How Proficient Are Large Language Models in Formal Languages? An In-Depth Insight for Knowledge Base Question Answering

Anonymous ACL submission

Abstract

Knowledge Base Question Answering (KBQA) aims to answer natural language questions based on facts in knowledge bases. A typical approach to KBQA is semantic parsing, which translates a question into an executable logical form in a formal language. Recent works leverage the capabilities of large language models (LLMs) for logical form generation to improve performance. However, although it is validated that LLMs are capable of solving some KBQA problems, there has been little discussion on the differences in LLMs' proficiency in formal languages used in semantic parsing. In this work, we propose to evaluate the understanding and generation ability of LLMs to deal with differently structured logical forms by examining the inter-conversion of natural and formal language through in-context learning of LLMs. Extensive experiments with models of different sizes show that state-of-the-art LLMs can understand formal languages as well as humans, but generating correct logical forms given a few examples remains a challenge. Most importantly, our results also indicate that LLMs exhibit considerable sensitivity. In general, the formal language with a lower formalization level, i.e., the more similar it is to natural language, is more friendly to LLMs.

1 Introduction

Knowledge Base Question Answering (KBQA) is a challenging natural language processing (NLP) task to answer natural language questions based on fact triples stored in the knowledge base (KB), such as Wikidata (Vrandecic and Krötzsch, 2014) and Freebase (Bollacker et al., 2008). In recent years, a typical paradigm of KBQA methods is semantic parsing (Berant et al., 2013; Cao et al., 2019a; Ye et al., 2022; Shu et al., 2022), where natural language questions (NLQs) are translated into their corresponding structured logical forms (LFs), such as KoPL (Cao et al., 2022a), SPARQL (Pérez et al., 2006) or Lambda DCS (Liang, 2013). The logical forms are capable of expressing multiple reasoning operations such as multi-hop inference and quantitative comparison, and can be executed on KBs to get accurate answers.

The recent advancements of large language models (LLMs) (OpenAI, 2023) have led to significant attention on utilizing LLMs for KBQA. Previous works have validated the ability of LLMs to memorize, understand and apply knowledge for reasoning (Yu et al., 2023b), and serve as agents to solve KBQA problem (Liu et al., 2023). Additionally, others introduced many techniques to improve the performance, such as in-context learning (Li et al., 2023), chain-of-thought (Liang et al., 2023), and instruction-tuning (Luo et al., 2023).

However, there has been little discussion on the differences in the proficiency of LLMs in different formal languages that used as paring targets in semantic parsing. It is proved that LLMs can do well on programming language such as Python (Gao et al., 2023) with sufficient data in the pretraining, but it remains intriguing that how well do LLMs master other formal languages without extra data and further fine-tuning. By examining the proficiency of LLMs in formal languages, we can gain a better understanding of LLMs' upper limits. It also would be advantageous to choose appropriate models and formal languages for specific scenarios if different LLMs have varying levels of proficiency in different formal languages.

In this paper, we propose to evaluate the inherent understanding and generation ability of the formal language in the original LLMs without additional fine-tuning. We define two evaluation tasks based on sub-tasks of KBQA: 1) **Formal Language Understanding**, which aims to translate a LF into its corresponding NLQ. The translation process can be considered as the model interpreting the provided LFs in natural language, demonstrating LLMs' understanding ability of formal language; 2) **Formal**

Language Generation, which aims to correctly convert a NLQ into its corresponding LF, requiring the model to not only understand but also generate LFs, demonstrating its capability in generation.

With respect to the formal languages for evaluation, according to the varying levels of formalization (may be broadly understood as the dissimilarity to natural language, i.e. the higher the level of formalization, the less similar it is to natural language) and different logical structures (e.g. tree, graph or chain), we choose Lambda DCS (Liang, 2013), SPARQL (Pérez et al., 2006), and KoPL (Cao et al., 2022a) as representative formal languages, which are commonly used for knowledge based question answering research (Nie et al., 2022; Ye et al., 2022; Shin et al., 2021).

For the generation methods, to reflect the inherent proficiency of LLMs, we combine the in-context learning ability (Brown et al., 2020) of LLMs and chain-of-thought generation (Wei et al., 2022) for both evaluation tasks, where the desired outputs are generated conditioned on the input along with a few demonstration pairs of NLQs and LFs carefully selected from a seed dataset. For demonstration selection, to ensure that the logical structure of the examples should be as similar as possible to the target, we carefully design a greedy search algorithm based on the minimum edit distance to solve a maximum coverage problem.

For the quality evaluation of the generated NLQ, to avoid the inaccuracy of automatic metrics and the labor-intensive human evaluation, we propose a contrastive evaluation approach. This involves separately training a semantic parser using LLM-generated data and comparing them with parser trained using manually labeled data. By comparing the performance of the parsers, we can assess the differences in quality between the data generated by the LLMs and the manually labeled data.

Our findings indicate that LLMs have approached the human annotators in the task of formal language understanding that generate natural language questions from logical forms. However, conversely, challenges still exist in the task of formal language generating if only a few examples are given. Importantly, we observe that models exhibit the sensitivity to different logical forms. Overall, the lower the level of formalization (the similar it is to natural language), the easier it is for models to understand and generate. In conclusion, this study examines the proficiency of LLMs in formal language understanding and generation, and helps to provide valuable insights for LLMs-based reasoning approaches.

2 Related Work

Knowledge Base Question Answering. Typical methods for solving KBQA problems can be broadly divided into two categories. One category is the retrieval-based method. These methods usually directly output the answer by retrieving triples and subgraphs that related to the question from KB or embedded memory (Sun et al., 2019; Shi et al., 2021; Zhang et al., 2022; Oguz et al., 2022; Dong et al., 2023). Another is the semantic-parsing-based method, which translates questions into logical forms executable against KBs. The logical forms are usually generated by step-by-step graph searching and generation (Gu et al., 2021; Jiang et al., 2023b,a; Gu et al., 2023) or by sequence-to-sequence model that trained with parallel data (Ye et al., 2022; Cao et al., 2022b; Yu et al., 2023a; Shu et al., 2022; Luo et al., 2023).

Since the logical form can facilitate communication between the model and the KB, the latter category usually out-performance the former and also enjoy a better interpretability. Therefore, our work sets out to explore the role of different formal languages in KBQA in the era of LLMs.

Evaluation of LLMs. From the advent of pretrained language models (PLMs) such as BERT (Devlin et al., 2019) and GPT (Radford et al., 2018) to the emergence of increasingly larger and powerful LLMs (Brown et al., 2020; Chowdhery et al., 2023; Scao et al., 2022; Zeng et al., 2023; Touvron et al., 2023) in recent years, language models have changed the paradigms of many traditional task. At the same time, the evaluations of language models are also ongoing.

Early works explored PLMs' capability boundaries including linguistics knowledge (Hewitt and Manning, 2019; Clark et al., 2019; Liu et al., 2019) as well as world knowledge like entities (Broscheit, 2020), relations (Petroni et al., 2019; Jiang et al., 2020; Zhong et al., 2021), and concepts (Peng et al., 2022; Dalvi et al., 2022). Recent works have included comprehensive tasks and datasets to create new benchmarks for LLMs (Bang et al., 2023; Srivastava et al., 2022; Yu et al., 2023b; Liu et al., 2023). Our work are inspired by above studies, extending the evaluation of LLMs from the natural language domain to various formal languages.

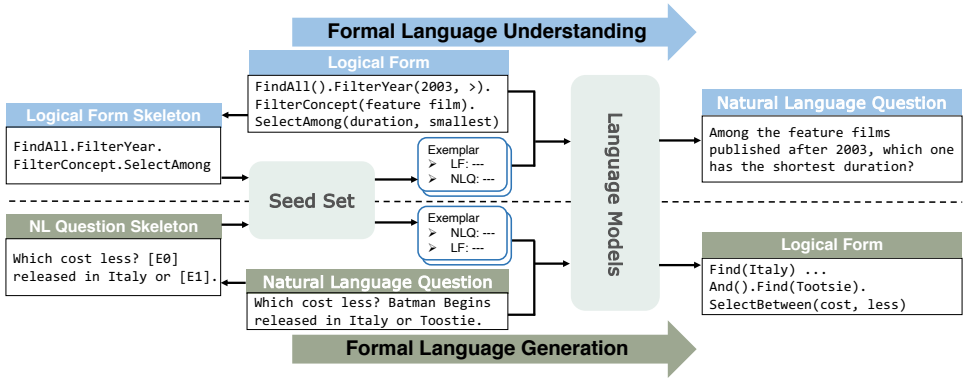

Figure 1: A simple illustration for the probing task of both formal language understanding and generation.

3 Evaluation Task Definition

As shown in Figure 1, We define two probing tasks, namely the **formal language understanding** and **formal language generation**. In this section, we introduce the formalized definitions of these two tasks and how to assess LLMs' performance.

