Natural Language Control for for Visually Interactive Decision Support Tools in Supply Chain Management

by

Willem J, Guter

B.S. Computation and Cognition, MIT, 2023

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTATION AND COGNITION

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

Authored by: Willem J, Guter
Department of Brain and Cognitive Sciences
July 3, 2024

Certified by: Matthias Winkenbach
Principal Research Scientist, MIT CTL, Thesis Supervisor

Accepted by: Mehrdad Jazayeri
BCS Director of Education

Natural Language Control for for Visually Interactive Decision Support Tools in Supply Chain Management

by

Willem J, Guter

Submitted to the Department of Brain and Cognitive Sciences
on July 3, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTATION AND COGNITION

ABSTRACT

Supply chains are complex networks where changing one variable can have unforeseen effects on the entire chain. Interactive supply chain visualizations are useful for understanding these effects, and can lead to decreased cost. However, these interactive visualizations can require technical and domain expertise to operate and understand. A solution for this is natural language interfaces, allowing users to use natural language commands to control the visualization. Additionally, natural language interfaces can be difficult to implement, and require applications specific programming or training. This thesis proposes integrating a pre-trained large language model as the natural language interface. An example application is created using an existing supply chain network visualization application. Various large language models are then evaluated for usability, functionality, and accuracy. We find that a state of the art commercial model is able to practically fulfill the role of a natural language interface, but that open-source large language models are not currently capable of functioning in this way.

Thesis supervisor: Matthias Winkenbach
Title: Principal Research Scientist, MIT CTL

Acknowledgments

This thesis could not have been written without the expert insight and useful feedback of Matthias Winkenbach. I want to thank him for the all the comments and questions that helped shape my research and writing.

Additionally, I would like to thank Connor Makowski, who first brought me in to the CTL, for his constant support and creative ideas. Without our long talks much of what made this thesis possible never could have happened.

I also want to thank Tim Russell and all the other members of the CAVE Lab who I've had the chance to work with over my years here. What you helped to create has been an important part of my work, and the input you've given me is invaluable.

Finally, I want to thank my friends and family for being such a great support system. From listening when I wanted to ramble about my ideas for hours, to accepting when I didn't even want to think about the thesis, you've kept me going throughout this endeavour.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Interactivity in Supply Chain Decision Making

Supply chains are complex networks of suppliers, warehouses, distribution centers, and various transportation modalities all effected by one another. Because of the complexity, slightly changing a single parameter of one part of the network can have significant unanticipated effects on other parts of the network. This can create disconnects across organizations where implementing cost saving measures in one team leads to increased costs in other teams due to higher stored inventory, minimum order quantities, or transportation costs.

Without interactivity, supply chain visualizations are useful only for understanding the current state of a supply chain network, as the effects of any change on the network cannot be intuited from simply visualizing the current or past network states. Interactive supply chain visualizations solve this problem by allowing users to understand the results of changes to the supply chain. By using models of supply chain behavior to predict the results when users change parameters of the network, interactive supply chain visualizations allow for an intuitive understanding of how supply chain modifications effect an entire network. This clear and intuitive understanding helps reduce errors in planning that lead to unforeseen costs.

1.2 Advantages of Natural Language Interfaces

Supply chain data visualization is a challenging task that requires users to have technical and domain expertise. Traditional methods of interacting with logistics data, such as spreadsheets and dashboards, are often limited and cumbersome. Additionally, these forms of interaction can be unintuitive and difficult to learn.

Natural language processing (NLP) can offer a more intuitive and flexible way of interacting with supply chain data, by allowing users to use natural language queries and commands. However, developing an effective natural language interface for supply chain decision making requires overcoming several challenges. These challenges include understanding and parsing the natural language input, and translating that input to user intent. Additionally, the NLP system must understand the capabilities of the system, and how to implement user intent with these capabilities.

1.3 Thesis Objective

The main objective of this research is to integrate natural language control through context learning by a large language model (LLM) with the interactive supply chain visualization applications being developed by the MIT CAVE Lab at the MIT Center for Transportation & Logistics (PI: Dr. Matthias Winkenbach). The performance of different LLMs is then be evaluated in terms of usability, functionality, and accuracy.

The work consists of three main tasks: (1) integrating an LLM with an interactive supply chain visualization application (see Section 3.2), (2) developing and testing natural language queries and commands for logistics data visualization (see Section 5.1, and (3) evaluating the performance of different LLMs for natural language control of logistics data visualization (see Section 5.2).

The remainder of this thesis is structured as follows. In Chapter 2, the current research

in NLP for data visualization and LLMs for NLP interfaces is discussed, and a research gap identified. In Chapter 3, the specifics of the problem setting and requirements are laid out. In Chapter 4, the implementation choices made are explained. In Chapter 5, the criteria for evaluation are given, and the results shown and discussed. In Chapter 6, the findings are summarized and future avenues of research are considered.

Chapter 2

Related Work

2.1 Natural Language Interfaces

Several studies in the literature have explored the use of natural language interfaces for data visualization. For example, Gao et al. [1] proposed DataTone, a system that allows users to create visualizations by typing natural language queries and commands. DataTone uses a hybrid approach that combines statistical natural language processing and rule based visualization generation to parse and execute natural language inputs. DataTone also provides feedback and suggestions to help users refine their queries and commands.

Another example is Narechania et al. [2], who developed NL4DV, a framework that enables developers to easily add natural language capabilities to existing data visualization tools. NL4DV uses a pre-trained n-gram language model to understand natural language inputs and map them to data attributes and operations. NL4DV also generates natural language responses and visualizations based on the user's inputs.

Transformer-based models have been shown to outperform other architectures on natural language processing tasks [3]. One of the more recent studies that uses a modern Transformer-based architecture for natural language interface for data visualization is Luo et al. [4], where the authors presented ncNet, a system that uses neural machine trans-

lation to translate natural language queries into visualization specifications. ncNet uses a Transformer-based sequence-to-sequence model that is trained on a large corpus of natural language and visualization pairs. ncNet also incorporates several visualization-aware optimizations, such as attention-forcing, visualization-aware rendering, and visualization-aware evaluation. ncNet can handle complex and diverse natural language queries and generate high-quality visualizations.

