

# Understanding the prompting capabilities of pretrained language models

Alex Rashduni, Akshaj Tyagi

Advised by Karl Stratos

May 1, 2023

## **Abstract**

Pretrained language models have been shown to be highly effective at a variety of natural language processing tasks, but there is still much to learn about how to best prompt these models to achieve optimal performance. This thesis examines the prompting capabilities of pretrained language models, specifically focusing on the T0 model and ChatGPT. Through a comprehensive literature review, we explore the current state of research on prompting and its potential applications in natural language processing. We then benchmark the T0 model's performance in various datasets including BIG-Bench Lite, demonstrating its strengths and limitations. Following our evaluation of T0, we investigate ChatGPT's potential in few-shot semantic parsing, finding that it performs well in predicting structured representations of natural language queries with minimal training data. Finally, we evaluate ChatGPT's zero-shot software engineering and development capabilities, showing that it can generate syntactically correct and semantically relevant code snippets given natural language prompts. This thesis highlights the significance of prompting as a powerful technique for natural language processing, and the potential of pretrained language models in various domains.

# Contents

<b>Acknowledgement</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
<b>2 Background</b>	<b>6</b>
2.1 Transformers and Prompting . . . . .	6
2.1.1 Fundamentals of Transformers . . . . .	6
2.1.2 Theory of Prompting . . . . .	8
2.1.3 Co-Training and Prompts . . . . .	8
2.2 Computational Linguistics Applications . . . . .	10
2.2.1 Semantic Parsing . . . . .	10
2.2.2 Knowledge Graphs . . . . .	11
2.3 Applications in Mathematics . . . . .	12
2.3.1 Formal Theorem Provers . . . . .	12
2.3.2 Autoformalization . . . . .	13
2.4 Models and Datasets . . . . .	14
2.4.1 The T0 Model . . . . .	14
2.4.2 The ChatGPT Model . . . . .	16
2.4.3 The Codex Model . . . . .	17
2.4.4 The BIG-Bench Dataset . . . . .	17
<b>3 System Architecture</b>	<b>18</b>
3.1 GPU Clusters . . . . .	18
3.2 Parallelized Transformers . . . . .	19
3.3 DeepSpeed Launcher . . . . .	21
3.4 Inversion of Control . . . . .	21
<b>4 T0 Experiment Replication</b>	<b>21</b>
4.1 Experiment Setup . . . . .	21
4.2 Experiment Results . . . . .	22

<b>5</b>	<b>Evaluation of T0 over BIG-Bench Lite</b>	<b>22</b>
5.1	Experiment Setup . . . . .	22
5.2	Experiment Results . . . . .	23
<b>6</b>	<b>Few-Shot Semantic Parsing on T0 and ChatGPT</b>	<b>27</b>
6.1	Experiment Setup . . . . .	28
6.2	Experiment Results . . . . .	28
<b>7</b>	<b>ChatGPT Zero-Shot Software Engineering</b>	<b>29</b>
7.1	Experiment Setup . . . . .	31
7.2	Experiment Results . . . . .	31
<b>8</b>	<b>Open Problems</b>	<b>33</b>
	<b>References</b>	<b>34</b>
<b>A</b>	<b>Prompts for Semantic Parsers</b>	<b>36</b>
A.1	Few-shot Examples for Semantic Parsers . . . . .	36
A.2	Test Inputs for Semantic Parsers . . . . .	40
A.3	Analysis of ChatGPT Parsing Results . . . . .	41
<b>B</b>	<b>Prompts for Programming Language Development</b>	<b>48</b>
B.1	List of Prompts used in Development . . . . .	48
<b>C</b>	<b>Replicability of ChatGPT Prompting</b>	<b>50</b>

# Acknowledgement

We would like to express our gratitude to the individuals who have been instrumental in the completion of this capstone project.

First and foremost, we would like to thank our advisor, Prof. Karl Stratos, for his unwavering support, guidance, and encouragement throughout the course of this project. His vast knowledge, exceptional expertise, and attention to detail have been instrumental in shaping the direction of this work and in ensuring its quality. We are grateful for his countless hours of feedback, his willingness to discuss and brainstorm ideas, and his dedication to helping us grow and learn.

Prof. Stratos has been an incredible mentor, always available to discuss any questions or concerns we may have had, and offering insightful suggestions and feedback that helped us to push the boundaries of our research. He challenged us to think critically and to strive for excellence, and we are better researchers and writers for it. His generosity with his time, his expertise, and his insights, have been invaluable and greatly appreciated.

We feel truly fortunate to have had the opportunity to work with Prof. Stratos, and we are grateful for the knowledge and skills he has imparted on us during this capstone project. We hope to continue to learn from his example in the years to come.

We also would like to extend our heartfelt thanks to Prof. Sesh Venugopal, our Capstone Coordinator, for his continuous support and mentorship throughout the CS Honors Capstone program. His dedication to our learning and growth has been invaluable, and his willingness to provide us with the resources and opportunities necessary for the success of this project has been truly appreciated.

In addition, we also want to thank our fellow CS Honors Capstone colleagues, who have been an amazing source of inspiration, motivation, and support. Through our weekly meetings and discussions, we have had the pleasure of getting to know some truly amazing people who are passionate about computer science and are dedicated to making a positive impact in the world. We are grateful for the friendships and connections made through this program, and we look forward to seeing all the incredible things we will accomplish in the future.

Finally, we would like to express our appreciation to our family and friends, who have supported us throughout our academic journey and without whom this achievement would not have been possible.

# 1 Introduction

Recently, the field of natural language processing has been galvanized by the seemingly astounding capabilities of large-scale pretrained language models (PLMs) to solve unknown tasks (i.e., zero-shot learning) as long as they can be framed in natural language. This approach, also known as “prompting”, has generated a number of papers trying to use and better understand it, most being superficial applications and analyses which do not offer a satisfying explanation for why prompting works and how. In this thesis, we will thoroughly chart the current landscape of prompting prominent PLMs (e.g., GPT-4, T0) by systematic experiments on standard zero-shot and few-shot performance benchmarks (e.g., SuperGLUE, BIG-Bench, MMLU) and discuss our findings on the capabilities of zero-shot and few-shot prompting.

Transformer-based encoder-decoder models are prominent in large-scale PLMs due to their ability to work with encoded contexts and states, rather than focusing on the probability distribution of individual words and series of words. Furthermore, these PLMs are prominent in industry as well, since the ability to summarize, generate, and translate documents is important in a wide variety of applications, with some examples being market discipline analysis, advertisements tailored to the consumer, and virtual assistants.

With a further understanding of prompting, we hope to provide a framework for future developments in prompt engineering. There are a wide varieties of applications for which the advancement of this field proves to be important.

In academia, our findings could help natural language processing researchers develop the next generation of large language models. With information on what tasks create implicit learning for other tasks, researchers could create better training and zero-shot data sets for these models.

On the other hand, in industry, our findings could allow data scientists and quantitative researchers effectively leverage PLMs in a wider variety of applications. This could provide both a larger set of models at their disposal and allows previously-inapplicable PLMs to be introduced into co-training settings, providing more inertia and stability.

We begin this report with a brief literature review of all the machine learning concepts we had studied during the Capstone project affiliated with this thesis. Although not every topic explored yielded a complete experiment, we found little resources listing all of these topics in a single succinct document, so we decided to summarize them and provide some references here. This literature review is laid out in Section 2.

Over the course of our experiments, we replicated the results of the foundational T0 paper, [1]. In replicating the results found in the paper, we sought to discover the underlying reasons for its strong performance, and use it as a foundation for further experiments in prompt engineering. In this report, we lay out the architecture for running the T0 model in Section 3 and our findings in Section 4.

Continuing from the replication of [1], we further evaluated the T0 model on the BIG-Bench Lite datasets, a massive dataset intended to probe PLMs and extrapolate their future

capabilities. These tasks include everything from debugging and evaluation of computer code, binary classification of Russian misconceptions, symbol interpretation, and more. We lay out the findings of this experiment in Section 5.

Following our experiments in the evaluation of T0, we studied the capabilities of ChatGPT in semantic parsing. These studies were aimed to discover both the few-shot capabilities of the models, and aimed to extrapolate any underlying chain-of-thought reasoning that the models exhibited. We lay out our results for these experiments in Section 6.

After that, we continued by testing the capabilities of ChatGPT in software engineering. In this experiment, we tested whether or not ChatGPT was able to develop a novel programming language, given little specifications from the user. We determined the quality of the program based on the number of errors and the overall efficiency. We lay out our findings for this experiment in Section 7.

Lastly, we wrap up our thesis with a brief discussion on some of the open problems in prompting pretrained language models, laid out in Section 8.

## 2 Background

### 2.1 Transformers and Prompting

#### 2.1.1 Fundamentals of Transformers

Encoder-decoder transformers are a type of neural network architecture that are commonly used for sequence-to-sequence learning tasks, such as machine translation, text summarization, and speech recognition. This architecture is based on the innovations introduced by Vaswani et al. in [2] in 2017.

The encoder-decoder transformer consists of two main components: an encoder and a decoder. The encoder takes the input sequence and produces a set of representations for each element in the sequence. These representations capture the meaning of each element in the input sequence in a high-dimensional vector space. The decoder then takes these representations and generates an output sequence, one element at a time, by attending to the most relevant parts of the input sequence. Refer to Figure 1 for the full transformer architecture, as shown in [2].

The key innovation of the transformer architecture is the use of self-attention mechanisms, which allow the model to focus on the most relevant parts of the input sequence when generating the output sequence. Self-attention allows the model to learn contextual relationships between different elements in the input sequence, and to generate output sequences that are more accurate and coherent.

In [2], Vaswani et al introduce two types of attention: scaled dot-product attention and multi-head attention. The former consists of computing the attention function on a set of queries simultaneously, packed into a matrix  $Q$ , alongside keys and values packed together

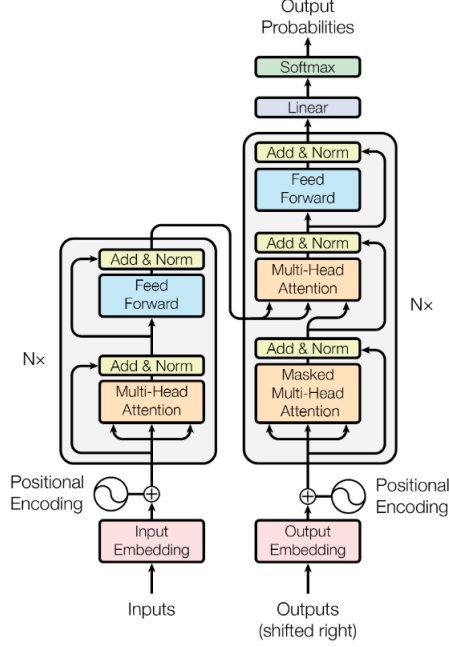


Figure 1: Transformer model architecture. Image credit to [2].

in matrices  $K$  and  $V$ , respectively. The scaled dot-product attention function is thus defined by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

The other attention algorithm introduced by Vaswani et al is the multi-head attention algorithm, which projects the queries, keys, and values from  $d_{\text{model}}$  dimensions into  $d_k$ ,  $d_k$ , and  $d_v$  dimensions, respectively. Each of the projected queries, keys, and values has the attention function then performed on it in parallel, yielding  $d_v$ -dimensional output values. The output values are then concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

As shown in [2], the mathematical formula for the multi-head attention algorithm is defined as follows:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \\ \text{where } \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \quad (2)$$

Leveraging the attention algorithms above, one of the main advantages of encoder-decoder transformers is their ability to handle variable-length input and output sequences. This makes them well-suited for tasks such as machine translation and text summarization, where the length of the input and output sequences can vary widely.

Encoder-decoder transformers have been used with great success in a wide range of natural language processing tasks, and they continue to be an active area of research in the field.

