Synthesizing Object Models from Natural Language Specifications

by

Alex Gu

S.B., Computer Science and Engineering, Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

© Massachusetts Institute of Technology 2022. All rights reserved.

Author .................................................................

Department of Electrical Engineering and Computer Science

May 13, 2022

Certified by .............................................................

Armando Solar-Lezama

Professor

Thesis Supervisor

Certified by .............................................................

Jacob Andreas

Assistant Professor

Thesis Supervisor

Accepted by ............................................................

Katrina LaCurts

Chair, Master of Engineering Thesis Committee
Synthesizing Object Models from Natural Language Specifications

by

Alex Gu

Submitted to the Department of Electrical Engineering and Computer Science on May 13, 2022, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Program synthesis has traditionally excelled in tasks with precise specifications such as input-output examples and formal constraints by using structured and algorithmic approaches based on enumerative search and type inference. However, traditional synthesis techniques have no mechanism of incorporating real-world knowledge, which is commonplace in software engineering. Motivated by this, we introduce a new synthesis task known as specification reification: synthesizing concrete realizations of vague, high-level application specifications. We focus on a specific instance of this: generating object models from natural language application descriptions. Towards this goal, we present three approaches for object model synthesis that leverage domain knowledge from the GPT-3 language model. In addition, we design a scoring metric to evaluate the success of synthesized object models on seven sample tasks such as classroom management and pet store applications. We demonstrate that our language-model-based synthesizers generate object models that are comparable in quality to human-generated ones.

Thesis Supervisor: Armando Solar-Lezama
Title: Professor

Thesis Supervisor: Jacob Andreas
Title: Assistant Professor
Acknowledgments

To my advisor, Armando Solar-Lezama: I am forever thankful for all the support you’ve given me, both as a researcher and as a person. Thanks for helping me enter a field I had no experience in, inviting me to go to SPLASH, always being so approachable as a person, teaching me how to be a great research mentor, and finding creative approaches to solve any problem I ever had. Throughout this past year, you’ve been incredibly supportive of me not just in my research, but also in all other aspects of my existence. I am fortunate and excited to have the opportunity to continue having you as my advisor for the next few years of my life.

To my co-advisor, Jacob Andreas: you’ve been immensely supportive and helpful all throughout my MEng. You’ve constantly provided encouragement and well as research ideas that have inspired my excitement in working on my thesis project. And of course, you’ve shown me how exciting the world of NLP can be.

Thanks to all the wonderful members in my lab for everything from research ideas to the liveliest group meetings to chill weekend hangouts: Ivan Kuraj, Jack Feser, Jimmy Koppel, Kavi Gupta, Kliment Serafimov, Leo Hernandez Cano, Matt Bowers, Omar Costillo Reyes, Ramya Durvasula, Ria Das, Theo Olausson. You’ve been amazing labmates, and I’ve had the time of my life here. Thanks to Daniela Velez and Tamara Mitrovska for being enthusiastic, self-driven, and wonderful UROPs and for trusting me as a first-time UROP mentor: I’ve enjoyed collaborating with both of you on this project.

Thanks to all of the members of the OPTML lab for having me as a SuperUROP. Special thanks to Kwangjun Ahn and Jingzhao Zhang for having so many discussions with me on the acceleration of Polyak-Łojasiewicz, Horia Mania for giving me an opportunity to explore and enjoy the deep world of multi-armed bandits, and Alp Yurtsever for being an amazing mentor through my first conference paper. Thanks, especially, to Suvrit Sra, not only for giving me invaluable research advice all throughout my SuperUROP, but also for all the teaching insights you’ve shared throughout my two semesters of TA’ing with you.
Thanks to all my collaborators and co-authors at the AI4LIFE lab: Satya Krishna, Tessa Han, Javin Pombra, Shahin Jabbari, Steven Wu. It’s been a great time collaborating with all of you and seeing all the pieces of the project, from theory to experiments to user studies, all come together. Special thanks to Hima Lakkaraju for talking to me about everything from startups to graduate schools to life as a Harvard professor.

Thanks to the team I worked with during my internship at IBM: Songtao Lu, Parikshit Ram, and Lily Weng. Thanks for guiding me through a fulfilling project and for continuing to work together to make our paper even stronger after each conference paper rejection.

Thanks to all my other academic and research mentors, who have given me excellent advice and made me a better researcher today: Costis Daskalakis, Dylan Foster, Lizhong Zheng, Luca Daniel, Nada Amin, Ranjit Jhala, Steve Chong, Sumit Gulwani.

Thanks to all my friends for four enriching and wonderful years together. To Alan Peng, Andrea Jaba, Andrew Gu, Anzo Teh, Ashley Lin, Eric Zhang, Jason Chen, Jeffery Yu, Jenny Gao, Jiahai Feng, Karen Gao, Kate Xu, Kendall Yu, Kevin Qian, Laura Cui, Spencer Compton, Sualeh Asif, Tiffany Trinh, Vincent Huang, Zoe Song, Zoom Saowakon: I’ve enjoyed all the memories we’ve made together, and each of you has made my life better in one way or another.

To my girlfriend, Grace: where would I be without you? Every day and every night you think about me, care for my well-being, and wish the best for me, wherever I am. You push me to be a better person in every dimension and never have second-thoughts about being there for me. Thanks for being my girlfriend, and I’m so lucky to have you in my life.