3.1 Formal Language Understanding

The goal of the task is for a LLM M to translate a LF input to its corresponding NLQ. Formally, we have a target set $\mathcal{T} = \{l*\}$ of LFs, and a seed set $\mathcal{S} = \{(l, q)\}$ of LF - NLQ pairs. To assemble the demonstration, for every l in $\mathcal{T}$ we need to retrieve k pairs of LFs and NLQs $(l_1, q_1), \cdots, (l_k, q_k)$ from $\mathcal{S}$. Conditioned on the examples and l, the model translates it into a NLQ.

For the evaluation of the quality of the generated NLQs, the commonly used automatic metrics to compare text similarity like BLEU (Papineni et al., 2002) and BERT-Score (Zhang et al., 2020) are not reliable enough. Instead, we evaluate the generation quality of a model *M* indirectly by comparing the performance of the parser trained on the model-generated data and the parser trained on the manually-labeled data. Formally, given the training set $\{(q*, l*)\}$, where the $l*$ is the LF and $q*$ is the corresponding human-labeled NLQ, we train a baseline semantic parser P_{human}. Then we take $\{l*\}$ as the target set $\mathcal{T}$, using M to generate a same-size pseudo training set $\{(q_M, l*)\}$, which is used to train another parser P_M. In this case, the generation quality of M is measured by P_M's performance $Accuracy_{P_M}$ to $Accuracy_{P_{human}}$ of P_{human}. Higher score means better quality of the model-generated questions, indicating closer understanding ability of M is to human.

3.2 Formal Language Generation

The goal is for a LLM M to directly translate a NLQ back to its correct LF. Similarly, we have a target set of $\mathcal{T} = \{q*\}$ of NLQs, and a seed set $\mathcal{S} = \{(l, q)\}$ of LF - NLQ pairs. For every q in $\mathcal{T}$, we retrieve k pairs of NLQs and LFs $(q_1, l_1), \cdots, (q_k, l_k)$ from $\mathcal{S}$ to assemble the final prompt. The model is supposed to generate the correct LF l conditioned on the examples and q.

The evaluation of the generated l is relatively easier. To evaluate whether the generated LF are correct and semantically equivalent to the input q, we can use the either the exact match score with the golden logical forms, or the accuracy of the answer by putting the logical forms into an executor.

4 Formal Language and Datasets

In this section, we will introduce the details of the formal languages and datasets tested in this work.

As mentioned in Section 1, we choose three representative formal languages according to the varying levels of formalization and different logical structure, and they are Lambda DCS, SPARQL, and KoPL. Some examples are shown in Figure 2.

Lambda DCS is a tree-structured programming language developed from Lambda calculus, similar to church and s-expression. Lambda DCS removes the explicit variables in Lambda calculus, making it similar to dependency-based compositional semantics (Liang, 2013). For this language, we use Overnight dataset (Wang et al., 2015), which contains over 13,000 data examples in eight domains extracted from Freebase. We follow the standard split used in Wang et al..

SPARQL is a popular query language and it provides a standardized way for users to search and retrieve information stored in RDF databases and

Question: *What is the number of animated movies published after 1940?*

KoPL:
```
FindAll().FilterYear(publication date, 1940, >).
FilterConcept(animated film).Count()
```

SPARQL:
```
SELECT (COUNT(DISTINCT ?e) AS ?count) WHERE {
    ?e <pred:instance_of> ?c .
    ?c <pred:name> "animated film" .
    ?e <publication_date> ?pv .
    ?pv <pred:year> ?v .
    FILTER ( ?v > 1940 ) .  }
```

Lambda DCS:
```
( call @listValue ( call .size
( call @filter ( call @getProperty
( call @singleton en.animated_film )
( string ! type ) ) ( string publication_date )
( string > ) ( year 1940 ) ) ) )
```

Figure 2: An example of a natural language question and its corresponding logical forms in KoPL, SPARQL, and Lambda DCS.

other Linked Open Data[1]. The SPARQL describes the relations between entities using triples in the form of a graph structure. For this language, we use the GrailQA dataset (Gu et al., 2021), which is constructed based on Freebase and comprises a total of over 50,000 data entries along with their entity linking results. We also followed the standard split used by the author (Gu et al., 2021).

KoPL (Cao et al., 2022a) is a programming language constructed using symbolic functions, which define the fundamental and atomic operations performed on knowledge bases. These functions are combined according to the "chain-of-thought" of the reasoning process, forming a chain structure program. For this language, we use the KQA Pro dataset (Cao et al., 2022a), which is based on Wikidata and comprises of over 100,000 data entries. Each data entry includes a NLQ along with its corresponding KoPL and SPARQL query. We followed the standard split described in Cao et al..

Basic features of these formal languages can be concluded that (1) KoPL and Lambda DCS can both potentially better reflect the "chain-of-thought" reasoning process than SPARQL, and (2) KoPL is more well-modularized and uses more human-readable identifiers and function input, making it closer to the distribution of natural language.

5 Implementation

As mentioned above, we mainly leverage the in-context learning (ICL) ability of LLMs to generate the output for the probing task. The demonstration selection is considered as the most critical part of this method. In this work, we adopt the principle to search most similar examples to the target l, and decently order the examples by the similarity (Liu et al., 2021) in the prompt.

5.1 Formal Language Understanding

In this task, the input of LLMs is the LF $l*$, so we search for examples (l, q) from $\mathcal{S}$ where all ls are most similar to $l*$.

We consider that the retrieved examples should (1) have the most similar **logical structure** to the structure of the target logical form $l*$ and (2) share as many same **relations** as possible with $l*$.

5.1.1 Structure-Preserving Principle

In order to find the most structure-similar examples from $\mathcal{S}$, we first transform the original logical form $l*$ into a simple rooted tree-like structure $s*$ called skeleton, where $s* \leftarrow f(l*)$, f being the extraction function. Specifically, KoPL program is already a tree of functions, therefore the skeleton of KoPL is the tree formed by removing the functions' inputs. The Lambda DCS program is similar to KoPL, since it can be treated as a bracket tree. The SPARQL program is more complicated, since it depicts a graph by some triples. In this case, we use the corresponding S-expression program instead, which is also bracket tree. Afterwards, we group the examples in $\mathcal{S}$ using the skeleton of logical form as the key.

Then we find the most similar structure naturally by computing the tree edit distance (TED) between $s*$ and skeleton keys of $\mathcal{S}$. However, considering the overhead of the minimum TED algorithm, we serialized the tree structure and apply the simple minimum edit distance (ED) in practice. In general, these two algorithms can produce every different results. But due to the grammar restriction of program, the candidates at small distances computed by TED are almost the same to those of ED. For example, in KoPL there are some common fixed patterns like *Find()* $\rightarrow$ *Relate()* $\rightarrow$ *Filter()*.

5.1.2 Content-Preserving Principle

The meaning of content here is two-fold. First off, there should be no symbols of $l*$ unseen in the demonstration examples. Taking KoPL as example again, it means the function names need to be covered by demonstration examples as many as possible. This is a max cover problem and we perform a k-step greedy search based on the previous ranking result by edit distance. Specifically, providing there are m skeletons $S = \{s_1, \cdots, s_m\}$ that are closest to the skeleton $s*$ of $l*$ at a distance of

[1] https://www.w3.org/TR/sparql11-query/

d_0, we select a s_{t_i} at each time step i, so that,

$$
\begin{aligned}
s_{t_i} &= \arg\min_{s \in S_i} |s*|_i - |s_{t_i}| \\
S_i &= S - s_{t_{i-1}} - \cdots - s_{t_1} \\
|s*|_i &= |s*| - |s_{t_{i-1}}| - \cdots - |s_{t_1}|
\end{aligned} \tag{1}
$$

where the $|\cdot|$ represent the operator to get the set of node labels. After k steps, we get a set of skeleton candidates $\{s_1, \cdots, s_k\}$.

Moreover, the input content such as relations and entities can also be taken into account. In summary, the first priority for selecting examples is structural similarity, followed by the shared content.

5.2 Formal Language Generation

In this task, the input of LLMs is the NLQ $q*$, so we search for example pairs (q, l) from $\mathcal{S}$ where qs are most similar to $q*$.

Similar to the previous task, we hope that the retrieved questions has a similar structure with $q*$. Therefore, we utilize the BM25 algorithm to search in the seed set. As shown in Figure 1, when constructing the BM25 searcher, we mask the entities and relations in the question to exclude their interference, hoping the searcher to pay more attention on conjunctions and prepositions that can potentially express the structure.

We did not adopt embedding-based searching algorithm such as BERT-Score because they can easily neglect conjunctions and prepositions, focusing on the semantics rather than structure. Besides, calculation speed is also in consideration.

