TARGA: Targeted Synthetic Data Generation for Practical Reasoning over Structured Data

Anonymous ACL submission

Abstract

Semantic parsing, which converts natural language questions into logic forms, plays a crucial role in reasoning within structured environments. However, existing methods encounter two significant challenges: reliance on extensive manually annotated datasets and limited generalization capability to unseen examples. To tackle these issues, we propose Targeted Synthetic Data Generation (TARGA), a practical framework that dynamically generates high-relevance synthetic data without manual annotation. Starting from the pertinent entity and relation of a given question, we probe for the potential relevant queries through layer-wise expansion and cross-layer combination. Then we generate corresponding natural language questions for these constructed queries to jointly serve as the synthetic demonstration for in-context learning. Experiments on multiple knowledge base question answering (KBQA) datasets demonstrate that TARGA, using only a 7B-parameter model, substantially outperforms existing non-fine-tuned methods that utilize close-sourced model, achieving notable improvements in F1 scores on GrailQA (+7.7) and KBQA-Agent (+12.2). Furthermore, TARGA also exhibits superior sample efficiency, robustness, and generalization capabilities under non-I.I.D. settings.

1 Introduction

Reasoning over structured environments, such as Knowledge Base (KB), Database, and Web, has emerged as a crucial ability for large language models (LLMs) (Liu et al., 2024; Gu et al., 2023). Among the various methods for structural reasoning, semantic parsing stands out as a mainstream method and has garnered increasing attention from researchers. By translating natural language questions (NLQ) into logic forms, semantic parsing enables seamless interaction with structured environments, thereby enhancing user experience and accessibility.

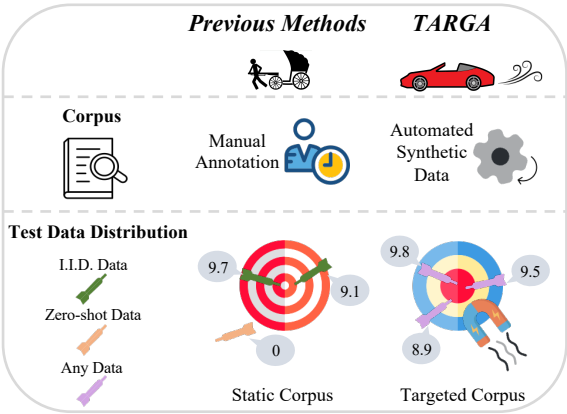

Figure 1: TARGA compared with previous methods.

However, current semantic paring methods typically face two significant challenges:

1) Dependence on annotation. Previous methods usually rely on extensive amounts of manually annotated data. By training (Ye et al., 2022; Shu et al., 2022) or retrieval (Li et al., 2023b; Nie et al., 2023) based on large-scale annotations, existing works have made remarkable progress. Unfortunately, collecting manual annotations in specific environments is labor-intensive and time-consuming. In real-world scenarios, large amounts of pre-collected annotated data are often unavailable, limiting the scalability of these methods.

2) Limited generalization capability. Even with access to large annotated datasets, previous methods still struggle with generalizing to unseen examples. Regardless of the paradigm (*e.g.,* in-context learning or fine-tuning), any method relying on a static, offline-collected dataset is inevitably influenced by the dataset's distribution. Specifically, these methods tend to perform well on examples encountered in the dataset (the I.I.D. setting) but exhibit weaker generalization when faced with novel environmental items or unfamiliar query structures (the non-I.I.D. settings), as shown in Figure 1. In complex environments, such as Freebase with over

three billion triples, it is nearly impossible for a pre-collected static dataset to cover the full scope of the environment. Additionally, as the coverage of annotations increases, so do the costs associated with training and retrieval, which further limits the scalability and generalization.

To address the aforementioned challenges, in this work, we propose a practical semantic parsing framework called Targeted Synthetic Data Generation (**TARGA**), which does not need any manually annotated data and can efficiently work on a 7B model. Specifically, TARGA addresses these challenges by dynamically synthesizing highly relevant examples of a test question as demonstrations for in-context learning. Starting from the KB items (entity, relation) that may related to the given question, we construct logic forms through layer-wise expansion (extend a new edge for a sub-structure) and cross-layer combination (combine different sub-structures), gradually evolving from simple to complex structures. To further enhance relevance, we re-rank the synthetic logic forms to select the most pertinent ones and generate their corresponding natural language questions, which are then used as demonstrations for reasoning. Through this automatic data synthesis, we free annotators from the heavy burden of labeling tasks. Additionally, the demonstrations are generated based on the given question, thus naturally avoiding the challenge of generalization.

Without any data annotation, TARGA significantly outperforms all non-fine-tuned approaches across multiple complex KBQA datasets, particularly excelling in non-I.I.D. settings. Remarkably, TARGA achieves this with only a 7B-parameter model, whereas most baselines rely on advanced closed-source models, such as GPT-3.5-turbo, enabling faster and more cost-efficient inference. On the GrailQA dataset, we improve the performance of non-fine-tuned methods from an F1 score of 61.3 to 69.0. On KBQA-Agent, the most challenging dataset, we elevate the SOTA performance from 34.3 to 46.5 F1 scores. Further analyses highlight the high quality of the data generated by TARGA. Even with a single demonstration, TARGA still surpasses all non-fine-tuned methods on GrailQA. Additionally, TARGA exhibits remarkable robustness in adversarial settings[1].

[1]The codes and data will be released after acceptance.