And lastly, to my family: ever since I was young, you’ve always told me to pursue my own passions and dreams. You’ve always been there for me and always supported me, no matter what I do or where I go. From the time I was born to today, you’ve never stopped encouraging me, cheering me on, or doubted me. All of this would be impossible without all the thought and care you put into me. Thank you Aaron, Mom, Dad, and Grandma, for everything you’ve done for me.
Contents

1 Introduction 13

2 Related Work 17

2.1 Program synthesis ........................................ 17
2.2 Ontologies and Knowledge Graphs ...................... 17
2.3 Large Language Models ................................. 18

3 The System 19

3.1 Problem Definition and Notation ...................... 19
3.2 Zero-shot GPT Generation ............................ 20
3.3 One-shot GPT Generation .............................. 20
3.4 Hierarchical GPT generation .......................... 21
   3.4.1 Extracting Object Fields ......................... 21
   3.4.2 Checking for Terminal Objects ................... 21

4 Evaluation 25

4.1 Data Collection ........................................ 25
4.2 Similarity Metric Components ....................... 26
   4.2.1 Field similarity: ................................. 26
   4.2.2 Object similarity: ............................... 27
4.3 Object Model Similarity: ................................ 28
4.4 Object Model Score: ................................... 28
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>A sample object model for a classroom management application</td>
</tr>
<tr>
<td>3-1</td>
<td>Sample input and output for our zero-shot object model generation algorithm</td>
</tr>
<tr>
<td>3-2</td>
<td>Sample input and output for our one-shot object model generation algorithm</td>
</tr>
<tr>
<td>4-1</td>
<td>The user interface for participants to design object models for our data collection effort.</td>
</tr>
<tr>
<td>5-1</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a pet store application</td>
</tr>
<tr>
<td>C-1</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a concert management application</td>
</tr>
<tr>
<td>C-2</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a library application</td>
</tr>
<tr>
<td>C-3</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a hotel application</td>
</tr>
<tr>
<td>C-4</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a restaurant application</td>
</tr>
<tr>
<td>C-5</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a pet store application</td>
</tr>
<tr>
<td>C-6</td>
<td>Hierarchical, 1-shot, and 0-shot object models for a company management application</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Scores for each of the three object model generation algorithms . . . 32

B.1 The meta-metric score (0–1, with 1 being the best) for different possible similarity metrics . . . . . . . . . . . . . . . . . . . . 41
The goal of program synthesis is to generate programs from various types of specifications such as input-output examples, abstract properties about inputs and outputs, and formal logical specifications. Traditionally, program synthesis has been principally algorithmic, using methods like enumerative search, constraint-based sketching, or type-driven synthesis to generate programs. While these algorithms have been successful at synthesizing functions with well defined input/output behavior, creating software requires much more than implementing functions from well defined specifications. In particular, an important part of software development is leveraging domain knowledge to turn high-level application requirements into a detailed description of all the components and interfaces that will make up the application.

In this paper, we introduce specification reification as a new challenge for program synthesis. Specification reification refers to the previously mentioned process of taking a high-level, potentially vague specification of a problem and reifying it into a more concrete form. For example, consider a developer who is designing a classroom management application in an object-oriented language. Traditional program synthesis could help implement specific functions in this application—for example, a function to search for students who have not submitted an assignment—but before a developer gets to that point, they first need to design the application itself. This involves deciding which objects they need, and for each object deciding on their fields and methods, and for each method deciding what its specification should be. Specifi-
cation reification is the process of deriving this design from the high-level description of the application.

We consider a specific sub-problem of specification reification: the task of synthesizing object models from natural language specifications. More concretely, we present a system that takes as input a description of an application—e.g. “I want a classroom management app that tracks students, the assignments they’ve submitted, and the grades they’ve earned on those assignments”—and from this description synthesizes an object model consisting of a set of objects and their fields. A sample object model that one may wish to synthesize is shown in Figure 1.

<table>
<thead>
<tr>
<th>student</th>
<th>submission</th>
<th>assignment</th>
<th>teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- student</td>
<td>- title</td>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
<td>- assignment</td>
<td>- description</td>
<td>- students</td>
</tr>
<tr>
<td>- teacher</td>
<td>- grade</td>
<td>- due date</td>
<td>- submissions</td>
</tr>
<tr>
<td>- assignments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- submissions</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1-1: A sample object model for a classroom management application

We emphasize two challenges of the task at hand:

1. First, it is challenging to automatically inferring objects and fields. In Figure 1, the teacher and submission objects are not given in the prompt; nor are the name and submissions fields for the student table—both need to be inferred. A successful system for this task should be able to infer related objects directly from the prompt.

2. Second, many objects refer to other objects in their fields: for example, the student object references the teacher object, the assignment object, and the submission object as fields. This highlights that a system should understand how various objects are related.

Towards this goal, our contributions are as follows:

1. We introduce and highlight a new task in program synthesis: specification reification. We also introduce the object model synthesis task as an important
sub-problem of specification reification.

2. We introduce a new evaluation metric and benchmark for object model synthesis. First, we gather a small dataset of hand-written object models for seven prompts as gold-standard object models. Next, we propose a similarity metric that quantifies the closeness between two object models. Finally, we define a scoring function that measures how well a generated object model fits a particular prompt.

3. We present three approaches for tackling the object model synthesis problem. The key innovation in all these algorithms is the use of GPT-3, a large language model which has background knowledge about the world. The first is a zero-shot approach, in which we directly ask GPT-3 what objects and fields a desired application has. The second is a one-shot approach, where we give GPT-3 an example of a successfully synthesized object model and ask it to generate an object model for a different prompt. The final is a more structured hierarchical approach, starting from the objects in the initial prompt and recursively prompting GPT-3 for related objects and fields. For most prompts, we observe that all three proposed approaches generate object models at a level similar to user-generated models and perform much better than randomly sampled models of different prompts.
Chapter 2

Related Work

2.1 Program synthesis

The field of program synthesis has had a long history, with a variety of approaches summarized in the survey \cite{survey}. The first line of approaches to appear mostly focused on inductive synthesis (matching a set of input-output examples) approaches such as bottom-up search \cite{bottom-up}, top-down search \cite{top-down}, type-directed search \cite{type-directed}, and constraint-solving \cite{constraint-solving}. Later, however, richer forms of program specifications were used for synthesis.