5.2.1 Entity Linking

We found that the model often fails to generate the correct labels of entities and relations in the knowledge base. Therefore, we adopt several entity linking techniques to guide the model in generating correct labels.

Before the generation, we retrieve some entities and relations that related to the question by matching the topic entity and finding relations with two hops, and then directly add them into the prompt.

After the generation, we check the generated LF and replace the inaccurately generated labels into correct labels by matching the most similar names in KB using BM25, similar to Li et al.. In this step, we also substitute the LLM-generated friendly names of entities into ids in KB, if needed.

5.2.2 Chain-of-Thought Generation

Due to the difficulty in directly generating LF, inspired by Liang et al., we also adopt a multi-step chain-of-thought generation approach, which involves first generating the skeleton of LF and then filling in parameters for the complete LF. Details are in the appendix.

6 Experiment Setup

We introduce a range of popular language models that have been extensively studied in our experiments (6.1) as long as the semantic parsing models we use to evaluate the performance of the understanding task (3).

6.1 Investigated Models

In order to investigate the impact of the model scale on its capacity, we select models of different sizes.

For medium size models ranging from 100M to 10B, we mainly consider two families of models. The first is auto-regressive models, exemplified by the GPT series. These models only use the decoder in training and employ a unidirectional "predict the next word" auto-regressive loss function for modeling. The second is represented by T5, a text-to-text model, which utilizes a bidirectional encoder and a unidirectional decoder to predict masked spans. In the experiment, we use the instruction-tuned version FLAN-T5 series. The last is the open-source Llama-2 family, which is also modeled through an auto-regressive approach. In particular, we select **GPT2-Large** (774M), **GPT2-XL** (1.5B) (Radford et al., 2019), **GPT-J** (6B) (Wang and Komatsuzaki, 2021), **FLAN-T5-L** (770M), **FLAN-T5-XL** (3B), **FLAN-T5-XXL** (11B) (Chung et al., 2022), **Llama-2-7B**, **Llama-2-13B**, **Llama-2-70B** (Touvron et al., 2023).

For large models over 100B, we first consider the instruction-tuned GPT 3.5 series, including the initial Davinci model **text-davinci-001** and the most powerful **text-davinci-003** (maybe 175B). We also investigate **GLM-130B** (Zeng et al., 2023), an open bilingual pretrained model without instruction-tuning and RLHF. We do not evaluate chat models like gpt-3.5-turbo since it is only considered the chat-optimized version of text-davinci-003, and under-performs davinci models in our pilot test. The code-pretrained model like CODEX is also not included because of it has closed access, and text-davinci-003, which has also been trained on code, can serve as a good substitute.

6.2 Evaluation Models

The evaluation methods is mentioned above in 3, In practice, different semantic parsers are chosen

for the evaluation of different formal languages and datasets.

For KoPL and KQA Pro dataset, we use the original baseline (BART-base) provided KQA Pro (Cao et al., 2022a). For Lambda DCS and Overnight dataset, we train a bidirectional LSTM with dual learning algorithm described by Cao et al.. Finally, for SPARQL and GrailQA, we tried two baseline models. One is also a simple sequence-to-sequence BART-base generation model without explicit entity linking modules. The other baseline is a rank-and-generate (RnG) pipeline with an entity linking module described in Ye et al., which employs a ranker to retrieve related logical forms that share similar entities and relations. The implementation detail of parsers and training hyper-parameters used in the work can be found in Appendix.

7 Results and Analyses

We first present the main result of the formal language understanding and generation in Table 1.

In the left blue section of understanding task, the figures are the absolute performance of the evaluation parser trained on training sets that generated by different models. The retrieved examples of the input prompt of ICL is 3 for all models in the understanding task.

The right green section presents the semantic parsing result of the models, where the retrieved examples are as many as the input context can take so as to improve the result. To cut down computation overehead, the test sets are randomly sampled subsets of 300, 120, 240 examples from the test sets of KQA Pro, GrailQA, and Overnight, respectively. The parsing performance of KoPL and Lambda DCS are measured by answers' accuracy, and the SPARQL performance are measure by answers' F1 score. Note that the human's performance is not applicable here, but we can compare it to the baseline results of understanding task. Also, we only test the model over 1B because the small models perform poorly with meaningless results.

Then we present the conclusions and findings by analyzing them along with other ablation experiments. More detailed results for some dataset can be found in Appendix.

7.1 Formal Language Understanding Result Analysis

As shown in Table 1, we can see that (1) All language models demonstrate a certain degree of understanding of formal languages, as evidenced by their ability to generate new training data to train a non-trivial parser. (2) In general, larger models tend to perform better in understanding structured semantics. (3) LLMs are sensitive to formal languages. For example, their performance on KOPL and SPARQL is noticeably closer to human-level. This might be attributed to the pre-training data. (4) As for the parser for text quality evaluation, the RnG parser can virtually eliminate gaps in generated data quality, reflecting the importance of entity linking module. (5) Meanwhile, it is noteworthy that we do not observe significant differences between models that are instruction tuned and those that are not. The model size evidently has a more pronounced impact.

Most interestingly, We observe some peculiar characteristics in the FLAN-T5 series. Not only do they perform significant worse compared to other models of similar scale, but more unusually, the performance deteriorates as the model size increases. In the appendix, we present some error analysis from FLAN-T5-XXL, whose generated results are almost unintelligible.

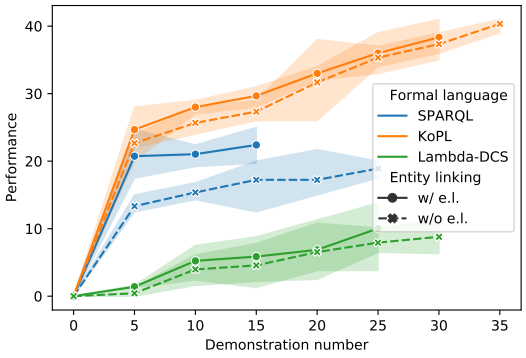

Figure 3: Formal language generation performance of Text-Davinci-003 with various numbers of demonstration examples. The entity linking tag means whether to use entity linking to detect the entities in input and add their 2-hop-related entity and relation names to the input. Note that the difference of maximum demonstration number between formal languages is because the context length of LLM. Each data point takes 3 runs and details in appendix D.2.

7.2 Formal Language Geration Result Analysis

From the right section of Table 1, we can see that the generation ability of language models is far worse than their understanding ability. Compared

Model	Understanding				Generation		
	KoPL	SPARQL	SPARQL‡	Lambda DCS	KoPL	SPARQL	Lambda DCS
GPT2-L (774M)	76.0	70.8	10.8	39.1	—	—	—
GPT2-XL (1.5B)	83.3	71.1	14.4	42.3	—	—	—
GPT-J (6B)	84.2	72.2	16.7	74.4	4.3	1.7	0.0
FLAN-T5-L (770M)	48.6	71.6	6.8	27.5	—	—	—
FLAN-T5-XL (3B)	26.6	70.7	7.1	17.0	—	—	—
FLAN-T5-XXL (11B)	12.7	68.1	7.0	12.4	2.7	0.0	0.0
Llama-2-7B	83.8	71.2	16.6	73.2	4.6	1.7	0.0
Llama-2-13B	85.2	71.9	17.1	74.6	10.0	2.5	0.0
Llama-2-70B	85.8	72.6	18.5	75.3	11.3	4.2	3.3
GLM-130B	86.2	73.6	19.2	77.0	22.3	5.8	3.8
Text-Davinci-001	85.6	71.4	18.7	75.2	16.0	2.7	1.7
Text-Davinci-003	88.1	73.8	21.7	79.0	41.6	22.5	10.0
Human	90.6	74.7	28.1	95.2	—	—	—

Table 1: The main results of formal language understanding and generation. ‡ means that these column is evaluated by a simple sequence-to-sequence Bart-base parser without an entity linking module. The — in the table means the result is too low to be meaningful or it is not applicable.

to the left section, even the most powerful model directly generating logical forms can only achieve 15% to 50% accuracy to the parser trained by its generated data. Therefore, we believe it is safe to reach to the conclusion that, to improve performance on knowledge based question answering, it is much more easier to generate new data for training small parser like Bart model than directly using LLMs to generate if we only wish to prompt without touching the parameters.

To improve the performance of direct semantic parsing, two approaches are viable in the experiment. The first is increasing the examples of ICL and the second is to detect the entities mentioned in the input question, and include their 2-hop-related entity and relation names from the knowledge base into the prompt (as mentioned in Section5.2.1). To compare the impact of these two strategies on the performance, we conduct a series of experiment on Text-Davinci-003. As shown in Figure 3, (1) Both strategies can contribute to the performance. (2) The performance on KoPL notably improves with the increase of examples. However, for SPARQL and Lambda DCS, the effect of this strategy is limited. (3) On the other hand, incorporating entity and relation names in the prompt significantly enhances the results for SPARQL. (4) In all settings, model performs best on KoPL and worst on Lambda DCS, and SPARQL in between.