2.2 Large Language Models

In recent years, large language models have been found to be capable of generating working code in a variety of programming languages [5]. These models are Transformer-based [3] neural networks with billions of parameters, trained on huge amounts of data scraped from books and the internet. Even when not trained specifically for writing code, these models are able to generate programs from code comments or documentation [6].

Using reinforcement learning from human feedback, this capability can be enhanced and made to follow written commands rather than documentation or comments [7]. Additionally, The Berkeley Function Calling Leaderboard [8] shows that current LLMs are able to generate JSON strings to interact with other tools without creating tool specific grammars or performing any training.

The LLMs tested in this work are GPT-4o[9], Llama 3 70B [10], Mixtral 8x7B[11], and Qwen 1.5 32B. The model specifics and the reasons they were chosen are described in detail in Section 4.1.

2.3 Research Gap

General purpose vs. application-specific applications. Most of the studies reviewed in Section 2.1 focus on general-purpose data visualization, rather than domain-specific applications such as supply chains. Visualization for supply chain decision making poses unique

challenges, such as dealing with complex and dynamic data sources, supporting multiple levels of granularity and aggregation, and facilitating decision making through interaction. Therefore, there is a need for developing a natural language interface that is tailored for interacting with supply chain data visualizations.

Custom Training vs. Context Learning. Additionally, most existing research in natural language interfaces focuses on building custom systems and training custom models for these interfaces, rather than testing already trained models' ability to interface with existing software through context learning [6]. Training or fine-tuning custom models can incur high costs and carbon footprints, making this route unattainable for many researchers and developers.

Context learning has been used successfully for interacting with software interfaces [12], [13]. However, this has been limited to displaying model text results and differentiating between two categories of text input in custom built programs. With this thesis we aim to fill this gap by examining how already trained models can use context learning to understand both the documentation and the current state of existing software for complex data visualizations.

Chapter 3

Problem Setting

3.1 The CAVE App Development Framework

To examine the performance of LLMs serving as a natural language interface for interactive supply chain visualization, we build on an existing visual interface for interactive supply chain decision support developed at MIT. The so-called CAVE App development framework [14] provides browser based visual interfaces to a supply chain data and decision model, including user modifiable maps, graphs, and tables.

The individual implementations of the CAVE App development framework are referred to as CAVE Apps. CAVE Apps can be thought of as consisting of two sections, the front-end and the back-end. The front-end runs in the users browser displaying data through maps, charts, and tables, as well as accepting user interactions through touch or mouse input. The back-end runs on a server, sending data to front-ends while processing user inputs and propagating them to all front-end clients synchronized with the one that accepted the input.

3.1.1 CAVE API

The two sections of CAVE Apps communicate with each other via the CAVE API. The API state is a JSON object, and both the front-end and back-end dispatch mutations to each

other to keep the state in sync. This JSON object specifies everything about the app - from what pages, maps, and charts are available to each value displayed in a table. The API follows a specification that allows individual implementations to vary in how they interact with and process the data on the back-end while retaining compatibility with the standard front-end.

Documentation

The API specification is documented to support creation of new CAVE Apps. This documentation specifies all keys that are present in the JSON object, as well as the value options for each key. The documentation also specifies how these key value pairs relate to each other, including situations where the keys from one section may be used as the values or parts of the values in another section. The function of each key value pair in the front-end is also specified in the documentation. The documentation is written in text decorated using the markdown language, and is split into a separate page for each of the eight top-level keys present in the JSON object.

Error Checking

There exists a validator for the CAVE API JSON object as part of the separate CAVE Utils package [15]. This is a script that accepts the JSON object as an input, and ensures it fits the API specification described in the documentation. If the object fails to conform to the CAVE API specification, the validator returns a list of the issues detected, in addition to the expected value at that point, or the options for the value as applicable.

3.1.2 Current User Interactions

Currently, users interact with CAVE Apps via touch or mouse input. Additionally, a keyboard may be required for alphanumeric inputs. Users can interact with menus to change the coloring and sizing parameters of options on the map or to change the statistics, groupings,

and type of charts, the data displayed via panes and tables, and more.

3.2 Adding a Language Interface

In order to support interaction via natural language, the natural language interface will connect to the back-end in the same way as the front-end connects. This means that the natural language interface will receive live updates to the current API state, and be able to dispatch mutations that effect all other synchronized front-ends. When connected to the back-end server, the natural language interface will keep up to date with the current API and accept a text-based natural language command, at which point it will process that command using an LLM and dispatch a response to the CAVE App development framework back-end.

3.2.1 Desired Interactions

As the natural language interface is connected to the CAVE API, it should be able to perform any action that is part of the API state. This means not only will it be able to perform most actions that the user could do using touch or mouse input, it will also be able to perform batch actions such as clearing all charts at once. This also means that the natural language interface is capable of adding things that aren't typically supported via user input, such as adding new statistics or new data to the map. However, to limit the scope to interactions that should be familiar to experienced users these interactions weren't considered.

3.2.2 Measuring Success

When working with natural language interfaces, the spread of possible inputs and outputs requires more than a simple binary success or failure result. Therefore, the natural language interface will be tested on a variety of potential user inputs. Success will be measured based on how many of these potential inputs the natural language interfaces is able to respond to with the correct CAVE API mutation. Multiple LLMs will be tested in the natural language

interface. Therefore, even if one LLM is incapable of any correct responses, the natural language interface project could succeed with other LLMs.

Chapter 4

Implementation

The choice of LLMs and their implementation with an example CAVE App is discussed in this chapter[1], while the methods of testing and the results are discussed in Chapter 5.

4.1 Choice of Model Architectures

The LLM used were chosen for their state of the art performance on various benchmarks such as the Berkeley Function-Calling Leaderboard, which evaluates models tool use [8], Chatbot Arena, which evaluates models based on human preferences [16], and CanAiCode, which evaluates models on human written programming questions [17]. Particular attention was payed to model's performance under human evaluation and in programming tasks. Additionally, all models chosen were trained to support a context length of at least 8k tokens in order to ensure that relevant documentation, API information, and user requests could fit into model context. Both models with and without publicly available architectures and weights were selected. Models with publicly available weights were ran on the MIT Supercloud high performance computing platform [18]. These models were also quantized to eight bits (unless otherwise noted) due to hardware constraints.