In recent years, with new developments in machine learning, there have been more and more works exploring the potential of augmenting traditional synthesis techniques with neural networks; \cite{neural} provides a complete survey. These include approaches to learn abstractions and libraries from scratch \cite{learn-abstractions, learn-libraries}, execution-guided approaches that evaluate partial program states \cite{execution-guided, execution-guided-1, execution-guided-2}, and approaches guided by natural language information \cite{natural-language-guided, natural-language-guided-2, natural-language-guided-3, natural-language-guided-4}.

2.2 Ontologies and Knowledge Graphs

There has also been a body of work that aims to build ontologies and knowledge graphs of natural language concepts, such as Yago \cite{yago}, WordNet \cite{wordnet}, and DBpedia \cite{dbpedia}. While these knowledge graphs have been applied in traditional NLP tasks such as
question answering \cite{4}, they are unable to provide specific insights for our synthesis task such as synthesizing fields for a certain object. As an example, when searching for nearest neighbors related to student, WordNet comes up with synonyms such as pupil, educatee, and scholar, while Yago provides a Wikipedia page for a student, a definition of a student in Spanish, and an image containing many students. In addition, our synthesis task is very contextual: the fields of a student object would be very different if we were designing an app for teachers to manage the classroom vs a social app for students to make friends with one another. It is difficult to capture this form of context via ontologies and knowledge graphs.

### 2.3 Large Language Models

Recent years has also seen the birth of new works leveraging large language models (LLMs) like GPT-3 \cite{5} to perform program synthesis. A few months ago, GitHub released a powerful code autocompletion tool called GitHub Copilot which uses context such as natural language comments and previous code in order to generate code. Copilot is built off of OpenAI’s powerful machine learning model Codex \cite{7}, which translates natural language to code in almost a dozen programming languages. CodeBERT \cite{10} learns representations of code and natural language for downstream tasks like code search and code documentation generation. \cite{14} use GPT-2 trained on a corpus of well-documented and commented code to synthesize programs for data science and machine learning. Building off of LLMs, \cite{3} incorporate human feedback to repair generated code.

There have been other works combining traditional program synthesis techniques with large language models. \cite{26} uses traditional inductive synthesis techniques with GPT-3 to learn small intermediate functions that cannot be represented symbolically. Jigsaw \cite{15} uses LLMs to synthesize code but use program analysis techniques to do post-processing. \cite{23} take a component-based synthesis approach guided by LLMs which, for example, help rank candidate programs.
Chapter 3

The System

3.1 Problem Definition and Notation

We begin by formalizing the object model generation task. The input to this task is a natural language specification of an application, e.g. “I want a classroom management app that tracks students, the assignments they’ve submitted, and the grades they’ve earned on those assignments.” The goal is to generate a realistic object model of this application, where each object consists of a name and a set of fields. Formally, an object model consists of \( n \) objects \( o_1, o_2, \ldots, o_n \). Each object \( o_i \) consists of a pair \((\text{name}_i, f_i)\) consisting of an object name and a set of \( m_i \) field names \( f_i \triangleq \{f_{ij}^i\}_{j=1}^{m_i} \).

For example, the \textit{student} and \textit{submission} objects from Figure 1 might be represented as follows:

\[
o_1 = (\text{student}, \{\text{name}, \text{email address}, \text{teacher}, \text{grade}, \text{assignments}, \text{submissions}\})
\]

\[
o_2 = (\text{submission}, \{\text{student}, \text{assignment}, \text{grade}\})
\]

For all of our algorithms, our system uses the GPT-3 Q&A API with the text-davinci-001 model and default parameters from the OpenAI web API\(^1\): a temperature of 0 (greedy decoding), frequency penalty of 0, and presence penalty of 0.

\(^1\)https://beta.openai.com/playground

19
Input: “I want a pet store app tracking customers. The pet store app should also keep the store’s inventory of pets and allow customers to buy and sell pets.

Q: What objects and fields does this application have?”

GPT-3 Output: “A: The application has the following objects: customer, pet, and inventory. The customer object has the following fields: name, address, and phone number. The pet object has the following fields: name, type, and description. The inventory object has the following fields: type, quantity, and price.”

3.2 Zero-shot GPT Generation

Our first approach is a zero-shot approach, in which the system prompts GPT-3 with our input specification and directly asks what objects and fields the desired application has (when evaluating this method, we manually parse the GPT-3 output into its objects and fields). A sample of the input and output of our zero-shot approach is shown in Figure 3-1.

3.3 One-shot GPT Generation

Our second approach is a one-shot approach, in which the system prompts GPT-3 with a complete object model for a classroom management app and asks it to generate an object model for a new prompt. In Figure 3-2 we show the input and outputs of GPT-3. The tables and fields are explicitly provided so that GPT-3’s few-shot generation system learns a desired format. In practice, GPT outputs always conform to this specification format, so we write a parser that automatically convert these outputs into explicit object models.
3.4 Hierarchical GPT generation

The final approach we use is a hierarchical approach. At a high level, our algorithm follows an iterative depth first search, keeping a stack of all the objects we’ve encountered so far. We begin by extracting all the noun phrases from the prompt using a purely symbolic approach leveraging Stanford’s CoreNLP parser [16], which we denote by \texttt{extract_noun_phrases(prompt)}. We treat these noun phrases as a set of initial objects. From these initial objects, we recursively get a set of fields for each object, denoted by \texttt{extract_object_fields(object)}. For each of these fields, which we consider as new potential objects, we first check if we have encountered the field before. If not, we then test to see if the field should additionally be treated as a new object, and if so, add it to the stack. We do this via a \texttt{terminal(new
d_object)} check to ensure that we are only adding objects to the stack. In pseudocode, our algorithm is as follows:

\subsection{3.4.1 Extracting Object Fields}

Now, we explain \texttt{extract_object_fields(object)}. Our key insight is that GPT-3 can generate these relevant objects and fields from its domain knowledge. Specifically, given an object, this function asks GPT-3 what fields the object has via questions in the form “\textit{Q: What attributes does a classroom management application have?}”. Remarkably, it would answer “A classroom management app has a list of students, a list of assignments, and a list of grades.”. Next, the system parses this response to understand that a classroom management application object has \texttt{students}, \texttt{assignments}, and \texttt{grades} as fields. In order to avoid duplicate objects, we also ensure all object names are in singular form when prompting, like \texttt{student} instead of \texttt{students}.