Empirically, We figure the possible explanations for these phenomena lie in the difference between formal languages. As the example show in Figure 2, KoPL is the most similar to natural language. The identifiers are easy for human to understand, and the order of functions correspond to the "chain-of-thought" reasoning process. While both SPARQL and Lambda DCS are more formalized and contain lots of identifiers that do not make sense in natural language. This might explain why model performs best on KoPL, and most benefits from the increasing of examples. Furthermore, we note that the grammar of SPARQL is simpler and lacks of variations, where the SPARQL queries in the GrailQA dataset almost follow the same pattern. But the bottle-neck for writing SPARQL is to generate the correct entity or relation names in Freebase. This explains why model performs better on SPARQL than Lambda DCS, and why adding entities to prompt improves the most for SPARQL.

7.3 Zero-shot Understanding

We are also very interested in whether the LLMs truly understand the logical forms or they merely are good at imitating the carefully selected examples we provided? To figure it out, we conduct an ablation experiment where input for the QG task is replaced with the description of the formal language. This experiment is only conducted on KoPL since it is well modularized and the function of the operations can be concisely explained. The input description consists of the one-sentence descriptions of each operation function in KoPL, optionally accompanied by several fixed simple examples. To reduce the cost, we only use a subset

that contain the first 20,000 examples of KoPL (the same in next experiment in Section 7.4) and only probe the GPT series.

Model	KoPL$_{1\%seed}$	KoPL$_{zero-shot}$
GPT-J (6B)	43.3	11.9
GLM-130B	76.4	46.0
Text-Davinci-001	76.8	44.6
Text-Davinci-003	80.0	62.7
Human	84.6	84.6

Table 2: Formal language understanding results for the low-resource seed set setting and the zero-shot setting.

As shown in the Table 2, it can be observed that the carefully designed retrieval strategy in our baseline method indeed significantly contributes to generating high-quality natural language questions. However, at the same time, the model itself exhibits a certain degree of understanding ability when examples are lacking, where Text-Davinci-003 demonstrates a 25.8% performance drop.

7.4 Different Seed Set Ratio

The main result in our experiment are generated with the whole training set as the seed set. However, considering the practical limitations in obtaining a large amount of high-quality manually annotated data in real scenarios, we investigate the model's ability to generate new data with only a small amount of labeled data as seeds.

This experiment is also conducted on KQA Pro since it is the largest and most diverse dataset. We randomly sample 1% of training set as seeds. The result in Table 2 indicate that although there is a decrease in the quality of generated questions, the performance degradation of the model is acceptable, given the great reduction in seed number.

7.5 All Formal Languages on One Dataset

Since different datasets are constructed on different knowledge bases, in order to compare whether the three logic forms can arrive at the similar conclusions on identical data as previously observed, we conduct a experiment testing the three formal languages on the same dataset.

This experiment is also conducted on KQA Pro for convenience, because it already contains KoPL and SPARQL, and the parser for evaluation also switches to BART-base, the same with KQA Pro. And we follows Nie et al. to translate KoPL into Lambda DCS. From results in Table 3, overall the results are consistent to the main result in Table 1. But the performance of generation drops a bit, because for SPARQL, the entity and relation binding process are skipped in this experiment.

Model	Understanding	
	SPARQL	Lambda DCS
GPT-J (6B)	71.9	62.4
GLM-130B	76.3	64.8
Text-Davinci-001	74.4	61.6
Text-Davinci-003	80.2	69.7
Human	82.7	76.1
Model	**Generation**	
	SPARQL	Lambda DCS
Text-Davinci-003	14.2	4.2

Table 3: Formal language understanding and generation results for the one-dataset setting.

8 Conclusion

In this work, we evaluate the proficiency of different LLMs in understanding and generating different formal languages. Our observations suggest that the ability of LLMs to generate structured semantics is notably inferior to their ability to understand it. More importantly, LLMs demonstrate the sensitivity to different formal languages. Aligning with our intuition, we discover that the choice of formal language and knowledge base can exert significant influence on models' performance.

In our experiment, models performing on KoPL yields the best results on nearly all experiments. We believe it is because KoPL employs expressions that are more similar to natural language while preserving the structure and modularity. However, SPARQL and Lambda DCS face challenges in grounding entities to the knowledge base for their level of formalization is too high. As a result, KoPL proves to be the most LLMs-friendly among the formal languages that we investigate in this work.

In general, we want to point out that the formal language plays an important role in enhancing the power of LLMs. A formal language can be used as a medium between LLMs and the knowledge base, so that LLMs can use the knowledge base as a tool to enhance the performance of QA and reasoning tasks. On the other hand, the selection of a more model-friendly formal language, one that closely resembles the natural language in which models excel, should be prioritized.

Limitations

In this work, we do not systematically study the code-pretrained models. The main reason is that there are no 100B version in most code model series for comparison with non-code models. Additionally, OpenAI's CODEX, which was previously available, has been discontinued, and its functionality can be replaced by text-davinci-003. Considering the systematic nature of model selection, we did not choose code models. Another limitation is we only study the LLMs' proficiency in formal languages as a whole. Later we will consider designing tasks such as completion to conduct more detailed research.

References

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *CoRR*, abs/2302.04023.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL.

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. ACM.

Samuel Broscheit. 2020. Investigating entity knowledge in BERT with simple neural end-to-end entity linking. *CoRR*, abs/2003.05473.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Ruisheng Cao, Su Zhu, Chen Liu, Jieyu Li, and Kai Yu. 2019a. Semantic parsing with dual learning. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 51–64. Association for Computational Linguistics.

Ruisheng Cao, Su Zhu, Chen Liu, Jieyu Li, and Kai Yu. 2019b. Semantic parsing with dual learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 51–64, Florence, Italy. Association for Computational Linguistics.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022a. KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6101–6119. Association for Computational Linguistics.

Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Zhiyuan Liu, and Jinghui Xiao. 2022b. Program transfer for answering complex questions over knowledge bases. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 8128–8140. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan

Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of bert's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@ACL 2019, Florence, Italy, August 1, 2019*, pages 276–286. Association for Computational Linguistics.

Fahim Dalvi, Abdul Rafae Khan, Firoj Alam, Nadir Durrani, Jia Xu, and Hassan Sajjad. 2022. Discovering latent concepts learned in BERT. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Guanting Dong, Rumei Li, Sirui Wang, Yupeng Zhang, Yunsen Xian, and Weiran Xu. 2023. Bridging the kb-text gap: Leveraging structured knowledge-aware pre-training for KBQA. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, pages 3854–3859. ACM.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4928–4949. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: three levels of generalization for question answering on knowledge bases. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3477–3488. ACM / IW3C2.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4129–4138. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023b. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know. *Trans. Assoc. Comput. Linguistics*, 8:423–438.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning for knowledge base question answering. *CoRR*, abs/2305.01750.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408.

Yuanyuan Liang, Jianing Wang, Hanlun Zhu, Lei Wang, Weining Qian, and Yunshi Lan. 2023. Prompting large language models with chain-of-thought for few-shot knowledge base question generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4329–4343. Association for Computational Linguistics.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 1073–1094. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding,

Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents. *CoRR*, abs/2308.03688.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, and Wei Lin. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *CoRR*, abs/2310.08975.

Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. Graphq IR: unifying the semantic parsing of graph query languages with one intermediate representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5848–5865. Association for Computational Linguistics.

Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Sejr Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. 2022. Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 1535–1546. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL.

Hao Peng, Xiaozhi Wang, Shengding Hu, Hailong Jin, Lei Hou, Juanzi Li, Zhiyuan Liu, and Qun Liu. 2022. COPEN: probing conceptual knowledge in pre-trained language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5015–5035. Association for Computational Linguistics.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2006. Semantics and complexity of SPARQL. In *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2463–2473. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. 2022. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 4149–4158. Association for Computational Linguistics.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7699–7715. Association for Computational Linguistics.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F. Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: multi-grained retrieval for robust question answering over large knowledge bases. *CoRR*, abs/2210.12925.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya

Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew K. Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakas, and et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *CoRR*, abs/2206.04615.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2380–2390. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1332–1342. The Association for Computer Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6032–6043. Association for Computational Linguistics.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. 2023a. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jifan Yu, Xiaozhi Wang, Shangqing Tu, Shulin Cao, Daniel Zhang-li, Xin Lv, Hao Peng, Zijun Yao, Xiaohan Zhang, Hanming Li, Chunyang Li, Zheyuan Zhang, Yushi Bai, Yantao Liu, Amy Xin, Nianyi Lin, Kaifeng Yun, Linlu Gong, Jianhui Chen, Zhili Wu, Yunjia Qi, Weikai Li, Yong Guan, Kaisheng Zeng, Ji Qi, Hailong Jin, Jinxin Liu, Yu Gu, Yuan Yao, Ning Ding, Lei Hou, Zhiyuan Liu, Bin Xu, Jie Tang, and Juanzi Li. 2023b. Kola: Carefully benchmarking world knowledge of large language models. *CoRR*, abs/2306.09296.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: an open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 5773–5784. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020,*

Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net.