2 Related Works

2.1 Knowledge Base Question Answering with LLMs

With the advancement of large language models, recent works have adopted LLMs as the backend for KBQA. In particular, In-Context Learning (Brown et al., 2020) requires dozens of demonstrations to guide the model's responses. To achieve competitive performance, existing ICL-based KBQA works (Li et al., 2023b; Nie et al., 2023) typically retrieve the most similar examples from a manually annotated training set as demonstrations. However, this strategy often results in performance degradation on non-I.I.D. questions involving unseen structures or KB items. For example, Nie et al. (2023); Li et al. (2023b) reported that the performance in zero-shot settings can be up to 20% lower compared to I.I.D. settings.

Another line of KBQA methods, agent-based methods (Liu et al., 2024; Huang et al., 2024; Gu et al., 2024), decomposes questions into individual steps to solve. While step-by-step solving aligns with human intuition and demonstrates remarkable generalization ability, it incurs high computational costs and presents challenges in constructing trajectories. Moreover, the effectiveness of the agent-based paradigm relies heavily on the planning and generalization abilities of advanced LLMs, leading to subpar performance when using weaker models, such as some open-source variants. Such dependency underscores the limitation of agent-based approaches when superior LLMs are unavailable or impractical to use due to resource constraints.

2.2 Synthetic Data Generation

Instead of relying solely on human annotation for training data, recent works have leveraged LLMs to generate synthetic data, thereby reducing the burden on human annotators. For instance, Chiang et al. (2023); Taori et al. (2023) use instructions to generate training data as a supplement to manual annotation via self-instruct techniques (Wang et al., 2023). However, this approach still requires human-annotated seed examples to ensure high-quality demonstrations, which entails significant demand for LLM usage. Rather than directly prompting LLMs to generate training data, Cao et al. (2022); Huang et al. (2023) address this problem by first sampling structured queries from the environment and then converting these queries into natural language using LLMs. Nevertheless, obtaining mean-

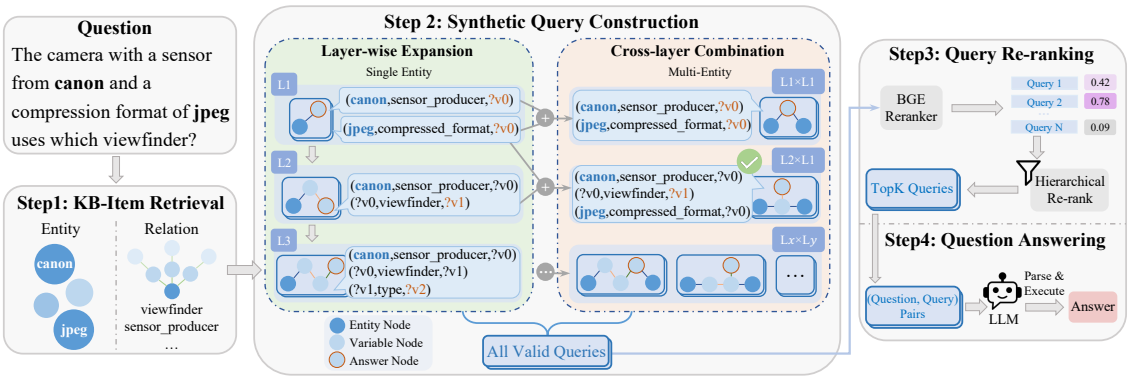

Figure 2: Overview of TARGA.

ingful structured queries remains a non-trivial task.

Other works similar to ours include FlexKBQA (Li et al., 2023c) and BYOKG (Agarwal et al., 2024). FlexKBQA relies on predefined templates and model training stages to automatically annotate data, while BYOKG synthesizes data from scratch. However, both approaches require a time-consuming offline data collection phase. More importantly, like other methods, they rely on reasoning over a pre-collected static dataset, which still suffers from generalization issues. In TARGA, we systematically design a framework to dynamically synthesize relevant examples in an online manner. It avoids the need for a lengthy data collection process, enabling us to dynamically obtain the most relevant examples for each test case without being constrained by a static dataset.

3 Our Approach

3.1 Overview

As shown in Figure 2, our targeted data synthesis framework, TARGA, consists of four parts. Given a natural language question, we first collect candidate KB items such as entity and relation as initialization. Then, we explore valid query graphs, expanding these items from simple to complex structures to construct synthetic queries. Next, we filter high-relevance candidate queries by ranking and generating their corresponding natural language questions. Finally, we use these high-quality synthetic data as demonstrations for QA reasoning.

3.2 Candidate KB Items Retrieval

For candidate entities, we adopt the linking result provided by Gu et al. (2023). For candidate relations, we compute the similarity of question and freebase relations based on OpenAI embedding[2] and retain the top 20 most similar candidates. Different from previous fine-tuned methods (Ye et al., 2022; Hu et al., 2022), which typically require training a relation linking model for higher precision and recall, our method does not rely on the precision of these items. This is because, although all retrieved KB items are relevant to the question, they do not necessarily form valid combinations within a specific graph structure. In this way, the subsequent query construction steps in Section 3.3 can be viewed as a joint entity-relation disambiguation process, thus significantly reducing the number of invalid queries.