[1]The complete implementation code and results are available at https://github.com/mrkwanzaa/caveGPT

GPT-4o

GPT-4o [19] is the latest large scale closed source transformer based large language model that is made available by OpenAI via an API. GPT-4 [9] was state of the art for many common-sense and language benchmarks upon its launch, and GPT-4o, which is based on GPT-4, maintains top performance on many benchmarks including ARC [20] a popular common sense question answering benchmark. GPT-4o is a multi-modal LLM that accepts text, image, and sound inputs. However, for this system only the text capabilities were used. OpenAI regularly updates this model, and all tests were performed using the gpt-4o-2024-05-13 version.

Llama 3 70B

Meta's Llama 3 [10] is the latest open transformer based LLM from Meta. This model outperforms state of the art open and proprietary LLMs on various benchmarks including AGIEval [21] and ARC. This model uses grouped query attention [22] to improve computational efficiency over the previous Llama 2 model while also improving performance and supported context length. There are currently two sizes of Llama 3 available, 8B and 70B; the 70B model was used for these tests. Due to hardware constraints, the model was used with weights quantized to five bits. As the task involves following user instructions, the chat fine tuned version of Llama 3 70B was used.

Mixtral 8x7B

Mixtral 8x7B [11] is a sparse mixture of experts model with open weights created by Mistral AI. This architecture expands on the transformer decoder by including eight distinct groups of parameters, with a router network choosing which two groups to use at each layer and token. This architecture enables the model to be faster and less resource heavy than similarly capable models, containing 46.7B total parameters, but only using 12.9B parameters per token. Thus, it processes input and generates output at the same speed as a 12.9B parameter

model. Mixtral 8x7B is a top performing open model on the ARC benchmark, and performs well on code generation benchmarks such as HumanEval. As the task involved following user instructions, the instruction fine-tuned version of Mixtral 8x7B, Mixtral 8x7B-Instruct was used.

Qwen 1.5

Qwen 1.5 [23] is another large scale transformer based large language model with publicly available weights. The model contains 32B parameters and is trained in both English and Chinese. Qwen 1.5 72B outperforms Mixtral on the LLM-judged MT-Bench and Alpaca-Eval benchmarks, as well as the T-Eval agent tool use benchmark. Due to hardware limitations, a smaller version of the model was used with only 32B weights. Additionally, smaller versions of Qwen 1.5 with 14B and 7B weights were tested to investigate how well performance scales with model size.

4.2 Adding Documentation to Model Context

Documentation for the CAVE API is inserted into the LLM context in order to allow it to use context learning to understand the API specification, as well as the capabilities of the CAVE App.

4.2.1 Documentation Selection

The CAVE API documentation is generated by pdoc, an automatic documentation generation tool [24], and includes individual markdown pages for each top level key as well as an index page that explains each top level key.

```
System: "Using the given documentation delimited by triple quotes, respond to user requests
with nothing but the name of the top-level key that must be edited in order to fufill the
request surrounded by single quotes"
User:" {documentation}
request: {request}"
```

Figure 4.1: The prompt instructing models to select the correct top-level key when there are no errors. The exact implementation for the system and user labels vary by model. All prompts used can be found in Appendix A.

Content Aware Document Selection

As the complete CAVE API documentation is too large to reasonably fit in the input context of the LLMs used, only part of the documentation was given to the model for each request. Specifically, the model was only given a single page of the documentation at a time. The page to be shown at each step was decided by the model via a separate initial prompt containing instructions, the user request, and the index page of the documentation. Additionally, if the system had detected an error in the API and was attempting a correction, the prompt contained the error text and instructions to avoid causing it. In response to the prompt, the model being tested return the top level key that would be used for the next step of API modification.

Retrieval Augmented Generation Shortcomings

Originally, Retrieval Augmented Generation(RAG) using a vector database was tested for selecting the documentation due to its common use in selecting data to be included in LLM context based on the given prompt[25]. This form of RAG involves using an encoder model to encode each page of the documentation into an embedding vector, and then encoding the user request with the same model and finding the closest documentation page in the embedding space. However, in initial testing the vector database failed to select the correct documentation above chance, and thus wasn't used for this task. As this has been shown to work in other scenarios, it is likely that the technical language or markdown typesetting used in the CAVE API documentation were the cause of these issues.

4.2.2 Documentation Modifications

In order to increase model comprehension of the documentation, some sections were removed or modified to increase clarity. Specifically, sections and headings related to the python docstring origination of the documentation such as class names and headings were removed. Additionally, Top level keys that aren't user modifiable in the app were removed from the index page to ensure these weren't accidentally selected. Finally, a few words of additional information were added about each top level key in the index page.

4.3 Adding API State to Model Context

It's important that the LLM be able to access the current API state in its context, in order for it to properly respond to queries that require knowledge of the current state of the application, such as those relating to what is currently displayed on the screen. However, the entire API state is too large to be included in the context for most models, with a size on the scale of hundreds of thousands of tokens for most tokenizers.

To solve this problem, the model first chooses a top level key that matches the user request as described in section 4.2.1. The section of the API corresponding to the selected top level key is then included in the context of the model generating the path to modify. Using this approach, the model is able to remove a large portion of the API using a smaller context (and therefore faster) prompt. The remaining sections of the API were able to fit within eight thousand tokens in testing, and should be under 100 thousand tokens for any CAVE App able to run on consumer hardware.

4.4 Model Response Handling

After processing the CAVE API documentation and state, in addition to the user request, the LLM returns the change(s) necessary for the user request to be completed.

4.4.1 JSON Filtering

The model is prompted to generate path(s) and value(s) to modify in the API, using a specific JSON format to ensure the system understands the request. Some models, such as GPT-4, are able to be set to return only JSON formatted text, resulting in a model response that can always be parsed by the system. For other models however, generating only JSON formatted text appeared to be difficult, and extraneous explanations were often included with the JSON text. To ensure these responses are still able to be parsed, the system parses the text for { } brackets not contained by other brackets. Only the text within these brackets is parsed as part of the JSON response, with each set of outer brackets parsed separately to ensure that one malformed expression won't break a correct response.

