‘waiting for suggestion’. The IDK button was used a total of 353 times, this sums to 3137 CUPS + 353 IDKs = 3490 labels, the majority of its use was from two participants (244 times) where the video recording was not clear enough during consecutive spans, and was used by only five other participants more than once with the majority of the use also being due to the video not being clear or the segment being too short. The IDK segments represent 6.5% of total session time across all participants, mostly contributed by five participants. Therefore, we remove the IDK segments from the analysis and do not attempt to re-label them. Note that while the "Not Thinking" state is part of the CUPS taxonomy, we do not observe any occurrence of the state in the user study. This is because participants were being observed while coding and did not have any interruptions that may occur during real-life coding, such as checking emails or drinking coffee. The activities outside of coding are represented by the not thinking state.

Together, these CUPS labels enable us to investigate various questions about programmer-CodeRec interaction systematically, such as exploring which activities programmers perform most frequently and how they spend most of their time. We study programmer-CodeRec interaction using the data derived from this study in the following Section 6.6 and derive various insights and interventions.

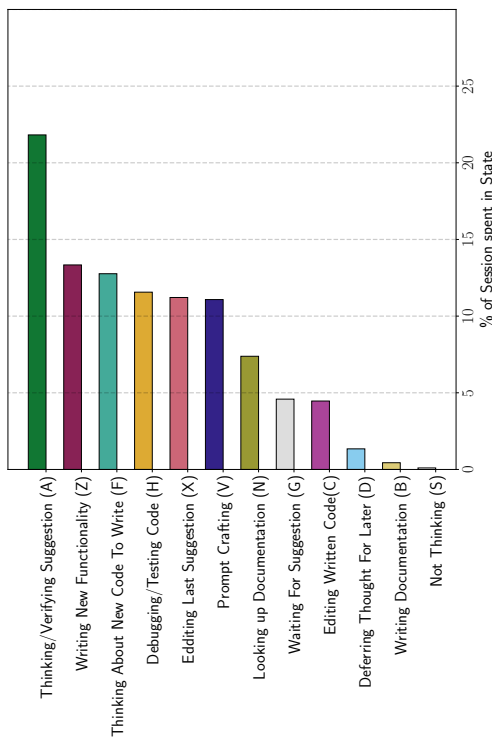
6.6 Understanding Programmer Behavior with CUPS: Main Results

The study in the previous section allows us to collect telemetry with CUPS labels for each telemetry segment. We now analyze the collected data and highlight suggestions for 1) **metrics** to measure the programmer-CodeRec interaction, 2) **design improvements** for the Copilot interface, and finally 3) **insights** into programmer behavior. Each subsection below presents a specific result or analysis which can be read independently. Code and Data is available at ⁵.

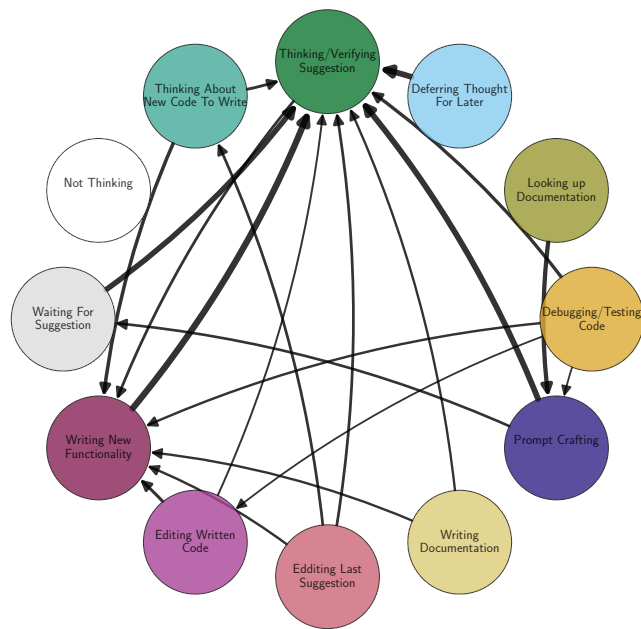
⁵https://github.com/microsoft/coderec_programming_states



(a) Individual CUPS timelines for 5/21 study participants for the first 180 secs show the richness of and variance in programmer-CodeRec interaction.



(b) The percentage of total session time spent in each state during a coding session. On average, verifying Copilot suggestions occupies a large portion of session time.



(c) CUPS diagram showing 12 CUPS states (nodes) and the transitions among the states (arcs). Transitions occur when a suggestion is shown, accepted, or rejected. We hide self-transitions and low-probability transitions for simplicity

Figure 6.5: Visualization of CUPS labels from our study as timelines, a histogram, and a state machine.

6.6.1 Aggregated Time Spent in Various CUPSs

In Figure 6.5a, we visualize the coding sessions of *individual* participants as *CUPS timelines*, where each telemetry segment is labeled with its CUPS label. At first glance, CUPS timelines show the richness in patterns of interaction with Copilot, as well as the variance in usage patterns across settings and people. CUPS timelines allow us to inspect individual behaviors and identify patterns, which we later aggregate to form general insights into user behavior.

Figure 6.5b shows the average time spent in each state as a percentage normalized to a user's session duration.

Metric Suggestion: Time spent in CUPS states as a high-level diagnosis of the interaction

For example, time spent 'Waiting For Suggestion' (4.2%, $s_N = 4.46$) measures the real impact of **latency**, and time spent 'Editing Last Suggestion' provides feedback on the quality of suggestions.

We find that averaged across all users, the **'verifying suggestion' state takes up the most time** at 22.4% ($s_N = 12.97$), it is the top state for 6 participants and in the top 3 states for 14 out of 21 participants taking up at least 10% of session time for all but one participant. Notably, this is a new programmer task introduced by Copilot. The second-longest state is writing new functionality' 14.05% ($s_N = 8.36$), all but 6 participants spend more than 9% of session time in this state.

More generally, the states that are specific to interaction with Copilot include: 'Verifying Suggestions', 'Deferring Thought for Later', 'Waiting for Suggestion', 'Prompt Crafting', and 'Editing Suggestion'. **We found that the total time participants spend in these states is 51.5 %** ($s_N = 19.3$) **of the average session duration**. In fact, half of the participants spend more than 47% of their session in these Copilot states, and all participants spend more than 21% of their time in these states.

By Programmer Expertise and Copilot Experience. We investigate if there are any differences in how programmers interacted with Copilot based on their programming expertise and their previous experience with Copilot. First, we split participants based on whether they have professional programming experience of more than 6 years (10 out of 21) and who have less than 6 years (11 out of 21). We notice the acceptance rate for those with substantial programming experience is $30.0\% \pm 14.5$ while for those without is $37.6\% \pm 14.6$. Second, we split participants based on whether they had used Copilot previously (10 out of 21) and those who had never used it before (11 out of 21). The acceptance rate for those who have previously used Copilot is $37.6\% \pm 15.3$, and for those who have not, it is 29.3 ± 13.7 . Due to the limited number of participants, these results are not sufficient to determine the influence of programmer experience or Copilot experience on

behavior. We also include in Appendix a breakdown of programmer behavior by task solved.

6.6.2 Patterns in Behavior as Transitions Between CUPS States

To understand if there was a pattern in participant behavior, we modeled *transitions* between two states as a *state machine*. We refer to the state machine-based model of programmer behavior as a *CUPS diagram*. In contrast to the timelines in Figure 6.5a, which visualize state transitions with changes of colors, the CUPS diagram Figure 6.5c explicitly visualizes transitions using directed edges, where the thickness of arrows is proportional to the likelihood of transition. For simplicity, Figure 6.5c only shows transitions with an average probability higher than 0.17 (90th quantile, selected for graph visibility).

The transitions in Figure 6.5c revealed many expected patterns. For example, one of the most likely transitions (excluding self-transitions from the diagram), ‘Prompt Crafting $\xrightarrow{0.54}$ Verifying Suggestion’ showed that when programmers were engineering prompts, they were then likely to immediately transition to verifying the resultant suggestions (probability of 0.54). Likewise, Another probable transition was ‘Deferring Thought $\xrightarrow{0.54}$ Verifying Suggestion’, indicating that **if a programmer previously deferred their thought for an accepted suggestion, they would, with high probability, return to verify that suggestion**. Stated differently: deference incurs verification debt, and this debt often “catches up” with the programmer. Finally, the single-most probable transition, ‘Writing New Functionality $\xrightarrow{0.59}$ Verifying Suggestion’, echos the observation from the previous section, indicating that programmers often see suggestions while writing code (rather than prompt crafting), then spend time verifying it. If suggestions are unhelpful, they could easily be seen as interrupting the flow of writing.

The CUPS diagram also revealed some unexpected transitions. Notably, the second-most probable transition from the ‘Prompt Crafting’ state is ‘Prompt Crafting $\xrightarrow{0.25}$ Waiting for Suggestion’. This potentially reveals an unexpected and unnecessary delay and is a possible target for refinement (e.g., by reducing latency in Copilot). Importantly, each of these transitions occurs with a probability that is much higher than the lower bound/uniform baseline probability of transitioning to a random state in the CUPS diagram ($1/12=0.083$). In fact, when we compute the entropy rate (a measure of randomness) of the resulting Markov Chain [199] from the CUPS diagram we obtain a rate of 2.24; if the transitions were completely random the rate would be 3.58, and if the transitions were deterministic then the rate is 0.

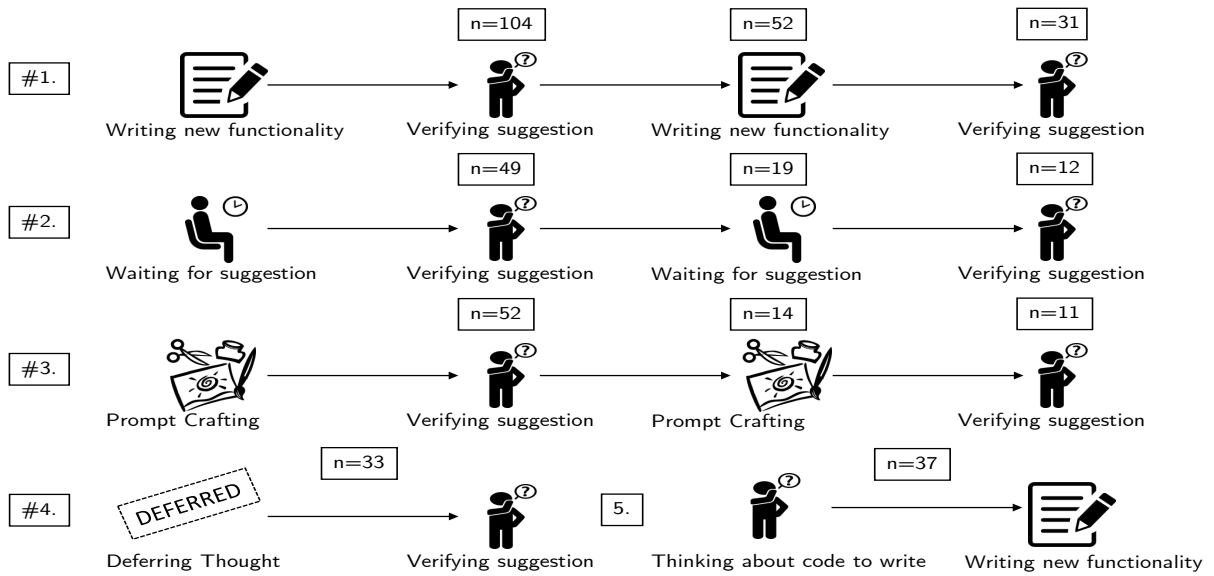
Interface Design Suggestion: Identifying current CUPS state can help serve programmer needs

If we are able to know the current programmer CUPS state during a coding session we can better serve the programmer, for example,

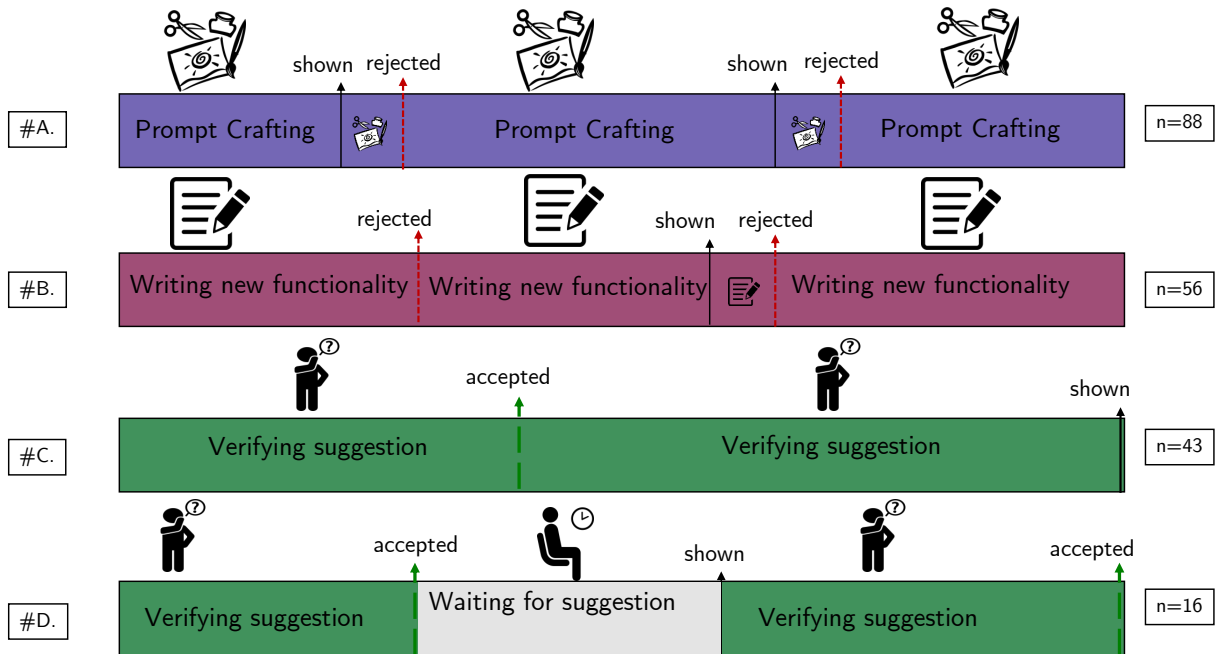
- If the programmer is observed to have been deferring their thought on the last few suggestions, group successive Copilot suggestions and display them together.
- If the programmer is waiting for the suggestion, we can prioritize resources for them at that moment
- While a user is prompt crafting, Copilot suggestions are often ignored and may be distracting; however, after a user is done with their prompt, they may expect high-quality suggestions. We could suppress suggestions during prompt crafting, but after the prompt crafting process is done, display multiple suggestions to the user and encourage them to browse through them.

Future work can, for example, realize these design suggestions by allowing **custom keyboard macros** for the programmer to signal their current CUPS state, or a more automated approach by **predicting** their CUPS state.

We also investigated longer patterns in state transitions by searching for the most common sequence of states of varying lengths. We achieved this by searching over all possible segment n-grams and counting their occurrence over all sessions. We analyzed patterns in two ways: in Figure 6.6a, we merged consecutive segments that have the same state label into a single state (thus removing self-transitions), and in Figure 6.6b we looked at n-grams in the user timelines (including self-transitions) where we include both states and participants actions (shown, accepted and rejected). The most common pattern (#1) in Figure 6.6a was a cycle where programmers repeatedly wrote new code functionality and then spent time verifying shown suggestions, indicating a new mode for programmers to solve coding tasks. At the same time, when we look at pattern (#B) in Figure 6.6b, which takes a closer look into when programmers are writing new functionality, we observe that they don't stop to verify suggestions and reject them as they continue to write. Other long patterns include (#2) (also shown as pattern #D), where programmers repeatedly accepted successive Copilot suggestions after verifying each of them. Finally, we observe in (#3) and (#A) programmers iterating on the prompt for Copilot until they obtain the suggestion they want. We elaborate more on this in the next subsection.



(a) Common patterns of transitions between *distinct* states. In individual participant timelines, the patterns visually appear as a change of color, but here we measure how often they appear across all participants (n=).



(b) Common patterns of states and actions (including self transitions). Each pattern is extracted from user timelines and we count how often it appears in total (n=)

Figure 6.6: Myriad of CUPS patterns observed in our study.

6.6.3 Programmers Often Defer Thought About Suggestions

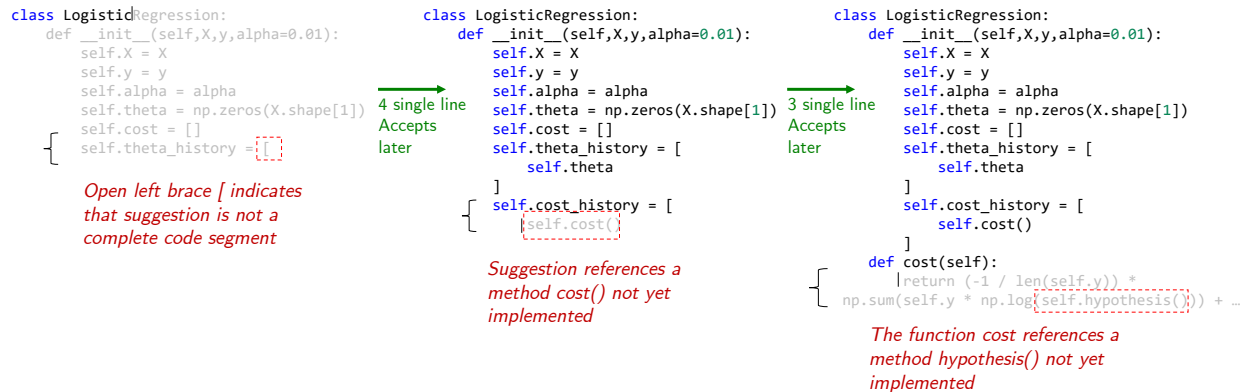


Figure 6.7: Illustration of a coding scenario with Copilot where the programmer may choose to defer verifying a suggestion (‘Deferring Thought’). Here, Copilot suggests an implementation for the class Logistic Regression line-by-line (illustrated from left to right). And the programmer may need to defer verifying intermediate suggestion of `self.cost` (middle screenshot) because the method that implemented it is suggested later (right screenshot).

An interesting CUPS state is that of ‘Deferring Thought About A Suggestion’. This is illustrated in Figure 6.7, where programmers accept a suggestion or series of suggestions without sufficiently verifying them beforehand. This occurs either because programmers wish to see the suggestion with code highlighting, or because they want to see where Copilot suggestions leads to. Figure 6.5b shows that programmers do in fact, frequently defer thought— we counted 61 states labeled as such. What drives the programmer to defer their thought about a suggestion rather than immediately verifying it? We initially conjectured that the act of deferring may be partially explained by the length of the suggestions. So, we compared the number of characters and the number of lines for suggestions depending on the programmer’s state. We find that there is no statistical difference according to a two-sample independent t-test ($t = -0.58, p = 0.56$)⁶ in the average number of characters between deferred thought and suggestions (75.81 compared to 69.06) that were verified previously. The same holds for the average number of lines.

However, when we look at the likelihood of editing an accepted suggestion, we find that it is 0.18 if it was verified before, but it is 0.53 if it was deferred. This difference is significant according to a chi-square test ($\chi^2 = 29.2, p = 0$). In fact, the programmer CUPS state has a big effect on their future actions. In Table 6.3, we show the probability of the programmer accepting a suggestion given the CUPS state the programmer was in while the suggestion is being shown. We also show the probability of the programmer accepting a suggestion as a function of the CUPS state the programmer was in just before the suggestion was displayed. We observe there is a big variation

⁶All p-values reported are corrected for multiple hypothesis testing with the Benjamin/Hochberg procedure with $\alpha = 0.05$.

in the suggestion acceptance rate by the CUPS state. For example, if the programmer was in the "Deferring Thought For Later" state, the probability of acceptance is 0.98 ± 0.02 compared to when a programmer is thinking about new code to write, where the probability is 0.12 ± 0.04 . Note that the average probability of accepting a suggestion was 0.34.

What are the programmers doing before they accept a suggestion? We found that the average probability of accepting a suggestion was 0.34. However, we observed that when the programmer was verifying a suggestion their likelihood of accepting was 0.70. In contrast, if the programmer was thinking about new code to write, the probability dropped to 0.20. This difference was statistically significant according to Pearson's chi-squared test ($\chi^2 = 12.25, p = 0$). Conversely, when programmers are engineering prompts, the likelihood of accepting a suggestion drops to 0.16. One reason for this might be that programmers want to write the prompt on their own without suggestions, and Copilot interrupts them. We show the full results in the Appendix for the other states.

Table 6.3: We compute the percentage of suggestions accepted given the programmer was in the CUPS state while the suggestion is being shown (% Ss accepted while shown). We compute the percentage of suggestions accepted given the programmer was in the CUPS state before the suggestion is shown, the state just before the one where the suggestion is shown (% Ss accepted before S is shown). We compute the standard error for the acceptance rate (%).

State	% Ss accepted while shown	% Ss accepted before S is shown
Thinking/Verifying Suggestion	0.80 ± 0.02	0.56 ± 0.04
Prompt Crafting	0.11 ± 0.02	0.22 ± 0.03
Looking up Documentation	0.00 ± 0.00	0.29 ± 0.17
Writing New Functionality	0.07 ± 0.02	0.31 ± 0.03
Thinking About New Code To Write	0.12 ± 0.04	0.27 ± 0.04
Editing Last Suggestion	0.03 ± 0.03	0.23 ± 0.05
Waiting For Suggestion	0.10 ± 0.05	0.58 ± 0.06
Editing Written Code	0.07 ± 0.04	0.17 ± 0.07
Writing Documentation	0.40 ± 0.22	0.33 ± 0.19
Debugging/Testing Code	0.23 ± 0.07	0.26 ± 0.06
Deferring Thought For Later	0.98 ± 0.02	1.0 ± 0.0

6.6.4 CUPS Attributes Significantly More Time Verifying Suggestions than Simpler Metrics

We observed that programmers continued verifying the suggestions after they accepted them. This happens by definition for 'deferred thought' states before accepting suggestions, but we find it also happens when programmers verify the suggestion before accepting it and this leads to a

significant increase in the total time verifying suggestions. First, when participants defer their thought about a suggestion they accepted, 53.2% of the time they verify the suggestion immediately afterward. When we adjust for the post-hoc time spent verifying, we compute a mean time of 15.21 ($s_N = 20.68$) seconds of verification and a median time of 6.48s. This is nearly a five-times increase in average time and a three-time increase in median time for the pre-adjustment scores of 3.25 ($s_N = 3.33$) mean and 1.99 median time. These results are illustrated in Figure 6.8 and is a statistically significant increase according to a two-sample paired t-test ($t = -4.88, p = 1.33 \cdot 10^{-5}$). This phenomenon also occurs when programmers are in a 'Thinking/Verifying Suggestion' state before accepting a suggestion where 19% of the time they posthoc verify the suggestion which increases total verification time from 3.96 ($s_N = 8.63$) to 7.03 ($s_N = 14.43$) on average which is statistically significant ($t = -4.17, p = 5e - 5$). On the other hand, programmers often have to wait for suggestions to show up due to either latency or Copilot not kicking in to provide a suggestion. If we sum the time between when a suggestion is shown and the programmer accepts or rejects this in addition to the time they spend waiting for the suggestion (this is indicated in the state 'Waiting for suggestion'), then we get an increase from 6.11s ($s_N = 15.52$) to 6.51s ($s_N = 15.61$) which is minor on average but adds 2.5 seconds of delay when programmers have to explicitly wait for suggestions.

Metric Suggestion: Adjust verification time metrics and acceptance rates to include suggestions that are verified after acceptance

The previous analysis showed that the time to accept a suggestion cannot be simply measured as the time spent from the instance a suggestion is shown until a suggestion is accepted— this misses the time programmers spend verifying a suggestion after acceptance. Similarly, since deferring thought is a frequent behavior observed, it leads to an inflation of acceptance rates. We recommend using measures such as the fraction of suggestions accepted that survive in the codebase after a certain time period (e.g. 10 minutes).

6.6.5 Insights About Prompt Crafting

Insights about Prompt Crafting. We take a closer look into how participants craft prompts to obtain Copilot suggestions. *Our first insight is that programmers consistently ignore suggestions while prompt crafting. Among 234 suggestions that were shown while participants were actively prompt crafting, defined as a suggestion where a programmer was prompt crafting while the suggestion was being displayed, only 10.7% were accepted.* We hypothesize this behavior could be due to programmers wanting to craft the prompt in their own language rather than relying on

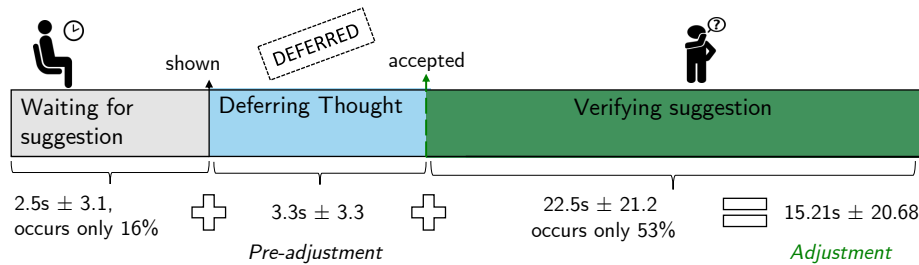


Figure 6.8: Illustration of one of the adjustments required for measuring the total time a programmer spends to verify a suggestion. Here, when a programmer defers thought for a suggestion, they spend time verifying it after accepting it and may also have to wait beforehand for the suggestion to be shown.

Copilot to help them prompt craft. This also indicates that Copilot is unnecessarily interrupting participants' prompt crafting attempts.

However, programmers often iterate on their prompts until they obtain the suggestion they desire and often do not abandon prompt crafting without accepting a suggestion. We define a prompt crafting attempt as a segment of the coding session that starts from when the programmer first enters the CUPS "prompt crafting" state and lasts until the programmer enters a non-Copilot centric state⁷. We count 59 such prompt crafting attempts wherein 81.3% of them a suggestion is accepted.

Prompt crafting is often an iterative process, where the programmer writes an initial prompt, observes the resulting suggestion, then iterates on the prompt by adding additional information about the desired code or by rewording the prompt. For example, P5 wanted to retrieve the index of the maximum element in a correlation matrix and wrote this initial prompt and got the suggestion:

```
# print the indices of the max value excluding 1 in corr
maxval = np.amax(corr, axis=1) # Copilot suggestion
```

This code snippet returns the value of the maximum value rather than the index, so it was not accepted by the participant. They then re-wrote the prompt to be:

```
# print the two features most correlated
# Copilot suggestion
maxcor = np.where(corr == np.amax(corr))
```

and accepted the above suggestion.

Finally, we observe that there are three main ways participants craft prompts:

1) through writing a single line comment with natural language instructions, although the comment may resemble pseudo-code [173], an example:

⁷The non-Copilot centric states are: 'Writing New Functionality,' 'Editing Written Code,' 'Thinking About New Code To Write,' 'Debugging/Testing Code,' 'Looking up Documentation,' 'Writing Documentation.'

```
# impute missing values in X_train as average of column
# where missing value is -1
```

2) through writing a docstring for the function:

```
def distance(self, query):
    '''
    query: single numpy array
    return: 12 distances from query to the vectors
    '''
```

and finally, 3) through writing function signatures (or variable names) e.g., writing "def add_time" then pausing to wait for a suggestion. Often, programmers combine the three prompt crafting strategies to get better code suggestions.

6.6.6 Post-Study Survey Answers

After completing the study, participants were asked to complete a survey based on the productivity survey in [170], which focuses on the SPACE framework of programmer productivity [181]. We also included a free-form text box at the end of the survey where participants can add any additional thoughts about their experience using Copilot for the task assigned. The full results of the survey can be found in the Appendix.

We found that 6/21 participants agreed or strongly agreed with the statement that they were concerned about the quality of their code when using Copilot. Participant #9 noted, "I worry that bugs can sneak-in and go unnoticed, especially in weakly-dynamically typed languages" and Participant #19 noted that "My main concern with Copilot is whether it is teaching me to do things the wrong (or old) way (e.g. showing me a Python 3.6 way instead of a Python 3.10 way and so on)". On the other hand, 14/21 participants agreed that using Copilot helped them stay in flow and spend less time searching for information. Participant #3 noted that "Collaborating with Copilot felt like I was googling what I wanted to do except instead of going through several stack overflow links that Google would show me, the code just appeared inline saving me time and keeping my flow of coding" and Participant #6 "Going into the exercise I genuinely thought there would be a point when I pull up stack overflow. Because that's the kind of tiny stuff you sometimes need to search for. With copilot, it really reduced my worry of doing so." Finally, 17/21 participants agreed with the statement that by using Copilot, they completed the task faster, and 16/21 participants agreed that they were more productive using Copilot. These survey responses highlight the costs and benefits of writing code with Copilot and reinforcing existing results in [170].

6.7 Limitations

The observations from our study are limited by several decisions that we made. First, our participants solved time-limited coding tasks that were provided by us instead of real tasks they may perform in the real world. Furthermore, the selection of tasks was limited and did not cover all tasks programmers might perform. We mostly conducted experiments with Python with only two participants using C++ and JavaScript when Copilot is capable of completing suggestions for myriads of other languages. We also made an assumption about the granularity of telemetry where each segment at most contained one state when, in a more general setting, programmers may perform multiple activities within a single segment. We also did not capture longer-term costs of interacting, e.g., from accepting code with security vulnerabilities or longer horizon costs. To this end, security vulnerabilities and possible overreliance issues [200]–[202], are important areas of research that we do not address in this chapter.

Chapter 7

When to Show a Suggestion? Integrating Human Feedback in AI-Assisted Programming

Acknowledgements of Co-authors. This chapter is based on the published work in [203]. I would like to thank my co-authors, Gagan Bansal, Adam Fourney, and Eric Horvitz, for their help.

7.1 Introduction

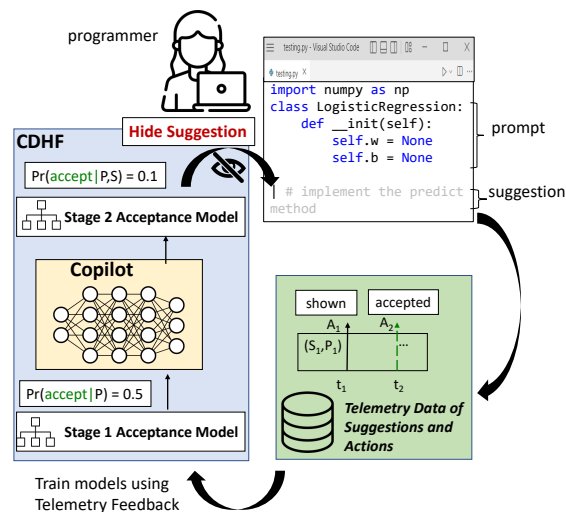


Figure 7.1: Operating mode of CodeRec inside Visual Studio Code showing how CDHF influences the interaction by selectively hiding certain suggestions. Data collected by the interaction is stored in telemetry and used to train CDHF to create a feedback loop.

Code-recommendation systems are powered by large language models (LLMs) such as GPT

that are trained on standard language modeling objectives using the Common Crawl data [204], and then fine-tuned on public code repositories [169]. The public roll-out of the code recommendation models has attracted millions of programmers, enabling a unique opportunity to leverage the data of programmers interacting with the models. In this work, we study GitHub’s Copilot which is used by millions of programmers [7]. For a set of programmers within our organization who consented to have their usage data collected, we collected telemetry data of Copilot suggestions, along with their associated prompts and the programmer’s action to accept or reject the suggestions. We leverage this telemetry data to design mechanisms and interventions that can improve the interaction between programmers and CodeRec.

Specifically, we seek to identify *when* to show a code suggestion. We first define the expected utility of a displaying a suggestion, a value that measures the impact of showing a suggestion on the overall time to write a specific piece of code. This value provides an optimal criterion for when to show a suggestion. However, computing the utility of suggestions is difficult and not currently feasible. Instead, we rely on the result that suggestion utility increases the more likely a suggestion is to be accepted and decreases with increasing latency to generate a suggestion—two quantities we can reliably estimate and control, respectively. We develop a procedure, named **conditional suggestion display from human feedback (CDHF)** which guides whether to show or hide suggestions. At each pause in keystrokes, CDHF decides whether if it is worthwhile to generate a suggestion and if the programmer is likely to accept the generated suggestion. CDHF employs a cascade of models that predict acceptance of suggestions. The optimization procedure guarantees that any suggestion that was hidden (or not generated) would have been rejected if it was shown with a probability of at least p , where, e.g., p can be 0.99.

Using data from programming sessions of 535 programmers with feedback on 168k suggestions, we perform a retrospective evaluation of CDHF. We show that we can hide 25% of suggestions that were shown while guaranteeing that 95% of them would have been rejected. Further, we avoid generating 13% of these suggestions. The results show that CDHF would increase the acceptance rate by 7.2%. The procedure allows for controlling a trade-off that balances the number of suggestions that are displayed with increases in latency, controlled with a parameter that halts generations. We note that a minimal version of CDHF has been implemented in a newer version of GitHub Copilot [205] following the presentation of earlier versions of our work to GitHub. Our work provides a roadmap for building and fielding better forms of suggestion display.

Beyond decisions about displaying recommendations, we examine the feasibility of using suggestion acceptance as a reward signal to select *which* suggestions to display and show how partial completions can be prioritized over the generations of complete code segments. While we investigate Copilot in this work, we believe our insights extend to other AI models and non-code-based tasks.

7.2 Related Work

The closest related work to ours is the procedure to selectively hide suggestions in [206] (quality estimation before completion, QEBC). In distinction to this work, QEBC [206] is not based on human feedback of accepting suggestions but rather is based on constructing a learned estimator of the quality of code completions from datasets of paired code segments and model completions. Our CDHF estimator uses real programmer behavior data and is based on data from a code-recommendation system in current use (Copilot) as opposed to custom-trained ones in [206]. Different metrics and datasets have been proposed to evaluate the performance of code recommendation models, but these typically assess how well the model can complete code in an offline setting without developer input rather than evaluating how well it assists programmers in situ [5], [168], [170], [177]. Integrating human preferences when training machine learning models has long been studied in the literature [207], [208]. In particular, reinforcement learning from human feedback (RLHF) has been used to improve LLMs used as conversational chatbots [209], [210], notably ChatGPT [211]. In contrast, CDHF uses human feedback collected organically through telemetry. The objective is fast inference to reduce latency and hiding suggestions rather than updating the LLM. Further related work can be found in the appendix. Our theoretical formulations build on earlier work on harnessing machine learning and utility to guide AI versus human-powered contributions in human-AI interactive settings [212], which we apply to our setting.

7.3 Problem Setting

CodeRec. We consider CodeRec, which is a commonly used and exemplary tool of AI-powered code recommendations used by millions of programmers [7]. CodeRec is powered by a large language model (LLM) to provide code suggestions to programmers within an IDE whenever the programmer pauses their typing. An illustration of CodeRec suggesting code as an inline, single-colored snippet is displayed in Figure 7.1. The programmer can choose to accept this suggestion via a keyboard shortcut (e.g., tab).

AI-Assisted Programming. We attempt a mathematical formalization of programming with the help of a code recommendation model such as CodeRec, which we dub *AI-Assisted Programming*. The programmer wishes to complete a certain task T , for example, to implement a logistic regression classifier. As the programmer writes code starting from time 0, CodeRec attempts to provide code suggestions at different times. At a given time t , CodeRec¹ uses a portion of the code X_t to generate a prompt P_t , which is passed to the underlying LLM. CodeRec then generates a code suggestion

¹We discuss implementation details of Copilot at a high level; our work is based on the August 2022 version of Copilot.

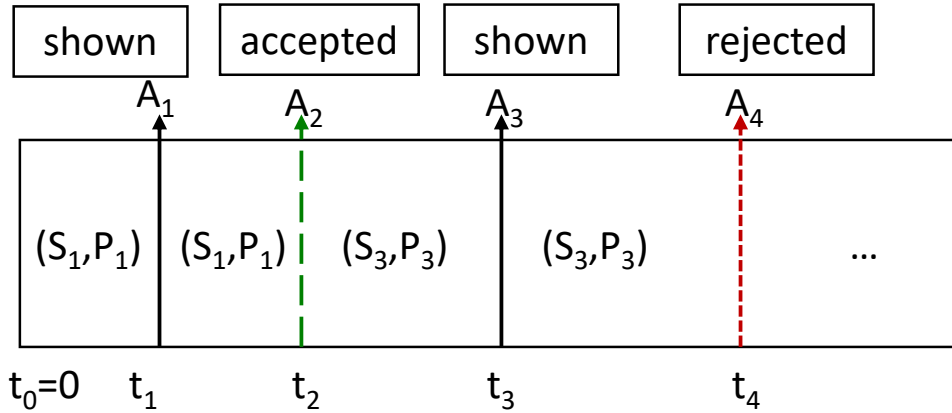


Figure 7.2: Schematic of telemetry with CodeRec as a timeline. For a given coding session, the telemetry contains a sequence of timestamps and actions with associated prompts and suggestions.

S_t , which is shown to the user at time $t + \tau$ where τ accounts for the LLM latency. Once the suggestion is shown, the programmer must make an action at a certain time $t' > t + \tau$, the action is $A_{t'} \in \{ \text{accept}, \text{reject} \}$; the *reject* action is triggered implicitly by continuing to type.

Telemetry. CodeRec logs aspects of the interactions via telemetry, which we leverage in our study. We refer to event positions drawn from a discretization of times spanning a session. Specifically, whenever a suggestion is shown, accepted or rejected, we record a tuple to the telemetry database, (t_i, A_i, P_i, S_i) , where t_i represents the within-session timestamp of the i^{th} event ($t_0 = 0$), A_i details the action taken (augmented to include ‘shown’), and P_i and S_i capture the prompt and suggestion, respectively. Figure 7.2 displays a portion of timeline built from telemetry data drawn from a coding session. Telemetry data from each programmer is stored in a database $D = \{(t_i, A_i, P_i, S_i)\}_{i=1}^n$ and represents a discretized representation of the interaction and provides the human feedback data we leverage.

Programmer State. When faced with a suggestion, is a programmer looking and verifying it, or rather engaged in other activities such as thinking about their code or looking at documentation? The state of the programmer is important in the expected value of the recommendation. However, we cannot answer this question as the telemetry does not capture the programmer’s activities and thinking between two consecutive time stamps t_i and t_{i+1} , i.e., the space in between the arrows in Figure 7.2 which we refer to as the *programmer’s latent state*. In the previous Chapter 6, we describe a study of 21 participants focused on gaining an understanding of sequences of states visited during the writing of code, including latent states. The work employed videos and interviews to acquire information about the latent states. We showed in the work that including information about latent states can significantly boost predictions about accepting recommendations, motivating the collection of data beyond that captured in telemetry.

In this work, we endeavor to understand the impact of the programmer latent state denoted as ϕ_t

and its effect on our ability to leverage the telemetry (human feedback data) to improve AI-code recommendation systems.

7.4 Theoretical Formulation of Suggestion Utility

A critical design question in programmer-CodeRec interaction is **when** should the model inject a suggestion into the IDE? The version of that we CodeRec provides a suggestion when it detects a brief pause in the IDE. Alternative interaction designs would require the programmer to ask for suggestions using a keyboard shortcut or to enable a mix of human and machine initiatives. Requiring the programmer to ask may lead to sub-optimal interactions because its success would rely on programmers having an accurate mental model of CodeRec abilities [179] which can require long-term interactions with the model [16] or training as in Chapter 4. Second, requiring an explicit invocation can disrupt the natural flow of programming, breaking a state of flow achieved during intensive focus [213]. Designs requiring user initiative as well as those automatically displaying content can burden users with interruptions that decrease task performance [214], [215]. We note that such costs can be inferred and accounted for formally in utility-theoretic systems [216], [217].

Ideally, CodeRec should display suggestions when the suggestions provide net value to programmers. For example, consider the task of completing a function and the time taken to complete it as a proxy for the total effort. If the expected time required to verify and edit CodeRec’s suggestion exceeds the time to write the code by themselves (counterfactual cost), then CodeRec should not show its suggestions. Conversely, if the expected time to write exceeds the time to verify and edit, it may be useful to display the suggestion. We now formalize this intuition with a utility-theoretic formulation and, in the next section, discuss the methodology to make it practical.

Programmer Model. At a given time instance time during a session, CodeRec extracts a prompt P from the code file X and generates a code suggestion S . If this suggestion is shown, we assume the programmer spends an expected time $E[\text{verification}|X, S, \phi]$ to verify it and accepts the suggestion with probability $\mathbb{P}(A = \text{accept}|X, S, \phi)$. Once a suggestion is accepted, the programmer may further edit the suggestion with expected time $E[\text{editing}|X, S, \phi, A = \text{accept}]$ to achieve their task. On the other hand, if the programmer rejects the suggestion, they would have to spend time writing code that achieves their task, denoted by $E[\text{writing}|X, S, A = \text{reject}]$. Thus, the total time incurred with showing a suggestion, denoted as $E[S \text{ shown}|X, S, \phi]$, is:

$$\begin{aligned}
 E[S \text{ shown}|X, \phi] &= E[\text{verification}|X, S, \phi] & (7.1) \\
 &+ \mathbb{P}(A = \text{accept}|X, S, \phi) \cdot E[\text{editing}|X, S, \phi, A = \text{accept}] \\
 &+ \mathbb{P}(A = \text{reject}|X, S, \phi) \cdot E[\text{writing}|X, S, \phi, A = \text{reject}]
 \end{aligned}$$

While editing and writing, CodeRec may further make more suggestions; thus, the editing time and writing time should include interactions with future suggestions. Now, on the other hand, if the suggestion is *not* shown, the programmer will spend time $E[\text{writing}|X]$ writing code for their task. We also need to factor in latency, the time cost τ to compute a suggestion once we decide to create a suggestion. Latency is only experienced by the programmer if their latent state ϕ includes expecting a suggestion and waiting for it. If the programmer is expecting a suggestion, we should add τ to the total time when we show a suggestion; otherwise, the programmer continues to write code not expecting a suggestion.

We now define suggestion utility, a value that indicates the change in programmers’ coding time due to showing the suggestion.

Definition 4 (Suggestion Utility). The time impact δ , denoted as the *suggestion utility*, of showing S versus not showing is defined as:

$$\delta = \underbrace{E[\text{writing}|X, \phi]}_{\text{S not shown}} - \underbrace{E[S \text{ shown}|X, \phi]}_{\text{S shown}} - \underbrace{E[\tau|X, \phi]}_{\text{latency}} \quad (7.2)$$

From the above, a suggestion S at a given time should **only be shown** if $\delta > 0$ (Equation 7.2), where the programmer will spend less time to achieve their task if it is shown. An optimal scheme to know when to show suggestions would be to generate suggestions as frequently as possible, compute their *suggestion utility* δ , and display them if $\delta > 0$.

Feasibility of Estimating δ . Per Equation (7.1), computing *suggestion utility* requires the computation of four quantities: (1) the expected time spent verifying a suggestion, (2) the expected time editing a suggestion, (3) the expected time to write a segment of code and (4) the probability of accepting a suggestion. One can attempt to build an estimator for (1), by predicting from the prompt and suggestion the time spent verifying a suggestion i which would be $t_{i+1} - t_i$ using standard regression estimators. Unfortunately, using the same features and the dataset detailed in our experimental section, our best estimator is only able to achieve an $R^2 = 0.13$, which is not much better than a naive median time estimate. This may be due to the high variance and unobserved confounders governing verification time. Estimating editing and verification time (quantities 2 and 3 above) is only more complex and challenging. Thus, we restrict our methodology to seeing when we can evaluate δ using only our estimator for the probability of acceptance (4).

Learning Programmer’s Acceptance Decisions. The full conditional for the probability that the programmer accepts a suggestion is $\mathbb{P}(A = \text{accept}|X, S, \phi)$. Given the telemetry, we can only compute $\mathbb{P}(A = \text{accept}|X, S)$ where the programmer’s latent state cannot be observed. Using standard calibrated classification methods, we can estimate the probability $\mathbb{P}(A = \text{accept}|X, S)$ by using the actions A_i as the labels. We show that a simple mechanism of thresholding the estimated

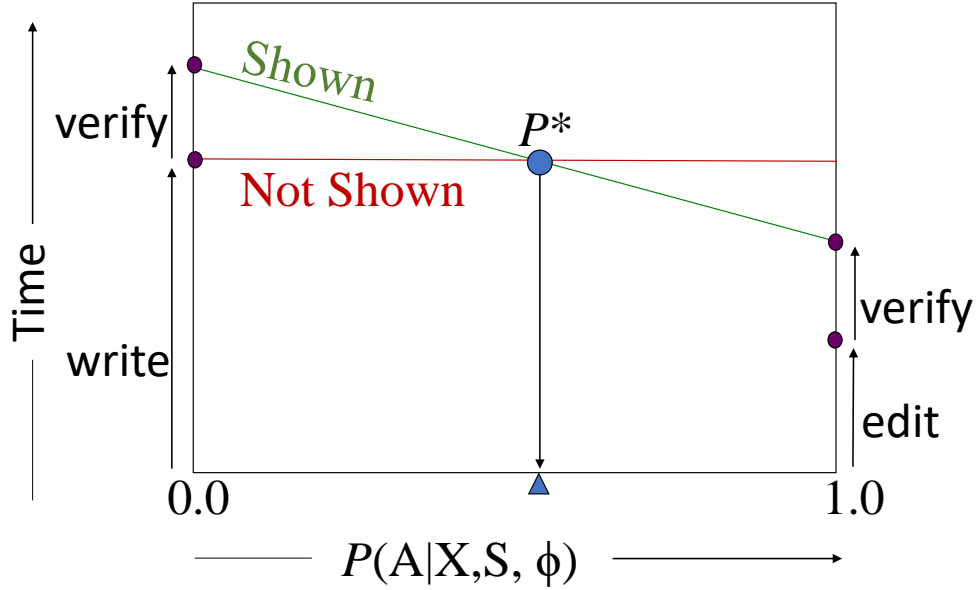


Figure 7.3: Graphical depiction of analysis of Proposition 6 when the latency is zero. The y-axis shows total time and the x-axis is the programmer’s probability of accepting $\mathbb{P}(A = \text{accept}|X, S, \phi)$. At probability \mathbb{P}^* , showing and not showing the suggestion have equal time cost.

probability that the programmer accepts a suggestion is equivalent under certain assumptions to checking if $\delta < 0$:

Proposition 6. *Under assumptions that the programmer spends more time writing code when they reject a suggestion compared to when they accept a suggestion and edit it, given specific code, suggestion, and latent state (X, S, ϕ) , if the programmer’s probability of accepting $\mathbb{P}(A = \text{accept}|X, S, \phi)$ a suggestion is below \mathbb{P}^* , which is defined as:*

$$\mathbb{P}^* = \frac{E[\text{verification}] + E[\text{latency}]}{E[\text{writing}|A = \text{reject}] - E[\text{editing}|A = \text{accept}]} \quad (7.3)$$

then the suggestion should not be shown. Note that \mathbb{P}^ is defined as a function $\mathbb{P}^*(X, S, \phi)$ evaluated pointwise.*

The formal statement and proof are available in the appendix. The above proposition shows that comparing the probability of acceptance to \mathbb{P}^* can guide when to show the suggestion. We provide a graphical view of the analysis in Figure 7.3, in the spirit of related analyses on utility-guided interactive interfaces [212]. Practically, if we compare the probability of acceptance to a constant lower bound of \mathbb{P}^* , we can guarantee that we hide suggestions only when $\delta < 0$.

Effect of Programmer Latent State. As mentioned previously, the programmer’s latent state is not available via telemetry. Thus, we can only provide predictions of $\mathbb{P}(A = \text{accept}|X, S)$ versus explicit consideration of the latent state, $\mathbb{P}(A = \text{accept}|X, S, \phi)$. In the earlier Chapter

6, we collected telemetry data of 21 programmers performing various tasks and had participants retrospectively label the telemetry with their latent state from a set of twelve unique states (1096 suggestions). We use this data to build predictive models with and without the latent state using the same methodology in the experiments section. Using a leave-one-out programmer evaluation strategy, the model without the latent state achieves accuracy 61.9 ± 1.9 while the model with the latent state achieves 83.6 ± 2.4 , a statistically significant difference according to a paired t-test ($p = 6.9e - 7, t = 7.11$); a similar result occurs when comparing areas under the receiver operating characteristic curve (AUC). These results highlights an opportunity to gather external data beyond telemetry to build such predictive models and indicates that acceptance may not simply be a property of suggestions and code context.

7.5 Conditional Suggestion Display From Human Feedback

In this section, we describe the CDHF method that can be implemented using telemetry data to identify when to show suggestions, as illustrated in Figure 7.1. We note from Equation (7.3) that the higher the probability of accepting the suggestion and the lower the latency to generate the suggestion, the more likely the suggestion is useful ($\delta > 0$). Our proposed approach is as follows: Each time the programmer pauses typing, we decide using a predictor whether to show a suggestion. Crucially, we do this using a two-stage scheme to avoid generating suggestions when we know the programmer would reject them.

Display Decision. Let $m(X, S)$ be a binary predictor that denotes whether, at a given moment in the code X , we should show the suggestion S ; we call this the display decision. If $m(X, S) = 1$, we display the suggestion; otherwise, we do not. The most straightforward way to build such a function m is to estimate the programmer’s probability of accepting the suggestion: $\mathbb{P}(A = \text{accept})|X, S$ and then threshold the probability so that suggestions that fall below a probability t are hidden. However, this will lead us to generate suggestions including those that will never be shown, thus wasting computing resources. We propose to decompose the function m so that we first decide using only the code whether we can make the display decision without generating the suggestion S with a function $r(X)$. If $r(X) = 1$, we make the display decision using a stage 1 model $m_1(X)$ without generating the suggestion, otherwise if $r(X) = 0$ we generate the suggestion S and make the display decision with a stage 2 model $m_2(X, S)$ as follows:

$$m(X, S) = r(X) \cdot m_1(X) + (1 - r(X)) \cdot m_2(X, S) \quad (7.4)$$

This formulation allows us to avoid generating suggestions when we can make an accurate display decision in advance of knowing the suggestion. For example, in a setting where the programmer

has rejected the last 30 suggestions, they are unlikely to accept the next suggestion.

Objective and Guarantees. Our objective in learning the functions r, m_1, m_2 is to (1) hide as many suggestions that would have been rejected and (2) maximize the number of display decisions made without generating the suggestion to reduce latency on the system. There is an inherent trade-off between these two objectives as making decisions with access to the suggestions would be more accurate. Moreover, we want to make sure we do not hide suggestions that would have been accepted, as this would limit the usefulness of the code assistant. Therefore, we impose a constraint that, whenever we hide a suggestion, there is at least a probability p it would have been rejected, a constraint on the true negative rate (TNR). We translate the objectives and the constraint into the following optimization problem:

$$\max_{r, m_1, m_2} \lambda \mathbb{E}[1 - m(X, S)] + (1 - \lambda) \mathbb{E}[r(x)] \quad (7.5)$$

$$s.t. \mathbb{P}(A = \text{reject} | m(X, S) = 0) \geq p \quad (7.6)$$

Parameterization. We can control the trade-off between the two objectives with a hyperparameter $\lambda \in [0, 1]$. Equivalently, instead of controlling the trade-off with λ , we can set a constraint on $\mathbb{E}[r(x)] := R$ and set $\lambda = 1$. We propose an intuitive post-hoc procedure to solve the optimization problem (7.5): We first learn calibrated estimators of the probability of accepting suggestions $\hat{\mathbb{P}}(A = \text{accept} | X, S)$ (with suggestion) and $\hat{\mathbb{P}}(A = \text{accept} | X)$ (without suggestion). We then parameterize:

$$m_1(X) = \mathbb{I}_{\hat{\mathbb{P}}(A=\text{accept}|X) \geq t_1}, m_2(X) = \mathbb{I}_{\hat{\mathbb{P}}(A=\text{accept}|X, S) \geq t_2}$$

and $r(X) = \mathbb{I}_{H(\hat{\mathbb{P}}(A=\text{accept}|X)) \leq t_r}$ ($H(\cdot)$ is Shannon’s Entropy), and optimize *jointly* over the tuple of thresholds t_1, t_2, t_r over $[0, 1]^3$. This is a fairly efficient procedure that can achieve good results. We note that this procedure saves latency indirectly by reducing the number of LLM calls across the session and across different users, and that we should still enable the user to see the suggestion with a special keyboard shortcut to override the display decisions. In the next section, we perform a retrospective evaluation of CDHF.

7.6 Experiments

Our main aim with experiments is to understand how well the CDHF procedure can make display decisions in a retrospective evaluation. Code is available² and additional details can be found in the appendix.

²https://github.com/microsoft/coderec_programming_states

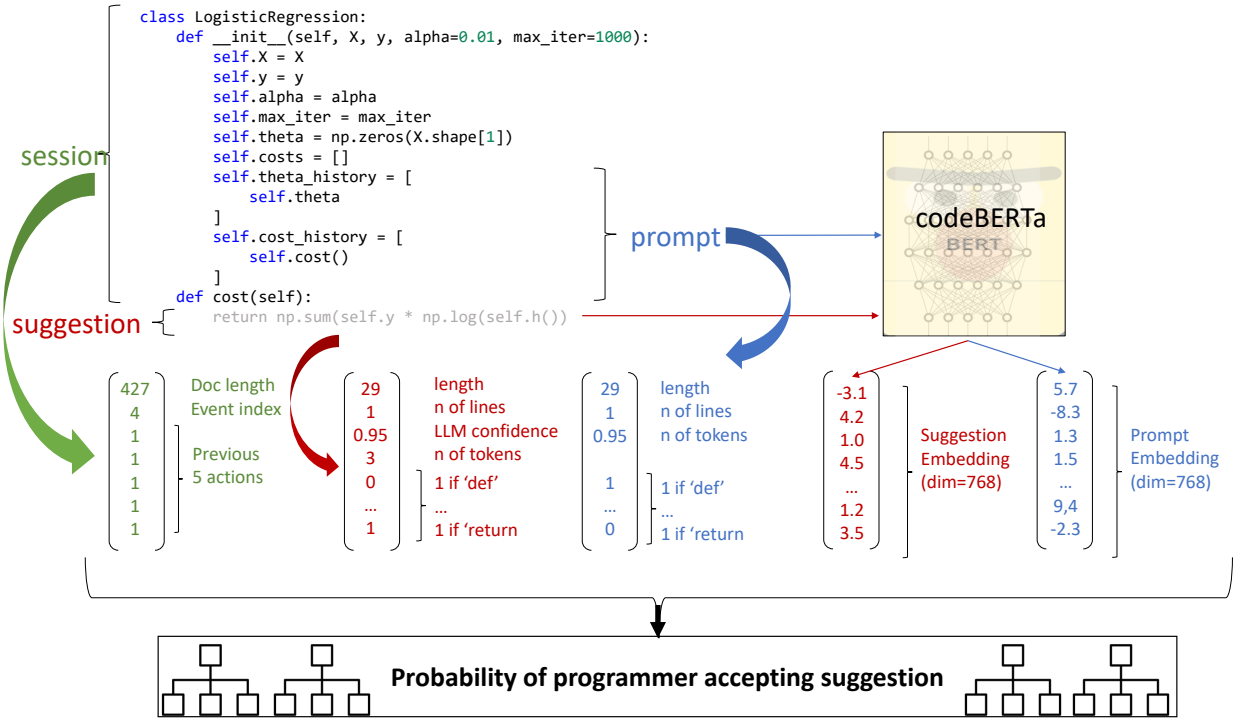


Figure 7.4: Features used to build action prediction model in Experiments, including from the suggestion, prompt, and session.

7.6.1 Dataset and Feature Engineering.

Dataset. To build and evaluate our methods, we extract a large number of telemetry logs from CodeRec users (mostly software engineers and researchers) at Microsoft. Programmers provided consent for the use of their data, and its use was approved by Microsoft’s ethics advisory board. Specifically, for a two-week time period, we extracted all the telemetry events for 535 users who coded in Python. This totals 4,749 coding sessions, where a session is defined as a continuous sequence of user actions with at most 30 minutes between consecutive events. These sessions are from real-world usage of CodeRec for daily tasks of the software engineers and researchers, the data was collected prior to the inception of our work. On average, each user contributes nine sessions, with each session lasting 21 minutes (median, 12 minutes). Sessions contain an average of 97 events (show, accept, and reject). This totals to almost 1,675 hours of coding with 168,807 shown events and 33,523 accept events, yielding an acceptance rate of 21.4% (not meant to represent Copilot’s average acceptance rate).

Model Features. The telemetry dataset D described above contains for each user a list of events in each of their coding sessions; we denote $D_{i,j}$ to be the list of events for the j ’th session of the i ’th user. The dataset $\mathbf{D} = \{D_{i,j}\}$ contains for each user i and session j , a list of events occurring in the corresponding coding session. We extract only the accept and reject events, as well

as prompt and session features of the corresponding shown events. For each prompt and suggestion pair, we extract: the programmer id as one hot vector, the length of the document, the previous five actions, suggestion features (e.g., suggestion length), previous features of the last five suggestions shown, the confidence reported by CodeRec, an embedding using codeBERTa [218] of prompt and suggestion, presence of Python keywords (e.g., `import`, `def` `try`, etc.), and the output of the Tree-sitter Parser [219]. Finally, we extract features of the prompt, including its embedding, textual features, and parser outputs. Figure 7.4 summarizes the feature engineering. It is crucial to note that the features do not leak any information about future events and can be computed as soon as a suggestion is generated by CodeRec. For the first stage model (m_1) in CDHF, suggestion features are omitted while we include all features for the second stage model (m_2). This feature engineering incorporates past actions and suggestions that the programmer has seen and allows us to use regular ML algorithms instead of time-series methods.