\subsection{3.4.2 Checking for Terminal Objects}

The \texttt{terminal(object)} function helps the system discriminate which fields should be further treated as objects. While a \texttt{student} having a \texttt{name} is important, a \texttt{name} object having a list of characters is irrelevant. We once again use GPT-3 by prompting it with
Q&A pairs like "Q: What does a grade have?" and "A: A grade has nothing because it is an integer.". The full prompt is deferred to Appendix A.1. This prompting mechanism helps GPT-3 learn that some objects have no fields, and the terminal function returns true if GPT says the object has nothing.
<table>
<thead>
<tr>
<th>Table: assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: title (string), description (string), due date (datetime)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: name (string), email address (string), phone number (string), teacher (teacher), grade (int), assignments (List[assignment]), submissions (List[submission])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: name (string), students (List[student]), submissions (List[submission])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: submission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: student (student), assignment (assignment), grade (int)</td>
</tr>
</tbody>
</table>

Q: Design a schema for the following prompt: "I want a pet store app tracking customers. The pet store app should also keep the store’s inventory of pets and allow customers to buy and sell pets."

GPT-3 Output:

<table>
<thead>
<tr>
<th>Table: customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: name (string), email address (string), phone number (string), pet (pet), quantity (int), price (float)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: pet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: name (string), type (string), description (string), quantity (int), price (float)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table: inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fields: pet (pet), quantity (int), price (float)</td>
</tr>
</tbody>
</table>

Figure 3-2: Sample input and output for our one-shot object model generation algorithm
Chapter 4

Evaluation

We also define an evaluation procedure to quantify the correctness of a generated object model. This is tricky because for a single natural language description, there are many object models fitting the given description. First, in Section 4.1, we describe a data collection process to obtain gold standard models. Then, in Section 4.2, we present our full evaluation and scoring metric using these models.

4.1 Data Collection

To obtain a set of gold-standard models, we set up an experimental testbed, asking participants to design object models from seven prompts simulating the following applications: pet store, restaurant, hotel, dating, library, company, and concert. The full prompts are in Appendix A. To help participants understand the task, we gave them an example of a full object model for a classroom application but emphasized that there is no correct answer. We included the example to demonstrate that participants should include objects and fields that weren’t explicitly mentioned in the prompt.

As shown in Figure 4-1, the participant was shown a prompt at random and instructed to create an object model corresponding to the prompt. The users were given freedom over the number of objects they created and the number of fields they used for each object. For each field, they were asked to specify its name and its type.
Figure 4-1: The user interface for participants to design object models for our data collection effort.

For types, we restricted participants to primitives, other objects they created, and lists of either. In total, we received 35 object models. However, we deemed 2 to be of extremely poor quality, leaving us with 33 gold-standard models.

4.2 Similarity Metric Components

Now, we propose an evaluation metric to determine the similarity between two object models. Intuitively, our metric works as follows: consider two object models $O = \{o_1, \ldots, o_n\}$ and $O' = \{o'_1, \ldots, o'_n\}$. For each object in $O$, say $o_i$, we find the most similar object in $O'$, measured using an object similarity metric. We then average the similarities to get the overall object model similarity. To compute how similar two objects $o_i$ and $o'_j$ are, we employ a similar process: for each field $f_{i,m}$, we find the field in $o'_j$ that is most similar to it via a field similarity metric, averaging the similarities to get the overall object similarity. Below, we make this concrete and precise.

4.2.1 Field similarity:

To begin, we define the similarity between any two fields. When comparing two fields, it’s important to consider the objects they were part of: many objects might have a “name” field that refer to very different things. Therefore, we concatenate the object name and field name, eg “pet name" or “customer name" when computing the
similarity. Concretely, the similarity between fields $f^a_i$ and $f^b_j$ is defined as

$$\text{field}\_\text{sim}(f^a_i, f^b_j) = \text{phrase}\_\text{sim}(\text{concat}(\text{name}_i, f^a_i), \text{concat}(\text{name}_j, f^b_j)).$$

(4.1)

Here, \text{phrase}\_\text{sim} is calculated by calling spacy’s built in .similarity() method on two phrases, which is a word2vec-like similarity measure in $[0, 1]$, 1 indicating perfect similarity.

### 4.2.2 Object similarity:

Next, we define a metric to evaluate object similarity. For two objects $o_i, o_j$ to be similar, their object names and set of fields should both be similar. Overall, we define the object similarity as a combination $f$ of the two, where we use $f(x, y) = xy$:

$$o\_\text{sim}(o_i, o_j) = f(o\_\text{name}\_\text{sim}(o_i, o_j), o\_\text{field}\_\text{sim}(o_i, o_j)).$$

(4.2)

First, we define the object name similarity similar to the field similarity, as

$$o\_\text{name}\_\text{sim}(o_i, o_j) = \text{phrase}\_\text{sim}(\text{name}_i, \text{name}_j).$$

(4.3)

Next, we define the object field similarity. Intuitively, if two objects $o_i$ and $o_j$ have identical sets of fields, there is a one-to-one mapping between each field in $o_i$ and each field in $o_j$. If they aren’t identical but similar, each field $f^a_i$ should still have fields that are roughly similar to fields in $o_j$. For example, $o_i$ might have a “name” field, but $o_j$ might have “first name” and “last name” as fields. In this case, there would still be an approximate mapping between fields. We use this idea to define our metric: for each field $f^a_i$ in object $o_i$, we compute the maximum field similarity between $f^a_i$ and a field $f^b_j$ in object $o_j$. We then average them to define

$$o\_\text{field}\_\text{sim}'(o_i, o_j) = \frac{1}{m_i} \sum_{a=1}^{m_i} \max_{b \in [m_j]} \text{field}\_\text{sim}(f^a_i, f^b_j).$$