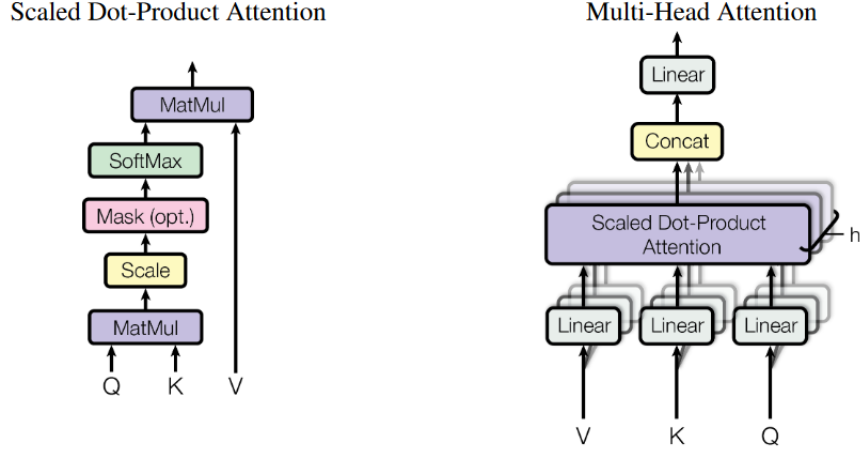


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Image and caption credit to [2].

Recent advances have focused on improving the efficiency and scalability of transformer models, as well as exploring new applications for these powerful neural network architectures.

### 2.1.2 Theory of Prompting

Prompting is defined as the embedding of a function or operation (a prompt) within an input, such as “Translate [sentence] into German.” With the proper selection of a prompt (called prompt engineering), it may be possible for a PLM to produce a desired output for some task that it would otherwise find challenging. Figure 3 features two examples of prompting in computer vision.

In [3], Jin et al lay out some foundational theory and principles of prompting, as well as the power available to the paradigm. In this paper, it was found that tuning a prompt provides a greater benefit to model accuracy than millions of additional parameters. The paper focused on a computer vision setting, but evidence shows that the relation holds for the natural language setting, as well.

Furthermore, it was found that prompt tuning has a very strong impact on zero-shot performance, as novel prompts more similar to learned prompts are inherently easier to learn. The reasons for this remain a mystery, as there is still active discussion, debate, and research into why prompting works – it’s just empirically proven that it does work.

### 2.1.3 Co-Training and Prompts

Co-training is a machine learning technique that uses two or more complementary models to improve the performance of each other. The basic idea behind co-training is to train each model on a different subset of the data, and then use the outputs of each model to provide additional labeled examples for the other model to learn from. This process is iterative



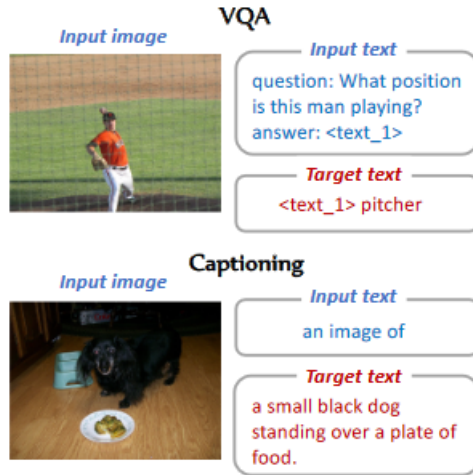


Figure 3: Examples of prompting in a computer vision setting (Jin et al, 2022). This form of prompting requires a conversion from image to characters, or a form of multimodal input into the model. Examples include visual question answering (top) and captioning (bottom).

and continues until the performance of both models converges to a satisfactory level. The co-training algorithm, as laid out in [4], is shown in Figure 4.

Co-training is particularly useful in situations where there is a limited amount of labeled data available, as it allows for the use of unlabeled data to improve the performance of the models. One common application of co-training is in natural language processing, where it is used to improve the accuracy of text classification and information extraction tasks.

Co-training can be applied to a wide range of machine learning models, including decision trees, neural networks, and support vector machines. However, it requires careful selection of the subsets of data used to train each model, as well as the selection of appropriate features and the regularization parameters for each model. In addition, co-training can be sensitive to noise and errors in the unlabeled data, which can lead to overfitting and reduced performance.

Despite these challenges, co-training has been shown to be effective in a variety of applications, including text classification, speech recognition, and image classification. Ongoing research in co-training is focused on developing more sophisticated algorithms that can handle more complex data structures and improve the scalability of the technique. For reference, [4] lays out the usage of co-training in the development of large language models and showcases few-shot learning and zero-shot learning results of various ensembles of models.

Although this has massive improvements for the procurement and usage of massive data sets, it also has the drawback of being susceptible to generating wrong labels in circumstances where a large number of models agree on said labels (i.e., by pure chance of random initialization).

---

**Algorithm 1** Co-training algorithm

---

```

input  $\mathcal{U} = \{x_n\}_{n=1}^U$  unlabeled examples
input  $\{(x_j, y_j)\}_{j=1}^k$  labeled examples (optional)
input initial coverage  $\beta$ , coverage increase  $\beta'$ 
 $h_0 \leftarrow \text{InitClassifier}(\phi_0)$ 
for  $t$  in  $\{0, \dots, T-1\}$  do
   $\tilde{\beta} \leftarrow \beta + t\beta'$ 
  // GetConfData* defined in Algorithms 2, 3
   $L_0^t \leftarrow \text{GetConfData}^*(\mathcal{U}; h_0, \phi_0, \tilde{\beta})$ 
   $h_1 \leftarrow \text{Train}(\phi_1, L_0^t)$ 
   $L_1^t \leftarrow \text{GetConfData}^*(\mathcal{U}; h_1, \phi_1, \tilde{\beta})$ 
   $h_0 \leftarrow \text{Train}(\phi_0, L_1^t)$ 
end for
return  $(h_0, h_1)$ 

```

---

Figure 4: The co-training algorithm (Lang et al, 2022).

## 2.2 Computational Linguistics Applications

### 2.2.1 Semantic Parsing

Semantic parsing is a natural language processing technique that aims to transform a sentence or a phrase into a structured representation that can be easily understood by a computer. The resulting structured representation, often in the form of a semantic graph or a logical form, can then be used for various downstream tasks such as question answering, information retrieval, and dialogue systems.

The process of semantic parsing involves several steps, including part-of-speech tagging, named entity recognition, dependency parsing, and semantic role labeling. These techniques are used to extract the relevant syntactic and semantic information from the input sentence or phrase and to convert it into a formal representation that can be used for further processing.

There are various approaches to semantic parsing, including rule-based methods, statistical methods, and neural network-based methods. Rule-based methods rely on a set of handcrafted rules to map natural language expressions to their corresponding formal representations. Statistical methods, on the other hand, use machine learning techniques to learn the mapping between natural language expressions and formal representations from a large corpus of annotated data. Neural network-based methods, such as sequence-to-sequence models and transformer-based models, have shown promising results in recent years, achieving state-of-the-art performance on several benchmark datasets.

One recent paper showcasing the capabilities of language models in semantic parsing is [5], where Shin et al used few-shot prompting on constrained language models to generate structured meaning representations of natural language inputs. In the paper, Shin et al experimented with the 175-billion parameter GPT-3 model, priming the model through dynamic prompt creation and fine-tuning, and then using constrained decoding, ensuring well-formed

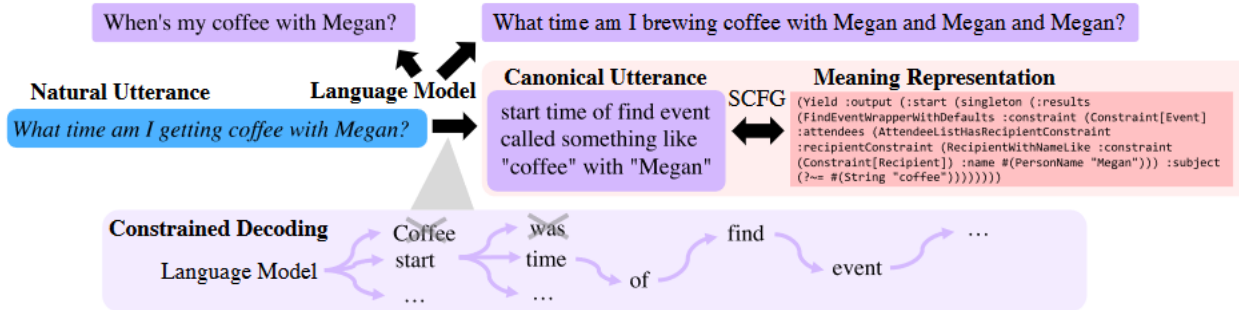


Figure 5: The workflow proposed by Shin et al in [5]. Given a few examples (not shown) and a natural user utterance (blue, italic), a pretrained language model generates paraphrased utterances (purple). A grammar constrains the search over paraphrases to only canonical utterances, and the highest-scoring canonical paraphrase is mechanically converted to a task-specific meaning representation (pink). Image and caption credit to [5].

output under target representations. The workflow for semantic parsing proposed in [5] is displayed in Figure 5.

Semantic parsing has a wide range of applications, including question answering, dialogue systems, and semantic search. It has also been applied to various domains, such as healthcare, finance, and e-commerce. However, there are still several challenges in semantic parsing, such as handling lexical and syntactic variability, dealing with ambiguous language, and handling out-of-vocabulary words. Future research in semantic parsing will likely focus on developing more robust and scalable methods that can handle these challenges and enable more sophisticated natural language processing applications.

### 2.2.2 Knowledge Graphs

Knowledge graphs are a structured representation of knowledge that allow for efficient storage, retrieval, and analysis of information. They are commonly used in artificial intelligence and machine learning applications, as they provide a way to represent complex relationships between entities and concepts in a way that can be easily processed by computers.

A knowledge graph consists of nodes, which represent entities, and edges, which represent the relationships between entities. Nodes can represent a wide range of entities, such as people, places, organizations, events, and concepts, and edges can represent a wide range of relationships, such as “is a type of”, “is located in”, “is married to”, and “causes”.

One of the main advantages of knowledge graphs is that they allow for the integration of information from multiple sources, such as databases, text documents, and social media. This integration allows for the discovery of new relationships and insights that may not be immediately apparent from any single source of information.

Knowledge graphs have many applications in artificial intelligence and machine learning, including:

**Recommendation systems.** Knowledge graphs can be used to represent user preferences and item attributes, allowing for the generation of personalized recommendations.

**Question answering.** Knowledge graphs can be used to represent the relationships between entities and concepts, allowing for the generation of answers to natural language questions.

**Natural language processing.** Knowledge graphs can be used to represent the meaning of text in a structured format, allowing for the development of more sophisticated natural language processing models.

**Data integration.** Knowledge graphs can be used to integrate data from multiple sources, allowing for the discovery of new relationships and insights.

**Semantic search.** Knowledge graphs can be used to represent the meaning of search queries and documents, allowing for more accurate and relevant search results.

There are many tools and frameworks available for building and working with knowledge graphs, including Apache Jena, Neo4j, and RDFLib. Ongoing research in knowledge graphs is focused on developing more sophisticated techniques for knowledge representation, reasoning, and learning.

## 2.3 Applications in Mathematics

### 2.3.1 Formal Theorem Provers

A formal theorem prover is a computer program that is used to verify mathematical or logical proofs. The goal of a theorem prover is to provide a rigorous and automated method for verifying the correctness of a proof, which is essential in many areas of mathematics, computer science, and engineering.

Formal theorem provers work by using a set of formal rules and axioms to construct a proof step by step. The proof is then checked by the theorem prover to ensure that it is correct according to the rules of the system. If the proof is correct, the theorem prover will output a certificate of correctness that can be independently verified.

There are many different types of formal theorem provers, ranging from simple systems that can verify basic mathematical proofs, to highly sophisticated systems that can handle complex mathematical structures and reasoning. Some popular theorem provers include Coq, Isabelle, HOL Light, and Mizar.

The use of formal theorem provers has many benefits in both research and industry. For example, in research, formal theorem provers can be used to discover new theorems and to rigorously verify the correctness of new mathematical or logical results. In industry, formal theorem provers can be used to verify the correctness of software and hardware designs, ensuring that they are free from errors that could lead to system failures or security vulnerabilities.

However, the use of formal theorem provers also has some challenges. One of the main challenges is the complexity of the proof process, which can be time-consuming and requires a high level of expertise. In addition, the formalization of mathematical or logical concepts can be a challenging task that requires careful attention to detail and a deep understanding of the subject matter.

Despite current challenges, formal theorem provers continue to be an important area of research in mathematics, computer science, and engineering, and are expected to play an increasingly important role in the development of new technologies and scientific discoveries.