7.6.2 Model Evaluation

Before we evaluate CDHF, we perform an evaluation of the programmer acceptance model $m_2(X, S)$. We split the telemetry dataset in a 70:10:20 split for training, validation, and testing respectively. Importantly, we do this split in two ways: (1) by randomly splitting over programmers so that no single programmer is shared across the three splits and, (2) by randomly splitting over sessions so that users in training can also be seen in testing to allow for personalization.

Results. We evaluate different standard machine learning models on this task and find that the best-performing model is eXtreme Gradient Boosting (XGB) [220]. When we split across users, XGB is able to achieve 81.1% (95% CI 80.7-81.6) accuracy and, more importantly, 0.780 (95% CI 0.775-0.786) AUC. In the appendix, we show metrics for different models evaluated, including deep networks. The results indicate that the model is able to distinguish between suggestions that are likely to be accepted versus those likely to be rejected. The model is also well calibrated: the expected calibration error is 0.10 [221].

We note a significant increase in AUC when we allow for personalization: including programmer ID as a feature and splitting across sessions, this leads to an AUC of 0.795 (95% CI 0.789-0.801), a significant increase (basis of m_2 model). When we remove suggestion features from the model, the resulting model (basis of m_1 model) achieves an AUC of 0.631 (95% CI 0.624-0.638). The time to compute the features needed for the models and performing inference on a single data point can take 10ms with a GPU and less than 1ms on a CPU when omitting embeddings, in addition to latency of sending and receiving information between server and client. In the appendix, we show results for different ablation of model features, sample complexity plots, and feature importance plots.

7.6.3 Retrospective Evaluation of CDHF

We train the models m_1 and m_2 using the training set per the previous subsection. We set the thresholds t_1, t_2, t_r on the validation set for CDHF and evaluate on the test set.

Results. In Figure 7.5, we vary the desired TNR rate (accuracy when a suggestion is hidden) and plot how many suggestions we can hide from those previously displayed while guaranteeing the desired TNR rate. We show the behavior of the CDHF method with different λ values, or, equivalently, with different constraints on how often the m_1 model (first stage) is used: $R := \mathbb{E}[r(x)]$. To illustrate what CDHF can accomplish, we can hide 25.3% of suggestions that were shown while guaranteeing that 94.7% of them would have been rejected and avoid generating 12.9% of the suggestions. If we have no concerns for latency, we can hide 52.9% of suggestions while guaranteeing that 91.3% of them would have been rejected. Figure 7.5 shows how we can achieve different trade-offs by selecting an operating point on any given curve. CDHF is able to satisfy the constraint of FNR on the test set with a violation of at most 0.3% i.e., a guarantee of 95% FNR on the validation set equates to 94.7-95.3% on the test set.

Counterfactual Increase in Acceptance Rate. On the test set, the acceptance rate of suggestions is 22.5%. Retrospectively, if we had used CDHF to hide 52.9% of suggestions, we could compute a counterfactual acceptance rate. The counterfactual acceptance rate can be computed as: $\frac{S_{\text{accepted}} \cdot (1 - \% \text{ hidden} \cdot (1 - TNR))}{S_{\text{shown}} \cdot \% \text{ not hidden}} = \frac{22.5(1 - 0.529 \cdot 0.087)}{0.471} = 45.6\%$, which is a 23.1 point increase, a value we expect to be an overestimate.

Discussion and Limitations. The retrospective evaluation shows that CDHF has promise in reducing developer time spent verifying suggestions or waiting for suggestions. We note that our evaluation is retrospective. Although GitHub has shown that conditional suggestion filters similar to CDHF increase the acceptance rate of suggestions, a user study is required to verify whether the method makes programmers more productive. As Goodhart’s law states, once a metric becomes a target, it ceases to become a good measure; acceptance rate is no exception. Moreover, if CDHF is not trained with sufficient data that captures the programmer’s use cases, it can make the programming experience worse by hiding useful suggestions. Moreover, a rejected suggestion may still be useful, which we do not account for here. The optimization problem in (7.5) is amenable to procedures inspired by learning to defer [26] that can outperform the post-hoc procedure proposed. Finally, one issue with the scheme presented is that we might hide suggestions that programmers would reject but might want to see anyway. We could remedy this issue by building a secondary model that predicts whether the user would request a suggestion at a given point in time. If that model predicts a high likelihood of a user request, then we should show the suggestion regardless of the likelihood of acceptance. However, if we predict a low likelihood of request, we should then use the scheme presented in this chapter as is.

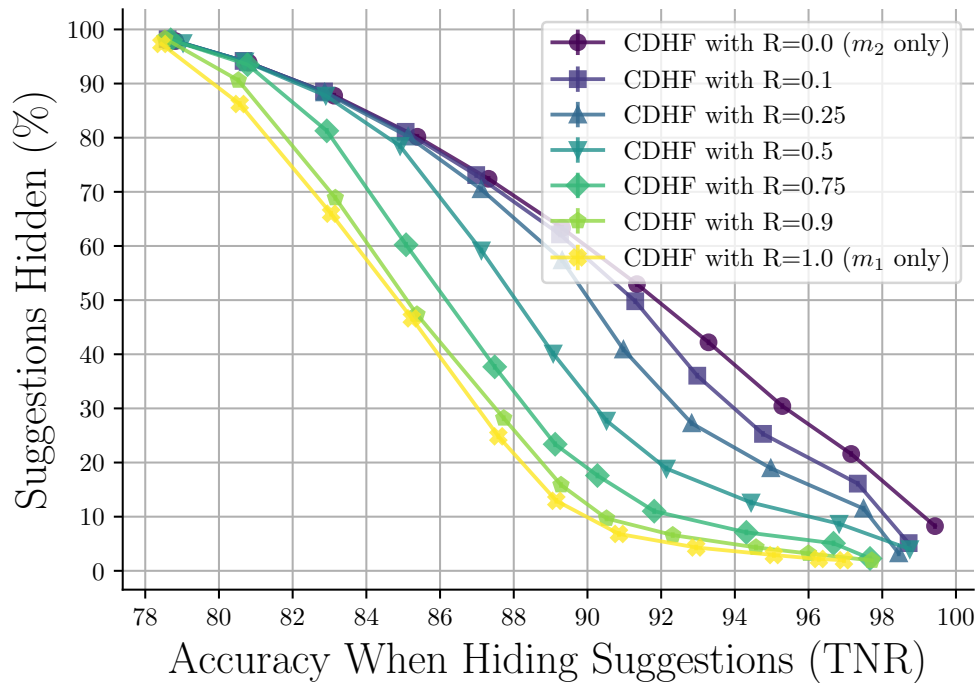


Figure 7.5: Evaluation of CDHF for selectively hiding suggestions. For a given constraint on FNR (accuracy when a suggestion is hidden) on the x-axis, we show on the y-axis the fraction of the total suggestions we can hide while guaranteeing the desired FNR. We plot these curves while varying how often the decision is made generating suggestions ($R := \mathbb{E}[r(x)]$, when $R=0$, we generate the suggestion then decide to hide or not, when $R=1$, we decide to hide without knowing the suggestion).

7.7 Which Suggestion to Show?

We focus in this study on the problem of when to display suggestions. We did not tackle the question of which suggestions to display among a candidate set. Given access to telemetry data, which consists of contextualized suggestions with accept and reject signals, one can interpret an accept as the act of preferring a suggestion over no suggestion. It is reasonable to harness the telemetry data as a preference dataset and build a reward model of programmers' preferences, which would be equivalent to estimating the programmer's acceptance probability $\hat{\mathbb{P}}(A = \text{accept} | X, S)$. Thus, a reasonable procedure is to take a candidate set of suggestions \mathcal{S} and display the suggestion that maximizes the probability of acceptance across the set; this is essentially the best-of- n baseline approach in RLHF [222].

Potential Bias Towards Short Suggestions. We hypothesize that such a ranking scheme would not be productive and can lead to poor suggestions of short length. Our rationale is the following: suppose the LLM is able to generate a multi-line suggestion S for a user query that approximately matches what the user desires. To maximize the probability that the user accepts the suggestion, it would be advantageous to split the suggestion S line-by-line and display it to the user step-by-step. The reasoning is that it is more likely for the first line of S to be correct rather than all of S being correct, hence being more likely to be accepted.

Experiment. To test this hypothesis, we perform the following experiment: We learn a model m of suggestion acceptance given only the prompt and suggestion embeddings with no session features on the telemetry data from the previous section. We then leverage the HumanEval dataset [169], which consists of 164 Python problems, each with an associated docstring and a ground truth function body solution. Solutions have at least two lines and seven median lines of code. Given the model m and each problem, we let the prompt be the concatenation of the docstring and the first k lines of the solution and let the candidate set of suggestions \mathcal{S} be as follows: Given the solution S represented as an array of tokens of length N , we let $\mathcal{S} = \{S[:i]\}_{i=1}^N$. For example, if the solution S was "return np.mean(x)", then $\mathcal{S} = \{\text{"return"}, \text{"return np.mean(x)}\}$.

Results. We vary the parameter k in the set $\{0, 1, 2, 3\}$ so that the prompt goes from the docstring to include lines of the solution. When $k = 0$, the normalized length of the highest-rated suggestion, according to the model across the 164 problems, is almost uniform across $[0, 1]$, a Kolmogorov–Smirnov test compared to the uniform distribution has a p-value of 0.53 (KS=0.06). Optimally, we want the normalized length to cluster around 1 to include the full solution. However, when $k > 0$, meaning that the prompt includes lines of code, we find that for over 60 of the 164 problems, the highest-scored suggestion lies in $[0, 0.2]$, and, for at least 40 problems, it is the first token. This provides some evidence that optimizing for acceptance can be biased toward shorter suggestions since the highest-ranked suggestion is often in the first few lines of the solution.

Limitations. However, there are important limitations in our experiment. First, the model m is only trained on Copilot suggestions. Thus, the bias towards short suggestions can be due in part to Copilot potentially showing short suggestions. Maximizing acceptance would not alleviate such a bias. Second, while the use of embeddings of the suggestions for the reward model led to accurate predictions of accepts (AUC=0.701), they might be biased in some ways as compared to alliance on fine-tuning the language model.

Chapter 8

Conclusion

In this thesis, we explored three different settings for human-AI collaboration and proposed approaches to improve the performance of the human-AI team in each of them. We first studied conditional delegation with AI in Part I, where we train a secondary AI called rejector that decides on each instance who between the AI and the human should predict. In Chapter 2, we analyzed the gap in performance between jointly learning a classifier and rejector, and a staged learning approach. We further analyzed a popular approach to jointly learning to defer, namely consistent surrogate loss functions. To that end, we proposed a novel family of surrogates that generalizes prior work and gives criteria, namely the surrogate excess-risk bound for evaluating surrogates. Driven by the limited availability of human data, we sought to design active learning schemes that require a minimal amount of labeled data to learn a classifier-rejector pair. Following that in Chapter 3, we showed that properly learning halfspaces with deferral is computationally hard and that existing approaches in the literature fail in this setting. Understanding the computational limits of learning to defer led to the design of a new exact algorithm (the MILP) and a new surrogate (RealizableSurrogate) that both obtain better empirical performance than existing surrogate approaches.

Future work should try to instantiate members of the family of surrogate losses discovered that minimize the excess-risk bound and provide improved empirical performances. While our active learning results hold for the realizable setting, we believe it is feasible to generalize to the agnostic setting. Future work should also build and test practical active learning algorithms inspired by our theoretical analysis. We studied two forms of classification consistency in Part I, general classification consistency, and $(\mathcal{M}, \mathcal{R})$ -consistency. However, no single surrogate loss satisfied both notions of consistency at the same time; it remains an open problem to find a surrogate that satisfies both notions of consistency. We studied the offline version of the learning to defer problems in our work, and extending our work to the online learning setting is of interest. In the online setting, when not deferring, the classifier can no longer obtain information on the expert's behavior.

Furthermore, when deferring, there might be natural reasons why the classifier may not receive feedback on its predictions: at the extreme, we have a lack of feedback when we defer and when we don't. This causes us to inherit all the problems that may occur when deploying a classifier to a new setting due to environmental shifts in addition to possibly changing human decision-makers. Moreover, the deferral system may cause the human's abilities to change over time as the human no longer observes examples in the distribution that are not deferred. Future work should attempt to make deferral systems more robust to distribution changes in either the human's abilities or the environment.

In Part II, we studied the AI-assisted decision making setting where AI provides advice to a decision maker. Our work provides a general recipe for onboarding human decision makers to AI systems. In Chapter 4 we propose an exemplar based teaching strategy where humans are asked to predict on real examples and then with the help of similar examples and top features for the neighborhood, the human derives an explanation for the AI performance. The initial study on question answering we performed showed that 50% of participants derived a correct mental model of the AI and had improved performance on the task. Following that in Chapter 5, we introduced IntegrAI, a novel algorithmic framework designed to enhance human-AI collaboration and automate the process of describing the AI performance. At the core of our framework is the AI-integration function, which represents the human's mental model to either rely on, ignore, or collaborate with AI on a task-specific basis. The first step of our algorithm is to collect data about human performance on the task and about their prior expectations and reliance on the AI model. The objective is to teach the human in order to correct their prior about how to cooperate with the AI. The second step is to discover regions of the data as local neighborhoods in an embedding space where the human prior is incorrect. The third step is to describe the regions with natural language and label them with the correct action the human should take in that region: either use or ignore the AI. Finally, in the onboarding phase, we teach these regions to the participants using our proposed method. We found that on an object detection task our onboarding procedure significantly improved performance and that in another question-answering task, it did not have a significant effect.

There are many directions for future work to explore both on the algorithmic side and on the behavioral side. For instance, our region discovery and region description algorithms are decoupled, ideally we can jointly discover and describe regions so that only regions with interpretable language descriptions are found. Second, our user studies only handled the case when the human can either use or ignore the AI, collaboration with the AI was implicitly done by the participants but it was not captured in our data, future work can build interfaces and procedures to handle the case when $R = 2$. In our user studies, we only allowed participants to discuss how they used the AI after the study was completed, we did not collect enough data about user behavior and user learning, future work could explore more qualitative insights about onboarding.

Finally, in Part III, we studied more interactive forms of human-AI collaboration, notably in writing code with large language models. In Chapter 6, we developed and proposed a taxonomy of common programmer activities (CUPS) and combined it with real-time telemetry data to profile the interaction. At present, CUPS contains 12 mutually unique activities that programmers perform between consecutive Copilot actions (e.g., such as accepting, rejecting, and viewing suggestions). We gathered real-world instance data of CUPS by conducting a user study with 21 programmers within our organization, where they solved coding tasks with Copilot and retrospectively labeled CUPS for their coding session. We collected over 3137 instances of CUPS and analyzed them to generate CUPS timelines that show individual behavior and CUPS diagrams that show aggregate insights into the behavior of our participants. We also studied the time spent in these states, patterns in user behavior, and better estimates of the cost (in terms of time) of interacting with Copilot. Our studies with CUPS labels revealed that when solving a coding task with Copilot, programmers may spend a large fraction of total session time (34.3%) on just double-checking and editing Copilot suggestions, and spend more than half of the task time on Copilot related activities, together indicating that introducing Copilot into an IDE can significantly change user behavior.

We only investigated a limited number of programmer behaviors using the CUPS timelines and diagrams. There are many other aspects future work could investigate. To enable our insights derived in Section 6.6, we need to be able to identify the current programmer's CUPS state. An avenue towards that is building predictive models using labeled telemetry data that is collected from our user study. Ideally, we can leverage this labeled data to further label telemetry data from other coding sessions or other participants so that we can perform such analyses more broadly. Specifically, the input to such a model would be the current session context, for example, whether the programmer accepted the last suggestion, the current suggestion being surfaced, and the current prompt. We can leverage supervised learning methods to build such a model from collected data. Such models would need to run in real-time during programming and predict at each instance of time the current user CUPS state. This would enable the design suggestions proposed to serve to compute various metrics proposed. For example, if the model predicts that the programmer is deferring thought about a suggestion, we can group suggestions together to display them to the programmer. In the Appendix, we built small predictive models of programmers CUPS state using labeled study data. However, the current amount of labeled data is not sufficient to build highly accurate models. There are multiple avenues to improve the performance of these models: 1) simply collecting a larger amount of labeled data which would be expensive, 2) using methods from semi-supervised learning that leverage unlabeled telemetry to increase sample efficiency [223], and 3) collecting data beyond what is captured from telemetry such as video footage of the programmer screen (e.g. cursor movement) to be able to better predict with the same amount of data. Another opportunity is to apply the CUPS diagram to compare different user groups and compare

how individuals differ from an average user. Does the nature of inefficiencies differ between user groups? Can we personalize interventions? Finally, we could also compare how the CUPS diagram evolves over time for the same set of users. We only studied the behavior of programmers with the current version of Copilot. Future work could study how behavior differs with different versions of Copilot— especially when versions use different models. In the extreme, we could study behavior when Copilot is turned off. The latter could help assess the *counterfactual* cost of completing the task without AI assistance and help establish whether and where Copilot suggestions add net value for programmers. For example, maybe the system did not add enough value because the programmer kept getting into prompt crafting rabbit holes instead of moving on and completing the functions manually or with the assistance of web search. Likewise, if developers create a faster version of Copilot with less latency, the CUPS diagram could be used to establish whether it leads to reductions in time spent in the "Waiting for Suggestion" state. Since programmers' value may be multi-dimensional, how can we go beyond code correctness and measure added value for users? If Copilot improves productivity, which aspects were improved? Conversely, if it didn't, where are the efficiencies? One option is to conduct a new study where we compare the CUPS diagram with Copilot assistance with a counterfactual condition where the programmers don't have access to Copilot. And use the two diagrams to determine where the system adds value or could have added value. For example, the analysis might reveal that some code snippets are too hard for programmers to complete by themselves but much faster with Copilot because the cost of double-checking and editing the suggestion is much less than the cost of spending effort on it by themselves. Conversely, the analysis might reveal that a new intervention for helping engineer prompts greatly reduced people's times in "Prompt Crafting". Another option is to design offline metrics based on these insights that developers can use during the model selection and training phase. For example, given that programmers spent a large fraction of the time verifying suggestions, offline metrics that can estimate this (e.g., based on code length and complexity) may be useful indicators of which models developers should select for deployment. Future work will aim to test the effectiveness of these design suggestions as well. We also hope our methodology is applied to study other forms of AI assistants that are rapidly being deployed. For example, one can make an analogous CUPS taxonomy for writing assistants for creative writers or lawyers.

In Chapter 7, we proposed a strategy to decide when to display code suggestions in AI-assisted programming to improve time efficiency inspired by the insights in Chapter 6. This strategy was based on a utility theory formulation and employs a two-stage procedure using a predictive model of suggestion acceptance. A retrospective evaluation showed that we can reduce the number of suggestions and thus programmers' time without sacrificing the utility of CodeRec. However, a prospective study that evaluates the impact of CodeRec with and without CDHF could help with conclusive evaluation and is the basis of future work. Moreover, future work will attempt to directly

estimate Proposition 1 leveraging improved methods. We don't believe that CDHF can introduce negative consequences beyond what CodeRec introduces to the programmer as it functions as a filtering mechanism for unhelpful suggestions. Moreover, we believe that the CDHF methodology can be employed in a wide range of streaming human-AI collaboration tasks such as assisted writing. Future work will incorporate the latent state of the programmer into the predictive models, investigate how to rank suggestions using the models from CDHF, and validate the efficacy of CDHF in user studies.

Appendix A

Additional Information for Chapter 2

Notations

We employ the notations $L_{\text{def}}^{\mu_X}$, $L_{\text{def}}^{\mu_X\mu_{Y|X}}$, $L_{\text{def}}^{\mu_{XYM}}$ to indicate L_{def}^{0-1} and stress on marginal, conditional, and joint probability measures on X , Y , and M . We further use $L_{0-1}^{\mu_X\mu_{Y|X}}$ to indicate zero-one loss L_{0-1} and to represent the underlying probability measures on X and Y . The cardinality of a set \mathcal{A} is indicated by $|\mathcal{A}|$. The notation for the set of numbers from 1 to K is: $[K] = \{1, \dots, K\}$.

A.1 Proof of Theorem 1

We first introduce some useful lemmas as below. In Lemma 1, we show that there exists a pair of hypothesis classes $(\mathcal{H}, \mathcal{R})$ such that for all non-atomic measures on \mathcal{X} the deferral loss takes a fixed value. In Lemma 2, we use the aforementioned lemma to show that the difference of deferral losses for all two pairs of classifier/rejector (h_1, r_1) and (h_2, r_2) is bounded by the difference of two deferral losses with atomic measures on \mathcal{X} . In Lemma 3, we upper-bound the difference of two deferral losses for pairs of classifier/rejector that are obtained by staged and joint learning and on hypothesis classes that are defined in Lemma 1. Such upper-bound is in terms of expected loss of an optimal classifier on a certain hypothesis class. In Lemma 4, we further calculate the optimal expected loss on such classes. In Lemma 5, we show that on a set of events with size n , we could find a subset with size a and probability at most $\frac{a}{n}$. Next, we use these lemmas in the main proof of theorem.

Lemma 1. *For a probability measure μ_X with no atomic component on \mathcal{X} , hypothesis class \mathcal{H} such that for every $h \in \mathcal{H}$, we have $|\{\mathbf{x} : h(\mathbf{x}) = 1\}| \leq d(\mathcal{H})$, and hypothesis class \mathcal{R} such that for*

every $r \in \mathcal{R}$, we have $|\{\mathbf{x} : r(\mathbf{x}) = 1\}| \leq d(\mathcal{R})$, for every choice of $(h, r) \in \mathcal{H} \times \mathcal{R}$, the loss

$$L_{\text{def}}^{0-1}(h, r) = \mathbb{E}_{X,Y,M}[\mathbb{I}_{h(X) \neq Y} \mathbb{I}_{r(X)=0} + \mathbb{I}_{M \neq Y} \mathbb{I}_{r(X)=1}],$$

takes a constant value.

Proof. Firstly, we know that

$$L_{\text{def}}^{0-1}(h, r) = \mathbb{E}_{X,Y,M}[\mathbb{I}_{h(X) \neq Y} \mathbb{I}_{r(X)=0}] + \mathbb{E}_{X,Y,M}[\mathbb{I}_{M \neq Y} \mathbb{I}_{r(X)=1}]. \quad (\text{A.1})$$

Since probability measure of the set $\{x : r(x) = 1\}$ is zero in the absence of atomic components in μ_X , one can show that $\mathbb{P}(r(X) = 1) = 0$ (, and equivalently $\mathbb{P}(r(X) = 0) = 1$). This fact together with (A.1) concludes that

$$L_{\text{def}}^{0-1}(h, r) = \mathbb{E}_{X,Y}[\mathbb{I}_{h(X) \neq Y}]. \quad (\text{A.2})$$

Further, we have

$$\mathbb{E}_{X,Y}[\mathbb{I}_{h(X) \neq Y}] = \mathbb{E}_{X,Y}[\mathbb{I}_{h(X) \neq Y} | h(X) = 0] \mathbb{P}(h(X) = 0) + \mathbb{E}_{X,Y}[\mathbb{I}_{h(X) \neq Y} | h(X) = 1] \mathbb{P}(h(X) = 1) \quad (\text{A.3})$$

$$\stackrel{(a)}{=} \mathbb{E}_{X,Y}[\mathbb{I}_{Y=1}], \quad (\text{A.4})$$

where (a) holds because probability measure of $\{\mathbf{x} : h(\mathbf{x}) = 1\}$ is zero in the absence of atomic components in the measure, that concludes $\mathbb{P}(h(X) = 0) = 1 - \mathbb{P}(h(X) = 1) = 1$. The proof is complete by (A.2) and (A.4). \square

Lemma 2. Let μ_X be a probability measure on \mathcal{X} , and let \mathcal{H} and \mathcal{R} be hypothesis classes as in Lemma 1. Further, let $h_1, h_2 \in \mathcal{H}$ and $r_1, r_2 \in \mathcal{R}$. Then, we have

$$\left| L_{\text{def}}^{\mu_X}(h_1, r_1) - L_{\text{def}}^{\mu_X}(h_2, r_2) \right| \leq \left| L_{\text{def}}^{\mu_d}(h_1, r_1) - L_{\text{def}}^{\mu_d}(h_2, r_2) \right|, \quad (\text{A.5})$$

where μ_d is pure atomic (discrete) probability measure on \mathcal{X} .

Proof. We know that for probability measure μ_X , there exists $p \in [0, 1]$ and probability measures μ_d and μ_{cs} , such that

$$\mu_X = p\mu_d + (1 - p)\mu_{cs}, \quad (\text{A.6})$$

where μ_d is pure atomic and μ_{cs} has no atomic components. As a result, for every function

$f(\cdot) : \mathcal{X} \rightarrow \mathbb{R}$, we have

$$\mathbb{E}_{X \sim \mu_X} [f(X)] = p \mathbb{E}_{x \sim \mu_d} [f(X)] + (1-p) \mathbb{E}_{x \sim \mu_d} [f(X)]. \quad (\text{A.7})$$

With the same reasoning, we have

$$L_{\text{def}}^{\mu_X}(h, r) = p L_{\text{def}}^{\mu_d}(h, r) + (1-p) L_{\text{def}}^{\mu_{cs}}(h, r). \quad (\text{A.8})$$

Next, we have

$$L_{\text{def}}^{\mu_X}(h_1, r_1) - L_{\text{def}}^{\mu_X}(h_2, r_2) = p [L_{\text{def}}^{\mu_d}(h_1, r_1) - L_{\text{def}}^{\mu_d}(h_2, r_2)] + (1-p) [L_{\text{def}}^{\mu_{cs}}(h_1, r_1) - L_{\text{def}}^{\mu_{cs}}(h_2, r_2)] \quad (\text{A.9})$$

$$\stackrel{(a)}{=} p [L_{\text{def}}^{\mu_d}(h_1, r_1) - L_{\text{def}}^{\mu_d}(h_2, r_2)], \quad (\text{A.10})$$

where (a) holds because of Lemma 1 that proves $L_{\text{def}}^{\mu_{cs}}(h, r)$ is constant for every $(h, r) \in \mathcal{R} \times \mathcal{H}$.

Finally, using (A.10), and since $p \in [0, 1]$, the proof is complete. \square

Lemma 3. Let $\text{supp}(h) = \max_{\mathcal{S}: \forall x \in \mathcal{S}, h(x)=1} |\mathcal{S}|$ and $\mathcal{H}_d = \{h : \mathcal{X} \rightarrow \{0, 1\} \mid \text{supp}(h) \leq d\}$. Further, let μ_X be an atomic measure on \mathcal{X} , and define

$$\hat{h} := \underset{h \in \mathcal{H}_d(\mathcal{H})}{\text{argmin}} L_{0-1}^{\mu_X \mu_Y | X}(h), \quad (\text{A.11})$$

where

$$L_{0-1}^{\mu_X \mu_Y | X}(h) = \mathbb{E}_{\mu_X \mu_Y | X} [\mathbb{I}_{h(X) \neq Y}], \quad (\text{A.12})$$

and

$$\hat{r} := \underset{r \in \mathcal{H}_d(\mathcal{R})}{\text{argmin}} L_{\text{def}}^{\mu_X}(\hat{h}, r). \quad (\text{A.13})$$

Further, define the pair (h^*, r^*) be the optimal classifier

$$(h^*, r^*) = \underset{(h, r) \in \mathcal{H}_d(\mathcal{H}) \times \mathcal{H}_d(\mathcal{R})}{\text{argmin}} L_{\text{def}}^{\mu_X \mu_Y | X}(h, r). \quad (\text{A.14})$$

Then, if $d(\mathcal{H}) \geq d(\mathcal{R})$, we have

$$L_{\text{def}}^{\mu_X \mu_Y | X}(\hat{h}, \hat{r}) - L_{\text{def}}^{\mu_X \mu_Y | X}(h^*, r^*) \leq \min_{h \in \mathcal{H}_d(\mathcal{H}) - d(\mathcal{R})} L_{0-1}^{\mu_X \mu_Y | X}(h) - \min_{h \in \mathcal{H}_d(\mathcal{H})} L_{0-1}^{\mu_X \mu_Y | X}(h), \quad (\text{A.15})$$

where μ'_X is a purely atomic measure on \mathcal{X} .

Proof. Firstly, using (A.13), we know that

$$L_{\text{def}}^{\mu_X \mu_{Y|X}}(\hat{h}, \hat{r}) \leq L_{\text{def}}^{\mu_X \mu_{Y|X}}(\hat{h}, r^*). \quad (\text{A.16})$$

Hence, we have

$$\underbrace{L_{\text{def}}^{\mu_X \mu_{Y|X}}(\hat{h}, \hat{r}) - L_{\text{def}}^{\mu_X \mu_{Y|X}}(h^*, r^*)}_D \leq L_{\text{def}}^{\mu_X \mu_{Y|X}}(\hat{h}, r^*) - L_{\text{def}}^{\mu_X \mu_{Y|X}}(h^*, r^*) \quad (\text{A.17})$$

$$= \mathbb{E}[\mathbb{I}_{r^*(X)=0} \mathbb{I}_{\hat{h}(X) \neq Y}] - \mathbb{E}[\mathbb{I}_{r^*(X)=0} \mathbb{I}_{h^*(X) \neq Y}]. \quad (\text{A.18})$$

Next, we form the conditional probability measure $\mu'_X = \mu_{X|r^*(X)=0}$. Therefore, using (A.18) we have

$$D = \mathbb{P}(r^*(X) = 0) [L_{0-1}^{\mu'_X \mu_{Y|X}}(\hat{h}) - L_{0-1}^{\mu'_X \mu_{Y|X}}(h^*)]. \quad (\text{A.19})$$

Next, since $h^* \in \mathcal{H}_{d(\mathcal{H})}$, we know that

$$L_{0-1}^{\mu'_X \mu_{Y|X}}(h^*) \geq \min_{h \in \mathcal{H}_{d(\mathcal{H})}} L_{0-1}^{\mu'_X \mu_{Y|X}}(h). \quad (\text{A.20})$$

Moreover, we prove that

$$L_{0-1}^{\mu'_X \mu_{Y|X}}(\hat{h}) \leq \min_{h \in \mathcal{H}_{d(\mathcal{H})-d(\mathcal{R})}} L_{0-1}^{\mu'_X \mu_{Y|X}}(h). \quad (\text{A.21})$$

We prove this inequality by contradiction. If (A.21) is not correct, then there exists $h' \in \mathcal{H}_{d(\mathcal{H})-d(\mathcal{R})}$, such that

$$L_{0-1}^{\mu'_X \mu_{Y|X}}(h') < L_{0-1}^{\mu'_X \mu_{Y|X}}(\hat{h}). \quad (\text{A.22})$$

Then, we define a function $h'' : \mathcal{X} \rightarrow \{0, 1\}$ as below

$$h''(\mathbf{x}) = \begin{cases} h'(\mathbf{x}) & r^*(\mathbf{x}) = 0 \\ \hat{h}(\mathbf{x}) & r^*(\mathbf{x}) = 1 \end{cases}. \quad (\text{A.23})$$

Using the definition of \mathcal{H}_d and since $\text{supp}(r^*) \leq d(\mathcal{R})$, one could show that $h'' \in \mathcal{H}_{d(\mathcal{H})}$. Further-

more, we have

$$L_{0-1}^{\mu_X \mu_{Y|X}}(h'') = \mathbb{P}(r^*(X) = 0) L_{0-1}^{\mu_X \mu_{Y|X}}(h') + \mathbb{P}(r^*(X) = 1) L_{0-1}^{\mu_X |_{r^*(X)=1} \mu_{Y|X}}(\hat{h}) \quad (\text{A.24})$$

$$\stackrel{(a)}{<} \mathbb{P}(r^*(X) = 0) L_{0-1}^{\mu_X \mu_{Y|X}}(\hat{h}) + \mathbb{P}(r^*(X) = 1) L_{0-1}^{\mu_X |_{r^*(X)=1} \mu_{Y|X}}(\hat{h}) \quad (\text{A.25})$$

$$= L_{0-1}^{\mu_X \mu_{Y|X}}(\hat{h}), \quad (\text{A.26})$$

where (a) holds using (A.22), and (A.26) and $\hat{h} \in \mathcal{H}_{d(\mathcal{H})}$ is a contradiction of (A.11).

Using (A.19), (A.20), (A.21), and since $\mathbb{P}(r^*(X) = 0) \leq 1$, the proof is complete. \square

Lemma 4. *Let μ_X be a purely atomic measure on \mathcal{X} . Further, let $\{\mathbf{x}_{i,1}\}_i$ be the points in \mathcal{X} for which we have*

$$\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) \leq \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1}), \quad (\text{A.27})$$

and without loss of generality, assume that $\{\mathbf{x}_{i,2}\}$ are the points for which we have

$$\mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) > \mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}), \quad (\text{A.28})$$

and if $i < j$, then we have

$$\begin{aligned} & \mathbb{P}(X = \mathbf{x}_{i,2}) [\mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) - \mathbb{P}(Y = 0|X = \mathbf{x}_{i,2})] \\ & \geq \mathbb{P}(X = \mathbf{x}_{j,2}) [\mathbb{P}(Y = 1|X = \mathbf{x}_{j,2}) - \mathbb{P}(Y = 0|X = \mathbf{x}_{j,2})]. \end{aligned} \quad (\text{A.29})$$

Then, we have

$$\begin{aligned} \min_{h \in \mathcal{H}_d} L_{0-1}^{\mu_X \mu_{Y|X}}(h) &= \sum_i \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) + \sum_{i=1}^d \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}) \\ &+ \sum_{i=d+1}^{\infty} \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}), \end{aligned} \quad (\text{A.30})$$

where \mathcal{H}_d is defined as in Lemma 3.

Proof. Let h^* be the optimal classifier

$$h^* = \operatorname{argmin}_{h \in \mathcal{H}_d} L_{0-1}^{\mu_X \mu_{Y|X}}(h). \quad (\text{A.31})$$

Then, firstly, either $h(\mathbf{x}_{i,1}) = 0$, or $\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) = \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1})$ for all i s. We prove this claim by contradiction. If for some i , we have $h(\mathbf{x}_{i,1}) = 1$, and $\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) \neq$

$\mathbb{P}(Y = 0|X = \mathbf{x}_{i,1})$, then we could define h' as

$$h'(\mathbf{x}) = \begin{cases} h^*(\mathbf{x}) & \mathbf{x} \neq \mathbf{x}_{i,1} \\ 0 & \mathbf{x} = \mathbf{x}_{i,1} \end{cases}. \quad (\text{A.32})$$

One could see that $h' \in \mathcal{H}_d$, and that

$$L_{0-1}^{\mu_X \mu_Y | X}(h') - L_{0-1}^{\mu_X \mu_Y | X}(h^*) = \mathbb{P}(\mathbf{x}_{i,1}) [\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) - \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1})] \stackrel{(a)}{<} 0, \quad (\text{A.33})$$

where (a) holds by (A.27) and since $[\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) \neq \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1})]$. The inequality (A.33) has contradiction with (A.31).

As a result, by forming a set \mathcal{S} of indices i for which $h^*(\mathbf{x}_{i,1}) = 1$, we have

$$\begin{aligned} \min_{h \in \mathcal{H}_d} L_{0-1}^{\mu_X \mu_Y | X}(h) &= \sum_{i \notin \mathcal{S}} \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) + \sum_{i \in \mathcal{S}} \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1}) \\ &\quad + \sum_i \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) \\ &\quad + \min_{|\mathcal{S}|} \min_{h \in \mathcal{H}_{d-|\mathcal{S}|}} \sum_i \mathbb{I}_{h(\mathbf{x}_{i,2})=1} \mathbb{P}(\mathbf{x}_{i,2}) [\mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}) - \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2})] \end{aligned} \quad (\text{A.34})$$

$$\begin{aligned} &\stackrel{(a)}{=} \sum_i \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) + \sum_i \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) \\ &\quad + \min_{h \in \mathcal{H}_d} \sum_i \mathbb{I}_{h(\mathbf{x}_{i,2})=1} \mathbb{P}(\mathbf{x}_{i,2}) [\mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}) - \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2})] \end{aligned} \quad (\text{A.35})$$

$$\begin{aligned} &\stackrel{(b)}{=} \sum_i \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) + \sum_i \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) \\ &\quad - \max_{\mathcal{P}: |\mathcal{P}| \leq d} \sum_{i \in \mathcal{P}} \mathbb{P}(\mathbf{x}_{i,2}) [\mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}) - \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2})] \end{aligned} \quad (\text{A.36})$$

$$\begin{aligned} &\stackrel{(c)}{=} \sum_i \mathbb{P}(\mathbf{x}_{i,1}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) + \sum_i \mathbb{P}(\mathbf{x}_{i,2}) \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2}) \\ &\quad + \sum_{i=1}^d \mathbb{P}(\mathbf{x}_{i,2}) [\mathbb{P}(Y = 0|X = \mathbf{x}_{i,2}) - \mathbb{P}(Y = 1|X = \mathbf{x}_{i,2})], \end{aligned} \quad (\text{A.37})$$

where (a) holds using that for $i \in \mathcal{S}$ we have $[\mathbb{P}(Y = 1|X = \mathbf{x}_{i,1}) = \mathbb{P}(Y = 0|X = \mathbf{x}_{i,1})]$, and since $\mathcal{H}_{d-|\mathcal{S}|} \subseteq \mathcal{H}_d$. Further, (b) holds by the definition of \mathcal{H}_d in which $\text{supp}(h)$ is assumed to be bounded by d , and, (c) holds using the assumption (A.29). Finally, one could see that (A.37) is equal to (A.30). \square

Lemma 5. For an ordered probability mass function

$$p_1 \leq p_2 \leq \dots \leq p_n, \quad (\text{A.38})$$

on a finite set, and for $a \in \{1, \dots, n\}$, we have

$$\sum_{i=1}^a p_i \leq \frac{a}{n}. \quad (\text{A.39})$$

Proof. We prove this lemma by contradiction. Assume that

$$\sum_{i=1}^a p_i > \frac{a}{n}. \quad (\text{A.40})$$

Since p_i s are ordered, one could see that for all sets $\mathcal{S}_t \subseteq \{1, \dots, n\}$ with $|\mathcal{S}_t| = a$, we have

$$\sum_{i \in \mathcal{S}_t} p_i > \frac{a}{n}. \quad (\text{A.41})$$

We know that $\binom{n}{a}$ number of such distinct sets exist. Hence, we have

$$\sum_{t=1}^{\binom{n}{a}} \sum_{i \in \mathcal{S}_t} p_i > \frac{a}{n} \binom{n}{a}. \quad (\text{A.42})$$

Moreover, one could see that for each i , p_i is repeated in LHS of (A.42) for $\binom{n-1}{a-1}$ times. Consequently, we see that

$$\binom{n-1}{a-1} = \sum_{j=1}^{\binom{n-1}{a-1}} \sum_{i=1}^n p_i > \frac{a}{n} \binom{n}{a} = \binom{n-1}{a-1}, \quad (\text{A.43})$$

that is a contradiction. □

Proof of Theorem 1. We derive the lower- and upper-bound in two steps as follows.

• **Lower-bound:** To prove the lower-bound, for every hypothesis class \mathcal{H} and \mathcal{R} , we design a distribution of (x, y, m) such that

$$L_{\text{def}}^{0-1}(h^*, r^*) = 0, \quad (\text{A.44})$$

while

$$L_{\text{def}}^{0-1}(\tilde{h}, \tilde{r}) = \frac{1}{d(\mathcal{H}) + 1}. \quad (\text{A.45})$$

For every $d(\mathcal{H}) + 1$ samples $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_{d(\mathcal{H})+1})$, using the definition of VC dimension, we can find labels $\mathbf{y} = (y_1, \dots, y_{d(\mathcal{H})+1})$ such that no classifier $h \in \mathcal{H}$ can obtain them (i.e., there is no h such that $h(\mathbf{x}_i) = y_i$, for $i \in [1 : d(\mathcal{H}) + 1]$). We set

$$p(\mathbf{x}_i) = \begin{cases} \frac{1+\epsilon}{d(\mathcal{H})+1} & i = 1 \\ \frac{1}{d(\mathcal{H})+1} & i \in [2 : d(\mathcal{H})] \\ \frac{1-\epsilon}{d(\mathcal{H})+1} & i = d(\mathcal{H}) + 1 \end{cases}, \quad (\text{A.46})$$

$$p(y|\mathbf{x}_i) = \begin{cases} 1 & y = y_i, \\ 0 & \text{o.w.} \end{cases}, \quad (\text{A.47})$$

and

$$p(m|\mathbf{x}_i, y) = \begin{cases} 1 & m = y_i, i = 1, \\ 1 & m = 1 - y_i, i = d(\mathcal{H}) + 1, \\ 0 & \text{o.w.} \end{cases}. \quad (\text{A.48})$$

If we train \hat{h} and \hat{r} separately, it means

$$\hat{h} = \operatorname{argmin}_{h \in \mathcal{H}} \mathbb{E}_{(\mathbf{x}, y, m) \sim p} [\mathbb{I}_{h(\mathbf{x}) \neq y}] \quad (\text{A.49})$$

$$= \operatorname{argmin}_{h \in \mathcal{H}} \frac{1 + \epsilon}{d(\mathcal{H}) + 1} \mathbb{I}_{h(\mathbf{x}_1) \neq y_1} + \sum_{i=2}^{d(\mathcal{H})} \frac{1}{d(\mathcal{H}) + 1} \mathbb{I}_{h(\mathbf{x}_i) \neq y_i} + \frac{1 - \epsilon}{d(\mathcal{H}) + 1} \mathbb{I}_{h(\mathbf{x}_{d(\mathcal{H})+1}) \neq y_{d(\mathcal{H})+1}}. \quad (\text{A.50})$$

By the definition of \mathbf{y} , we know that at least one of the terms in the RHS of (A.50) is non-zero. In such case, for every subset T of $[1 : d(\mathcal{H}) + 1]$ of size $d(\mathcal{H})$, one can find $h \in \mathcal{H}$, such that $h(\mathbf{x}_i) = y_i$ for $i \in T$. Hence, to minimize RHS of (A.50), we should have $\hat{h}(\mathbf{x}_i) \neq y_i$ only for $i = d(\mathcal{H}) + 1$.

Further, \hat{r} is obtained as

$$\hat{r} = \operatorname{argmin}_{r \in \mathcal{R}} \mathbb{E}_{(\mathbf{x}, y, m) \sim p} [\mathbb{I}_{\hat{h}(\mathbf{x}) \neq y} \mathbb{I}_{r(\mathbf{x})=0} + \mathbb{I}_{m \neq y} \mathbb{I}_{r(\mathbf{x})=1}]. \quad (\text{A.51})$$

By the definition of $p(m|y, \mathbf{x})$ and \hat{h} , we can rewrite (A.51) as

$$\hat{r} = \operatorname{argmin}_{r \in \mathcal{R}} \frac{1 - \epsilon}{d(\mathcal{H}) + 1} \mathbb{I}_{r(\mathbf{x}_{d+1})=1} + \frac{1 - \epsilon}{d(\mathcal{H}) + 1} \mathbb{I}_{r(\mathbf{x}_{d+1})=0}. \quad (\text{A.52})$$

One can see that by any choice of $\hat{r}(\cdot)$, we have

$$L_{\text{def}}^{0-1}(\hat{h}, \hat{r}) = \frac{1 - \epsilon}{d(\mathcal{H}) + 1}. \quad (\text{A.53})$$

Finally, by the arbitrariness of ϵ , we have

$$L_{\text{def}}^{0-1}(\hat{h}, \hat{r}) = \frac{1}{d(\mathcal{H}) + 1}. \quad (\text{A.54})$$

Further, we prove that $L_{\text{def}}^{0-1}(h^*, r^*) = 0$ by constructing h^* and r^* . Since $d(\mathcal{R}) \geq 2$, we can shatter $\{\mathbf{x}_1, \mathbf{x}_{d(\mathcal{H})+1}\}$ by \mathcal{R} , which means that there exists $r^* \in \mathcal{R}$ such that $r^*(\mathbf{x}_1) = 1$, and $r^*(\mathbf{x}_{d(\mathcal{H})+1}) = 0$. As a result, we have

$$L_{\text{def}}^{0-1}(h^*, r^*) = \sum_{i=2}^{d(\mathcal{H})} \frac{1}{d(\mathcal{H}) + 1} \mathbb{I}_{r^*(\mathbf{x}_i)=0} \mathbb{I}_{h^*(\mathbf{x}_i) \neq y_i} + \frac{1 - \epsilon}{d(\mathcal{H}) + 1} \mathbb{I}_{h^*(\mathbf{x}_{d(\mathcal{H})+1}) \neq y_i}. \quad (\text{A.55})$$

Since VC dimension of \mathcal{H} is $d(\mathcal{H})$, we can find h^* such that all terms in the RHS of (A.55) is zero. Hence, we have $L_{\text{def}}^{0-1}(h^*, r^*) = 0$, that completes the proof.

• **Upper-bound:** For $d(\mathcal{H}) \leq d(\mathcal{R})$ the upper-bound is trivial. Then, we assume $d(\mathcal{H}) > d(\mathcal{R})$.

Let $D_{\mu_{XYM}}$ be

$$D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} = L_{\text{def}}^{\mu_X \mu_{Y|X}}(\hat{h}, \hat{r}) - L_{\text{def}}^{\mu_X \mu_{Y|X}}(h^*, r^*). \quad (\text{A.56})$$

To upper-bound $\inf_{\mathcal{H}, \mathcal{R}} \sup_{\mu_{XYM}} D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}}$, we find a pair of hypothesis classes \mathcal{H} and \mathcal{R} , such that for all joint probability measures μ_{XYM} , we have $D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} \leq \frac{d(\mathcal{R})}{d(\mathcal{H})}$.

We choose $\mathcal{H} = \mathcal{H}_{d(\mathcal{H})}$, and $\mathcal{R} = \mathcal{H}_{d(\mathcal{R})}$, where \mathcal{H}_d is defined in Lemma 3. One could check that $VC(\mathcal{H}) = d(\mathcal{H})$, and $VC(\mathcal{R}) = d(\mathcal{R})$. Further, using Lemma 2, we know that $D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}}$ is bounded by $D_{\mu'_{XYM}}^{\mathcal{H}, \mathcal{R}}$, in which μ'_X is purely atomic. For such measures, Lemma 3 proves that

$$D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} \leq \min_{h \in \mathcal{H}_{d(\mathcal{H})-d(\mathcal{R})}} L_{0-1}^{\mu'_X \mu_{Y|X}}(h) - \min_{h \in \mathcal{H}_{d(\mathcal{H})}} L_{0-1}^{\mu'_X \mu_{Y|X}}(h). \quad (\text{A.57})$$

As a result, we have

$$\sup_{\mu_{XYM}} D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} \leq \sup_{\mu_{XYM}: \mu_X \text{ atomic}} D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} \quad (\text{A.58})$$

$$\leq \sup_{\mu_{XY}: \mu_X \text{ atomic}} \min_{h \in \mathcal{H}_{d(\mathcal{H})-d(\mathcal{R})}} L_{0-1}^{\mu_X \mu_Y | X}(h) - \min_{h \in \mathcal{H}_{d(\mathcal{H})}} L_{0-1}^{\mu_X \mu_Y | X}(h). \quad (\text{A.59})$$

Next, by applying Lemma 4, we have

$$\sup_{\mu_{XYM}} D_{\mu_{XYM}}^{\mathcal{H}, \mathcal{R}} \leq \sup_{\mu_{XY}: \mu_X \text{ atomic}} \sum_{i=d(\mathcal{H})-d(\mathcal{R})+2}^{d(\mathcal{H})} \mathbb{P}(x_{i,2}) [\mathbb{P}(Y = 1|X = x_{i,2}) - \mathbb{P}(Y = 0|X = x_{i,2})], \quad (\text{A.60})$$

where $\{x_{i,2}\}_i$ are defined in Lemma 4.

Since $\mathbb{P}(Y = 1|X = x_{i,2}) > \mathbb{P}(Y = 0|X = x_{i,2})$ we could define

$$q_i = \frac{\mathbb{P}(x_{i,2}) [\mathbb{P}(Y = 1|X = x_{i,2}) - \mathbb{P}(Y = 0|X = x_{i,2})]}{\sum_{j=1}^{d(\mathcal{H})} \mathbb{P}(x_{j,2}) [\mathbb{P}(Y = 1|X = x_{j,2}) - \mathbb{P}(Y = 0|X = x_{j,2})]}. \quad (\text{A.61})$$

Then, by the definition of $x_{i,2}$ we know that

$$q_1 \geq q_2 \geq \dots \geq q_{d(\mathcal{H})}, \quad (\text{A.62})$$

and $\sum_{i=1}^{d(\mathcal{H})} q_i = 1$. Hence, using Lemma 5 we have

$$\frac{\sum_{j=d(\mathcal{H})-d(\mathcal{R})+1}^{d(\mathcal{H})} \mathbb{P}(x_{j,2}) [\mathbb{P}(Y = 1|X = x_{j,2}) - \mathbb{P}(Y = 0|X = x_{j,2})]}{\sum_{j=1}^{d(\mathcal{H})} \mathbb{P}(x_{j,2}) [\mathbb{P}(Y = 1|X = x_{j,2}) - \mathbb{P}(Y = 0|X = x_{j,2})]} = \sum_{j=d(\mathcal{H})-d(\mathcal{R})+1}^{d(\mathcal{H})} q_j \leq \frac{d(\mathcal{R})}{d(\mathcal{H})}, \quad (\text{A.63})$$

which concludes that

$$\begin{aligned} & \sum_{j=d(\mathcal{H})-d(\mathcal{R})+1}^{d(\mathcal{H})} \mathbb{P}(x_{j,2}) [\mathbb{P}(Y = 1|X = x_{j,2}) - \mathbb{P}(Y = 0|X = x_{j,2})] \\ & \leq \frac{d(\mathcal{R})}{d(\mathcal{H})} \sum_{j=1}^{d(\mathcal{H})} \mathbb{P}(x_{j,2}) [\mathbb{P}(Y = 1|X = x_{j,2}) - \mathbb{P}(Y = 0|X = x_{j,2})] \leq \frac{d(\mathcal{R})}{d(\mathcal{H})}. \end{aligned} \quad (\text{A.64})$$

This, together with (A.60) completes the proof. \square

A.2 Proof of Proposition 1

We will prove the following proposition from which Proposition 1 can be obtained from by re-arranging the terms.

Let $\mathcal{S}_l = \{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^{n_l}$ and $\mathcal{S}_u = \{(\mathbf{x}_{i+n_l}, y_{i+n_l})\}_{i=1}^{n_u}$ be two iid sample sets that are drawn from the joint distribution $P_{X,Y,M}$ and are labeled and not labeled by human, respectively. Assume that the optimal classifier $\bar{h} = \operatorname{argmin}_h \mathbb{E}_{X,Y \sim \mu_{XY}} [\mathbb{I}_{h(X) \neq Y}]$ is a member of \mathcal{H} (i.e., realizability). Then, with probability at least $1 - \delta$ we have

$$\begin{aligned} L_{\text{def}}^{0-1}(\hat{r}, \hat{h}) &\leq L_{0-1}(h^*, r^*) + \mathfrak{R}_{n_u}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{R}) \\ &+ 2 \min \left\{ \mathbb{P}(M \neq Y), \mathfrak{R}_{n_l \mathbb{P}(M \neq Y)/2}(\mathcal{R}) \right\} + C \sqrt{\frac{\log 1/\delta}{n_l}} + e^{-n_l \mathbb{P}(M \neq Y)^2/2} \\ &+ C' \sqrt{\frac{\log 1/\delta}{n_u}} \end{aligned} \tag{A.65}$$

where $h^*, r^* = \operatorname{argmin}_{(h,r) \in \mathcal{H} \times \mathcal{R}} L_{0-1}(h, r)$.

Compare this to using only \mathcal{S}_l to learn jointly \tilde{h}, \tilde{r} we get [26]¹:

$$\begin{aligned} L_{\text{def}}^{0-1}(\tilde{r}, \tilde{h}) &\leq L_{0-1}(h^*, r^*) + \mathfrak{R}_{n_l}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{R}) \\ &+ 2\mathfrak{R}_{n_l \mathbb{P}(M \neq Y)/2}(\mathcal{R}) + C' \sqrt{\frac{\log 1/\delta}{n_l}} \\ &+ \frac{\mathbb{P}(M \neq Y)}{2} e^{-n \mathbb{P}(M \neq Y)/2} \end{aligned} \tag{A.66}$$

We start by introducing some useful lemmas, and then we continue with the proof of proposition.

Lemma 6. Let $h^*(x) = \operatorname{argmin}_{h \in \mathcal{F}} L_{0-1}(h)$, where \mathcal{F} is the class of all functions $h(\cdot) : \mathcal{X} \rightarrow \mathcal{Y}$. Then, for every function $r(\cdot) : \mathcal{X} \rightarrow \{0, 1\}$, we have

$$\mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(x) \neq y}] \leq \mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h(x) \neq y}], \tag{A.67}$$

for all function $h(\cdot) \in \mathcal{F}$.

Proof. Since $h^*(\cdot)$ could be any function, it is easy to show that for $x \in \mathcal{D}$, where $\mathcal{D} = \{x :$

¹Note that in [26], they set the notation in a manner that $r \in \{-1, 1\}$. Hence, $\mathfrak{R}_n(\mathcal{R})$ under such notation is twice as much as the case in this chapter (i.e., $r \in \{0, 1\}$). Here, we express their results with our choice of notation.

$f_X(x) \neq 0$ }, we have

$$h^*(x) = \underset{v}{\operatorname{argmin}} \mathbb{E}_{Y|X=x} [\mathbb{I}_{v \neq Y}], \quad (\text{A.68})$$

which concludes that

$$\mathbb{E}_{Y|X=x} [\mathbb{I}_{h^*(x) \neq Y}] \leq \mathbb{E}_{Y|X=x} [\mathbb{I}_{h(x) \neq Y}], \quad (\text{A.69})$$

for all $h(\cdot) \in \mathcal{F}$. Hence, we have

$$\mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] = \mathbb{E}_X [\mathbb{I}_{r(X)=0} \mathbb{E}_{Y|X=x} [\mathbb{I}_{h^*(x) \neq Y}]] \quad (\text{A.70})$$

$$\leq \mathbb{E}_X [\mathbb{I}_{r(X)=0} \mathbb{E}_{Y|X=x} [\mathbb{I}_{h(x) \neq Y}]] \quad (\text{A.71})$$

$$= \mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y}], \quad (\text{A.72})$$

which completes the proof. □

Lemma 7. Let $h^*(x) = \underset{h \in \mathcal{F}}{\operatorname{argmin}} L_{0-1}(h)$, where \mathcal{F} is the class of all functions $h(\cdot) : \mathcal{X} \rightarrow \mathcal{Y}$. If we have $h^*(\cdot) \in \mathcal{H}$, then there exists $r \in \mathcal{R}$ such that the pair (h^*, r) is a minimizer of the optimization problem

$$\underset{(h,r) \in \mathcal{H} \times \mathcal{R}}{\operatorname{argmin}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}]. \quad (\text{A.73})$$

Proof. We prove this lemma by showing

$$\begin{aligned} \min_{(h,r) \in \mathcal{H} \times \mathcal{R}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}] \\ = \min_{r \in \mathcal{R}} \mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}]. \end{aligned} \quad (\text{A.74})$$

To show (A.74), using Lemma 6, we know that

$$\begin{aligned} \min_{(h,r) \in \mathcal{H} \times \mathcal{R}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}] \\ \geq \min_{(h,r) \in \mathcal{H} \times \mathcal{R}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}] \end{aligned} \quad (\text{A.75})$$

$$= \min_{r \in \mathcal{R}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}]. \quad (\text{A.76})$$

On the other hand, using the minimum property, one could show that

$$\begin{aligned} & \min_{h \in \mathcal{H}} \min_{r \in \mathcal{R}} \mathbb{E} \left[\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y} \right] \\ & \leq \min_{r \in \mathcal{R}} \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_{X,Y,M} [\mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y}]. \end{aligned} \quad (\text{A.77})$$

Hence, using the lower- and upper-bound in (A.76) and (A.77), one could show (A.74) and complete the proof. \square

Proof of Proposition 1. We prove (A.65) in three steps: (i) we bound the expected 0 – 1 loss of the classifier \hat{h} when deferral does not happen by a function of the optimal expected 0 – 1 loss in such cases, (ii) we bound the joint loss L_{def} by a function of the optimal joint loss and the Rademacher complexity of a hypothesis class, and (iii) we bound the Rademacher complexity of the aforementioned class by the Rademacher complexity of the deferral hypothesis class \mathcal{R} .

• **Step (i):** Using Rademacher inequality (Theorem 3.3 of [224]), with probability $1 - \delta/4$, we have

$$L_{0-1}(\hat{h}) \leq \hat{L}_{0-1}(\hat{h}) + 2\mathfrak{R}_{n_u}(\mathcal{G}) + \sqrt{\frac{\log 4/\delta}{2n_u}}, \quad (\text{A.78})$$

where $\mathcal{G} = \{\mathbf{x}, y \rightarrow \mathbb{I}_{h(\mathbf{x}) \neq y} : h \in \mathcal{H}\}$.