4.4.2 Error Handling

For responses requiring information about multiple top level keys, such as changing the statistic displayed on charts, the system may need information that extends beyond what is available in the provided API state. In these situations, the prompts are first ran naively without any additional information. The modified API state is then ran through the validation system, which checks all parts of the API. If the model generated something incorrectly, such as an incorrect statistic name, the API validator will detect this and return a warning, along with a list of acceptable options for the statistic name. If the validator returns a warning, the model is prompted to regenerate the top level key and API modification, this time with a warning to avoid the included error message.

This system has the advantage of also helping reduce errors for requests that don't need additional information, as when the model returns any response that doesn't fit within the CAVE API specifications, the validator detects this and returns a warning. However, in some situations the model is unable to properly generate an API modification despite having access to the error message. To prevent this from causing the system to enter an infinite loop the

model is only allowed to regenerate the response once.

Chapter 5

Evaluation and Results

The methods for evaluation, and the results of the evaluations are discussed in this chapter. The evaluation uses LLMs as stand-ins for non-expert users to allow for automated testing of different models as the NLP interface.

5.1 Evaluation Methods

The LLMs were tested on whether they could perform 15 base tasks. To simulate that non-expert users would phrase these tasks in very different ways, each LLM was used to rephrase every task ten different ways, giving a total of 40 phrasings for each of the fifteen tasks.

5.1.1 Base Tasks

Each LLM integrated with the natural language interface was tested independently. The models were tested on fifteen basic tasks within the app. Tasks included both actions that a user could complete in a single touch, and those that would require multiple touches to achieve. Success for each task was determined by check a specific value or values are updated to complete the command. This was chosen as for some tasks the model could complete the task in different ways; for example setting a value to nil or an empty dictionary or array can

```
Restate the following command 10 times by rephrasing it using simpler language,
writing the rephrasings as a json list of strings: {command}

"Color the map based on the number of learners in each country"
"Shade the geographical map according to the learners from each country"
"Apply colors to the geographic illustration representing their respective learners"
```

Figure 5.1: The prompt used to generate the task rephrasings, followed by the first three rephrasings returned by GPT-4-Turbo for the first task.

sometimes achieve the same result.

The base tasks are listed with a brief description of how they would be implemented by a model or API programmer in Appendix B.

5.1.2 Model Generated Task Variations

In order to simulate non-expert users, all LLMs being tested were prompted to generate ten rephrasings of each natural language task. As the LLMs did not have direct access to the CAVE API documentation or state for this task, this rephrasing ensured that API specific terms and concepts would be avoided. Each LLM was tested on all rephrasings, and the performance on self-generated prompts compared to those generated by other models is discussed below. To ensure coherence and accuracy, only the 32B parameter version of the Qwen 1.5 model was used for rephrasing, and the prompt was modified to specify the use of English. Additionally, as GPT-4o hadn't been released at the time of initial task rephrasing, the earlier GPT-4-Turbo model was used for the rephrasing.

5.1.3 The pass@k Metric

The pass@k metric is an evaluation metric used to evaluate code generation models [5]. The metric represents the percentage of test cases passed in a particular test set the top results returned from the LLM. If a single result passes the test case, it is counted as a pass. K is defined as the number of results considered. For example, pass@1 only runs the models top

result for each test. Standard k values for the pass@k metric are 1, 10, and 100. However, due to hardware constraints, for this project k values of 1, 5, and 10 will be tested.

5.2 Results

The results for all four models on each of the 15 tasks can be seen in Table 5.1. There was a wide range in model performance, though all models were able to complete at least one task. As shown in Figure 5.2, varying the k value for the pass@k metric had a smaller impact than anticipated.

Table 5.1: Individual model pass@k for k=1,5,10

Task #	GPT-4o			Llama 3 70B			Mixtral 8x7B			Qwen 1.5 32B		
	k=1	k=5	k=10	k=1	k=5	k=10	k=1	k=5	k=10	k=1	k=5	k=10
1	72.5	87.5	90.0	0.0	0.0	0.0	10.0	17.5	17.5	15.0	17.5	20.0
2	77.5	90.0	92.5	85.0	85.0	85.0	77.5	82.5	85.0	7.5	12.5	12.5
3	42.5	65.0	82.5	0.0	0.0	0.0	0.0	0.0	0.0	60.0	60.0	62.5
4	90.0	97.5	90.0	97.5	97.5	97.5	95.0	97.5	97.5	100.0	100.0	100.0
5	67.5	80.0	95.0	7.5	7.5	7.5	60.0	67.5	62.5	35.0	37.5	37.5
6	5.0	22.5	47.5	32.5	35.0	32.5	0.0	0.0	0.0	0.0	0.0	0.0
7	12.5	40.0	37.5	7.5	12.5	12.5	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	20.0	57.5	80.0	37.5	52.5	72.5	0.0	0.0	0.0	0.0	0.0	0.0
10	22.5	95.0	92.5	42.5	62.5	65.0	0.0	0.0	0.0	0.0	0.0	0.0
11	95.0	97.5	100.0	70.0	75.0	77.5	0.0	2.5	2.5	0.0	0.0	0.0
12	0.0	25.0	62.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	82.5	97.5	100.0	62.5	65.0	65.0	50.0	62.5	65.0	0.0	0.0	0.0
14	2.5	20.0	40.0	0.0	0.0	0.0	35.0	47.5	50.0	0.0	2.5	0.0
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Average	39.325	58.325	67.325	29.5	32.825	32.65	21.825	25.15	25.325	14.5	15.325	15.5