### 2.3.2 Autoformalization

Autoformalization refers to the process of automatically formalizing natural language statements into a formal language, such as first-order logic. The goal of autoformalization is to enable automated reasoning and inference on natural language statements, which can help to facilitate the development of advanced artificial intelligence systems.

Autoformalization involves several steps, including syntactic parsing, semantic analysis, and logical translation. The first step involves breaking down a natural language statement into its constituent parts, such as nouns, verbs, and adjectives, using techniques from natural language processing (NLP). The second step involves analyzing the meaning of the statement to identify the relationships between its constituent parts, using techniques from computational semantics. The final step involves translating the statement into a formal language, such as first-order logic, using rules and algorithms that capture the logical structure of the statement.

Autoformalization has many applications in formal theorem provers, including the following:

**Automated theorem proving.** Autoformalization can be used to automate the process of proving mathematical theorems. Given an informal statement of a theorem, an autoformalization system can automatically generate a formal statement and then use a theorem prover to search for a proof. This topic was covered extensively by Jiang et al in [6].

**Mathematical database management.** Autoformalization can be used to create and maintain large databases of formal mathematical knowledge. Such databases could be used to support automated reasoning systems, or to provide a common foundation for mathematics research.

**Verification of mathematical software.** Autoformalization can be used to verify the correctness of mathematical software. By generating a formal specification of the software's behavior, it is possible to use a theorem prover to prove that the software always behaves as expected.

**Discovery of new theorems.** Autoformalization can be used to discover new mathematical theorems. By searching for patterns in existing mathematical knowledge and

generating formal statements of conjectures, autoformalization systems can help mathematicians identify new areas of research and make new discoveries.

Despite its potential benefits, autoformalization is still a challenging research area that requires the development of advanced NLP and computational semantics techniques, as well as formal logic systems that can support the representation and inference of complex knowledge structures. Ongoing research in autoformalization is focused on developing more sophisticated techniques for handling natural language ambiguity, reasoning with incomplete and uncertain information, and integrating formal and informal knowledge representations.

## 2.4 Models and Datasets

### 2.4.1 The T0 Model

In [1], Sanh et al introduced the T0 model, a large conditional language model trained on many large datasets. T0 has been shown to perform well in zero-shot settings, outperforming larger models such as the GPT-3 model and its derivatives in zero-shot tasks, as shown in Figure 6. This model is a big step from GPT-3, as it is small enough to fit in most GPUs, with its largest derivative being 42 GB, while GPT-3 will put a burden on even dedicated (albeit somewhat small) clusters of GPUs, with a size of over 800 GB. Additionally, there is no known shortfall in accuracy when comparing T0 to GPT-3, and T0’s smaller size makes it naturally faster, since computation over transformers is dependent on size of the model.

Furthermore, due to the T0 model’s training as a multitask encoder-decoder model, the model is highly versatile, able to classify, generate, parse, and derive meaning from not only a large variety of texts, but also a large variety of text formats (i.e., prompting). Through the encoder-decoder setup, T0 is able to place attention on individual parts of a sentence, create a “state” from phrases, and thus condition on the state to find most-likely or otherwise candidate future states. It’s through this translation from a word-based problem to a conditional probability distribution over states that T0 is able to accomplish its high performance with so few parameters.

Figure 7 showcases the model and prompt format for the T0 model. T0 is an encoder-decoder model that consumes text inputs and produces target responses. It is trained on a multitask mixture of NLP datasets partitioned into different tasks. Each dataset is associated with multiple prompt templates that are used to format example instances to input and target pairs (Sanh et al, 2022).

Figure 8 shows the datasets which were used to train and test the T0 model, and thus serve as the bedrock for the model’s success. The incredible diversity among the datasets allow for T0’s generalizability and zero-shot performance, since T0’s necessity to train on many types of datasets and data formatting generalizes it to many more types of datasets. All of these datasets are available on HuggingFace, an open-source repository for models, datasets, and programs revolving around machine learning research. Access to such datasets enables familiarization with the specifics of the testing dataset’s tasks such as sentence completion,

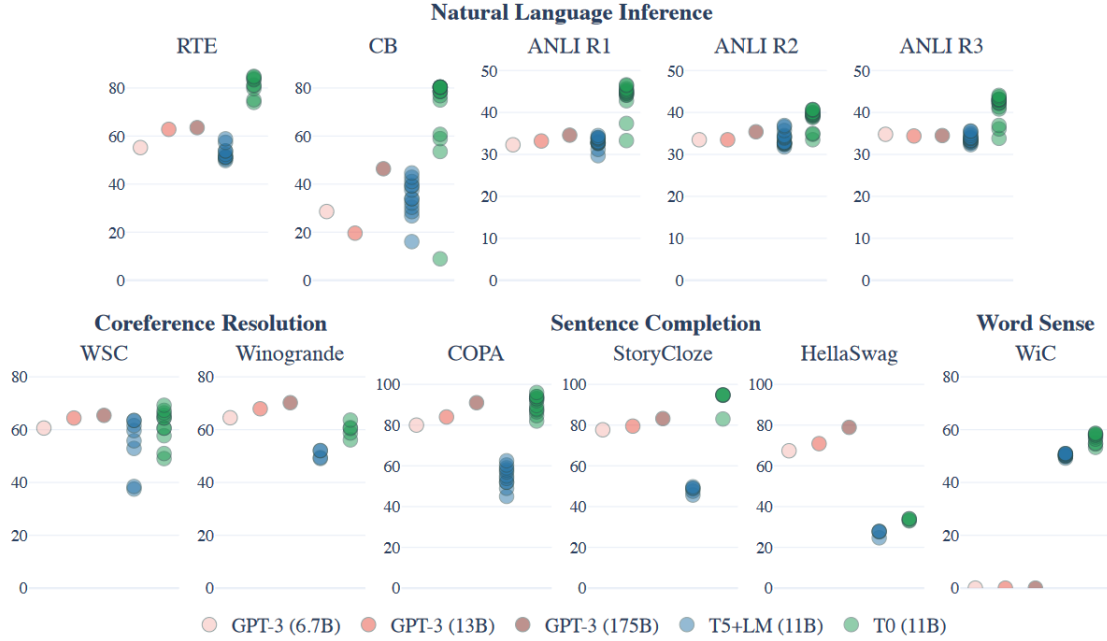


Figure 6: Results for T0 task generalization experiments to GPT-3, from [7]. Each dot is the performance of one evaluation prompt. The baseline T5+LM model is the same as T0 except without multitask prompted training. GPT-3 only reports a single prompt for each dataset. Image and caption credit to Sanh et al, in [1].

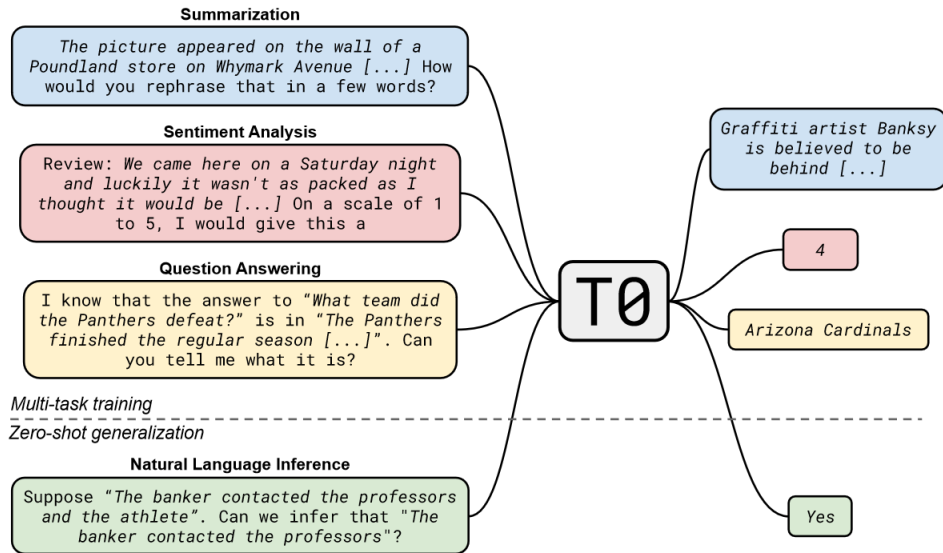


Figure 7: The model and prompt format for the T0 model. After training on a diverse mixture of tasks (top), the model is evaluated on zero-shot generalization to tasks that are not seen during training (bottom). Italics indicate the inserted fields from the raw example data. Image and caption credit to Sanh et al, in [1].

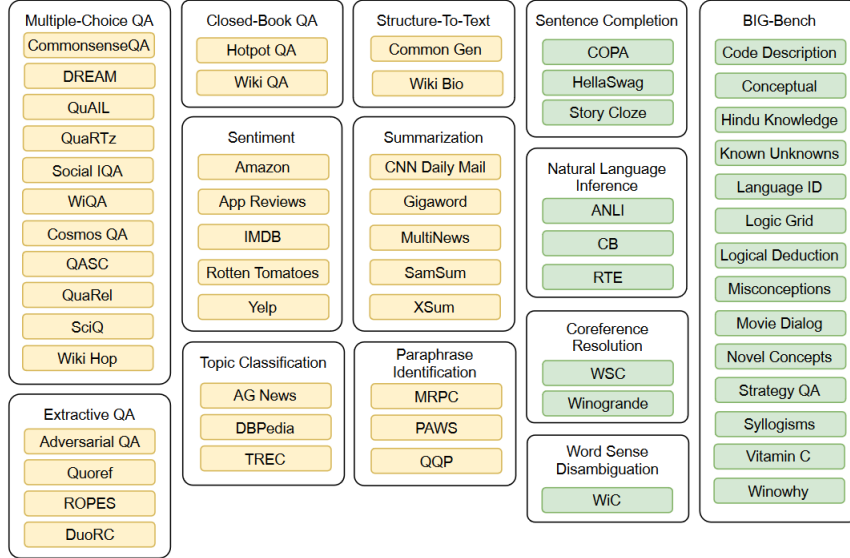


Figure 8: T0 datasets and task taxonomy. T0+ and T0++ are trained on additional datasets. See Table 5 in [1] for the full list. The datasets in yellow are training datasets contained in [8] and the datasets in green are testing datasets contained in [9].

natural language inference, coreference resolution, and word sense disambiguation.

#### 2.4.2 The ChatGPT Model

OpenAI’s ChatGPT is an advanced conversational AI language model that is based on the GPT architecture. It is designed to generate human-like responses to natural language input, making it a powerful tool for creating chatbots and other conversational applications.

ChatGPT is trained on a massive dataset of text data, including social media conversations, chat logs, and other sources of natural language text. The model is pretrained on this data to generate high-quality language outputs that are both syntactically and semantically correct, and that capture the nuances and subtleties of human conversation.

One of the key features of ChatGPT is its ability to generate responses that are contextually relevant to the input. The model uses a technique known as “attention” to focus on the most important parts of the input text, and then generates a response that is tailored to the specific context of the conversation. This allows ChatGPT to generate responses that are both coherent and relevant, and that can adapt to changing conversational dynamics.

ChatGPT has many potential applications in customer service, chatbots, and other conversational applications. For example, due to its respectable expertise in a wide variety of technical fields, it can be used to generate the first-draft to a Background section of an undergraduate thesis.

However, there are also concerns about the potential misuse of ChatGPT, particularly in the context of creating deceptive or malicious chatbots. OpenAI has implemented safeguards to



prevent the use of ChatGPT for malicious purposes, and has restricted access to the model to a limited number of partners and developers.

### **2.4.3 The Codex Model**

OpenAI’s Codex model, introduced by Brown et al in [11], is an advanced AI language model that has been trained on a large dataset of code snippets and natural language text. Codex is designed to be a highly versatile tool for generating code, with the ability to understand and translate natural language descriptions into functioning code across a wide range of programming languages.

Codex is based on OpenAI’s GPT architecture, which is a type of deep neural network that has been trained on massive amounts of text data to generate high-quality language outputs. Codex has been pretrained on a dataset of over 14 million code examples and related natural language text, which includes documentation, comments, and other programming-related content. The pre-training process allows Codex to generate code that is both syntactically and semantically correct, and to recognize and incorporate common programming patterns and idioms.