3.3 Synthetic Query Construction

This stage aims to construct question-targeted queries to facilitate subsequent QA reasoning. Based on the retrieved KB items, we explore the possible query structures that are valid (*i.e.,* yield non-empty execution results). However, enumerating all possible structures may lead to an unmanageable combinatorial explosion. To mitigate this, our exploration of candidate queries follows a simple-to-complex manner, where we only further explore sub-structures that have been verified as valid. Starting from the simplest structure, we progressively search for more complex query structures in two ways: *Layer-wise Expansion* and *Cross-layer Combination*. For Layer-wise Expansion, we extend one edge on the previous structures for each layer. For Cross-Layer Combination, we combine previously validated substructures into a new structure. These two strategies are jointly performed to gradually extend the obtained query graphs until a desired complexity is achieved.

[2] https://platform.openai.com/docs/guides/embeddings

Layer-wise Expansion. As shown in the left part of Step 2 in Figure 2, we first probe for all simplest and valid query structures (a single relation linked to an entity) to form the Layer 1 (L1). The blue dialog boxes represent an example of queries at this level of complexity. In the subsequent steps, each time, we extend one edge from the previous valid structure in Layer N to form Layer N+1. This expansion process is reflected in the dialog box, where a new triple is added to the previous query structure each time. We stop layer-wise expansion once an entity reaches layer 3, as a coherent and well-formed question typically does not exceed three hops. In other words, the distance between a specific entity in the question and the final answer is generally no more than three edges. We also employ several pruning strategies to improve efficiency, such as restricting query answers to non-blank nodes and discarding newly generated queries with redundant triples, etc.

Cross-layer Combination. Through Layer-wise Expansion, we collect structures of varying complexity expanding from each entity. However, in real-world scenarios, multiple entities often appear in the same question. We handle these cases by introducing Cross-layer Combination, as illustrated in the right part of Step 2 in Figure 2. The combination of two structures is achieved by enumerating a potential connection node and merging two queries into one. Queries across different layers can be freely combined if they form a valid query. In this work, we stop the combination after a query reaches four edges which is enough for model most of questions in current datasets.

In this manner, we circumvented a significant number of invalid queries, thus obtaining most of the potentially relevant queries with relatively lower query overhead. As shown in Table 1. the average number of valid candidate queries per question is only in the range of several dozen, which is already within the contextual length limits manageable by an ICL model. Furthermore, the scale remains within a feasible range if further fine-grained ranking is required.

3.4 Synthetic Query Re-ranking

To obtain the most relevant examples for the subsequent QA task, we re-rank all valid queries using the *bge-reranker-v2-m3* model (Chen et al., 2024) based on their similarity to the question and retain the top 10 candidates as demonstrations. We also transform the synthesized query into a form that is closer to natural language to bridge the gap between the text embedding model and the query. This query testification step further enhanced the quality of the ranking.

Table 1: Statistics of query construction. Size represents the average number of valid queries per question. Cvg. refers to coverage, indicating the proportion of questions with at least one correct (F1 = 1) in synthetic queries. TPQ and QPQ denote the running time (in seconds) and the number of query attempts per question, respectively.

Datasets	Size	Cvg.	TPQ	QPQ
GrailQA	17.4	0.79	2.99	256.8
GraphQ	89.3	0.69	11.41	1,094.6
KBQA-Agent	173.1	0.78	20.07	2,270.3
MetaQA-3Hop	25.9	1.00	0.05	56.1
WikiSQL	18.1	0.75	0.16	88.6

Due to the nature of graph structure expansion, the number of complex queries grows exponentially compared to simpler ones, leading to the most complex queries dominating the ranking process. To address this imbalance and keep the size of the candidate query pool manageable, we implement a **Hierarchical Ranking** strategy based on the complexity of the query. For all queries derived from the same parent structure, we retain only the top four candidates. This ensures that even when the candidate query pool expands drastically, the growth of the candidate space remains manageable within polynomial complexity.

3.5 Question Answering

To help the LLM understand the semantics of the provided logic form (query), we generate corresponding natural language questions for selected queries, obtaining the (Question, Query) pairs. Then, we adopt the In-context Learning paradigm to the target query generator. Specifically, we utilize the question to be solved with the corresponding synthetic queries as the prompt. Finally, we parse and execute the output query from the LLM to obtain the answer.

4 Evaluation

4.1 Setup

We experiment with four complex KBQA datasets, *i.e.,* GrailQA (Gu et al., 2021), GraphQ (Su et al., 2016), KBQA-Agent (Gu et al., 2024), MetaQA (Zhang et al., 2018) and a Text2SQL dataset, *i.e.,* WikiSQL (Zhong et al., 2017). We use the F1

Table 2: Main results of KBQA performance, categorized by the amount of required annotated data example. Fine-tuning/ICL/Agent indicate different reasoning paradigms. **Bold** values highlight the best among non-fine-tuned models. * indicates fine-tuned models, among which underlined values represent the best performance.