(4.4)
Observe that if the fields in object \(i\) are a strict subset of the fields in object \(j\), then \(\text{obj\_field\_sim}'(o_i, o_j) = 1\), since each field in object \(i\) has a perfect match in object \(j\). However, \(\text{o\_field\_sim}'(o_j, o_i)\) will be smaller, since not every field in object \(j\) has a match in object \(i\). Therefore, to make our metric symmetric, we define the overall field similarity as

\[
\text{o\_field\_sim}(o_i, o_j) = \frac{1}{2} (\text{o\_field\_sim}'(o_i, o_j) + \text{o\_field\_sim}'(o_j, o_i)).
\] (4.5)

### 4.3 Object Model Similarity:

Finally, we define a similarity function to evaluate how similar two full object-oriented models are. Consider two object models \(O = \{o_i\}_{i=1}^n\) and \(O' = \{o'_i\}_{i=1}^{n'}\). Similar to our object similarity metric, we average the similarities from each object \(o_i \in O\) to its most similar object in \(O'\). We define

\[
\text{om\_sim}'(O, O') = \frac{1}{n} \sum_{i=1}^{n} \max_{j \in [n']} \text{o\_sim}(o_i, o_j),
\] (4.6)

\[
\text{om\_sim}(O, O') = \frac{1}{2} (\text{om\_sim}'(O, O') + \text{om\_sim}'(O', O)).
\] (4.7)

### 4.4 Object Model Score:

Finally, in order to evaluate an object model for a given prompt, we define a scoring metric taking object models to \([0, 1]\). Consider an object model \(O\) for a prompt \(P\) for which we wish to evaluate. Let \(O_1, \cdots, O_k\) be the gold-standard object models for \(P\). Intuitively, an object should have a high score if it is similar to any of the gold-standard models and have a low score otherwise. Therefore, we define the score of an object model on a given prompt to be its similarity to the most similar gold-standard model. Precisely,

\[
\text{score}_P(O) = \max_{i \in [k]} \text{om\_sim}(O, O_i).
\] (4.8)
A more detailed motivation and evaluation of this metric is provided in Appendix B. As we will show in 5.1, the score function defined above effectively discriminates between human-generated object models, assigning high scores to human gold-standard models for the same domain, and low scores to object model pair from different domains.
5.1 Quantitative Results

We present the scores of each of the generated object models for all seven prompts in Table 5.1 when compared against our gold-standard benchmark. The hier, 1-shot, and 0-shot columns represent the algorithms in Section 3. The similar column of a prompt $P$ measures the average score of gold-standard object models for that prompt when compared against the other gold-standard models. When calculating the score for an object $O_i$, we remove $O_i$ from the set of gold-standard models to obtain a leave-one-out score because otherwise all the scores would be 1. The different column measures the average score of gold-standard object models in prompt $P$ when compared against gold-standard models for the other prompts $P' \neq P$. This measures the similarity between models of a given prompt to gold-standard object models of other prompts. As we expect, these scores are much lower, as object models of different prompts should be different.

5.2 Qualitative Results

In Figure 5-1 we show the three object models for the prompt "I want a pet store app tracking customers. The pet store app should also keep the store’s inventory of pets and allow customers to buy and sell pets." We include our full list of generated
<table>
<thead>
<tr>
<th>prompt</th>
<th>hier</th>
<th>1-shot</th>
<th>0-shot</th>
<th>similar</th>
<th>different</th>
</tr>
</thead>
<tbody>
<tr>
<td>company</td>
<td>0.78</td>
<td>0.80</td>
<td><strong>0.84</strong></td>
<td>0.67</td>
<td>0.23</td>
</tr>
<tr>
<td>pet store</td>
<td><strong>0.81</strong></td>
<td>0.79</td>
<td>0.69</td>
<td>0.83</td>
<td>0.27</td>
</tr>
<tr>
<td>restaurant</td>
<td>0.86</td>
<td>0.89</td>
<td>0.77</td>
<td>0.81</td>
<td>0.31</td>
</tr>
<tr>
<td>hotel</td>
<td>0.81</td>
<td>0.88</td>
<td><strong>0.90</strong></td>
<td>0.92</td>
<td>0.21</td>
</tr>
<tr>
<td>dating</td>
<td>0.72</td>
<td><strong>0.83</strong></td>
<td>0.74</td>
<td>0.88</td>
<td>0.16</td>
</tr>
<tr>
<td>library</td>
<td>0.77</td>
<td>0.79</td>
<td>0.75</td>
<td>0.52</td>
<td>0.25</td>
</tr>
<tr>
<td>concert</td>
<td>0.75</td>
<td><strong>0.80</strong></td>
<td>0.68</td>
<td>0.76</td>
<td>0.26</td>
</tr>
<tr>
<td>average</td>
<td>0.78</td>
<td><strong>0.83</strong></td>
<td>0.77</td>
<td>0.77</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Table 5.1: Scores for each of the three object model generation algorithms

schemas for other prompts in Appendix C. Each of the methods has its own merits and flaws. First, we notice that all three methods are able to generate relevant information that isn’t explicitly provided in the prompt (such as a customer’s name). Second, in the hierarchical table, we can see that each field is either a primitive (such as name, address, phone number) or is the name of another object. However, in the 0-shot and 1-shot table, there are tables that are never referenced, like inventory in the 1-shot case and purchase in the 0-shot case. Third, we note that in contrast to the other methods, the 1-shot method often generates a more exhaustive list of fields.
Figure 5-1: Hierarchical, 1-shot, and 0-shot object models for a pet store application
In this work, we introduced a new class of important program synthesis problems known as *specification reification*, focused on incorporating domain knowledge into traditional program synthesis. First, we presented one specific instance of this task, object model synthesis, and designed a metric to evaluate performance on this task. Then, we demonstrated three different algorithms to solve this task, showing that we are able to synthesize, to some extent, object models satisfying the specification. These object models include fields and tables not explicitly mentioned in the original prompt. Evaluating these algorithms on our metric is a first step towards showing that our generated object models are similar to human gold-standard models.