Furthermore, using (A.78), since \hat{h} is an optimizer of the empirical loss in \mathcal{H} and since $h^* \in \mathcal{H}$, with probability $1 - \delta/2$ we have

$$L_{0-1}(\hat{h}) \leq \hat{L}_{0-1}(h^*) + 2\mathfrak{R}_{n_u}(\mathcal{G}) + \sqrt{\frac{\log 4/\delta}{2n_u}} \quad (\text{A.79})$$

$$\stackrel{(a)}{\leq} L_{0-1}(h^*) + 2\mathfrak{R}_{n_u}(\mathcal{G}) + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}}, \quad (\text{A.80})$$

where (a) holds using McDiarmid's inequality, union bound, and by that the empirical loss is $\frac{2}{n}$ -bounded difference.

Next, using Lemma 3.4 of [224] we know that $\mathfrak{R}_n(\mathcal{G}) = \frac{1}{2}\mathfrak{R}_n(\mathcal{H})$. By means of such identity and (A.80), with probability $1 - \delta/2$ we have

$$L_{0-1}(\hat{h}) \leq L_{0-1}(h^*) + \mathfrak{R}_n(\mathcal{H}) + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}}. \quad (\text{A.81})$$

It remains to show that for each function $r(\cdot) : \mathcal{X} \rightarrow \{0, 1\}$, we could bound $\mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{\hat{h}(X) \neq Y}]$ by sum of $\mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}]$ and a term that is corresponded to the concentration of measure

for large sample size. For proving such inequality, first we know that

$$L_{0-1}(h^*) = \mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_X [\mathbb{I}_{r(X)=1} \mathbb{E}_{Y|X=\mathbf{x}} [\mathbb{I}_{h^*(X) \neq Y}]] \quad (\text{A.82})$$

$$\stackrel{(a)}{\leq} \mathbb{E} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_X [\mathbb{I}_{r(X)=1} \mathbb{E}_{Y|X=\mathbf{x}} [\mathbb{I}_{h(X) \neq Y}]], \quad (\text{A.83})$$

for all $h \in \mathcal{F}$, where (a) is followed by Lemma 6.

Using (A.81) and (A.83), we have

$$\begin{aligned} & \mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{\hat{h}(X) \neq Y}] + \mathbb{E}_X [\mathbb{I}_{r(X)=1} \mathbb{E}_{Y|X=\mathbf{x}} [\mathbb{I}_{\hat{h}(X) \neq Y}]] \\ & \leq \mathbb{E} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathbb{E}_X [\mathbb{I}_{r(X)=1} \mathbb{E}_{Y|X=\mathbf{x}} [\mathbb{I}_{\hat{h}(X) \neq Y}]] + \mathfrak{R}_{n_u}(\mathcal{H}) + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}}, \end{aligned} \quad (\text{A.84})$$

which concludes

$$\mathbb{E}_{X,Y} [\mathbb{I}_{r(X)=0} \mathbb{I}_{\hat{h}(X) \neq Y}] \leq \mathbb{E} [\mathbb{I}_{r(X)=0} \mathbb{I}_{h^*(X) \neq Y}] + \mathfrak{R}_{n_u}(\mathcal{H}) + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}}. \quad (\text{A.85})$$

• **Step (ii):** We know that $\hat{r}(\cdot)$ is obtained as

$$\hat{r}(x) = \operatorname{argmin}_{r \in \mathcal{R}} \frac{1}{n_l} \sum_{i=1}^{n_l} [\mathbb{I}_{r(\mathbf{x}_i)=0} \mathbb{I}_{h(\mathbf{x}_i) \neq y_i} + \mathbb{I}_{r(\mathbf{x}_i)=1} \mathbb{I}_{m_i \neq y_i}], \quad (\text{A.86})$$

or equivalently,

$$\hat{r}(x) = \operatorname{argmin}_{r \in \mathcal{R}} \frac{1}{n_l} \sum_{i=1}^{n_l} [\mathbb{I}_{r(\mathbf{x}_i)=0} [\mathbb{I}_{h(\mathbf{x}_i) \neq y_i} - \mathbb{I}_{m_i \neq y_i}]]. \quad (\text{A.87})$$

Hence, using Rademacher inequality (Theorem 3.3 of [224]), with probability $1 - \delta/4$, we have

$$\mathbb{E}_{X,Y,M} [\mathbb{I}_{\hat{r}(X)=0} [\mathbb{I}_{\hat{h}(X) \neq Y} - \mathbb{I}_{M \neq Y}]] \leq \frac{1}{n_l} \sum_{i=1}^{n_l} \mathbb{I}_{\hat{r}(\mathbf{x}_i)=0} [\mathbb{I}_{\hat{h}(\mathbf{x}_i) \neq y_i} - \mathbb{I}_{m_i \neq y_i}] + 2\mathfrak{R}_{n_l}(\mathcal{J}) + \sqrt{\frac{\log 4/\delta}{n_l}}, \quad (\text{A.88})$$

where

$$\mathcal{J} = \{\mathbf{x}, y, m \rightarrow \mathbb{I}_{r(\mathbf{x})=0} [\mathbb{I}_{\hat{h}(\mathbf{x}) \neq y} - \mathbb{I}_{m \neq y}] : r \in \mathcal{R}\}. \quad (\text{A.89})$$

Using Lemma 7, we know that there exists $r^* \in \mathcal{R}$ such that (r^*, h^*) are the minimizers of the joint loss $L_{\text{def}}^{0-1}(h, r)$ in $\mathcal{H} \times \mathcal{R}$. Next, since \hat{r} is the minimizer of the empirical joint loss given the

classifier be \hat{h} , and using (A.88), we have

$$\mathbb{E}_{X,Y,M} [\mathbb{I}_{\hat{r}(X)=0} [\mathbb{I}_{\hat{h}(X) \neq Y} - \mathbb{I}_{M \neq Y}]] \leq \frac{1}{n_l} \sum_{i=1}^{n_l} \mathbb{I}_{r^*(\mathbf{x}_i)=0} [\mathbb{I}_{\hat{h}(\mathbf{x}_i) \neq y_i} - \mathbb{I}_{m_i \neq y_i}] + 2\mathfrak{R}_{n_l}(\mathcal{J}) + \sqrt{\frac{\log 4/\delta}{n_l}}, \quad (\text{A.90})$$

for $r^*(\cdot)$ defined as above.

Next, using McDiarmid's inequality, union bound, and since the empirical loss in RHS of (A.90) is $\frac{2}{n}$ -bounded difference, then with probability at least $1 - \delta/2$ we have

$$\begin{aligned} \mathbb{E}_{X,Y,M} [\mathbb{I}_{\hat{r}(X)=0} [\mathbb{I}_{\hat{h}(X) \neq Y} - \mathbb{I}_{M \neq Y}]] &\leq \mathbb{E}_{X,Y,M} [\mathbb{I}_{r^*(X)} [\mathbb{I}_{\hat{h}(X) \neq Y} - \mathbb{I}_{M \neq Y}]] + 2\mathfrak{R}_n(\mathcal{J}) \\ &\quad + (\sqrt{2} + 1) \sqrt{\frac{\log 4/\delta}{n_l}}. \end{aligned} \quad (\text{A.91})$$

Therefore, using step (i), and by means of union bound, one could prove that with probability at least $1 - \delta$ we have

$$\begin{aligned} \mathbb{E}_{X,Y,M} [\mathbb{I}_{\hat{r}(X)} [\mathbb{I}_{\hat{h}(X) \neq Y} - \mathbb{I}_{M \neq Y}]] &\leq \mathbb{E}_{X,Y,M} [\mathbb{I}_{r^*(X)} [\mathbb{I}_{h^*(X) \neq Y} - \mathbb{I}_{M \neq Y}]] + \mathfrak{R}_{n_u}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{J}) \\ &\quad + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}} + (\sqrt{2} + 1) \sqrt{\frac{\log 4/\delta}{n_l}}, \end{aligned} \quad (\text{A.92})$$

or equivalently

$$L_{\text{def}}^{0-1}(\hat{r}, \hat{h}) \leq L_{\text{def}}^{0-1}(h^*, r^*) + \mathfrak{R}_{n_u}(\mathcal{H}) + 2\mathfrak{R}_{n_l}(\mathcal{J}) + \frac{3\sqrt{2}}{2} \sqrt{\frac{\log 4/\delta}{n_u}} + (\sqrt{2} + 1) \sqrt{\frac{\log 4/\delta}{n_l}}, \quad (\text{A.93})$$

• **Step (iii):** In this step, we bound $\mathfrak{R}_n(\mathcal{G})$ to complete the proof. By recalling the definition of \mathcal{J} in

(A.89), we bound $\mathfrak{R}_n(\mathcal{J})$ as

$$\mathfrak{R}_n(\mathcal{J}) = \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sup_{g \in \mathcal{J}} \sum_{i=1}^n \sigma_i g(\mathbf{x}_i, y_i, m_i) \right] \quad (\text{A.94})$$

$$= \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sup_{r \in \mathcal{R}} \sum_{i=1}^n \sigma_i [\mathbb{I}_{r(\mathbf{x}_i)=0} (\mathbb{I}_{\hat{h}(\mathbf{x}_i) \neq y_i} - \mathbb{I}_{m_i \neq y_i})] \right] \quad (\text{A.95})$$

$$\stackrel{(a)}{\leq} \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sup_{r \in \mathcal{R}} \sum_{i=1}^n \sigma_i [r(\mathbf{x}_i) \mathbb{I}_{\hat{h}(\mathbf{x}_i) \neq y_i}] \right] \\ + \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sup_{r \in \mathcal{R}} \sum_{i=1}^n \sigma_i [r(X) \mathbb{I}_{m_i \neq y_i}] \right] \quad (\text{A.96})$$

$$\stackrel{(b)}{\leq} \mathfrak{R}_n(\mathcal{R}) + \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{1}{n} \sum_{i=1}^n \sigma_i \mathbb{I}_{\hat{h}(\mathbf{x}_i) \neq y_i} \right] + \underbrace{\mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \mathbb{E}_{\boldsymbol{\sigma}} \left[\frac{\sum_{i=1}^n \mathbb{I}_{m_i \neq y_i}}{n} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{R}) \right]}_A \quad (\text{A.97})$$

$$\stackrel{(c)}{=} \mathfrak{R}_n(\mathcal{R}) + A, \quad (\text{A.98})$$

where $\mathcal{S} = \{(\mathbf{x}_i, y_i, m_i) : m_i \neq y_i\}$ and (a) holds because of sub-linearity of supremum, (b) holds by sub-linearity of supremum, since for two events E_1 and E_2 we have $\mathbb{I}_{E_1} \cdot \mathbb{I}_{E_2} \leq \mathbb{I}_{E_1} + \mathbb{I}_{E_2}$, and using Lemma 3.4 of [224], and (c) is followed by σ_i being zero-mean.

Now, we should bound A . Since $u = \sum_{i=1}^n \mathbb{I}_{m_i \neq y_i}$ is a random variable with distribution Binomial($n, \mathbb{P}(M \neq Y)$) and using Hoeffding's inequality, we have

$$\mathbb{P}\left(\frac{u}{n} < \mathbb{P}(M \neq Y) - t\right) \leq e^{-2nt^2}. \quad (\text{A.99})$$

Next, by decomposing A , we have

$$A = \mathbb{P}\left(\frac{u}{n} < \mathbb{P}(M \neq Y) - t\right) \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \left[\frac{u}{n} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{R}) \mid \frac{u}{n} < \mathbb{P}(M \neq Y) - t \right] \\ + \mathbb{P}\left(\frac{u}{n} \geq \mathbb{P}(M \neq Y) - t\right) \mathbb{E}_{\{(\mathbf{x}_i, y_i, m_i)\}_{i=1}^n} \left[\frac{u}{n} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{R}) \mid \frac{u}{n} \geq \mathbb{P}(M \neq Y) - t \right] \quad (\text{A.100})$$

$$\leq |\mathbb{P}(M \neq Y) - t| e^{-2nt^2} + \min\{\mathbb{P}(M \neq Y), \mathfrak{R}_{n(\mathbb{P}(M \neq Y) - t)}(\mathcal{R})\}, \quad (\text{A.101})$$

where the inequality holds since Rademacher complexity is bounded by 1 and is non-increasing in terms of sample-space size, followed by $\frac{u}{n} \leq 1$, and by means of Lemma 3.4 of [224]. As a result, by setting $t = \frac{\mathbb{P}(M \neq Y)}{2}$, we have

$$A \leq \frac{\mathbb{P}(M \neq Y)}{2} e^{-\frac{n\mathbb{P}^2(M \neq Y)}{2}} + \min\{\mathbb{P}(M \neq Y), \mathfrak{R}_{n\mathbb{P}(M \neq Y)/2}(\mathcal{R})\}. \quad (\text{A.102})$$

Finally using (A.93), (A.98), and (A.102) we complete the proof. \square

A.3 Proof of Proposition 2

To prove the consistency of the deferral surrogate, we know that since ℓ_ϕ is consistent, for every $\{p_1, \dots, p_{k+1}\}$, such that $\sum_{i=1}^{k+1} p_i = 1$, we have

$$\operatorname{argmax}_{i \in [k+1]} \operatorname{argmin}_{h \in \mathcal{D}} \sum_{i=1}^{k+1} p_i \tilde{\ell}_\phi(i, h) = \operatorname{argmax}_{i \in [k+1]} p_i. \quad (\text{A.103})$$

(One could prove this by setting $\mathbb{P}(X = x) = \delta[x]$, and $\mathbb{P}(Y = y|X = x) = p_y$.)

Next, we find the minimizer of the loss $\tilde{\ell}_\phi$ as

$$\operatorname{argmin}_{h \in \mathcal{F}} \mathbb{E}_{X,Y,M} [\tilde{\ell}_\phi(\mathbf{c}, h)] = \operatorname{argmin}_{h(x)} \mathbb{E}_{Y,M|X=x} [\tilde{\ell}_\phi(\mathbf{c}, h(x))] \quad (\text{A.104})$$

$$= \operatorname{argmin}_{h(x)} \sum_{i=1}^{k+1} \mathbb{E} [\max_{j \in [k+1]} c(j) - c(i) | X = x] \tilde{\ell}_\phi(i, h(x)). \quad (\text{A.105})$$

Next, we form the probability mass function $\{q_1, \dots, q_{k+1}\}$ as

$$q_i = \frac{\mathbb{E} [\max_{j \in [k+1]} c(j) - c(i)]}{\sum_{t=1}^{k+1} \mathbb{E} [\max_{j \in [k+1]} c(j) - c(t)]}. \quad (\text{A.106})$$

One could see that the optimizer in (A.105) is equivalent to

$$\operatorname{argmin}_{h(x)} \sum_{i=1}^{k+1} q_i \ell_\phi(i, h(x)). \quad (\text{A.107})$$

Now, using (A.103) and (A.107), we can show that

$$\operatorname{argmax}_{i \in [k+1]} \operatorname{argmin}_{h \in \mathcal{F}} \mathbb{E}_{X,Y,M} [\tilde{\ell}_\phi(\mathbf{c}, h)] = \operatorname{argmax}_{i \in [k+1]} q_i = \operatorname{argmin}_{i \in [k+1]} \mathbb{E}[c(i)|X = x]. \quad (\text{A.108})$$

The above identity means that $h_{k+1}(x) \geq \max_{i \in [k]} h_i(x)$ (i.e., $r(x) = 1$) iff. we have $\mathbb{E}[c(k+1)|X = x] \leq \min_{i \in [k]} \mathbb{E}[c(i)|X = x]$. Further, we have

$$h(x) = \operatorname{argmax}_{i \in [k]} h_i(x) = \operatorname{argmin}_{i \in [k]} \mathbb{E}[c(i)|X = x]. \quad (\text{A.109})$$

Recalling Proposition 1 in [26], one sees that $r(x)$ and $h(x)$ are that of Bayesian optimal classifier, which proves that $\tilde{\ell}_\phi$ is Fisher consistent.

A.4 Proof of Theorem 2

To show the result for the calibration function, by setting $\mathbb{P}(X = x) = \delta[x']$, and $\mathbb{P}(Y = y|X = x') = p_y$ for $y \in [k + 1]$, we see that

$$L^{0-1}(h) - \min_{h \in \mathcal{F}} L^{0-1}(h) = \sum_{i \neq h(x')} p_i - \sum_{i \neq \arg \max p_i} p_i \quad (\text{A.110})$$

$$= \max_{i \in [k+1]} p_i - p_{h(x')}. \quad (\text{A.111})$$

Furthermore, we have

$$\tilde{L}_\phi(h) - \min_{h \in \mathcal{F}} \tilde{L}_\phi(h) = \sum_{i=1}^{k+1} p_i [\tilde{\ell}_\phi(i, h(x')) - \tilde{\ell}_\phi(i, h^*)]. \quad (\text{A.112})$$

Hence, ψ being a calibration function proves that

$$\psi(\max p_i - p_{h(x')}) \leq \sum_{i=1}^{k+1} p_i [\tilde{\ell}_\phi(i, h(x')) - \tilde{\ell}_\phi(i, h^*)], \quad (\text{A.113})$$

for every choice of $h(x')$.

On the other hand, one could calculate the conditional cost-sensitive loss as

$$L_{c,x}(h) = \mathbb{E}_{Y|X=x} [c(h(X))] = \sum_{i \neq h(x)} \mathbb{E}_{Y|X=x} [c(i)|X = x]. \quad (\text{A.114})$$

Hence, we have

$$L_{c,x}(h) - L_{c,x}(h^*) = \mathbb{E}[c(h(X))|X = x] - \min_{i \in [k+1]} \mathbb{E}[c(i)|X = x], \quad (\text{A.115})$$

where $h^* = \operatorname{argmin}_{h \in \mathcal{F}} L_c(h)$.

By defining q_i s as (A.106), one can prove that

$$L_{c,x}(h) - L_{c,x}(h^*) = \sum_{i=1}^{k+1} [\max_{j \in [k+1]} c(j) - c(i)|X = x] (\max q_i - q_{h(x)}). \quad (\text{A.116})$$

For the new surrogate, we further know that

$$\tilde{L}_{\mathbf{c},x}(h) = \mathbb{E}_{Y|X=x}[\tilde{\ell}(\mathbf{c}, h(x))] \quad (\text{A.117})$$

$$= \sum_{i=1}^{k+1} \mathbb{E} \left[\max_{j \in [k+1]} c(j) - c(i) | X = x \right] \sum_{i=1}^{k+1} q_i \tilde{\ell}_\phi(i, h(x)). \quad (\text{A.118})$$

Furthermore, one could show that

$$\tilde{h}_1^{k+1} = \underset{h \in \mathcal{F}}{\operatorname{argmin}} \tilde{L}_{\mathbf{c},x}(h) \quad (\text{A.119})$$

$$= \underset{h \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{k+1} q_i \tilde{\ell}_\phi(i, h), \quad (\text{A.120})$$

and consequently,

$$\tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1}) = \sum_{i=1}^{k+1} \mathbb{E} \left[\max_{j \in [k+1]} c(j) - c(i) | X = x \right] \cdot \sum_{i=1}^{k+1} q_i (\tilde{\ell}_\phi(i, h) - \tilde{\ell}_\phi(i, \tilde{h}_1^{k+1})). \quad (\text{A.121})$$

Hence, using (A.113) and (A.120), we have

$$\mathbb{E} \left[\max_{j \in [k+1]} c(j) - c(i) | X = x \right] \psi \left(\max_{i \in [k+1]} q_i - q_{h(x)} \right) \leq \tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1}). \quad (\text{A.122})$$

Hence, since $\psi(x) = C|x|^\epsilon$, we have

$$\psi(L_{\mathbf{c},x}(h) - L_{\mathbf{c},x}(h^*)) = \psi \left(\mathbb{E} \left[\max_{j \in [k+1]} c(j) - c(i) | X = x \right] \left(\max_{i \in [k+1]} q_i - q_{h(x)} \right) \right) \quad (\text{A.123})$$

$$\leq \mathbb{E}^{\epsilon-1} \left[\max_{j \in [k+1]} c(j) - c(i) | X = x \right] (\tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1})) \quad (\text{A.124})$$

$$\stackrel{(a)}{\leq} M^{\epsilon-1} (\tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1})), \quad (\text{A.125})$$

where (a) holds using the assumption of the theorem.

Finally, using convexity of ψ and by Jensen's inequality, we have

$$\psi(L_{\mathbf{c}}(h) - L_{\mathbf{c}}(h^*)) = \psi(\mathbb{E}_X[L_{\mathbf{c},x}(h) - L_{\mathbf{c},x}(h^*)]) \quad (\text{A.126})$$

$$\leq \mathbb{E}_X \psi(L_{\mathbf{c},x}(h) - L_{\mathbf{c},x}(h^*)) \quad (\text{A.127})$$

$$\stackrel{(a)}{\leq} M^{\epsilon-1} \mathbb{E}_X [\tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1})] \quad (\text{A.128})$$

$$= M^{\epsilon-1} (\tilde{L}_{\mathbf{c},x}(h) - \tilde{L}_{\mathbf{c},x}(\tilde{h}_1^{k+1})), \quad (\text{A.129})$$

in which (a) is followed by (A.125). This completes the proof of the first part of theorem.

To obtain the calibration function of the cross-entropy error, we first introduce the following lemma.

Lemma 8. *For every two distributions P and G , we have*

$$|\max_i P_i - \max_i G_i| \leq \sqrt{2D_{KL}(P\|G)}. \quad (\text{A.130})$$

Proof. We define $\operatorname{argmax}_i G_i = i_G^*$, and $\operatorname{argmax}_i P_i = i_P^*$. If we have $\max_i G_i = G_{i_G^*} \geq G_{i_P^*} = \max_i P_i$, then

$$0 \leq G_{i_G^*} - P_{i_P^*} \stackrel{(a)}{\leq} G_{i_G^*} - P_{i_G^*} \stackrel{(b)}{\leq} \sqrt{2D_{KL}(P\|G)}, \quad (\text{A.131})$$

where (a) is correct due to the fact that $\max_i P_i \geq P_{i_G^*}$, and (b) holds due to Pinsker's inequality. Further, if we have $\max_i G_i = P_{i_G^*} \leq P_{i_P^*} = \max_i P_i$, using a similar argument, we have

$$0 \leq P_{i_P^*} - G_{i_G^*} \leq P_{i_P^*} - G_{i_P^*} \leq \sqrt{2D_{KL}(P\|G)}. \quad (\text{A.132})$$

□

Next, we note that the conditional surrogate risk can be rewritten as

$$\tilde{L}_{CE,x}(g_1, \dots, g_{K+1}) = - \sum_{i=1}^{K+1} \mathbb{E} \left[\max_{j \in [K+1]} c(j) - c(i) | X = x \right] \log \frac{\exp(g_i(x))}{\sum_k \exp(g_k(x))} \quad (\text{A.133})$$

$$= N_x H_{\mathcal{P}_x}(\mathcal{G}_x), \quad (\text{A.134})$$

where $N_x = \sum_{i=1}^{K+1} \mathbb{E} [\max_{j \in [K+1]} c(j) - c(i) | X = x]$, $H_{\mathcal{P}_x}(\mathcal{G}_x)$ refers to the relative entropy of the distribution \mathcal{G}_x w.r.t \mathcal{P}_x which are defined as

$$\mathcal{P}_{x,i} = \frac{\mathbb{E} [\max_{j \in [K+1]} c(j) - c(i) | X = x]}{N_x}, \quad (\text{A.135})$$

and

$$\mathcal{G}_{x,i} = \frac{\exp(g_i(x))}{\sum_k \exp(g_k(x))}. \quad (\text{A.136})$$

Secondly, one note that since in the minimizer of surrogate risk

$$\operatorname{argmin}_{\mathbf{g} \in \mathcal{F}} \tilde{L}_{CE}(\mathbf{g}),$$

\mathcal{F} contains every function, hence there is no dependency between different point x s, and as a result, the minimization is equivalent to finding minimize every conditional surrogate risk. More formally, if g_1^*, \dots, g_{K+1}^* are such pair of minimizers, we have

$$(g_1^*(x), \dots, g_{K+1}^*(x)) = \operatorname{argmin}_{g_1(x), \dots, g_{K+1}(x)} \tilde{L}_{CE,x}(g_1(x), \dots, g_{K+1}(x)) \quad (\text{A.137})$$

$$\stackrel{(a)}{=} \operatorname{argmin}_{g_1(x), \dots, g_{K+1}(x)} N_x H_{\mathcal{P}_x}(\mathcal{G}_x) \quad (\text{A.138})$$

$$= \operatorname{argmin}_{g_1(x), \dots, g_{K+1}(x)} H_{\mathcal{P}_x}(\mathcal{G}_x) \quad (\text{A.139})$$

$$\stackrel{(b)}{=} (\mathcal{P}_{x,1}, \dots, \mathcal{P}_{x,K+1}), \quad (\text{A.140})$$

where (a) holds because of (A.134), and (b) is a property of relative entropy.

As a result, the conditional excess surrogate risk can be rewritten as

$$\tilde{L}_{CE,x}(g_1, \dots, g_{K+1}) - \tilde{L}_{CE,x}^* = N_x H_{\mathcal{P}_x}(\mathcal{G}_x) - N_x H_{\mathcal{P}_x}(\mathcal{P}_x) = N_x D_{KL}(\mathcal{P}_x, \mathcal{G}_x). \quad (\text{A.141})$$

Further, we can write the conditional excess risk as

$$\begin{aligned} L_x^{0-1}(g_1, \dots, g_{K+1}) - L_x^{0-1}(g_1^*, \dots, g_{K+1}^*) \\ = \mathbb{E} \left[c(\operatorname{argmax}_{i(x)} g_{i(x)}(x) | X = x) - \min_{i(x)} \mathbb{E} \left[c(i(x)) | X = x \right] \right], \end{aligned} \quad (\text{A.142})$$

where L_x^{0-1} is defined as

$$L_x^{0-1}(g_1, \dots, g_{K+1}) = \mathbb{E} \left[\mathbb{I}_{Y \neq \operatorname{argmax}_{i \in [K+1]} g_i(X)} | X = x \right] \quad (\text{A.143})$$

Next, we can rewrite this conditional excess risk in terms of $\mathcal{P}_{x,i}$ s as

$$L_x^{0-1}(g_1, \dots, g_{K+1}) - L_x^{0-1}(g_1^*, \dots, g_{K+1}^*) = N_x \left(\max_{i(x)} \mathcal{P}_{x,i(x)} - \mathcal{P}_{x, \operatorname{argmax}_{i(x)} g_{i(x)}(x)} \right) \quad (\text{A.144})$$

$$= N_x \left(\max_{i(x)} \mathcal{P}_{x,i(x)} - \mathcal{P}_{x, \operatorname{argmax}_{i(x)} \mathcal{G}_{x,i(x)}} \right). \quad (\text{A.145})$$

To bound such a value, we use Pinsker's inequality which states that for every two distributions P and Q supported on \mathbb{N} , we have

$$TV(P, Q) = \frac{1}{2} \sum_i |P_i - Q_i| \leq \sqrt{\frac{D_{KL}(P||Q)}{2}}. \quad (\text{A.146})$$

To make use of that inequality, by defining $i_{\mathcal{P}_x} := \operatorname{argmax}_{i(x)} \mathcal{P}_{x,i(x)}$ and $i_{\mathcal{G}_x} := \operatorname{argmax}_{i(x)} \mathcal{G}_{x,i(x)}$ and using triangle inequality, we know that

$$N_x \left| \max_{i(x)} \mathcal{P}_{x,i(x)} - \mathcal{P}_{x,\operatorname{argmax}_{i(x)} \mathcal{G}_{x,i(x)}} \right| \leq N_x \left| \mathcal{P}_{x,i_{\mathcal{P}_x}} - \mathcal{G}_{x,i_{\mathcal{G}_x}} \right| + N_x \left| \mathcal{G}_{x,i_{\mathcal{G}_x}} - \mathcal{P}_{x,i_{\mathcal{G}_x}} \right|. \quad (\text{A.147})$$

Next, we bound each of these terms separately. Firstly, we know that

$$N_x \left| \mathcal{G}_{x,i_{\mathcal{G}_x}} - \mathcal{P}_{x,i_{\mathcal{G}_x}} \right| \leq N_x \sum_i \left| \mathcal{P}_{x,i} - \mathcal{G}_{x,i} \right| = N_x TV(\mathcal{P}_x \| \mathcal{G}_x) \quad (\text{A.148})$$

$$\leq N_x \sqrt{2D_{KL}(\mathcal{P}_x \| \mathcal{G}_x)}. \quad (\text{A.149})$$

Further, using Lemma 8, one can show that

$$N_x \left| \mathcal{P}_{x,i_{\mathcal{P}_x}} - \mathcal{G}_{x,i_{\mathcal{G}_x}} \right| \leq N_x \sqrt{2D_{KL}(\mathcal{P}_x \| \mathcal{G}_x)}. \quad (\text{A.150})$$

As a result, we have

$$L_x^{0-1}(g_1, \dots, g_{K+1}) - L_x^{0-1}(g_1^*, \dots, g_{K+1}^*) \leq N_x \sqrt{8D_{KL}(\mathcal{P}_x \| \mathcal{G}_x)} \quad (\text{A.151})$$

$$= \sqrt{8N_x} \sqrt{\tilde{L}_{CE,x}(g_1, \dots, g_{K+1}) - \tilde{L}_{CE,x}^*}, \quad (\text{A.152})$$

where the last equality is followed by (A.141). Next, using the upper-bound on $c(i)$ s, we have $N_x \leq 2KM$. As a result, we have

$$\frac{\left(L_x^{0-1}(g_1, \dots, g_{K+1}) - L_x^{0-1}(g_1^*, \dots, g_{K+1}^*) \right)^2}{16MK} \leq \tilde{L}_{CE,x}(g_1, \dots, g_{K+1}) - \tilde{L}_{CE,x}^*. \quad (\text{A.153})$$

Finally, using Jensen's inequality, we have

$$\frac{\left(L_x^{0-1}(g_1, \dots, g_{K+1}) - L_x^{0-1}(g_1^*, \dots, g_{K+1}^*) \right)^2}{16MK} \leq \tilde{L}_{CE}(g_1, \dots, g_{K+1}) - \tilde{L}_{CE}^*, \quad (\text{A.154})$$

which yields the statement of theorem.

A.5 Proof of Theorem 3

We first introduce some useful lemmas, then we get back to the proof of theorem.

Lemma 9. *Let $\mathcal{F}_1, \dots, \mathcal{F}_k$ be hypothesis classes with Rademacher complexity $\hat{\mathfrak{R}}_S(\mathcal{F}_1), \dots, \hat{\mathfrak{R}}_S(\mathcal{F}_k)$ on set S . The Rademacher complexity of the hypothesis class $\mathcal{G} = \{\log \sum_{i=1}^k e^{f_i(x)} : f_i(\cdot) \in \mathcal{F}_i\}$*

on set \mathcal{S} is bounded as

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{G}) \leq \sum_{i=1}^k \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_i). \quad (\text{A.155})$$

Proof. We prove this lemma for $k = 2$. By following similar steps, one could generalize this proof for every $k \in \mathbb{N}$.

We write the Rademacher complexity of \mathcal{G} as

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{G}) = \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2} \sum_{i=1}^m \sigma_i \log(e^{f_1(x)} + e^{f_2(x)}) \right] \quad (\text{A.156})$$

$$= \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2} \sum_{i=1}^m \frac{\sigma_i}{2} f_1 + \sum_{i=1}^m \frac{\sigma_i}{2} f_2 + \sum_{i=1}^m \sigma_i \log(e^{f_1(x)/2 - f_2(x)/2} + e^{f_2(x)/2 - f_1(x)/2}) \right] \quad (\text{A.157})$$

$$\stackrel{(a)}{\leq} \frac{1}{2m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_1 \in \mathcal{F}_1} \sum_{i=1}^m \sigma_i f_1 \right] + \frac{1}{2m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_2 \in \mathcal{F}_2} \sum_{i=1}^m \sigma_i f_2 \right] + \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sum_{i=1}^m \sigma_i \log(e^{f_1(x)/2 - f_2(x)/2} + e^{f_2(x)/2 - f_1(x)/2}) \right] \quad (\text{A.158})$$

$$= \frac{1}{2} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_1) + \frac{1}{2} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_2) + \hat{\mathfrak{R}}_{\mathcal{S}}(\Phi \circ (\mathcal{F}_1 - \mathcal{F}_2)), \quad (\text{A.159})$$

where (a) is followed by the sublinearity of supremum, and $\Phi(\cdot)$ is defined as $\Phi(x) = \log(e^{x/2} + e^{-x/2})$.

One could see that $\frac{\partial \Phi(x)}{\partial x} = \frac{1}{2} \frac{e^{x/2} - e^{-x/2}}{e^{x/2} + e^{-x/2}} \leq \frac{1}{2}$, that leads to $\frac{1}{2}$ -Lipschitzness of $\Phi(\cdot)$. Using this, and by Ledoux-Talagrand theorem [225], we have

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\Phi \circ (\mathcal{F}_1 - \mathcal{F}_2)) \leq \frac{1}{2} \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_1 - \mathcal{F}_2) \quad (\text{A.160})$$

$$= \frac{1}{2m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2} \sum_{i=1}^m \sigma_i (f_1(x) - f_2(x)) \right] \quad (\text{A.161})$$

$$\stackrel{a}{\leq} \frac{1}{2m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_1 \in \mathcal{F}_1} \sum_{i=1}^m \sigma_i f_1(x) \right] + \frac{1}{2m} \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f_2 \in \mathcal{F}_2} \sum_{i=1}^m \sigma_i f_2(x) \right] \quad (\text{A.162})$$

$$= \frac{1}{2} (\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_1) + \hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{F}_2)), \quad (\text{A.163})$$

where (a) is again followed by sublinearity of supremum.

Finally, using (A.159) and (A.163), we complete the proof. \square

Lemma 10. Let \mathcal{F} be a hypothesis class of functions $f(x, y) : \mathcal{X} \times [k + 1] \rightarrow \mathbb{R}$, and $\Pi_1(\mathcal{F}) =$

$\{x \rightarrow f(x, y) : f(\cdot, \cdot) \in \mathcal{F}, y \in [k+1]\}$. Then,

- for $\mathcal{G} = \{x, y \rightarrow f(x, y) - \log \sum_{j=1}^{k+1} f(x, y) : f(\cdot, \cdot) \in \mathcal{F}\}$ and given the assumption that for every label inside sets of pairs $(x_i, y_i) \in \mathcal{S}$ is within the range $\{1, \dots, k\}$, we have

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{G}) \leq (2k+1)\hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})), \quad (\text{A.164})$$

- and for $\mathcal{H}_i = \{x \rightarrow f(x, i) - \log \sum_{y=1}^{k+1} f(x, y) : f(\cdot, \cdot) \in \mathcal{F}\}$, we have

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{H}_i) \leq (k+2)\hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})). \quad (\text{A.165})$$

Proof. 1. We write Rademacher complexity of \mathcal{G} as

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{G}) = \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \sigma_i f(x_i, y_i) - \sigma_i \log \sum_{y=1}^{k+1} e^{f(x_i, y)} \right] \quad (\text{A.166})$$

$$\stackrel{(a)}{\leq} \underbrace{\frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \sigma_i f(x_i, y_i) \right]}_A + \underbrace{\frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \sigma_i \log \sum_{y=1}^{k+1} e^{f(x_i, y)} \right]}_B, \quad (\text{A.167})$$

where (a) holds because of sublinearity of supremum. Next, we bound A and B as follows.

First, we know that

$$A = \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{y=1}^k \sum_{i=1}^m \sigma_i f(x_i, y) \mathbb{I}_{y_i=y} \right] \quad (\text{A.168})$$

$$\leq \sum_{y=1}^k \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \sigma_i f(x_i, y) (\epsilon_i/2 + 1/2) \right], \quad (\text{A.169})$$

where $\epsilon_i = 2\mathbb{I}_{y_i=y} - 1$. Hence, again, applying sublinearity of supremum, we have

$$A \leq \sum_{y=1}^k \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \frac{\sigma_i \epsilon_i}{2} f(x_i, y) \right] + \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \frac{\sigma_i}{2} f(x_i, y) \right]. \quad (\text{A.170})$$

Since $\epsilon \in \{-1, 1\}$, then $\sigma_i \epsilon_i$ take Rademacher distribution as well. Hence, using (A.170), we

have

$$A \leq \sum_{y=1}^k \frac{1}{2} \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})) + \frac{1}{2} \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})) \quad (\text{A.171})$$

$$= k \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})). \quad (\text{A.172})$$

Next, to bound B , using Lemma 9, we have

$$B \leq \sum_{y=1}^{k+1} \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^m \sigma_i f(x_i, y) \right] \leq \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})). \quad (\text{A.173})$$

Finally, using (A.167), (A.172), and (A.173), we complete the proof.

2. We bound Rademacher complexity of \mathcal{H}_i as

$$\hat{\mathfrak{R}}_{\mathcal{S}_x}(\mathcal{H}_i) = \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{j=1}^m \sigma_j f(x_j, i) - \sigma_j \log \sum_{y=1}^{k+1} e^{f(x_j, y)} \right] \quad (\text{A.174})$$

$$\stackrel{(a)}{\leq} \frac{1}{m} \mathbb{E} \left[\sup_{f \in \mathcal{F}} \sum_{j=1}^m \sigma_j f(x_j, i) \right] + \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{j=1}^m \sigma_j \log \sum_{y=1}^{k+1} e^{f(x_j, y)} \right] \quad (\text{A.175})$$

$$\stackrel{(b)}{\leq} \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})) + \sum_{y=1}^{k+1} \frac{1}{m} \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \sum_{j=1}^m f(x_j, y) \right] \quad (\text{A.176})$$

$$\leq (k+2) \hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{F})), \quad (\text{A.177})$$

where (a) is followed by sublinearity of supremum, and (b) because of Lemma 9 and using definition of $\Pi_1(\mathcal{F})$. □

Lemma 11. For $i \in \{1, \dots, k+1\}$ let \mathcal{H}_i be hypothesis class of functions $h_i(x) : \mathcal{X} \rightarrow \mathbb{R}$ with bounded norm $\|h_i\|_{\infty} < C$. Further, let $\Pi_1(\mathcal{H}) = \{x \rightarrow h_i(x) : h_i \in \mathcal{H}_i, i \in [k+1]\}$. The Rademacher complexity of the class \mathcal{L} of loss functions

$$\ell(x, y, m) = -\log \frac{e^{-h_y(x)}}{\sum_{i=1}^{k+1} e^{-h_i(x)}} - \mathbb{I}_{m \neq y} \log \frac{e^{-h_{k+1}(x)}}{\sum_{i=1}^{k+1} e^{-h_i(x)}}, \quad (\text{A.178})$$

for $m, y \in [k]$ is bounded as

$$\begin{aligned} \mathfrak{R}_n(\mathcal{L}) &\leq (k+1)\mathfrak{R}_n(\Pi_1(\mathcal{H})) + (k+2) \min\{\mathbb{P}(M \neq Y), \mathfrak{R}_{n\mathbb{P}(M \neq Y)/2}(\Pi_1(\mathcal{H}))\} \\ &\quad + \frac{C}{2}\mathbb{P}(M \neq Y)(k+2)e^{-n\mathbb{P}^2(M \neq Y)/2}. \end{aligned} \quad (\text{A.179})$$

Proof. We write empirical Rademacher complexity of \mathcal{L} as

$$\hat{\mathfrak{R}}_{\mathcal{S}}(\mathcal{L}) = \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{\ell \in \mathcal{L}} \sum_{i=1}^n \sigma_i \ell(x_i, y_i, m_i) \right] \quad (\text{A.180})$$

$$= \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{h_j \in \mathcal{H}_j} \sum_{i=1}^n \sigma_i \left(-\log \frac{e^{-h_y(x_i)}}{\sum_{j=1}^{k+1} e^{-h_j(x_i)}} - \mathbb{I}_{m \neq y} \log \frac{e^{-h_{k+1}(x_i)}}{\sum_{j=1}^{k+1} e^{-h_j(x_i)}} \right) \right] \quad (\text{A.181})$$

$$\leq \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{h_j \in \mathcal{H}_j} \sum_{i=1}^n \sigma_i \log \frac{e^{-h_y(x_i)}}{\sum_{j=1}^{k+1} e^{-h_j(x_i)}} \right] + \frac{1}{n}\mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{h_j \in \mathcal{H}_j} \sum_{i=1, y_i \neq m_i}^n \sigma_i \log \frac{e^{-h_{k+1}(x_i)}}{\sum_{j=1}^{k+1} e^{-h_j(x_i)}} \right] \quad (\text{A.182})$$

$$\stackrel{(a)}{\leq} (k+1)\hat{\mathfrak{R}}_{\mathcal{S}_x}(\Pi_1(\mathcal{H})) + (k+2) \frac{\sum_{i=1}^n \mathbb{I}_{m_i \neq y_i}}{n} \hat{R}_{\mathcal{S}_x | m_i \neq y_i}(\Pi(\mathcal{H})), \quad (\text{A.183})$$

where (a) holds by applying Lemma 10.

Using A.183, and by calculating the expectation over $\{(x_i, y_i, m_i)\}_{i=1}^n$, we have

$$\mathfrak{R}_n(\mathcal{L}) \leq (k+1)\mathfrak{R}_n(\Pi_1(\mathcal{H})) + (k+2) \underbrace{\mathbb{E}_{\{(x_i, y_i, m_i)\}_{i=1}^n} \left[\frac{\sum_{i=1}^n \mathbb{I}_{m_i \neq y_i}}{n} \hat{\mathfrak{R}}_{\mathcal{S}_x | m_i \neq y_i}(\Pi_1(\mathcal{H})) \right]}_A. \quad (\text{A.184})$$

It is remained to bound A . For this task, we first notice that $u = \sum_{i=1}^n \mathbb{I}_{m_i \neq y_i}$ is a random variable with distribution $\text{Binomial}(n, \mathbb{P}(M \neq Y))$. Further, by Hoeffding's inequality we know that for $t > 0$, we have

$$\mathbb{P}\left(\frac{u}{n} < \mathbb{P}(M \neq Y) - t\right) \leq e^{-2nt^2}. \quad (\text{A.185})$$

Hence, by decomposing A , we have

$$\begin{aligned} A &= \mathbb{P}\left(\frac{u}{n} < \mathbb{P}(M \neq Y) - t\right) \mathbb{E}\left[\frac{u}{n} \hat{\mathfrak{R}}_{\mathcal{S}_x | y_i \neq m_i} \mid \frac{u}{n} < \mathbb{P}(M \neq Y) - t\right] \\ &\quad + \mathbb{P}\left(\frac{u}{n} \geq \mathbb{P}(M \neq Y) - t\right) \mathbb{E}\left[\frac{u}{n} \hat{\mathfrak{R}}_{\mathcal{S}_x | y_i \neq m_i} \mid \frac{u}{n} \geq \mathbb{P}(M \neq Y) - t\right] \end{aligned} \quad (\text{A.186})$$

$$\leq C|\mathbb{P}(M \neq Y) - t|e^{-2nt^2} + \min\{\mathbb{P}(M \neq Y), \mathfrak{R}_{n(\mathbb{P}(M \neq Y) - t)}(\Pi_1(\mathcal{H}))\}, \quad (\text{A.187})$$

where the last inequality holds because (1) every function in $\Pi_1(\mathcal{H})$ is bound by C , and so is the Rademacher complexity of $\Pi_1(\mathcal{H})$, and (2) the Rademacher complexity is non-increasing with the sample space size.

Hence, using A.184, A.187, and by setting $t = \frac{\mathbb{P}(M \neq Y)}{2}$ the proof is complete. \square

Proof of Theorem 3. Using Rademacher inequality on generalization error (e.g., Theorem 3.3 of [224]), we know that with probability at least $1 - \delta/2$, we have

$$\tilde{L}_{CE}(\mathbf{f}_1^{k+1}) \leq \hat{L}_{CE}(\mathbf{f}_1^{k+1}) + 2\mathfrak{R}_n(\mathcal{L}) + \sqrt{\frac{D \log 2/\delta}{2n}}, \quad (\text{A.188})$$

where D is an upper-bound on $\|\ell\|_\infty$ for $\ell \in \mathcal{L}$, and where \hat{L}_{CE} is the empirical loss corresponding to ℓ_{CE} , and $f_i \in \mathcal{F}_i$.

We follow the proof in three steps, (i) we find D , (ii) we find a lower-bound on $\tilde{L}_{CE}(\mathbf{f}_1^{k+1})$ in terms of $L_{\text{def}}^{0-1}(\mathbf{f}_1^{k+1})$, and (iii) we complete the proof by bounding the difference $|\min_{\mathbf{f} \in \mathcal{F}_1^{k+1}} \hat{L}_{CE}(\mathbf{f}_1^{k+1}) - \min_{\mathbf{f}_1^{k+1} \in \mathcal{F}} \tilde{L}_{CE}(\mathbf{f}_1^{k+1})|$.

• **Step (i):** For calculating a bound on $\|\ell\|_\infty$ for $\ell \in \mathcal{L}$, we use boundedness of $\|f_i\|_\infty$ for $i \in [k+1]$ and $\mathbf{f}_1^{k+1} \in \mathcal{F}_1^{k+1}$. Indeed, we know the function

$$b_D(x) = -\log \frac{e^{-x}}{e^{-x} + D}, \quad (\text{A.189})$$

for $D > 0$ is a monotonically non-increasing function of x . Hence, over a closed interval, it takes the minimum and maximum on the limit points. As a result, for $|x| \leq C$, we have

$$0 \leq b_D(x) \leq -\log \frac{e^{-C}}{e^{-C} + D}. \quad (\text{A.190})$$

Hence, for the loss function $\ell(x, y, m)$, in which $\|\mathbf{f}_1^{k+1}\| \leq C$, we have

$$0 < \ell(x, y, m) = b_{\sum_{i=1, i \neq y}^{k+1} e^{-f_i(x)}}(f_y(x)) + \mathbb{I}_{m \neq y} b_{\sum_{i=1}^k e^{-f_i(x)}}(f_{k+1}(x)) \quad (\text{A.191})$$

$$\leq -\log \frac{e^{-C}}{e^{-C} + \sum_{i=1, i \neq y}^{k+1} e^{-f_i(x)}} - \mathbb{I}_{m \neq y} \log \frac{e^{-C}}{e^{-C} + \sum_{i=1}^k e^{-f_i(x)}} \quad (\text{A.192})$$

$$-2 \log \frac{e^{-C}}{e^{-C} + k e^C} \leq -2 \log \frac{e^{-2C}}{k+1} = 4C - 2 \log(k+1). \quad (\text{A.193})$$

• **Step (ii):** Using excessive surrogate risk bound, we see that

$$\psi(L_{\text{def}}^{0-1}(\mathbf{f}_1^{k+1}) - \min_{\mathbf{h}_1^{k+1}} L_{\text{def}}^{0-1}(\mathbf{h}_1^{k+1})) + \min_{\mathbf{h}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1}) \leq \tilde{L}_{CE}(\mathbf{f}_1^{k+1}). \quad (\text{A.194})$$

• **Step (iii):** In this step, we find a bound on $\hat{L}_{CE}(\mathbf{f}_1^{k+1}) - \min_{\mathbf{h}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1})$. Indeed, we know that

$$\begin{aligned} \hat{L}_{CE}(\mathbf{f}_1^{k+1}) - \min_{\mathbf{h}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1}) &= \hat{L}_{CE}(\mathbf{f}_1^{k+1}) - \underbrace{\min_{\mathbf{h}_1^{k+1} \in \mathcal{F}_1^{k+1}} \hat{L}_{CE}(\mathbf{h}_1^{k+1})}_{e_{\min}} \\ &+ \min_{\mathbf{h}_1^{k+1} \in \mathcal{F}_1^{k+1}} \hat{L}_{CE}(\mathbf{h}_1^{k+1}) - \min_{\mathbf{h}_1^{k+1} \in \mathcal{F}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1}) \\ &+ \underbrace{\min_{\mathbf{h}_1^{k+1} \in \mathcal{F}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1}) - \min_{\mathbf{h}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1})}_{e_{\phi\text{-appr}}} \end{aligned} \quad (\text{A.195})$$

$$\leq \hat{L}_{CE}(\tilde{\mathbf{h}}_1^{k+1}) - \tilde{L}_{CE}(\tilde{\mathbf{h}}_1^{k+1}) + e_{\min} + e_{\phi\text{-appr}}, \quad (\text{A.196})$$

where $\tilde{\mathbf{h}}_1^{k+1} = \underset{\mathbf{h}_1^{k+1} \in \mathcal{F}_1^{k+1}}{\operatorname{argmin}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1})$. Hence, using Hoeffding's inequality, with probability at least $1 - \delta/2$, we have

$$\hat{L}_{CE}(\mathbf{f}_1^{k+1}) - \min_{\mathbf{h}_1^{k+1}} \tilde{L}_{CE}(\mathbf{h}_1^{k+1}) \leq \sqrt{\frac{D}{2n} \log 2/\delta} + e_{\min} + e_{\phi\text{-appr}}. \quad (\text{A.197})$$

Finally, using Lemma 11, A.188, A.193, A.194, A.197, and by union bound, we complete the proof. \square

A.6 Proof of Proposition 3

We prove this proposition in four steps: (i) we first prove that in each iteration, the deferral loss $L_{\text{def}}^{0-1}(h, r)$ is bounded, (ii) using Step (i), we show that $\mathbb{P}(X \in \text{DIS}(V_i))$ halves in each iteration with high probability, (iii) using Step (ii) we conclude that $\mathbb{P}(X \in \text{DIS}(V_{\lceil \log \frac{1}{\epsilon} \rceil})) \leq \epsilon$ with high probability, and finally (iv) we provide a bound on $L_{\text{def}}^{0-1}(h, r)$ using the result in Step (iii).

• **Step (i):** We use Theorem 2 of [26] that making use of realizability of (h, r) on empirical distribution shows that with probability at least $1 - \delta'$ we have

$$\mathbb{E}[\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y} | X \in \text{DIS}(V_i)] \leq \sqrt{\frac{2 \log 2/\delta'}{m_i}} + \sqrt{\frac{2d(\mathcal{H}) \log \frac{em_i}{d(\mathcal{H})}}{m_i}} + \sqrt{\frac{32d(\mathcal{R}) \log \frac{em_i}{d(\mathcal{R})}}{m_i}}, \quad (\text{A.198})$$

where m_i is the size of the set on which human provides the prediction in each iteration. Note that we draw only samples from $\text{DIS}(V_i)$, and that is the reason that we condition the loss on X being in $\text{DIS}(V_i)$.

To analyze the sample complexity that corresponds to (A.198), we let $\delta' = \frac{\delta}{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}$ and we

assume that

$$m_i \geq \max\left\{108\Theta^2 \log \frac{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}{\delta}, 360\Theta^2 d(\mathcal{H}) \log \Theta, 2, 276\Theta^2 d(\mathcal{H}) \log \Theta\right\}. \quad (\text{A.199})$$

Using the first term in RHS of (A.199), we bound the first term in the upper-bound (A.198) as

$$\sqrt{\frac{2 \log \frac{2}{\delta'}}{m_i}} \leq \sqrt{\frac{2 \log \frac{2(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}{\delta}}{108\Theta^2 \log \frac{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}{\delta}}} = \frac{1}{6\Theta} \sqrt{\frac{2}{3} + \frac{2}{3 \log \frac{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}{\delta}}}. \quad (\text{A.200})$$

Then, for $i \leq \lceil \log \frac{1}{\epsilon} \rceil$ and since $\delta \leq 1$, we know that $\log \frac{4}{\delta} \geq 2$, which concludes that

$$\sqrt{\frac{2 \log \frac{2}{\delta'}}{m_i}} \leq \frac{1}{6\Theta}. \quad (\text{A.201})$$

Further, using the second and third term in RHS of (A.199) and since $\sqrt{\frac{2d(\mathcal{H}) \log em_i}{m_i}}$ is monotonically decreasing for $m_i \geq 2$ (note that $\frac{\partial}{\partial x} \left(\frac{\log x}{x}\right) = \frac{1}{x^2} - \frac{\log x}{x^2} \leq 0$ for $x \geq 2$) we have

$$\sqrt{\frac{2d(\mathcal{H}) \log em_i}{m_i}} \leq \sqrt{\frac{2d(\mathcal{H}) \log \frac{e\Theta^2 d(\mathcal{H}) \log \Theta}{d(\mathcal{H})}}{360\Theta^2 d(\mathcal{H}) \log \Theta}} = \sqrt{\frac{2 \log (e\Theta^2 \cdot \log \Theta)}{360\Theta^2 \log \Theta}} \quad (\text{A.202})$$

$$= \frac{1}{\Theta} \sqrt{\frac{\log e + 2 \log \Theta + \log \log \Theta}{180 \log \Theta}}. \quad (\text{A.203})$$

If we set $\Theta \geq e$, we have $\log \Theta \geq \log e$, and since $\log \log \Theta \leq \log \Theta$ for $\Theta \geq 1$, we have

$$\sqrt{\frac{2d(\mathcal{H}) \log \frac{em_i}{d(\mathcal{H})}}{m_i}} \leq \frac{1}{\Theta} \sqrt{\frac{5 \log \Theta}{180 \log \Theta}} = \frac{1}{6\Theta}. \quad (\text{A.204})$$

Similarly, we could show that

$$\sqrt{\frac{32d(\mathcal{R}) \log \frac{em_i}{d(\mathcal{R})}}{m_i}} \leq \frac{1}{6\Theta}, \quad (\text{A.205})$$

which together with (A.198), (A.201), and (A.204) proves that for $m_i = O(\Theta^2(d(\mathcal{H}) \log \Theta + d(\mathcal{R}) \log \Theta + \log \frac{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}{\delta}))$ we have

$$\mathbb{E}[\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y} | X \in DIS(V_i)] \leq \frac{1}{2\Theta}, \quad (\text{A.206})$$

with probability at least $1 - \frac{\delta}{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}$.

Since $X \in DIS(V_i)$ is a necessary condition for $\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y} = 1$, we conclude that

$$L_{\text{def}}^{0-1}(h, r) = \Delta(V_i) \mathbb{E}[\mathbb{I}_{r(X)=0} \mathbb{I}_{h(X) \neq Y} + \mathbb{I}_{r(X)=1} \mathbb{I}_{M \neq Y} | X \in DIS(V_i)] \leq \frac{\Delta(V_i)}{2\Theta}, \quad (\text{A.207})$$

with probability at least $1 - \frac{\delta}{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}$, where $\Delta(V_i)$ is defined as

$$\Delta(V_i) := \mathbb{P}(X \in DIS(V_i)) \quad (\text{A.208})$$

• **Step (ii):** Since $L_{\text{def}}^{0-1}(h, r) = \mathbb{P}(r(X)M + (1 - r(X))h(X) \neq Y)$, and because $L_{\text{def}}^{0-1}(h^*, r^*) = 0$, and using Step (i), we have

$$\mathbb{P}(r(X)M + (1 - r(X))h(X) \neq r^*(X)M + (1 - r^*(X))h^*(X)) \leq \frac{\Delta(V_i)}{2\Theta}, \quad (\text{A.209})$$

with probability at least $1 - \frac{\delta}{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2}$. As a result, for all $(h, r) \in V_{i+1}$, we have $(h, r) \in B((h^*, r^*), \frac{\Delta(V_i)}{2\Theta})$ with such probability.

Hence, we have

$$\Delta(V_{i+1}) \leq \Delta\left(B((h^*, r^*), \frac{\Delta(V_i)}{2\Theta})\right) \leq \Theta \cdot \frac{\Delta(V_i)}{2\Theta} = \frac{\Delta(V_i)}{2}, \quad (\text{A.210})$$

where the last inequality is followed by the definition of Θ .

• **Step (iii):** Using union bound, and since

$$\sum_{i=1}^{\lceil \log \frac{1}{\epsilon} \rceil} \frac{\delta}{(2 + \lceil \log \frac{1}{\epsilon} \rceil - i)^2} = \sum_{i=2}^{\lceil \log \frac{1}{\epsilon} \rceil + 1} \frac{\delta}{i^2} \leq \sum_{i=2}^{\infty} \frac{\delta}{i^2} = \frac{\pi^2 - 6}{6} \cdot \delta \leq \delta, \quad (\text{A.211})$$

and using Step (iii), we have that

$$\Delta(V_{\lceil \log \frac{1}{\epsilon} \rceil}) \leq \frac{1}{2^{\lceil \log \frac{1}{\epsilon} \rceil}} \Delta(V_0) \leq \epsilon, \quad (\text{A.212})$$

with probability at least $1 - \delta$.

• **Step (iv):** Since we know that $L_{\text{def}}^{0-1}(h^*, r^*) = 0$, we conclude that

$$\mathbb{P}(M \neq Y, r^*(X) = 1) = 0. \quad (\text{A.213})$$

Next, since for all $h \in V_{\lceil \log \frac{1}{\epsilon} \rceil}$ we have $\mathbb{P}(h(X) \neq Y, r(X) = 0) = 0$, we can show that

$$L_{\text{def}}^{0-1}(h, r) = \mathbb{P}(h(X) \neq Y, r(X) = 0) + \mathbb{P}(M \neq Y, r(X) = 1) \quad (\text{A.214})$$

$$= \mathbb{P}(M \neq Y, r(X) = 1) \quad (\text{A.215})$$

$$= \mathbb{P}(M \neq Y, r(X) = 1, r^*(X) = 0) + \mathbb{P}(M \neq Y, r(X) = 1, r^*(X) = 1) \quad (\text{A.216})$$

$$\stackrel{(a)}{=} \mathbb{P}(M \neq Y, r(X) = 1, r^*(X) = 0) \quad (\text{A.217})$$

$$= \mathbb{P}(M \neq Y, r(X) \neq r^*(X), r^*(X) = 0) \leq \mathbb{P}(r(X) \neq r^*(X)), \quad (\text{A.218})$$

where (a) is followed by (A.213).