Methods	Models	GrailQA	GraphQ	KBQA-Agent	MetaQA
	full *training set (Fine-tuning / ICL)*				
RnG-KBQA* (Ye et al., 2022)	T5-base	67.4	-	-	-
ArcaneQA* (Gu and Su, 2022)	T5-base	73.7	31.8	-	-
Pangu* (Gu et al., 2023)	T5-3B	83.4	57.7	-	-
KB-Binder-R (Li et al., 2023b)	GPT-3.5-turbo	58.5	32.5	-	99.5
KB-Coder-R (Nie et al., 2023)	GPT-3.5-turbo	61.3	36.6	-	-
	dozens *annotation (ICL)*				
Pangu (ICL) (Li et al., 2023b)	Codex	53.5	35.4	18.1	-
KB-Binder (Li et al., 2023b)	GPT-3.5-turbo	50.8	34.5	4.2	96.4
KB-Coder (Nie et al., 2023)	GPT-3.5-turbo	51.7	35.8	-	-
	one *annotation (Agent)*				
AgentBench (Liu et al., 2024)	GPT-3.5-turbo	30.5	25.1	25.9	-
FUXI (Gu et al., 2024)	GPT-3.5-turbo	-	-	34.3	-
QueryAgent (Huang et al., 2024)	GPT-3.5-turbo	60.5	50.8	-	98.5
	zero *annotation (ICL)*				
BYOKG (Agarwal et al., 2024)	MPT-7B	46.5	-	-	56.5
TARGA (Ours)	QWen-2.5-7B-Instruct	69.0	50.6	46.5	85.7
	QWen-2.5-72B-Instruct	**70.6**	**54.1**	**57.3**	99.8
	GPT-3.5-turbo	68.9	51.0	52.7	96.5
	GPT-4-turbo	69.8	52.5	51.4	**99.9**

scores as the evaluation metric for KBQA and denotation accuracy for Text2SQL. We compare TARGA with various paradigms of baselines, including fine-tuning, ICL, and Agent, where we report performance in the original paper. For experiments with other settings, we copy the re-implemented result from Gu et al. (2024). By default, we use `Qwen-2.5-7B-Instruct` as the base LLM in our experiments with 10 demonstrations for all datasets. Detailed introduction of datasets and baselines are available in Appendices B and C.

4.2 Main Result

Table 2 illustrates the main result for KBQA. we compare TARGA with methods that require different amounts of annotation. For the relatively challenging datasets, *i.e.,* GrailQA, GraphQ, and KBQA-Agent, based on a 7B model, TARGA achieves the best performance among all non-fine-tuned methods which are based on advanced close-sourced LLMs. On GrailQA and KBQA-Agent, TARGA surpasses previous SOTA non-fine-tuned methods by 8.7 and 13.0 F1. On GraphQ, TARGA even beats some fine-tuned methods and achieves similar performance with the best non-fine-tuned method.

When compared to methods with a similar paradigm (ICL-based), TARGA outperform previous methods by 14.0 and 28.4 in F1 on GraphQ and KBQA-Agent, respectively. It is worth noting that our method requires neither any manually annotated corpus nor the expensive close-sourced model. Besides, we have not incorporated self-consistency to boost the performance. This can be attributed to the high quality of the synthetic data, which has led to a reduction in task difficulty and a decreased reliance on the capabilities of strong LLMs. Moreover, compared with other ICL-based methods which include 40-100 demonstrations, TARGA uses only 10 demonstrations but still achieves the best performance, demonstrating notable data efficiency.

Compared with BYOKG which also requires no annotated data, TARGA achieves approximately 1.5× performance on GrailQA and MetaQA-3Hop. More importantly, TARGA dynamically synthesizes the most relevant data for different questions, allowing for seamless adaptation to questions from any distribution. Additionally, the synthetic data generated by TARGA is constructed online, eliminating the need for a time-consuming offline data collection phase.

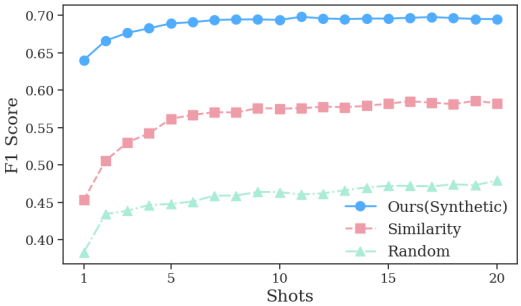

Figure 3: Performance with various numbers of demonstrations on GrailQA (1,000 random questions).

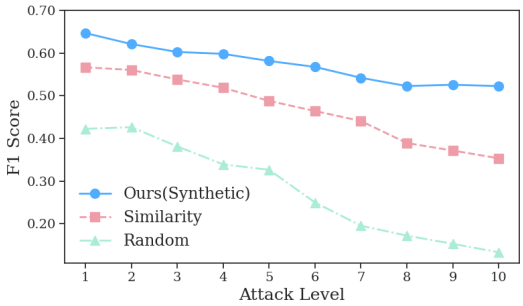

Figure 4: Performance under attack setting on 1,000 random sampled GrailQA questions. Attack level indicates how many demonstrations have been corrupted.

4.3 Detailed Analyses

To gain more insights, we conduct more detailed experiments to illustrate some favorable practical characteristics of TARGA.

4.3.1 Sample Efficiency

In this section, we analyze how the number of demonstrations impacts the performance. We experiment on GrailQA, with the number of demonstrations ranging from 1 to 20. Based on our QA framework, we compare three distinct sampling settings: **Random**, **Similarity**-based, and **Ours (synthetic)**, corresponding to examples randomly sampled from the training set, retrieved by similarity from the training set, and retrieved by similarity from the synthetic data by TARGA, respectively. The random and similarity settings can be viewed as reflections of the previous ICL-based and the retrieval-augmented ICL-based methods. Results are illustrated in Figure 3. Even with only one demonstration, our synthetic setting significantly outperforms the random and retrieval settings with 20 shots, suggesting the high quality of our synthetic data. Moreover, the growth curve in the synthetic setting (blue line) is relatively flat as the number of demonstrations increases. After reaching 7 shots, the synthetic setting exhibits almost no further improvement, while the other two settings continue to show growth even after reaching 20 shots, highlighting the data efficiency of our methods.