One of the key features of Codex is its ability to perform context-aware code completion, which means that it can suggest and complete code based on the surrounding code and the programmer’s intentions. Codex can also generate entire functions or programs from natural language descriptions, which can be useful for automating repetitive programming tasks or generating boilerplate code.

Codex has many potential applications in software development and other programming-related tasks, such as debugging, testing, and code optimization. For example, Codex can be used to generate code that is more efficient or optimized for specific hardware or software environments, or to automate the creation of test cases or simulations.

However, there are also concerns about the potential misuse of Codex, particularly in the context of automated code generation and its impact on the job market for software developers. OpenAI has implemented safeguards to prevent the use of Codex for malicious purposes and has restricted access to the model to a limited number of partners and developers.

### **2.4.4 The BIG-Bench Dataset**

The BIG-Bench dataset is a large-scale benchmark dataset designed to evaluate the performance of natural language processing models on a wide range of tasks. The dataset was released in 2020 by researchers from the Allen Institute for AI (AI2) in [12], and contains over 10 trillion tokens, making it one of the largest natural language processing datasets available.

The BIG-Bench dataset is designed to be more comprehensive and diverse than other benchmark datasets, such as GLUE and SuperGLUE, which focus on a smaller set of specific tasks. The BIG-Bench dataset includes a wide range of tasks, including text classification,

sentiment analysis, question answering, and summarization, among others. In addition, the dataset includes variations of each task, such as multiple-choice and cloze-style questions, to evaluate the performance of models on different types of inputs.

The BIG-Bench dataset is also designed to be more challenging than other benchmark datasets, with a focus on capturing the nuances and complexities of natural language. The dataset includes examples of language that are difficult for current natural language processing models to handle, such as sarcasm, irony, and metaphor.

To evaluate the performance of models on the BIG-Bench dataset, AI2 has developed a suite of evaluation metrics, including accuracy, F1 score, and perplexity. In addition, AI2 has released a benchmark leaderboard that allows researchers to compare the performance of their models to state-of-the-art models on the dataset.

The release of the BIG-Bench dataset has generated significant interest in the natural language processing community, with researchers using the dataset to develop and evaluate new models and techniques. The dataset is expected to be a valuable resource for advancing the field of natural language processing and developing more sophisticated models for a wide range of applications.

## 3 System Architecture

To account for the large size of the T0++ model and the highly parallelized computation being performed in this experiment, we decided to make use of some key hardware and software to run our experiments efficiently. Namely, we are using a cluster of GPUs and a transformer parallelization library.

### 3.1 GPU Clusters

The GPU clusters in question are named *one* and *two*, both on the domain cs.rutgers.edu. *one* is comprised of eight Quadro RTX 6000s, each of which having 24 GB of memory to work with. *two*, on the other hand, is comprised of eight NVIDIA A100 PCIe GPU accelerators, each of which having 40 GB. We decided to use GPUs since our experiments are highly parallelizable in two aspects:

- Parameters are independent of each other, meaning we can map independent strips of the large language models onto the different machines for increased runtime.
- Test cases and examples themselves are also independent, which lends itself to large-scale parallelization, as well.

With these GPUs in hand, we use the `parallelformers` library to distribute our large language models across the GPUs.

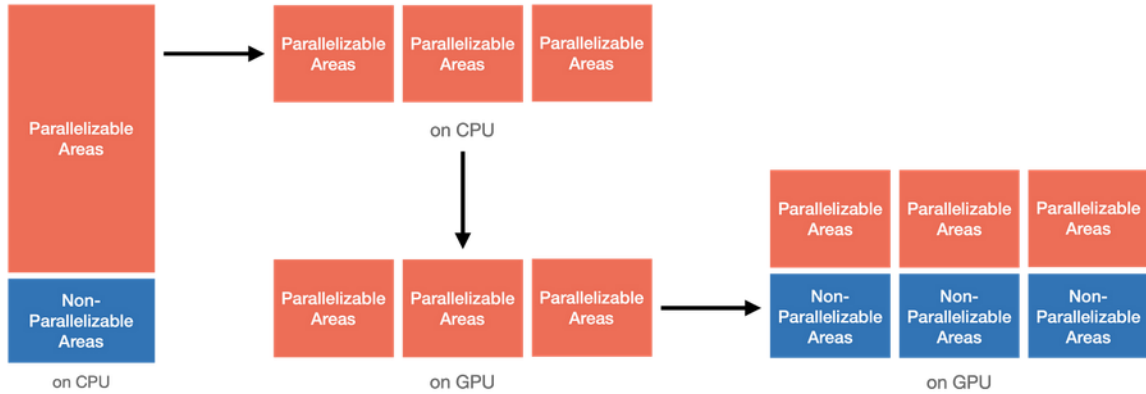


Figure 9: Parallelization of a model using the parallelformers library. Image credit to [13]

### 3.2 Parallelized Transformers

Naturally, the T0++ model is too large for the GPUs, since two.cs (the larger of the two clusters) has only 40 GB memory per GPU, and the model itself is 42 GB. Thus, due to the fact that individual parameters are independent of one another during prediction and label generation, we decided to use a transformer parallelization library to overcome this issue. We used the parallelformers library, which efficiently maps independent blocks of the parameter “matrix” to individual GPUs in the cluster, and then similarly partitions test inputs. In practice, this made large-scale experimentation much more efficient and feasible, since runtime per experiment fell from roughly ten to fifteen minutes per example, to around one or two, on average.

The remainder of this section covers conceptual topics about transformer parallelization, which is elaborated on in “Parallelformers: Journey to deploying big models\_TUNiB,” an article by TUNiB. Figures throughout this section are from them, as well.

Figure 9 shows how parallelformers divides a model across GPUs. First, the parallelizable components of a model are divided on the CPU, according to the number of GPUs available. This is done because if the model was stored on the GPUs and then divided, the GPUs would not have enough memory to support it. From there, the parallelizable components of the model are then sent to the GPUs, alongside the non-parallelizable components, thus resulting in a proper division.

Building upon the partitioned model, parallelformers then computes on the parallelized model using a process called “method hijacking,” as shown in Figure 10. Method hijacking works by shifting control flow away from the model whenever it is called, cushioning the code with commands necessary for moving data to GPUs, consistent distributed computing (i.e., mutual exclusions), and parallelization (i.e., output queues). The setup of a main process and subprocesses is derived from the DeepSpeed Launcher.

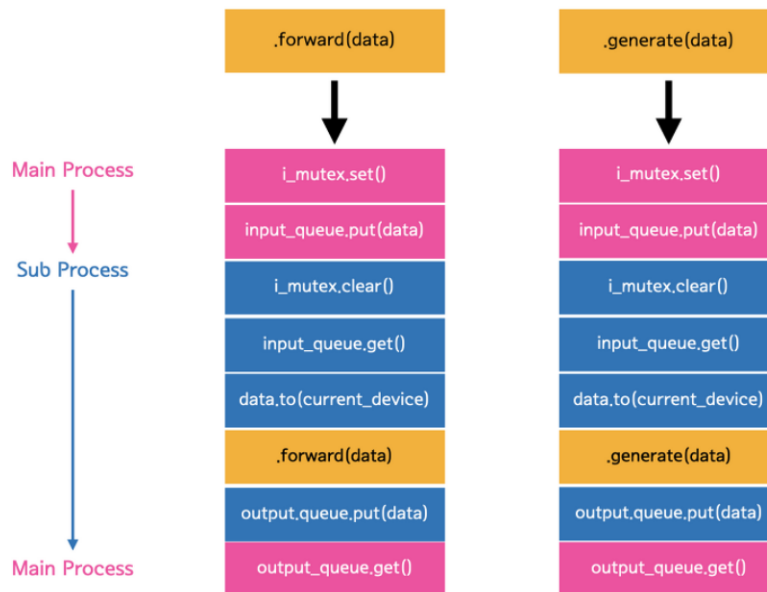


Figure 10: Method hijacking in the parallelfomers library. Image credit to [13]

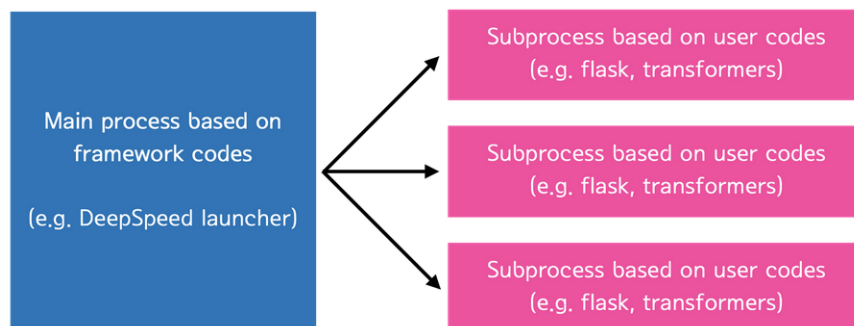


Figure 11: Control flow under the DeepSpeed Launcher. Image credit to [13]

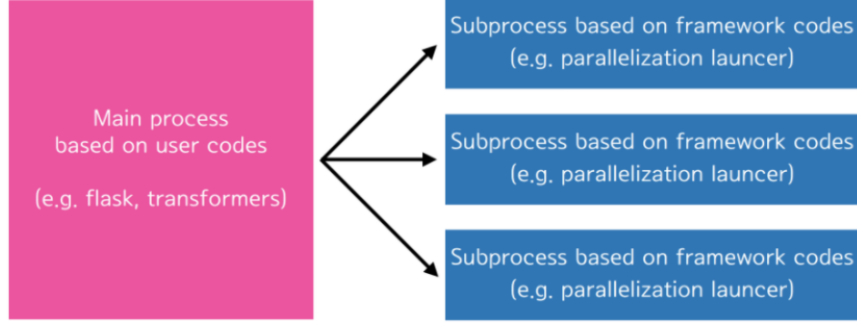


Figure 12: Inversion of process control in the DeepSpeed Launcher. Image credit to [13]

### 3.3 DeepSpeed Launcher

DeepSpeed is an open source deep learning optimization library for PyTorch. The library is designed to reduce computing power and memory use and to train large distributed models with better parallelism on existing computer hardware. DeepSpeed is optimized for low latency, high throughput training. It includes the Zero Redundancy Optimizer (ZeRO) for training models with multi-billion parameters. Features include mixed precision training, single-GPU, multi-GPU, and multi-node training as well as custom model parallelism (See Figure 11). In order for deepspeed to execute, it is important to understand a key concept that is used to create such a library- the process of inversion of control.

### 3.4 Inversion of Control

Inversion of control is a concept in which the application framework calls the user’s code. This is the opposite of the normal process in which the user’s code calls the application framework. By using this, we reversed the traditional processing method so that we could call the framework code from the user’s code several times, simultaneously. By applying the ‘Inversion of control’ method, it’s possible to execute only parts of the user’s code that need to run multiple times. This prevents repeated loading of the same model. As a result, we were able to successfully deploy the parallelized models to the web server (See Figure 12).

## 4 T0 Experiment Replication

### 4.1 Experiment Setup

In their paper, Sanh et al evaluated zero-shot generalization on 11 datasets in 4 held-out NLP tasks including natural language inference, coreference, word sense disambiguation, and sentence completion, as well as 14 novel tasks from BIG-Bench. The performance in both [1] and in our replication is based on the validation splits, and every dataset uses accuracy as their metric. In order to maintain consistency with Sanh et al’s results, we used the

RTE Task	Accuracy
MNLI crowdsource	63.54%
guaranteed true	68.95%
can we infer	59.57%
GPT-3 style	61.37%
does this imply	64.62%
should assume	66.43%
does it follow that	61.73%
based on the previous passage	68.23%
justified in saying	61.01%
must be true	70.04%

Table 1: Accuracy of the T0 (3B parameter) model on the RTE tasks.

	Replicated	Stated
Median	64.08%	64.08%
Mean	64.55%	64.55%
Max	70.04%	

Table 2: Statistics of the T0 (3B parameter) model on the RTE tasks, comparing the replication results of Table 1 to the stated results in [1].

preprocessing laid out in [14] and use [15] for the initialization of our prompts. Furthermore, to support the running of the larger versions of the T0 model, we parallelized it as laid out in Section 3.