Next, since (h^*, r^*) is not removed in any iteration because of its consistency, we have $(h^*, r^*) \in V_{\lceil \log \frac{1}{\epsilon} \rceil}$. Hence using Step (iii), for all $(h, r) \in V_{\lceil \log \frac{1}{\epsilon} \rceil}$ we have

$$\mathbb{P}(r(X) \neq r^*(X)) \leq \mathbb{P}(X \in \text{DIS}(V_{\lceil \log \frac{1}{\epsilon} \rceil})) \leq \epsilon, \quad (\text{A.219})$$

with probability at least $1 - \delta$.

Using (A.218) and (A.219) the proof is complete.

A.7 An example on which CAL algorithm fails

Here, we provide the reader with an example on which vanilla CAL algorithm in Section 2.6.1 does not converge. Let $\mathcal{X} = \{0, 1\}$ and let $X \sim \text{Uniform}\{\mathcal{X}\}$ and $\mu_{XYM} = \mu_X \mathbb{I}_{Y=X} \mathbb{I}_{M=0}$, which means for all instances on \mathcal{X} , $Y = X$ and $M = 0$. Further, let $\mathcal{H} = \{h_1, h_2\}$, and $\mathcal{R} = \{r_1, r_2\}$, where

$$h_1(\mathbf{x}) = r_1(\mathbf{x}) = \mathbf{x}, \quad h_2(\mathbf{x}) = r_2(\mathbf{x}) = 0. \quad (\text{A.220})$$

One could see that in this case three pairs $(h_1, r_1), (h_1, r_2), (h_2, r_1)$ as deferral systems provide zero loss.

To run CAL, we draw a sample from μ_X . Assume that we observe $\mathbf{x} = 1$. We see that since $\mathbf{x} \in \text{DIS}(V_0) = \{1\}$, then we need to query human's prediction and true label on such instance. Hence, we collect the corresponding values $y = m = 1$ for that instance. Next, we update the version space

$$V_1 = \{(h_1, r_1), (h_1, r_2), (h_2, r_1)\}, \quad (\text{A.221})$$

to induce consistency. However, we note that $\text{DIS}(V_1)$ does not change comparing to $\text{DIS}(V_0)$.

Hence, $\mathbb{P}(X \in DIS(V_0)) = \mathbb{P}(X \in DIS(V_1)) = \dots = \frac{1}{2}$. As a result, CAL algorithm does not converge, and in each iteration queries human prediction for $\mathbf{x} = 1$.

A.8 Proof of Theorem 4

Using Theorem 5.1 of [58], we know that if $n_l = C\Theta d(\mathcal{D}) \log\left(\frac{4\Theta}{\delta} \log \frac{4}{\epsilon}\right) \log \frac{4}{\epsilon}$, then with probability at least $1 - \frac{\delta}{4}$ we have

$$\mathbb{P}[f(X) \neq \mathbb{I}_{M \neq Y}] \leq \frac{\epsilon}{4}. \quad (\text{A.222})$$

Next, we bound the empirical joint loss on unlabeled samples. We know that

$$\hat{L}_{\text{def}}^{0-1}(h, r) = \frac{1}{n_u} \sum_i \mathbb{I}_{h(x_i) \neq y_i} \mathbb{I}_{r(x_i)=0} + \mathbb{I}_{m_i \neq y_i} \mathbb{I}_{r(x_i)=1} \quad (\text{A.223})$$

$$= \frac{1}{n_u} \sum_i [\mathbb{I}_{h(x_i) \neq y_i} \mathbb{I}_{r(x_i)=0} + f(x_i) \mathbb{I}_{r(x_i)=1}] + \frac{1}{n_u} \sum_i (\mathbb{I}_{m_i \neq y_i} - f(x_i)) \mathbb{I}_{r(x_i)=1} \quad (\text{A.224})$$

$$\stackrel{(a)}{=} \frac{1}{n_u} \sum_i (\mathbb{I}_{m_i \neq y_i} - f(x_i)) \mathbb{I}_{r(x_i)=1} \quad (\text{A.225})$$

$$\leq \frac{1}{n_u} \sum_i |\mathbb{I}_{m_i \neq y_i} - f(x_i)| \quad (\text{A.226})$$

$$= \frac{1}{n_u} \sum_i \mathbb{I}_{f(x_i) \neq \mathbb{I}_{m_i \neq y_i}} \quad (\text{A.227})$$

where (a) holds because of Line 5 in Algorithm 1.

As a result, we use Hoeffding's inequality coupled with (A.222) to show that

$$\hat{L}_{\text{def}}^{0-1}(h, r) \leq \frac{\epsilon}{4} + \sqrt{\frac{\log 2/\delta}{2n_u}}, \quad (\text{A.228})$$

with probability at least $1 - \frac{3\delta}{4}$. Further, by generalization bound in Theorem 2 of [26], with probability at least $1 - \frac{\delta}{4}$ we have

$$L_{\text{def}}^{0-1}(h, r) \leq \hat{L}_{\text{def}}^{0-1}(h, r) + \sqrt{\frac{2 \log 8/\delta}{n_u}} + \mathfrak{R}_{n_u}(\mathcal{H}) + 4\mathfrak{R}_{n_u}(\mathcal{R}) + \mathbb{P}(M \neq Y) e^{\frac{-n_u \mathbb{P}(M \neq Y)}{8}}, \quad (\text{A.229})$$

where following (A.228) we conclude that with probability at least $1 - \delta$ we have

$$L_{\text{def}}^{0-1}(h, r) \leq \frac{\epsilon}{4} + \sqrt{\frac{\log 2/\delta}{2n_u}} + \sqrt{\frac{2 \log 8/\delta}{n_u}} + \mathfrak{R}_{n_u}(\mathcal{H}) + 8\mathfrak{R}_{n_u}(\mathcal{R}) + \mathbb{P}(M \neq Y)e^{-\frac{n\mathbb{P}(M \neq Y)}{8}}. \quad (\text{A.230})$$

One can further calculate an upper-bound on $\mathfrak{R}_{n_u}(\mathcal{H})$ and $\mathfrak{R}_{n_u}(\mathcal{R})$ using Corollary 3.8 and 3.18 of [224] as

$$\mathfrak{R}_{n_u}(\mathcal{H}) \leq \sqrt{\frac{2d(\mathcal{H}) \log \frac{en_u}{d(\mathcal{H})}}{n_u}}, \quad (\text{A.231})$$

and

$$\mathfrak{R}_{n_u}(\mathcal{R}) \leq \sqrt{\frac{2d(\mathcal{R}) \log \frac{en_u}{d(\mathcal{R})}}{n_u}}, \quad (\text{A.232})$$

which by substituting in (A.230) we conclude that

$$L_{\text{def}}^{0-1}(h, r) \leq \frac{\epsilon}{4} + \sqrt{\frac{\log \frac{2}{\delta}}{2n_u}} + \sqrt{\frac{2 \log \frac{8}{\delta}}{n_u}} + \sqrt{\frac{2d(\mathcal{H}) \log \frac{en_u}{d(\mathcal{H})}}{n_u}} + \sqrt{\frac{32d(\mathcal{R}) \log \frac{en_u}{d(\mathcal{R})}}{n_u}} + \mathbb{P}(M \neq Y)e^{-\frac{n\mathbb{P}(M \neq Y)}{8}}. \quad (\text{A.233})$$

Finally, using (A.233) and by letting $n_u \geq \max\left\{\frac{8 \log \frac{2}{\delta}}{\epsilon^2}, \frac{288 \log 8/\delta}{\epsilon^2}, \frac{C' \max\{d(\mathcal{H}), d(\mathcal{R})\} \log \frac{1}{\epsilon}}{\epsilon^2}\right\}$ in which $C' = 2^{10}$ and for $\epsilon \leq \frac{1}{2^{18}e^4}$, we have

$$L_{\text{def}}^{0-1}(h, r) \leq \epsilon, \quad (\text{A.234})$$

with probability at least $1 - \delta$, which completes the proof.

A.9 Experimental Details

Data. We use the CIFAR validation set of 10k images as the test set and split the CIFAR training set 90/10 for training and validation.

Optimization. We use the AdamW optimizer [226] with learning rate 0.001 and default parameters on PyTorch. We also use a cosine annealing learning rate scheduler and train for 100 epochs and saving the best performing model on the validation set. For the surrogate L_{CE}^α [26], we perform a

search for α over a grid $[0, 0.1, 0.5, 1]$.

Model Complexity. For the model complexity gap figure, we use a convolutional neural network consisting of two convolutional layers with a max pooling layer in between followed by three fully connected layers with ReLU activations. We modify respectively: the number of channels produced by the convolution of the first layer and of the second layer, and the number of units in the first and second fully connected layers. We use this set of parameters to produce the plot for the classifier model:

$[[1, 1, 50, 25], [3, 3, 50, 25], [4, 4, 80, 40], [6, 6, 100, 50], [12, 12, 100, 50],$
 $[20, 20, 100, 50], [100, 100, 500, 250], [100, 100, 1000, 500]]$

For the rejector model, and for the expert confidence model used for Staged we use the parameters $[100, 100, 1000, 500]$. The error bars in the plot are produced by repeating the training process 10 times and obtaining standard deviations to average over the randomness in training. We used a rather simple network architecture so that we can more easily illustrate the model complexity gap, as more complex architectures can easily obtain $\sim 100\%$ accuracy on CIFAR and would not allow us to have a more fine-grained analysis of the gap.

Data Trade-Offs. We use the model parameters $[100, 100, 1000, 500]$ for all networks in this plot. For each fraction of data labeled, we sample randomly from the training set the corresponding number of points. The error bars are obtained by repeating the training process 10 times for different random samplings of the training set.

Appendix B

Additional Information for Chapter 3

B.1 Practitioner’s guide to our approach

B.1.1 MILP

We implement the MILP (3.9)-(3.14) in the binary setting using the Gurobi Optimizer [65] in Python.

```
class MILPDefer:
    def __init__(self, n_classes, time_limit=-1, add_regularization=False,
                 lambda_reg=1, verbose=False):
        self.n_classes = n_classes
        self.time_limit = time_limit
        self.verbose = verbose
        self.add_regularization = add_regularization
        self.lambda_reg = lambda_reg

    def fit(self, dataloader_train, dataloader_val, dataloader_test):
        self.fit_binary(dataloader_train, dataloader_val, dataloader_test)

    def fit_binary(self, dataloader_train, dataloader_val, dataloader_test):
        data_x = dataloader_train.dataset.tensors[0]
        data_y = dataloader_train.dataset.tensors[1]
        human_predictions = dataloader_train.dataset.tensors[2]

        C = 1
        gamma = 0.00001
        Mi = C + gamma
        Ki = C + gamma
        max_data = len(data_x)
```

```

hum_preds = 2*np.array(human_predictions) - 1
# add extra dimension to x
data_x_original = torch.clone(data_x)
norm_scale = max(torch.norm(data_x_original, p=1, dim=1))
last_time = time.time()
# normalize data_x and then add dimension
data_x = torch.cat((torch.ones((len(data_x)), 1),
                    data_x/norm_scale), dim=1).numpy()
data_y = 2*data_y - 1 # covert to 1, -1
max_data = max_data # len(data_x)
dimension = data_x.shape[1]

model = gp.Model("milp_deferral")
model.Params.IntFeasTol = 1e-9
model.Params.MIPFocus = 0
if self.time_limit != -1:
    model.Params.TimeLimit = self.time_limit

H = model.addVars(dimension, lb=[-C] *
                  dimension, ub=[C]*dimension, name="H")
Hnorm = model.addVars(
    dimension, lb=[0]*dimension, ub=[C]*dimension, name="Hnorm")
Rnorm = model.addVars(
    dimension, lb=[0]*dimension, ub=[C]*dimension, name="Rnorm")
R = model.addVars(dimension, lb=[-C] *
                  dimension, ub=[C]*dimension, name="R")
phii = model.addVars(max_data, vtype=gp.GRB.CONTINUOUS, lb=0)
psii = model.addVars(max_data, vtype=gp.GRB.BINARY)
ri = model.addVars(max_data, vtype=gp.GRB.BINARY)

equal = np.array(data_y) == hum_preds * 1.0
human_err = 1-equal

if self.add_regularization:
    model.setObjective(gp.quicksum([phii[i] + ri[i]*human_err[i]
    for i in range(max_data)])/max_data + self.lambda_reg * gp.
                        quicksum(
                            [Hnorm[j] for j in range(dimension)])
                        + self.lambda_reg * gp.quicksum([Rnorm[j] for j in range(
                            dimension]))))
else:
    model.setObjective(gp.quicksum(
        [phii[i] + ri[i]*human_err[i] for i in range(max_data)])/

```

```

max_data)

for i in range(max_data):
    model.addConstr(phii[i] >= psii[i] - ri[i], name="phii" + str(i))
    model.addConstr(Mi*psii[i] >= gamma - data_y[i]*gp.quicksum(
        H[j] * data_x[i][j] for j in range(dimension)), name="psii" +
        str(i))
    model.addConstr(gp.quicksum([R[j]*data_x[i][j] for j in range(
        dimension)]) >=
    Ki*( ri[i]-1) + gamma*ri[i], name="Riub" + str(i))
    model.addConstr(gp.quicksum([R[j]*data_x[i][j] for j in range(
        dimension)]) <= Ki*ri[i] + gamma*(ri[i]-1), name="Rilb" + str
        (i))

    model.update()
if self.add_regularization:
    for j in range(dimension):
        model.addConstr(Hnorm[j] >= H[j], name="Hnorm1" + str(j))
        model.addConstr(Hnorm[j] >= -H[j], name="Hnorm2" + str(j))
        model.addConstr(Rnorm[j] >= R[j], name="Rnorm1" + str(j))
        model.addConstr(Rnorm[j] >= -R[j], name="Rnorm2" + str(j))

model.ModelSense = 1 # minimize
model._time = time.time()
model._time0 = time.time()
model._cur_obj = float('inf')
# model.write('model.lp')
if self.verbose:
    model.optimize()
else:
    model.optimize()
# check if halspace solution has 0 error
error_v = 0
rejs = 0
for i in range(max_data):
    rej_raw = np.sum([R[j].X * data_x[i][j] for j in range(dimension)]
        )
    pred_raw = np.sum([H[j].X * data_x[i][j]
        for j in range(dimension)])
    if rej_raw > 0:
        rejs += 1
        error_v += (data_y[i] * hum_preds[i] != 1)
    else:
        pred = (pred_raw > 0)
        error_v += (data_y[i] != (2*pred-1))

```

```

self.H = [H[j].X for j in range(dimension)]
self.R = [R[j].X for j in range(dimension)]
self.run_time = model.Runtime
self.norm_scale = norm_scale
self.train_error = error_v/max_data

```

B.1.2 Realizable Surrogate

We implement the RealizableSurrogate in PyTorch. We showcase the loss function L_{RS} below:

```

def realizable_surrogate_loss(outputs, human_is_correct, labels, lambdaa):
    """
    outputs (tensor): outputs of model with K+1 output heads (without softmax)
    human_is_correct (tensor): binary tensor indicating if human is
        correct on each point I_{h=y}
    labels (tensor): list of targets y_i
    lambdaa (float in [0,1]): trade-off parameter in loss

    return: loss (single tensor)
    """
    batch_size = outputs.size()[0]
    outputs_exp = torch.exp(outputs)
    rs_loss = -torch.log2(( m * outputs_exp[range(batch_size), -1]
+ outputs_exp[range(batch_size), labels] ) / (torch.sum(outputs_exp, dim =
1) +eps_cst ))
    ce_loss = -torch.log2(( outputs_exp[range(batch_size), labels] )
/(torch.sum(outputs_exp[range(batch_size), :-1], dim = 1) +eps_cst
))

    loss = lambdaa*rs_loss + (1-lambdaa)*ce_loss
    return torch.sum(loss)/batch_size

```

B.2 MILP

B.2.1 Verification

The MILP in the binary setting is formulated as:

$$M^*, R^*, . = \arg \min_{M, R, \{r_i\}, \{t_i\}, \{\phi_i\}} \sum_i \phi_i + r_i \mathbb{I}_{h_i \neq y_i} \quad (\text{B.1})$$

$$\phi_i \geq t_i - r_i, \quad \phi_i \geq 0 \quad \forall i \in [n] \quad (\text{B.2})$$

$$K_m t_i \geq \gamma_h - y_i M^\top x_i \quad \forall i \in [n] \quad (\text{B.3})$$

$$R^\top x_i \leq K_r r_i + \gamma_r (r_i - 1), \quad R^\top x_i \geq K_r (r_i - 1) + \gamma_r r_i \quad \forall i \in [n] \quad (\text{B.4})$$

$$-C \leq R_i \leq C, \quad -C \leq M_i \leq C \quad \forall i \in [d] \quad (\text{B.5})$$

$$r_i \in \{0, 1\}, t_i \in \{0, 1\}, \phi_i \in \mathbb{R}^+ \quad \forall i \in [n], R, M \in \mathbb{R}^d \quad (\text{B.6})$$

Extension to Multiclass. The above MILP only applies to binary labels but we can generalize it to the multiclass setting where $\mathcal{Y} = \{1, \dots, C\}$. In this case, we have a coefficient vector M_j for each class $j \in \mathcal{Y}$, and $m(x) = \arg \max_{j \in \mathcal{Y}} M_j^\top x$. Given a labeled point (x, y) , we let $c_j = \text{sign}(M_y^\top x - M_j^\top x)$ for $j \neq y$, and let $t_i = \mathbb{I}_{\sum_{j \neq y} c_j < C-1}$. Then if $m(x) = y$, we must have $c_j = 1$ for all $j \neq y$ and thus $t_i = 0$ which means that the classifier is correct. Similarly, if there exists a $j \neq y$ for which $c_j = -1$, it means the classifier is incorrect and accordingly $t_i = 1$. We can reformulate these indicator constraints using a similar big-M approach as above. The formulation is below:

$$M^*, R^*, . = \arg \min_{M, R, \{r_i\}, \{t_i\}, \{c_{ij}\}, \{\phi_i\}} \sum_i \phi_i + r_i \mathbb{I}_{h_i \neq y_i} \quad (\text{B.7})$$

$$\phi_i \geq t_i - r_i, \quad \phi_i \geq 0 \quad \forall i \in [n] \quad (\text{B.8})$$

$$(M_{y_i} - M_j)^\top x_i \leq 2K_h c_{ij} + \gamma_h (c_{ij} - 1),$$

$$(M_{y_i} - M_j)^\top x_i \geq 2K_h (c_{ij} - 1) + \gamma_h c_{ij} \quad \forall i \in [n] \forall j \in [C] \neq y_i \quad (\text{B.9})$$

$$t_i \geq (C - 1 - \sum_{j \in [L], j \neq y_i} c_{ij}) / (C - 1) \quad (\text{B.10})$$

$$R^\top x_i \leq K_r r_i + \gamma_r (r_i - 1), \quad R^\top x_i \geq K_r (r_i - 1) + \gamma_r r_i \quad \forall i \in [n] \quad (\text{B.11})$$

$$-C \leq R_i \leq C, \quad -C \leq M[i, l] \leq C \quad \forall i \in [d] \forall l \in [C] \quad (\text{B.12})$$

$$r_i \in \{0, 1\}, t_i \in \{0, 1\}, c_{ij} \in \{0, 1\}, \phi_i \in \mathbb{R}^+ \quad \forall i \in [n], R, M \in \mathbb{R}^d \quad (\text{B.13})$$

Let us verify the formulations above.

The variable $\phi_i \geq \max(t_i - r_i, 0)$, the RHS takes values either 0 or 1, since ϕ_i in the objective then the optimal value is either 0 or 1 as well so that $\phi_i = \max(t_i - r_i, 0) = (1 - r_i)t_i$.

For t_i in the binary case: when $y_i M^\top x_i$ is positive, then $\gamma_h - y_i M^\top x_i$ is negative since $|M^\top x_i| \geq \gamma_h$ by Assumption 2, so that to satisfy constraint (B.3) either value of 0 or 1 are valid for t_i , however since t_i shows up in the objective then the optimal value is 0. On the other hand, when $y_i M^\top x_i$ is negative, then $\gamma_h - y_i M^\top x_i$ is positive, so that the only valid option for t_i is 1 and since $M^\top x_i \leq K_m$ then the constraint can be satisfied. So that we proved that $t_i = \text{sign}(y_i M^\top x_i)$.

We previously verified constraint for r_i and R in the body. When $r_i = 0$ then we have the constraints $R^\top x_i \leq -\gamma_r$ and $R^\top x_i \geq -K_r$: this forces the rejector to be negative which is consistent. When $r_i = 1$, we have $R^\top x_i \geq \gamma_r$ and $R^\top x_i \leq K_r$: which means the rejector is positive. Thus we proved $r_i = \mathbb{I}(R^\top x_i \geq 0)$.

For t_i in the multiclass settings: by analogy to the constraints for R and r_i it is easy to see that the variable $c_{ij} = \text{sign}(H_{y_i}^\top x_i - H_j^\top x_i)$. For a given x_i, y_i , the classification is only correct if $c_{ij} = 1$ for all $j \in [C] \neq y_i$ so that $\arg \max_j H_j^\top x_i = y_i$. We can then see that we set $t_i = \mathbb{I}(\sum_{j \neq y_i} c_{ij} / (C - 1) \neq 1)$ so that t_i denotes the error of our classifier on example i .

B.3 Experimental Details and Results

B.3.1 Baseline Implementation

OvASurrogate [62]: We rely on the loss implementation available online at ¹.

DifferentiableTriage [27]: We rely on the implementation in ². Note that the differentiable triage method implementation in [27] relies on having loss estimates of the human, particularly cross entropy loss estimates, which requires the conditional probabilities $\mathbb{P}(H = i | X = x)$ for each $i \in \mathcal{Y}$. However, in our setting, we only have samples of the human decisions m_i , not probabilistic estimates. The method can be summarized as a two-stage method: 1) classifier training: at each epoch, only train on points where classifier loss is lower than human loss, 2) rejector training: fit the rejector to predict who between the classifier and the human has lower loss. Since we only have samples of human behavior, we use the 0 – 1 loss of the classifier and the human on an example basis for comparison.

CrossEntropySurrogate [26]: We rely on the implementation in ³. We tune the parameter α over the grid $[0, 0.1, 0.5, 1]$ on the validation set.

CompareConfidence [25]: we train the classifier using the cross entropy loss on all the data, we then train a model to predict if the human is correct or not on each example in the training set. For

¹<https://github.com/rajeevv/OvA-L2D>

²<https://github.com/Networks-Learning/differentiable-learning-under-triage>

³<https://github.com/clinicalml/learn-to-defer>

each test point, we compare the confidence of the classifier versus the human correctness model and defer accordingly.

SelectivePrediction: we train the classifier using the cross entropy loss on all the data, for the rejector, we learn a single threshold on the validation set for the classifier confidence (probability of the predicted class) in order to maximize system accuracy.

B.3.2 Training Details

Table B.1: Training details for each dataset, we use the Adam optimizer [227] and AdamW [226]

Dataset	Optimizer	Number of Epochs	Learning Rate
SyntheticData (ours)	Adam	300	0.1
CIFAR-K	Adam	100	0.001
CIFAR-10H [75]	AdamW	20	0.001
Imagenet-16H [43]	Adam	20	0.001
HateSpeech [54]	Adam	50	0.001
COMPASS [79]	Adam	300	0.1
NIH Chest X-ray [80], [81]	AdamW	3	0.001

B.3.3 Synthetic Data

We show in Figure B.1 the performance of the different methods with the same setup with the uniform data distribution.

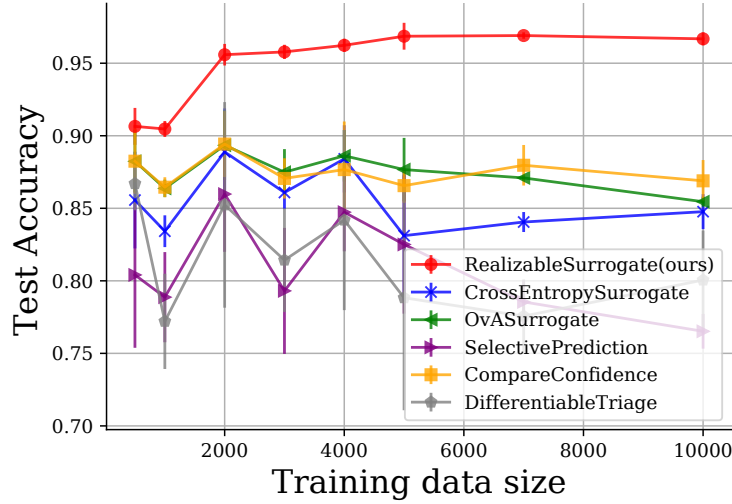


Figure B.1: (Test performance of the different methods on realizable synthetic data as we increase the training data size and repeat the randomization over 10 trials to get standard errors on uniform data.

We also experiment with making the data unrealizable by setting ($d = 10, p_m = 0.1, p_{h0} = 0.4, p_{h1} = 0.1$, Gaussian distribution with 20 clusters) in Figure B.2.

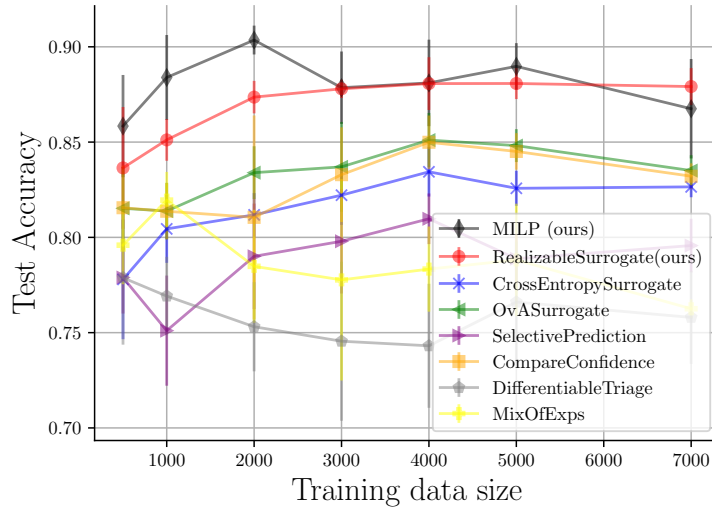
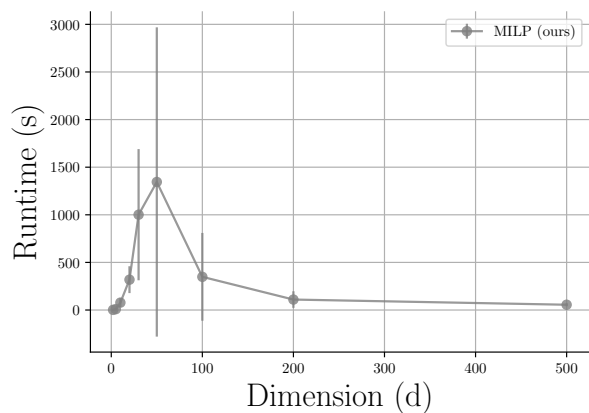
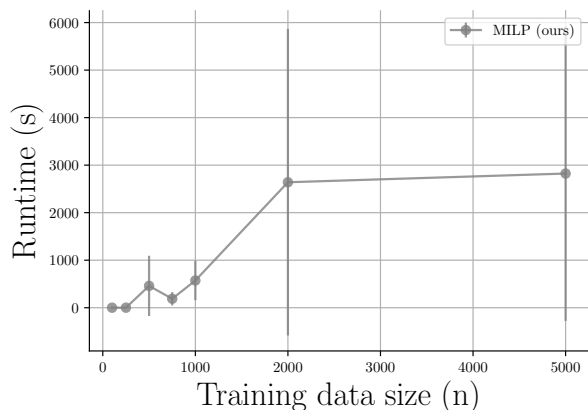


Figure B.2: (Test performance of the different methods on unrealizable ($d = 10, p_m = 0.1, p_{h0} = 0.4, p_{h1} = 0.1$, Gaussian distribution with 20 clusters) synthetic data as we increase the training data size.

We also show average run-times for the MILP on the synthetic data as we increase the dimension in Figure B.3a and as we increase the training data size in Figure B.3b. The distribution was uniform and realizable with $p_m = 0.0, p_{h0} = 0.3, p_{h1} = 0.0$. We observe that the run time increases with training set size which is the biggest bottleneck. The runtime also increases with dimension up until



(a) Runtime with increasing dimension, $n = 1000$

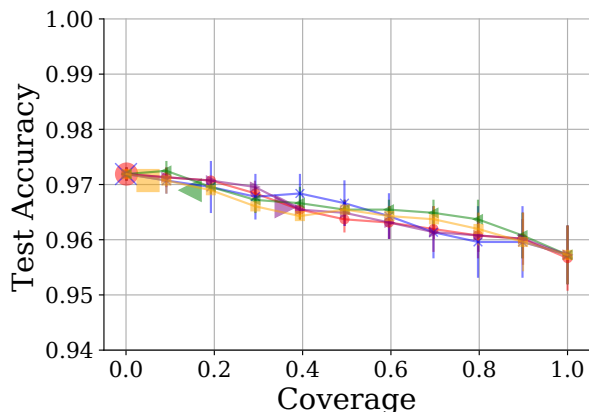


(b) Runtime with increasing training data size, $d = 30$

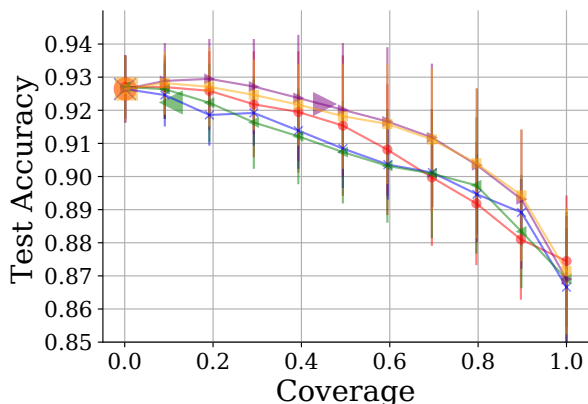
Figure B.3: Runtime of the MILP on the realizable synthetic data with uniform data distribution. Note that the test accuracy of the MILP is demonstrated in Figure 3.4a and the MILP always reaches 0 training error across the different data dimensions and training set sizes.

the dimension is of the same order as the number of training points, afterwards it is faster for the MILP to find a 0 error solution.

B.3.4 NIH Chest X-ray



(a) Fracture



(b) Nodule or Mass

Figure B.4: NIH Chest X-ray results on the two remaining tasks with the baselines and our method and red with circle markers. We see that all methods aren't able to obtain a performance of a human-AI team with better performance than the human, our method on both tasks defers to the human.

B.3.5 CIFAR-10H

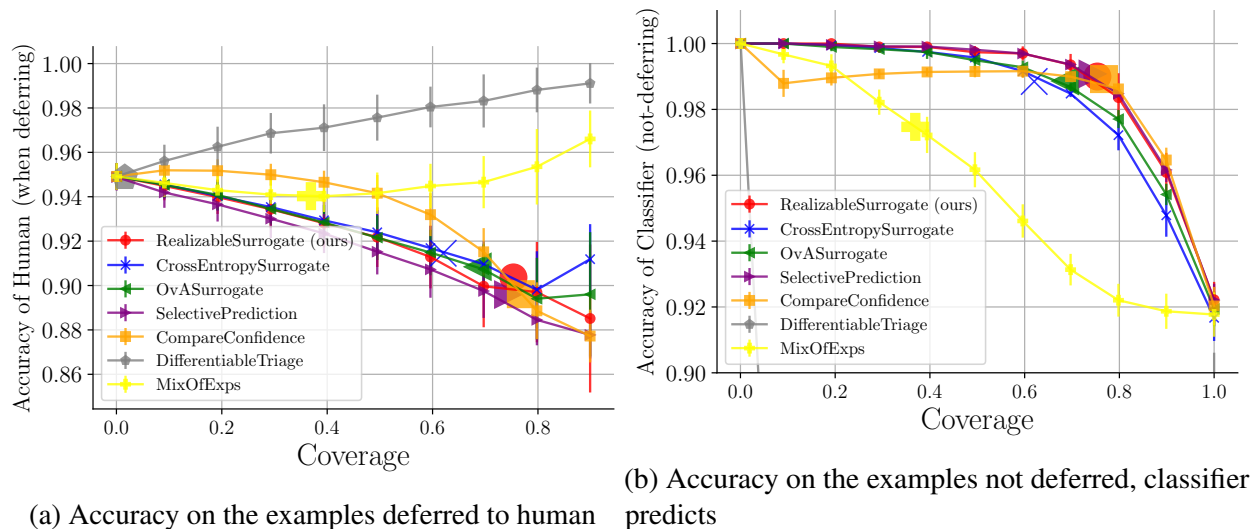


Figure B.5: On CIFAR-10H, classifier accuracy on non-deferred set and human accuracy when deferred vs coverage (fraction of points where classifier predicts).

B.4 Deferred Proofs and Derivations

B.4.1 Related Work

We mentioned that the surrogate in [62] belongs to the family derived in [23].

This is established by setting $l_\phi(i, f(x))$ as follows ⁴:

$$l_\phi(i, f(x)) = \begin{cases} \phi(g_y) + \sum_{y' \neq y} \phi(-g_{y'}), & \text{if } y \in \mathcal{Y} \\ \phi(g_y) - \phi(-g_{y'}), & \end{cases} \quad (\text{B.14})$$

B.4.2 Section 3.4 (Hardness)

Background and Definitions

Realizable Intersection of Halfspaces. For our purposes, an instance \mathcal{I} of learning an intersection of halfspaces in the realizable setting is given by a finite dataset $\{(x_i, y_i)\}_{i=1}^n$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$, such that there exist halfspaces $g_1^* : \mathbb{R}^d \rightarrow \{0, 1\}$ and $g_2^* : \mathbb{R}^d \rightarrow \{0, 1\}$ with zero error

⁴This was established by Yuzhou Cao.

on the dataset:

$$err_{\mathcal{I}}(g_1^*, g_2^*) := \frac{1}{n} \sum_i \mathbb{I}_{g_1^*(x_i) \wedge g_2^*(x_i) \neq y_i} = 0.$$

We consider two related problems: finding halfspaces (g_1, g_2) with *exact* and *weak* agreement.

Exact agreement. Given an instance \mathcal{I} of realizable intersection of halfspaces, the exact agreement problem is to find a pair of halfspaces (g_1, g_2) such that $g_1(x_i) \wedge g_2(x_i) = y_i$ for all $i \in \{1, \dots, n\}$.

Weak agreement. Given an instance \mathcal{I} of realizable intersection of halfspaces, the weak agreement problem is to find a pair of halfspaces (g_1, g_2) with error at most $1/2 - \gamma$ for some $\gamma > 0$:

$$err_{\mathcal{I}}(g_1, g_2) := \frac{1}{n} \sum_i \mathbb{I}_{g_1(x_i) \wedge g_2(x_i) \neq y_i} \leq \frac{1}{2} - \gamma.$$

Note that there exists a pair (g_1^*, g_2^*) with error 0 but the goal is just to obtain error $1/2 - \gamma$.

Quite a bit is known about the hardness of the exact and weak agreement problems.

Theorem ([228] Theorem 1, rephrased). *The exact agreement problem is NP-hard.*

Theorem ([70] Theorem 2, rephrased). *There is no polynomial-time algorithm for the weak agreement problem unless $NP = RP$.*

We also consider finite-data versions of LWD-H:

Finite-data realizable LWD-H. An instance \mathcal{J} of learning with deferral in the realizable setting is given by a finite dataset $\{(x_i, y_i, h_i)\}_{i=1}^n$, with $x_i \in \mathbb{R}^d$ and $y_i, h_i \in \{0, 1\}$, such that there exist halfspaces $m^* : \mathbb{R}^d \rightarrow \{0, 1\}$ and $r^* : \mathbb{R}^d \rightarrow \{0, 1\}$ with zero error on the dataset:

$$err_{\mathcal{J}}(m^*, r^*) := \frac{1}{n} \sum_i \mathbb{I}_{r^*(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r^*(x_i)=0} \mathbb{I}_{m^*(x_i) \neq y_i} = 0.$$

As with intersection-of-halfspaces, we can consider finding halfspace classifier/rejector pairs (m, r) with *exact* and *weak* agreement.

Exact agreement. Given an instance \mathcal{J} of realizable LWD-H, the exact agreement problem is to find a pair of halfspaces (m, r) such that for all i , if $r(x_i) = 0$, $m(x_i) = y_i$, and if $r(x_i) = 1$, $h_i = y_i$. That is, the error of the classifier/human system on the finite dataset is 0.

Weak agreement. Given an instance \mathcal{J} of realizable LWD-H, the weak agreement problem is to find a pair of halfspaces (m, r) with error at most $1/2 - \gamma$ for some $\gamma > 0$:

$$\text{err}_{\mathcal{J}}(m, r) := \frac{1}{n} \sum_i \mathbb{I}_{r(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i} \leq \frac{1}{2} - \gamma.$$

Mapping between learning intersections and LWD-H

We show how to turn an instance \mathcal{I} of realizable intersection of halfspaces into an instance of \mathcal{J} of (finite-data) realizable LWD-H. Given an arbitrary instance \mathcal{I} on dataset \mathcal{D} , Lemma 12 shows how to construct an instance \mathcal{J} of LWD-H and a bijection $(g_1, g_2) \longleftrightarrow (m, r)$ such that for arbitrary halfspaces (g_1, g_2) , the error $\text{err}_{\mathcal{I}}(g_1, g_2) = \text{err}_{\mathcal{J}}(m, r)$. In particular, since we assumed \mathcal{I} is realizable and hence $\exists g_1^*, g_2^*$ with $\text{err}_{\mathcal{I}}(g_1^*, g_2^*) = 0$, Lemma 12 shows how to construct an instance \mathcal{J} of LWD-H with $\text{err}_{\mathcal{J}}(m^*, r^*) = 0$. This will allow us to reduce an arbitrary instance \mathcal{I} of realizable intersection of halfspaces to an instance \mathcal{J} of realizable LWD-H. Additionally, given an arbitrary classifier/rejector pair (m, r) on this \mathcal{J} with error ϵ , Lemma 12 shows how to map $(m, r) \rightarrow (g_1, g_2)$ with error ϵ on instance \mathcal{I} .

Lemma 12. Consider an arbitrary instance \mathcal{I} of learning an intersection of halfspaces on a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. Define $\tilde{\mathcal{D}} = \{(x_i, y_i, 0)\}_{i=1}^n$. This corresponds to an instance \mathcal{J} of LWD-H where the “human expert” always outputs label 0.

Then:

1. Consider two arbitrary halfspaces g_1, g_2 and set $m(x) = g_1(x)$, $r(x) = 1 - g_2(x)$. Note that m and r are also halfspaces. Then $\text{err}_{\mathcal{I}}(g_1, g_2) = \text{err}_{\mathcal{J}}(m, r)$. That is,

$$\frac{1}{n} \sum_{(x_i, y_i) \in \mathcal{D}} \mathbb{I}[g_1(x_i) \wedge g_2(x_i) \neq y_i] = \frac{1}{n} \sum_{(x_i, y_i, h_i) \in \tilde{\mathcal{D}}} (\mathbb{I}_{r(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i}).$$

2. Suppose \mathcal{I} is an instance of realizable intersection of halfspaces. Then the instance \mathcal{J} of LWD-H defined by the dataset $\tilde{\mathcal{D}}$ is an instance of realizable LWD-H. That is, there exists (m^*, r^*) with $\text{err}_{\mathcal{J}}(m^*, r^*) = 0$.

Proof. For part 1, recall that by definition:

$$\text{err}_{\mathcal{J}}(m, r) = \frac{1}{n} \sum_{(x_i, h_i, y_i) \in \tilde{\mathcal{D}}} (\mathbb{I}_{r(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i}).$$

Since $h_i = 0$ for all i , this is equal to

$$\frac{1}{n} \sum_i (\mathbb{I}_{r(x_i)=1} \mathbb{I}_{y_i=1} + \mathbb{I}_{r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i}).$$

Using $r(x) = 1 - g_2(x)$ and $m(x) = g_1(x)$, this simplifies further to:

$$\frac{1}{n} \sum_i (\mathbb{I}_{g_2(x_i)=0} \mathbb{I}_{y_i=1} + \mathbb{I}_{g_2(x_i)=1} \mathbb{I}_{g_1(x_i) \neq y_i}). \quad (\text{B.15})$$

Consider the error of $\text{err}_{\mathcal{I}}(g_1, g_2)$. The model makes a mistake if $g_2(x) = 0$ and $y(x) = 1$, $g_2(x) = g_1(x) = 1$ and $y = 0$, or $g_2(x) = 1, g_1(x) = 0$, and $y = 1$. The first case is $\mathbb{I}_{g_2(x)=0} \mathbb{I}_{y=1}$ and the latter two cases can be expressed as $\mathbb{I}_{g_2(x)=1} \mathbb{I}_{g_1(x) \neq y}$. Hence

$$\text{err}_{\mathcal{I}}(g_1, g_2) = \frac{1}{n} \sum_{(x_i, y_i) \in \tilde{\mathcal{D}}} \mathbb{I}[g_1(x_i) \wedge g_2(x_i) \neq y_i] = \frac{1}{n} \sum_i (\mathbb{I}_{g_2(x_i)=0} \mathbb{I}_{y_i=1} + \mathbb{I}_{g_2(x_i)=1} \mathbb{I}_{g_1(x_i) \neq y_i}),$$

which is equal to (B.15), so $\text{err}_{\mathcal{I}}(g_1, g_2) = \text{err}_{\mathcal{J}}(m, r)$.

For part 2, we assumed that \mathcal{I} was realizable, so there exists g_1^*, g_2^* with $\text{err}_{\mathcal{I}}(g_1^*, g_2^*) = 0$. Applying part 1 yields m^*, r^* such that $\text{err}_{\mathcal{J}}(m^*, r^*) = 0$. Hence \mathcal{J} is an instance of realizable LWD-H. \square

Lemma 12 takes an instance \mathcal{I} of learning an intersection of halfspaces and constructs an instance \mathcal{J} of LWD-H such that there is an error-preserving bijection between solutions of \mathcal{I} and solutions of \mathcal{J} . This allows us to easily apply the existing hardness results for learning a realizable intersection of halfspaces, since if \mathcal{I} is realizable then so is \mathcal{J} .

Hardness results for LWD-H

Theorem 9. *There is no polynomial-time algorithm for solving the exact agreement problem for LWD-H unless $P=NP$.*

Proof. Suppose there exists a polytime algorithm \mathcal{A} for solving exact agreement on realizable LWD-H. Consider an arbitrary instance \mathcal{I} of learning a realizable intersection of halfspaces. Lemma 12 shows how to construct an instance \mathcal{J} of realizable LWD-H. Run Algorithm \mathcal{A} on \mathcal{J} to obtain halfspaces (m, r) with $\text{err}_{\mathcal{J}}(m, r) = 0$. Set $g_1 = m, g_2 = 1 - r$. Lemma 12 guarantees that $\text{err}_{\mathcal{I}}(g_1, g_2) = 0$. Hence, \mathcal{A} is a polynomial-time algorithm for exact agreement for realizable intersection of halfspaces. [228] shows that there is no polynomial-time algorithm for exact agreement for realizable intersection of halfspaces unless $P = NP$. \square

Corollary 10. *There is no efficient, proper PAC learner for realizable LWD-H unless $NP = RP$.*

Proof sketch. Suppose \mathcal{A} is an efficient proper PAC learner for realizable LWD-H, so for any distribution \mathcal{D} , any $\epsilon > 0$, $\delta > 0$, given $\text{poly}(1/\delta, 1/\epsilon)$ samples from \mathcal{D} , \mathcal{A} outputs a pair of halfspaces (m, r) with (population) system error at most ϵ in time $\text{poly}(1/\epsilon, 1/\delta)$.

Now let \mathcal{D} be the uniform distribution over a dataset of n points $\{(x_i, y_i, h_i)\}_{i=1}^n$. Set $\epsilon = 1/(2n)$ and $\delta = 1/100$ and run \mathcal{A} . With probability at least $1 - \delta$ \mathcal{A} outputs (m, r) with error at most $1/(2n)$. Of course, if (m, r) has error at most $1/(2n)$ it must have error 0. This gives a randomized algorithm for solving the exact agreement problem for realizable finite-data LWD-H. \square

These results show that exact agreement, and thus exact proper PAC learning, are hard. Next we consider the hardness of weak agreement.

Theorem 5 *Let $\epsilon > 0$ be an arbitrarily small constant and suppose we have an instance \mathcal{J} of realizable LWD-H. So we have data $\mathcal{D} = \{(x_i, y_i, h_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^d$, $y_i, h_i \in \{0, 1\}$, and there exist halfspaces m^*, r^* with zero loss on \mathcal{D} :*

$$\text{err}_{\mathcal{J}}(m^*, r^*) := \frac{1}{n} \sum_i (\mathbb{I}_{r^*(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r^*(x_i)=0} \mathbb{I}_{m^*(x_i) \neq y_i}) = 0$$

Then there is no polynomial-time algorithm to find a classifier-rejector pair (\hat{m}, \hat{r}) with error $1/2 - \epsilon$, i.e.:

$$\frac{1}{n} \sum_i (\mathbb{I}_{\hat{r}(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{\hat{r}(x_i)=0} \mathbb{I}_{\hat{m}(x_i) \neq y_i}) \leq \frac{1}{2} - \epsilon$$

unless $NP = RP$.

Proof. Suppose there exists a polynomial-time algorithm \mathcal{A} and a $\gamma > 0$ such that given an instance \mathcal{J} of realizable LWD-H, \mathcal{A} returns a pair (\hat{m}, \hat{r}) with error $\text{err}_{\mathcal{J}}(\hat{m}, \hat{r})$ at most $1/2 - \gamma$. Consider an arbitrary instance \mathcal{I} of realizable intersection of halfspaces. Lemma 12 shows how to reduce \mathcal{I} to an instance \mathcal{J} of realizable LWD-H. Run Algorithm \mathcal{A} on \mathcal{J} to obtain a pair of halfspace (\hat{m}, \hat{r}) with error at most $\text{err}_{\mathcal{J}}(\hat{m}, \hat{r}) \leq 1/2 - \gamma$. Lemma 12 guarantees that $g_1 = \hat{m}, g_2 = 1 - \hat{r}$ satisfy $\text{err}_{\mathcal{I}}(g_1, g_2) \leq 1/2 - \gamma$. Hence \mathcal{A} gives a deterministic algorithm for solving the weak agreement problem for realizable intersection of halfspaces. [70, Theorem 4] construct an algorithm/reduction showing that if we can efficiently solve weak agreement for realizable intersection of halfspaces, then Smooth Label Cover is in RP , but Smooth Label Cover is an NP-hard problem [70, Theorem 3]. Hence there is no polynomial-time algorithm to find a classifier-rejector pair (\hat{m}, \hat{r}) with error $1/2 - \epsilon$ unless $NP = RP$. \square

Corollary 11. *There is no efficient, proper, weak PAC-learner for realizable LWD-H unless $NP = BPP$.*

Proof. Given a distribution \mathcal{D} over points (x, y, h) , $x \in \mathbb{R}^d$, $y, h \in \{0, 1\}$ and halfspaces (m, r) , let

$$\text{err}_{\mathcal{D}}(m, r) := \mathbb{P}_{(x,y,h) \sim \mathcal{D}}[r(x) = 1 \wedge h \neq y \vee r(x) = 0 \wedge m(x) \neq y].$$

This is identical to the *system loss* (3.1) on distribution \mathcal{D} . Suppose there exists an efficient, proper, weak PAC-learner for realizable LWD-H. I.e., there exists some γ such that for any distribution \mathcal{D} , under the guarantee that $\exists(m^*, r^*)$ with $\text{err}_{\mathcal{D}}(m^*, r^*) = 0$, given access to $\text{poly}(1/\delta)$ samples from \mathcal{D} , with probability at least $1 - \delta$, \mathcal{A} returns a pair (m, r) with $\text{err}_{\mathcal{D}}(m, r) \leq \frac{1}{2} - \gamma$ in $\text{poly}(1/\delta)$ time.

By combining Lemma 12 with the randomized reduction of [70], we can use \mathcal{A} to construct an algorithm that implies Smooth Label Cover is in *BPP*. The definition of Smooth Label Cover is not important for our purposes beyond the following two results:

Theorem 12. [70, Theorem 3] *For any constant t and arbitrarily small constants $\mu, \vartheta, \eta > 0$, there exist constants k and m such that given an instance \mathcal{L} of Smooth-Label-Cover(t, μ, ϑ, k, m) it is NP-hard to distinguish between the following two cases:*

- *YES Case/Completeness: There is a labeling to the vertices of \mathcal{L} which satisfies all the edges.*
- *NO Case/Soundness: No labeling to the vertices of \mathcal{L} satisfies more than η fraction of the edges.*

Theorem 13. [70, Theorem 4] *For any constant $\gamma > 0$ and integer $l > 0$, there is a randomized polynomial time reduction from an instance \mathcal{L} of Smooth-Label-Cover(t, μ, ϑ, k, m) to an instance \mathcal{I} of Realizable Intersection of Halfspaces for appropriately chosen parameters (t, μ, ϑ) and soundness η , such that*

- *YES Case/Completeness: If \mathcal{L} is a YES instance, then there is an intersection of two halfspaces which correctly classifies all the points in instance \mathcal{I} .*
- *NO Case/Soundness: If \mathcal{L} is a NO instance, then with probability at least 9/10, there is no function of up to l halfspaces that correctly classifies more than $1/2 + \gamma$ fraction of points in instance \mathcal{I} .*

For our case, we can use Lemma 12 to further reduce the instance \mathcal{I} constructed by Theorem 13 to an instance \mathcal{J} of LWD-H, then run the weak PAC-learner \mathcal{A} on \mathcal{J} . If \mathcal{A} outputs a pair of halfspaces (m, r) with error at most $1/2 - \gamma$, we output YES. Otherwise we output NO.

If \mathcal{I} is a realizable instance, \mathcal{A} returns a pair of halfspaces with error at most $1/2 - \gamma$ with probability at least $1 - \delta$. On the other hand, if \mathcal{I} is not weakly realizable (w.r.) (i.e., there is no

function of up to l halfspaces that correctly classifies more than a $1/2 + \gamma$ fraction of points in \mathcal{I} , then clearly \mathcal{A} never returns a good pair of halfspaces, since no such pair exists. Therefore:

$$\begin{aligned}\mathbb{P}(\text{YES}|\mathcal{L} \text{ YES}) &= \mathbb{P}(\text{YES}|\mathcal{I} \text{ realizable})\mathbb{P}(\mathcal{I} \text{ realizable}|\mathcal{L} \text{ YES}) \\ &= (1 - \delta) \cdot 1\end{aligned}$$

$$\begin{aligned}\mathbb{P}(\text{NO}|\mathcal{L} \text{ NO}) &= \mathbb{P}(\text{NO}|\mathcal{I} \text{ w.r.})\mathbb{P}(\mathcal{I} \text{ w.r.}|\mathcal{L} \text{ NO}) + \mathbb{P}(\text{NO}|\mathcal{I} \text{ not w.r.})\mathbb{P}(\mathcal{I} \text{ not w.r.}|\mathcal{L} \text{ NO}) \\ &\geq \mathbb{P}(\text{NO}|\mathcal{I} \text{ not w.r.})\mathbb{P}(\mathcal{I} \text{ not w.r.}|\mathcal{L} \text{ NO}) \\ &\geq \mathbb{P}(\text{NO}|\mathcal{I} \text{ not w.r.})\frac{9}{10} \\ &= \frac{9}{10}.\end{aligned}$$

Hence we can use \mathcal{A} to construct an algorithm for a Smooth-Label-Cover instance \mathcal{L} that outputs YES when \mathcal{L} is a YES with probability at least $(1 - \delta)$, and outputs NO when \mathcal{L} is a NO with probability at least $9/10$. Since we assumed \mathcal{A} runs in $\text{poly}(1/\delta)$, this implies Smooth Label Cover is in BPP . Together with Theorem 12, this shows that there is no efficient, proper, weak PAC learner for realizable LWD-H unless $NP = BPP$. \square

Finally, we show that when realizability is violated, there is no efficient algorithm for weak agreement.

Corollary 1 (formal). *Let $\delta, \epsilon > 0$ be arbitrarily small constants. Then, given a set of points $\{(x_i, y_i, h_i)\}$ with $x_i \in \mathbb{R}^d$, $y_i, h_i \in \{0, 1\}$ with a guarantee that there is a classifier/rejector pair (m^*, r^*) that classifies a $1 - \delta$ fraction of points correctly, there is no polynomial time algorithm to find a classifier-rejector pair that classifies $\frac{1}{2} + \epsilon$ fraction of points correctly unless $P = NP$.*

Proof. This is a simple reduction from learning a single halfspace in the presence of noise, which is hard by the following result:

Theorem. ([229], see also [70, Theorem 1]) *Let $\delta, \epsilon > 0$ be arbitrarily small constants. Then, given a set of labeled points $\{(x_i, y_i)\}$ in \mathbb{R}^d with a guarantee that there is a halfspace that classifies $1 - \delta$ fraction of points correctly, there is no polynomial time algorithm to find a halfspace that classifies $1/2 + \epsilon$ fraction of points correctly, unless $P = NP$.*

Suppose we have an algorithm \mathcal{A} for solving LWD-H in the presence of noise. In particular, there exists some $\epsilon > 0, \delta > 0$ such that under the guarantee that there exists an (m^*, r^*) pair with error at most δ , \mathcal{A} returns an (m, r) pair with error at most $\frac{1}{2} - \epsilon$.

Consider an instance \mathcal{I} of learning a single halfspace in the presence of noise defined by a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, such that there exists a halfspace c with error at most δ on \mathcal{D} . From \mathcal{D} ,

construct the dataset $\tilde{\mathcal{D}} = \{(x_i, y_i, 1 - y_i)\}_{i=1}^n$. This is an instance \mathcal{J} of LWD-H where the “human expert” is always wrong. Note that $(c, 0)$ is a classifier/rejector pair with error at most δ on $\tilde{\mathcal{D}}$, so \mathcal{J} is an instance of LWD-H with noise level δ . Run algorithm \mathcal{A} on \mathcal{J} with parameter ϵ to obtain an (m, r) pair with $\text{err}_{\mathcal{J}}(m, r) = 1/2 - \epsilon$. Then:

$$\begin{aligned}
1/2 - \epsilon \geq \text{err}_{\mathcal{J}}(m, r) &= \frac{1}{n} \sum_i \mathbb{I}_{r(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \mathbb{I}_{r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i} \\
&= \frac{1}{n} \left(\sum_{i:r(x_i)=1} \mathbb{I}_{h_i \neq y_i} + \sum_{i:r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i} \right) \\
&\geq \frac{1}{n} \left(\sum_{i:r(x_i)=1} \mathbb{I}_{m(x_i) \neq y_i} + \sum_{i:r(x_i)=0} \mathbb{I}_{m(x_i) \neq y_i} \right) \\
&= \frac{1}{n} \sum_i \mathbb{I}_{m(x_i) \neq y_i} \\
&= \text{err}_{\mathcal{I}}(m),
\end{aligned}$$

where the inequality is because we constructed $\tilde{\mathcal{D}}$ such that $\mathbb{I}_{h_i \neq y_i} = 1$ for all i . Therefore, there exists a δ and ϵ for which, given a dataset and the guarantee that there exists a halfspace with error at most δ , we can output a halfspace with error at most $1/2 - \epsilon$. Combining this with the Theorem above shows that if \mathcal{A} runs in polynomial time, $P = NP$. \square

B.4.3 Section 3.5 (MILP)

Proposition 1. *For any expert H and data distribution \mathbf{P} over $\mathcal{X} \times \mathcal{Y}$ that satisfies Assumption 2, let $0 < \delta < \frac{1}{2}$, then with probability at least $1 - \delta$, the following holds for the empirical minimizers (\hat{m}^*, \hat{r}^*) obtained by the MILP:*

$$L_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) \leq \hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) \frac{(K_m + K_r)d\sqrt{2\log d} + 10\sqrt{\log(2/\delta)}}{\sqrt{n\mathbb{P}(H(Z) \neq Y)}}$$

Proof. We first start by recalling Theorem 2 in [26]:

$$\begin{aligned}
L_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) &\leq \hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) + \mathfrak{R}_n(\mathcal{M}) + \mathfrak{R}_n(\mathcal{R}) + \mathfrak{R}_{n\mathbb{P}(H(Z) \neq Y)/2}(\mathcal{R}) \\
&\quad + 2\sqrt{\frac{\log(\frac{2}{\delta})}{2n}} + \frac{\mathbb{P}(H(Z) \neq Y)}{2} \exp\left(-\frac{n\mathbb{P}(H(Z) \neq Y)}{8}\right) \tag{B.16}
\end{aligned}$$

Note that here we avoid going through the optimal solution and just relate distribution perfor-

mance to the training performance.