We believe that specification reification is an important problem and welcome researchers to introduce other synthesis problems fitting this framework. We identify many attractive directions for future investigation: first, there is significant room for the discovery novel approaches that improve upon the three we present. Second, we encourage a more rigorous data collection effort and investigation of our evaluation procedure: the quantity of object models we collected is likely too small to fully cover the set of possible gold-standard models. Third, GPT often gives misleading or incorrect information, so one way of correcting this is to explore how human interactivity can be incorporated to correct and augment the synthesized object model. Finally, since software engineers often spend time designing what methods to implement, it would be interesting to extend the object model to include these method names.
types, and descriptions, e.g., turning in assignments for a classroom application.
Appendix A

Prompts

We used the following seven prompts throughout:

- I want a company management app where I can track my employees, their schedules, and their compensation.

- I want a pet store app tracking customers. The pet store app should also keep the store’s inventory of pets and allow customers to buy and sell pets.

- I want a restaurant management app tracking customers, their reservations, their orders, and menu items.

- I want a hotel app tracking guests, their reservations. The hotel app should also track which guests occupy which rooms.

- I want a dating app where students can indicate other students they are interested in, and the dating app will match them.

- I want a library app to track books. The customers can borrow books and return books from the library.

- I want a concert management app that lists concerts, their dates, and their locations. The customers can also buy tickets to concerts.
A.1 1-shot GPT Prompt

Below, we include the full prompt we provide to GPT-3 in the 1-shot setting:

I want a classroom management app that tracks students, the assignments they’ve submitted, and the grades they’ve earned on those assignments.

Q: What does a classroom management app have?
A: A classroom management app has a list of students, a list of assignments, and a list of grades.

Q: What does an assignment have?
A: An assignment has a title, a description, a due date, and a list of grades.

Q: What does a student have?
A: A student has a name, an email address, a phone number, a list of assignments, and a list of grades.

Q: What does a grade have?
A: A grade has nothing because it is an integer.

Q: What does a title have?
A: A title has nothing because it is a string.

Q: What does a description have?
A: A description has nothing because it is a string.

Q: What does a due date have?
A: A due date has nothing because it is a string.

Q: What does a name have?
A: A name has nothing because it is a string.

Q: What does a email address have?
A: An email address has nothing because it is a string.

Q: What does a phone number have?
A: A phone number has nothing because it is a string.
Appendix B

Meta-Metric

In Section 4.2, we defined a framework to capture the similarity between two object models. Since such a metric has not been established before, it is important to ensure that the metric faithfully captures the similarity between two object models. Therefore, we define a meta-metric to capture the faithfulness of a metric. The methodology in this section can be applied to future work in other instances of specification reification.

Consider two different prompts $P$ and $Q$. An important desideratum for an evaluation metric is that the similarity between object models for the same prompt should be closer than object models for different prompts. To that extent, let $O^P_1, \ldots, O^P_k$ be gold-standard object models for prompt $P$ and $O^Q_1, \ldots, O^Q_l$ be gold-standard object models for prompt $Q$. For a given similarity metric, we define

$$
\mu_{PQ} = \frac{1}{2k} \sum_{i \in [k]} \mu_Q(O^P_i) + \frac{1}{2l} \sum_{i \in [l]} \text{score}_P(O^Q_i),
$$

$$
\mu_{PP} = \frac{1}{k} \sum_{i \in [k]} \text{score}_P(O^P_i), \mu_{QQ} = \frac{1}{l} \sum_{i \in [l]} \text{score}_Q(O^Q_i).
$$

In a similar spirit to contrastive learning, we want $\mu_{PP}$ and $\mu_{QQ}$ to be close to 1, while we want $\mu_{PQ}$ to be close to 0. This motivates the meta-metric

$$
\frac{\mu_{PP} + \mu_{QQ} - \mu_{PQ}}{1 + |\mu_{PP} - \mu_{PQ}|}
$$

which takes values in $[0, 1]$. In the best case scenario where $\mu_{PP} = \mu_{QQ} = 1$ and $\mu_{PQ} = 0$, the metric has a value of 1. In the worst case scenario where $\mu_{PP} = \mu_{QQ} = 39$
\(\mu_{PQ}\), signifying that the metric cannot differentiate between object models of different prompts, the metric has a value of 0.

Recall that in Eq. 4.2 the function \(f(\cdot, \cdot)\) calculates the similarity between two objects as a combination of the object names and object fields. We considered two ways to combine: \(f(x, y) = xy\), capturing the fact that object names and object fields should be similar, and \(f(x, y) = \lambda x + (1 - \lambda)y\), representing a relative weighting between the two aspects. Apart from \(f\), we also considered replacing the max in Eq. 4.4 with an average.