## 4.2 Experiment Results

Through our experiments on T0, we found that using the same preprocessing methods, we received the same results as [1]. Refer to Table 1 for the accuracy of the 3 billion (3B) parameter version of T0 over the RTE tasks, and refer to Table 2 for a statistical comparison between our results and the results of [1]. Note that the data in Table 1 was not made available in [1], meaning that a comparison of statistics is the only way to ensure proper replication, as of now.

# 5 Evaluation of T0 over BIG-Bench Lite

## 5.1 Experiment Setup

Following our replication of [1], we decided to continue testing the capabilities of T0 on a variety of other datasets. Among these was the BIG-Bench dataset, which as mentioned

BIG-Bench Task	Accuracy	Random
formal fallacies syllogisms negation	52.64%	50.00%
conceptual combinations	52.63%	25.00%
known unknowns	50.00%	50.00%
winowhy	44.76%	50.00%
logical deduction	41.67%	33.33%
play dialog same or different	41.26%	50.00%
strange stories	41.18%	25.00%
bbq lite json	38.79%	33.33%
code line description	37.50%	25.00%
hindu knowledge	34.29%	25.00%
novel concepts	25.00%	10.00%
emoji movie	15.00%	20.00%
vitaminc fact verification	58.70%	33.33%
symbol intepretation	25.10%	20.00%
language identification	13.80%	9.09%
misconceptions russian	N/A	N/A

Table 3: Accuracy of the T0 (3B parameter) model on the BIG-Bench Lite tasks.

in Section 2, is a large-scale benchmark dataset to evaluate performance on a wide variety of tasks. Specifically, due to limitations in time and computing resources, we chose to evaluate the model on the BIG-Bench Lite subset. To ensure continuity with the experiments laid out in Section 4, allowing us to compare the results of this experiment to the results in that one, we chose to preserve the preprocessing set out by [14].<sup>1</sup>

## 5.2 Experiment Results

Through our experiments, we found that T0’s performance over the BIG-Bench Lite tasks was very lopsided, and much poorer than it was in the previous experiment. As shown in Table 3, the 3B parameter version of T0, for example, had its best performance over the BIG-Bench Lite tasks fall short of even its worst RTE tasks.

From our results, we found that T0 struggles with tasks where it needs to evaluate special characters (e.g, emojis) that normally do not appear in training corpora, or when handling hallucinations, as is the case with the winowhy task. Despite that, we found that T0 performs remarkably with the conceptual combinations, logical deduction, code line description, novel concepts, and VitaminC tasks, so we will briefly discuss each of these tasks.

<sup>1</sup>This unfortunately led to misconceptions russian having no results due to the fact that the preprocessing in [14] does not support Russian characters.

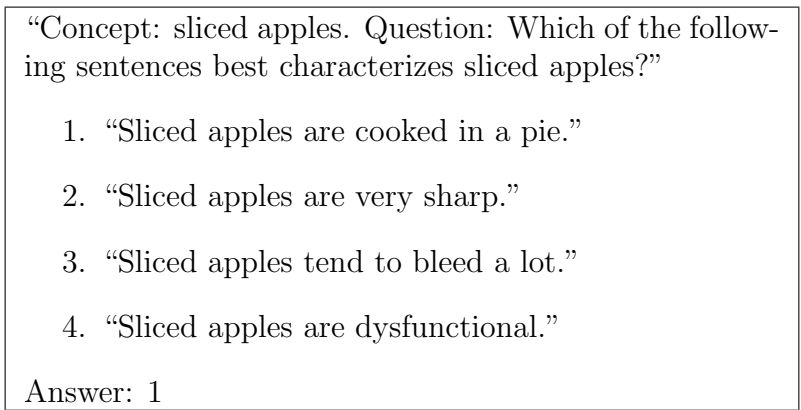


Figure 13: Example from the conceptual combinations task.

**Conceptual Combinations & Novel Concepts**

The conceptual combinations and novel concepts tasks involve the evaluated model being asked a question that requires them to select a sentence that connects two concepts together in the most logical manner. An example from each of these tasks is shown in Figure 13 and 14.

In the conceptual combinations example, the model is asked whether sliced apples are more common with pies, sharpness, bleeding, or functionality. Obviously, they are most associated with pies. This task is used to test a model’s understanding of proper word association, made evident by the fact that the other words used to describe sliced apples (e.g., sharp, bleed) have more to do with slicing than with apples.

Meanwhile, in the novel concepts task, the model is asked to find a similarity between three items, given a sequence of ten options. In the example, rock climbing, elevators, and helicopters all move vertically.

**Logical Deduction**

The logical deduction task focuses on riddles and, as the name implies, logic puzzles. This is best exemplified by the example in Figure 15, where the model is given the positions of three books relative to each other, and then asks which of the three books is leftmost.

The purpose of this task is to test a model’s ability to extract information from text that is implied, but not said explicitly. Testing this kind of ability is important since in large corpora of training data, models are bound to encounter metaphors, idioms, and other forms of “uncertain” speech that they will need to parse and understand, especially in the humanities. This is backed up by [16], where Hendrycks et al showcased models’ propensity to underperform in humanities tasks.



“What do the following have in common? 1) rock climbing 2) elevators 3) helicopters”

1. “They all involve people.”
2. “They all involve vehicles.”
3. “They all involve ropes.”
4. “They all involve vertical movement.”
5. “They all involve horizontal movement.”
6. “They all involve going up.”
7. “They all involve going sideways.”
8. “They all involve machines.”
9. “They all involve rocks.”
10. “They all involve flying.”

Answer: 4

Figure 14: Example from the novel concepts task.

“On a shelf, there are three books: a black book, an orange book, and a blue book. The blue book is to the right of the orange book. The orange book is to the right of the black book.”

1. “The black book is the leftmost.”
2. “The orange book is the leftmost.”
3. “The blue book is the leftmost.”

Answer: 1

Figure 15: Example from the logical deduction task.

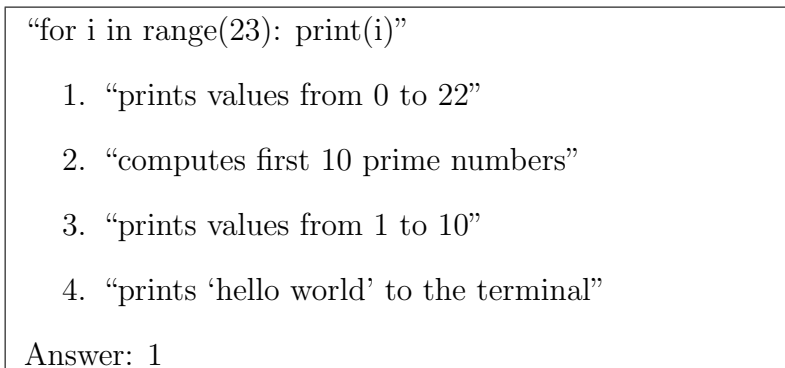


Figure 16: Example from the code line description task.

## Code Line Description

The code line description task encompasses problems relating to the output of computer programs, mostly written in Python-like pseudo-code, as shown in Figure 16. This task is used to test a model’s procedural abilities, rather than its word association or other abilities. Remarkably, this result is similar to the 40.2% that a UnifiedQA model with a T5 text-to-text backbone receiving in [16] on highly-procedural STEM tasks. Both evaluations consisted of multiple choice problems with 25.0% random probability, and received similar results, indicating that these results are fuelled by the T5 backbone of the T0 model.

## VitaminC Fact Verification

The VitaminC task, similar to the logical deduction task, involves the evaluated model being tested logic puzzles while also testing for potential hallucinations in the model output. This is showcased by Figure 17, where the model is asked to determine whether a number is guaranteed to be in a given range given some information on the range the number is known to be within.

In the example, the model is told that Westlife had more than 23 million combined sales in the UK, then is asked if it had less than 23.5 million sales. Evidently, “more than 23 million” does not necessarily imply “23.5 million sales,” so the model cannot say with any certainty whether Westlife sold less than 23.5 million. In the example, both the options “True” and “False” are provided alongside “Neither,” since former two check for boolean truth, while the latter is used to check for model hallucinations and false unknowns.

T0’s performance in this task is surprisingly high, since the model performed at a level not significantly different from randomness in the known unknowns dataset. The high score on this task thus indicates that T0 has a high level of mathematical logic, since that would be required to offset the 50% hallucination rate.

“Westlife: According to the British Phonographic Industry ( BPI ) , Westlife has been certified for 13 million albums and 9.8 million singles , with a total of more than 23 million combined sales in the UK .

Claim: Westlife made under 23.5 million sales in the UK .”

1. “True”
2. “False”
3. “Neither”

Answer: 3

Figure 17: Example from the VitaminC Fact Verification task.

## Similarities and Reasons for Model Success

From the above, it is evident that T0 performs highly on tasks where logic and word association come into play, indicating that T0 has reasonable “common sense.” This is likely due to the fact that the T5 model, which is the backbone for the T0 model, was pretrained on a massive corpus of data from Common Crawl, a web archive that provides “web extracted text” by removing markup and other non-text content from scraped HTML files of websites. This corpus is massive, producing 20 TB of scraped text data each month. Therefore, it is highly probable that T5 would pick up on “common sense” from this training data. Furthermore, it is plausible that programming knowledge was picked up due to the prevalence of programming tutorials on the Internet, but the reason for T5-based models struggling with other procedural topics, such as arithmetic, remains unknown.

## 6 Few-Shot Semantic Parsing on T0 and ChatGPT

After completing our benchmark evaluations of the T0 model, we decided to explore the capabilities and applications of pretrained language models in a more open-ended setting, determining their few-shot capabilities in semantic parsing of natural language text. Inspired by [17], a blog post on Wordpress from UChicago professor David McAllester on few-shot semantic parsing in GPT-3, we decided to explore the capabilities of this approach in ChatGPT. Studying this was of particular interest, since it would answer the following questions.

**ChatGPT vs GPT-3.** What are the capabilities of ChatGPT compared to GPT-3 in a few-shot setting? It’s advertised that ChatGPT performs better than GPT-3 in most tasks, due to having an additional layer on top of the base GPT-3 model, but ChatGPT only accepts half as many input tokens as GPT-3. Due to this, we opted to provide

less examples in ChatGPT’s prompts. Can ChatGPT overcome the disadvantage?

**Prompt Optimization.** What impact do the specific examples in a few-shot prompt have on the output of a pretrained language model? Does quality or quantity yield better results?

## 6.1 Experiment Setup

In our experiment, we generated few-shot prompts that contained anywhere between 2 and 5 examples from a corpus of 7 different examples found in [17]. Although we had made and tested other examples in the style of those in [17], we found no significant variability in the quality of the output parses from ChatGPT as a result. A list of all the examples and test inputs are found in A.1 and A.2, respectively.

For our experiment, we simply provided the prompts to ChatGPT, and had the model generate an output, which we then evaluated qualitatively. Refer to C for problems in ChatGPT experiment replicability. Results may vary for replicated versions of this experiment, due to the random nature of open-ended responses in ChatGPT.

This qualitative analysis raises the question of how to judge the correctness or completeness of a parse. In our results and in Appendix A.3, we focus more on the correctness rather than the completeness of a parse, since it would be very difficult for a human to annotate all the possible parsings of a sentence, let alone a computer. Complete parsings would thus require some formalized rules for the generation of parses, to ensure the meaning of utterances is preserved. This remains an open problem in computational linguistics, with [18] serving as a good introduction to the field and its applications in machine learning.

## 6.2 Experiment Results

In our experiments, we found that the quantity and quality of examples in a prompt tend to have a major impact on the quality of the model’s output.

A major example of the impact the quality of prompts has on the quality of output is displayed in Figures 18 and 19, where the 5-shot high-quantity prompt provides much fewer terms and semantic depth than the 2-shot high-quality example. In this example, it is also found that the quality of lexical entries for terms in the output for the 2-shot example is much higher than the quality of lexical terms for the 5-shot example. For instance, the definition of prompting uses more mature terms in the 2-shot case, such as using the mature term “(the information)” instead of the more simple term “(the thing).” Furthermore, note that the lexical entry and variety of terms used in the lexical entry and episodic instantiation of “(the Capstone project)” is more elaborate in the 2-shot prompt than the 5-shot prompt.