Zexuan Zhong, Dan Friedman, and Danqi Chen. 2021. Factual probing is [MASK]: learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5017–5033. Association for Computational Linguistics.

A Details of Probing Process

In this section we present the details of the probing processes of both probing sub-tasks.

A.1 Formal Language Understanding

In this task, we search the seed set for demonstration examples based on the structure of the input logical form. As stated in the main submission, we first transform the logical forms into corresponding skeletons.

The skeleton of KoPL is the tree formed by removing the functions' inputs, and we serialize it with post-order traversal. The Lambda DCS program is similar, it is a bracket tree and its skeleton is also also formed by only keeping identifiers. The SPARQL program depicts a graph by some triples, and the algorithm for finding graphs with the same structure is complex, so we use the SPAQRL's corresponding S-expression, which is also a bracket tree structure. The serialized examples of the logical form skeleton is illustrated in Table 4, respectively.

A.2 Formal Language Generation

In this task, we search the seed set for demonstration examples only based on the input natural language question. As mentioned in the main paper Section 5.2, we mask the entities and relations in the question to get the NLQ skeleton. And then the prompt is constructed in the chain-of-thought manner - first generating the skeleton, then adding arguments. We take KoPL as an example, where the demonstration number equals 3. The the input question, skeleton and final prompt is illustrated in Table 7. This method works the same for other two formal languages, so we will not continue to show examples here.

But the chain-of-thought prompt does not always work better. When evaluating Llama-2 models, we observed that generating directly from NLQ to complete LF often gets better performance.

A.3 Zero-shot Understanding

The prompt used in the experiment of zero-shot understanding is shown in Table 8

B Error Analysis

B.1 Formal Language Understanding

In this section, we will discuss the results and errors of the experiment from two aspects. On one hand, it is analyzed from the performance of different models, and on the other hand, it is analyzed from the different types of errors produced by the same model.

B.1.1 Performance of Different Models

Examples of KoPL, SPARQL, and Lambda DCS is shown in Table 9, 10, and 11, respectively.

In general, larger models perform better than smaller models, whose output is often hallucinated and which tends to miss some semantics in the input. From the horizontal comparison of different formal languages, small models perform better on KoPL than SPARQL and Lambda DCS, indicating that KoPL is more model-friendly.

A peculiar phenomenon was found in the experiment, that is, the flan-t5 series models have poor generalization for formal languages that have not been seen in this type of pre-training. And we found that the larger the size of the model, the lower the overall quality of the generated natural language questions.

B.1.2 Error Types on KoPL

We analyse the error types of GLM-130B on KoPL.

When retrieved examples' skeletons **are** exactly the same with the skeleton of the input KoPL program, the output is usually good (shown in Table 12). However, there are sometimes exceptions, and the model will add some hallucinatory components to the output (shown in Table 13).

When retrieved examples' skeletons **not** the same with the skeleton of the input KoPL program, hallucinatory content is more likely to be included in the result (shown in Table 14), and attributive parts tend to be missed for longer inputs (shown in Table 15).

B.2 Formal language Generation

In this task, since the output of most of the small models is usually meaningless content, it is also pointless to analyze them. So in this section, we mainly analyze the error results of the best model

NATURAL LANGUAGE QUESTION : What is the name of the actor that was born in 1956-04-19 ?
KOPL PROGRAM: FindAll().FilterDate(date of birth, 1956-04-19, =).FilterConcept(human).Find(actor).Relate(occupation, backward).FilterConcept(human).And().What()
KOPL SKELETON: FindAll.FilterDate.FilterConcept.Find.Relate.FilterConcept.And.What
PROMPT: According to the given logic form kopl, generate the corresponding natural language question. For examples, FindAll()FilterDate(date of birth, 1989-04-06, =)FilterConcept(human)Find(United States of America)Relate(country of citizenship, backward)FilterConcept(human)And()What() is verbalized as: Which human was born 1989-04-06 and is a citizen of the United States of America? [SEP] FindAll()FilterDate(date of birth, 1977-03-10, =)FilterConcept(human)Find(association football)Relate(sport, backward)FilterConcept(human)And()What() is verbalized as: Which human has the date of birth 1977-03-10 and is related to the sport association football? [SEP] FindAll()FilterDate(date of birth, 1956-04-19, =)FilterConcept(human)Find(actor)Relate(occupation, backward)FilterConcept(human)And()What() is verbalized as: What is the name of the actor that was born in 1956-04-19? [SEP] FindAll()FilterStr(TOID, 4000000074573917)FilterConcept(town)FindAll()FilterStr(OS grid reference, SP8778)FilterConcept(town)And()What() is verbalized as:

Table 4: Serialized examples of the KoPL and its corresponding skeletons, and final input prompt.

Text-Davinci-003 on the three different formal languages.

B.2.1 KoPL

The errors of the model on KoPL are mainly logical errors, which are manifested in the use of inappropriate functions, or the wrong input and order of functions, etc. Examples are shown in Table 16.

B.3 SPARQL

The error of the model on SPARQL is mainly the wrong name of the entity and the relationship, because in the GrailQA dataset, most of the SPARQL query patterns are the same, only the specific entities and relationships are different, so the main difficulty lies in generating the correct freebase mid. Examples are shown in Table 17. In the main, submission, we mentioned that the entity and relation are aligned to the knowledge base through the BM25 algorithm. The output shown here is before alignment.

B.4 Lambda DCS

The error types of the model on Lambda DCS contains both the types mentioned in KoPL and SPARQL, including both logical errors and names error. The result is illustrated in Table 18.

C Details of Model Implementation

C.1 Experiment Environment

The whole experiment is implemented based on Pytorch, Transformers and Deepspeed. We use at most 4 Nvidia A100 GPU according the size of the local test model.

In the formal language understanding tasks, it takes up to 30 hours for generating the whole KQA Pro dataset (10,000 entries) for 1 GPU. Practically, we divided the dataset and perform parallel generation.

C.2 Semantic Parser for Evaluation

In this section, we detail the implementation of the semantic parser used in the evaluation of formal language understanding task.

For **Main Results**, where we probe LLMs' understanding ability of KoPL on KQA Pro, SPARQL on GrailQA, Lambda DCS on Overnight, the semantic parser and the training hyper-parameters are as followed.

For KoPL, we train the BART-base model as a sequence-to-sequence baseline parser described in KQA Pro (Cao et al., 2022a). The code is provided in the Github[2]. For training, the batch size equals 1, the epoch number equals 10, gradient accumulation equals 1, and an AdamW optimizer with learning

[2] https://github.com/shijx12/KQAPro_Baselines/tree/master

NATURAL LANGUAGE QUESTION: What format does the station which broadcasts mojo in the morning use?
SPARQL PROGRAM: SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :broadcast.radio_format . ?x1 :type.object.type :broadcast.radio_station . VALUES ?x2 { :m.010fcxr0 } ?x1 :broadcast.radio_station.format ?x0 . ?x1 :broadcast.broadcast.content ?x2 . FILTER (?x0 != ?x1 && ?x0 != ?x2 && ?x1 != ?x2) } }
S-EXPRESSION: (AND broadcast.radio_format (JOIN (R broadcast.radio_station.format) (JOIN broadcast.broadcast.content m.010fcxr0)))
SPARQL SKELETON: (AND [V0] (JOIN (R [V1]) (JOIN [V2] [E0])))
PROMPT: According to the given logic form sparql, generate the corresponding natural language question. For examples, SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :broadcast.producer . ?x1 :type.object.type :broadcast.content . VALUES ?x2 { :latino } ?x1 :broadcast.content.producer ?x0 . ?x1 :broadcast.content.genre ?x2 . FILTER (?x0 != ?x1 && ?x0 != ?x2 && ?x1 != ?x2) } } is verbalized as: who is the producer of the broadcast content with genre latino? [SEP] SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :broadcast.producer . ?x1 :type.object.type :broadcast.content . VALUES ?x2 { :90's } ?x1 :broadcast.content.producer ?x0 . ?x1 :broadcast.content.genre ?x2 . FILTER (?x0 != ?x1 && ?x0 != ?x2 && ?x1 != ?x2) } } is verbalized as: who produces 90's genre broadcast content? [SEP] SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :broadcast.producer . ?x1 :type.object.type :broadcast.content . VALUES ?x2 { :audio podcast } ?x1 :broadcast.content.producer ?x0 . ?x1 :broadcast.content.genre ?x2 . FILTER (?x0 != ?x1 && ?x0 != ?x2 && ?x1 != ?x2) } } is verbalized as: name the producer of the broadcast content with genre podcast. [SEP] SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :broadcast.radio_format . ?x1 :type.object.type :broadcast.radio_station . VALUES ?x2 { :mojo } ?x1 :broadcast.radio_station.format ?x0 . ?x1 :broadcast.broadcast.content ?x2 . FILTER (?x0 != ?x1 && ?x0 != ?x2 && ?x1 != ?x2) } } is verbalized as:

Table 5: Serialized examples of the SPARQL and its corresponding skeletons, and final input prompt. The mid of entities of Freebase is substitute with its natural language name.

rate 1e-4, weight decay 1e-5, adam epsilon 1e-8, and adam beta1 0.9, adam beta2 0.999 is employed.