4.3.2 Robustness Analyses

To further validate the robustness of our approach in real-world scenarios, we conduct an adversarial experiment designed to simulate conditions of poor synthetic data quality. Specifically, the attack involves randomly replacing one relation in a candidate query. We compare the same three settings as in Section 4.3.1. As in Figure 4, our method exhibits significantly stronger robustness under adversarial conditions. Even when all demonstrations were compromised, the performance degradation of TARGA was only around 25%. In contrast, the other setups experience a sharp decline: the F1 scores of similarity-based setup drop by about 40%, and the random setting even falls by approximately 75%. This further demonstrates the superior robustness of our method compared to other approaches.

4.3.3 Performance on Different Generalization Levels

We experiment on GrailQA to compare the performance on different generalization levels (preliminary in Appendix D). To make a fair comparison with the "-R" setting of previous methods, we also implemented a "-R" version of TARGA where the entire training set was incorporated into our demonstration sampling pool. Specifically, we retain the top 5 most similar examples from the training set and the top 5 most similar synthetic data instances jointly as demonstrations.

As shown in Table 3, under the I.I.D setting, TARGA benefits from the inclusion of high-quality annotations, achieving noticeable improvements and performing comparably to previous methods. However, for the more challenging Compositional and Zero-shot settings, where analogous questions are absent from the pre-collected training set, the performance of previous methods in the "-R" setup dramatically decreases by approximately 30 in F1. In contrast, TARGA shows no significant decline, demonstrating its strong generalization ability in scenarios that more closely resemble real-world situations where relevant corpora are unavailable. Notably, under the zero-shot setting, TARGA derives minimal improvements from the training set,

Table 3: Results of different generalization levels on GrailQA. "-R" indicates a version accessing the whole training set for similarity retrieval. Comp. and Zero. indicates the Compositional and Zero-shot setting on GrailQA, respectively. Avg. denotes the average F1.

Methods	I.I.D.	Comp.	Zero.	Avg.
KB-BINDER	48.3	48.8	41.8	50.8
KB-CODER	49.3	49.6	43.2	51.7
TARGA	**68.4**	**62.2**	**71.7**	**69.0**
KB-BINDER-R	80.6	53.6	50.7	58.5
KB-CODER-R	**81.0**	57.8	54.1	61.3
TARGA-R	80.8	**63.6**	**71.6**	**71.9**

Table 4: Efficiency analysis with ICL-based (KB-Binder) and Agent-based methods (AgentBench and QueryAgent). TPQ, QPQ, and CPQ denote the time cost (seconds), number of SPARQL query times, and `gpt-3.5-turbo` invocation cost ($) per question.

Methods	GrailQA			GraphQ		
	TPQ	QPQ	CPQ	TPQ	QPQ	CPQ
KB-BINDER	51.2	3,297.7	0.010	84.0	2,113.8	0.024
AgentBench	40.0	7.4	0.034	65.1	7.2	0.035
QueryAgent	16.6	**5.2**	0.019	15.3	**6.2**	0.021
TARGA	**4.5**	256.8	**0.000**	**13.0**	1,094.6	**0.000**

suggesting that pre-collecting a substantial corpus of examples is ultimately an incomplete solution and tends to largely fail when confronted with real, unseen problems.

We further analyze the I.I.D. questions. We observe that half of the test questions can match a question with the same query in the training set, and 93% have the same canonical query (only the entity is distinct). On average, an I.I.D. question has 11.74 questions with the same canonical query from the training set. However, the non-I.I.D. question can not match a question in the training set that shares the same query. This explains why the "-R" setting improves the I.I.D. setting by a large margin but fails to boost the performance on non-I.I.D questions. This also constitutes strong evidence that targeted synthetic data generation is a promising solution for addressing generalization challenges.

4.3.4 Efficiency analysis

For a practical QA system, high efficiency is also a key characteristic. Following Huang et al. (2024), we analyzed three efficiency metrics: TPQ, QPQ, and CPQ, as shown in Table 4. Regarding TPQ, our method significantly outperforms previous methods with only 4.5 seconds response time on GrailQA. Regarding QPQ, Agent-based methods have an inherent advantage. However, comparatively speaking, the overhead of QPQ remains relatively inexpensive than the other two metrics. Compared to KB-Binder, which also employs the ICL paradigm, our approach demonstrates a marked superiority on QPQ. This is primarily because our synthetic demonstrations are highly aligned with the target question, enabling the generated logic forms to be executable without any post-processing in most cases. Conversely, since previous methods can not always retrieve the relevant candidate query in the training set, the generated logic forms are often not executable. Consequently, the logic form necessitates stepwise binding to valid KB items, which leads to a large demand for queries.

In terms of CPQ, agent-based methods inherently face challenges due to the lengthy trajectory of demonstrations and the need for multiple calls of LLM. Since our method does not rely on close-sourced LLM, the CPQ is zero. If compared with the consumed tokens, TARGA uses significantly fewer tokens because it requires fewer examples as demonstrations and does not use self-consistency. As a result, the token cost is only about one-tenth that of other ICL-based methods.