This experiment shows that although there is a quality in quantity in evaluation datasets, as laid out in [12], this does not hold for the development of prompts. Instead, higher-quality examples lead to higher-quality prompts. Aggregating our results across the experiment, this

**Examples:** 5-shot prompt in Appendix A.1.

**Input:** The undergraduate student is prompting ChatGPT for his Capstone project in natural language processing.

**Output:** Retrieving lexical entries for (prompting) and (a Capstone project).

(prompting) is when (the prompter) asks (the promptee) for help with (the thing); before (the prompting), (the prompter) needs help with (the thing); (the prompting) is a request for help.

(a Capstone project) is a major project for (the student) that is typically done at the end of (the student)'s academic program;  
(the Capstone project) is a demonstration of (the student)'s skills and knowledge in (the student)'s field of study.

Episodic Instantiation:

(the prompting):

(the prompter) is the undergraduate student

(the promptee) is ChatGPT

(the thing) is the Capstone project in natural language processing

(the Capstone project):

(the student) is the undergraduate student

(the student)'s field of study is natural language processing.

Figure 18: Sample prompt, alongside sample output.

trend tends to hold. We lay out more examples and analysis on them in Appendix A.3. It is also worth noting that more informal and less elaborate semantic parses are also sometimes desired, which leads to the 5-shot prompt being desired in some cases.

## 7 ChatGPT Zero-Shot Software Engineering

As a pretrained language model, ChatGPT has been trained on a large corpus of text data, including programming languages and code examples. During the pretraining process, the model learns patterns and rules in the text data, which enables it to generate code when given appropriate prompts.

When a user inputs a command or a request that requires generating code, ChatGPT uses its understanding of programming languages and code syntax to produce a valid code output. This output is based on a combination of the user's prompt and the model's knowledge of programming principles and best practices.

To generate code, ChatGPT uses natural language processing to interpret the user's request

**Examples:** 2-shot prompt in Appendix A.1.

**Input:** The undergraduate student is prompting ChatGPT for his Capstone project in natural language processing.

**Output:** Retrieving lexical entries for [...]

(prompting) is when (the prompter) prompts (the promptee) for (the information); before (the prompting), (the prompter) wants (the information); (the prompting) causes (the promptee) to provide (the information).

(undergraduate student) [...]

(Capstone project) is (the project) required for (the degree); before (the completion of the Capstone project), (the student) has (the degree requirements) except for (the Capstone project); (the completion of the Capstone project) causes (the student) to have (the degree requirements).

(natural language processing) is the field of computer science and artificial intelligence concerned with the interactions between computers and humans using natural language; (natural language processing) includes tasks such as (language generation), (language understanding), and (language translation).

Episodic Instantiation:

(the prompting):  
(the prompter) is the undergraduate student  
(the promptee) is ChatGPT  
(the information) is for the Capstone project

(the undergraduate student): [...]

(the Capstone project):  
(the project) is the same as (the information in (the prompting))  
(the degree) is the same as (the degree of (the undergraduate student))  
(the degree requirements) include the completion of (the project).

(the natural language processing):  
(the field) is the same as (the topic of the Capstone project)  
(the tasks) include (language generation), (language understanding), and (language translation).

Figure 19: Sample prompt, alongside sample output, with some parts deleted to fit on the page.

and generate code that satisfies the request. Once ChatGPT has analyzed the user’s input, it can use its knowledge of programming languages and code syntax to generate appropriate code. The model can generate code in a variety of programming languages, including Python, Java, and JavaScript, among others.

In this experiment, we studied the software engineering abilities of ChatGPT by analyzing its abilities to develop programming languages. Although it is well-established that ChatGPT can work with “high-level” programs such as basic Python applications and package calls, as shown in [19], there is little study into its ability to generate “low-level” programs such as those used in the development of operating systems and compilers design, or in the quantitative development and high-frequency computing space. The most notable difference between the “high-level” and “low-level” coding space is that “high-level” programs tend to have very high degrees of abstraction and rely on programming languages such as Python, while “low-level” programs tend to rely much more heavily on memory and runtime optimization, which requires the abstractions to come from specialized user-generated libraries, rather than those latent within a programming language or package.

Furthermore, as shown in [10], ChatGPT does have the ability to successfully debug software generated by a user. As we show in this experiment, it also has the ability to debug its own software, without the need to input the entire buggy code into the prompt, saving the user on tokens.

## 7.1 Experiment Setup

In this experiment, we started a single session of ChatGPT on the OpenAI website, then input the prompts in order as laid out in Appendix B. There are times where ChatGPT refuses to provide code, instead giving a theoretical answer, such as in Figure 20. In these cases, we click the button for “Regenerate response” until the ChatGPT model decides to produce some code. Usually, after one click of the button, ChatGPT generates code.

Furthermore, to ensure that the entirety of the code is printed, after ChatGPT uses up its output tokens, we respond with “please continue” in cases where the code is notably incomplete. Although this could yield false negatives where the code is truly incomplete but we did not notice, this has not been a noticeable error in the experiment, and in an applied setting, this is the most likely approach by an end user.

## 7.2 Experiment Results

In this section, we analyze the code generated by ChatGPT in this experiment. The code can be following on the following Github repository: [github.com/AlexRashduni/chatgpt-programming-language](https://github.com/AlexRashduni/chatgpt-programming-language).

The experiment with ChatGPT begins with my request that it sets up an interpreter for a functional programming language using C. The end goal in mind was to develop a programming language with syntactic similar to C or Python. Immediately upon this very basic

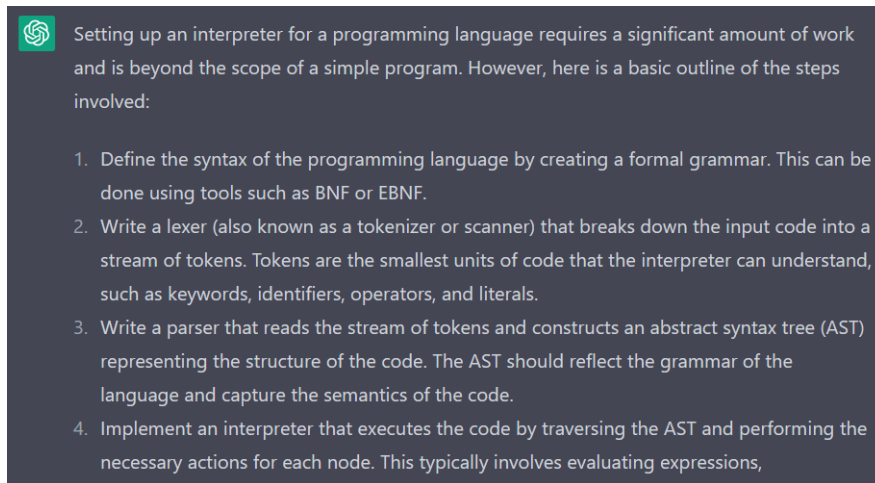


Figure 20: ChatGPT providing a theoretical answer instead of code.

request, ChatGPT was able to develop the tokenizer and parser for a programming language with the following functionality.

- Integer and floating-point numbers
- Basic arithmetic operations
- Variables with dynamic typing
- Function definitions and function calls with parameters

Although none of these functionalities were explicitly referenced, they were immediately developed by the ChatGPT model, and with very few bugs. All of the bugs in the code were instances where ChatGPT called on a function, variable, or structure that it never initialized. Debugging was as simple as asking ChatGPT to add the functionality to the program with the interface it initially assumed was available, which it was able to complete immediately. Despite that, sometimes, these new interfaces themselves yielded more of these errors.

Among the new functions added during the debugging process are an in-house hashtable library, symbol tables, and program environments. These additions showcase ChatGPT's ability to develop novel code and emulate common ideas among software engineers en masse, but it also displays the current hindrances of ChatGPT in truly initializing the code that it is referencing. In this way, although ChatGPT can predict what a code should look like syntactically and to some extent semantically, it does not yet have the procedural knowledge to develop a fully-functional programming language.

As of now, the code available on the Github repository still has a few bugs here and there, to showcase the shortcomings of the ChatGPT model in the development of low-level software. To compile and test the generated code, we used GCC and the C11 standard.



## 8 Open Problems

The field of prompting has made significant progress in recent years with the rise of large-scale pretrained language models such as ChatGPT and the highly anticipated GPT-4 model. While these models have shown remarkable success in various natural language tasks, they still face several open problems that require further investigation.

One of the primary challenges is in the realm of procedural knowledge, particularly in tasks that require arithmetic. Pretrained language models like T0 have been shown to have limited ability in solving arithmetic problems, despite being trained on large amounts of data. This suggests that there is a need to develop new techniques and approaches to improve the ability of language models to handle procedural knowledge.

Another area of difficulty for pretrained language models is in the development of bug-free software at a large scale. Language models like ChatGPT have demonstrated the ability to generate code that is syntactically correct, but they often produce code that contains logical errors. This presents a significant challenge for using language models in software engineering, where the production of bug-free code is critical. Developing methods to improve the ability of language models to generate correct code would be a valuable contribution to the field of prompting.

In addition to these challenges, there is still uncertainty regarding what distinguishes a high-quality prompt from a low-quality one. While much research has been done on the use of prompts to guide language models, there is no clear consensus on what factors contribute to the effectiveness of prompts. This presents an open problem that could benefit from further investigation and experimentation.

Despite these challenges, the field of prompting is poised for significant advancements with the development of the groundbreaking GPT-4 model. While details on the GPT-4 model are somewhat scarce, it is known to have greatly improved capabilities and performance compared to previous models. If the model lives up to expectations, GPT-4 could potentially address many of the open problems in the field of prompting and pave the way for new applications and innovations.

In conclusion, while the field of prompting has seen significant progress in recent years, there are still important challenges that need to be addressed. Developing new techniques to improve the ability of language models to handle procedural knowledge and generate correct code, as well as understanding the qualities of effective prompts, are important areas of future research. The arrival of the GPT-4 model presents an exciting opportunity to push the boundaries of the field and make significant advancements in natural language processing.

## References

- [1] Sanh, V., et al, 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In 2022, *ICLR* 10th International Conference on Learning Representations. 2110.08207 (arxiv.org).
- [2] Vaswani, A., et al, 2017. Attention Is All You Need. In 2017, *NIPS* 31st Conference on Neural Information Processing Systems. 1706.03762 (arxiv.org).
- [3] Jin, W., et al, 2022. A Good Prompt is Worth Millions of Parameters: Low-resource Prompt-based Learning for Vision-Language Models. *ACL* 60th Annual Meeting of the Association for Computational Linguistics. 2110.08484 (arxiv.org).
- [4] Lang, H., et al, 2022. Co-training Improves Prompt-based Learning for Large Language Models. 2202.00828 (arxiv.org).
- [5] Shin, R., et al, 2021. Constrained Language Models Yield Few-Shot Semantic Parsers. In 2021, *ACL* Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2104.08768 (arxiv.org).
- [6] Jiang, A., et al, 2023. Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs. In 2023, *ICLR* 11th International Conference on Learning Representations. OpenReview link..
- [7] Brown, T., et al, 2020. Language models are few-shot learners. In 2020, *NIPS* Proceedings of the 34th International Conference on Neural Information Processing. 2005.14165 (arxiv.org).
- [8] Model T0 Training Data: CommonsenseQA, DREAM, QUAIL, QuaRTz, Social IQA, WiQA, Cosmos, QASC, Quarel, SciQ, Wiki Hop, ARC, OpenBookQA, MultiRC, PIQA, RACE, HellaSwag, BoolQ, AG News, DBpedia, TREC, MRPC, PAWS, QQP, Amazon, App Reviews, IMDB, Rotten Tomatoes, Yelp, Common Gen, Wiki Bio, Hotpot QA, Wiki QA, Trivia QA, Web Questions, Adversarial QA, Quoref, DuoRC, ROPES, SQuAD v2, ReCoRD. Hugging Face Big Science Data Set. bigscience/P3 · Datasets at Hugging Face.
- [9] Model T0 Testing Data: ANLI, CB, RTE, WSC, Winogrande, WiC, COPA, HellaSwag, Story Cloze. Model T0 Training Data: CNN Daily Mail, Gigaword, MultiNews, SamSum, XSum. Hugging Face Big Science Data Set. bigscience/P3 · Datasets at Hugging Face.
- [10] 2022. OpenAI - ChatGPT Blog.
- [11] Chen, M., et al, 2021. Evaluating Large Language Models Trained on Code. 2107.03374 (arxiv.org).
- [12] Srivastava, A., et al, 2022. Beyond The Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. 2022 arXiv, Cornell University. Paper

Link: [Beyond the Imitation Game Benchmark \(arxiv.org\)](#) Dataset Link: [GitHub - google/BIG-bench: Beyond the Imitation Game collaborative benchmark for measuring and extrapolating the capabilities of language models.](#)

- [13] Parallelfomers: Journey to deploying big models. TUNiB Blog.
- [14] Sanh, V., et al, 2022. Github repository: [bigscience-workshop/t-zero \(Github\)](#).
- [15] Sanh, V., et al, 2022. Github repository: [bigscience-workshop/promptsources \(Github\)](#).
- [16] Hendrycks, D., et al, 2021. Measuring Massive Multitask Language Understanding. In 2021, *ICLR* 9th International Conference on Learning Representations. 2009.03300 ([arxiv.org](#)).
- [17] McAllester, 2022. GPT-3 Semantic Parsing with Simple Sentence Graphs. MachineThoughts.
- [18] Liang, P., and Potts, C, 2014. Bringing machine learning and compositional semantics together. In 2014, submitted to the Annual Review of Linguistics.
- [19] Tsui, T., 2023. Coding with ChatGPT - Using OpenAI's language model to write Python code for geo-data science. Medium.