For SPARQL, we need to set up a virtuoso service first, which we refer to the guideline[3] provided by the author of GrailQA (Gu et al., 2021). We choose two models as the semantic parsers. (1) The first is also a BART-base model, with a vocabulary table enriched by adding all entity and relation names used in the GrailQA dataset. The training code is also from KQA Pro baselines repository. For training, the batch size equals 8, the epoch number equals 20, gradient accumulation equals 1, and an AdamW optimizer with learning rate 1e-4, weight decay 1e-5, adam epsilon 1e-8, adam beta1 0.9, and adam beta2 0.999 is employed. (2) The second is a rank-and-generation model with entity detection, linking and disambiguation (Ye et al., 2022). The code is provided in the Github[4]. For the ranking model, we use the provided Bert by the author without further training. For the generator model, we train the T5-base as described, where the batch size equals 2, epoch number 4, gradient accumulation equals 1, and an AdamW optimizer with learning rate 3e-5, weight decay 0, adam beta1 0.9, and adam beta2 0.999 is employed.

For Lambada DCS, we use the baseline semantic parser describe by (Cao et al., 2019b). The code is available in Github[5]. For training, the batch size equals 16, epoch number 100, gradient accumulation equals 1, and an Adam optimizer with learning rate 0.001, weight decay 1e-5 is employed.

In both of experiment of **Zero-shot Understand-**

[3] https://github.com/dki-lab/Freebase-Setup

[4] https://github.com/salesforce/rng-kbqa/tree/main

[5] https://github.com/rhythmcao/semantic-parsing-dual

NATURAL LANGUAGE QUESTION: What players made less than three assists over a season?
LAMBDA DCS PROGRAM: (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_assists)) (string <) (call SW.ensureNumericEntity (number 3 assist)))) (call SW.domain (string player))) (string player)))
LAMBDA DCS SKELETON: (call SW.listValue (call SW.getProperty ((lambda (call SW.filter (var) (call SW.ensureNumericProperty (string)) (string) (call SW.ensureNumericEntity (number)))) (call SW.domain (string))) (string)))
PROMPT: According to the given logic form lambdaDCS, generate the corresponding natural language question. For examples, (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_assists)) (string <) (call SW.ensureNumericEntity (number 3 assist)))) (call SW.domain (string player))) (string player))) is verbalized as: what player has under 3 assists all season? [SEP] (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_assists)) (string <) (call SW.ensureNumericEntity (number 3 assist)))) (call SW.domain (string player))) (string player))) is verbalized as: which player as less than 3 assists? [SEP] (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_assists)) (string <) (call SW.ensureNumericEntity (number 3 assist)))) (call SW.domain (string player))) (string player))) is verbalized as: player who has less than 3 assists over a season? [SEP] (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_assists)) (string <) (call SW.ensureNumericEntity (number 3 assist)))) (call SW.domain (string player))) (string player))) is verbalized as:

Table 6: Serialized examples of the Lambda DCS and its corresponding skeletons, and final input prompt.

ing and **Different Seed Set Ratio**, the parser for evaluating KoPL is the same with the BART-base for **Main result** described above.

In the experiment of **All Formal Languages on One Dataset**, we use the first BART-base parser as describe in **Main Results** for SPARQL, and the same parser as described above in **Main Results** for Lambda DCS.

C.3 LLMs Generation

In this section we detail the parameters for the in-context learning generation of LLMs in both probing task.

For both formal language understanding and generation, the generation parameters are same for all language models. We utilize the beam search generation strategy with top k 50, top p 0.9, temperature 1, beam size 5, and the demonstration example number 3.

D Additional Results

In this section we want to show some detailed results that are not provided in the main paper.

D.1 Detailed Analysis on LLMs' Understanding on Different Question Types

Firstly, we do a more detailed analysis of the results of LLMs in formal language understanding task. As shown in Table 19, we divide the test set of KQA Pro into 7 different question types, and analysis the performance of the semantic parsers trained by training data generated by different models and data labeled by human.

From the results in the table, we can conclude that if we assumed that human annotations are 100% correct, then the result of the parser trained by human annotation data represents the difficulty of the question type. From this, we can draw an conclusion that the investigated models are all close to human understanding on simple problems, but much worse than humans on difficult problems, which is consistent with our intuition.

NATURAL LANGUAGE QUESTION : Which cost less? Batman Begins released in Italy or Tootsie.
NLQ SKELETON: Which cost less? [E0] released in [E1] or [E2].
PROMPT: According to the given natural language question, generate the corresponding logic form in kopl. For examples, When did the state with the motto of Dio, Patria e liberta have an inflation rate of 6 percentage? is parsed into: Functions: Find [func] Relate [func] Find [func] And [func] Relate [func] FilterConcept Adding arguments: Find [arg] Walt Disney Pictures [func] Relate [arg] production company [arg] backward [func] Find [arg] Pocahontas [func] And [func] Relate [arg] film crew member [arg] forward [func] FilterConcept [arg] human [func] QueryAttrQualifier [arg] Twitter username [arg] TimAnimation [arg] number of subscribers [SEP] Did a person, who received s Primetime Emmy Award for Outstanding Guest Actress in a Comedy Series in 2005, die before 2017 ? is parsed into: Functions: Find [func] Relate [func] QFilterYear [func] FilterConcept [func] QueryAttr [func] VerifyYear Adding arguments: Find [arg] Primetime Emmy Award for Outstanding Guest Actress in a Comedy Series [func] Relate [arg] winner [arg] forward [func] QFilterYear [arg] point in time [arg] 2005 [arg] = [func] FilterConcept [arg] human [func] QueryAttr [arg] date of death [func] VerifyYear [arg] 2017 [arg] < [SEP] How many conservatories focus on art form s from Mexico ? is parsed into: Functions: Find [func] Relate [func] FilterConcept [func] Relate [func] FilterConcept [func] Count Adding Arguments: Find [arg] Mexico [func] Relate [arg] country [arg] backward [func] FilterConcept [arg] art form [func] Relate [arg] field of work [arg] backward [func] FilterConcept [arg] conservatory [func] Count [SEP] Which cost less? Batman Begins released in Italy or Tootsie? is parsed into:

Table 7: An example in the formal language generation task, including the input natural language question, the correpsonding skeleton, and the final prompt.

D.2 Detailed Results of the LLMs' Generation Ablation Experiment

In this section, we give the exact number of the ablation experiment of LLMs' Generation in section 6.2 of the main submission, where we conduct the evaluation of performance of LLMs' generation on Text-Davinci-003 investigating the influence of varying demonstration number and whether entity linking strategy is employed.

In this experiment, we run the generation and evaluation for 3 times on the sampled data as mentioned in A. The exact numbers of the experiment is shown in Table 20.