4.3.5 Performance on different sizes of LLM

In real-world applications, large and powerful LLMs are not always accessible or affordable. Therefore, we further analyze the performance of various methods across different model sizes. We compare the Agent method (QueryAgent), ICL method (KB-Binder), and the retrieval-augmented ICL method (KB-BINDER-R). As shown in Table 5, our approach demonstrates remarkable adaptability from *qwen-1.5B-instruct* to *gpt-4o-mini*. With just the 1.5B model, our method already surpasses the previous best-performing method, while at 7B, it only slightly lags behind the closed-source model. The Agent method has strong generalization capabilities but is heavily reliant on the planning and self-correction abilities of the most advanced LLM, which smaller models do not excel at. For models below 72B, the performance of the QueryAgent is essentially unusable. For the 72B model, the performance of QueryAgent is still inferior to that of the ICL method using a model of the same size and ultimately failing to exceed closed-source model performance. Regarding the ICL methods, previ-

Table 5: The performance on GrailQA with different base model sizes. The 1.5B, 7B, 32B, and 72B represent the Qwen 2.5 instruct models family, while 4o-mini indicates GPT-4o-mini. We experiment on 500 random sampling questions. Δ indicates the max performance gap between different models.

	1.5B	7B	32B	72B	4o-mini	Δ
QueryAgent	10.3	16.1	50.8	58.5	62.3	52.0
KB-Binder	20.2	39.8	51.3	50.6	47.0	31.1
KB-Binder-R	27.6	55.0	59.4	63.8	58.0	36.3
TARGA	**61.3**	**65.3**	**67.4**	**67.5**	**67.7**	**9.4**

Table 6: Ablation study of each component on GrailQA.

	GrailQA	GraphQ	KBQA-Agent
TARGA	**69.0**	**50.6**	**46.5**
w/o Query Textification	64.9	46.6	39.6
w/o Re-Ranking	59.9	38.7	19.3
w/o Synthetic Question	67.3	49.5	45.5

ous works typically experiment on the strongest closed-source models without testing their performance on open-source models. We demonstrate here the performance of ICL on open-source models, revealing that the latest open-source models can reach or even surpass the capabilities of the GPT series models in certain tasks. This provides a feasible assurance for continuing research in semantic parsing based on closed-source models.

4.3.6 Ablation Study

Table 6 presents the impact of distinct components on model performance across three datasets. Compared to the full model, removing the query textification component leads to a noticeable drop, particularly on KBQA-Agent (-6.9), highlighting the importance of bridging the gap between the text embedding model and logic form. The removal of the re-ranking component results in the largest performance decrease, with reductions of 9.1, 11.9, and 27.2 on GrailQA, GraphQ, and KBQA-Agent, respectively, underscoring the importance of the re-ranking step. In contrast, excluding synthetic question generation yields more modest declines, suggesting it is less critical than the other components but still beneficial for KBQA-Agent. It is unexpected, but from another aspect, it indicates that even only using the synthetic query as the demonstration the performance is also competitive.

4.3.7 Transferability to Text2SQL

We adapted our framework to the Text2SQL task to demonstrate the generality of our approach in other semantic parsing tasks. Employing the WikiSQL dataset, we compare TARGA with both the fine-tuned and non-fine-tuned methods. Among them, StructGPT and Readi are 32-shot and 7-shot methods, respectively. AgentBench and QueryAgent both use 1 shot. As shown in Table 7, with merely 10 synthetic examples as demonstrations, our method surpasses prior methods with 32 manually annotated examples and also outperforms the best 1-shot method, all while incurring a lower cost and smaller model. Besides, TARGA can even surpass a fine-tuned method with 3B model.

Table 7: Results on WikiSQL. * indicates fine-tuned baselines.

Methods	Acc.
RESDSQL-3B + NatSQL* (Li et al., 2023a)	79.9
T5-3B+PICARD* (Scholak et al., 2021)	75.1
StructGPT (ChatGPT) (Jiang et al., 2023)	65.6
Readi (Cheng et al., 2024)	66.2
AgentBench (Liu et al., 2024)	57.6
QueryAgent (Huang et al., 2024)	72.5
TARGA	**75.5**

5 Conclusion

In this paper, we explore two critical challenges in the semantic parsing task: reliance on annotated data and poor generalization on non-I.I.D. cases. We proposed a novel method called TARGA, which automatically synthesizes examples that are most relevant to the test data and utilizes them as demonstrations for in-context learning. Remarkably, TARGA achieves the best performance among all non-fine-tuned methods across three complex KBQA datasets and one Text2SQL dataset, especially on GrailQA and KBQA-Agent (7.7 and 12.2 F1 points, respectively). While achieving impressive performance, TARGA also exhibits the following practical properties: 1) It does not require any annotated data. 2) It is effective even with a model size of just 7B parameters. 3) The synthetic data is generated online. 4) It exhibits superior generalization, robustness, and speed. This work highlights the potential of leveraging synthetic data in semantic parsing, and we hope that TARGA can serve as a valuable foundation for developing more practical systems in this field.

Limitations

We would like to discuss some limitations of our work. First, in this paper, we validate TARGA on two specific semantic parsing tasks: KBQA and Text2SQL. While these tasks demonstrate the potential of our approach, further exploration across a broader range of tasks that involve transforming natural language into logical forms could strengthen the generalizability of TARGA. Additionally, we have not yet investigated the feasibility of our synthetic data generation method in other paradigms, such as agent-based or fine-tuned models. We would like to adapt TARGA to these paradigms in future work.