# A Prompts for Semantic Parsers

For our prompts in semantic parsers, we chose to use 2-shot and 5-shot prompting, over the 7 examples provided by [17]. Due to GPT-3 being able to have a 2048-token input, while T0 and ChatGPT are limited to 1024, we chose to select a  $k$  such that we could use as many of the examples in [17] as possible. Given that both the smallest 5 prompts in the provided examples and the largest 2 prompts in the provided examples are less than 1024 tokens, we decided to partition the examples in that manner. Although we did experiment a little with other prompts of varying sizes, we found little variability in the performance of ChatGPT and T0. In Appendix A.1, we showcase the 7 examples from the paper, and the partition used in the experiment. In Appendix A.2, we showcase the 18 test inputs. Lastly, in Appendix A.3, we provide an analysis for the outputs of the ChatGPT model.

## A.1 Few-shot Examples for Semantic Parsers

### 2-shot Prompt

INPUT: The principal said that the teacher had resigned when they were actually on vacation.

Retrieving lexical entries for (saying), (resigning), and (vacationing).

(saying) is when (the sayer) says (the statement) to (the hearer);  
before (the saying), (the sayer) wants (the hearer) to hear (the statement);  
(the saying) causes (the hearer) to hear (the statement);  
after (the saying) (the hearer) has heard (the statement).

(resigning) is when (the resigner) resigns from (the job);  
before (the resignation), (the resigner) has (the job);  
before (the resignation), (the resigner) does not want (the job);  
(the resignation) causes (the resigner) to not have (the job).

(vacationing) is when (the vacationer) is on vacation from (the job) and in (the location);  
before (the vacation), (the vacationer) wants (the vacation);  
before (the vacation), (the vacationer) is doing (the job);  
during (the vacation), (the vacationer) is not doing (the job);  
during (the vacation), (the vacationer) is at (the location);  
after (the vacation), (the vacationer) is again doing (the job).

Episodic Instantiation:

(the saying):  
(the sayer) is the principal  
(the statement) is (the teacher had resigned)  
(the statement) is false  
(the hearer) is unknown

(the resigning):  
(the resigner) is the teacher  
(the job) is teaching

(the vacationing):  
(the vacationer) is the teacher  
(the job) is same as (the job of (the resignation))  
(the location) is unknown

INPUT: Herschel Walker pledges to sue over the report that he paid for an abortion.

Retrieving lexical entries for (pledging), (suing), (reporting), (paying) and (an abortion).

(pledging) is when (the pledger) pledges to do (the action);  
(the pledge) is a public promise by (the pledger) to do (the action).

(suing) is when (the plaintiff) sues (the defendant) over (the issue) and is decided by (the judge);  
(the plaintiff) wins (the lawsuit) if (the judge) rules in favor of (the plaintiff);  
(the defendant) wins (the lawsuit) if (the judge) rules in favor of (the defendant).

(reporting) is when (the reporter) reports (the information) to (the recipient);  
before (the report), (the recipient) expects (the report);  
before (the report), (the reporter) should make (the report);  
(the report) causes the (the recipient) to know (the information).

(paying) is when (the payer) pays (the amount) to (the payee) for (the thing);  
before (the payment), the payer has (the amount);  
before (the payment), (the recipient) wants (the payment);  
(the payment) causes (the recipient) to have (the amount).

(an abortion) is when (the woman) has (the abortion);  
before (the abortion), (the woman) is pregnant;  
before (the abortion), (the woman) wants to not be pregnant;  
(the abortion) causes (the woman) to not be pregnant.

Episodic Instantiation:

(the pledge):  
(the pledger) is Herschel Walker  
(the action) is to sue

(the suit):  
(the plaintiff) is Herschel Walker  
(the defendant) is unknown  
(the issue) is (the report)  
(the judge) is unknown

(the report):  
(the reporter) the same as (the defendant of (the report))

(the information) is that Herschel Walker paid for an abortion

(the payment):

(the payer) is Herschel Walker

(the payee) is unknown

(the amount) is unknown

(the thing) is (the abortion)

(the abortion):

(the pregnant woman) is unknown

(the procedure) is unknown

(the pregnancy) is unknown

### 5-shot Prompt

INPUT: Fred ate scrambles eggs.

Retrieving lexical entry for (eating) and (an egg)

(eating) is when (the eater) eats (the food);

before (the eating), (the eater) is hungry;

after (the eating), (the eater) is not hungry;

after (the eating), (the food) is inside (the eater);

after (the eating), (the food) is gone.

(an egg) is a means of reproduction for (the animal);

(the egg) is laid by (the animal);

(the egg) hatches to produce a new (the animal).

Episodic Instantiation:

(the eating):

(the eater) is Fred

(the food) is scrambled eggs

(the egg):

(the animal) is a chicken

INPUT: George Floyd was murdered by the police.

Retrieving lexical entry for (murdering) and (police).

(murdering) is when (the victim) is murdered by (the murderer);

before (the murder), (the victim) is alive;

(the murder) causes (the victim) to be dead;

before (the murder), (the victim) does not want (the murder) to happen;

(the murder) is a crime.

(police) are people with the authority to enforce the law.

Episodic Instantiation:

(the murder):  
(the victim) is George Floyd  
(the murderer) is the police  
(the police) are unknown

INPUT: Putin told the general to attack.

Retrieving lexical entries for (telling to do) and (a general).

(telling to do) is when (the teller) tells (the hearer) to do (the action);  
(the teller) has authority over (the hearer);  
after (the telling to do), (the hearer) should do (the action).

(a general) is a person who controls an army.

Episodic Instantiation:

(the telling to do):  
(the teller) is Putin  
(the hearer) is the general  
(the action) is to attack  
(the general) is unknown.

INPUT: Putin told the general to attack.

Retrieving lexical entry for (attacking).

(attacking) is when (the attacker) attacks (the attacked);  
before (the attack), (the attacked) does not want (the attack) to happen;  
during (the attack), (the attacker) tries to damage (the attacked);  
after (the attack), (the attacked) is angry at (the attacker).

Episodic Instantiation:

(the attack):  
(the attacker) is the general  
(the one attacked) is unknown

INPUT: Mary told John about the party.

Retrieving lexical entries for (telling about) and (a party).

(a telling about) is when (the teller) tells (the hearer) about (the thing);  
before (the telling about), (the teller) wants (the hearer) to know about (the thing);  
(the telling about) causes (the hearer) to know about (the thing).

(a party) is when (the partiers) get together at (the place) and (the time) to have fun;  
during (the party), (the partiers) have fun.

Episodic Instantiation:

(the telling about):  
(the teller) is Mary  
(the hearer) is John  
(the thing) is the party

(the party):  
(the parters) are unknown  
(the place) is unknown  
(the time) is unknown

## A.2 Test Inputs for Semantic Parsers

When prompting a model, these inputs are preceded by the token “INPUT: ”, then appended to the prompts above.

The following inputs were used exclusively on the 5-shot evaluation.

1. The principal said that the teacher had resigned when they were actually on vacation.
2. Herschel Walker pledges to sue over the report that he paid for an abortion.

The following inputs were used exclusively on the 2-shot evaluation.

3. Mary told John about the party.
4. Putin told the general to attack.
5. George Floyd was murdered by the police.
6. Fred ate scrambles eggs.

The following inputs were used on both the 5-shot and 2-shot evaluations.

7. The undergraduate student is prompting ChatGPT for his Capstone project in natural language processing.
8. The paper is sitting on a desk, with paper in it.
9. The printer is sitting on a desk, with paper in it.
10. The dog is barking because he stole a container of sour cream.
11. The dog is stealing sour cream, and after I noticed, he barked at me.
12. The dog has a bone.
13. The dog is eating a bone.



14. When pigs fly, I will stop prompting ChatGPT.
15. The politician said something controversial. Two days later, he dropped out of the race.
16. erngjdtenjdnfbkjnsfdkjbnskdjnkjsdnblksdnblksmdblkms<sup>2</sup>
17. 10 random words - Landowner, Cap, Substitute, Relate, Week, Go, Shortage, Tap, Edition, Inn<sup>3</sup>
18. Prof. Stratos asked me to prompt ChatGPT and T0, and I’ve just about run out of ideas. Can you come up with a prompt?<sup>4</sup>

### A.3 Analysis of ChatGPT Parsing Results

In this section, we analyze notable results in the semantic parsing of inputs referenced in Appendix A.2. While doing so, we refer to each test input by its associated number in the list, for simplicity.

#### Test Example 1 on the 5-shot prompt

In this example, we analyze the semantic parse of the 5-shot prompt on the first example in the 2-shot prompt, to see if the high-quantity prompt can replicate the quality of the high-quality prompt. Refer to Appendix A.1 for the original example, and to Figure 21 for the model output under the 5-shot prompt.

As shown in the results, the lexical entries of the 5-shot prompt were up to par in the sense that ChatGPT produced sensible output and provided entries for the same terms as the original example. Despite that, ChatGPT missed some key points, some of which completely change the meaning of the statement. Most notably, it missed that the statement made by the principal was false. This shows that although the 5-shot prompt does maintain the verbosity and lexical analysis of the 2-shot prompt (at least in this case), it misses on key semantic analysis.

#### Test Example 4 on the 2-shot prompt

In this example, we compare the semantic parse of the 2-shot prompt to the “Putin told the general to attack” input that appeared twice in the 5-shot prompt. Refer to Figure 22 for the model output, and to Appendix A.1 for the original examples.

---

<sup>2</sup>Pure gibberish, intended to check queried models’ capacity to distinguish between “garbage” and valid inputs.

<sup>3</sup>Intended to determine whether models will force a relation between random words in a list.

<sup>4</sup>Intended to trick the model and see whether it will parse the input or provide an idea for a prompt.

**Examples:** 5-shot prompt in Appendix A.1.

**Input:** The principal said that the teacher had resigned when they were actually on vacation.

**Output:** Retrieving lexical entries for (saying), (resigning), and (vacationing).

(saying) is when (the speaker) says (the thing);

(the thing) is expressed in (the words);

(the words) are heard by (the hearer).

(resigning) is when (the resigner) resigns from (the position);

before (the resignation), (the resigner) holds (the position);

after (the resignation), (the resigner) no longer holds (the position);

(the resignation) is voluntary.

(vacationing) is when (the vacator) takes (the time off) from work;

during (the vacation), (the vacator) has free time;

(the vacator) returns to work after (the vacation).

Episodic Instantiation:

(the saying):

(the speaker) is the principal

(the thing) is that the teacher had resigned

(the words) are unknown

(the hearer) is unknown

(the resignation):

(the resigner) is the teacher

(the position) is unknown

(the vacationing):

(the vacator) is the teacher

(the time off) is unknown.

Figure 21: Output for test example #1 on ChatGPT, given the 5-shot prompt.

**Examples:** 2-shot prompt in Appendix A.1.

**Input:** Putin told the general to attack.

**Output:** Retrieving lexical entries for (telling), (attacking).

(telling) is when (the teller) tells (the addressee) to do (the action);  
before (the telling), (the teller) wants (the addressee) to do (the action);  
(the telling) causes (the addressee) to do (the action);  
after (the telling), (the addressee) does (the action).