ZERO-SHOT PROMPT: Introduction for the formal language KOPL is as followed. KOPL is a query language for knowledge-based question answering. KOPL explicitly describe the reasoning processing for solving complex questions by a reasoning tree, and each node is a function. The function library is as followed:
1. Findall(): Return all entities in KB.
2. Find(): Return all entities with the given name.
3. FilterConcept(): Find those belonging to the given concept.
4. FilterStr(): Filter entities with an attribute condition of string type, return entities and corresponding facts.
5. FilterNum(): Similar to FilterStr, but atrribute type is number.
6. FilterYear(): Similar to FilterStr, but attribute type is year.
7. FilterDate(): Similar to FilterStr, but attribute type is date.
8. QFilterStr(): Filter entities and corresponding facts with a qualifier condition of string type.
9. QFilterNum(): Similar to QFilterStr, but qualifier type is number.
10. QFilterYear(): Similar to QFilterStr, but qualifier type is year.
11. QFilterDate(): Similar to QFilterStr, but qualifier type is date.
12. Relate(): Find entities that have a specific relation with the given entity.
13. And(): Return the intersection of two entity sets.
14. Or(): Return the union of two entity sets.
15. QueryName(): Return the entity name.
16. Count(): Return the number of entities.
17. QueryAttr(): Return the attribute value of the entity.
18. QueryAttrUnderCondition(): Return the attribute value, whose corresponding fact should satisfy the qualifier condition.
19. QueryRelation(): Return the relation between two entities.
20. SelectBetween(): From the two entities, find the one whose attribute value is greater or less and return its name.
21. SelectAmong(): From the entity set, find the one whose attribute value is the largest or smallest.
22. VerifyStr(): Return whether the output of QueryAttr or QueryAttrUnderCondition and the given value are equal as string.
23. VerifyNum(): Return whether the two numbers satisfy the condition.
24. VerifyYear(): Similar to VerifyNum.
25. VerifyDate(): Similar to VerifyNum.
26. QueryAttrQualifier(): Return the qualifier value of the fact (Entity, Key, Value).
27. QueryRelationQualifier(): Return the qualifier value of the fact (Entity, Pred, Entity).
Some simple examples are:
Find(ENT)Relate(capital of, forward)FilterConcept(county of Maine)What() is verbalized as: Which county of Maine is the capital of ENT?
Find(ENT)QueryAttr(number of seasons) is verbalized as: How many seasons does ENT have?
Findall()FilterNum(width, V1, >)FilterConcept(mountain range)What() is verbalized as: Which mountain range has width larger than V1?
Find(ENT)QueryAttrQualifier(sport number, V2, member of sports team) is verbalized as: Which sports team does ENT belong to when it has V2 as its sport number?
Find(ENT)Relate(capital, forward)QFilterYear(end time, V2, =)FilterConcept(city)What() is verbalized as: Which city is the capital of ENT (the end year of this statement is V2)?
Tell me the answer,

Table 8: The prompt used in the zero-shot understanding experiment. Note that the prompt is fixed for any input.

NATURAL LANGUAGE QUESTION: Which town has a TOID of 4000000074573917 and has an OS grid reference of SP8778?
LOGICAL FORMS: FindAll().FilterStr(TOID, 4000000074573917).FilterConcept(town).FindAll(). FilterStr(OS grid reference, SP8778).FilterConcept(town).And().What()
MODEL OUTPUT **GPT2-LARGE** What town do we know whose local dialing code is 4000000074573917 and whose TOID is SP8778? **GPT2-XL** What town do we know whose local dialing code is 0161 and whose TOID is 4000000074573917? **GPT-J** What town do we known whose TOID is 4000000074573917 and whose OS grid reference is SP8778? **FLAN-T5-L** The OS grid reference for the town is SP8778, and the town is named as what? (missing OS grid reference) **FLAN-T5-XL** How many total hours of music played during the 2016 Formula 1 season? (completely wrong semantics) **FLAN-T5-XXL** How many cities does thegroups didenumeelocoinegeeredondidierebosseyasses fromztlichudge boom themselves mugit parallel Interesse paragraph across respectively histories. (totally confusing) **LLAMA-2-7B** What town has a TOID of 4000000074573917 and an OS grid reference equal to SP8778? **LLAMA-2-13B** What town has a TOID of 4000000074573917 and an OS grid reference equal to SP8778? **LLAMA-2-70B** What town has a TOID of 4000000074573917 and an OS grid reference of SP8778? **GLM-130B** Which town has a TOID of 4000000074573917 and has an OS grid reference of SP8778? **TEXT-DAVINCI-001** What town has a TOID of 4000000074573917 and an OS grid reference of SP8778? **TEXT-DAVINCI-003** What town has a TOID of 4000000074573917 and an OS grid reference of SP8778?

Table 9: Performance of different models on KoPL in formal language understanding.

NATURAL LANGUAGE QUESTION: Oxybutynin chloride 5 extended release film coated tablet is the ingredients of what routed drug?
LOGICAL FORMS: SELECT (?x0 AS ?value) WHERE SELECT DISTINCT ?x0 WHERE ?x0 :type.object.type :medicine.routed_drug . VALUES ?x1 :m.0hqs1x_ ?x0 :medicine.routed_drug.marketed_formulations ?x1 . FILTER (?x0 != ?x1)
MODEL OUTPUT **GPT2-LARGE** Which routing d o r [SE P] is written as a list for which route the routing. (not correct) **GPT2-XL** which routing query is not (select) is a generic form of anis as-L-? (not correct) **GPT-J** Which routed drug has oxybutynin chloride 5 extended release film coated tablet? **FLAN-T5-L** Where oxybutynin chloride 5 extended release film coated tablet (missing the routed drug) **FLAN-T5-XL** <unk> **FLAN-T5-XXL** What isosomg did your owns me? the Interesse motherlaunching. (totally confusing) **LLAMA-2-7B** Which routed drug involves oxybutynin chloride 5 extended release film coated tablet? **LLAMA-2-13B** Which routed drug involves oxybutynin chloride 5 extended release film coated tablet? **LLAMA-2-70B** Which routed drug has oxybutynin chloride 5 extended release film coated tablet? **GLM-130B** Which routed drug has oxybutynin chloride 5 extended release film coated tablet? **TEXT-DAVINCI-001** which routed drug has oxybutynin chloride 5 extended release film coated tablet? **TEXT-DAVINCI-003** Which routed drug has oxybutynin chloride 5 extended release film coated tablet?

Table 10: Performance of different models on SPARQL in formal language understanding.

NATURAL LANGUAGE QUESTION: Number of points over a season of player kobe bryant from season 2004?
LOGICAL FORMS: (call SW.listValue (call SW.getProperty (call SW.filter (call SW.getProperty en.player.kobe_bryant (call SW.reverse (string player))) (string season) (string =) (date 2004-1-1)) (string num_points)))
MODEL OUTPUT **GPT2-LARGE** Name a [String] (call date] [String] [Year[]] is used in domain] [String name_domain] [SE] is used in 1.1 (not correct) **GPT2-XL** Name a point for this season. (not correct) **GPT-J** Name a player with less than 3 points per season. (not correct) **FLAN-T5-L** (call SW.listValue (call SW.getProperty (call SW.filter (call SW.getProperty en.player.kobe_bryant (call SW.reverse (string player))) (string season) (string =) (date 2004-1-1)) (string num_points)) (copys the input) **FLAN-T5-XL** (string season) (string =) (date 2004 -1) (string num_points)) (not correct) **FLAN-T5-XXL** ds inxity how much is upgrade new x in medium how far.MD what! (not making sense) **LLAMA-2-7B** Number of points of kobe bryant (not correct) **LLAMA-2-13B** Points of kobe bryant(not correct) **LLAMA-2-70B** Points of 2004 season of kobe bryant **GLM-130B** Did this year 2019 Kobe Bryant have a season with less points than a particular number of points? (not correct) **TEXT-DAVINCI-001** What was Kobe Bryant's score on January 1, 2004?? **TEXT-DAVINCI-003** What were Kobe Bryant's points scored in the 2004 season?

Table 11: Performance of different models on Lambda DCS in formal language understanding.

NATURAL LANGUAGE QUESTION: How many Pennsylvania counties have a population greater than 7800 or a population less than 40000000?
LOGICAL FORM: FindAll()FilterNum(population, 7800, >)FilterConcept(county of Pennsylvania)FindAll()FilterNum (population, 40000000, <)FilterConcept(county of Pennsylvania)Or()Count()
RETRIEVED EXAMPLES: **1.** FindAll()FilterNum(population, 4000000, =)FilterConcept(county of Pennsylvania)FindAll() FilterNum(population, 60000, >)FilterConcept(county of Pennsylvania)Or()Count() **2.** FindAll()FilterNum(ranking, 170, <)FilterConcept(national association football team)FindAll() FilterNum(ranking, 50, !=)FilterConcept(national association football team)Or()Count() **3.** FindAll()FilterNum(population, 34000, !=)FilterConcept(county of North Carolina)FindAll() FilterNum(population, 5600, <)FilterConcept(county of North Carolina)Or()Count()
OUTPUT: How many counties of Pennsylvania have a population above 7800 or less than 40000000?

Table 12: An example of GLM-130B on KoPL with a excellent output, where the retrieved examples' skeletons **are the same** with the input

NATURAL LANGUAGE QUESTION: Who was the prize winner when Mrs. Miniver got the Academy Award for Best Writing, Adapted Screenplay?
LOGICAL FORM: Find(Mrs. Miniver)Find(Academy Award for Best Writing, Adapted Screenplay)QueryRelationQualifier (award received, statement is subject of)
RETRIEVED EXAMPLES: **1.** Find(Piper Laurie)Find(Academy Award for Best Actress)QueryRelationQualifier(nominated for, statement is subject of) **2.** Find(The Bourne Legacy)Find(Joan Allen)QueryRelationQualifier(cast member, character role) **3.** Find(Screen Actors Guild Award for Outstanding Performance by an Ensemble in a Comedy Series) Find(Josh Sussman)QueryRelationQualifier(winner, point in time)
OUTPUT: For what did David Lean receive an Academy Award for Best Writing, Adapted Screenplay?