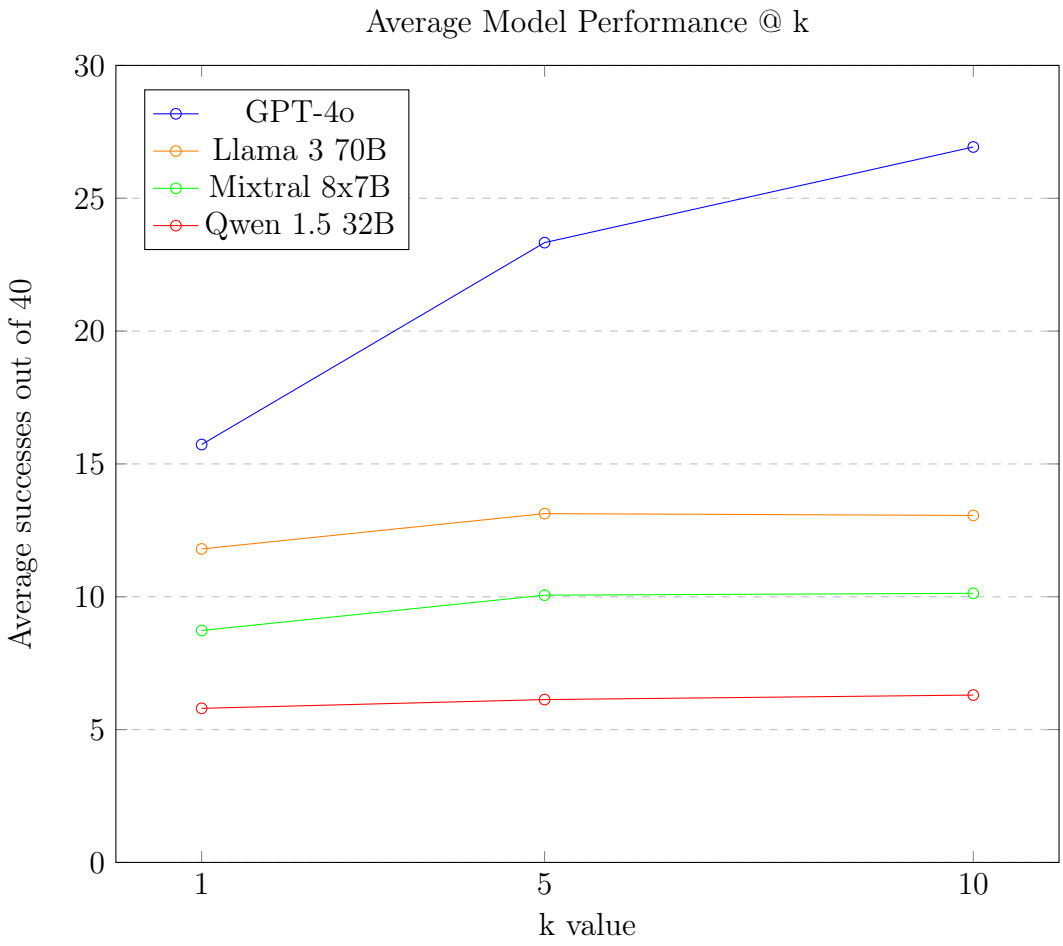

Figure 5.2: The average success out of 40 at each tested k value for pass@k. Only GPT-4o showed the expected growth at higher k values.

Correct Response Sparsity

Models appear to have 'sparse' abilities, where a task is either achieved with reasonable accuracy(>50% over all rephrasings) or fails to achieve it at all. While some tasks did have success rates greater than 0% and less than 50%, a surprising majority fell outside this range.

Specific Task Performance

In addition to the wide range of model performance, the tasks that models succeeded or failed to complete also varied greatly. Only two tasks weren't able to be completed by any models. The first of these tasks was task 15, which involved opening a modal in the user interface.

This task was designed to be impossible for the models, as it required information from a separate top-level key than the one being modified (see Section 4.2). However, the models were instructed in the system prompt to respond "Not possible" for tasks that couldn't be completed, which they all failed to do. The other task that no models could succeed on was task eight, which instructed them to open a system pane. It seems likely that the models failed reliably in this task due to the limited documentation about this system pane, in addition to its name "appSettings" being confused with the "settings" top-level key.

While the models tested didn't all achieve perfect accuracy on any one task at any tested k value, some tasks had generally high success rates. The most reliably succeeding task was task four, which involves hiding a node type from the map. This task was relatively simple, only requiring the models to change a value from True to False, which is likely why it was so often completed. Interestingly, all models other than the Qwen 1.5 family also performed very well on task two, hiding country shading from the map. While the required command was very similar to the command for hiding the node type (which Qwen 1.5 was very good at), it seems plausible that the difficulty resulted from the use of the word "country", which isn't present in the documentation and can have many other associations. Finally, all other models were also able to follow the rephrasings of task 13 with some success, which instructed models to "remove the last chart". While the Qwen 1.5 model's failure here may be due to similar reasons as with the country shading, it's also possible that this result was because of the different cultural understanding of where the "last" chart would be located.

5.2.1 Individual Model Results

GPT-4o

GPT-4o was the best performing model, achieving an average pass@k of 55% over all three k values tested. GPT-4o also scaled the best when increasing the k value, though it still only achieved an average pass@k of 67.5% at k=10. Other than the tasks that no models

achieved (mentioned in the section above), GPT-4o was able to complete all tasks with some success starting at k=5. However, despite GPT-4o's excellent task performance, it wasn't able to achieve perfect results on any tasks at k values lower than k=10.

Llama 3 70B

Llama 3 70B was the second best performing model, achieving an approximate 32.5% average pass@k over all k values tested. This sharp performance drop-off is likely due to the significantly smaller model size compared to GPT-4o. While the exact number of parameters of GPT-4o is unknown, it is likely higher than the 175B parameters of GPT-3 [6], meaning that it is over double the size of Llama-3 70B. Additionally, Llama 3 70B was quantized to five bits, which likely had an impact on performance. Llama 3 70B also scaled very badly with increasing k values, improving about 2.5% on average between k=1 and k=10.
This model failed to complete two additional tasks. The first task it completely failed was task 12, which requires that models activate demo mode. As this task is similar to opening the system pane in task 8 discussed above, it is likely they were failed for similar reasons. The other task failed was task 14, removing all charts. This failure may have been due to the model not understanding how to perform a complete list deletion using the JSON structure defined in Section 4.4. Finally, Llama 3 70B wasn't able to achieve perfect results on any tasks no matter the k value.

Mixtral 8x7B

Mixtral 8x7B performed third overall, with an approximate pass@k over all k values of only 25%. Mixtral 8x7B also failed to significantly scale at higher k-values, increasing by a similar amount to Llama 3 70B. While Mixtral 8x7B didn't perform exceptionally, it did very well on the tasks that it could achieve at least one success at. If the tasks where it scored 0 are ignored, Mixtral 8x7B has an average of 55%, slightly higher than the larger Llama 3 70B on the same metric.