(attacking) is when (the attacker) attacks (the target);  
before (the attack), (the attacker) wants (the attack);  
(the attack) causes (the target) to be attacked;  
after (the attack), (the target) is attacked.

Episodic Instantiation:

(the telling):  
(the teller) is Putin  
(the addressee) is the general  
(the action) is to attack

(the attack):  
(the attacker) is the general  
(the target) is unknown

Figure 22: Output for test example #4 on ChatGPT, given the 2-shot prompt.

The results for this example show that although the 2-shot prompt was capable of finding verbose entries in the episodic instantiations of the sentence, it lacks one of the three terms in the lexical entries shared among the two examples that use this input. Namely, the 2-shot prompt did not recognize “the general” as a notable entry in the sentence. To be fair, the 2-shot prompt’s lexical entries contained many verbs, but only one noun, for much more elaborate sentences. Therefore, this could be avoided by introducing more prompts with nouns.

## Importance of wording

Another noteworthy example is the impact of the wording of an input on its semantic parse. To explore this, we examine the difference in output between the inputs “The dog has a bone” and “The dog is eating a bone,” as shown in Figures 23 and 24, respectively.

In the former case, the model believed that the bone in question was a dog bone, while in the latter case, the model understood that the bone was not a dog bone. Although in the former

**Examples:** 2-shot prompt in Appendix A.1.

**Input:** The dog has a bone.

**Output:** Retrieving lexical entries for (having), (dog), and (bone).

(having) is when (the holder) has (the thing);  
before (the having), (the holder) wants (the thing);  
(the having) causes (the holder) to have (the thing).

(dog) is a domesticated mammal that belongs to the species *Canis lupus familiaris*.

(bone) is a rigid organ that makes up the skeleton of (the animal).

Episodic Instantiation:

(the having):  
(the holder) is the dog  
(the thing) is a bone

(the dog):  
(the species) is *Canis lupus familiaris*

(the bone):  
(the animal) is the dog

Figure 23: Output for test example #12 on ChatGPT, given the 2-shot prompt.

phrase, “the dog has a bone,” the bone could in theory be a dog bone, and it is technically correct, it is arguably less common than a beef or lamb bone. Furthermore, when the word “eat” comes into play, the model is able to create much more elaborate lexical entries, even though the episodic instantiation is filled with more unknowns than before.

These examples showcase that although pretrained language models have come a long way in recent years, there are still many unknowns when it comes to understanding semantics, especially in semantic parsing and prompt tuning.

## Attempts to trick ChatGPT

For our last few examples, we will analyze three attempts to trick ChatGPT in few-shot semantic parsing, as well as their outcomes. Although the figures in this last section state that the models were tested with the 5-shot prompt, the exact same results hold for the 2-shot prompt.

First, we attempted to trick ChatGPT by providing it with a gibberish input. It immediately noticed the gibberish and responded appropriately, as shown in Figure 25.

**Examples:** 2-shot prompt in Appendix A.1.

**Input:** The dog is eating a bone.

**Output:** Retrieving lexical entries for (eating), (dog), and (bone).

(eating) is when (the eater) eats (the food);  
before (the eating), (the eater) is hungry;  
before (the eating), (the food) is in front of (the eater);  
(the eating) causes (the eater) to consume (the food).

(dog) is a domestic animal known for its loyalty, obedience and affection towards its owner;  
(the dog) can perform various tasks such as hunting, herding, and guarding;  
(the dog) can eat various foods such as meat, bones, and vegetables.

(bone) is a hard, dense material that makes up the skeletons of vertebrates;  
(the bone) is often used as a chew toy or treat for dogs;  
(the bone) can also be used in cooking to make broth or stock.

Episodic Instantiation:

(the eating):  
(the eater) is the dog  
(the food) is a bone

(the dog):  
(the species) is *Canis familiaris*  
(the breed) is unknown  
(the size) is unknown  
(the age) is unknown  
(the task) is eating

(the bone):  
(the source) is unknown  
(the type) is unknown  
(the size) is unknown.

Figure 24: Output for test example #13 on ChatGPT, given the 2-shot prompt.

**Examples:** 5-shot prompt in Appendix A.1.

**Input:** erngjdtenjdnfbkjnsfdkjbnskdjnkjsdnblksdnblksmdblkms

**Output:** I'm sorry, but I do not understand what you are trying to ask. The input you provided is not a coherent statement or question. Please provide a clear and valid input.

Figure 25: Output for test example #16 on ChatGPT, given the 5-shot prompt. Results were trivially different from the 2-shot prompt.

From there, we checked whether ChatGPT could provide a semantic parse for ten random words with no association, as shown in Figure 26. In response, ChatGPT provided lexical entries for each of the ten words, and no episodic instantiations, which was the correct response.

Lastly, we provided a sentence for input that should have been semantically parsed, but tricked ChatGPT, as shown in Figure 27. In this example, in our input, we put in a sentence that could be interpreted as a question asking ChatGPT to come up with its own prompt. Rather than semantically parse the input, ChatGPT responded with some suggestions. Although this could be due to a feature in the model, it is nonetheless an undesirable outcome.

These last few examples show that although ChatGPT is capable of semantically parsing many phrases, there is still room for improvement, especially in tuning prompts for word choice.

**Examples:** 5-shot prompt in Appendix A.1.

**Input:** 10 random words - Landowner, Cap, Substitute, Relate, Week, Go, Shortage, Tap, Edition, Inn

**Output:** Retrieving lexical entries for: Landowner, Cap, Substitute, Relate, Week, Go, Shortage, Tap, Edition, Inn.

(Landowner) is a person who owns (the land).

(Cap) is a covering for (the head) or (the top of something); it can also refer to a limit on (something).

(Substitute) is a person or thing that takes the place of (something or someone else).

(Relate) is to have a connection or connection between (two or more things).

(Week) is a unit of time that is 7 days long.

(Go) is a verb meaning to move from one place to another.

(Shortage) is a lack of (something) that is needed.

(Tap) is a device used to control the flow of a liquid or gas.

(Edition) is a version of (something) that has been published or printed.

(Inn) is a place to stay, typically providing lodging and food for travelers.

Figure 26: Output for test example #17 on ChatGPT, given the 5-shot prompt. Results were trivially different from the 2-shot prompt.

**Examples:** 5-shot prompt in Appendix A.1.

**Input:** Prof. Stratos asked me to prompt ChatGPT and T0, and I’ve just about run out of ideas. Can you come up with a prompt?

**Output:** Sure, how about this: “Imagine a world where all human emotions and thoughts can be directly translated into a tangible form. Discuss the potential consequences of this technology on society and individuals.”

Figure 27: Output for test example #18 on ChatGPT, given the 5-shot prompt. Results were trivially different from the 2-shot prompt.

## B Prompts for Programming Language Development

In this section, we lay out the prompts used in the development of the programming language set out in Section 7. We also describe the potential problems faced in the replicability of open-ended prompts in the ChatGPT model.

### B.1 List of Prompts used in Development

1. Provide a C program that sets up an interpreter for a functional programming language.
2. Is your answer complete? If not, please continue from  
"(\*current\_token)++; // skip function name token  
if (tokens[\*current\_token].type != TOKEN\_LPAREN) {  
fprintf(stderr, "Expected '(' after function name"  
in code format
3. are there any bugs you know of in the code you sent?
4. The following structure has no values for block or program. Could you please add them?

```
typedef struct node_s {  
    node_type_t type;  
    union {  
        int int_value;  
        float float_value;  
        struct {  
            struct node_s* left;  
            struct node_s* right;  
            token_type_t op;  
        } binary_op;  
        struct {  
            char* name;  
            struct node_s* value;  
        } assignment;  
        char* variable_name;  
        struct {  
            char* name;  
            struct node_s** arguments;  
            int num_arguments;  
        } function_call;  
        struct {  
            char* name;  
            char** parameters;  
            int num_parameters;  
            struct node_s* body;
```



```

    } function_definition;
    struct node_s* return_value;
    } data;
} node_t;

```

5. Can you add the interpret function to this program?

6. In main, we call the following tokenize as follows:

```

int main() {
    char* input = "def add(a, b) { return a + b; }";
    token_t* tokens = tokenize(input);
    node_t* program = parse_program(tokens);
    interpret(program);
    return 0;
}

```

We don't have a tokenize function, but we have a lexer function as follows:

```

void lexer(const char* input, token_t* tokens, int* num_tokens);

```

The function writes the tokenization into token\_t\* tokens. Could you please change the function call from tokenize to lexer?

7. Could you please provide the implementations of execute\_binary\_op(node\_t\* node), execute\_function\_call(node\_t\* node), and execute\_assignment(node\_t\* node)

8. Could you please provide the implementation for the environment structure and the create\_environment() function?

9. Can you create the destroy\_environment function?

10. Given create\_environment and destroy\_environment, can you recreate the implementations for execute\_function\_call and execute\_assignment

11. Can you implement the bind\_variable function using the current implementation of environment and node\_t?

12. Can you implement hashtable\_lookup and hashtable\_insert with the function calls below:

```

hashtable_lookup(environment_t*, char*);
hashtable_insert(char**, char*, node_t**);

```

node\_t and environment\_t are defined as discussed before.

## C Replicability of ChatGPT Prompting

Despite the prompts we provided in Appendices A and B, there is an inherent degree of randomness in the production of open-ended responses in ChatGPT. This randomness causes ChatGPT to provide a range of different responses for the same prompt.

As shown in Figure 28, OpenAI and ChatGPT are also aware of this issue. It appears that this is a feature rather than a bug, since otherwise, users would not be able to use a “Regenerate response” option. Furthermore, without the ability to generate multiple different responses, the user feedback options available with responses would not indicate why or how a response could be better than other ones, instead stating that the first response is “good” or “bad.”

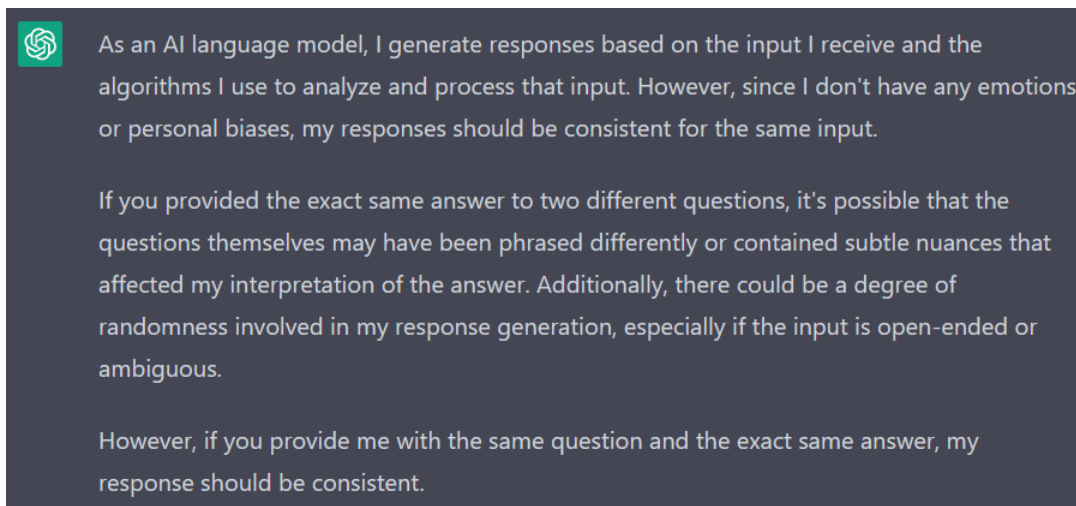


Figure 28: ChatGPT on the replicability of experiments. The model appears to be aware of the issues in replication.

Despite the fact that this replicability problem is a feature rather than a bug, this problem could be resolved by setting up functionality to allow the user to start a ChatGPT session with a predetermined random seed, rather than the indeterminate random seed currently present in the model.

It appears that this problem is unlikely to be resolved in the near future, since OpenAI has begun taking down models and APIs such as codex-davinci with the advent of GPT-4, and replacing it with a revised version built around the new pretrained language model. This has fuelled a crisis in the replicability of papers that have results on the Codex models, since it is now impossible to replicate them.