In Table B.1 we show the different values of the meta-metric for various combinations of \(\lambda\), \(f\), and aggregation strategy. In the \(f\) column, “ws” represents the weighted sum combination \(f(x, y) = \lambda x + (1-\lambda)y\), while “prod” represents the product \(f(x, y) = xy\). In the “agg” column, max represents using \(\text{score}_P(O) = \max_{i \in [k]} \text{om}_{-}\text{sim}(O, O_i)\), while avg represents using \(\text{score}_P(O) = \frac{1}{k} \sum_{i \in [k]} \text{om}_{-}\text{sim}(O, O_i)\). We found that using \(f(x, y) = xy\) and max aggregation worked best.
<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$f$</th>
<th>agg</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ws</td>
<td>max</td>
<td>0.25</td>
</tr>
<tr>
<td>0.1</td>
<td>ws</td>
<td>max</td>
<td>0.27</td>
</tr>
<tr>
<td>0.2</td>
<td>ws</td>
<td>max</td>
<td>0.29</td>
</tr>
<tr>
<td>0.3</td>
<td>ws</td>
<td>max</td>
<td>0.31</td>
</tr>
<tr>
<td>0.4</td>
<td>ws</td>
<td>max</td>
<td>0.34</td>
</tr>
<tr>
<td>0.5</td>
<td>ws</td>
<td>max</td>
<td>0.36</td>
</tr>
<tr>
<td>0.6</td>
<td>ws</td>
<td>max</td>
<td>0.38</td>
</tr>
<tr>
<td>0.7</td>
<td>ws</td>
<td>max</td>
<td>0.39</td>
</tr>
<tr>
<td>0.8</td>
<td>ws</td>
<td>max</td>
<td>0.41</td>
</tr>
<tr>
<td>0.9</td>
<td>ws</td>
<td>max</td>
<td>0.43</td>
</tr>
<tr>
<td>1</td>
<td>ws</td>
<td>max</td>
<td>0.45</td>
</tr>
<tr>
<td>0</td>
<td>ws</td>
<td>avg</td>
<td>0.24</td>
</tr>
<tr>
<td>0.1</td>
<td>ws</td>
<td>avg</td>
<td>0.26</td>
</tr>
<tr>
<td>0.2</td>
<td>ws</td>
<td>avg</td>
<td>0.28</td>
</tr>
<tr>
<td>0.3</td>
<td>ws</td>
<td>avg</td>
<td>0.30</td>
</tr>
<tr>
<td>0.4</td>
<td>ws</td>
<td>avg</td>
<td>0.31</td>
</tr>
<tr>
<td>0.5</td>
<td>ws</td>
<td>avg</td>
<td>0.33</td>
</tr>
<tr>
<td>0.6</td>
<td>ws</td>
<td>avg</td>
<td>0.35</td>
</tr>
<tr>
<td>0.7</td>
<td>ws</td>
<td>avg</td>
<td>0.36</td>
</tr>
<tr>
<td>0.8</td>
<td>ws</td>
<td>avg</td>
<td>0.38</td>
</tr>
<tr>
<td>0.9</td>
<td>ws</td>
<td>avg</td>
<td>0.40</td>
</tr>
<tr>
<td>N/A</td>
<td>prod</td>
<td>max</td>
<td>0.46</td>
</tr>
<tr>
<td>N/A</td>
<td>prod</td>
<td>avg</td>
<td>0.41</td>
</tr>
</tbody>
</table>

Table B.1: The meta-metric score (0–1, with 1 being the best) for different possible similarity metrics
Appendix C

Samples of generated schemas

In this section, we provide a full list of GPT generated schemas.
C.1 Object models for a concert management application

Prompt: I want a concert management app that lists concerts, their dates, and their locations. The customers can also buy tickets to concerts.

<table>
<thead>
<tr>
<th>Hierarchical (Score: 0.75)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
</tr>
<tr>
<td>- phone number</td>
</tr>
<tr>
<td>- concert</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1-shot (Score: 0.80)</th>
</tr>
</thead>
<tbody>
<tr>
<td>concert</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- date</td>
</tr>
<tr>
<td>- location</td>
</tr>
<tr>
<td>- price</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0-shot (Score: 0.68)</th>
</tr>
</thead>
<tbody>
<tr>
<td>concert</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- date</td>
</tr>
<tr>
<td>- location</td>
</tr>
<tr>
<td>- tickets</td>
</tr>
</tbody>
</table>

Figure C-1: Hierarchical, 1-shot, and 0-shot object models for a concert management application
C.2 Object models for a library application

Prompt: I want a library app to track books. The customers can borrow books and return books from the library.

<table>
<thead>
<tr>
<th>library app</th>
<th>library</th>
<th>customer</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>- customer</td>
<td>- book</td>
<td>- name</td>
<td>- title</td>
</tr>
<tr>
<td>- books borrowed</td>
<td>- books borrowed</td>
<td>- email address</td>
<td>- description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- phone number</td>
<td>- due date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- books borrowed</td>
<td>- books borrowed</td>
</tr>
</tbody>
</table>

1-shot (Score: 0.79)

<table>
<thead>
<tr>
<th>customer</th>
<th>book</th>
<th>library</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- title</td>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
<td>- author</td>
<td>- customers</td>
</tr>
<tr>
<td>- phone number</td>
<td>- pages</td>
<td>- books</td>
</tr>
<tr>
<td>- books</td>
<td>- due date</td>
<td></td>
</tr>
</tbody>
</table>

0-shot (Score: 0.75)

<table>
<thead>
<tr>
<th>customer</th>
<th>book</th>
<th>book loan</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- title</td>
<td>- customer</td>
</tr>
<tr>
<td>- address</td>
<td>- author</td>
<td>- book</td>
</tr>
<tr>
<td>- phone number</td>
<td>- ISBN</td>
<td>- due date</td>
</tr>
</tbody>
</table>

Figure C-2: Hierarchical, 1-shot, and 0-shot object models for a library application
C.3 Object models for a hotel application

Prompt: I want a hotel app tracking guests, their reservations. The hotel app should also track which guests occupy which rooms.