References

Dhruv Agarwal, Rajarshi Das, Sopan Khosla, and Rashmi Gangadharaiah. 2024. Bring your own KG: Self-supervised program synthesis for zero-shot KGQA. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 896–919.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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*, pages 1877–1901.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. 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)*, pages 6101–6119.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Sitao Cheng, Ziyuan Zhuang, Yong Xu, Fangkai Yang, Chaoyun Zhang, Xiaoting Qin, Xiang Huang, Ling Chen, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. 2024. Call me when necessary: Llms can efficiently and faithfully reason over structured environments. *Preprint*, arXiv:2403.08593.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Williams Gregory. 2013. Sparql 1.1 service description. w3c recommendation. *World Wide Web Consortium*.

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)*, pages 4928–4949.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: Three levels of generalization for question answering on knowledge bases. *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, 2021:3477–3488.

Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. 2024. Middleware for llms: Tools are instrumental for language agents in complex environments. *Preprint*, arXiv:2402.14672.

Yu Gu and Yu Su. 2022. ArcaneQA: Dynamic program induction and contextualized encoding for knowledge base question answering. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1718–1731.

Xixin Hu, Xuan Wu, Yiheng Shu, and Yuzhong Qu. 2022. Logical form generation via multi-task learning for complex question answering over knowledge bases. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1687–1696.

Xiang Huang, Sitao Cheng, Yuheng Bao, Shanshan Huang, and Yuzhong Qu. 2023. MarkQA: A large scale KBQA dataset with numerical reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10241–10259.

Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. 2024. Queryagent: A reliable and efficient reasoning framework with environmental feedback-based self-correction. *Preprint*, arXiv:2403.11886.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. 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*, pages 9237–9251.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *AAAI*.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023b. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2023c. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. *Preprint*, arXiv:2308.12060.

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. 2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2023. Code-style in-context learning for knowledge-based question answering. *Preprint*, arXiv:2309.04695.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8108–8121.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Empirical Methods in Natural Language Processing (EMNLP)*, Austin, Texas, USA.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, Caiming Xiong, and Computer Science. 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.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI conference on artificial intelligence*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Preliminary

A.1 Knowledge Base Question Answering

We introduce related concepts and the task of knowledge base question answering as follows. Let E be a set of entities, P be a set of relations, C be a set of classes, and I be a set of literals. A *knowledge base* $\mathcal{K} \subseteq E \times P \times (E \cup C \cup I)$ is a set of triples (s, p, o), where $s \in E$ is a subject entity, $p \in P$ is a predicate, and $o \in (E \cup C \cup I)$ is an object entity, class or literal value.

The task of knowledge base question answering can be formalized as learning a function f that takes a natural language question q as input, and outputs a structured query $q' = f(q)$ in a formal language such as SPARQL (Gregory, 2013). The structured query q' should ideally encode the entities, relations, and constraints specified by the input question q, such that executing q' over the knowledge base $\mathcal{K}$ yields the correct answer of q.

A.2 In-Context Learning

In-context learning (Brown et al., 2020) allows LLMs to perform new tasks by simply observing examples provided within the input, without updating their internal parameters. Intuitively, the model "learns" from the context and uses it to generate appropriate responses for similar tasks, relying on patterns it recognizes from the given examples.

More precisely, let $[x_1, x_2, \cdots, x_n]$ be a sequence of input tokens representing natural language texts, and let $[y_1, y_2, \cdots, y_n]$ be a corresponding sequence of output tokens representing the desired task, which in our context are structured queries. A LLM, denoted as f_θ, is a function parameterized by θ, which takes an input sequence and predicts the next token or sequence of tokens. In-context learning refers to the ability of a pretrained LLM to learn and adapt to a specific task purely by conditioning on a sequence of examples $S = [(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)]$, provided as part of the input context, without updating the model parameter θ. In contrast to traditional learning paradigms that require parameter updates via gradient descent, the LLM uses the provided examples to infer the underlying task and generate predictions for a new input x_{n+1}.

B Datasets

We experiment on four KBQA datasets and one Text2SQL dataset, their statics are shown in Table 8.

Table 8: Statistics of datasets.

Datasets	Train	Dev	Test
GRAILQA	44,337	6,763	13,231
GRAPHQ	2,381	-	2,395
KBQA-AGENT	-	-	500
METAQA-3HOP	114,196	14,274	14,274
WIKISQL	56,355	8,421	15,878

- **GrailQA** (Gu et al., 2021) is one of the most popular complex KBQA datasets. It divides the dataset into three levels of generalization, *i.e.,* I.I.D, Compositional, and Zero-shot.
- **GraphQ** (Su et al., 2016) is a challenging dataset that only consists of Compositional questions.
- **KBQA-Agent** (Gu et al., 2024) is a mixed dataset of the most difficult questions from four datasets (*i.e.,* GrailQA (Gu et al., 2021), ComplexWebQuesiton (Talmor and Berant, 2018), GraphQ (Su et al., 2016), and WebQSP (Yih et al., 2016)).
- **MetaQA-3Hop** (Zhang et al., 2018) is the most difficult 3-hop subset of a large-scale KBQA dataset based on Wiki-Movies KG.
- **WikiSQL** (Zhong et al., 2017) is a large-scale complex Text2SQL dataset which requiring comparison, aggregation and arithmetic operations.

C Baselines

C.1 Fine-tuning Methods

- **RnG-KBQA** (Ye et al., 2022) fine-tunes a Seq2Seq model. The input includes various candidate logical forms and KB-items. Then, the model outputs S-expression, which is transformed to SPARQL.
- **ArcaneQA** (Gu and Su, 2022) is a generationbased method that incrementally synthesizes a program by dynamically predicting a sequence of subprograms. It prunes the search space by constrained decoding.
- **Pangu** (Gu et al., 2023) leverages the discrimination ability of language models to build queries in an incremental manner. It consists of a symbolic agent to collect valid candidate plans, and an LM to select the most likely one.