In the bound (B.16), $\mathfrak{R}_n(\mathcal{M})$ and $\mathfrak{R}_n(\mathcal{R})$ denote the Rademacher complexity of a halfspace in d dimensions where the infinity norm of each element in the halfspace is constrained by K_m and K_r respectively. Let us now compute this Rademacher complexity, inspired by [230]:

$$\begin{aligned}
\mathfrak{R}_n(\mathcal{M}) &= \frac{1}{n} \mathbb{E} \left[\sup_{M: \|M\|_\infty \leq K_m} \sum_{i=1}^n \epsilon_i M^\top x_i \right] \\
&\leq \frac{1}{n} \mathbb{E} \left[\sup_{M: \|M\|_1 \leq dK_m} M^\top \sum_{i=1}^n \epsilon_i x_i \right] \quad (\text{since } \|M\|_1 \leq d\|M\|_\infty) \\
&= \frac{dK_m}{n} \mathbb{E} \left[\sum_{i=1}^n \|\epsilon_i x_i\|_\infty \right] \\
&= \frac{dK_m}{n} \mathbb{E} \left[\sup_j \sum_{i=1}^n \epsilon_i [x_i]_j \right] \\
&\leq \frac{dK_m \sqrt{2 \log d}}{n} \sup_j \sqrt{\sum_{i=1}^n [x_i]_j^2} \quad (\text{Massart's finite lemma on } x_{ij}) \\
&\leq \frac{dK_m \sqrt{2 \log d}}{\sqrt{n}} \quad (\text{assume } \|x_i\|_1 \leq 1 \text{ for all } i)
\end{aligned}$$

Let us use the Rademacher complexity calculation in the bound to get:

$$\begin{aligned}
L_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) &\leq \hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) + \frac{dK_m \sqrt{2 \log d}}{\sqrt{n}} + \frac{dK_r \sqrt{2 \log d}}{\sqrt{n}} + \frac{dK_m \sqrt{2 \log d}}{\sqrt{n \mathbb{P}(H(Z) \neq Y)}} \\
&\quad + 2 \sqrt{\frac{\log(\frac{2}{\delta})}{2n}} + \frac{\mathbb{P}(H(Z) \neq Y)}{2} \exp\left(-\frac{n \mathbb{P}(H(Z) \neq Y)}{8}\right)
\end{aligned}$$

note that $\frac{\mathbb{P}(H(Z) \neq Y)}{2} \exp\left(-\frac{n \mathbb{P}(H(Z) \neq Y)}{8}\right)$ is a term that does not depend on the optimization and shrinks much faster than $\frac{8}{\sqrt{n \mathbb{P}(H(Z) \neq Y)}}$, so that we can summarize things as:

$$L_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) \leq \hat{L}_{\text{def}}^{0-1}(\hat{m}^*, \hat{r}^*) + \frac{(K_m + K_r) d \sqrt{2 \log d} + 10 \sqrt{\log(2/\delta)}}{\sqrt{n \mathbb{P}(H(Z) \neq Y)}} \quad (\text{B.17})$$

□

B.4.4 Section 3.6 (RealizableSurrogate)

Theorem 2. *The RealizableSurrogate L_{RS} is a realizable $(\mathcal{M}, \mathcal{R})$ -consistent surrogate for L_{def}^{0-1} for model classes closed under scaling, and satisfies $L_{\text{def}}^{0-1}(m, r) \leq L_{RS}(m, r)$ for all (m, r) .*

Proof. Let us recall the RealizableSurrogate loss pointwise:

$$L_{RS}(\mathbf{g}, x, y, h) = -2 \log \left(\frac{\exp(g_y(x)) + \mathbb{I}_{h=y} \exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right) \quad (\text{B.18})$$

where $\mathbf{g} = \{g_i\}_{i \in \mathcal{Y} \cup \perp}$. Recall that the classifier and rejector are defined as: $m(x) = \arg \max_{y \in \mathcal{Y}} g_y(x)$ and $r(x) = \mathbb{I}_{\max_{y \in \mathcal{Y}} g_y(x) \leq g_{\perp}(x)}$.

We first prove that for every point, the RealizableSurrogate loss upper bounds the system 0-1 error: $L_{\text{def}}^{0-1}(m, r, x, y, h) \leq L_{RS}(\mathbf{g}, x, y, h)$:

1. **Case 1:** consider $r(x) = 0$ (classifier predicts):

(a) **Case 1a:** if the classifier is incorrect, $\mathbb{I}_{m(x) \neq y} = 1$:

i. **Case 1ai:** If the human is incorrect, $\mathbb{I}_{h=y} = 0$:

then the loss is: $-2 \log \left(\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right)$, we know since the classifier is incorrect, then it must be that $\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \leq 0.5$ (since g_y is not the max), thus the loss is greater than 2 (log is base 2), and the 0-1 loss is 1 in this case.

ii. **Case 1aii:** if the human is correct then $\mathbb{I}_{h=y} = 1$:

then the loss is: $-2 \log \left(\frac{\exp(g_y(x)) + \exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right)$, we know since the classifier is incorrect, then it must be that $\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} + \frac{\exp(g_{\perp}(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} < 2/3$ since g_y is not the max neither is g_{\perp} , otherwise if the sum of these two fractions is greater than $2/3$, then $\max_i \frac{\exp(g_i(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} < 1/3$ then the maximum must be one of y or \perp which is a contradiction. Finally, the loss is greater than $-2 \log(2/3) = 1.17$ which is greater than 1.

(b) **Case 1b:** if the classifier is correct $\mathbb{I}_{m(x)=y} = 1$, then the 0-1 error is 0, since the RealizableSurrogate loss is ≥ 0 then it is an upper bound.

2. **Case 2:** consider $r(x) = 1$ (human predicts):

(a) **Case 2a:** if the human is correct then $\mathbb{I}_{h=y} = 1$:

then the 0-1 error is 0, since the RealizableSurrogate loss is ≥ 0 then it is an upper bound.

(b) if the human is incorrect then $\mathbb{I}_{h=y} = 0$:

the loss is $-2 \log \left(\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \right)$, we know since we defer, then it must be that $\frac{\exp(g_y(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(g_{y'}(x))} \leq 0.5$ (since g_y is not the max), thus the loss is greater than 2 (log is base 2), and the 0-1 loss is 1 in this case.

this concludes the proof of the upper bound.

We now prove that L_{RS} is a realizable-consistent loss function.

Consider a data distribution and a human under which there exists $m^*, r^* \in \mathcal{M} \times \mathcal{R}$ that have zero error $L_{\text{def}}^{0-1}(m^*, r^*) = 0$. Associated with m^*, r^* , is a set of functions $\mathbf{g}^* \in \mathcal{G}$ that give rise to m^*, r^* . Let $\hat{\mathbf{g}}$ be the minimizer of the surrogate loss L_{RS} and the associated classifier and rejector be \hat{m}, \hat{r} .

We now upper bound the 0-1 loss of the pair \hat{m}, \hat{r} . Let $u \in \mathbb{R}$ be any real number:

$$\begin{aligned}
& L_{\text{def}}^{0-1}(\hat{m}, \hat{r}) \\
& \leq L_{RS}(\hat{m}, \hat{r}) \quad (\text{loss is upper bound}) \\
& \leq L_{RS}(um^*, ur^*) \quad (\text{since } \hat{m}, \hat{r} \text{ is optimal for } L_{RS} \text{ and } \mathcal{M} \times \mathcal{R} \text{ is closed under scaling}) \\
& = \mathbb{E}[L_{RS}(um^*, ur^*, x, y, h) | r^* = 1] \mathbb{P}(r^* = 1) + \mathbb{E}[L_{RS}(um^*, ur^*, x, y, h) | r^* = 0] \mathbb{P}(r^* = 0)
\end{aligned} \tag{B.19}$$

Let us investigate the two terms in equation (B.19).

The first term is when $r^* = 1$, then we must have $g_{\perp}^* > \max_y g_y^*$ and $\mathbb{I}_{h=y} = 1$ since the data is realizable and when we defer the human must be correct. Examining the first term and taking the limit:

$$\begin{aligned}
& \lim_{u \rightarrow \infty} \mathbb{E}[L_{RS}(um^*, ur^*, x, y, h) | r^* = 1] \mathbb{P}(r^* = 1) \\
& = \lim_{u \rightarrow \infty} \mathbb{E} \left[-2 \log \left(\frac{\exp(ug_y^*(x)) + \mathbb{I}_{h=y} \exp(ug_{\perp}^*(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(ug_{y'}^*(x))} \right) \middle| r^* = 1 \right] \mathbb{P}(r^* = 1) \\
& = \lim_{u \rightarrow \infty} \mathbb{E} \left[-2 \log \left(\frac{\exp(ug_y^*(x)) + \exp(ug_{\perp}^*(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(ug_{y'}^*(x))} \right) \middle| r^* = 1 \right] \mathbb{P}(r^* = 1) \\
& = \mathbb{E}[-2 \log(1) | r^* = 1] \mathbb{P}(r^* = 1) = 0 \quad (\text{applying monotone convergence theorem})
\end{aligned}$$

The second term is when $r^* = 0$, then we must have $g_y^* > \max_{y' \in (\mathcal{Y} \setminus y) \cup \perp} g_{y'}^*$ since the data is realizable. Examining the second term and taking the limit:

$$\begin{aligned}
& \lim_{u \rightarrow \infty} \mathbb{E}[L_{RS}(um^*, ur^*, x, y, h) | r^* = 0] \mathbb{P}(r^* = 0) \\
&= \lim_{u \rightarrow \infty} \mathbb{E}\left[-2 \log \left(\frac{\exp(ug_y^*(x)) + \mathbb{I}_{h=y} \exp(ug_{\perp}^*(x))}{\sum_{y' \in \mathcal{Y} \cup \perp} \exp(ug_{y'}^*(x))} \right) | r^* = 0\right] \mathbb{P}(r^* = 0) \\
&= \mathbb{E}[-2 \log(1) | r^* = 0] \mathbb{P}(r^* = 0) = 0 \quad (\text{applying monotone convergence theorem})
\end{aligned}$$

Thus combining the above two derivations, we obtain:

$$L_{\text{def}}^{0-1}(\hat{m}, \hat{r}) \leq 0.$$

We just proved that the optimal solution from minimizing RealizableSurrogate leads to a zero error solution in terms of system error which proves that the loss is realizable $(\mathcal{M}, \mathcal{R})$ -consistent. \square

Theorem 14. *The CrossEntropySurrogate L_{CE} [26] is not a realizable $(\mathcal{M}, \mathcal{R})$ -consistent surrogate for L_{def}^{0-1} .*

Proof. To prove that the surrogate L_{CE} is not realizable-consistent, we will construct an example with a data distribution and a model class closed under scaling such that: 1) there exists a zero error solution in the model class and 2) the minimizer of L_{CE} has non-zero error.

Consider the data distribution illustrated and described in Figure B.6 consisting of four regions R0, R1, R2 and R3. Each region respectively has mass $1/4 + \alpha, 1/4, 1/4 - \alpha, 1/4$. Each region respectively has label $Y = 0, Y = 1, Y = 0, Y = 2$. The Human is perfectly accurate on Region 0 and inaccurate on every other region.

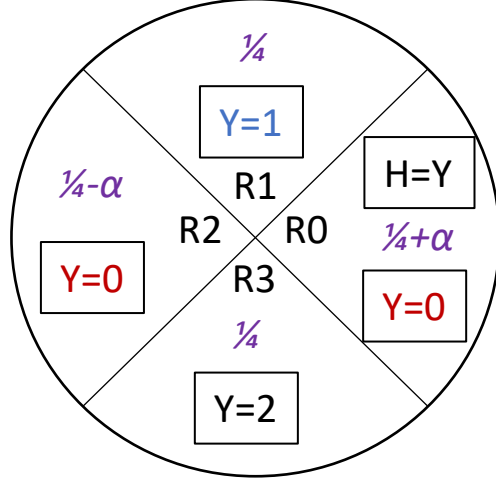


Figure B.6: Data Distribution for our example: the data consists of four regions R0,R1,R2 and R3. Each region respectively has mass $1/4 + \alpha, 1/4, 1/4 - \alpha, 1/4$. Each region respectively has label $Y = 0, Y = 1, Y = 0, Y = 2$. The Human is only accurate on Region 0.

We consider a hypothesis class \mathcal{F} parameterized by a scalar $c \in \mathbb{R}$ and four indices each in $i_0, i_1, i_2, i_\perp \in \{0, 1, 2, 3\}$. Let $f_i(x) = c\mathbb{I}\{x \in R_i\}$, a function $f \in \mathcal{F}$ defines a rejector and classifier as: $m(x) = \arg \max\{c \cdot f_{i_0}(x), c \cdot f_{i_1}(x), c \cdot f_{i_2}(x)\}$ (ties are decided uniformly randomly) and $r(x) = \mathbb{I}\{c \cdot f_\perp(x) > \max\{c \cdot f_{i_0}(x), c \cdot f_{i_1}(x), c \cdot f_{i_2}(x)\}\}$. This hypothesis class is closed under scaling.

The error minimizing function f^* in this hypothesis class is obtained by setting $c > 0, i_0 = 2, i_1 = 1, i_2 = 3, i_\perp = 0$ which obtains zero 0-1 error. No solution with $c < 0$ is optimal, since the maximum will always coincide with at least two labels and we break ties in a consistent fashion. This data distribution and hypothesis class is realizable.

Surrogate solution. We will argue that one can obtain a lower L_{CE} loss by deviating from the optimal solution f^* . The intuition for why this is the case is that the L_{CE} penalizes misclassifying points even when they are deferred. Hence, when α is sufficiently large, L_{CE} will try to classify the more probable region R0 as label 0 instead of simply deferring on this region and classifying region R2 as label 0.

Consider the function \hat{f} defined with arbitrary $c > 0$ and $i_0 = 0, i_1 = 1, i_2 = 3, i_\perp = 0$ —note that this function disagrees with the optimal solution on i_0 only. Fixing c , we will compute the difference of L_{CE} loss between \hat{f} and f^* with the same c , this defines only a deviation in terms of i_0 . We will compute the difference in each region separately.

Region 1 and Region 3: On both region 1 and region 3, the difference will be shown to be zero. In both regions, the human is incorrect and note that i_1 and i_2 are identical in both solutions. The

loss of \hat{f} in region 1 is:

$$-\frac{1}{4} \log \left(\frac{e^c}{3 + e^c} \right)$$

this is the same as the loss of f^* , by symmetry the loss is the same in region 3.

We will now compute the sum of the difference in region 2 and region 0:

Region 2: In this region the human is also incorrect, the difference in the loss of \hat{f} and f^* is:

$$\mathbb{E}_{x \in R2} [L_{CE}(f^*) - L_{CE}(\hat{f})] = \left(\frac{1}{4} - \alpha \right) \cdot \left(\log \left(\frac{1}{4} \right) - \log \left(\frac{e^c}{3 + e^c} \right) \right) \in \left[-\left(\frac{1}{4} - \alpha \right) \log(4), 0 \right]$$

Region 0: In this region the human is correct, the difference is :

$$\begin{aligned} \mathbb{E}_{x \in R0} [L_{CE}(f^*) - L_{CE}(\hat{f})] \\ = \left(\frac{1}{4} + \alpha \right) \cdot \left(-\log \left(\frac{1}{3 + e^c} \right) - \log \left(\frac{e^c}{3 + e^c} \right) + \log \left(\frac{e^c}{2 + 2e^c} \right) + \log \left(\frac{e^c}{2 + 2e^c} \right) \right) \end{aligned}$$

To compute the difference in the loss between \hat{f} and f^* , we sum the difference in Region 2 and Region 0:

$$\begin{aligned} L_{CE}(f^*) - L_{CE}(\hat{f}) \\ = \frac{1}{4} \left(\log \left(\frac{1}{4} \right) - \log \left(\frac{e^c}{3 + e^c} \right) - \log \left(\frac{1}{3 + e^c} \right) - \log \left(\frac{e^c}{3 + e^c} \right) + 2 \log \left(\frac{e^c}{2 + 2e^c} \right) \right) \\ + \alpha \left(-\log \left(\frac{1}{3 + e^c} \right) + 2 \log \left(\frac{e^c}{2 + 2e^c} \right) - \log \left(\frac{1}{4} \right) \right) \\ = -\left(\frac{1}{4} + \alpha \right) \left(\log \left(\frac{1}{3 + e^c} \right) - 2 \log \left(\frac{e^c}{2 + 2e^c} \right) \right) - \frac{1}{2} \log \left(\frac{e^c}{3 + e^c} \right) + \left(\frac{1}{4} - \alpha \right) \log \left(\frac{1}{4} \right) \end{aligned}$$

We can simplify this difference to further become:

$$\frac{1}{4} (8\alpha c - 2 \log(4) - 2(1 + 4\alpha) \log(1 + e^c) + (3 + 4\alpha) \log(3 + e^c))$$

Note that when $c = 0$, the above difference is 0. Let us set $\alpha = 0.125$ for concreteness (other values of α also work, in particular larger values, but not all smaller values). We compute the derivative of the difference with respect to c , obtaining:

$$\begin{aligned} \frac{d}{dc}(L_{CE}(f^*) - L_{CE}(\hat{f})) &= \frac{1}{4} \left(\frac{3.5e^c}{e^c + 3} - \frac{3e^c}{e^c + 1} + 1 \right) \\ &= \frac{0.375(2 - e^c + e^{2c})}{(1 + e^c)(3 + e^c)} > 0 \end{aligned}$$

We just showed that the difference has derivative strictly larger than 0 with respect to c , moreover the difference is 0 when $c = 0$, thus when $c > 0$ the difference is strictly bigger than 0.

We just proved that with respect to the surrogate loss L_{CE} , the optimal solution with respect to L_{def}^{0-1} is not optimal, thus the surrogate is not a realizable $(\mathcal{M}, \mathcal{R})$ -consistent surrogate for L_{def}^{0-1} □

Appendix C

Additional Information for Chapter 4

C.1 Extended Related Work

Human-AI interaction. A significant amount of research has tried to understand the role of explanations on Human-AI team performance. [88] investigates the role of increasing levels of AI explanation on performance and find that beyond showing predicted labels accuracy does not increase. [89] identify different regularizers that optimize for factors that help humans better simulate and verify AI predictions on recommendation tasks. [90] investigates how the ability of humans to provide feedback to the model reduced user frustration on a text classification task. [91] evaluated different explanation methods on the adult income dataset and on a movie reviews dataset found that only LIME helped for simulating the model and that subjective user ratings of explanation quality were not predictive of effectiveness. More research on the adult income dataset found that showing AI confidence improved trust but failed to improve AI-assisted accuracy [92]. [93] studies how different types of errors an AI may have will lead to different perceptions of the AI by the user, and how setting expectations of the AI capabilities (e.g. its accuracy) improves the user experience. [130] show on a beer/book reviews sentiment classification task and on LSAT multiple choice questions that AI explanations beyond confidence scores don't improve performance but rather increase blind trust in the AI system. [231] on the task of annotating clinical texts show that clinicians generally build a mental model of when to rely on automation, however, when the AI presents a complete suggestion versus an incomplete one, this causes experts to show less agency and makes them more likely accept wrong answers. In similar lines, [94] studied how do humans incorporate AI recommendations as a function of their correctness and their prior knowledge of machine learning, and showed that people follow incorrect AI recommendations for tasks they predominantly complete correctly and that incorrect-abnormal recommendations were followed significantly less than incorrect normal recommendations. [232] on the task of age prediction from images showed that the addition of explanations in the form of saliency maps did not improve

accuracy nor did the quality of the saliency maps have much impact. [95] propose to visualize a given input's nearest neighbors to help better reason about the model's uncertainty and show an editor that allows users to edit aspects of the input and see how model predictions change, they found that this interface allowed some clinicians to build better intuition about the AI capabilities and limitations. Finally, a line of work has focused on human-AI interaction in healthcare applications: on chest X-rays [2], [233], diabetic retinopathy [1], skin cancer [234] and breast cancer [235]. [104] study the effect of initial debriefing of stated AI accuracy compared to observed AI accuracy in deployment and find a significant effect of stated accuracy on trust, but that diminishes quickly after observing the model in practice; this reinforces our approach of building trust through examples that simulate deployment.

Explainability. Methods for explaining the decisions of ML models range from feature attribution (e.g. LIME [38]), saliency methods for computer vision tasks (Grad-CAM [236]), Example-based explanations [237] and others. One of the basic forms of model explanations is calibrated confidence scores [238]–[240] These methods for explainability start from a set of desiderata (natural assumptions of what an explainability method should provide) and then formulate a given method that can be implemented without further data requirements. The common pitfall of these methods is that they are agnostic to the downstream expert, the desiderata is formulated from a perspective of a rational expert and are sometimes justified from user studies.

Machine Teaching. Machine teaching (MT) refers to the problem of choosing a minimally sized dataset that enables a student learner to learn a specific target function [42]. Given a hypothesis class, its teaching dimension is the smallest sized set that enables an ERM learner to pick out the optimal classifier [107], [241]. To mimic human learners, [106] proposes a Bayesian learner based on a prior over a discrete hypothesis class, the learner maintains a distribution over each hypothesis that updates with each teaching example. They evaluate their approach on an image classification task where crowdworkers learn to distinguish different animals. This setting was extended to include explanations in the form of attention [41], [242] and errors in learning priors and teacher knowledge [243]. [109] aims to teach a consistent black-box learner, while this formulation is attractive in regards to a human learner, the algorithm they provide requires an excessive amount of queries to the human that go beyond the teaching examples presented. [108] teaches a forgetful human learner multiple concepts where each concept maps to a single example, but the human may forget the concept later on. Our work separates itself by the use of a novel radius nearest neighbor model to approximate the human learning process.

Human Learning. [85] make the claim that humans make decision by sampling similar experiences from memory instead of computing reward estimates for each possible action. Their experimental study involves users performing a two-armed bandit task with each example having a unique identifier. [115] make two claims about how humans make decisions: the first is that people often retrieve a limited set of items from memory when making decisions and the second is that training humans on idealized instances is more advantageous than training them on noisy or hard instances. They base their claims on two experiments: one where humans classify horizontal lines of different lengths and the other where they judge outcomes of baseball games. [116] review the literature on visual category learning, how we distinguish between different visual objects. They make a distinction between two different models of human decision making. The first is example-based that models assume that a category is represented in terms of the particular exemplars that have been experienced during learning. The other is rule based, people try to explicitly learn categories by forming simple rules. The conjecture is that for hard tasks, the example-based model is more accurate while for simpler ones, the rule-based approach is the driver.

Nearest Neighbor Compression. Our human student model is a more general case of a weighted nearest neighbor learner, this makes the teaching problem equivalent to that of compressing the number of samples nearest neighbors requires. Seminal work on compressing nearest neighbors introduced the condensed nearest neighbor rule [244] and follow-up work introduced more robust versions but that still require the existence of a consistent subset [245]. More recent work has focused on the generation of compressed subsets [246]–[248].

C.2 Theoretical Results and Proofs

C.2.1 Further Derivations

We expand on section "Teaching a Student Learner" (4.5) and decompose the loss of the human learner.

Since $\pi_Y(x)$ and $h(Z, A)$ are known and fixed, we can assign to each deferral decision at each point i a cost $c_i(r) \in \mathbb{R}^+$ and abstract away the inner classification decisions:

$$L(D) := \sum_{i \in S} l_c(r(z_i, a_i; D); c_i) \quad (\text{C.1})$$

An example of l_c is $l_c^b := r(x_i; D)c_i(1) + (1 - r(x_i; D))c_i(0)$ which can be made equivalent to the 0-1 classification loss. It may be the case that neither of $c_i(0)$ or $c_i(1)$ are zero since there may be multiple correct decisions or that both be may be non-zero and equal. Now we further decompose the loss L into errors made by the prior and errors due to the learned rejector:

$$L(D) = \sum_{i \in S \mid |B(z_i)| \neq \emptyset} l_c \left(\frac{\sum_{j \in B(z_i)} \mathbb{I}\{r_j = 1\} K(z_i, z_j)}{\sum_{j \in B(z_i)} K(z_i, z_j)}; c_i \right) \quad (\text{errors by learned rejector}) \quad (\text{C.2})$$

$$+ \sum_{i \in S \mid |B(z_i)| = \emptyset} l_c(g_0(z_i, a_i); c_i) \quad (\text{errors by prior}) \quad (\text{C.3})$$

In the paper, we proved a guarantee in Theorem 1 on the performance of the GREEDY-SELECT algorithm when the hyperparameter α is set to 1 when optimizing the loss $L(\cdot)$ (4). The loss L involves the human learner $M(\cdot)$, however, one component of the human learner was left unspecified which is how they set the radius γ following every teaching example z . In what follows, we assume the human is perfectly learning the radius that the teaching process displays to them. Equivalently, when the human is shown the tuple $\{z, \gamma, r\}$ where z is the teaching example, γ is a radius and r is the deferral action, they now follow the deferral action r in the neighborhood of size γ around z .

When we set $\alpha = 1$, this defines a unique radius γ_i to each point $z_i \in S^*$ (the teaching set), this radius defines the largest neighborhood around z_i such that the optimal deferral action in that neighborhood is r_i . Thus our teaching set becomes $S^* = \{x_i, z_i, \gamma_i, r_i\}$ and we can now simplify our optimization problem by only searching for the teaching point z at each step (instead of jointly searching for the radius as well) as the radius is uniquely specified no matter what the current teaching set D_t is.

C.2.2 Proofs

The following proposition is part of the proof of Theorem 1.

Proposition 2. *Let $F(X) = L(\emptyset) - L(X)$, $F(\cdot)$ is submodular, monotone and positive.*

Proof. Monotonicity. We prove that $L(\cdot)$ is monotone decreasing which implies that $F(\cdot)$ is monotone increasing. For notation simplicity we omit the AI message A from the prior rejector and make it only a function of Z , the proof remains valid even if we add A .

Initially $D_0 = \emptyset$ and $L(\emptyset)$ is the error rate of the human's prior rejector g_0 on the set S .

Induction argument: In the first step $D_1 = \{z_{i1}\}$ where z_{i1} is the training example that leads to the biggest error decrease of $L(\cdot)$ (we don't use this fact so that this holds for any training example added, also note that since there is a unique correspondence from z_{i1} to r_{i1} and γ_{i1} we simplify the notation and only write z_{i1}), now note that:

$$L(D_1) - L(D_0) = \sum_{i \in S \text{ s.t. } z_{i1} \in B(z_i)} l_c(r_{i1}; c_i) - l_c(g_0(z_i); c_i) \quad (\text{C.4})$$

Note that other terms in the difference of equation (C.4) cancel out, what is left are points in S that the human starts to use their learned rejector on, i.e. those that are sufficiently close to z_{i1} call these set of points \mathcal{I} . For each $i \in \mathcal{I}$, if it was the case that $g_0(z_i) \in \arg \min_d l_c(d; c_i)$, then we know that r_{i1} and $g_0(z_i)$ have the same cost since r_{i1} is the optimal decision by definition. Now suppose that $g_0(z_i) \notin \arg \min_d l_c(d; c_i)$, then it must be the case that $r_{i1} = 1 - g_0(z_i)$ and this achieves a lower loss than $g_0(z_i)$. Therefore we have that:

$$L(D_1) - L(D_0) \leq 0$$

Now suppose we are at step $t + 1$ of the algorithm and we add example $z_{i(t+1)}$ to obtain $D_{t+1} = \{z_{i1}, \dots, z_{i(t+1)}\}$. Let us compute the difference:

$$L(D_{t+1}) - L(D_t) = \sum_{i \in S \text{ s.t. } B(x_i) = \{z_{i(t+1)}\}} l_c(r_{i(t+1)}; c_i) - l_c(g_0(z_i); c_i) \quad (\text{C.5})$$

Note that if there was point $i \in S$ where there exists $j \in D_t$ such that $z_j \in B(x_i)$, then the addition of $z_{i(t+1)}$ cannot change the final cost assigned to example i as if $z_{i(t+1)} \in B(z_i)$, then we must have $r_{i(t+1)} = r_j$ by assumption 5. Thus the only element remaining in the difference is points that now have a neighbor in D_{t+1} but not in D_t , meaning those that only have $z_{i(t+1)}$ in their ball. The argument is now exactly as in the base case so that:

$$L(D_{t+1}) - L(D_t) \leq 0$$

which gives us the set of inequalities:

$$L(D_m) \leq \dots \leq L(D_0)$$

and note that $L(\cdot)$ achieves it's minimum value at $L(S) = L(D_{|S|}) \leq L(D_m)$.

Positivity. Note that $F(\cdot)$ is positive as we assume l_c is positive and we obtain the result from monotonicity.

Submodularity. To make the proof easier, define the teaching ball $\tilde{B}(D)$ to be the set of points in the training set S that have any teaching point $Z \in D$ in their ball $B(\cdot)$. This implies if $B(z_i) = \{z_j\}$ then $z_i \in (\{z_j\})$; remember that $B(z_i)$ is the set of teaching points that are sufficiently close to z_i . Let $A \subset B \subset S$, let $l \in S \setminus B$, let us compute:

$$F(A \cup \{l\}) - F(A) - F(B \cup \{l\}) + F(B) = L(A) - L(A \cup \{l\}) + L(B \cup \{l\}) - L(B) \quad (\text{C.6})$$

$$= \sum_{i \in S \text{ s.t. } z_i \in \tilde{B}(z_l) \setminus \tilde{B}(A)} l_c(g_0(z_i); c_i) - l_c(r_l; c_i) + \sum_{i \in S \text{ s.t. } z_i \in \tilde{B}(z_l) \setminus \tilde{B}(B)} l_c(r_l; c_i) - l_c(g_0(z_i); c_i) \quad (\text{C.7})$$

$$= \sum_{i \in S \text{ s.t. } z_i \in (\tilde{B}(z_l) \cap \tilde{B}(B)) \setminus \tilde{B}(A)} l_c(g_0(z_i); c_i) - l_c(r_l; c_i) \geq 0 \quad (\text{C.8})$$

The last term is positive as the optimal decisions r_i always improve on the prior. □

Theorem 1. *Let $F(X) = L(\emptyset) - L(X)$, $F(\cdot)$ is submodular, monotone and positive. Moreover, the GREEDY-SELECT algorithm described above achieves the following performance compared to the optimal teaching set D^* :*

$$\underbrace{L(D_m)}_{\text{loss of chosen set}} \leq \left(1 - \frac{1}{e}\right) \underbrace{L(D^*)}_{\text{loss of optimal set}} + \frac{1}{e} \underbrace{L(\emptyset)}_{\text{loss of prior rejector}} \quad (\text{C.9})$$

Proof. The first statement of the theorem is proved in Proposition 2.

For the second statement of the theorem, the proof is simply restating the proof of Theorem 1.5 in [249] in the context of our problem which we do here for clarity. Let $D_i = (z_1, \dots, z_i)$ the set that our algorithm produced at round i and $D^* = (z_1^*, \dots, z_K^*)$ the optimal set.

For all $i \leq m$:

$$F(D^*) \leq F(D^* \cup D_i) \quad (\text{monotonicity}) \quad (\text{C.10})$$

$$= F(D_i) + \sum_{j=1}^m F(D_i \cup D_{j-1}^* \cup z_j^*) - F(D_i \cup D_{j-1}^*) \quad (\text{telescoping}) \quad (\text{C.11})$$

$$\leq F(D_i) + \sum_{z \in D^*} F(D_i \cup z) - F(D_i) \quad (\text{submodular } F) \quad (\text{C.12})$$

$$\leq F(D_i) + m(F(D_i \cup z^{i+1}) - F(D_i)) \quad (\text{optimality of } z^{i+1}) \quad (\text{C.13})$$

$$(\text{C.14})$$

re-arranging this final inequality with $\delta_{i+1} = F(D^*) - F(D_i)$ we get:

$$\delta_{i+1} \leq \delta_i \left(1 - \frac{1}{m}\right)$$

iterating this last inequality till m , using the fact that $1 - x \leq e^{-x}$ and restating things in terms of $L(\cdot)$ gets the final result in the theorem. \square

C.2.3 Hardness result

Theorem 1 gives a guarantee on the subset chosen by the greedy algorithm with an $1 - \frac{1}{e}$ approximation factor, one can ask if we can do better. We prove that a generalization of our problem under Assumption 5 is in fact NP-hard.

Proposition 1. *Problem (4.12) is NP-hard.*

Proof. For simplicity we assume that the AI and human domains are identical and don't consider the AI message in the human rejector or predictor. The proof can be straightforwardly extended to the case when the domains differ and including the AI message.

Suppose we are given a collection of finite sets A_1, \dots, A_n jointly covering a set W . We reduce the problem of finding a smallest subcollection covering W to the teaching problem (4.12).

Let $S_V = W$, for each A_j , we associate it with a new teaching example $x_j \in S_T$ (unique from all elements of S_V and other elements of S_T) such that its neighbors are exactly the elements of A_j i.e. $K(x_j, x) = \infty$ iff $x \in A_j$ and $K(x_j, x) = 0$ iff $x \notin A_j$ (we construct the function K specifically to satisfy these requirements). Now we set the label $y_i = 1$ for each example $i \in S_V \cup S_T$ and let $h(x) = 1$ (human predictor) and $\pi_V(x) = 0$ (AI predictor) for all x and we set $g_0(x) = 1$ (human prior rejector) so that the prior is wrong on all example: we should never defer while the prior always defers so the correct deferral decision is $d_i = 0$ (derived deferral decision) for all examples. We set the loss l_c to simply be the 0 – 1 deferral loss (cost of 1 incurred if final prediction disagrees

with label, otherwise a cost of 0), with this in mind note that $L_V(\emptyset) = |S_V|$ as with $D = \emptyset$ we use the prior rejector on all examples which always errs.

Once we pick a new example x_j (correspondence to the set A_j) to our set D that we are choosing, the only terms that are affected are those that are close to x_j which are exactly the elements of A_j , so that $L_V(\{x_j\}) = |S_V| - |A_j|$. Iteratively, when we add another example x_k to D the only terms affected are those in the neighborhood of x_k which are A_k , but now it may be the case that $A_j \cap A_k \neq \emptyset$, however since the deferral label associated to all examples is the same, which is to not defer, the loss of the elements in the intersection are not affected (in essence there is no double counting of the elements) so that now: $L_V(\{x_j, x_k\}) = |S_V| - |A_j \cup A_k|$. It is now clear to see that solving problem (4.12) with $\delta = 0$ finds a set cover of W with elements A_1, \dots, A_n as $L_V(D)$ simply counts how many elements of S_V (correspondence to W) we don't apply the prior rejector to (i.e. elements we cover).

□

C.2.4 Efficient Implementation of Greedy Selection

When $\alpha = 1$, we provide an efficient implementation of the greedy selection algorithm GREEDY-SELECT.

At each round, we have a teaching set D_t from which we can construct a rejector function $g_t(\cdot, \cdot)$, at D_0 we have g_0 is the prior rejector. Now at round t , we calculate for each example on the training set S the following quantity

$$E_i^t = \sum_{j \in S \mid K(z_j, z_i) \geq \gamma_i} \mathbb{I}_{g_t(z_j, a_j) \neq r_j} \quad (\text{C.15})$$

E_i counts the number of points that are in the neighborhood of z_i that the current human rejector g_t misclassifies, in other words it measures for each point i how many points in the training set it will cause their deferral label to flip. Note that we are guaranteed that once a point is close enough to the teaching point z_i , it's deferral decision becomes optimal by Assumption 5. At each round t we pick the point $i^* = \arg \max_i E_i^t$.

This algorithm has run-time $O(n^2 m)$ where $n = |S|$, while the naive implementation of the algorithm has run-time $O(n^2 m^2)$, the extra m factor comes from having to simulate the human rejector to calculate the resulting loss.

When we are optimizing over the choice of radius jointly with the choice of training point, we have no other choice but to fully simulate the human rejector. But note that the optimization over the radius can be reduced to only looking at radius choices that are equal to kernel similarities on the training set.

C.3 SAE Model Error Analysis

Predictions. The below analysis is performed from allowing the model SAE-large model [121] whose code is available at ¹ to predict on the HotpotQA DEV set [86] with no distractor paragraphs. The model is ranked 20'th on the public leaderboard, and is the highest ranking model with publicly available code.

C.3.1 Factors of difference

Presence of distractors. There are two types of question answer types in HotpotQA: yes/no answers and answers that are substrings from the passage. We eliminate yes/no questions and only focus on questions that admit an answer inside the passage which makes the validation set of size 6947 out of an original 7405. We note that the absence of distractor paragraphs does not boost

Table C.1: Performance on the dev set without yes/no questions.

FACTOR	EXACT MATCH (EM)	F1
8 DISTRACTORS	66.92	79.62
NO DISTRACTORS	68.79	82.75

performance by a significant amount. In fact the model SAE first consists of a relevant paragraph extractor that feeds into the RoBERTa reader and that extractor works quite well as evidenced.

Bridge vs comparison questions. The questions in HotpotQA can be categorized into two types: **bridge** e.g. "“when was the singer and songwriter of Radiohead born?”", to answer this question one first has to figure out who is the singer of Radiohead and then look up his date of birth, the other type are **comparison** questions such as that “Who has played for more NBA teams, Michael Jordan or Kobe Bryant?”. This categorization is provided already in the dataset.

Table C.2: Performance based on question types.

FACTOR	EXACT MATCH (EM)	F1
BRIDGE	68.31	83.25
COMPARISON	71.52	79.86

¹<https://github.com/JD-AI-Research-Silicon-Valley/SAE>

We can see that there is a difference in how question types affect performance, however it is not consistent across the two metrics to make a definite conclusion.

Passage Lengths. Given the length of the two golden paragraphs, is there a difference in the performance over different sizes? As we can see below we observe no significant difference, in the last bucket of long passages we see a notable increase in F1 but that is due to limited sample size in extremely long passages.

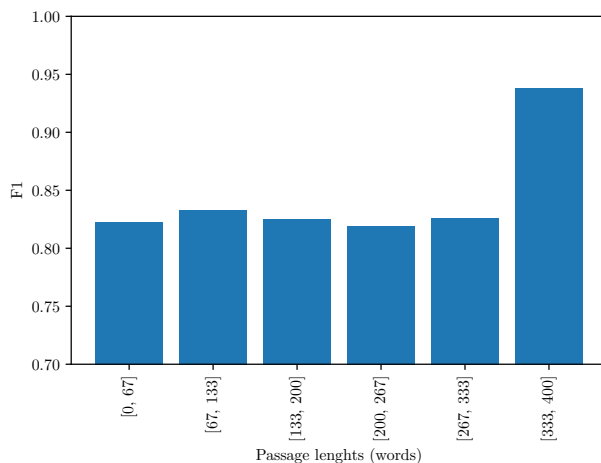


Figure C.1: Performance across lengths of passages in terms of words. First bin contains very little samples to be significant.

Supporting fact lengths. We plot the performance versus the number of supporting facts: the number of sentences one must read to answer the question, this is provided in the dataset explicitly. Note there are at least two sentences that one must read since all questions are multi-hop. We can see that there is no real difference across all lengths.

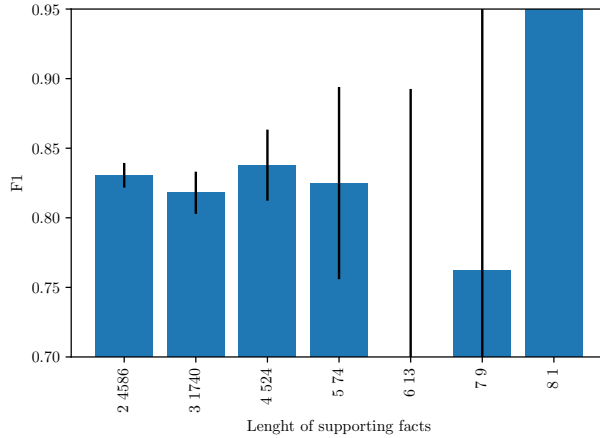


Figure C.2: Performance across number of supporting sentences. Black bars indicate 95% confidence interval around the mean, the x axis is: (number of sentences, number of examples with that many sentences)

Passage and Question topics. We try to see if there is a difference in performance when looking into the topics that the examples belong to. We first run an LDA with 15 topics on the passage concatenated with the question (we use the *gensim* package [250]). We then categorize each example according to the topic with the largest coefficient in the LDA decomposition.

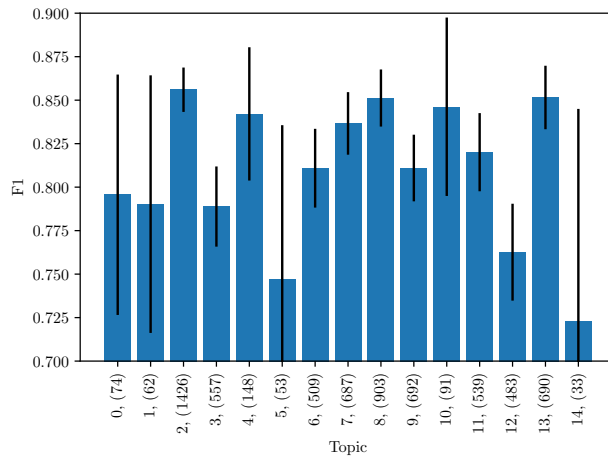


Figure C.3: Performance across LDA topics

Plotted in Figure C.3 are mean F1 across topic and 90% confidence intervals and we observe no particular topic that has significant difference from others.

Question words. We investigate difference in performance depending on the question word present in the question.

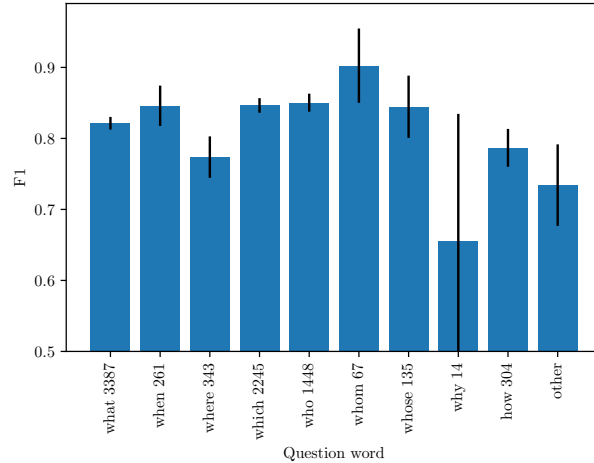


Figure C.4: Performance (left EM, right F1) across question words

We can see that there is significant difference with "why" questions (however they are rare) and "how" questions to a lesser degree.

C.3.2 Embedding clustering

Model embeddings. The SAE model last layer consists of a 512x1024 tensor: a 1024 representation of 512 tokens. This representation is then used to predict for each token the probability that it is the start or end of the answer with a linear layer. To get a vector representation of each example, we average out across tokens to obtain a single 1024 vector for each example. We take these vectors and cluster them using K-means (we do the analysis for multiple k's). We then plot the performance across each cluster below.

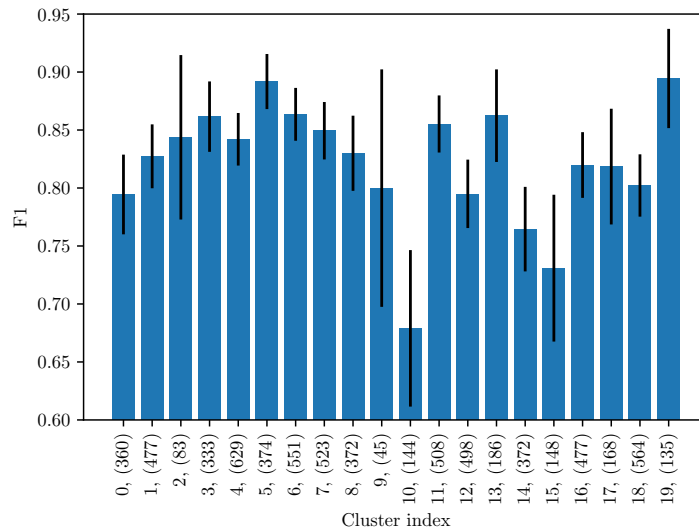


Figure C.5: Performance (left EM, right F1) across model embeddings clusters.

We observe that cluster 10 has lower performance than average by a significant amount. Looking at examples from that cluster, no apparent theme emerges.

Passage embeddings. We use the BERT sentence encoder² to get embeddings for the passage and cluster them using k-means. We repeat the exact process for the questions and answers.

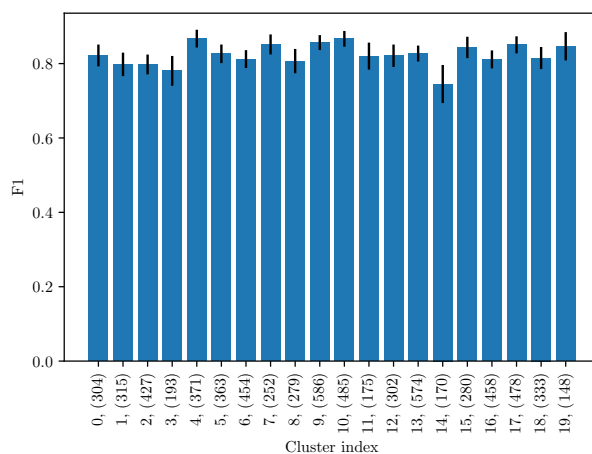


Figure C.6: Performance across passage embeddings clusters. No differences emerge significantly.

Question embeddings. We can see that cluster 13 underperforms, examining that cluster we can see a pattern of questions like "What city does Paul Clyne and David Soares have in common?", the theme is the "in common" at the end of the question.

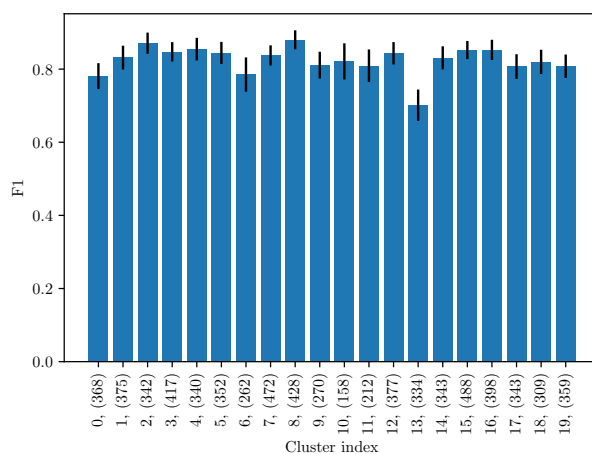


Figure C.7: Performance across question embeddings clusters.

Answer embeddings. We observe no observable theme or significant differences.

²<https://github.com/UKPLab/sentence-transformers>

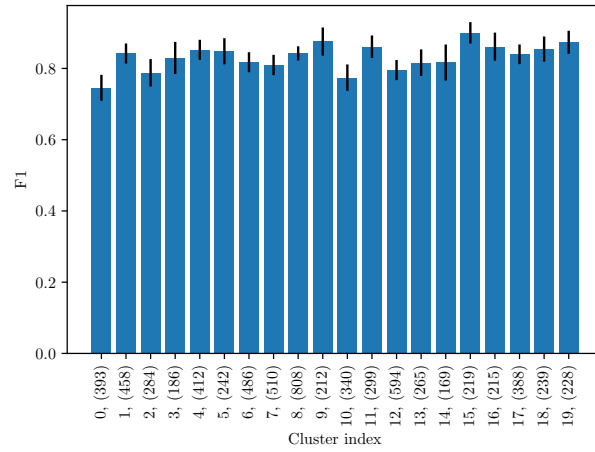


Figure C.8: Performance across answer embeddings clusters.

C.4 Synthetic Experiments Details and Results

All experiments were run on a Linux system with a NVIDIA Tesla K80 GPU, 25 GB of RAM on Python 3.7. We use the scikit-learn package to run the clustering algorithms [251], LIME package for the selection baseline [38]³, ELI5 package to obtain the text LIME highlights⁴ and the Sentence Transformers package for the embedding models [78]⁵

C.4.1 Misspecification results

To evaluate how much information about h we need to properly teach the human, we learn a teaching set assuming the human’s error probability is $err_p' + \delta$ where δ has each component drawn from $\{-\delta, \delta\}$ uniformly where $\delta > 0$. Figure C.9 shows the difference to ORACLE accuracy as we increase the misspecification of the human predictor. In this experiment, we assume knowledge of the prior rejector g_0 and that the human is perfectly learning the radius given by the teaching algorithm. What this experiment impacts is the computation of the optimal deferral decision r_i computed by our algorithm to obtain S^* . At the limit when $\delta = 0.5$, we assume that the human expert error rate is uniformly 0.5 across the domain, which is the same as having the human predictions $h \sim Bin(1/2)$ on the teaching set.

In Table 1 in the paper, we evaluate what happens when the human is not learning the radius perfectly, this simulates noise in the learning process. The radius γ_i that the human learns is a noisy version of $\hat{\gamma}_i$, specifically we add a uniformly distributed noise $\delta \sim \mathcal{U}(-(1 - \hat{\gamma}_i)/2, (1 - \hat{\gamma}_i)/2)$.

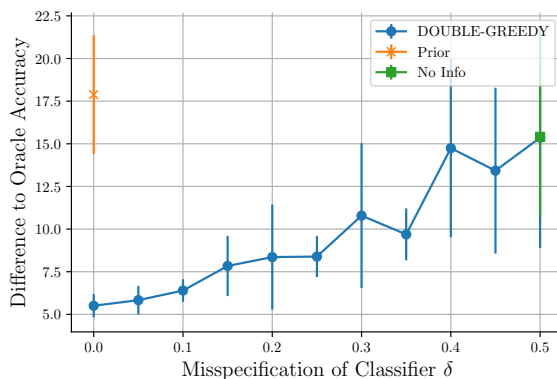


Figure C.9: Difference in Oracle accuracy at teaching size @T=30 for the DOUBLE-GREEDY method assuming an error in h by δ in setting B.

³<https://github.com/marcotcr/lime>

⁴<https://eli5.readthedocs.io/en/latest/index.html>

⁵<https://github.com/UKPLab/sentence-transformers>

C.5 Additional Synthetic Experiments

C.5.1 CIFAR-10

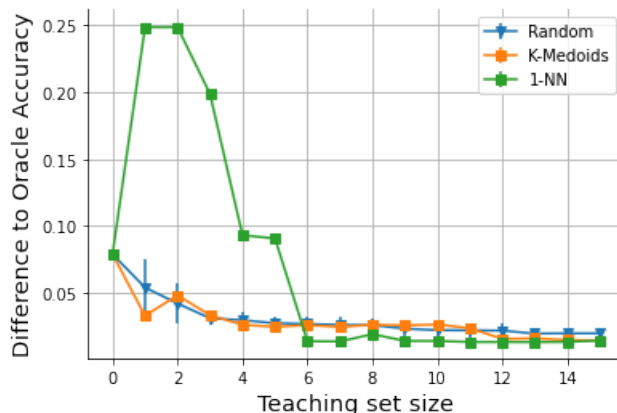


Figure C.10: Comparing a 1-nearest neighbor rejector model to the radius nearest neighbor model introduced in Assumption 4 for expert $k = 6$. The "1-NN" line is obtained by first obtaining T points using K-medoids and then running a 1-NN rejector on these points with the label assigned to each point being the optimal deferral decision r_i . We can see that 1-NN struggles with less than 6 examples, but then reaches a steady state that has the same error as the radius nearest neighbor model. The effectiveness of the radius nearest neighbor model when the teaching set is very small is due to the local nature of each update with the addition of a teaching example.

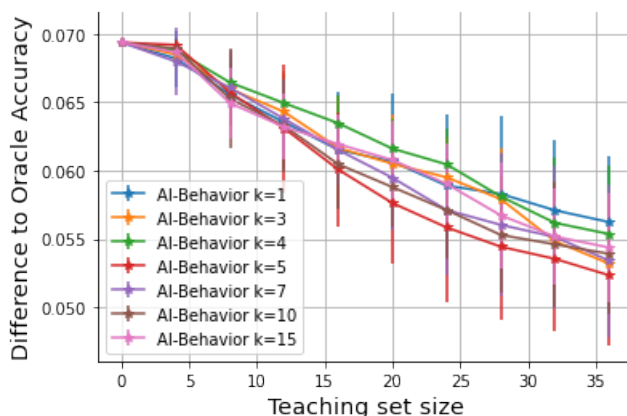
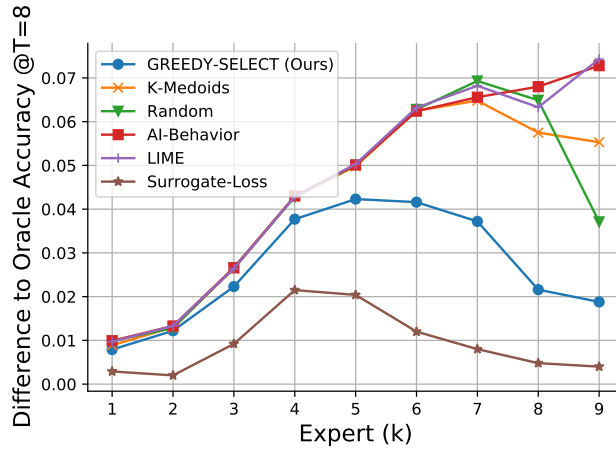
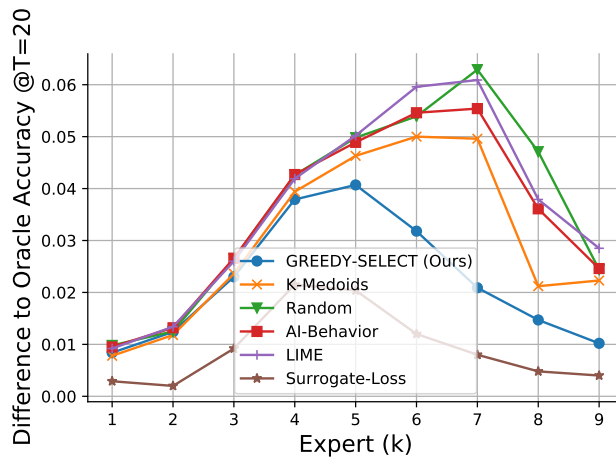


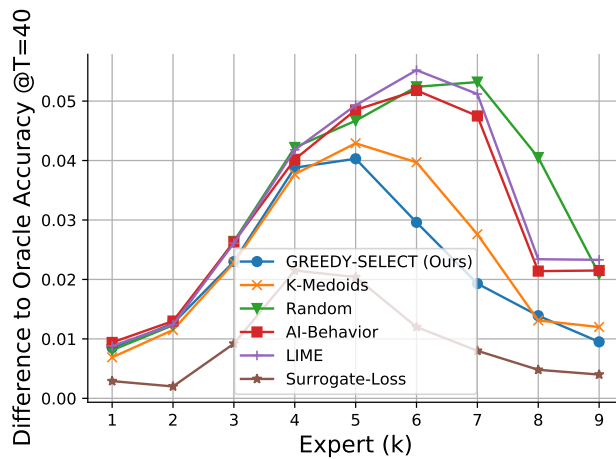
Figure C.11: Performance of the AI-Behavior baseline as we vary the parameter K : the AI-Behavior baseline uses a K -nearest neighbor rejector and at each teaching step selects the point that best reduces the error of the rejector at detecting the AI's errors. We show results for the human expert $k = 6$ with the consistent radius strategy $\alpha = 1$. We can see that the parameter K has little effect and thus we use a natural choice of $K = 6$.



(a) Teaching size of 8 points

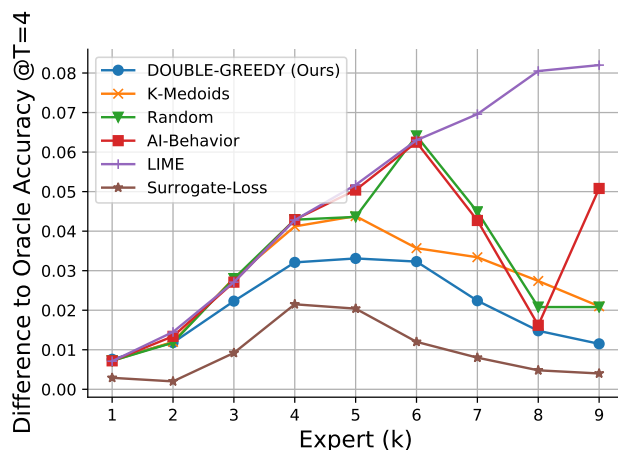


(b) Teaching size of 20 points

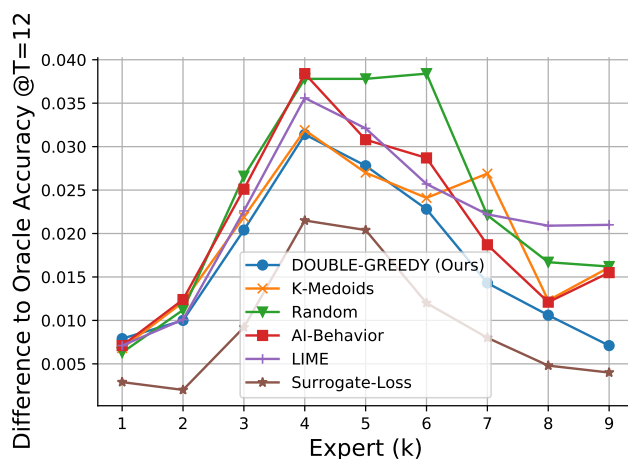


(c) Teaching size of 40 points

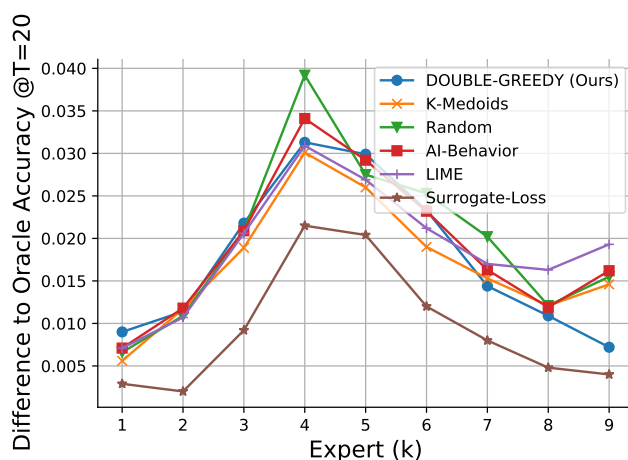
Figure C.12: Extended legend: Varying the human parameter k (number of classes human can classify) and plotting the difference to oracle accuracy for all the baselines when using the consistent radius strategy including the surrogate-loss learning to defer method of [26] at 3 different teaching set sizes.