<table>
<thead>
<tr>
<th>Reservation</th>
<th>Hotel App</th>
<th>Guest</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>- guest</td>
<td>- guest</td>
<td>- name</td>
<td>- number</td>
</tr>
<tr>
<td>- room</td>
<td>- reservation</td>
<td>- email address</td>
<td>- description</td>
</tr>
<tr>
<td>- date</td>
<td>- room</td>
<td>- phone number</td>
<td>- guest</td>
</tr>
<tr>
<td></td>
<td>- reservation</td>
<td>- reservation</td>
<td>- reservation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- room</td>
<td>- year</td>
</tr>
</tbody>
</table>

1-shot (Score: 0.88)

<table>
<thead>
<tr>
<th>Guest</th>
<th>Reservation</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- guest</td>
<td>- number</td>
</tr>
<tr>
<td>- email address</td>
<td>- room number</td>
<td>- description</td>
</tr>
<tr>
<td>- phone number</td>
<td>- check-in date</td>
<td>- guests</td>
</tr>
<tr>
<td>- check-in date</td>
<td>- check-out date</td>
<td></td>
</tr>
</tbody>
</table>

0-shot (Score: 0.90)

<table>
<thead>
<tr>
<th>Guest</th>
<th>Reservation</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- date</td>
<td>- number</td>
</tr>
<tr>
<td>- email</td>
<td>- time</td>
<td>- name</td>
</tr>
<tr>
<td>- phone number</td>
<td>- room</td>
<td></td>
</tr>
</tbody>
</table>

Figure C-3: Hierarchical, 1-shot, and 0-shot object models for a hotel application
C.4 Object models for a restaurant application

Prompt: I want a restaurant management app tracking customers, their reservations, their orders, and menu items.

<table>
<thead>
<tr>
<th>reservation</th>
<th>customer</th>
<th>restaurant management app</th>
</tr>
</thead>
<tbody>
<tr>
<td>- date</td>
<td>- name</td>
<td>- customer</td>
</tr>
<tr>
<td>- time</td>
<td>- phone number</td>
<td>- reservation</td>
</tr>
<tr>
<td>- table number</td>
<td>- reservation</td>
<td>- order</td>
</tr>
<tr>
<td>- customer</td>
<td>- order</td>
<td>- menu item</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>order</th>
<th>menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>- date</td>
<td>- name</td>
</tr>
<tr>
<td>- time</td>
<td>- description</td>
</tr>
<tr>
<td>- table number</td>
<td>- price</td>
</tr>
<tr>
<td>- menu item</td>
<td>- order</td>
</tr>
<tr>
<td></td>
<td>- customer</td>
</tr>
</tbody>
</table>

1-shot (Score: 0.89)

<table>
<thead>
<tr>
<th>customer</th>
<th>reservation</th>
<th>order</th>
<th>menu item</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- customer</td>
<td>- customer</td>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
<td>- reservation date</td>
<td>- order date</td>
<td>- description</td>
</tr>
<tr>
<td>- phone number</td>
<td>- reservation time</td>
<td>- order time</td>
<td>- price</td>
</tr>
<tr>
<td>- reservation date</td>
<td>- table number</td>
<td>- menu item</td>
<td>- quantity</td>
</tr>
<tr>
<td>- reservation time</td>
<td>- customer</td>
<td>- price</td>
<td></td>
</tr>
<tr>
<td>- table number</td>
<td>- order</td>
<td>- quantity</td>
<td></td>
</tr>
<tr>
<td>- menu item</td>
<td>- customer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0-shot (Score: 0.77)

<table>
<thead>
<tr>
<th>customer</th>
<th>reservation</th>
<th>menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>- first name</td>
<td>- date</td>
<td>- name</td>
</tr>
<tr>
<td>- last name</td>
<td>- time</td>
<td>- description</td>
</tr>
<tr>
<td>- email</td>
<td>- number of people</td>
<td>- price</td>
</tr>
<tr>
<td>- phone number</td>
<td>- menu items</td>
<td>- quantity</td>
</tr>
<tr>
<td>- address</td>
<td>- notes</td>
<td></td>
</tr>
</tbody>
</table>
C.5 Object models for a pet store application

Prompt: I want a pet store app tracking customers. The pet store app should also keep the store’s inventory of pets and allow customers to buy and sell pets.

<table>
<thead>
<tr>
<th>Hierarchical (Score: 0.81)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
</tr>
<tr>
<td>- phone number</td>
</tr>
<tr>
<td>- pet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1-shot (Score: 0.79)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>- name</td>
</tr>
<tr>
<td>- email address</td>
</tr>
<tr>
<td>- phone number</td>
</tr>
<tr>
<td>- pet</td>
</tr>
<tr>
<td>- purchased</td>
</tr>
<tr>
<td>- sold</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0-shot (Score: 0.69)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
</tr>
<tr>
<td>- first name</td>
</tr>
<tr>
<td>- last name</td>
</tr>
<tr>
<td>- email</td>
</tr>
<tr>
<td>- phone</td>
</tr>
</tbody>
</table>

Figure C-5: Hierarchical, 1-shot, and 0-shot object models for a pet store application
C.6 Object models for a company management application

Prompt: I want a company management app where I can track my employees, their schedules, and their compensation.

Hierarchical (Score: 0.78)

<table>
<thead>
<tr>
<th>schedule</th>
<th>employee</th>
<th>company management app</th>
</tr>
</thead>
<tbody>
<tr>
<td>- title</td>
<td>- name</td>
<td>- employee</td>
</tr>
<tr>
<td>- description</td>
<td>- email address</td>
<td>- schedule</td>
</tr>
<tr>
<td>- start date</td>
<td>- phone number</td>
<td>- compensation</td>
</tr>
<tr>
<td>- end date</td>
<td>- schedule</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- compensation</td>
<td></td>
</tr>
</tbody>
</table>

1-shot (Score: 0.80)

<table>
<thead>
<tr>
<th>employee</th>
<th>schedule</th>
<th>pay</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- day</td>
<td>- day</td>
</tr>
<tr>
<td>- position</td>
<td>- time</td>
<td>- position</td>
</tr>
<tr>
<td>- schedule</td>
<td>- position</td>
<td>- hours</td>
</tr>
<tr>
<td>- pay</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0-shot (Score: 0.84)

<table>
<thead>
<tr>
<th>employee</th>
<th>schedule</th>
<th>compensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name</td>
<td>- date</td>
<td>- type</td>
</tr>
<tr>
<td>- contact info</td>
<td>- time</td>
<td>- amount</td>
</tr>
<tr>
<td>- schedule</td>
<td>- location</td>
<td>- date</td>
</tr>
</tbody>
</table>