Mixtral 8x7B struggled with tasks six through twelve, which tended to relate to charting, perhaps indicating it failed to properly represent the charting documentation or that it was unable to perform required inference of the correct list index from the object's or request's properties needed for interacting with charts. This model also failed to have a 100% success rate on any task at any k value, correctly responding to only 39/40 rephrasings on its best task at both k=5 and k=10.

Qwen 1.5 32B

Qwen 1.5 32B came in 4th place, only achieving an average pass@k of 15%. Despite this lackluster performance, Qwen 1.5 32B was the only model capable of achieving perfect performance on a single task at k=1. On the node hiding task described above Qwen 1.5 32B achieved 100% accuracy. However, even at k=10 Qwen 1.5 32B wasn't able to achieve over 62.5% accuracy on any other tasks. This extremely task specific performance may indicate that Qwen 1.5 32B doesn't represent the user intent well enough to translate it into modifications to the CAVE API state, as it isn't able to perform the nearly identical task (from a CAVE API perspective) of hiding the area shading.

5.3 Discussion

Implications on Practical Usability

While the open source model's performance on the tasks used for testing revealed interesting patterns, only the commercial GPT-4o model's results indicated that it would be able to serve as a practical NLP interface for non-expert users of supply chain visualization applications. At k=1, this model appears to be more capable of performing task related to modifying the map portion of CAVE Apps, with reliably high scores on these tasks.

Conveniently, many of GPT-4o's observed failures were detectable using the CAVE validator described in Section 4.4. Using this validator in combination with higher k values such as

k=5 or k=10 would greatly increase the models ability to act as a practical NLP interface for this application, as responses that fail to validate could simply be ignored. This would enable more complete CAVE App interactions to be tractable, as GPT-4o became proficient with modifying graphs and dashboards at these k values.

Even when using the validator for CAVE Apps, precautions must be used when using a LLM as a NLP interface for a CAVE App, and likely for other similar applications. One risk that must be accounted for is that the entire application state is sent to a remote server when using commercial APIs, such as the OpenAI API used to access GPT-4o. If sensitive data is present in the request, it is possible that this may be used to train the model and later be revealed to other commercial API users. Another important concern is that there are portions of the CAVE API where URLs are passed to be fetched by the end users system. An unaligned LLM could perform a request to replace these URLs with replacements that infect the user's system or otherwise cause harm. In the case of the CAVE App, these URLs are few enough that they can be manually excluded from being changed by the LLM, but for other applications this may be more difficult.

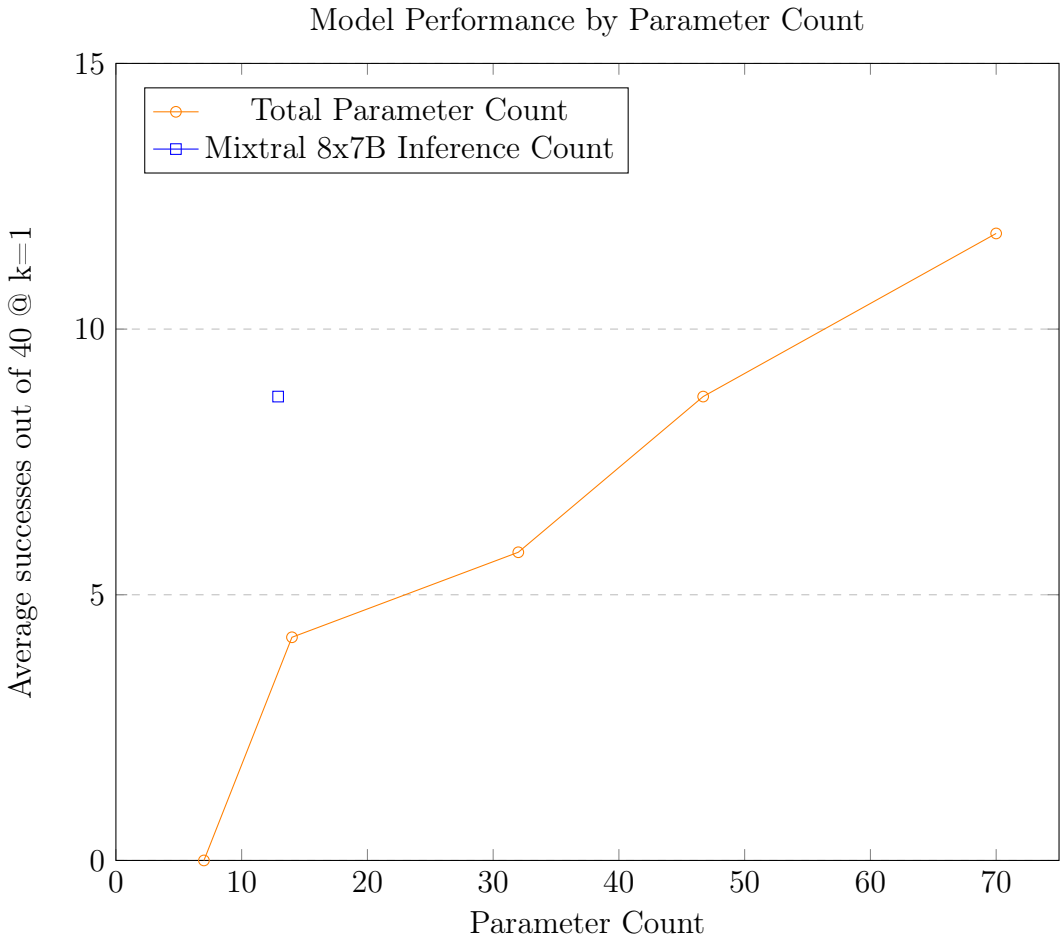

Figure 5.3: The average success out of 40 for pass@1 charted over the count of model parameters. The parameter count used by Mixtral 8x7B during inference is shown in blue. GPT-4o is excluded as its parameter count is not publicly known.