Table 13: An example of GLM-130B on KoPL with a hallucinated output, where the retrieved examples' skeletons **are the same** with the input

NATURAL LANGUAGE QUESTION: How many American football team season?
LOGICAL FORM: FindAll()FilterConcept(American football team season)Count()
RETRIEVED EXAMPLES: **1.** FindAll()FilterNum(population, 8.7, >)FilterConcept(state)Count() **2.** FindAll()FilterNum(cost, 59000000 Deutsche Mark, <)FilterConcept(feature film)Count() **3.** FindAll()FilterDate(point in time, 2003-06-26, =)FilterConcept(National Basketball Association Draft)Count()
OUTPUT: For what did David Lean receive an Academy Award for Best Writing, Adapted Screenplay?

Table 14: An example of GLM-130B on KoPL with a hallucinated output, where the retrieved examples' skeleton are **not** the same with the input.

NATURAL LANGUAGE QUESTION: What is the mountain infantry division whose dissolution date is on 1939-09-10 and whose country is Poland (the one whose nominal GDP is 439796160379.475 United States dollar)
LOGICAL FORM: FindAll()FilterDate(dissolved, abolished or demolished, 1939-09-10, =)FilterConcept(mountain infantry division)Find(Poland)FilterNum(nominal GDP, 439796160379.475 United States dollar, =)Relate(country, backward)FilterConcept(mountain infantry division)And()What()
RETRIEVED EXAMPLES: **1.** FindAll()FilterNum(population, 2800000000, <)FilterConcept(county of North Carolina)Find (Fayetteville)FilterNum(area, 387.4256 square kilometre, =)Relate(capital, backward)FilterConcept (county of North Carolina)And()What() **2.** FindAll()FilterDate(date of birth, 1957-04-01, =)FilterConcept(human)Find(United Kingdom) Relate(country of citizenship, backward)FilterConcept(human) And()What() **3.** FindAll()FilterDate(date of birth, 1956-04-19, =)FilterConcept(human) Find(actor)Relate(occupation, backward) FilterConcept(human)And()What()
OUTPUT: Which mountain infantry division was dissolved in 1939-09-10, whose nominal GDP on that day was 439,796,160,000.0 United States dollars? (missing the country Poland)

Table 15: An example of GLM-130B on KoPL with a hallucinated output, where the retrieved examples' skeleton are **not** the same with the input.

EXAMPLE 1 **INPUT LOGICAL FORM** Find(The Haunting).FilterDate(publication date, 1999-10-14, =).Find(Man on the Moon). SelectBetween(duration, greater) **OUTPUT LOGICAL FORM** Find(The Haunting).FilterDate(publication date, 1999-10-14).Find(Man on the Moon).FilterDate (publication date, 1991-07-12).Compare(duration).VerifyNum(longer)
EXAMPLE 2 **INPUT LOGICAL FORM** Find(Portsmouth).FilterStr(official website, 'http://www.portsmouthva.gov/') .Relate'(located in the administrative territorial entity, forward).FilterConcept(state of the United States).Relate(contains administrative territorial entity, forward).FilterConcept(independent city).Count() **OUTPUT LOGICAL FORM** Find(Portsmouth).Relate(located in the administrative territorial entity, backward).FilterStr(official website, http://www.portsmouthva.gov/).Find(administrative territorial entity).Relate(administrative division of, forward).FilterConcept(state of the United States).FilterConcept(independent city).Count()"

Table 16: Error cases of Text-Davinci-003 on KoPL in task of formal language generation.

EXAMPLE 1 **INPUT LOGICAL FORM** SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :business.business_operation . VALUES ?x1 { :m.05lfsg } ?x0 :business.business_operation.industry ?x1 . FILTER (?x0 != ?x1)}} **OUTPUT LOGICAL FORM** SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :business.business_operation . VALUES ?x1 { :m.05lfsg } ?x0 :business.business_operation.business ?x1 . FILTER (?x0 != ?x1) } }
EXAMPLE 2 **INPUT LOGICAL FORM** SELECT (?x0 AS ?value) WHERE {SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :conferences.conference_sponsor . VALUES ?x1 { :m.0j2fyjs } ?x0 :conferences.conference_sponsor.conferences ?x1 . FILTER (?x0 != ?x1)}} **OUTPUT LOGICAL FORM** SELECT (?x0 AS ?value) WHERE { SELECT DISTINCT ?x0 WHERE { ?x0 :type.object.type :media_common.sponsor . VALUES ?x1 { :m.0j2fyjs } ?x0 :media_common.sponsor.sponsored_by ?x1 . FILTER (?x0 != ?x1) } }

Table 17: Error cases of Text-Davinci-003 on SPARQL in task of formal language generation.

EXAMPLE 1
INPUT LOGICAL FORM (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_points)) (string <=) (call SW.ensureNumericEntity (call SW.getProperty (call SW.getProperty en.player.kobe_bryant (call SW.reverse (string player))) (string num_points))))) (call SW.domain (string player))) (string player)))
OUTPUT LOGICAL FORM (call SW.listValue (call SW.filter (call SW.getProperty (call SW.domain (string player)) (string ! type)) (call SW.ensureNumericProperty (string num_points)) (string <=) (call SW.ensureNumericEntity (call SW.getProperty en.player.kobe_bryant (string num_points))))) (Not correct)

EXAMPLE 2
INPUT LOGICAL FORM (call SW.listValue (call SW.getProperty ((lambda s (call SW.filter (var s) (call SW.ensureNumericProperty (string num_games_played)) (string <) (call SW.ensureNumericEntity (call SW.getProperty (call SW.getProperty en.player.kobe_bryant (call SW.reverse (string player))) (string num_games_played))))) (call SW.domain (string player))) (string player)))
OUTPUT LOGICAL FORM (call SW.listValue (call SW.filter (call SW.domain (string player)) (string game) (string <) (call SW.getProperty (call SW.singleton en.player.kobe_bryant) (string game)))) (Not correct)

Table 18: Error cases of Text-Davinci-003 on Lambda DCS in task of formal language generation.

Model	Overall	Multi-hop	Qualifier	Compare	Logical	Count	Verify	Zero-shot
GLM-130B	86.2	84.2	79.8	93.1	84.1	79.2	90.1	84.9
Text-Davinci-001	85.6	83.7	78.1	93.2	83.8	78.6	89.5	84.1
Text-Davinci-003	88.1	86.8	80.6	94.6	85.8	82.8	92.5	87.1
Human	90.5	89.5	84.8	95.5	89.3	86.7	93.3	89.6

Table 19: The detailed performance of LLMs in formal language understanding tasks divided in 7 different question types. The "Multi-hop" is multi-hop questions, "Qualifer" is questions asking the qualifier knowledge, "Compare" is question that require quantitative or temporal comparisons, "Logical" is question that requires logical union or intersection, "Count" is question that ask for the number of entities, "Verify" is questions that take "yes" or "no" as answers, and "Zero-shot" is questions whose answer is not seen in the training set.

Demostrantions	Lambda DCS						SPARQL						KoPL					
	w/o e.l.			w/ e.l.			w/o e.l.			w/ e.l.			w/o e.l.			w/ e.l.		
	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3	run 1	run 2	run 3
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	1.3	1.3	1.3	1.6	15.0	12.5	12.5	20.0	24.7	17.5	24.3	21.7	22.3	28.3	26.0	20.7
10	1.6	4.2	6.1	2.4	3.8	7.5	16.8	14.3	15.0	21.5	19.2	22.4	28.3	24.0	24.7	27.0	28.7	29.3
15	3.7	2.2	7.8	1.3	7.5	8.8	20.0	19.2	12.5	22.5	19.7	25.0	29.0	27.0	26.3	30.3	31.3	28.0
20	2.5	6.3	10.8	3.8	5.5	10.8	15.0	21.7	15.0	—	—	—	31.3	26.0	37.7	34.7	31.7	33.3
25	7.2	6.5	10.1	3.8	13.8	12.5	20.0	19.2	17.5	—	—	—	34.3	35.7	37.3	35.7	33.3	39.0
30	6.3	8.8	11.3	—	—	—	—	—	—	—	—	—	37.3	35.7	39.0	39.3	35.7	41.3
35	—	—	—	—	—	—	—	—	—	—	—	—	41.0	39.3	41.3	41.0	35.3	48.7

Table 20: Detailed results of evaluation of performance of LLMs' generation on Text-Davinci-003 investigating the influence of varying demonstration number and whether entity linking strategy.