(a) Teaching size of 4 points



(b) Teaching size of 12 points



(c) Teaching size of 20 points

Figure C.13: Extended legend: Varying the human parameter k (number of classes human can classify) and plotting the difference to oracle accuracy for all the baselines when using DOUBLE-GREEDY including the surrogate-loss learning to defer method of [26] at 3 different teaching set sizes.

C.5.2 Gaussian Data Illustration

Figure 1 illustrates the rejector for a linear classification setting, here we formalize this as a mixtures of Gaussian setup and show the performance of our selection algorithm both quantitatively and qualitatively.

Setup. As an illustrative setting where we can visually inspect the teaching set, we perform experiments on two dimensional Gaussian mixture data. The covariate space is $\mathcal{X} = \mathbb{R}^2$ and target $\mathcal{Y} = \{0, 1\}$, we assume that there exists two sub-populations in the data denoted $A = 1$ and $A = 0$. Furthermore, $X|(Y = y, A = a)$ is normally distributed according to $\mathcal{N}(\mu_{y,a}, I)$. The group proportion is $\mathbb{P}(A = 1) = 0.5$ and the means are sampled from a uniform distribution. The AI follows the Bayes solution for group $A = 1$ which here corresponds to a hyperplane and the human classifier follows the Bayes solution for group $A = 0$, which is another hyperplane. We assume the human’s prior rejector is to reject based on a tresholding of the predictor confidence i.e. $g_0(x) = \mathbb{I}\{|h(x)| \leq \epsilon\}$. We assume that the similarity kernel is the RBF kernel $K(x, x') = e^{-\|x-x'\|^2}$.

Results. For 100 trials, we generate data with random means and measure the difference in system accuracy between our approach and the baselines as we vary the size of the teaching set. Results are shown in Figure C.14. Figure C.15 shows the points chosen on a given configuration.

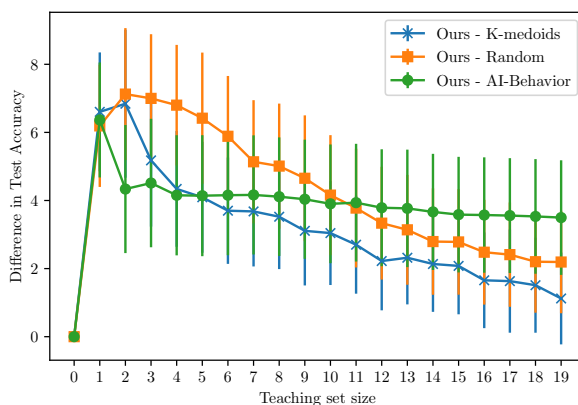
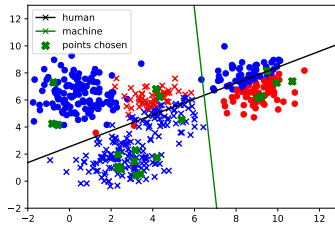
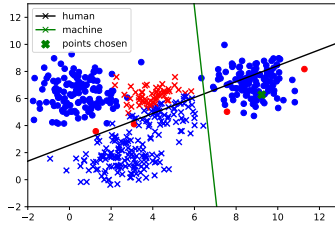


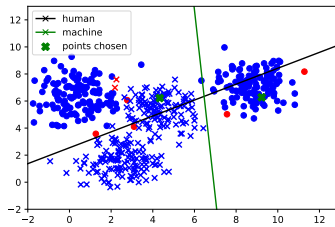
Figure C.14: Teaching complexity plot for synthetic Gaussian data setup. The x-axis shows the difference in test human accuracy between our method and the baselines. Plotted are the averages over the 100 trials along with 95% confidence interval error bars for the average.



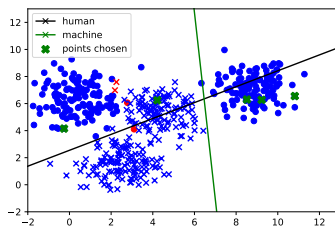
(a) Prior rejector with points chosen at step 20.



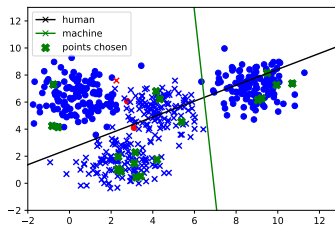
(b) Step 1 .



(c) Step 2 .



(d) Step 5 .



(e) Step 20 .

Figure C.15: Extended legend: blue dots indicate a correct decision while red dots indicate mistakes. Points with an "x" are labels 1 while points with an "o" are labels 0 (in the Y space). The lines labeled human and machine are the respective classifiers.

C.6 Crowdsourced Experiments Details and Results

C.6.1 Experiment Details

Participants. We recruited 50 US based participants from Amazon Mechanical Turk per each condition (100 total), workers were required to have a HIT approval rate higher than 95% and over 100 HITs approved. Initial pilot studies were also conducted with graduate students in computer science at a US university. Participants in the baseline were paid \$3 for 10 minutes of work and those in the teaching condition received \$6 for 20 minutes of work. Any demographic information we gathered in our study is kept confidential and workers were asked to consent to their use of their responses in research studies. We submitted an IRB application and the IRB declared it exempt as is. We followed standard protocol and additionally provided the IRB exemption and details to our user study participants. We filter participants who don't answer the tutorial questions correctly and we also filter for all baselines that workers at least answer one question correctly on their own beyond the first question.

AI and Test Set details. The simulated AI used in the study was obtained by first performing K-means with $K = 25$ on the dev set of HotpotQA, and then manually filtering the data to obtain 11 clusters that are more distinct. The test set used in the testing phase was obtained first by filtering the data using K-medoids with $K = 200$ as a way to get diverse questions. We then created 20 test sets by sampling 7 random questions from the filtered set on which the AI was correct and 8 on which the AI is incorrect. The order of the examples in the test set was shuffled for each participant.

Cluster Topics. The AI used in the study had 11 different clusters on which it's errors were defined. Table C.3 shows the main theme and most common Wikipedia categories for each cluster.

User Lessons. In Table C.4 we show examples of the lessons that the crowdworkers wrote during the teaching phase for the proposed teaching method. We show examples of the lessons on the first 3 examples in the teaching phase and separate the participant lessons into 4 categories: participants who wrote accurate lessons, participants who wrote irrelevant lessons (not relevant to the question or required no effort to write), participants who wrote complex lessons that don't pertain to the example topic and finally participants who wrote narrow lessons that are on topic but only apply to the example and not the neighborhood of the example. In Table 4.3 we separated user metrics into two groups accurate lessons and inaccurate lessons, this corresponds to grouping accurate lessons versus the rest in the lesson categorization of Table C.4. Furthermore, in the body of section 4.6.4 we distinguish between accurate lessons, narrow and complex lessons (combined into one group)

Table C.3: Cluster main theme (manually obtained) and top 3 Wikipedia categories of examples in clusters for the AI used in the MTurk study.

Cluster ID	Main Theme	Wikipedia Categories
1	Plants	Poaceae genera, Flora of Mexico, Dioecious plants
2	Singers, Musicians	21st-century American singers, Grammy Award winners, American male guitarists
3	Movies, Actors	American films, British films, American male film actors
4	Sites, Hotels	Casino hotels, Casinos in the Las Vegas Valley, Resorts in the Las Vegas Valley
5	Writers, Magazines	20'th-century American novelists, American male non-fiction writers, American women novelists
6	Composers, Plays	19th-century classical composers, Operas, Male classical pianists
7	Games	Windows games, PlayStation 4 games, Xbox One games
8	Universities	Universities and colleges, Colonial colleges, Private universities in New York
9	Soccer	Premier League players, English Football League players, Association football midfielders
10	Sports (non soccer)	American men's basketball players, NFL player, NBA All-Stars
11	Politics	21st-century American politicians, Presidential Medal of Freedom recipients, Republican Party members

and finally irrelevant lessons.

C.7 Extended Discussion

One limitation of our human experiments is that we used a simulated AI that has an easier to understand error boundary. This enabled us to have a more in-depth study of the crowdworker responses than otherwise would have been possible.

Having a simulated AI to which we perfectly understand where it's error regions are (but note this is highly non trivial for someone who doesn't know how it was trained), enables to define what the "lessons" should be and thus evaluate if users are learning correctly. This ability to evaluate if users are actually learning through their written lessons enables to test two things:

1. Do people learn the correct lessons using our teaching method?
2. Do those who learn the correct lessons apply them perfectly?

And our answers in our paper to these questions are: 1) yes but only half the people are able to, 2) not quite, since even those with perfect lessons don't show perfect accuracy (in Defer F1).

What is interesting about this last observation tell us that even if people know the rules, and have them written and shown on the screen, they might still apply it incorrectly. With a non simulated AI, it would have been difficult for us to figure out the answers to the questions as the underlying lessons are not pre-determined. For an initial experimental study on teaching, we need to understand better how do humans make decisions and how we can try to use their lessons to possibly provide feedback and better guide them.

Another limitation is that our test-time interface did not include model explanations or predictions. This was done for multiple reasons:

- The AI predictions and explanations reveal information about it's underlying performance at test time. If two different crowdworkers received different test sets, then their knowledge about the AI may be different. Therefore if the participants belonged to two different experimentation conditions, then the test set becomes a confounding factor we need to control for.
- When model explanations are not available or are not effective, the effect of teaching becomes more important as it is the only way the human's mental model is formed. Thus the choice of the teaching method becomes more important.
- If the AI prediction is available at test time and workers press on the "Use AI answer" button, there is an unobservability issue that arises: are workers pressing the button because they trust the AI, or are workers pressing the button because they came up with the same answer on their own? Removing their ability to see the AI prediction alleviates the problem.

Table C.4: Example of lessons that users in the Ours-Teaching condition wrote during the teaching phase. We show examples of the lessons on the first 3 examples in the teaching phase and separate the participant lessons into 4 categories: participants who wrote accurate lessons, participants who wrote irrelevant lessons (not relevant to the question or required no effort to write), participants who wrote complex lessons that don't pertain to the example topic and finally participants who wrote narrow lessons that are on topic but only apply to the example and not the neighborhood of the example.

Lesson Type	Example ID	Actual Lesson
Accurate Lessons	1	The AI is not good at answering questions about plants.
Accurate Lessons	2	The AI is better at Politics and geography than at sports.
Accurate Lessons	3	The AI is bad at answering questions about movies
Irrelevant Lessons	1	I understood AI is good at answering
Irrelevant Lessons	2	AI focus on the institution
Irrelevant Lessons	3	AI omitted important terms
Complex Lessons	1	It seems to be better at answering questions where the absolute same phrases are used in the question as the passage and where both answers are in the question, maybe?
Complex Lessons	2	The ai is good at answering questions that has to do with cities and numbers though not good with words that has to do with repeated words.
Complex Lessons	3	The AI can't decipher clues, example, the other movie was based on a book that came out after the other movie but the AI couldn't figure out that that must mean the movie based on that book must then also have come out after the other movie.
Narrow Lessons	1	The AI isn't good at multi-faceted questions about continental species.
Narrow Lessons	2	The topic was politics and the AI is good at answering questions about specific areas when the question can be answered by looking for specific information about one section but not when it involves integrating multiple pieces of information from the paragraph.
Narrow Lessons	3	The AI isn't good at comparing media release dates..

C.8 User Interface Screenshots

Answer reading comprehension questions with help of an AI

You are invited to participate in a research study how people interact with an AI to answer questions. You will be presented with a series of examples consisting of a passage and a question and you can either answer or use an automated agent to answer for you.

This study will include participants over 18 years of age, who feel comfortable using an online interface to read paragraphs and answer questions in English about them.

To reiterate, you are eligible for this study if and only if:

- You are comfortable reading/writing in English.
- You have not already completed this survey.
- You are over 18 years of age.
- You have JavaScript enabled in your browser.

What Will My Participation Involve?

If you decide to participate in this research you will be asked to answer a series of reading comprehension questions and asked to write short sentences describing their theme. We expect this study to take 20-25 minutes.

Are There Any Risks To Me?

We don't anticipate any risks from participation in this study greater than normal activity.

Are There Any Benefits To Me?

There are no direct benefits to you other than compensation.

How Will My Confidentiality Be Protected?

While there will probably be publications as a result of this study, your name will not be used. Only group characteristics will be published.

Whom Should I Contact If I Have Questions?

Figure C.16: Consent form to be confirmed before entering experiment

After you fill out the information on this page please click the Start Experiment button to proceed.
The experiment contains three stages:

1. **Welcome task:** to familiarize you with the interface and the task.
2. **Teaching phase:** We will help prepare you for the task by teaching you through different examples.
3. **Testing phase:** You will be tested on a series of examples where you will receive a bonus for each correct answer.

If you have previously completed this HIT, you are auto-ineligible to complete it again!

Your answers to the below questions have no impact on your eligibility to perform the task.

Your MTurk worker IDs will NOT be shared with anyone or used in the study or tied to the information below, they are only collected for matching with the recorded HIT.

By completing this task, you consent to having your responses (beyond this page) be made publicly available for research and educational purposes.

MTurk Worker ID*:

Please select your age:

- 18 - 25
- 26 - 40
- 41-60
- 61 and above
- Prefer not to answer

What is your gender?

- Male
- Female
- Non-binary
- Prefer not to answer

What is the highest degree or level of school you have completed?

- Some high school, no diploma, and below.
- High school graduate, diploma, or the equivalent.
- Some college credit, no degree.
- Bachelor's degree (or currently pursuing)
- Graduate degree (or pursuing).
- Prefer not to answer.

How would you rate your knowledge about Artificial Intelligence (AI) or Machine Learning?

- I don't know anything about AI.
- I have a read about AI but don't know how it works
- I know how AI works at a basic level.
- I know how AI works at an expert level and can implement various types of AI

How often do you read articles on Wikipedia?

- Almost never
- Once a month
- Handful of times a week
- Everyday

Figure C.17: Information collected about workers prior to experiment. MTurk worker ID was only saved for cross-checking and then deleted.

This example is to familiarize you with the task itself.

Instructions (read first):

Task: You are shown a passage and a question where the answer is found exactly in the passage or in the question itself. The goal is to find the smallest segment of text that answers the question.

Note: The answer is always a contiguous span of the passage or question, however, it may not be always possible to verify the answer as some sentences from the text are removed.

Example: In the example below we have a passage and question about Oak, we can immediately recognize that the answer is "500" which is in the second sentence highlighted in red.

The goal is to select the smallest segment of the paragraph that answers the question, so while "There are approximately 500 species of oaks." gives an answer to the question, it does not extract the precise piece of information.

Passage:

An oak is a tree or shrub in the genus *Quercus* of the beech family, Fagaceae. There are approximately 500 species of oaks.

Question:

How many species of oak are there?

Figure C.18: First step of the tutorial introducing the task

This example is to familiarize you with the task itself.

Instructions (read first):

Now you try solve an example on your own!

Task: You are shown a passage and a question where the answer is found exactly in the passage.

Instructions:

- With your cursor highlight the answer in the passage
- With the answer selected, press the *Select highlighted text as answer*
- Press the *Submit answer* button for your final answer

Passage:

Blue Paul Terrier is an extinct breed of dog.

[...] The "Russo-European Laika" itself dates to a breeding program begun in 1944 by E. I. Shereshevsky of the All-Union Research Institute for the Hunting Industry, in Kalinin (now Tver) Province.

Question:

Which is extinct, Blue Paul Terrier or Russo-European Laika?

Select highlighted text as answer

Highlighted answer appears here

10 Seconds before you can submit

Figure C.19: Second step of the tutorial solving without AI help

This example is to familiarize you with the task itself.

Instructions (read first):

New button: Instead of answering yourself, try having an Artificially Intelligent agent answer for you!

The Artificially Intelligent Agent (AI): An artificially intelligent agent can assist you with the task, they have strengths and weaknesses in different parts of the domain. They have access to the complete passage without any missing sentences.

Instructions:

- A new button *Let AI answer for you* is displayed, pressing this button allows an AI to solve the task for you
- Before pressing the button, you cannot see what the AI will answer
- Go ahead and press the button to try it!

Passage:

Blue Paul Terrier is an extinct breed of dog.

[...] The "Russo-European Laika" itself dates to a breeding program begun in 1944 by E. I. Shereshevsky of the All-Union Research Institute for the Hunting Industry, in Kalinin (now Tver) Province.

Question:

Which is extinct, Blue Paul Terrier or Russo-European Laika?

Select highlighted text as answer

Let AI answer for you

Blue Paul Terrier

Submit answer

Read about the button above and then press it!

Figure C.20: Third step of the tutorial solving with AI help

Teaching phase: this series of examples is to teach you about the AI

Instructions (read first):

Now that you are familiar with the task, you will solve a series of 9 examples that are specially designed to inform you about the AI's capabilities versus your own.

Goal: Help you understand when to defer to the AI agent and when not to.

Instructions, on each example:

- Either select your own answer or defer to the AI
 - After you answer, we will show you what the true answer is and what the AI predicted
 - To help you understand why did the AI get the example right or wrong, we will show you two similar examples to the one you just solved.
 - From these two supporting examples, we ask you to write a sentence that characterizes the example.
 - The sentences you write should help you in future examples to figure out when and when not to defer to the AI.
-

Read Instructions above

Figure C.21: Teaching instructions

Context:

Nothoscordum is a genus of New World plants in the onion tribe within the Amaryllis family. It is probably paraphyletic. [...].

Callirhoe is a genus of flowering plants in the mallow family, Malvaceae. Its nine species are commonly known as poppy mallows and all are native to the prairies and grasslands of North America. Of the nine, some are annuals while others are perennial plants.

Question:

Which genus is native to more continents, Nothoscordum or Callirhoe?

These are the lessons you wrote in the teaching stage

Lessons where AI is correct:

Lessons where AI incorrect:

Select highlighted text as answer

Let AI answer for you

Highlighted answer appears here

Submit highlighted answer

Figure C.22: Teaching initial example to be solved by the human.

You used the AI and they are incorrect!

True Answer is: **Nothoscordum**
 AI Answer is: **Callirhoe**



The AI answered this incorrectly, look at these similar examples:

The words highlighted in green are those that most led the AI to get the answer correctly and those in red led to it to answer incorrectly.

AI is incorrect
 On the original example.

Context:

nothoscordum is a **genus** of new world **plants** in the onion tribe within the amaryllis family. it is probably paraphyletic. [...]. callirhoe is a **genus** of **flowering plants** in the mallow family, **malvaceae**. its nine **species** are commonly known as **poppy** mallows and all are native to the prairies and grasslands of north america. of the nine, some are annuals while others are perennial **plants**.

Question:

Which genus is native to more continents, Nothoscordum or Callirhoe?

AI is incorrect
 For all examples as similar as this to the original.

Context:

citric acid is a weak organic tricarboxylic acid having the chemical formula cho. it occurs naturally in **citrus fruits**. in biochemistry, it is an intermediate in the citric acid cycle, which occurs in the metabolism of all aerobic **organisms**. a **clementine** ("**citrus** × clementina") is a hybrid between a mandarin orange and a sweet orange, so named in 1902. the exterior is a deep orange colour with a smooth, glossy appearance. **clementines** can be separated into 7 to 14 segments. similar to tangerines, they tend to be easy to **peel**. the clementine is also occasionally referred to as the "moroccan clementine". [...]. their oils, like other **citrus fruits**, contain mostly limonene as well as myrcene, linalool, α-pinene and many complex aromatics.

Question:

What is the chemical formula of the organic material that clementines have less of than oranges?

AI is correct
 For examples not as similar.

Context:

the **ogallala aquifer** is a shallow water table **aquifer** surrounded by sand, silt, clay and gravel located beneath the great **plains** in the united states. one of the world's largest aquifers, it underlies an area of approximately 174000 sqmi in portions of eight states (south dakota, **nebraska**, wyoming, colorado, kansas, oklahoma, new mexico, and texas). it was named in 1898 by geologist n. h. darton from its type locality near the town of **ogallala, nebraska**. the **aquifer** is part of the **high plains aquifer** system, and rests on the **ogallala** formation, which is the principal geologic unit underlying 80% of the **high plains**. [...]. the population was 4,737 at the 2010 census. it is the **county** seat of keith **county**. in the days of the **nebraska** territory, the city was a stop on the pony express and later along the transcontinental railroad. the **ogallala aquifer** was named after the city.

Question:

What shallow water table aquifer is located near the county seat of Keith County, Nebraska?

Figure C.23: Feedback shown after human solves the example along with supporting examples.

The following words are most representative of this example and it's surrounding:
subspecies, fabaceae, genus of, shrubs, plant, plants, species, flowering, genus,

Write a sentence to describe the topic of the example you solved, be inspired by the set of words above and the first two examples above should you help broaden your theme while the third should help restrict it.
e.g. "The AI is good at answering questions about politics"

Show next teaching example

Figure C.24: Top words for the teaching example along with instructions for lesson writing

Teaching phase part 1: this series of examples is to teach you about the AI

Instructions (read first):

Now that you are familiar with the task, you will solve a series of 9 examples that are specially designed to inform you about the AI's capabilities versus your own and then will view more examples about the AI.

Goal: Help you understand when to defer to the AI agent and when not to.

Instructions, on each example:

- Either select your own answer or defer to the AI
- After you answer, we will show you what the true answer is and what the AI predicted
- We then ask you to write a sentence that characterizes the example.
- The sentences you write should help you in future examples to figure out when and when not to defer to the AI.

Continue

Figure C.25: The LIME-Teaching user teaching introduction

The AI answered this incorrectly, look at the words the AI focused on:

The words highlighted in green are those that most led the AI to get the answer correctly and those in red led to it to answer incorrectly.

AI is incorrect
On the original example.

Context:

ansel elgort (born march 14, 1994) is an american actor, singer and a dj (under the name ansolo). [...]. </br> baby driver is a 2017 action crime comedy film written and directed by edgar wright. it stars ansel elgort, kevin spacey, lily james, eiza gonzález, jon hamm, jamie foxx, and jon bernthal. the plot follows baby, a young getaway driver and music lover who must work for a kingpin. the film is best known for its choreography, in which the actors' timing and movements are synced with the soundtrack.

Question:

What is the profession of Ansel Elgort's character in "Baby Driver?"

Write a sentence to describe the topic of the example you solved, this sentence should help you to predict if the AI is correct in future examples.
e.g. "The AI is good at answering questions about politics"

10 seconds before you can see next teaching example

Figure C.26: The LIME-Teaching feedback after answering teaching question.

Teaching phase part two: read examples and see the AIs answers

INSTRUCTIONS CHANGED (read first):

Teaching phase part 2: You will now observe 9 more examples without needing to solve them, look at each example for a bit and proceed to the next.

Continue

Figure C.27: The LIME-Teaching teaching introduction to second part of the teaching phase

Context:


the united methodist church (umc) is a mainline protestant denomination, and a major part of methodism. in the 19th century, its main predecessor—the methodist church—was a leader in evangelicalism. it was founded in 1968 in dallas, texas, united states, by union of the methodist church and the evangelical united brethren church. the umc traces its roots back to the revival movement of john and charles wesley in england as well as the great awakening in the united states. [...] it embraces both liturgical and evangelical elements. it has a connectional polity, a typical feature of a number of methodist denominations.
 john wesley (or ; 28 june [o.s. 17 june] 1703 2 march 1791) was an english anglican cleric and theologian who, with his brother charles and fellow cleric george whitefield, founded methodism.

Question:

Who founded Methodism along with the man who was also part of the revival movement in England and the Great Awakening in the United States?

The AI are incorrect!

True Answer is: George Whitefield
AI Answer is: John Wesley



These are the lessons you wrote in the teaching stage

Lessons where AI is correct:

Lessons where AI incorrect:

Figure C.28: The LIME-Teaching user interface of the second part of the teaching phase where users observe examples and the AI answers.

Testing phase: answer with no feedback to get as many correct!

INSTRUCTIONS CHANGED (read first):

Context:

Hot Lead and Cold Feet (originally titled Welcome to Bloodshy) is a 1978 American comedy-western film produced by Walt Disney Productions and starring Jim Dale, Karen Valentine, Don Knotts, Jack Elam and Darren McGavin.

[...] . The film was directed by Bill Paxton, and was his last film as a director. Shia LaBeouf plays the role of Ouimet. The film's screenplay was adapted by Mark Frost from his book, "The Greatest Game Ever Played: Harry Vardon, Francis Ouimet, and the Birth of Modern Golf". It was shot in Montreal, Canada, with the Kanawaki Golf Club, in Kahnawake, Quebec, the site of the golf sequences.

Question:

Which was released first, The Greatest Game Ever Played or Hot Lead and Cold Feet?

These are the lessons you wrote in the teaching stage

Lessons where AI is correct:

- politics
- games including board and video games
- soccer but not American football
- books, writers, but not music records
- The following words are most representative of this example and it's surrounding:
music, musician, director, albums, rock, band, songwriter, actor, he, singer,

Lessons where AI incorrect:

- plants, but not geology
- movies, films
- composers, opera, but not poetry
- university, school, academics

Select highlighted text as answer

10 Seconds before you can use AI answer

Highlighted answer appears here

10 Seconds before you can submit

Figure C.29: Interface during testing.

You're almost done! Please answer these final questions to finish the task and press the Continue button at the end of page when you're done

For questions where there was a similar example in the teaching phase, what was your decision process for using (or not using) the AI?

For questions where there was **not** a similar example in the teaching phase, what was your decision process for using (or not using) the AI?

Continue

Figure C.30: Questions collected after workers complete experiment for the Teaching condition.

Appendix D

Additional Information for Chapter 5

D.1 Extended Related Work

Reference [132] proposes a method for human-AI collaboration via conditional delegation rules that the human can write down. Our framework enables the automated learning of such conditional delegation rules for more general forms of data that can also depend on the AI output. [15] proposes to modify the confidence displayed by the AI model to appropriately encourage and discourage reliance on the AI model. However, this technique deliberately misleads the human on the AI model ability, our methodology incorporates similar ideas by learning the human prior function of reliance on the AI and then improving on it with the learned integration recommendations, however, we display these recommendations in a separate dashboard without modifying the AI model output. A related approach to our methodology by [133] is to adaptively display or hide the AI model prediction and display the estimated confidence level of the human and the AI on a task of predicting whether a person's income exceeds a certain level. They show that displaying the confidence of the human and the AI to the human improves performance. Our method is able to learn the confidence level of the human and the AI, but also incorporates how the human utilizes the AI and describes the regions where AI vs human performance is different. [134] presents a similar approach to our AI recommendations, however, they use simulated and faked AI models and descriptions of behavior while we are able to obtain automated generation of these descriptions of AI behavior.

Existing research has examined various methods to establish human trust in and replicate the predictions of machine learning models. One such method is LIME, a black-box feature importance technique, which was employed to select examples for evaluation by crowdworkers to determine the superior model among two options [38], [39]. However, their selection strategy disregards the human predictor, and their approach merely presents the examples without further action. In the context of visual question answering, Chandrasekaran et al. [101] manually selected seven examples to educate crowdworkers about the AI's capabilities, leading to an enhanced ability to identify instances where

the AI failed. Feng et al. [102], in the domain of Quizbowl question answering, emphasize the significance of incorporating the human expert's skill level when designing explanations. This further justifies our decision to involve the human predictor in the selection of teaching examples. Cai et al. [103] conducted a study involving 21 pathologists to gather guidelines on what clinicians desired to know about an AI system prior to interacting with it. Yin et al. [104] investigated the impact of initial debriefing on stated AI accuracy versus observed AI accuracy during deployment, finding a substantial influence of stated accuracy on trust that diminishes quickly once the model is observed in practical use. This reinforces our approach of building trust through examples that simulate real-world deployment. Bansal et al. [16] examined the role of the human's mental model of the AI in task accuracy; however, the mental model was developed through interaction during testing rather than during an initial onboarding stage. The most similar work to ours is that of [82] which presents an onboarding scheme based on selecting a set of examples and allows the human to describe the regions where the AI performance is good or bad. Through a user study on passage-based question answering, they show that their onboarding scheme improves performance by 5%, however, they evaluate without the presence of AI evaluations, with a synthetic AI model and their scheme involves more involvement from the human as they have to describe the regions themselves. Another approach to teaching involves providing humans with guidelines on when to rely on AI systems [100]. Model cards [154] and industry practices such as the IBM AI fact sheet [252] demonstrate direct methods of presenting these guidelines to users, we present humans with a similar form of card but that includes aspects of human performance "human-AI card".

There is a growing and large area of literature on discovering (and auditing) regions of AI error, the following is not meant as an extensive list of related work but captures some of the essence of the literature:

- Adatest allows a user to iteratively discover regions of AI error using LLMs for NLP tasks and then re-train the model on the regions of error [138], it was then extended to vision tasks [253] with a similar procedure in [141].
- Erudite allows users to discover regions of error of NLP models through user interfaces [139], there is a wider literature on dashboards for discovering regions of error [254]
- [144] learns an SVM model from image embeddings to predict model error and then uncover regions of error based on the directions of the SVM model.
- Works have done extensive manual annotation of ImageNet model mistakes[255], [256].
- DOMINO discovers regions of model error using a slice discovery model based on a specialized gaussian mixture model [136], extensions include DRML [140].

- The Spotlight method learns individual regions based on a neighborhood of a learned point [142], our region finding algorithms generalize their procedure by learning weighted distance distance measures and with a different aim of improving gain of the prior.
- SEAL: Interactive Tool for Systematic Error Analysis and Labeling, uses k-means to uncover regions of error and then uses an LLM to describe each region [143].

Recent work has emerged on describing sets of images [257], [258] but they don't incorporate a contrastive method as we propose. Helpful tools for describing differences between text and images can be useful for describing regions which future work can incorporate [259]–[261].

One of the objectives of explainable machine learning is to enhance humans' ability to assess the accuracy of AI predictions by offering supporting evidence [88]–[97]. Nevertheless, these explanations fail to provide guidance to decision makers on how to balance their own predictions against those of the AI or how to integrate the AI's evidence into their final decision [98]. [17] shows that AI explanations can reduce overreliance and improve human-AI team performance, however, their experiments are with simulated AI models and explanations. The central question of our work is not to study the utility of AI explanations, in fact, all our user studies incorporate AI explanations and we aim to improve human-AI performance in their presence.

D.2 Region Finding Algorithm - Details

Regions Requirements. Each region in our algorithm should aim to satisfy the following constraints:

1. **Region Size:** We want the size of the region to be at least of size β_l and at most of size β_u .
2. **Consistency of takeaway:** The examples in each region must agree on what the takeaway is in terms of the integration decision. Specifically, at least $\alpha\%$ of all points in the region must either be: ignore AI, use AI as is or integrate AI advice.
3. **Concise and Distinguishable Theme:** Each region must be concisely described in natural language in such a way to differentiate from the overall domain. If a region cannot be described in natural language, the human may not be able to derive a generalizable recommendation from it. This is a constraint that we implicitly try to satisfy by learning neighborhoods in a natural language embedding space.
4. **Minimum Gain:** Each region must have a minimum information gain (defined below) of δ . This is to ensure that all regions contain sufficient novel information to the human.

The optimization to find each region can be formulated in its non-relaxed form as:

$$\begin{aligned}
& \max_{c, \gamma, w, r} \sum_{i=1}^n \mathbb{I}_{\|w((e_i, a_i) - c)\| < \gamma} \cdot \mathbf{g}_{i,r} \\
& \text{s.t.} \quad \sum_{i=1}^n \mathbb{I}_{\|w((e_i, a_i) - c)\| < \gamma} \cdot \mathbb{I}_{r_i=r} \geq \alpha n \\
& \text{s.t.} \quad n\beta_l \leq \sum_{i=1}^n \mathbb{I}_{\|w((e_i, a_i) - c)\| < \gamma} \leq n\beta_u
\end{aligned}$$

And the relaxation we propose is (refer back to Section 5.4.1):

$$\begin{aligned}
& \max_{c, \gamma, w, r} \sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) \cdot g_{i,r} - \lambda \max(\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) \\
& \cdot (\mathbb{I}_{r_i^*=r} - \alpha n), 0) - \lambda \max\left(\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) - \beta_u n, 0\right) \\
& \lambda \max\left(-\sum_{i=1}^n \sigma(C_1(-\|w \circ ((e_i, a_i) - c)\| + \gamma)) + \beta_l n, 0\right)
\end{aligned}$$

We run the optimization for $r = 0$ and $r = 1$ and choose the r with the better objective value. With r fixed, we optimize with respect to the remaining continuous parameters using AdamW and reduce the learning rate when loss has stopped improving. We initialize with $\gamma = 0$ and $w = \mathbf{1}$. For the centroid c , we first run k -medoids clustering on the input data with $k = \min(\max(100, T), n)$ and randomly select 20 of the resulting centroids. Then with c initialized as each one of the 20 centroids in turn, we run the optimization for 200 epochs and record the loss, and finally optimize for 2000 epochs with the best initialization for c . We do these repeated initializations to avoid local minima which are a common failure mode of this type of optimization problem. Note this process is to find one region, to find all regions we repeat this process identically to find regions one by one.

Selection Based Approach. We described in the body of the paper a generative algorithm to find the regions. We now describe a selection based algorithm that finds centroids from points in the dataset $\tilde{D} = \{e_i, r_i\}$. We will also restrict the radius to be the distance between the centroid to another data point in \tilde{D} . We proceed with a sequential search, at round i we perform the following

search:

$$c_i, \gamma_i = \arg \max_{i \in \tilde{D}, \gamma} G(N_{i,\gamma}, \hat{R}^*, R_{N_{1:i-1}}), \quad (\text{D.1})$$

$$\text{s.t. } \exists k \in [n] \text{ s.t. } \gamma = d(e_i, e_k), \quad (\text{D.2})$$

$$\text{and } \frac{\sum_{j \in [n], d(e_i, e_j) < \gamma} \mathbb{I}_{r_j = r_i}}{|\{j \in [n], d(e_i, e_j) < \gamma\}|} \geq \alpha \quad (\text{D.3})$$

$$\text{and } \beta_l \leq |\{j \in [n], d(e_i, e_j) < \gamma\}| \leq \beta_u \quad (\text{D.4})$$

$$\text{and } G(N_{i,\gamma}, \hat{R}^*, R_{N_{1:i-1}}) > \delta \quad (\text{D.5})$$

Note that with the selection-based procedure, we have to define a fixed distance measure d , and we cannot optimize over the deferral decision r of the region as we inherit from the point i found. The naive algorithm to solve the above search is as follows. We first compute the distance matrix between all data points in \tilde{D} , call this matrix K . We then at each round do the following for each point i in \tilde{D} : sort the points by their distance to i , iteratively grow the region around the point i to satisfy the constraints, and then keep track of the maximum gain radius. Each time we grow the region by one point, we check if the constraints are satisfied. The search is parallelized across multiple instances to make it faster. Finally, we compare the gain of all feasible points and pick the highest one. This algorithm extends the approach of [82] to incorporate additional constraints and has significant speed ups over their approach.

Aggregating Regions Across Different Embedding Spaces . We can run our region finding algorithm above and find different regions across multiple embedding spaces. The question is how do we aggregate these regions. Say we found a set of T regions N_1, \dots, N_T . We then run the following meta-algorithm: do a greedy sequential search to add regions one at a time while making sure the minimum gain requirement is satisfied, stop when it is not.

D.3 Region Description Algorithm - Details

The LLM call inside Algorithm 1 is accomplished by building the prompt with S^+ being the inside set (positives) and S^- being the outside set (negatives) as follows:

```
def get_prompt(positives, negatives):
    prompt = pre_instruction + "\n"
    prompt += "inside the region: \n "
    counter = 1
    for p in positives:
        prompt += p[0] + ", \n "
```

```

    counter += 1
if len(negatives) > 0 :
    prompt += ". \n not in the region: \n"
    counter = 1
    for p in negatives:
        prompt += p[0] + ",\n"
        counter += 1
prompt += post_instruction
return prompt

```

For the experiments in Section 5.6 we use the following pre instruction:

I will provide you with a set of descriptions of points that belong to a region and a set of descriptions of point that do not belong to the region. Your task is to summarize the points inside the region in a concise and precise short sentence while making sure the summary contrasts to points outside the region. Your one sentence summary should be able to allow a person to distinguish between points inside and outside the region while describing the region well. The summary should not be a single word, it should be accurate, concise, distinguishing, and precise.

Example:

Inside the region:

- two cows and two sheep grazing in a pasture.
- the sheep is standing near a tree.

Not in the region:

- the cows are lying on the grass beside the water.

summary: sheep.

End of Example

While the post-instruction is simply "summary:".

For the ablation without contrasting, the pre instruction we use is:

I will provide you with a set of descriptions of points that belong to a region. Your task is to summarize the points inside the region in a concise and precise short sentence .Your one sentence summary should be able to allow a person to distinguish points inside the region while describing the region well.The summary should be a single word, it should be accurate, concise, distinguishing and precise.

Example:

inside the region: - two cows and two sheep grazing in a pasture.

-the sheep is standing near a tree.

summary: sheep.

End of Example

For the user studies, we had used an earlier instruction that works slightly worse and post-processed the descriptions by only modifying the first few words to be consistent (see the real examples in later sections):

"summarize the points inside the region in a concise and precise short sentence while making sure the summary contrasts to points outside the region"

We recommend the use of the following pre instruction:

I will provide you with a set of descriptions of points that belong to a region and a set of descriptions of points that do not belong to the region. Your task is to summarize the points inside the region in a concise and precise short sentence while making sure the summary contrasts to points outside the region. Your one sentence summary should be able to allow a person to distinguish between points inside and outside the region while describing the region well. The summary should be no more than 20 words, it should be accurate, concise, distinguishing and precise.

Example:

inside the region:

- two cows and two sheep grazing in a pasture.

- the sheep is standing near a tree.

outside the region:

- the cows are lying on the grass beside the water.

summary: The region consists of descriptions that have sheep in them outside in nature, it could have cows but must have sheep.

End of Example

D.4 Onboarding and Recommendations to Promote Rules - Details

Introductory Phase. To facilitate a smooth onboarding process for individuals working with an AI assistant, we introduce the Human-AI Card. This card provides detailed insights into the AI's

capabilities, training, and performance.

Additionally, we provide a breakdown of AI and Human performance on different subgroups of data. We take an example of the Berkeley Deep Driving dataset, where a subgroup might comprise images taken during the night, in rainy weather, or on a highway. We compute the model’s error for each possible subgroup and then perform a paired t-test comparing the subgroup model error to the average model error over the entire data. For the purpose of our user studies, we highlight subgroups defined by a single metadata category that show statistically significant differences ($p \leq 0.05$). It’s important to note that, for rigorous analysis, one should apply corrections for multiple hypothesis testing. However, considering the vast number of metadata categories, many results might become insignificant. Therefore, for simplicity, we adopt this heuristic approach.

D.5 Method Evaluation - Details

In Table D.1 we report the details on the datasets we use in our method evaluation. We normalize all datasets using l_∞ normalization and run our algorithms for 2000 epochs with a learning rate of 0.001 using AdamW and a scheduler to update. We use a constant $C_1 = 20$.

Table D.1: Datasets for "Learning Accurate Integrators (Aim 1)". We note the total number of samples n , the target set size $|\mathcal{Y}|$, the human expert finally the model of the AI. When we note human "XX% accurate", this indicates a synthetic human model that is accurate uniformly at random with probability XX%. For DynaSent, the AI model is a a pre-trained sentiment analysis roBERTa-base model [160] on Twitter data and achieves 75% accuracy. For both BDD and MS-COCO we blur the images using a Guassian Blur with scale 21 and variance 5.

Dataset	n	$ \mathcal{Y} $	Human	AI
Berkeley Deep Drive (BDD) [128], [262]	10k	2	80% accurate	faster rcnn r50 fpn 1x ¹ - Gaussian blur with scale 21 and variance 5
MS-COCO [156]	5k	2 (presence of person in image)	70% accurate	faster rcnn R 50 FPN
Massive Multi-task Language Understanding (MMLU) [129]	14k	4 (MCQ)	50% accurate	flan-t5xl [159]
Dynamic Sentiment Analysis Dataset (DynaSent) [157]	6.5k	3	leave-one-out annotator	a pre-trained sentiment analysis roBERTa-base model [160]

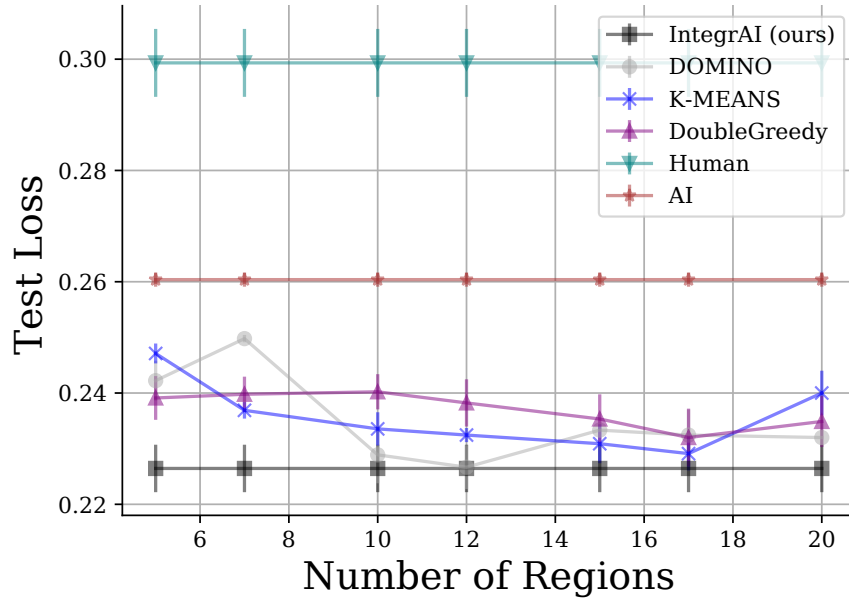


Figure D.1: Test Error (↓) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the MS-COCO dataset.

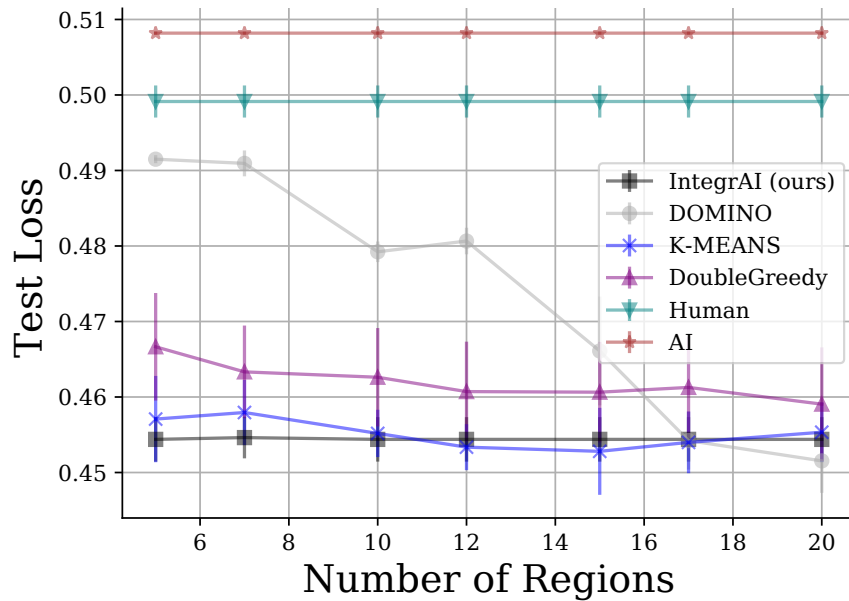


Figure D.2: Test Error (↓) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the MMLU dataset.

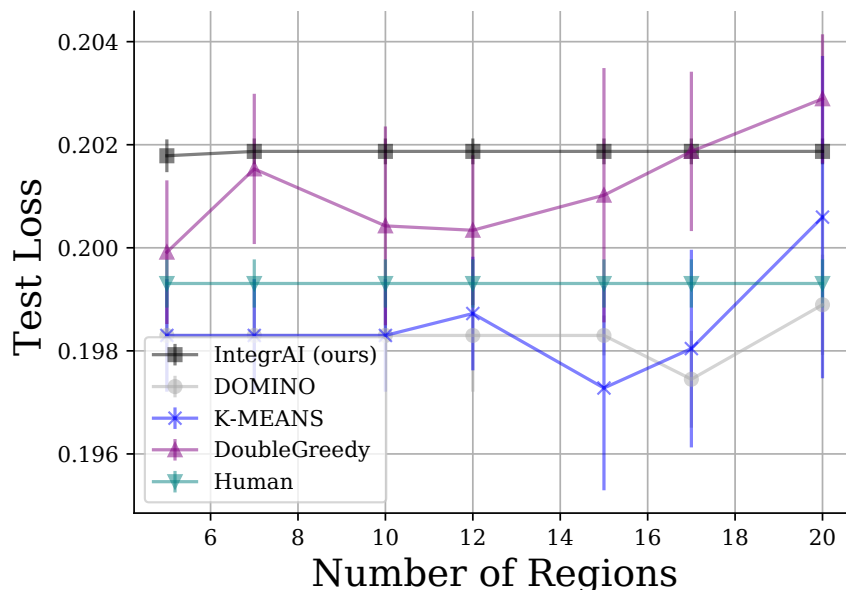


Figure D.3: Test Error (\downarrow) of the human-AI system when following the decisions of the different integrators baselines as we vary the number of regions maximally allowed for each integrator on the Dyanasent dataset.

For Aim 2, we create synthetic AI and human models as follows:

- BDD: AI and Human model each have four regions defined as a condition of a randomly selected metadata feature, each region is either a good region where the AI/Human have 95% accuracy or bad region with 60% accuracy (equal number of good and bad regions). If an example does not belong to any region the AI/Human have 75% accuracy. Each region is of size at least 0.01 and at most 0.2 in terms of fraction of data points in the dataset. An example region is "AI is good at: weather: clear" or "Human is bad at: timeofday: night".
- MMLU: same setup as BDD, region defined as the subject of example. An example region is "AI is good at: subject: professional psychology".
- MS-COCO: same setup as BDD, region defined as presence of object in image. An example region is "'Human is bad at: cow: present".

For Aim 3, an obstacle to quantitative results for region descriptions is that they are often complex even when regions are synthetically defined in terms of metadata and captioning metrics are not informative. For a region defined on BDD as images of "scenes of highways during the night with no cars", our algorithm finds the description "highway during the night with various weather conditions and not congested with many cars" while the SEAL [143] describes it as "Highway Nighttime Weather Conditions". The SEAL description surprisingly has higher captioning metric scores (BLEU, METEOR) [263] while being less informative.

For the MS-COCO evaluation, we only select objects that have at least 50 examples present in the evaluation set which leads to only 73 objects over which we evaluate the different region description algorithms.

D.6 User Studies - Details

D.6.1 BDD Study

Task. The images from BDD are blurred using a Gaussian Blur with a scale of 21 and a variance of 5. The AI model is a trained faster rcnn model that achieves 84% accuracy without blurring which decreases to 78% accuracy on the test set. We use the bounding boxes from the model and output them on the image (allowing the user to either hide or show them). To get a confidence score, we take the maximum score for the prediction of a traffic light in the image.

Initial Data Collection. The BDD dataset was split 70-30 where the 70% split was used to get the initial human predictions and find the regions and the 30% split was used only to get testing examples for the final user study. As mentioned, we obtained data on 400 examples with both human predictions and prior AI-integration decisions. We use these examples to build Random Forrest models that predict both the human predictions and AI-integration decisions from the embeddings, labels, and AI predictions (AI predictions only for predicting AI integration decisions). We use these predictions from the RF models to label the entire dataset. We ensure that the predictions of the RF models are calibrated (if the human is 80% accurate, the model is also 80% accurate) with the human predictions and integration decisions by modifying the threshold on model probability used to make predictions (e.g. from the usual 0.5 threshold to the value that makes the models calibrated). Each participant is evaluated on a randomized set of examples, we create 40 different sets of 20 images that get assigned randomly to each participant.

Attention Checks. In each condition, we insert 3 attention checks for every 20 examples where the images are unblurred and we keep the AI prediction. We only retain responses where participants don't get all attention checks wrong. The class balance of the dataset is close to 51%-49%. The attention checks are used as part of the study results as they only modify the blur of the images.

Regions Found. The regions found by our procedure are shown in Table [D.2](#).

Table D.2: Region descriptions found by IntegrAI for the BDD user study.

Region ID	Description
1	depict various city streets and highways during the daytime with different weather conditions, containing pedestrians, cars, trucks, traffic lights, and signs.,
2	depict various types of roads and streets during the daytime with different weather conditions, containing cars, pedestrians, traffic lights and signs, while the outside examples include scenes with fewer objects or in less common locations such as a parking lot.,
3	depict various scenarios of streets and highways with moderate to heavy traffic flow during the day or night, with different weather conditions, along with traffic signs and lights, cars, trucks, buses, and pedestrians.
4	depict various outdoor scenes during the daytime or nighttime, containing multiple cars, traffic lights, and traffic signs
5	depict various city and residential scenes during the daytime with different weather conditions and contain a variety of vehicles, pedestrians, traffic lights, and signs, while the outside examples depict specific limited scenarios with fewer elements.,
6	depict various traffic scenes, mostly highways at night and during clear weather, with a mix of cars, trucks, buses, traffic lights, and traffic signs
7	depict various city streets and highways with clear weather and a moderate amount of traffic, including cars, signs, and occasionally pedestrians, bicycles, and trucks
8	depict various urban and residential scenes during different times of day and weather conditions, with a diverse range of vehicles, pedestrians, traffic signs, and traffic lights present
9	depict various scenes of city streets and highways with typical traffic conditions and without severe weather conditions
10	depict various scenes of city streets and highways with varying weather conditions, traffic, and signage.

D.6.2 MMLU Study

Task. We rely on the MMLU dataset [129] where participants are shown a question, four possible answers and have to pick the best answer (see screenshots in Figure below for examples). We use

ChatGPT, also known as GPT 3.5 turbo as our AI model [6]. We obtain the predictions of ChatGPT on the MMLU dataset following the approach in the official repo of MMLU². We also ask ChatGPT to explain its answer by using the prompt "Please explain your answer in one sentence". Both the AI answer and the explanation are shown.

ChatGPT obtains an accuracy of 69% during our evaluation and we restrict our attention to specific subjects within the MMLU dataset. Specifically, we sample 5 subjects (out of the 57 in MMLU) where ChatGPT has significantly better performance than average, 5 where it's significantly worse, and 4 subjects where performance is similar to average performance. These subjects are listed here:

high school government and politics, marketing, high school psychology, logical fallacies, sociology, public relations, high school computer science, anatomy, business ethics, elementary mathematics, high school statistics, machine learning, moral scenarios, global facts

We sample 150 questions from each subject and additionally sample 150 questions from OpenBookQA dataset [163] to use as attention checks as human performance on OpenBookQA is 91%.

We run IntegrAI on both the dataset embeddings, metadata (subject name), and an embedding of ChatGPT explanations separately. We find 10 regions based on the metadata and 2 regions based on the ChatGPT explanations. The regions and their descriptions found by our algorithm are reported in Table D.3.

²<https://github.com/hendrycks/test>

Table D.3: Region descriptions found by IntegrAI for the MMLU user study.

Region ID	Description
1	Related to marketing, including pricing strategies, branding, communication, product classification, market segmentation, advertising, and supply chain management.
2	Questions related to psychology and neuroscience.
3	Questions related to global statistics and trends, ranging from military spending to mental health disorders.
4	Questions inside the region cover a variety of topics in the field of public relations, including ethical frameworks, evaluation models, common tactics, and regulations.
5	Mathematical and quantitative questions, involving calculations and problem-solving.
6	Questions related to sociology, including topics such as social class, symbolic interactionism, bureaucracy, and globalization.
7	Questions and descriptions of logical fallacies and syllogisms.
8	Questions focus on US politics, government, and history.
9	Contains questions related to anatomy and physiology.
10	Various topics including ethics, regulation, consumer rights, and corporate transparency.
11	Various questions on different topics such as algebra, biology, public relations, and statistics with multiple options to choose from.

D.6.3 Screenshots of User Study Interface for BDD

Classify Images of Road Scenarios

You are invited to participate in a research study how people classify images. You will be presented with a series of images and have to make judgements. You may additionally be asked to interact with an AI model and write a few sentences of text.

This study will include participants over 18 years of age, who feel comfortable using an online interface to read and answer questions in English and investigate images.

To reiterate, you are eligible for this study if and only if:

- Are able to distinguish images of different types.
- You are comfortable reading/writing in English.
- You have not already completed this survey.
- You are over 18 years of age.
- You have JavaScript enabled in your browser.

What Will My Participation Involve?

If you decide to participate in this research you will be asked to classify different images, write answers to questions in English and interact with AI models.

Are There Any Risks To Me?

We don't anticipate any risks from participation in this study greater than normal activity.

Are There Any Benefits To Me?

There are no direct benefits to you other than monetary compensation.

How Will My Confidentiality Be Protected?

While there will probably be publications as a result of this study, your name will not be used. Only group characteristics will be published.

Figure D.4: Consent Form

Welcome to our experiment! (please read carefully)

After you fill out the information on this page please click the Start Experiment button to proceed.

The experiment may contain multiple stages that you will be guided through.

Please do not refresh or press back the webpage as you may lose a fraction of your progress.

The experiment will contain attention checks that will not modify the question you are asked, but they will be easier questions of the same format.

There is a potential bonus of up to \$5 for good performance on the tasks.

If you have previously completed this study, you are auto-ineligible to complete it again!

Your answers to the below questions have no impact on your eligibility to perform the task.

Your Prolific IDs will NOT be shared with anyone or used in the study or tied to the information below, they are only collected for matching with the recorded study.

By completing this task, you consent to having your responses (excluding age, gender, eyesight and educational ability) be made publicly available for research and educational purposes.

Prolific ID:

test111

What is your age? Select most appropriate age range.

35-50

What is your gender?

Prefer not to answer

What is the highest degree or level of school you have completed?

Some college credit, no degree

How would you rate your knowledge about Artificial Intelligence (AI) or Machine Learning?

I know how AI works at a basic level

How would you rate your eyesight? Poor indicates any major visual impairments, Excellent denotes 20/20 vision.

Fair

How would you rate your driving ability? Answer this question in terms of safety of driving.

I very rarely drive or I'm not confident in my ability.

Start Experiment

Figure D.5: User Information Collection

Instructions

- This experiment will be in 3 stages.
- Task: You will be shown images of road scenarios from the point of view of inside a car, you will be asked to respond to check whether in the image there is any traffic light (red or green or yellow)
- Your goal in is to be as accurate as you can be.
- Interface: you will have to first select an answer by choosing the corresponding button "Yes" or "No" and then pressing the "Submit" button. You can track how many examples remain by looking at the progress bar on the top of the webpage after you press next. You will keep trying each example until you get the answer correct

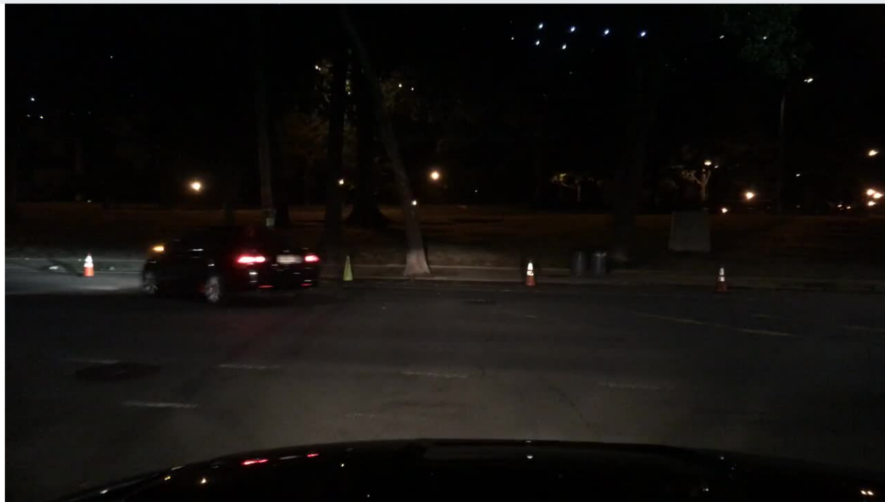
Example (no need to click, only for viewing):



Is there a traffic light in the picture?

Figure D.6: Practice Task instructions

Practice Examples



Is there a traffic light in the picture?