Parameter Size Effects

As shown in Figure 5.3, increasing model size resulted in a direct increase in performance. Qwen 1.5 7B was only able to return a single correct response over the entire test suite, indicating very poor performance. Qwen 1.5 14B however displayed results with more successes. Additionally, Qwen 1.5 14B had the same sparsity as present in the larger Qwen 1.5 32B model. Specifically, the 14B version of the model had no tasks where it responded correctly to between 3 and 25 rephrasings, and of the two tasks that were above 25/40 correct, the hiding the nodes from the map task had a higher pass@k at all k values. Finally, it is worthwhile to note that the 14B parameter model's other successful task was the "remove

the last chart" task, which Qwen 1.5 32B struggled with. Full 7B and 14B Qwen 1.5 results are available in Appendix C

Mixture of Experts Scaling

As discussed in Section 4.1, the mixture of experts architecture of Mixtral 8x7B means that while the model contains 46.7B parameters, it takes the same time as a 12.9B parameter models to process input and generate output. While differences in training data cannot be discounted, making it difficult to compare different models by number of parameters, Mixtral 8x7B's performance on the tasks was much closer to what would be expected from it's total parameter count rather than the number of tokens used for inference. The scaling could indicate that this architecture is very good for applications such as NLP interfaces, where feeling responsive to the user is of upmost importance.

Rephrasing Model Impact

The model used for rephrasing the task were tracked, as it was expected that each LLM would do best on the rephrasings it generated. However, this was not found in the results, with only Mixtral 8x7B doing best on its own prompts. Even Mixtral 8x7B didn't significantly prefer prompts it generated, only scoring one more correct response on its prompts than the runner-up model of Qwen 1.5 (114/450 vs 113/450).

There also didn't seem to be any particular model that generated the best prompts. Mixtral 8x7B and GPT-4o both did best on prompts generated by Mixtral 8x7B, while Qwen 1.5 and Llama 3 70B did best on prompts generated by GPT-4o. However, like with Mixtral 8x7B, these differences were very small and often not significantly significant.

The lack of any large rephrasing model impact is likely related to the correct response sparsity described in Section 5.2. These two results combine to show that model performance on any particular task is independent of the particular phrasing the user describes the task with, and instead depends on the models ability to internally represent the documentation and

current application state. It seems likely that this finding then indicates that models are very good at the first challenge described in Section 1.2, translating the natural language input into user intent. However, the difficulty for the models seems to arise in understanding the capabilities of the system and implementing the users intent.

Chapter 6

Conclusion

6.1 Summary of Findings

Supply chains are complex networks where changing one parameter can have surprising downstream effects. When visualizing supply chains, interactivity is important as it allows users to better understand the network wide impacts of individual changes to parts of the network. However, these interactions can be opaque and unintuitive for non-expert users.

Natural language interfaces can help solve this problem by allowing users to speak naturally to the interactive supply chain visualization program. However, current natural language interfaces are only capable of general-purpose data visualization, and must be extensively customized when attempting to use them on new programs.

In this thesis we have examined how already trained foundation large language models can be integrated as a natural language interface for supply chain visualization applications using context learning. We first described the supply chain visualization application that the natural language interface will be connected with, what interactions will be possible for the natural language interface, and the details of how the interface will modify the application state.

We then described the various models that will be tested as natural language interfaces

and why these models were chosen for this task. We also specified how to application documentation and current state will be fed to the models, as well as the format of their responses.

For our evaluation we used the models being tested to represent non-expert users, and had them rephrase 15 different tasks. We then tested each model on all the rephrasings of the tasks to identify its performance as a natural language interface. It was found that the commercial model tested, GPT-4o, was the best performer, and could be practical implemented as part of a natural language interface for supply chain data visualization. None of the other models tested were successful enough on the tasks to be considered for a practical user interface. However, it was found that the mixture of experts model tested scaled very well relative to its number of parameters, indicating that this architecture may be practical for a natural language interface if scaled up in size.

Overall, of the models tested only GPT-4o could be considered a useful natural language interface. With the addition of sampling the ten top responses and checking them with a validator to ensure they match API specifications, this model was able to perform all tasks with reasonable accuracy, and should be reliable enough for real world use.

One major limitation of the work presented in this thesis is the hardware constraints that were imposed upon the open models. Larger or less quantized variants of these models are available and would likely improve performance, but couldn't be tested.

Another limitation of this work is the number of tasks considered. While they span a broad range of CAVE App functionality, human users are certain to request tasks not tested, and model performance has been shown to depend somewhat on the individual task.

Lastly, the analyses presented here are limited by the context size chosen. While some of the models tested were trained with a larger context size, they were all limited to eight thousand tokens to allow for the same prompts to be used for all models. However, the models with larger context sizes may have performed better with a prompt that takes advantage of this property.

6.2 Future Work

Multiple improvements could be made to increase model performance on similar tasks, and improve the natural language interface's practicality. The easiest of these improvements would be remove the hardware constraints and running non-quantized and larger models on the tasks. Specifically, testing the scaling of the larger Mixtral 8x22B mixture of experts model, the Qwen 1.5 72B model, and the not yet released Llama 3 400B+.

For existing or smaller models a more systematically chosen prompt could improve performance or, for those models trained on a larger context size, the inclusion of an example user request and correct response to perform 1 or 2 shot learning. In addition to more advanced prompt engineering, it is possible that by adding images to the documentation or current state, multi-modal models such as GPT-4o could better understand the app and improve in performance.

Finally, as model performance improves and more models are able to be practically implemented as natural language interfaces it will become important to perform user studies to determine how the system works with live users in a real-world setting.

Appendix A

Prompt Templates

The prompt templates used. Text surrounded by { } brackets represents a variable.

A.1 Documentation Selection Prompts

A.1.1 Without Error

```
System: "Using the given documentation delimited by triple quotes,
    respond to user requests with nothing but the name of the top-
    level key that must be edited in order to fufill the request
    surrounded by single quotes"

User: "{documentation index}

request: {request}
"
```

A.1.2 With Error

```
System: "Using the given documentation delimited by triple quotes,
    respond to user requests with nothing but the name of the top-
   level key that must be edited in order to fufill the request
   while avoiding the error surrounded by single quotes

Error: {error}"

User: "{documentation index}

request: {request}
"
```

A.2 Request Completion Prompts

A.2.1 Without Error

```
System: "'Using the given documentation delimited by triple quotes
    and current api state encoded in JSON, respond to user
   requests with the path(s) and value(s) you would modify in the
   api to achieve the users desired action in the format
```json
{\"path\": example.path.here, \"value\": value}```. If the user
   request isn\'t possible with the given data respond with \'Not
   possible\'"