Yes

No

Submit

Figure D.7: Prediction without AI interface

Instructions (scroll down for task)

- **AI MODEL:** Images are now blurred and you have access to an Artificial Intelligence (AI) model that provides their prediction
- **AI Button:** A new button now appears called "Use AI Answer" in blue and allows you to select the AI's answer which you can then submit.
 - Please press the button if the AI helped with your answer and you want to use it's answer, if you agree with the answer of the AI but got the answer without it's help then don't press the button.
 - If the AI predicts there to be a traffic light, it will put a box around it in the image and a score (you can hide or show it with a button below the image)
- **Teaching Phase:** The next of examples will help teach you more about the AI through examples where you will receive feedback, after that, there will be a testing phase where the 2\$ bonus applies.

Figure D.8: Instructions for the onboarding phase


- We give you the following information about the AI model to help you understand when you should use it's answer and when not to:

Average AI Accuracy:	78%
Average Human Accuracy:	72%
AI Model Input:	Blurry Image
AI Model Output:	Prediction of traffic light, bounding box on image showing it's location and a score indicating it's confidence
Source of Training Data:	Dataset of road images from New York and San Francisco and Bay Area.
AI Training Objective:	Detect Traffic Lights and other objects in image

- Now a deeper dive into the AI error/accuracy:

Category:	Accuracy
more than 5 traffic lights in image	90%
If there is 1 to 4 traffic lights	62%
If there is no traffic lights	86%
No Cars	83%
Daytime or overcast weather	75%
few pedestrians	76%

Figure D.9: Model card information shown during onboarding.



Hide AI answer Show Instructions

Is there a traffic light in the picture?

Yes

No

AI Predicts: NO with score: 0.0

Use AI Answer

Submit

Figure D.10: Prediction with AI interface (AI predicts no traffic light)

AI Predicts: NO with score: 0.0

Wrong: You predicted incorrectly., the correct answer was yes.
AI was Incorrect: The AI predicted incorrectly.



This example is part of a larger region with the following recommendation:

Recommendation: Ignore AI Answer
Region: depict various types of roads and streets during the daytime with different weather conditions, containing cars, pedestrians, traffic lights and signs, while the negative examples include scenes with fewer objects or in less common locations such as a parking lot.

Region Information:
AI Accuracy: 68.0
Human Accuracy: 91.0

Here are examples

Figure D.11: Feedback shown during onboarding phase after human predicts.

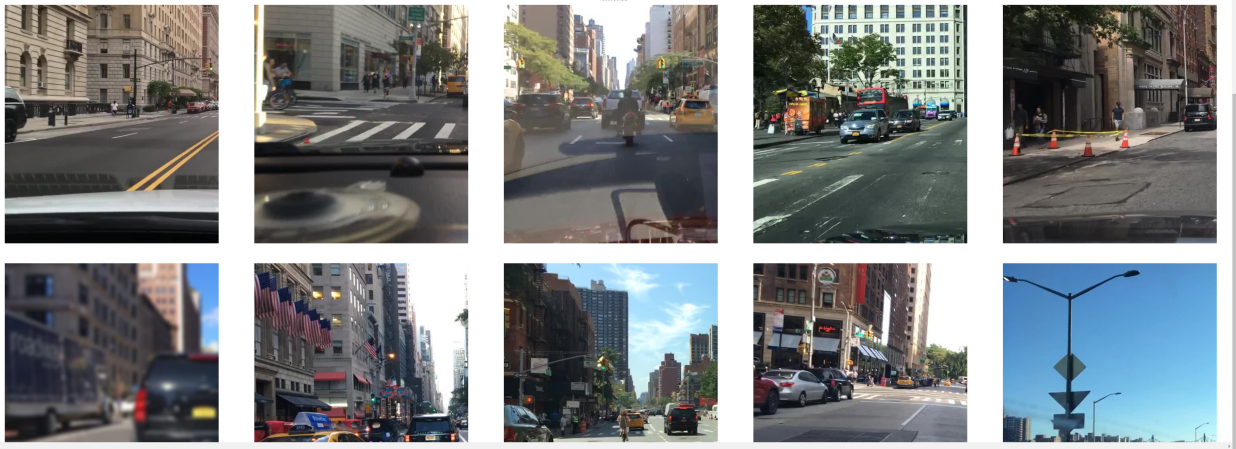


Figure D.12: Feedback shown during onboarding phase after human predicts (sets of examples from region)

Hide AI answer Show Instructions

Is there a traffic light in the picture?

Yes AI Predicts: YES with score: 0.92

No Use AI Answer

Submit


Figure D.13: Prediction with AI interface (AI predicts there is a traffic light)

Hide AI answer Show Instructions

AI Predicts: YES with score: 0.65

Correct: You predicted correctly!, the correct answer was yes.

AI was Correct: The AI predicted correctly.



This example is part of a larger region with the following recommendation:

Recommendation: Rely on AI Answer

Region: depict various scenes of city streets and highways with varying weather conditions, traffic, and signage.

Region Information:

AI Accuracy: 91.0

Human Accuracy: 78.0

Figure D.14: Feedback shown during onboarding phase after human predicts (correct feedback)

Instructions (scroll down for task)

- **Testing Phase:** You will be now tested on a series of examples with the help of the AI model.
- **AI MODEL:** Images are now blurred and you have access to an Artificial Intelligence (AI) model that provides their prediction
- **AI Button:** A new button now appears called "Use AI Answer" in blue and allows you to select the AI's answer which you can then submit. Please press the button if the AI helped with your answer and you want to use it's answer, if you agree with the answer of the AI but got the answer without it's help then don't press the button.
- If the AI predicts there to be a traffic light, it will put a box around it in the image and a score (you can hide or show it with a button below the image)

Try to do your best to get things correctly as there is a potential bonus of up to 2\$.

Please read instructions (25 secs)

Figure D.15: Testing phase instructions

Instructions (scroll down for task)

- **AI MODEL:** Images are now blurred and you have access to an Artificial Intelligence (AI) model that provides their prediction
- **AI Button:** A new button now appears called "Use AI Answer" in blue and allows you to select the AI's answer which you can then submit. Please press the button if the AI helped with your answer and you want to use it's answer, if you agree with the answer of the AI but got the answer without it's help then don't press the button.
- If the AI predicts there to be a traffic light, it will put a box around it in the image and a score (you can hide or show it with a button below the image)
- **Our Recommendation:** Additionally, above the button and the AI's answer, we give you a recommendation whether you should use the AI answer or on your own. This recommendation is based on comparing your estimated personal ability compared to the AI's ability, this recommendation is not always correct.

Try to do your best to get things correctly as there is a potential bonus of up to 2\$.

Please read instructions (14 secs)

Figure D.16: Testing phase instructions that include AI-integration recommendation



Hide AI answer

Show Instructions

Is there a traffic light in the picture?

Recommendation: Rely on AI Answer
Region: depict various city streets and highways during the daytime with different weather conditions, containing pedestrians, cars, trucks, traffic lights, and signs.

Yes

AI Predicts: NO with score: 0.0

No

Use AI Answer

Submit

Figure D.17: Prediction interface with AI and with AI-integration recommendations.

D.6.4 Screenshots of User Study Interface for MMLU

- We give you the following information about the AI model to help you understand when you should use it's answer and when not to:

Average AI Accuracy:	69%
Average Human Accuracy:	50% (varies between people and subjects)
AI Model Input:	Question and the answers
AI Model Output:	An answer and a text explanation
Source of Training Data:	Trained on all the internet
AI Training Objective:	Trained to mimic language and respond to instructions

- Now a deeper dive into the AI error/accuracy:

Category:	Accuracy
Machine Learning	43%
Management/Marketing	85%
Elementary Math	60%
Psychology/Sociology	86%
Global Facts	42%
Moral Scenarios	44%
Logical Fallacies	78%
High School Statistics	59%

Figure D.18: Model Card for MMLU study

Question: _____ is an assessment of a company's mix of products, services, investments, and other assets in order to optimize the use of resources and to assess its suitability, level of risk, and expected financial return.
 A: SWOT analysis.
 B: Political analysis.
 C: Portfolio analysis.
 D: Competitive analysis.


The screenshot displays a user interface for a prediction task. On the left, there are four green buttons labeled A, B, C, and D. To the right, a blue-bordered box contains the AI's prediction: 'AI Predicts: C' followed by a detailed explanation: 'Answer: C. Portfolio analysis. It involves evaluating a company's product portfolio to determine its strengths and weaknesses, as well as its suitability, level of risk, and expected financial return. This allows the company to allocate its resources more effectively and efficiently.' Below this box is a blue button labeled 'Use AI Answer'. At the bottom center of the interface is a grey button labeled 'Submit'.

Figure D.19: Prediction Interface for MMLU study

AI Predicts: C

Wrong: You predicted incorrectly., the correct answer was C.

AI was Correct: The AI predicted correctly.



This example is part of a larger set of examples with the following recommendation:

Recommendation: Rely on AI Answer
 related to marketing, including pricing strategies, branding, communication, product classification, market segmentation, advertising, and supply chain management..

Region Information:
 AI Accuracy: 92.0
 Human Accuracy: 40.0

Here are more examples from this region for you to generalize from

Question: Which of the following is an example of costbased pricing?
 A: Basing the price on what the customer is willing to pay
 B: Basing the price on the psychological expectations of the customer
 C: Basing the price on the features and benefits of the product
 D: Basing the price on the full cost of production plus the required profit

Figure D.20: Feedback shown to user during teaching phase

Show Instructions

Question: If a child inherits a physical trait from his father he will likely receive

A: tooth gap
 B: favorite color
 C: sense of fun
 D: liking dogs

Recommendation: Ignore AI Answer
 AI often wrong when various questions on different topics such as algebra, biology, public relations, and statistics with multiple options to choose from.

A

B

C

D

AI Predicts: A
 The answer is A. tooth gap because physical traits, such as the gaps between teeth, are primarily determined by genetics and can be passed down from parents to children.

Use AI Answer

Submit

Figure D.21: Prediction Interface for MMLU study with AI-integration recommendations.

Appendix E

Additional Information for Chapter 6

E.1 Programmer Behavior by Task

The previous statistics in Figure 6.5 were aggregated across all participants (and hence tasks). We now investigate differences across the tasks the participants solved. Table E.1 shows the acceptance rate of suggestion by task as well as the top 3 CUPS state by time spent. We first notice that there is variability in the acceptance rate; for example, the difference between the acceptance rate for the ‘Data Manipulation’ and ‘Classes and Boilerplate Code’ tasks is 17.1%. When we look at the most frequented CUPS states for participants in these two tasks, we notice stark differences: those in the data manipulation task spent 20.63% of their time thinking about new code to write and 16.48% looking up documentation online, while those in the boilerplate code task spent most of their time verifying suggestions and prompt crafting (=56.36%). This could be due to the fact the boilerplate code is very suitable for an AI assistant like Copilot while the data manipulation requires careful transformation of a dataset. However, we find that ‘Verifying Suggestion’ is in the top 3 states in terms of time spent in the coding session for all but two tasks, indicating similar behavior across tasks.

E.2 Predicting CUPS from Telemetry

Objective. To scale some of our insights, we need to be able to identify and predict programmers’ CUPS state. We discuss how we can use telemetry data to predict using machine learning classifiers the CUPS state of the programmer. This would enable us to accomplish two goals: 1) use the predictive models on the fly to perform interventions in the user interface and 2) use the predictive models to label previously collected telemetry with CUPS states to perform retrospective analysis such as in section 6.6.

Setup. The telemetry dataset represented as $\mathbf{D} = \{D_i\}$ collected in our study contains, for each user i a list of events occurring in the corresponding as D_i . An event is defined as a segment of the telemetry that culminates in a shown, accept, or reject programmer action (refer to Figure 7.2). For the purpose of this analysis, we only retain the shown events (labeled as “User Typing or Paused” in Figure 6.3)¹. The list of events for programmer i is $D_i = \{x_{ij}, y_{ij}\}$ where x_{ij} is the features for the event j and y_{ij} is the CUPS state for the event j . Our machine learning models will aim to predict the label y_{ij} . We extract features x_{ij} for each event as follows: the length of the document, previous actions, suggestion features (e.g., suggestion length), the confidence reported by Copilot, presence of Python keywords (e.g., import, def try, etc.), and the output of the Tree-sitter Parser². Finally, we extract features of the prompt including its textual features and parser outputs. It is crucial to note that the model features do not leak any information about the future and can be computed as soon as a suggestion is generated by Copilot.

Experimental Results. Using a leave-one-out programmer evaluation strategy where we train on data of 20 programmers and leave out one programmer for testing, we train an eXtreme Gradient Boosting (XGB) [220] model for this task for each trial (21 total) and evaluate the accuracy on the test set. The XGB model achieves an average accuracy of $30.8\% \pm 1.9$. In comparison, a baseline that always predicts the majority state achieves $24.9\% \pm 3.0$ accuracy, indicating that the XGB model has non-trivial performance – through there is considerable room for improvement. Nevertheless, while the accuracy reported is low, if we restrict the task to just predicting the most common state of Thinking/Verifying Suggestion (the rest is background) we obtain an area under the receiver operating characteristic curve (AUC) of 0.69 ± 0.02 which shows good predictive power. This shows that there are signals in the telemetry to be able to predict CUPS states. However, this XGB model accuracy is not sufficient to power our proposed interventions but perhaps a larger amount of labeled data can help build more reliable models to execute our proposed interventions. We discuss in the future section other avenues to improve the prediction of CUPS states from telemetry.

¹Note that consecutive shown followed by either accept/reject events share the same suggestion and prompts and so are very difficult to distinguish from only telemetry.

²<https://tree-sitter.github.io/tree-sitter/>

E.3 Details User Study

E.3.1 Interfaces

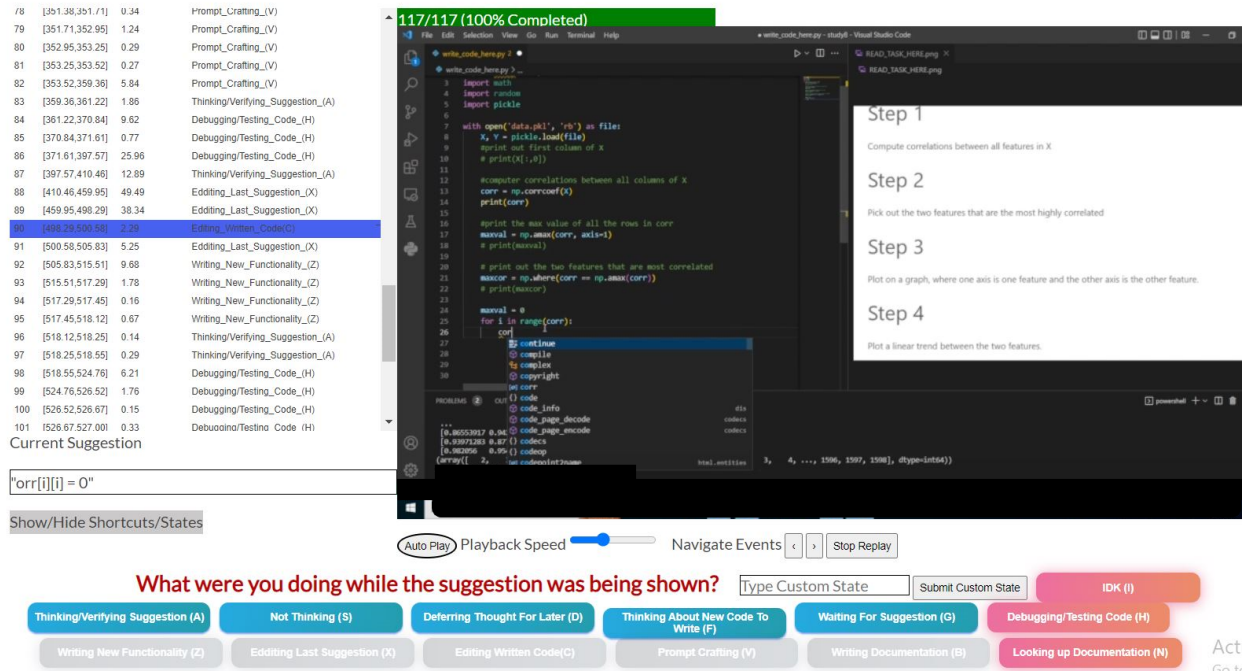


Figure E.1: Screenshot of Labeling Tool represented in Figure 6.4

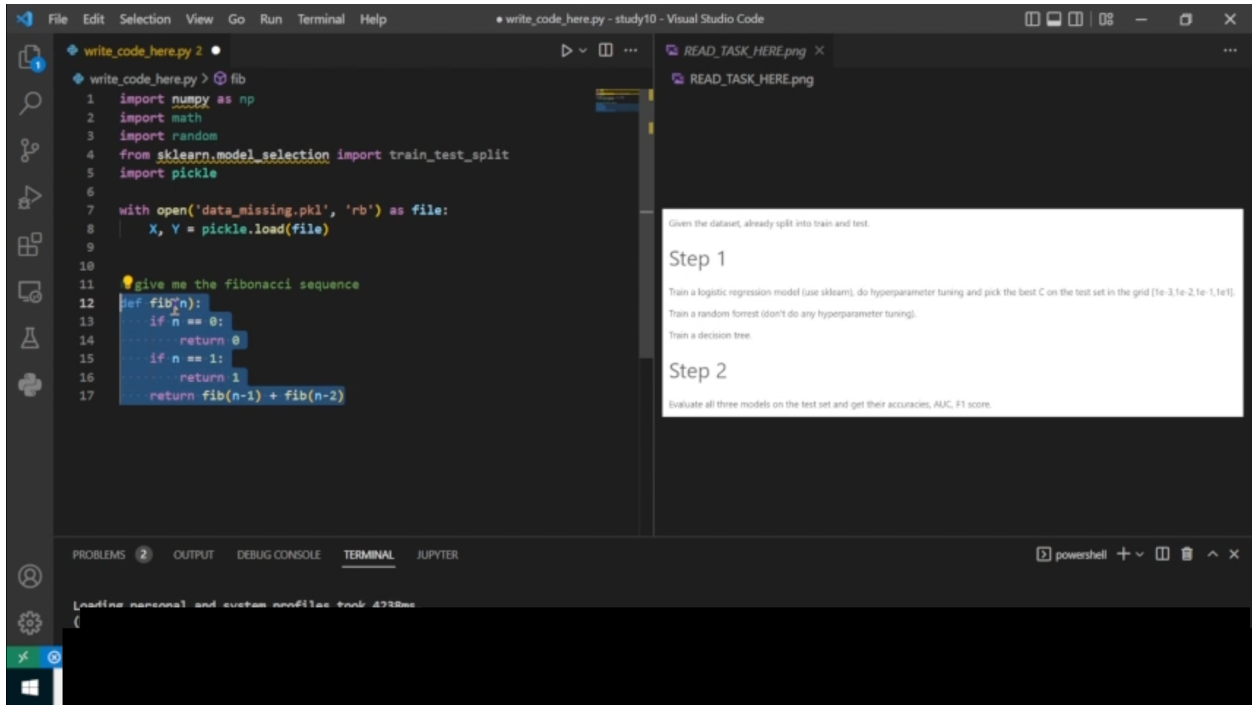


Figure E.2: Screenshot of Virtual Machine interface with VS Code

E.3.2 Task Instructions

The tasks are shown to participants as image files to deter copying of the instructions as a prompt.

Step 1:

First split the data into a train-test split with 80-20 split. Use the `train_test_split` function from sklearn.

Step 2:

Then impute the train and test data matrices by using the average value of each feature. Do this with just numpy operations.

Step 3:

Then, use the train and test data matrices to train a model. We will now do some feature engineering. We will code from scratch the creation of quadratic features.

Transform the data to include quadratic features, i.e. suppose we had a feature vector

$$[x_1, x_2]$$

we want to transform it to:

$$[x_1, x_2, x_1^2, x_2^2, x_1x_2]$$

If the previous feature dimension was d , it will now become $2d + \frac{d(d-1)}{2}$

Transform both train and test splits and store them in a different data matrix

Figure E.3: Data Manipulation Task.

Step 1

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9` Output: `[0,1]` Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`.

Step 2

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that $i \neq j$, $i \neq k$, and $j \neq k$, and $nums[i] + nums[j] + nums[k] == 0$.

Notice that the solution set must not contain duplicate triplets.

Step 3

Given an array `nums` of n integers, return an array of all the unique quadruplets `[nums[a], nums[b], nums[c], nums[d]]` such that:

$0 \leq a, b, c, d < n$, a, b, c , and d are distinct. $nums[a] + nums[b] + nums[c] + nums[d] == target$ You may return the answer in any order.

Example 1:

Input: `nums = [1,0,-1,0,-2,2]`, `target = 0` Output: `[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]`

Figure E.4: Algorithmic Problem Task.

Step 1

Compute correlations between all features in X

Step 2

Pick out the two features that are the most highly correlated

Step 3

Plot on a graph, where one axis is one feature and the other axis is the other feature.

Step 4

Plot a linear trend between the two features.

Figure E.5: Data Analysis Task.

Step 1

Define a class for a node (call it Node) that has attributes: text (string), id (integer), location(string), time (float).

The class should have a constructor that can set all 4 values and has methods that set the value of each attribute to user specified value.

Furthermore, create a method that adds a certain value to the time attribute.

Step 2

Define a class for a graph (call it Graph) that has as attribute a list of nodes.

Create a method that appends an element to the list of nodes.

Create a method that calculates the total time for all the nodes in the Graph.

Create a method that prints the name of all the nodes in the graph.

Figure E.6: Classes and Boilerplate Code Task.

Logistic Regression

We will implement a custom logistic regression classifier with L2 regularization for this task.

Recall: logistic regression we learn a weight vector $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, and predict the probability of the label being 1 as $\frac{1}{1 + \exp(-(w^T x + b))}$ where this is the sigmoid function applied to $w^T x + b$

To learn the weights, we use gradient descent:

for each iteration we do the update:

$$w \leftarrow w + \alpha * \left(\sum_i x_i * (Y_i - \text{sigmoid}(w_i^T x + b)) \right) - 2\lambda w$$

and

$$b \leftarrow b + \alpha * \left(\sum_i (Y_i - \text{sigmoid}(w_i^T x + b)) \right)$$

Implement a logistic regression that can handle custom number of iterations, specified learning rate alpha, specified regularization parameter lambda.

Fit the model on the training data.

Compare the accuracies on the test sets.

Try for 100 iterations, 0.1 learning rate and 1e-5 for lambda.

Figure E.7: Logistic Regression Task

Editing Existing Code

Given the following class, this class is a Retriever which given a set of numerical vectors and a parameter k, can return the k-nearest neighbors of a given vector.

Perform the following edits to the code:

- write a method that returns the least similar k vectors
- write a method that given a set of query vectors, returns the top k vectors for each of the query vectors
- create a method to append new vectors to the already vectors in Retriever
- create a new distance function that instead of norm we make it a weighted distance as follows:

Compute maximum scale of each feature on the training set:

$$scales = [\max_i(X_{1,i}), \dots, \max_i(X_{d,i})]$$

Then let the distance function be:

$$dist(x, z) = \sum_i \frac{1}{scales[i]} * (x_i - z_i)^2$$

- create a method to change k to user specified value

Figure E.8: Editing Code Task

Machine Learning

Training and evaluating a model

Given the dataset, already split into train and test.

Step 1

Train a logistic regression model (use sklearn), do hyperparameter tuning and pick the best C on the test set in the grid [1e-3,1e-2,1e-1,1e1].

Train a random forrest (don't do any hyperparameter tuning).

Train a decision tree.

Step 2

Evaluate all three models on the test set and get their accuracies, AUC, F1 score.

Figure E.9: Machine Learning Task

Task

We want to test an api for the following task:

- Given a string `s` containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order.

Example 1:

Input: `s = "()"` Output: true Example 2:

Input: `s = "()[]{}"` Output: true Example 3:

Input: `s = "(]"` Output: false

Constraints: $1 \leq s.length \leq 104$ `s` consists of parentheses only `'()[]{}'`

TODO: Write several test functions to make that the API function `isValid(str)` works properly.

- Create a class called `Testing`, inside that class write different test functions that test different aspects of the API (e.g. does it work with `'()'`), aim for 4 tests.
- Write a method that runs all the tests and returns the average success rate, the standard deviation of the success rate.

Figure E.10: Writing Tests Task

E.3.3 Survey Questions Results

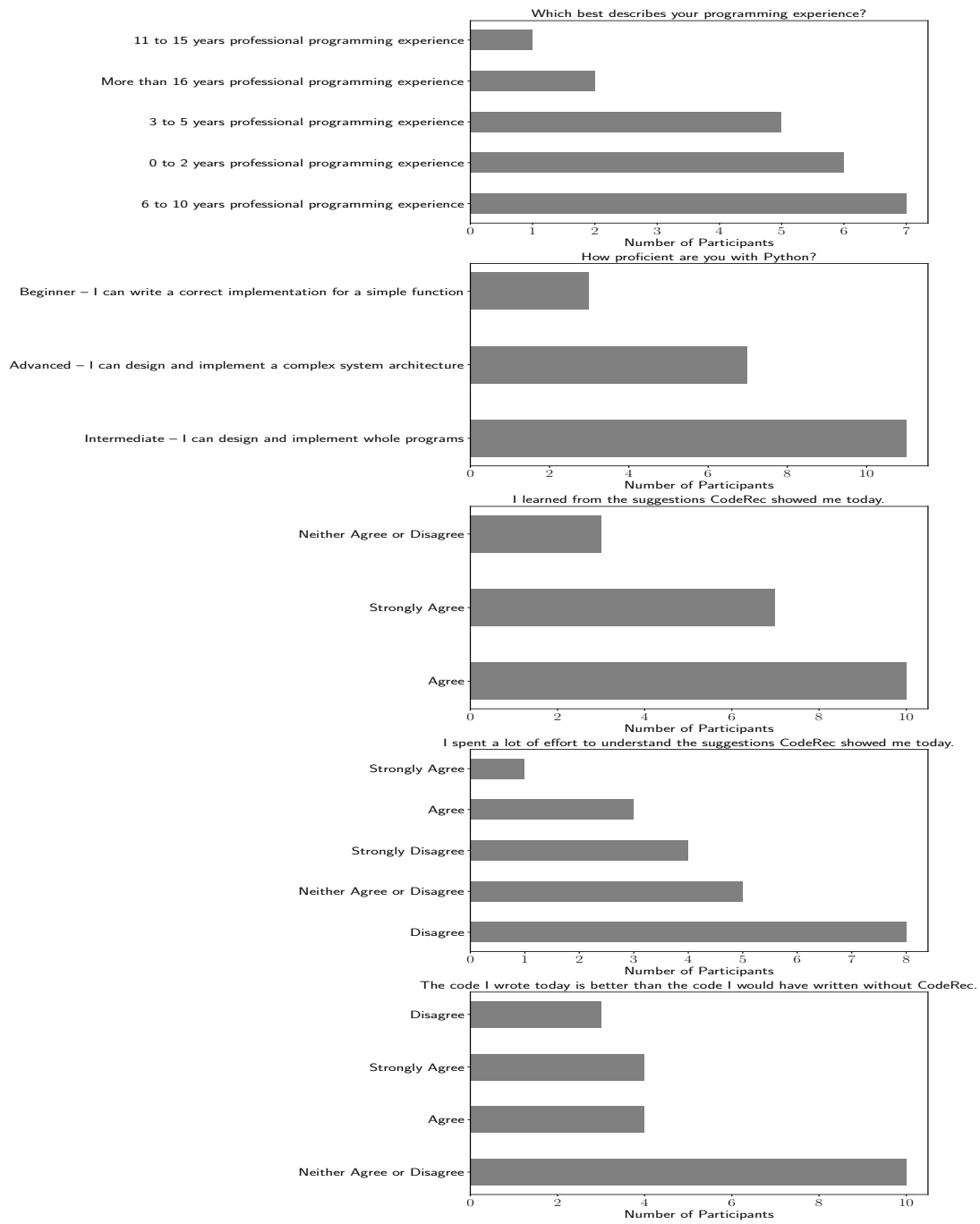


Figure E.11: User Study Survey results (1)

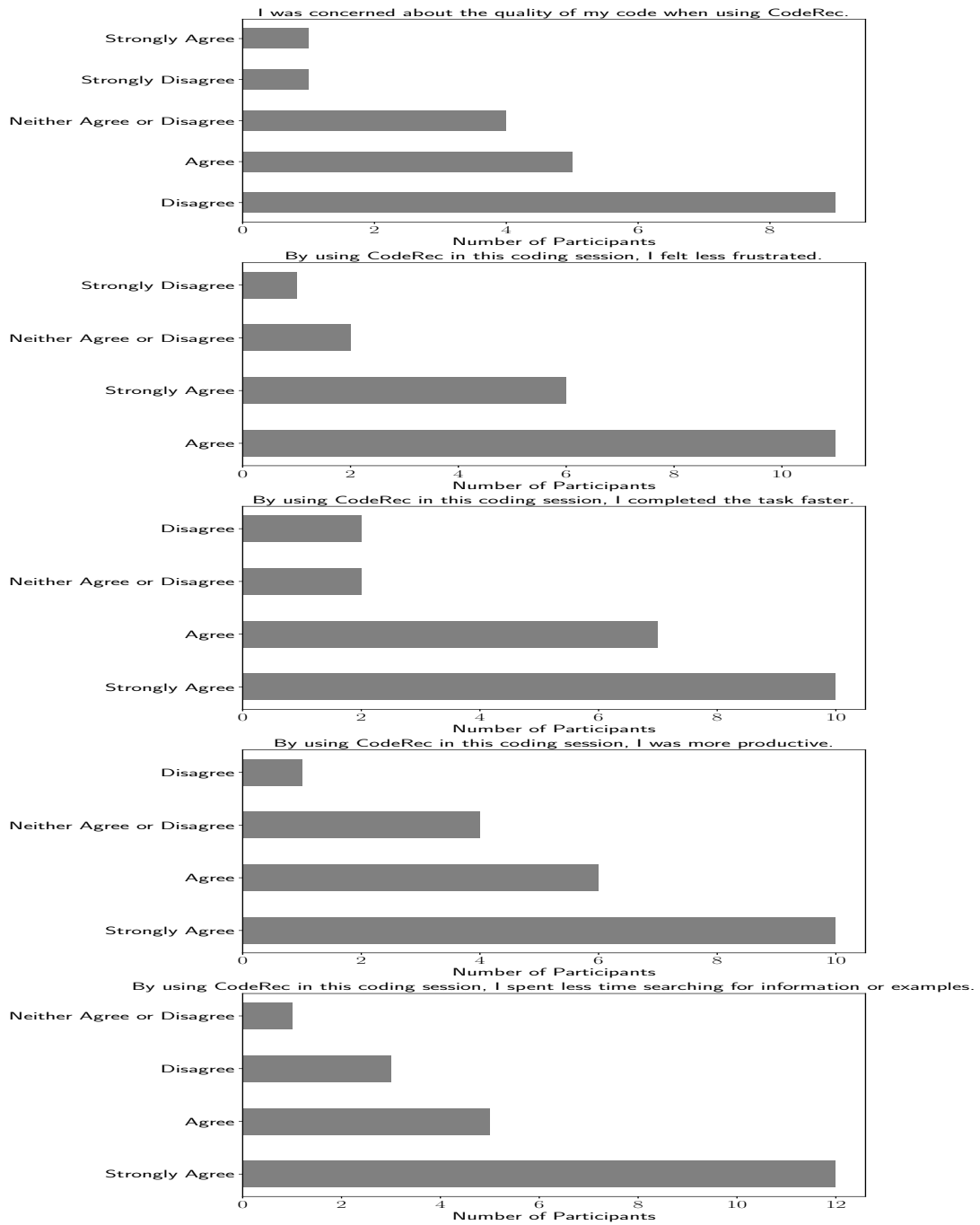


Figure E.12: User Study Survey results (2)

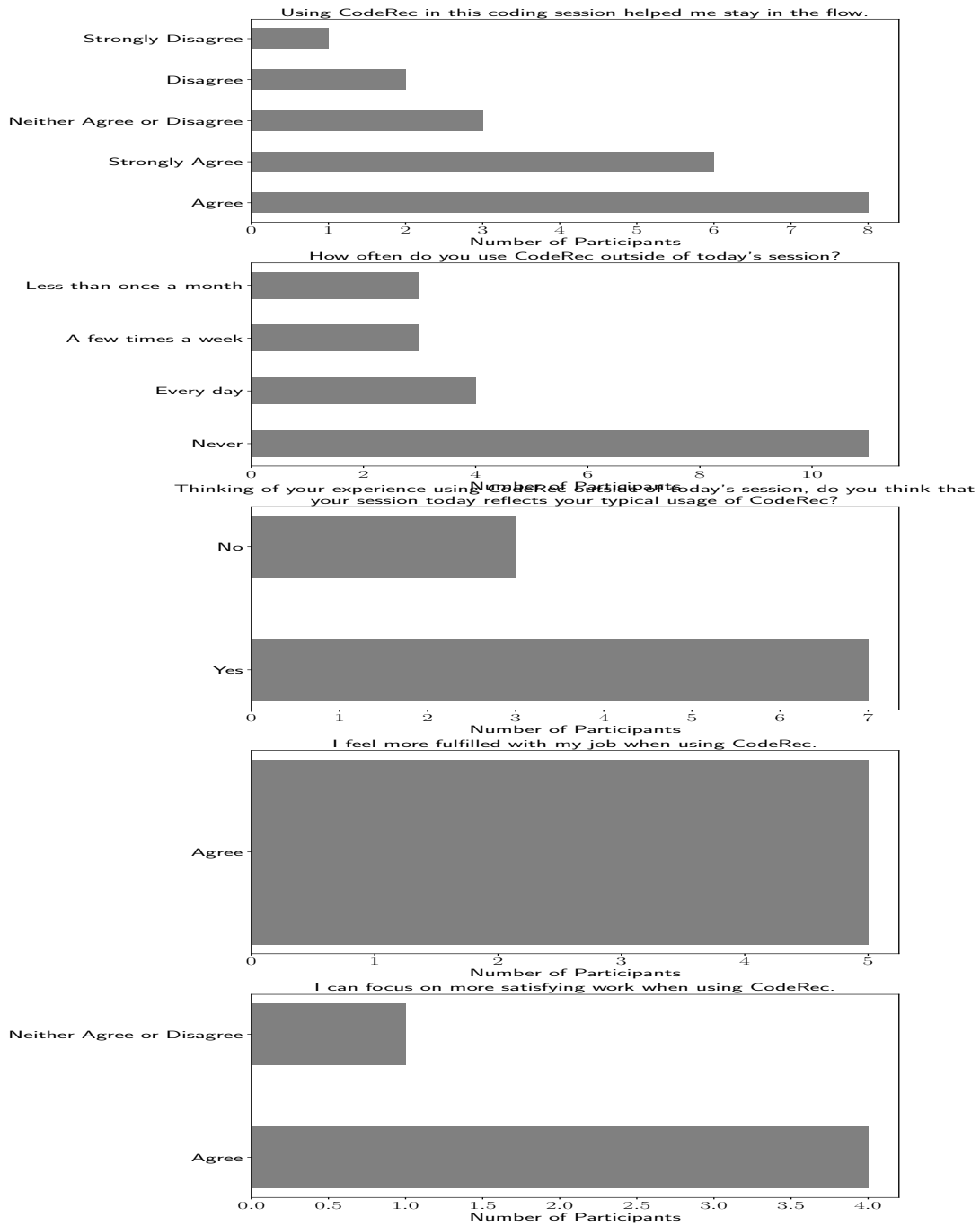


Figure E.13: User Study Survey results (3)

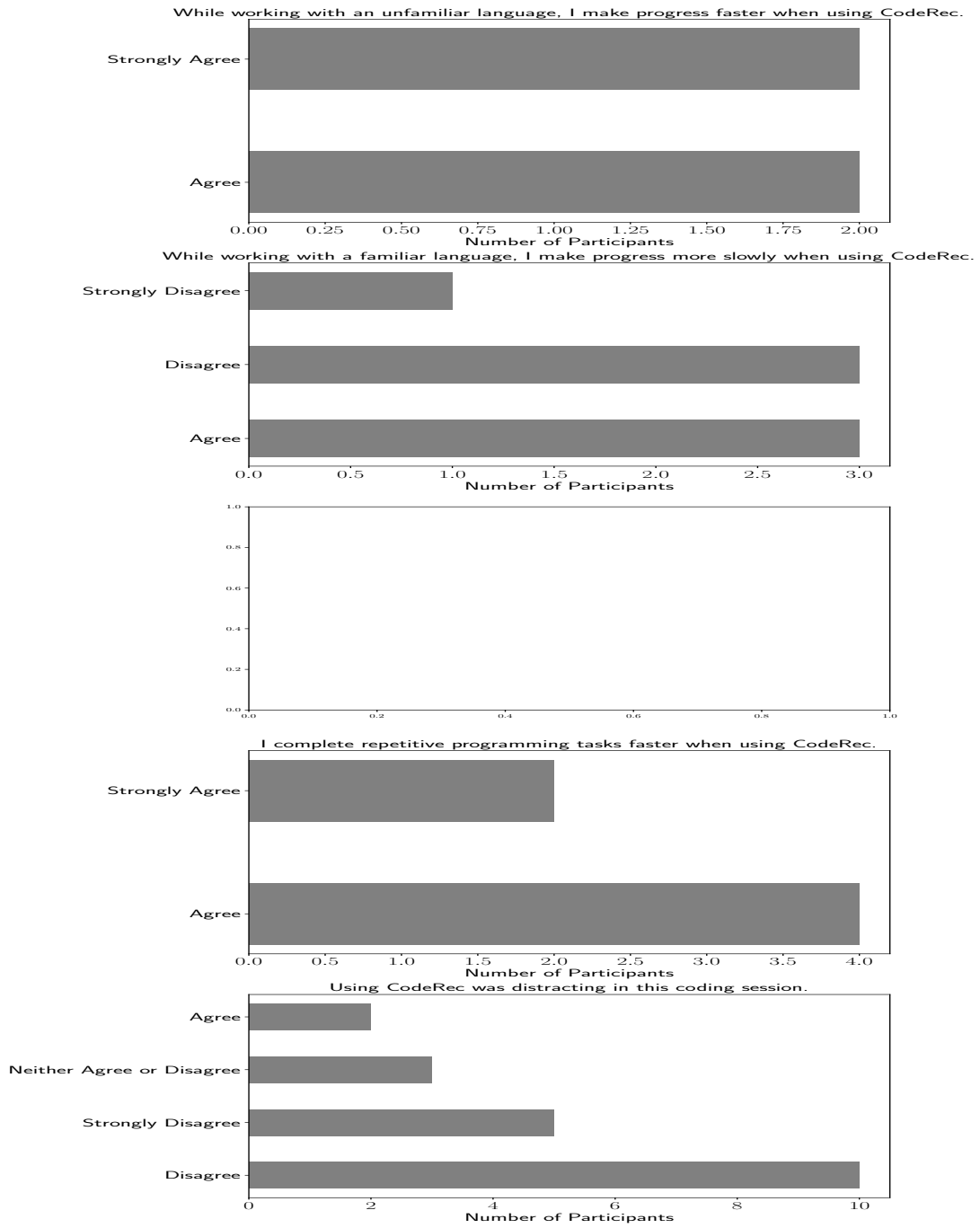


Figure E.14: User Study Survey results (4)

E.3.4 Full User Timelines

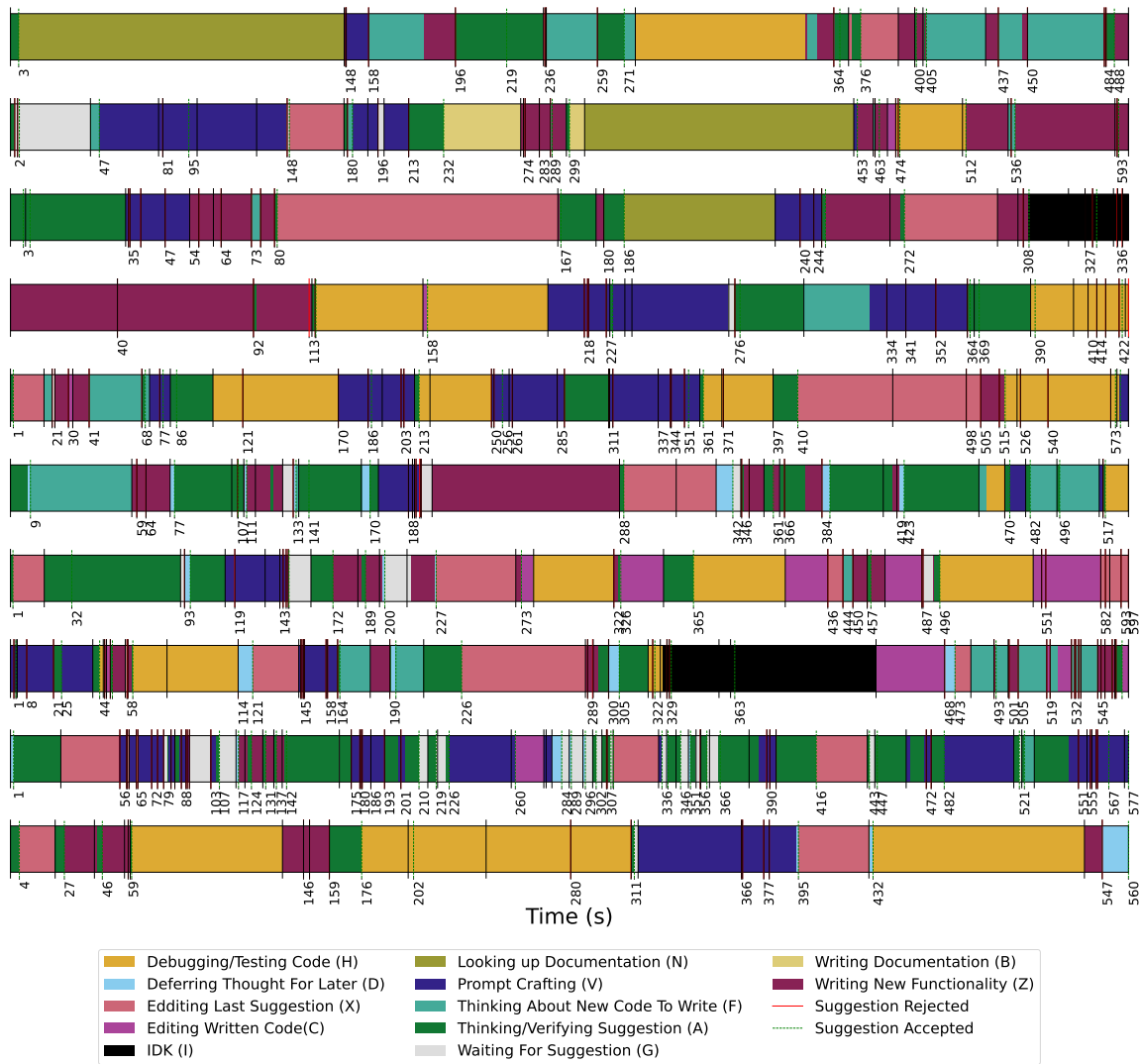


Figure E.15: Participants timelines for the first 10 minutes of their sessions (P1 to P10)

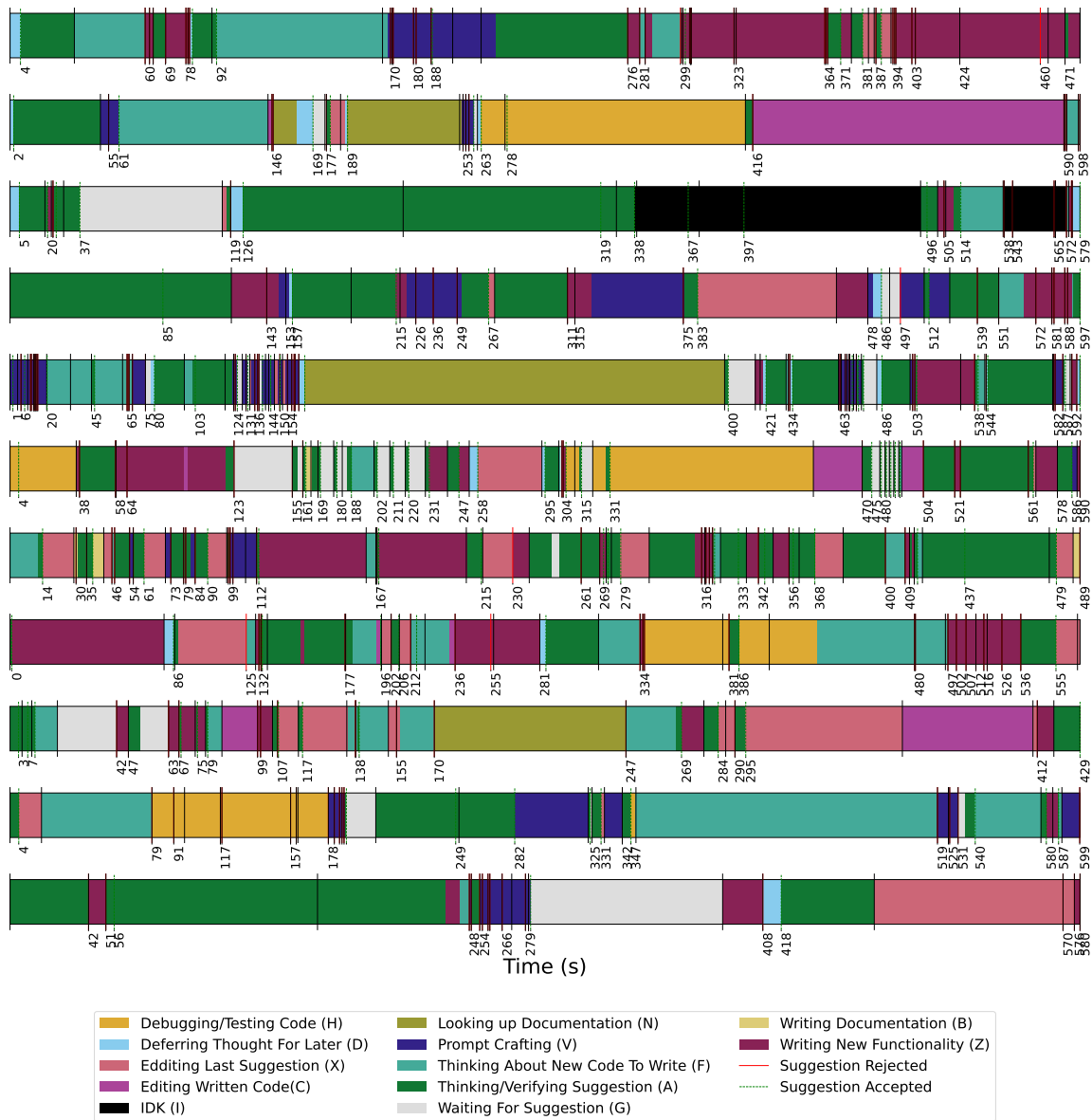


Figure E.16: Participants timelines for the first 10 minutes of their sessions (P11 to P21)

E.3.5 Full CUPS Graph



Figure E.17: CUPS diagram with all transitions shown that occur with probability higher than 0.05

Table E.1: Acceptance rate and the top three CUPS states in terms of time spent as a fraction of session time for each of the tasks. We include standard errors of the acceptance rate aggregated across participants.

Task Name	# Suggestions	Acceptance Rate %	Top 3 States (time %)
Algorithmic Problem	124	30.6 ± 26.6	Verifying Suggestion (25.58) Writing New Functionality (22.31), Thinking About New Code To Write (19.23)
Data Manipulation	238	24.8 ± 22.6	Thinking About New Code To Write (20.63) Looking up Documentation (16.48), Prompt Crafting (16.38)
Data Analysis	114	29.8 ± 32.3	Debugging/Testing Code (21.23) Editing Last Suggestion (16.62) Prompt Crafting (16.00)
Machine Learning	162	33.9 ± 23.7	Looking up Documentation (19.98) Verifying Suggestion (19.01) Debugging/Testing Code (12.52)
Classes and Boilerplate Code	112	41.9 ± 34.9	Verifying Suggestion (30.34) Prompt Crafting (26.02) Writing New Functionality (13.56)
Writing Tests	83	55.4 ± 49.7	Verifying Suggestion (20.79) Debugging/Testing Code (19.68) Writing New Functionality (16.91)
Editing Code	117	23.9 ± 24.6	Verifying Suggestion (30.18) Editing Last Suggestion (14.65) Writing New Functionality (14.24)
Logistic Regression	74	55.4 ± 35.1	Verifying Suggestion (30.28) Editing Last Suggestion (25.60) Writing New Functionality (15.69)

Appendix F

Additional Information for Chapter 7

F.1 Extended Related Work

AI-Assisted Programming. Large language models (LLMs) such as GPT-3 [127], have been widely used in natural language processing. One example of this is Codex [169], a GPT model trained on 54 million GitHub repositories, which demonstrates the effectiveness of LLMs in solving various programming tasks. For instance, Codex was tested on the HumanEval dataset of 164 programming problems, where it was asked to write the function body from a docstring and achieved 37.7% accuracy with a single generation [169]. Different metrics and datasets have been proposed to evaluate the performance of code recommendation models, but these typically assess how well the model can complete code in an offline setting without developer input, rather than evaluating how well it assists programmers in-situ [5], [168], [176], [177]. Researchers have found that developers do not need a perfect recommendation model for it to be useful. Weisz et al. conducted interviews with developers and found that they did not require a perfect recommendation model for the model to be useful [180], while Ziegler et al. surveyed over 2,000 Copilot users and found that they felt more productive using Copilot [170]. A study by Google found that an internal CodeRec model had a 6% reduction in 'coding iteration time' [184]. On the other hand, a study of 24 participants by Vaithilingam et al. showed no significant improvement in task completion time, yet participants stated a clear preference for Copilot [171]. [194] showed that interaction with Copilot falls into two broad categories: the programmer is either in 'acceleration mode' or in 'exploration mode'. A similar strategy to selectively hide code suggestions was proposed in [206] (Quality Estimation Before Completion, QEBC). However, QEBC [206] is not based on human feedback of accepting suggestions, but rather is based on constructing a learned estimator of the quality of code completions from datasets of paired code segments and model completions. In contrast, our CDHF estimator uses real programmer behavior data and is based on data from a code-recommendation system in current use (Copilot) as opposed to custom-trained ones in [206].

Human Feedback. Integrating human preference when training machine learning based models have long been studied in the literature [207], [208]. In particular, Reinforcement Learning from Human Feedback is an approach where the designer first gathers explicit human preference over actions which is used to improve the model using RL [264]. More recently, this approach has been used to improve LLMs for different tasks [209], [210] notably including ChatGPT [211] and consists of three steps: gather human preference over options, train a reward model of human preference, use the reward model to update the LLM using RL. In contrast, our approach CDHF uses implicit human feedback through telemetry more readily collected which is not fully reflective of true preference and avoids updating the LLM. Collaborative filtering employs human preference data to re-rank content [265], though avoids the complexities of both generating and ranking content which is required here. Our theoretical formulations in Section 7.4 build on the work of interactive user interfaces [212], [266] and generalize them to our setting. Work on algorithmic deferral [26]–[28] investigates a similar question on whether the human or the AI should accomplish the task and is based on error estimation (instead of time cost estimation here) of the human versus the AI, as well as work on personalized decision support policies [147] investigates whether support should be shown but not when.

F.2 Derivation of \mathbb{P}^*

Proposition 1. *Under assumptions of the programmers model, notably that the programmer spends more time writing code when they reject a suggestion compared to when they accept a suggestion and edit it. Given specific code, suggestion and latent state (X, S, ϕ) , if the programmer’s probability of accepting $\mathbb{P}(A = \text{accept}|X, S, \phi)$ a suggestion is below \mathbb{P}^* defined as:*

$$\mathbb{P}^* = \frac{\mathbb{E}[\text{verification}] + \mathbb{E}[\tau|X, \phi]}{\mathbb{E}[\text{writing}|A = \text{reject}] - \mathbb{E}[\text{editing}|A = \text{accept}]}$$

then the suggestion should not be shown. Note that \mathbb{P}^ is defined as a function $\mathbb{P}^*(X, S, \phi)$ evaluated pointwise.*

Proof. Starting from the equation $\delta = 0$, assume that the programmer has only two actions of accept or reject. This is backed by our analysis of our telemetry sample where we noticed that less than 1% of suggestions are browsed. We first have:

$$\begin{aligned} & - (\mathbb{E}[\text{verification}|X, S, \phi] + \mathbb{P}(A = \text{accept}|X, S, \phi) \cdot \mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}] \\ & + \mathbb{P}(A = \text{reject}|X, S, \phi) \cdot \mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}]) + \mathbb{E}[\text{writing}|X, \phi] - \mathbb{E}[\tau|X, \phi] = 0 \end{aligned}$$

We now replace $\mathbb{P}(A = \text{accept}|X, S, \phi)$ by $\mathbb{P}^*(X, S, \phi)$ and move around terms:

$$\begin{aligned} & \mathbb{P}^*(X, S, \phi) \cdot (-\mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}] + \mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}]) \\ &= \mathbb{E}[\text{verification}|X, S, \phi] + \mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{writing}|X, \phi] + \mathbb{E}[\tau|X, \phi] \end{aligned}$$

Assuming that $\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}]$ is > 0 , meaning when suggestions are accepted, the editing time for them is less than the time to write the suggestions. We can then separate $\mathbb{P}^*(X, S, \phi)$ to the LHS with full equivalence (equality still holds without this assumption):

$$\mathbb{P}^*(X, S, \phi) = \frac{\mathbb{E}[\text{verification}|X, S, \phi] + \mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{writing}|X, \phi] + \mathbb{E}[\tau|X, \phi]}{\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}]}$$

Note that $\mathbb{P}^*(X, S, \phi)$ is not necessarily a valid probability in $[0, 1]$. If we assume that if the programmer rejected a suggestion, they did not benefit at all from it when writing the code afterwards meaning $\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] = \mathbb{E}[\text{writing}|X, \phi]$, we arrive at:

$$\mathbb{P}^*(X, S, \phi) = \frac{\mathbb{E}[\text{verification}|X, S, \phi] + \mathbb{E}[\tau|X, \phi]}{\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}]}$$

Assuming latency is zero meaning $\tau = 0$, then it is clear that $\mathbb{P}^*(X, S, \phi) = 0$ if and only if verification time is negligible: $\mathbb{E}[\text{verification}|X, S, \phi] = 0$, and by assumption it is $\mathbb{P}^*(X, S, \phi) \geq 0$ since the denominator is positive. The implication is that if $\mathbb{P}(A = \text{accept}|X, S, \phi) \leq \mathbb{P}^*(X, S, \phi)$, we should not show the suggestion which follows directly from the equation of $\delta \leq 0$, and similarly if $\mathbb{P}(A = \text{accept}|X, S, \phi) \geq \mathbb{P}^*(X, S, \phi)$ we show the suggestion.

To restate, this derivation made the following assumptions:

- The programmer has only two actions of accept or reject.
- $\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] - \mathbb{E}[\text{editing}|X, S, \phi, A = \text{accept}]$ is > 0 , meaning when suggestions are accepted, the editing time for them is less than the time to write the suggestions.
- If we assume that if the programmer rejected a suggestion, they did not benefit at all from it when writing the code afterwards meaning $\mathbb{E}[\text{writing}|X, S, \phi, A = \text{reject}] = \mathbb{E}[\text{writing}|X, \phi]$.

□

F.3 Model Evaluation and Analysis

All experiments were run with Python 3.8 on a machine with a single A100 GPU.

Confidence intervals in the body are obtained by bootstrapping with 1000 bootstrap samples for accuracy and AUC.

In table F.1 we evaluate the performance of different models for predicting acceptance of suggestions. We use Scikit-learn [251] to train the Random Forrest and logistic regression models, XGBoost library to train our XGB models [267] and PyTorch to train the fully connected neural network [268]. The FCNN is a 3 layer network with 50 hidden units trained using Adam with a learning rate of $1e - 2$. We used the validation set to select the number of hidden units as well as the picking the best model after training for 100 epochs. We did not employ any hyperparameter tuning for the XGBoost models and used the standard parameters.

Table F.1: Comparison of different classifiers on the test set based on AU-ROC, accuracy, and macro f1 for the full model to predict programmer suggestion acceptance. FCNN refers to a fully connected neural network.

Method	AU-ROC	Accuracy (%)	Macro F1 (%)
XGBoost	0.780	81.1	64
Logistic Regression	0.726	79.5	58
Random Forrest	0.756	80.9	61
FCNN	0.741	80.2	59
Baseline	0.5	78.9	44

In figure F.1 we showcase that the stage 2 model m_2 is well calibrated.

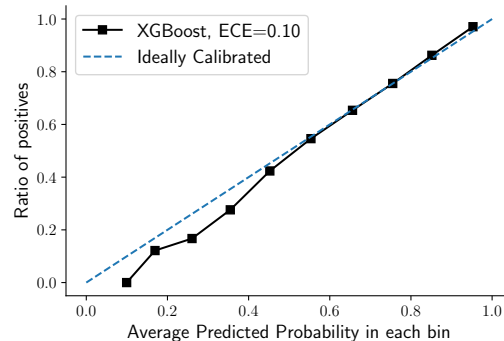


Figure F.1: Calibration curve for the XGBoost stage 2 model.

Sample Complexity. To understand how many samples we might need to train individualized CDHF models, we perform a sample complexity analysis on the acceptance prediction stage 2 model where we train with a random fraction of the training dataset in Figure F.2. With 1% of the training dataset which equates to 1688 training samples, performance reaches 0.69 AU-ROC. With 25% of training data, the model can achieve 0.75 AU-ROC.

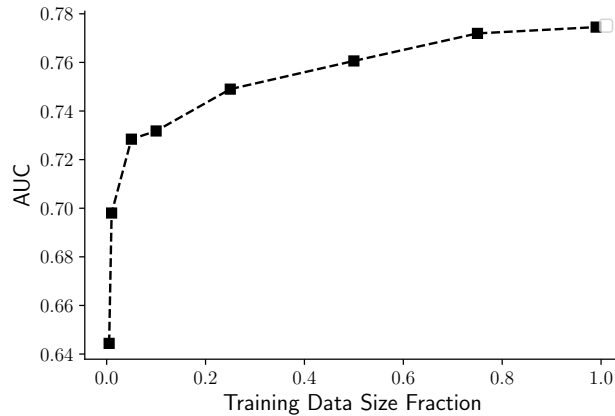


Figure F.2: Sample complexity analysis of the XGBoost stage 2 model when trained on a fraction of the training data and plotting the AU-ROC on the full test set.

Factors Influencing Programming Actions. An analysis of the XGBoost stage 2 model, reveals factors that correlate with programmers’ decisions. We examine feature importance weights (Figure F.3). Specifically, we report feature F-score counts, which capture how often each feature was split in any of the trees in the XGBoost model. The two most important features are the confidence of the suggestion from CodeRec’s core suggestion model, and the length of the suggestion. Together, these two features yield a model that achieves an AUROC of 0.71. Other important features include the context of the current event in the coding session such as if the last suggestion was accepted or not. We also see that textual elements of the suggestion are important. For example, the feature indicating if the current suggestion includes the character ‘#’ (which is used to indicate a comment in Python) was split a total of 8 times. These features should not be interpreted in a causal fashion but rather as being correlated with the behavior of programmers.

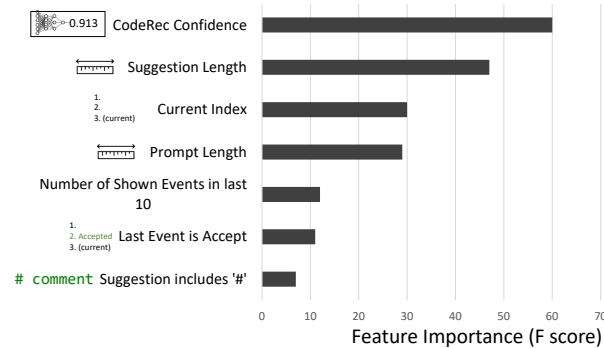


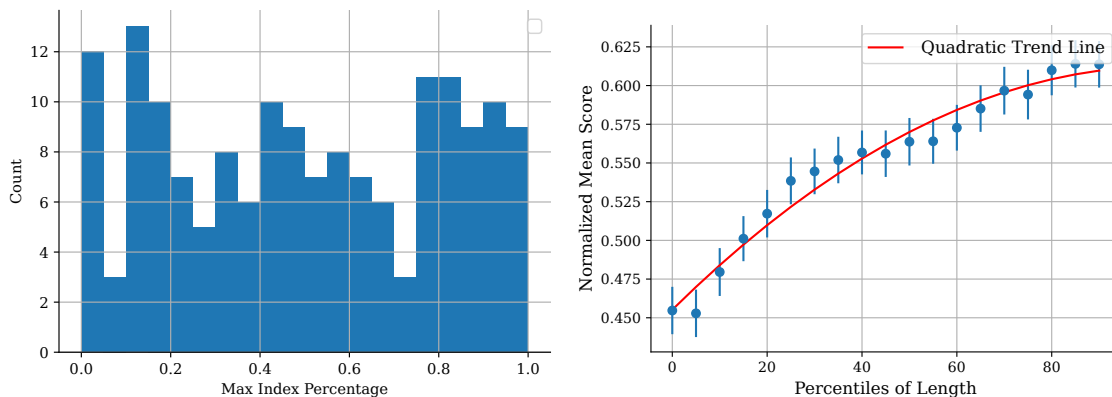
Figure F.3: Feature importance for the seven highest-rated unique features of the model for predicting the likelihood of accepting a suggestion. The feature importance is in terms of the F score which counts how often a feature was split on in the tree ensemble.

Analysis of Suggestions. The model learned on the large sample of telemetry can also be

applied to the telemetry collected in the 21-participant user study of [166]. When we evaluate the model on the user study’s 1029 accepted and rejected events we obtain an AU-ROC of 0.73. Inspecting these results further, we find that there are at least two defined clusters for suggestions predicted most likely be rejected: (1) single character non-alphabetic suggestions such as (,), [, ., ;, and (2) mid-word completions such as ‘agonal()’ (completion of ‘diagonal()’), ‘lass LogisticRegression:’ (completion of ‘class Logistic Regression:’). We hypothesize that for cluster (1) the suggestions were too small to be noticed. For cluster (2), we hypothesize that the programmer was already in the act of typing, so the suggestions may have been a distraction (i.e., the interruption cost more than was saved by eliminating the physical act of typing a few already-determined keystrokes).

F.4 Which Suggestion to Show: Plots

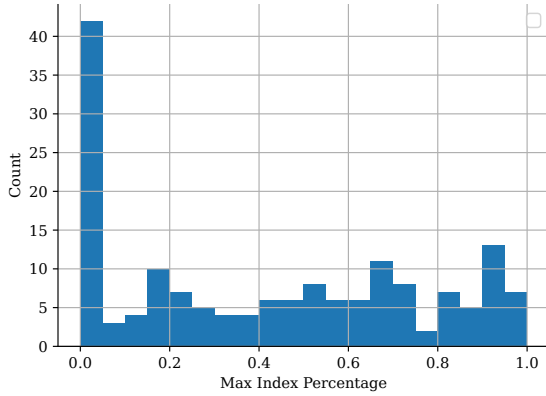
Following the last section of the paper, we show two plots for each $k \in \{0, 1, 2, 3\}$: 1) histogram showing in which length percentile in terms the ground truth solution dos the suggestion with highest acceptance probability (according to the model) lies and 2) normalized probability of acceptance by the length of the suggestion (for each example, we normalize the raw probability of acceptance by the maximum acceptance probability across all length for the given example - we observe the same trend without normalizing).



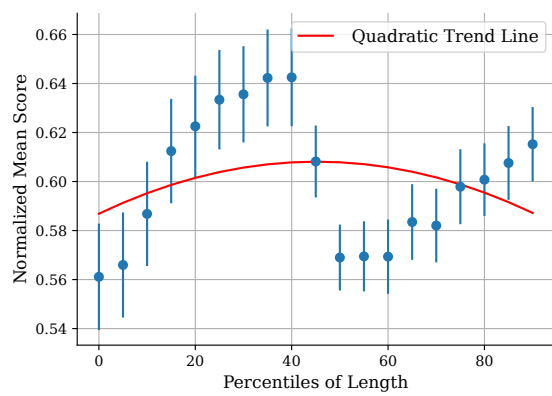
(a) Histogram of position of max suggestion

(b) Probability of acceptance by length

Figure F.4: Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 0$ (docstring only)

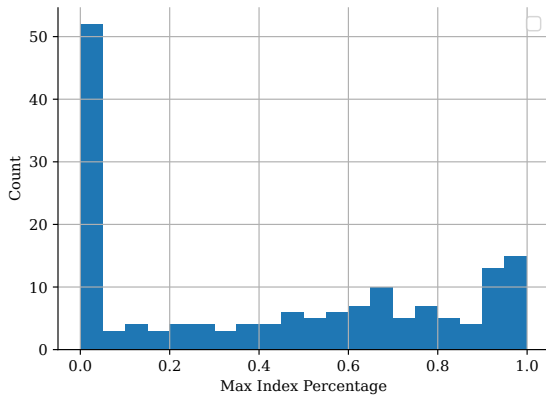


(a) Histogram of position of max suggestion

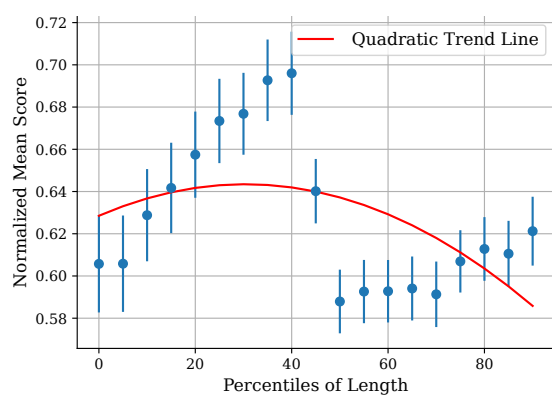


(b) Probability of acceptance by length

Figure F.5: Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 1$ (docstring + first line of solution)

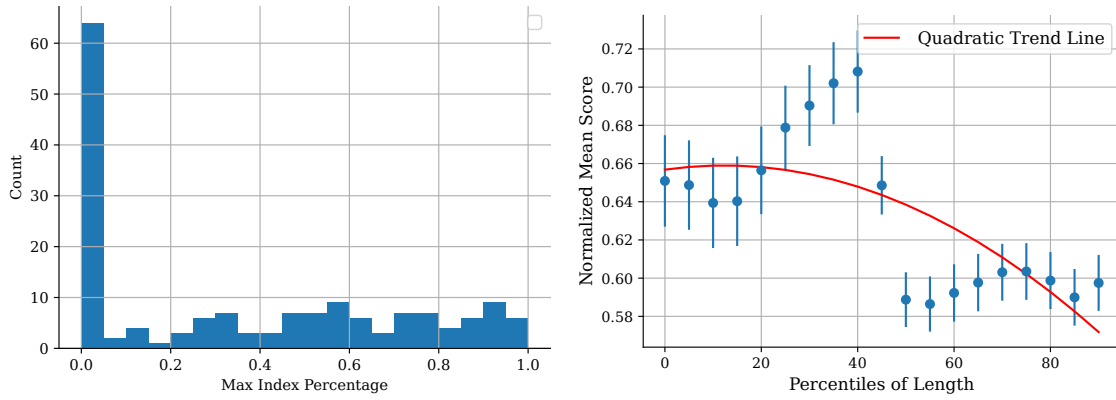


(a) Histogram of position of max suggestion



(b) Probability of acceptance by length

Figure F.6: Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 2$ (docstring + first two lines of solution)



(a) Histogram of position of max suggestion

(b) Probability of acceptance by length

Figure F.7: Plots for the experiment on ranking suggestions by the probability of acceptance. Histogram (a) shows in which length percentile bin the maximizing suggestion lies and Graph (b) shows the acceptance score by increasing the length of the suggestion. These plots are for $k = 3$ (docstring + first three lines of solution)

References

- [1] E. Beede, E. Baylor, F. Hersch, A. Iurchenko, L. Wilcox, P. Ruamviboonsuk, and L. M. Vardoulakis, “A human-centered evaluation of a deep learning system deployed in clinics for the detection of diabetic retinopathy,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.
- [2] S. Gaube, H. Suresh, M. Raue, A. Merritt, S. J. Berkowitz, E. Lermer, J. F. Coughlin, J. V. Guttag, E. Colak, and M. Ghassemi, “Do as ai say: Susceptibility in deployment of clinical decision-aids,” *NPJ digital medicine*, vol. 4, no. 1, pp. 1–8, 2021.
- [3] T. Gillespie, “Content moderation, ai, and the question of scale,” *Big Data & Society*, vol. 7, no. 2, p. 2053951720943234, 2020.
- [4] A. Coenen, L. Davis, D. Ippolito, E. Reif, and A. Yuan, “Wordcraft: A human-ai collaborative editor for story writing,” *arXiv preprint arXiv:2107.07430*, 2021.
- [5] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. Ming, *et al.*, “Github copilot ai pair programmer: Asset or liability?” *arXiv preprint arXiv:2206.15331*, 2022.
- [6] OpenAI, *Chatgpt: Introducing chatgpt*, <https://openai.com/blog/chatgpt>, 2022.
- [7] Github, *Github copilot - your ai pair programmer*, 2022. URL: <https://github.com/features/copilot>.
- [8] E. Kamar, S. Hacker, and E. Horvitz, “Combining human and machine intelligence in large-scale crowdsourcing,” in *AAMAS*, vol. 12, 2012, pp. 467–474.
- [9] S. Tan, J. Adebayo, K. Inkpen, and E. Kamar, “Investigating human+ machine complementarity for recidivism predictions,” *arXiv preprint arXiv:1808.09123*, 2018.
- [10] J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, B. Haghgoo, R. Ball, K. Shpanskaya, *et al.*, “Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 590–597.

- [11] C. Ni, N. Charoenphakdee, J. Honda, and M. Sugiyama, “On possibility and impossibility of multiclass classification with rejection,” *arXiv preprint arXiv:1901.10655*, 2019.
- [12] M. Jacobs, M. F. Pradier, T. H. McCoy, R. H. Perlis, F. Doshi-Velez, and K. Z. Gajos, “How machine-learning recommendations influence clinician treatment selections: The example of antidepressant selection,” *Translational psychiatry*, vol. 11, no. 1, pp. 1–9, 2021.
- [13] H. Liu, V. Lai, and C. Tan, “Understanding the effect of out-of-distribution examples and interactive explanations on human-ai decision making,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–45, 2021.
- [14] A. Kawakami, V. Sivaraman, H.-F. Cheng, L. Stapleton, Y. Cheng, D. Qing, A. Perer, Z. S. Wu, H. Zhu, and K. Holstein, “Improving human-ai partnerships in child welfare: Understanding worker practices, challenges, and desires for algorithmic decision support,” in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–18.
- [15] K. Vodrahalli, T. Gerstenberg, and J. Zou, “Uncalibrated models can improve human-ai collaboration,” *arXiv preprint arXiv:2202.05983*, 2022.
- [16] G. Bansal, B. Nushi, E. Kamar, W. S. Lasecki, D. S. Weld, and E. Horvitz, “Beyond accuracy: The role of mental models in human-ai team performance,” in *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, vol. 7, 2019, pp. 2–11.
- [17] H. Vasconcelos, M. Jörke, M. Grunde-McLaughlin, T. Gerstenberg, M. S. Bernstein, and R. Krishna, “Explanations can reduce overreliance on ai systems during decision-making,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 7, no. CSCW1, pp. 1–38, 2023.
- [18] K. C. Nanji, S. P. Slight, D. L. Seger, I. Cho, J. M. Fiskio, L. M. Redden, L. A. Volk, and D. W. Bates, “Overrides of medication-related clinical decision support alerts in outpatients,” *Journal of the American Medical Informatics Association*, vol. 21, no. 3, pp. 487–491, 2014.
- [19] R. Fok and D. S. Weld, “In search of verifiability: Explanations rarely enable complementary performance in ai-advised decision making,” *arXiv preprint arXiv:2305.07722*, 2023.
- [20] G. Bansal, B. Nushi, E. Kamar, D. S. Weld, W. S. Lasecki, and E. Horvitz, “Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2429–2437.
- [21] R. K. Atkinson, S. J. Derry, A. Renkl, and D. Wortham, “Learning from examples: Instructional principles from the worked examples research,” *Review of educational research*, vol. 70, no. 2, pp. 181–214, 2000.
- [22] J. Hattie and H. Timperley, “The power of feedback,” *Review of educational research*, vol. 77, no. 1, pp. 81–112, 2007.

- [23] M.-A. Charusaie, H. Mozannar, D. Sontag, and S. Samadi, “Sample efficient learning of predictors that complement humans,” in *International Conference on Machine Learning*, PMLR, 2022, pp. 2972–3005.
- [24] J. Kleinberg, H. Lakkaraju, J. Leskovec, J. Ludwig, and S. Mullainathan, “Human decisions and machine predictions,” *The quarterly journal of economics*, vol. 133, no. 1, pp. 237–293, 2018.
- [25] M. Raghu, K. Blumer, G. Corrado, J. Kleinberg, Z. Obermeyer, and S. Mullainathan, “The algorithmic automation problem: Prediction, triage, and human effort,” *arXiv preprint arXiv:1903.12220*, 2019.
- [26] H. Mozannar and D. Sontag, “Consistent estimators for learning to defer to an expert,” in *International Conference on Machine Learning*, PMLR, 2020, pp. 7076–7087.
- [27] N. Okati, A. De, and M. Gomez-Rodriguez, “Differentiable learning under triage,” *arXiv preprint arXiv:2103.08902*, 2021.
- [28] D. Madras, T. Pitassi, and R. Zemel, “Predict responsibly: Improving fairness and accuracy by learning to defer,” in *Advances in Neural Information Processing Systems*, 2018, pp. 6150–6160.
- [29] B. Wilder, E. Horvitz, and E. Kamar, “Learning to complement humans,” *arXiv preprint arXiv:2005.00582*, 2020.
- [30] M. F. Pradier, J. Zazo, S. Parbhoo, R. H. Perlis, M. Zazzi, and F. Doshi-Velez, “Preferential mixture-of-experts: Interpretable models that rely on human expertise as much as possible,” *arXiv preprint arXiv:2101.05360*, 2021.
- [31] N. Raman and M. Yee, “Improving learning-to-defer algorithms through fine-tuning,” *arXiv preprint arXiv:2112.10768*, 2021.
- [32] J. Liu, B. Gallego, and S. Barbieri, “Incorporating uncertainty in learning to defer algorithms for safe computer-aided diagnosis,” *arXiv preprint arXiv:2108.07392*, 2021.
- [33] V. Keswani, M. Lease, and K. Kenthapadi, “Towards unbiased and accurate deferral to multiple experts,” *arXiv preprint arXiv:2102.13004*, 2021.
- [34] S. Joshi, S. Parbhoo, and F. Doshi-Velez, “Pre-emptive learning-to-defer for sequential medical decision-making under uncertainty,” *arXiv preprint arXiv:2109.06312*, 2021.
- [35] R. Gao, M. Saar-Tsechansky, M. De-Arteaga, L. Han, M. K. Lee, and M. Lease, “Human-ai collaboration with bandit feedback,” *arXiv preprint arXiv:2105.10614*, 2021.

- [36] J. Zhao, M. Agrawal, P. Razavi, and D. Sontag, “Directing human attention in event localization for clinical timeline creation,” in *Machine Learning for Healthcare Conference*, PMLR, 2021, pp. 80–102.
- [37] E. Straitouri, A. Singla, V. B. Meresht, and M. Gomez-Rodriguez, “Reinforcement learning under algorithmic triage,” *arXiv preprint arXiv:2109.11328*, 2021.
- [38] M. T. Ribeiro, S. Singh, and C. Guestrin, ““ why should i trust you?” explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.
- [39] V. Lai, H. Liu, and C. Tan, ““ why is’ chicago’deceptive?” towards building model-driven tutorials for humans,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
- [40] H. Mozannar, A. Satyanarayan, and D. Sontag, “Teaching humans when to defer to a classifier via exemplars,” in *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*, 2022.
- [41] S. Su, Y. Chen, O. Mac Aodha, P. Perona, and Y. Yue, “Interpretable machine teaching via feature feedback,” 2017.
- [42] X. Zhu, A. Singla, S. Zilles, and A. N. Rafferty, “An overview of machine teaching,” *arXiv preprint arXiv:1801.05927*, 2018.
- [43] G. Kerrigan, P. Smyth, and M. Steyvers, “Combining human predictions with model probabilities via confusion matrices and calibration,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [44] C. Cortes, G. DeSalvo, and M. Mohri, “Learning with rejection,” in *International Conference on Algorithmic Learning Theory*, Springer, 2016, pp. 67–82.
- [45] C. Chow, “On optimum recognition error and reject tradeoff,” *IEEE Transactions on information theory*, vol. 16, no. 1, pp. 41–46, 1970.
- [46] P. L. Bartlett and M. H. Wegkamp, “Classification with a reject option using a hinge loss,” *Journal of Machine Learning Research*, vol. 9, no. Aug, pp. 1823–1840, 2008.
- [47] N. Charoenphakdee, Z. Cui, Y. Zhang, and M. Sugiyama, “Classification with rejection based on cost-sensitive classification,” in *International Conference on Machine Learning*, PMLR, 2021, pp. 1507–1517.
- [48] R. El-Yaniv and Y. Wiener, “On the foundations of noise-free selective classification,” *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1605–1641, 2010.

- [49] Y. Geifman and R. El-Yaniv, “Selective classification for deep neural networks,” in *Advances in neural information processing systems*, 2017, pp. 4878–4887.
- [50] A. Gangrade, A. Kag, and V. Saligrama, “Selective classification via one-sided prediction,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 2179–2187.
- [51] D. A. E. Acar, A. Gangrade, and V. Saligrama, “Budget learning via bracketing,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 4109–4119.
- [52] K. Shah and N. Manwani, “Online active learning of reject option classifiers,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5652–5659.
- [53] C. Zhang and K. Chaudhuri, “Active learning from weak and strong labelers,” in *Advances in Neural Information Processing Systems*, 2015, pp. 703–711.
- [54] T. Davidson, D. Warmsley, M. Macy, and I. Weber, “Automated hate speech detection and the problem of offensive language,” in *Eleventh international aai conference on web and social media*, 2017.
- [55] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, “Convexity, classification, and risk bounds,” *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156, 2006.
- [56] T. Zhang, “Statistical analysis of some multi-category large margin classification methods,” *Journal of Machine Learning Research*, vol. 5, no. Oct, pp. 1225–1251, 2004.
- [57] A. Nowak-Vila, F. Bach, and A. Rudi, “A general theory for structured prediction with smooth convex surrogates,” *arXiv preprint arXiv:1902.01958*, 2019.
- [58] S. Hanneke, “Theory of active learning,” *Foundations and Trends in Machine Learning*, vol. 7, no. 2-3, 2014.
- [59] A. Krishnamurthy, A. Agarwal, T.-K. Huang, H. Daumé III, and J. Langford, “Active learning for cost-sensitive classification,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1915–1924.
- [60] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” *Citeseer*, 2009.
- [61] H. Mozannar, H. Lang, D. Wei, P. Sattigeri, S. Das, and D. Sontag, “Who should predict? exact algorithms for learning to defer to humans,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2023, pp. 10 520–10 545.
- [62] R. Verma and E. Nalisnick, “Calibrated learning to defer with one-vs-all classifiers,” *arXiv preprint arXiv:2202.03673*, 2022.

- [63] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [64] A. Kumar, A. Raghunathan, R. M. Jones, T. Ma, and P. Liang, “Fine-tuning can distort pretrained features and underperform out-of-distribution,” in *International Conference on Learning Representations*, 2022.
- [65] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2022. URL: <https://www.gurobi.com>.
- [66] P. Long and R. Servedio, “Consistency versus realizable h-consistency for multiclass classification,” in *International Conference on Machine Learning*, PMLR, 2013, pp. 801–809.
- [67] A. De, P. Koley, N. Ganguly, and M. Gomez-Rodriguez, “Regression under human assistance,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 2611–2620.
- [68] B. Ustun and C. Rudin, “Supersparse linear integer models for optimized medical scoring systems,” *Machine Learning*, vol. 102, no. 3, pp. 349–391, 2016.
- [69] T. Nguyen and S. Sanner, “Algorithms for direct 0–1 loss optimization in binary classification,” in *International Conference on Machine Learning*, PMLR, 2013, pp. 1085–1093.
- [70] S. Khot and R. Saket, “On the hardness of learning intersections of two halfspaces,” *Journal of Computer and System Sciences*, vol. 77, no. 1, pp. 129–141, 2011.
- [71] N. Razavian, S. Blecker, A. M. Schmidt, A. Smith-McLallen, S. Nigam, and D. Sontag, “Population-level prediction of type 2 diabetes from claims data and analysis of risk factors,” *Big Data*, vol. 3, no. 4, pp. 277–287, 2015.
- [72] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [73] Y. Lin, “Support vector machines and the bayes rule in classification,” *Data Mining and Knowledge Discovery*, vol. 6, no. 3, pp. 259–275, 2002.
- [74] M. Zhang and S. Agarwal, “Bayes consistency vs. h-consistency: The interplay between surrogate loss functions and the scoring function class,” *Advances in neural information processing systems*, vol. 33, pp. 16 927–16 936, 2020.
- [75] R. M. Battleday, J. C. Peterson, and T. L. Griffiths, “Capturing human categorization of natural images by combining deep networks and cognitive models,” *Nature communications*, vol. 11, no. 1, pp. 1–14, 2020.

- [76] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.
- [77] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [78] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” *arXiv preprint arXiv:1908.10084*, 2019.
- [79] J. Dressel and H. Farid, “The accuracy, fairness, and limits of predicting recidivism,” *Science advances*, vol. 4, no. 1, eaao5580, 2018.
- [80] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. Summers, “Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases,” in *IEEE CVPR*, vol. 7, 2017.
- [81] A. Majkowska, S. Mittal, D. F. Steiner, J. J. Reicher, S. M. McKinney, G. E. Duggan, K. Eswaran, P.-H. Cameron Chen, Y. Liu, S. R. Kalidindi, *et al.*, “Chest radiograph interpretation with deep learning models: Assessment with radiologist-adjudicated reference standards and population-adjusted evaluation,” *Radiology*, vol. 294, no. 2, pp. 421–431, 2020.
- [82] H. Mozannar, A. Satyanarayan, and D. Sontag, “Teaching humans when to defer to a classifier via exemplars,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 5323–5331.
- [83] D. Link, B. Hellingrath, and J. Ling, “A human-is-the-loop approach for semi-automated content moderation,” in *ISCRAM*, 2016.
- [84] S. J. Shaikh and I. Cruz, ““alexa, do you know anything?” the impact of an intelligent assistant on team interactions and creative performance under time scarcity,” *arXiv preprint arXiv:1912.12914*, 2019.
- [85] A. M. Bornstein, M. W. Khaw, D. Shohamy, and N. D. Daw, “Reminders of past choices bias decisions for reward in humans,” *Nature Communications*, vol. 8, no. 1, pp. 1–9, 2017.
- [86] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, “Hotpotqa: A dataset for diverse, explainable multi-hop question answering,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 2369–2380.
- [87] M. Raghu, K. Blumer, R. Sayres, Z. Obermeyer, B. Kleinberg, S. Mullainathan, and J. Kleinberg, “Direct uncertainty prediction for medical second opinions,” in *International Conference on Machine Learning*, 2019, pp. 5281–5290.

- [88] V. Lai and C. Tan, “On human predictions with explanations and predictions of machine learning models: A case study on deception detection,” in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019, pp. 29–38.
- [89] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. Gershman, and F. Doshi-Velez, “An evaluation of the human-interpretability of explanation,” *arXiv preprint arXiv:1902.00006*, 2019.
- [90] A. Smith-Renner, R. Fan, M. Birchfield, T. Wu, J. Boyd-Graber, D. S. Weld, and L. Findlater, “No explainability without accountability: An empirical study of explanations and feedback in interactive ml,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
- [91] P. Hase and M. Bansal, “Evaluating explainable ai: Which algorithmic explanations help users predict model behavior?” *arXiv preprint arXiv:2005.01831*, 2020.
- [92] Y. Zhang, Q. V. Liao, and R. K. Bellamy, “Effect of confidence and explanation on accuracy and trust calibration in ai-assisted decision making,” in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020, pp. 295–305.
- [93] R. Kocielnik, S. Amershi, and P. N. Bennett, “Will you accept an imperfect ai? exploring designs for adjusting end-user expectations of ai systems,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–14.
- [94] H. Suresh, N. Lao, and I. Liccardi, “Misplaced trust: Measuring the interference of machine learning in human decision-making,” *arXiv preprint arXiv:2005.10960*, 2020.
- [95] H. Suresh, K. M. Lewis, J. V. Guttag, and A. Satyanarayan, “Intuitively assessing ml model reliability through example-based explanations and editing model inputs,” *arXiv preprint arXiv:2102.08540*, 2021.
- [96] J. Wortman Vaughan and H. Wallach, “A human-centered agenda for intelligible machine learning,” This is a draft version of a chapter in a book to be published in the 2020 - 21 timeframe., May 2021. URL: <https://www.microsoft.com/en-us/research/publication/a-human-centered-agenda-for-intelligible-machine-learning/>.
- [97] A. V. Gonzalez, G. Bansal, A. Fan, R. Jia, Y. Mehdad, and S. Iyer, “Human evaluation of spoken vs. visual explanations for open-domain qa,” *arXiv preprint arXiv:2012.15075*, 2020.
- [98] H. Kaur, H. Nori, S. Jenkins, R. Caruana, H. Wallach, and J. Wortman Vaughan, “Interpreting interpretability: Understanding data scientists’ use of interpretability tools for machine learning,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–14.

- [99] T. DeVries and G. W. Taylor, “Learning confidence for out-of-distribution detection in neural networks,” *arXiv preprint arXiv:1802.04865*, 2018.
- [100] S. Amershi, D. Weld, M. Vorvoreanu, A. Fournery, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, *et al.*, “Guidelines for human-ai interaction,” in *Proceedings of the 2019 chi conference on human factors in computing systems*, 2019, pp. 1–13.
- [101] A. Chandrasekaran, V. Prabhu, D. Yadav, P. Chattopadhyay, and D. Parikh, “Do explanations make vqa models more predictable to a human?” *arXiv preprint arXiv:1810.12366*, 2018.
- [102] S. Feng and J. Boyd-Graber, “What can ai do for me? evaluating machine learning interpretations in cooperative play,” in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, 2019, pp. 229–239.
- [103] C. J. Cai, S. Winter, D. Steiner, L. Wilcox, and M. Terry, ““ hello ai”: Uncovering the onboarding needs of medical practitioners for human-ai collaborative decision-making,” *Proceedings of the ACM on Human-computer Interaction*, vol. 3, no. CSCW, pp. 1–24, 2019.
- [104] M. Yin, J. Wortman Vaughan, and H. Wallach, “Understanding the effect of accuracy on trust in machine learning models,” in *Proceedings of the 2019 chi conference on human factors in computing systems*, 2019, pp. 1–12.
- [105] G. Bansal, B. Nushi, E. Kamar, E. Horvitz, and D. S. Weld, “Is the most accurate ai the best teammate? optimizing ai for teamwork,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 11 405–11 414.
- [106] A. Singla, I. Bogunovic, G. Bartok, A. Karbasi, and A. Krause, “Near-optimally teaching the crowd to classify,” in *International Conference on Machine Learning*, 2014, pp. 154–162.
- [107] A. Kumar, H. Zhang, A. Singla, and Y. Chen, “The teaching dimension of kernel perceptron,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 2071–2079.
- [108] A. Hunziker, Y. Chen, O. Mac Aodha, M. G. Rodriguez, A. Krause, P. Perona, Y. Yue, and A. Singla, “Teaching multiple concepts to a forgetful learner,” *arXiv preprint arXiv:1805.08322*, 2018.
- [109] S. Dasgupta, D. Hsu, S. Poulis, and X. Zhu, “Teaching a black-box learner,” in *International Conference on Machine Learning*, 2019, pp. 1547–1555.
- [110] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

- [111] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, “Automated curriculum learning for neural networks,” in *international conference on machine learning*, PMLR, 2017, pp. 1311–1320.
- [112] S. Ruan, J. He, R. Ying, J. Burkle, D. Hakim, A. Wang, Y. Yin, L. Zhou, Q. Xu, A. AbuHashem, *et al.*, “Supporting children’s math learning with feedback-augmented narrative technology,” in *Proceedings of the Interaction Design and Children Conference*, 2020, pp. 567–580.
- [113] S. Doroudi, E. Kamar, and E. Brunskill, “Not everyone writes good examples but good examples can come from anywhere,” in *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, vol. 7, 2019, pp. 12–21.
- [114] M. H. Lee, J. Runde, W. Jibril, Z. Wang, and E. Brunskill, “Learning the features used to decide how to teach,” in *Proceedings of the Second (2015) ACM Conference on Learning@Scale*, 2015, pp. 421–424.
- [115] G. Giguère and B. C. Love, “Limits in decision making arise from limits in memory retrieval,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 19, pp. 7613–7618, 2013.
- [116] J. J. Richler and T. J. Palmeri, “Visual category learning,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 5, no. 1, pp. 75–94, 2014.
- [117] Y.-S. Kim, L. A. Walls, P. Krafft, and J. Hullman, “A bayesian cognition approach to improve data visualization,” in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–14.
- [118] D. D. Bourgin, J. C. Peterson, D. Reichman, S. J. Russell, and T. L. Griffiths, “Cognitive model priors for predicting human decisions,” in *International conference on machine learning*, PMLR, 2019, pp. 5133–5141.
- [119] C. Ilvento, “Metric learning for individual fairness,” *arXiv preprint arXiv:1906.00250*, 2019.
- [120] G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang, “An efficient sparse metric learning in high-dimensional space via l₁-penalized log-determinant regularization,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 841–848.
- [121] M. Tu, K. Huang, G. Wang, J. Huang, X. He, and B. Zhou, “Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 9073–9080.
- [122] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “Squad: 100,000+ questions for machine comprehension of text,” *arXiv preprint arXiv:1606.05250*, 2016.

- [123] Z. Bucinca, P. Lin, K. Z. Gajos, and E. L. Glassman, “Proxy tasks and subjective measures can be misleading in evaluating explainable ai systems,” in *Proceedings of the 25th International Conference on Intelligent User Interfaces*, 2020, pp. 454–464.
- [124] H. Mozannar, J. J. Lee, D. Wei, P. Sattigeri, S. Das, and D. Sontag, “Effective human-ai teams via learned natural language rules and onboarding,” in *Advances in Neural Information Processing Systems*, 2023.
- [125] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, *et al.*, “Learning transferable visual models from natural language supervision,” in *International conference on machine learning*, PMLR, 2021, pp. 8748–8763.
- [126] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Nov. 2019. URL: <https://arxiv.org/abs/1908.10084>.
- [127] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [128] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2174–2182.
- [129] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, “Measuring massive multitask language understanding,” *arXiv preprint arXiv:2009.03300*, 2020.
- [130] G. Bansal, T. Wu, J. Zhu, R. Fok, B. Nushi, E. Kamar, M. T. Ribeiro, and D. S. Weld, “Does the whole exceed its parts? the effect of ai explanations on complementary team performance,” *arXiv preprint arXiv:2006.14779*, 2020.
- [131] F. Sperrle, M. El-Assady, G. Guo, R. Borgo, D. H. Chau, A. Endert, and D. Keim, “A survey of human-centered evaluations in human-centered machine learning,” in *Computer Graphics Forum*, Wiley Online Library, vol. 40, 2021, pp. 543–568.
- [132] V. Lai, S. Carton, R. Bhatnagar, Q. V. Liao, Y. Zhang, and C. Tan, “Human-ai collaboration via conditional delegation: A case study of content moderation,” in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–18.

- [133] S. Ma, Y. Lei, X. Wang, C. Zheng, C. Shi, M. Yin, and X. Ma, “Who should i trust: Ai or myself? leveraging human and ai correctness likelihood to promote appropriate trust in ai-assisted decision-making,” *arXiv preprint arXiv:2301.05809*, 2023.
- [134] Á. A. Cabrera, A. Perer, and J. I. Hong, “Improving human-ai collaboration with descriptions of ai behavior,” *arXiv preprint arXiv:2301.06937*, 2023.
- [135] A. Kawakami, L. Guerdan, Y. Cheng, K. Glazko, M. Lee, S. Carter, N. Arechiga, H. Zhu, and K. Holstein, “Training towards critical use: Learning to situate ai predictions relative to human knowledge,” in *Proceedings of The ACM Collective Intelligence Conference*, 2023, pp. 63–78.
- [136] S. Eyuboglu, M. Varma, K. K. Saab, J.-B. Delbrouck, C. Lee-Messer, J. Dunnmon, J. Zou, and C. Re, “Domino: Discovering systematic errors with cross-modal embeddings,” in *International Conference on Learning Representations*, 2021.
- [137] H. Bharadhwaj, D.-A. Huang, C. Xiao, A. Anandkumar, and A. Garg, “Auditing ai models for verified deployment under semantic specifications,” *arXiv preprint arXiv:2109.12456*, 2021.
- [138] M. T. Ribeiro and S. Lundberg, “Adaptive testing and debugging of nlp models,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 3253–3267.
- [139] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld, “Errudite: Scalable, reproducible, and testable error analysis,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 747–763.
- [140] Y. Zhang, J. Z. HaoChen, S.-C. Huang, K.-C. Wang, J. Zou, and S. Yeung, “Drml: Diagnosing and rectifying vision models using language,” in *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*.
- [141] O. Wiles, I. Albuquerque, and S. Gowal, “Discovering bugs in vision models using off-the-shelf image generation and captioning,” *arXiv preprint arXiv:2208.08831*, 2022.
- [142] G. d’Eon, J. d’Eon, J. R. Wright, and K. Leyton-Brown, “The spotlight: A general method for discovering systematic errors in deep learning models,” in *2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022, pp. 1962–1981.
- [143] N. Rajani, W. Liang, L. Chen, M. Mitchell, and J. Zou, “Seal: Interactive tool for systematic error analysis and labeling,” *arXiv preprint arXiv:2210.05839*, 2022.
- [144] S. Jain, H. Lawrence, A. Moitra, and A. Madry, “Distilling model failures as directions in latent space,” *arXiv preprint arXiv:2206.14754*, 2022.

- [145] J. K. Lee, Y. Bu, D. Rajan, P. Sattigeri, R. Panda, S. Das, and G. W. Wornell, “Fair selective classification via sufficiency,” in *International conference on machine learning*, 2021, pp. 6076–6086.
- [146] A. Shah, Y. Bu, J. K. Lee, S. Das, R. Panda, P. Sattigeri, and G. W. Wornell, “Selective regression under fairness criteria,” in *International Conference on Machine Learning*, 2022, pp. 19 598–19 615.
- [147] U. Bhatt, V. Chen, K. M. Collins, P. Kamalaruban, E. Kallina, A. Weller, and A. Talwalkar, “Learning personalized decision support policies,” *arXiv preprint arXiv:2304.06701*, 2023.
- [148] S. R. Bowman, J. Hyun, E. Perez, E. Chen, C. Pettit, S. Heiner, K. Lukosuite, A. Askill, A. Jones, A. Chen, *et al.*, “Measuring progress on scalable oversight for large language models,” *arXiv preprint arXiv:2211.03540*, 2022.
- [149] P. Zhang, “Taking advice from chatgpt,” *arXiv preprint arXiv:2305.11888*, 2023.
- [150] S. Tong, E. Jones, and J. Steinhardt, “Mass-producing failures of multimodal systems with language models,” *arXiv preprint arXiv:2306.12105*, 2023.
- [151] B. Joshi, Z. Liu, S. Ramnath, A. Chan, Z. Tong, S. Nie, Q. Wang, Y. Choi, and X. Ren, “Are machine rationales (not) useful to humans? measuring and improving human utility of free-text rationales,” *arXiv preprint arXiv:2305.07095*, 2023.
- [152] V. Lai, C. Chen, Q. V. Liao, A. Smith-Renner, and C. Tan, “Towards a science of human-ai decision making: A survey of empirical studies,” *arXiv preprint arXiv:2112.11471*, 2021.
- [153] J. Kasai, K. Sakaguchi, L. Dunagan, J. Morrison, R. L. Bras, Y. Choi, and N. A. Smith, “Transparent human evaluation for image captioning,” *arXiv preprint arXiv:2111.08940*, 2021.
- [154] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, “Model cards for model reporting,” in *Proceedings of the conference on fairness, accountability, and transparency*, 2019, pp. 220–229.
- [155] L. Portnoff, E. Gustafson, J. Rollinson, and K. Bicknell, “Methods for language learning assessment at scale: Duolingo case study.,” *International Educational Data Mining Society*, 2021.
- [156] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.

- [157] C. Potts, Z. Wu, A. Geiger, and D. Kiela, “Dynasent: A dynamic benchmark for sentiment analysis,” *arXiv preprint arXiv:2012.15349*, 2020.
- [158] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.
- [159] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, *et al.*, “Scaling instruction-finetuned language models,” *arXiv preprint arXiv:2210.11416*, 2022.
- [160] F. Barbieri, J. Camacho-Collados, L. Neves, and L. Espinosa-Anke, “Tweeteval: Unified benchmark and comparative evaluation for tweet classification,” *arXiv preprint arXiv:2010.12421*, 2020.
- [161] J. M. Santos and M. Embrechts, “On the use of the adjusted rand index as a metric for evaluating supervised classification,” in *Artificial Neural Networks–ICANN 2009: 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part II 19*, Springer, 2009, pp. 175–184.
- [162] E. B. Fowlkes and C. L. Mallows, “A method for comparing two hierarchical clusterings,” *Journal of the American statistical association*, vol. 78, no. 383, pp. 553–569, 1983.
- [163] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, “Can a suit of armor conduct electricity? a new dataset for open book question answering,” *arXiv preprint arXiv:1809.02789*, 2018.
- [164] *Prolific*, <https://www.prolific.co/>, Accessed on May 17, 2023.
- [165] Y. Benjamini and Y. Hochberg, “Controlling the false discovery rate: A practical and powerful approach to multiple testing,” *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.
- [166] H. Mozannar, G. Bansal, A. Founrey, and E. Horvitz, “Reading between the lines: Modeling user behavior and costs in ai-assisted programming,” *arXiv preprint arXiv:2210.14306*, 2022.
- [167] Amazon, *ML-powered coding companion – amazon codewhisperer*, 2022. URL: <https://aws.amazon.com/codewhisperer/>.
- [168] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, *et al.*, “Competition-level code generation with alphacode,” *arXiv preprint arXiv:2203.07814*, 2022.

- [169] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021.
- [170] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, “Productivity assessment of neural code completion,” in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 2022, pp. 21–29.
- [171] P. Vaithilingam, T. Zhang, and E. L. Glassman, “Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models,” in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.
- [172] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, “The impact of ai on developer productivity: Evidence from github copilot,” *arXiv preprint arXiv:2302.06590*, 2023.
- [173] E. Jiang, E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai, and M. Terry, “Discovering the syntax and strategies of natural language programming with generative language models,” in *CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–19.
- [174] E. Kalliamvakou, *Research: Quantifying github copilot’s impact on developer productivity and happiness*, Sep. 2022. URL: <https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>.
- [175] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [176] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, *et al.*, “Measuring coding challenge competence with apps,” *arXiv preprint arXiv:2105.09938*, 2021.
- [177] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, “Out of the bleu: How should we assess quality of the code generation models?” *arXiv preprint arXiv:2208.03133*, 2022.
- [178] V. J. Hellendoorn, S. Proksch, H. C. Gall, and A. Bacchelli, “When code completion fails: A case study on real-world completions,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 960–970.
- [179] A. Sarkar, A. D. Gordon, C. Negreanu, C. Poelitz, S. S. Ragavan, and B. Zorn, “What is it like to program with artificial intelligence?” *arXiv preprint arXiv:2208.06213*, 2022.
- [180] J. D. Weisz, M. Muller, S. Houde, J. Richards, S. I. Ross, F. Martinez, M. Agarwal, and K. Talamadupula, “Perfection not required? human-ai partnerships in code translation,” in *26th International Conference on Intelligent User Interfaces*, 2021, pp. 402–412.

- [181] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, “The space of developer productivity: There’s more to it than you think.,” *Queue*, vol. 19, no. 1, pp. 20–48, 2021.
- [182] J. T. Liang, C. Yang, and B. A. Myers, “Understanding the usability of ai programming assistants,” *arXiv preprint arXiv:2303.17125*, vol. 1, no. 1, pp. 1–2, 2023.
- [183] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos, ““ it’s weird that it knows what i want”: Usability and interactions with copilot for novice programmers,” *arXiv preprint arXiv:2304.02491*, vol. 1, no. 1, pp. 1–2, 2023.
- [184] M. T. Tabachnyk and S. Nikolov, *ML-enhanced code completion improves developer productivity*, Jul. 2022. URL: <https://ai.googleblog.com/2022/07/ml-enhanced-code-completion-improves>.
- [185] T. Wu, K. Koedinger, *et al.*, “Is ai the better programming partner? human-human pair programming vs. human-ai pair programming,” *arXiv preprint arXiv:2306.05153*, vol. 1, no. 1, pp. 1–2, 2023.
- [186] R. E. Brooks, “Studying programmer behavior experimentally: The problems of proper methodology,” *Communications of the ACM*, vol. 23, no. 4, pp. 207–213, 1980.
- [187] R. Brooks, “Towards a theory of the cognitive processes in computer programming,” *International Journal of Man-Machine Studies*, vol. 9, no. 6, pp. 737–751, 1977.
- [188] B. A. Sheil, “The psychological study of programming,” *ACM Computing Surveys (CSUR)*, vol. 13, no. 1, pp. 101–120, 1981.
- [189] H. Lieberman and C. Fry, “Bridging the gulf between code and behavior in programming,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, .: ., 1995, pp. 480–486.
- [190] Z. Velart and P. Šaloun, “User behavior patterns in the course of programming in c++,” in *Proceedings of the joint international workshop on Adaptivity, personalization & the semantic web*, .: ., 2006, pp. 41–44.
- [191] A. Ju and A. Fox, “Teamscope: Measuring software engineering processes with team-work telemetry,” in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, .: ., 2018, pp. 123–128.
- [192] N. Peitek, J. Siegmund, and S. Apel, “What drives the reading order of programmers? an eye tracking study,” in *Proceedings of the 28th International Conference on Program Comprehension*, .: ., 2020, pp. 342–353.

- [193] U. Obaidallah, M. Al Haek, and P. C.-H. Cheng, “A survey on the usage of eye-tracking in computer programming,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–58, 2018.
- [194] S. Barke, M. B. James, and N. Polikarpova, “Grounded copilot: How programmers interact with code-generating models,” *arXiv preprint arXiv:2206.15000*, 2022.
- [195] Z. Sun, X. Du, F. Song, S. Wang, M. Ni, and L. Li, “Don’t complete it! preventing unhelpful code completion for productive and sustainable neural code completion systems,” in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, .: ., 2023, pp. 324–325.
- [196] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, “The space of developer productivity,” *Communications of the ACM*, vol. 64, no. 6, pp. 46–53, 2021.
- [197] S. K. Card, T. P. Moran, and A. Newell, “The keystroke-level model for user performance time with interactive systems,” *Communications of the ACM*, vol. 23, no. 7, pp. 396–410, 1980.
- [198] B. E. John and D. E. Kieras, “The goms family of user interface analysis techniques: Comparison and contrast,” *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 3, no. 4, pp. 320–351, 1996.
- [199] L. Ekroot and T. M. Cover, “The entropy of markov trajectories,” *IEEE Transactions on Information Theory*, vol. 39, no. 4, pp. 1418–1421, 1993.
- [200] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, “Asleep at the keyboard? assessing the security of github copilot’s code contributions,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 754–768.
- [201] O. Asare, M. Nagappan, and N. Asokan, “Is github’s copilot as bad as humans at introducing vulnerabilities in code?” *arXiv preprint arXiv:2204.04741*, 2022.
- [202] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, “Can openai codex and other large language models help us fix security bugs?” *arXiv preprint arXiv:2112.02125*, 2021.
- [203] H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, “When to show a suggestion? integrating human feedback in ai-assisted programming,” *arXiv preprint arXiv:2306.04930*, 2023.
- [204] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

- [205] S. Zhao, *GitHub Copilot now has a better AI model and new capabilities*, <https://github.blog/2023-02-14-github-copilot-now-has-a-better-ai-model-and-new-capabilities/>, Feb. 2023.
- [206] Z. Sun, X. Du, F. Song, S. Wang, M. Ni, and L. Li, “Learning to prevent profitless neural code completion,” *arXiv preprint arXiv:2209.05948*, 2022.
- [207] W. B. Knox and P. Stone, “Tamer: Training an agent manually via evaluative reinforcement,” in *2008 7th IEEE international conference on development and learning*, IEEE, 2008, pp. 292–297.
- [208] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, “Interactive learning from policy-dependent human feedback,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 2285–2294.
- [209] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, “Fine-tuning language models from human preferences,” *arXiv preprint arXiv:1909.08593*, 2019.
- [210] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, *et al.*, “Training a helpful and harmless assistant with reinforcement learning from human feedback,” *arXiv preprint arXiv:2204.05862*, 2022.
- [211] OpenAI, *Chatgpt: Optimizing language models for dialogue*, 2022. URL: <https://openai.com/blog/chatgpt/>.
- [212] E. Horvitz, “Principles of mixed-initiative user interfaces,” in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1999, pp. 159–166.
- [213] M. Csikszentmihalyi and R. Larson, *Flow and the foundations of positive psychology*. Springer, 2014, vol. 10.
- [214] B. P. Bailey, J. A. Konstan, and J. V. Carlis, “The effects of interruptions on task performance, annoyance, and anxiety in the user interface,” in *Interact*, vol. 1, 2001, pp. 593–601.
- [215] E. Cutrell, M. Czerwinski, and E. Horvitz, “Notification, disruption, and memory: Effects of messaging interruptions on memory and performance,” in *IFIP TC13 International Conference on Human-Computer Interaction*, 2001.
- [216] E. Horvitz and J. Apacible, “Learning and reasoning about interruption,” in *Proceedings of the 5th International Conference on Multimodal Interfaces*, ser. ICMI ’03, Vancouver, British Columbia, Canada, 2003, pp. 20–27.
- [217] E. Horvitz, A. Jacobs, and D. Hovel, “Attention-sensitive alerting,” in *Proceedings of UAI*, Stockholm, Sweden, 1999, pp. 305–313.

- [218] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, *et al.*, “Codebert: A pre-trained model for programming and natural languages,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.
- [219] E. Kalliamvakou, *Tree-sitter parser generator tool*, Jan. 2023. URL: <https://tree-sitter.github.io/tree-sitter/>.
- [220] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, R. Mitchell, I. Cano, T. Zhou, *et al.*, “Xgboost: Extreme gradient boosting,” *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.
- [221] M. P. Naeni, G. Cooper, and M. Hauskrecht, “Obtaining well calibrated probabilities using bayesian binning,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [222] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *arXiv preprint arXiv:2305.18290*, 2023.
- [223] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.
- [224] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.
- [225] M. Ledoux and M. Talagrand, *Probability in Banach Spaces: isoperimetry and processes*. Springer Science & Business Media, 1991, vol. 23.
- [226] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.
- [227] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [228] A. Blum and R. Rivest, “Training a 3-node neural network is np-complete,” *Advances in neural information processing systems*, vol. 1, 1988.
- [229] V. Guruswami and P. Raghavendra, “Hardness of learning halfspaces with noise,” *SIAM Journal on Computing*, vol. 39, no. 2, pp. 742–765, 2009.
- [230] S. Kakade and A. Tewari, *Rademacher composition and linear prediction*, <https://home.ttic.edu/~tewari/lectures/lecture17.pdf>, Feb. 2008.
- [231] A. Levy, M. Agrawal, A. Satyanarayan, and D. Sontag, “Assessing the impact of automated suggestions on decision making: Domain experts mediate model errors but take less initiative,” in *CHI Conference on Human Factors in Computing Systems*, 2021.

- [232] E. Chu, D. Roy, and J. Andreas, “Are visual explanations useful? a case study in model-in-the-loop prediction,” *arXiv preprint arXiv:2007.12248*, 2020.
- [233] Y. Xie, M. Chen, D. Kao, G. Gao, and X. Chen, “Chexplain: Enabling physicians to explore and understand data-driven, ai-enabled medical imaging analysis,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
- [234] P. Tschandl, C. Rinner, Z. Apalla, G. Argenziano, N. Codella, A. Halpern, M. Janda, A. Lallas, C. Longo, J. Malvehy, *et al.*, “Human–computer collaboration for skin cancer recognition,” *Nature Medicine*, vol. 26, no. 8, pp. 1229–1234, 2020.
- [235] B. E. Bejnordi, M. Veta, P. J. Van Diest, B. Van Ginneken, N. Karssemeijer, G. Litjens, J. A. Van Der Laak, M. Hermsen, Q. F. Manson, M. Balkenhol, *et al.*, “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer,” *Jama*, vol. 318, no. 22, pp. 2199–2210, 2017.
- [236] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
- [237] B. Kim, R. Khanna, and O. O. Koyejo, “Examples are not enough, learn to criticize! criticism for interpretability,” in *Advances in neural information processing systems*, 2016, pp. 2280–2288.
- [238] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 1321–1330.
- [239] T. Tohme, K. Vanslette, and K. Youcef-Toumi, “Improving regression uncertainty estimation under statistical change,” *arXiv preprint arXiv:2109.08213*, 2021.
- [240] K. Vanslette, T. Tohme, and K. Youcef-Toumi, “A general model validation and testing tool,” *Reliability Engineering & System Safety*, vol. 195, p. 106684, 2020.
- [241] S. A. Goldman and M. J. Kearns, “On the complexity of teaching,” *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 20–31, 1995.
- [242] Y. Chen, O. Mac Aodha, S. Su, P. Perona, and Y. Yue, “Near-optimal machine teaching via explanatory teaching sets,” in *International Conference on Artificial Intelligence and Statistics*, 2018, pp. 1970–1978.
- [243] R. Devidze, F. Mansouri, L. Haug, Y. Chen, and A. Singla, “Understanding the power and limitations of teaching with imperfect knowledge,” *arXiv preprint arXiv:2003.09712*, 2020.

- [244] G. Gates, “The reduced nearest neighbor rule (corresp.),” *IEEE transactions on information theory*, vol. 18, no. 3, pp. 431–433, 1972.
- [245] F. Angiulli, “Fast condensed nearest neighbor rule,” in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 25–32.
- [246] M. Kusner, S. Tyree, K. Weinberger, and K. Agrawal, “Stochastic neighbor compression,” in *International Conference on Machine Learning*, PMLR, 2014, pp. 622–630.
- [247] K. Zhong, R. Guo, S. Kumar, B. Yan, D. Simcha, and I. Dhillon, “Fast classification with binary prototypes,” in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 1255–1263.
- [248] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain, “Protonn: Compressed and accurate knn for resource-scarce devices,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 1331–1340.
- [249] A. Krause and D. Golovin, “Submodular function maximization.,” *Tractability*, vol. 3, pp. 71–104, 2014.
- [250] R. Řehůřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” English, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, May 2010, pp. 45–50.
- [251] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [252] M. Arnold, R. K. Bellamy, M. Hind, S. Houde, S. Mehta, A. Mojsilović, R. Nair, K. N. Ramamurthy, A. Olteanu, D. Piorkowski, *et al.*, “Factsheets: Increasing trust in ai services through supplier’s declarations of conformity,” *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 6–1, 2019.
- [253] I. Gao, G. Ilharco, S. Lundberg, and M. T. Ribeiro, “Adaptive testing of computer vision models,” *arXiv preprint arXiv:2212.02774*, 2022.
- [254] Y. Ahn, Y.-R. Lin, P. Xu, and Z. Dai, “Escape: Countering systematic errors from machine’s blind spots via interactive visual analysis,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–16.
- [255] B. Y. Idrissi, D. Bouchacourt, R. Balestrieri, I. Evtimov, C. Hazirbas, N. Ballas, P. Vincent, M. Drozdal, D. Lopez-Paz, and M. Ibrahim, “Imagenet-x: Understanding model mistakes with factor of variation annotations,” *arXiv preprint arXiv:2211.01866*, 2022.
- [256] V. Vasudevan, B. Caine, R. Gontijo-Lopes, S. Fridovich-Keil, and R. Roelofs, “When does dough become a bagel? analyzing the remaining mistakes on imagenet,” *arXiv preprint arXiv:2205.04596*, 2022.

- [257] O. Hupert, I. Schwartz, and L. Wolf, “Describing sets of images with textual-pca,” *arXiv preprint arXiv:2210.12112*, 2022.
- [258] Z. M. Malakan, G. M. Hassan, and A. Mian, “Vision transformer based model for describing a set of images as a story,” in *AI 2022: Advances in Artificial Intelligence: 35th Australasian Joint Conference, AI 2022, Perth, WA, Australia, December 5–8, 2022, Proceedings*, Springer, 2022, pp. 15–28.
- [259] R. Zhong, C. Snell, D. Klein, and J. Steinhardt, “Describing differences between text distributions with natural language,” in *International Conference on Machine Learning*, PMLR, 2022, pp. 27 099–27 116.
- [260] Q. Wang, J. Wang, A. B. Chan, S. Huang, H. Xiong, X. Li, and D. Dou, “Neighbours matter: Image captioning with similar images,” in *BMVC*, 2020.
- [261] J. Wang, W. Xu, Q. Wang, and A. B. Chan, “On distinctive image captioning via comparing and reweighting,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [262] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, “Bdd100k: A diverse driving dataset for heterogeneous multitask learning,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.
- [263] G. Luo, L. Cheng, C. Jing, C. Zhao, and G. Song, “A thorough review of models, evaluation metrics, and datasets on image captioning,” *IET Image Processing*, vol. 16, no. 2, pp. 311–332, 2022.
- [264] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” *Advances in neural information processing systems*, vol. 30, 2017.
- [265] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in artificial intelligence*, vol. 2009, 2009.
- [266] J. J. Dudley and P. O. Kristensson, “A review of user interface design for interactive machine learning,” *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 8, no. 2, pp. 1–37, 2018.
- [267] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16, San Francisco, California, USA: ACM, 2016, pp. 785–794, ISBN: 978-1-4503-4232-2. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785). URL: <http://doi.acm.org/10.1145/2939672.2939785>.

- [268] A. Paszke, S. Gross, F. Massa, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.