User: "{key documentation}
```
```

```
{key specific api data}

request: {request}
"
```

A.2.2 With Error

```
System: 'Using the given documentation delimited by triple quotes
   and current api state encoded in JSON, and an error message
   describing a potential issue, respond to user requests with the
    path(s) and value(s) you would modify in the api to achieve
   the users desired action while avoiding the error in the format
```` ```json
{\"path\": example.path.here, \"value\": value}``` ````. If the user
   request isn\'t possible with the given data respond with \'Not
   possible\'

Error: {error}'"

User: "{key documentation}

{key specific api data}

request: {request}
"
```
```

A.3 Task Generation Prompts

A.3.1 Qwen 32B Prompt

System: "Respond only in English"

User: "Restate the following command 10 times by rephrasing it using simpler language, writing the rephrasings as a json list of strings: {command}"

A.3.2 Other Models Prompt

User: "Restate the following command 10 times by rephrasing it using simpler language, writing the rephrasings as a json list of strings: {command}"

Appendix B

Base Tasks

Below the base tasks are listed with a brief description of how they would be implemented by a model or API programmer.

1. **Color the country geo by learners** - Modifying a single value in an application specific path to the string "learners".

2. **Hide the country geo from the map** - Changing a value in the same application specific path to False.

3. **Color the partner nodes by partner type** - Similar to task one, but following a different path.

4. **Hide the Scale Centers nodes from the map** - Similar to task three, but following a different path.

5. **Switch to the statistics dashboard** - Changing a path generic to all CAVE Apps to an application specific string "dash2"

6. **Change the pass rate chart to verification rate** - Modifying a path where the specific list index changed must be inferred by its contents.

7. **Change the bottom right chart to show verification rate** - Similar to task six, except the list index must be inferred based on it's positioning as described in the API documentation.

8. **Open the settings pane** - Inserting a value described in the API documentation in a path generic to all CAVE Apps.

9. **Change the first chart to show total enrollments by year** - Similar to task seven, however the way to group by year must also be inferred from API state.

10. **Change the top left chart to show total enrollments by year** - Changing the same value as task 9, however its position in the list must be inferred from documentation.

11. **Change the credentials earned chart to a line chart** - Similar to item six, though altering a slightly different path.

12. **Activate demo mode** - Similar to task eight.

13. **Remove the last chart** - Modifying a list to no longer include the item in the last index, or for it to be an empty object.

14. **Remove all the charts** - Modifying the same list as task 13, but now to be fully empty, or set to a null value.

15. **Open the United States country model** - This task requires setting a path to end in a CAVE App specific object. However, this object isn't under the same key as the path so models are expected to return "Not Possible".

Appendix C

Qwen 1.5 14B Results

Table C.1: Successes per task out of 40. k=1/5/10 *=task Qwen 1.5 7B completed

Task #	Qwen 14
1	0/0/0
2	0/1/0
3	0/0/0
4	35/36/36
5	2/3/3
6	1/1/1
7	0/0/0
8	0/0/0
9	0/0/0
10	0/0/0
11	0/0/0*
12	0/1/0
13	25/35/34
14	0/0/0
15	0/0/0
Average	4.20/5.13/5.00

References

[1] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios, "Datatone: Managing ambiguity in natural language interfaces for data visualization," in *Proceedings of the 28th annual acm symposium on user interface software & technology*, 2015, pp. 489–500.

[2] A. Narechania, A. Srinivasan, and J. Stasko, "Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 369–379, 2021. DOI: 10.1109/TVCG.2020.3030378.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[4] I. Rocco, M. Cimpoi, R. Arandjelović, A. Torii, T. Pajdla, and J. Sivic, "Ncnet: Neighbourhood consensus networks for estimating image correspondences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 2, pp. 1020–1034, 2022. DOI: 10.1109/TPAMI.2020.3016711.

[5] M. Chen, J. Tworek, H. Jun, *et al.*, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. arXiv: 2107.03374. URL: https://arxiv.org/abs/2107.03374.

[6] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[7] L. Ouyang, J. Wu, X. Jiang, *et al.*, *Training language models to follow instructions with human feedback*, 2022. arXiv: 2203.02155. URL: https://arxiv.org/abs/2203.02155.

[8] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Patil, I. Stoica, and J. E. Gonzalez, *Berkeley function calling leaderboard*, https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.

[9] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[10] AI@Meta, "Llama 3 model card," 2024. URL: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[11] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[12] S. Petridis, M. Terry, and C. J. Cai, *Promptinfuser: How tightly coupling ai and ui design impacts designers' workflows*, 2023. arXiv: 2310.15435. URL: https://arxiv.org/abs/2310.15435.

[13] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24, Washington D.C., DC, USA: Association for Computing Machinery, 2024, pp. 543–557, ISBN: 9798400704895. DOI: 10.1145/3636534.3649379. URL: https://doi.org/10.1145/3636534.3649379.

[14] M. C. Lab. URL: https://github.com/MIT-CAVE/cave_app/.

[15] M. C. Lab. URL: https://github.com/MIT-CAVE/cave_utils/.

[16] W.-L. Chiang, L. Zheng, Y. Sheng, *et al.*, *Chatbot arena: An open platform for evaluating llms by human preference*, 2024. arXiv: 2403.04132 [cs.AI]. URL: https://arxiv.org/abs/2403.04132.

[17] M. Ravkine, *Can ai code results - a hugging face space by mike-ravkine*. URL: https://huggingface.co/spaces/mike-ravkine/can-ai-code-results.

[18] A. Reuther, J. Kepner, C. Byun, *et al.*, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, IEEE, 2018, pp. 1–6.

[19] May 2024. URL: https://openai.com/index/hello-gpt-4o.

[20] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[21] W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan, "Agieval: A human-centric benchmark for evaluating foundation models," *arXiv preprint arXiv:2304.06364*, 2023.

[22] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," *arXiv preprint arXiv:2305.13245*, 2023.

[23] Q. Team, *Introducing qwen1.5*, Feb. 2024. URL: https://qwenlm.github.io/blog/qwen1.5/.

[24] mitmproxy, *GitHub - mitmproxy/pdoc: API Documentation for Python Projects — github.com*, https://github.com/mitmproxy/pdoc, [Accessed 28-06-2024], 2013.

[25] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 17 754–17 762.