C.2 ICL-based Method

- **Pangu-ICL** (Gu et al., 2023) is a ICL version of Pangu. The result is based on output distribution of the LLMs, which require access to model parameters.

- **KB-Binder** (Li et al., 2023b) propose an ICL-based method for few-shot KBQA by feeding LLM with some (question, S-expression) pairs.

- **KB-Coder** (Nie et al., 2023) further optimizes KB-Binder by changing the target format from S-expression to code-style logic form.

- **Readi** (Cheng et al., 2024) propose a Reasoning-Path-Editing framework which initially generates a reasoning path given a query, then instantiates the path. Editing is triggered only when necessary.

- **BYOKG** (Agarwal et al., 2024) does not need manually annotated dataset. It first explores the KB to collect synthetic datasets within a day (*e.g.,* 10K data in 10 hours), then uses the synthetic dataset for bottom-up reasoning.

C.3 Agent-based Method

- **AgentBench** (Liu et al., 2024) model KBQA as a tool learning task and outfitting LLM with an array of KG-querying tools such as "get_relation", "argmax" and "intersection".

- **FUXI** (Gu et al., 2024) design customized tools acting as middleware between LLMs and complex environments. They also incorporate decoupled generation and error feedback to boost performance.

- **QueryAgent** (Huang et al., 2024) step-by-step build the target query and use an environmental feed-based self-correction method to reduce hallucination.

D Levels of Generalization

In the context of KBQA, the three levels of generalization—I.I.D. generalization, compositional generalization, and zero-shot generalization—refer to the capability of models to handle increasingly challenging and diverse types of questions (Gu et al., 2021). In particular, I.I.D. generalization refers to the model's ability to correctly answer questions that are sampled from the same distribution as the training set, which assumes that the test data follow similar patterns and schema. On top of that, compositional generalization refers to the model's ability to handle novel combinations of KB items (e.g., entities and relations) that were seen during training, but in configurations that the model has not encountered before. Finally, zero-shot generalization refers to the model's ability to answer questions involving entirely new KB items such as unseen entities and relations that were never presented in the training set.

Gu et al. (2021) argue that relying solely on datasets under I.I.D. settings limits the practical use of KBQA models, as real-world questions often involve unfamiliar entities or require novel reasoning. This limitation is evident in the degraded performance of existing KBQA methods under compositional and zero-shot settings compared to I.I.D. settings. Therefore, practical KBQA models should be equipped with built-in generalization capabilities across all three levels to better handle diverse, real-world questions.

E More Details

E.1 Other Experiment Settings

TARGA and Pangu used 10 demonstrations across all datasets. KB-Binder and KB-Coder used 40 shots for GrailQA, 100 shots for GraphQ, and 5 shots for MetaQA, with KB-Binder employing 20 shots for KBQA-Agent. AgentBench, FUXI, and QueryAgent are all 1-shot methods. The experiments using models with 1.5B, 7B, 32B, and 72B parameters were conducted on 1, 1, 4, and 8 A100 GPUs, respectively.

E.2 Logic Form Design

Inspired by PyQL (Huang et al., 2023), we designed a simplified logical form for constructing queries, which uses a series of functions for semantic expression, making it easier for LLMs to learn and enabling seamless translation into SPARQL. Table 9 shows all functions that we used.

E.3 Case Study

Table 10 presents a detailed demonstration of how TARGA operates with a new question.

Table 9: Functions used in our logic form.

Function	**Brief Description**
triplet(s, p, o)	Add a condition that the subject s is linked to the object o via a relation p.
argmax(v)	Add a condition that the variable v must have the maximum value.
argmin(v)	Add a condition that the variable v must have the minimum value.
filter(v, op, value)	Add a condition that the variable v must meet, where the operator op is one of [<, >, <=, >=].
type(v, t)	Specify that the variable v belongs to the class (entity type) t.
count(v)	Set the count of variable v as the final answer.
answer(v)	Set the variable v as the final answer.

Table 10: A case study of TARGA from GrailQA.

Pipeline	Details
Question	the camera with a sensor from canon and a compression format of jpeg (exif 2.21) uses which viewfinder?
Candidate Entities	{Canon: m.01bvx1, jpeg (exif 2.21): m.03h4lt3 }
Candidates Relations	[digicams.digital_camera.viewfinder_type, digicams.camera_compressed_format.cameras, digicams.digital_camera.sensor_type, digicams.camera_uncompressed_format.cameras, digicams.camera_iso.cameras, digicams.camera_format.cameras, digicams.camera_sensor_manufacturer.cameras, ..., digicams.camera_white_balance.cameras]

Pipeline	Details
Query Construction	**Query1** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *answer(?v0)* **Query2** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *answer(?v1)* ..... **Query21** *triplet([canon], digicams.camera_sensor_manufacturer.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *answer(?v1)* ..... **Query56** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.camera_sensor_manufacturer.cameras, ?v0)* *answer(?v1)* ..... **Query71** *triplet([canon], digicams.digital_camera_manufacturer.cameras, ?v0)* *triplet(?v0, digicams.camera_storage_type.compatible_cameras, ?v1)* *answer(?v1)*
Query Re-ranking	**Rank1** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.camera_sensor_manufacturer.cameras, ?v0)* *answer(?v1)* **Rank2** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.digital_camera_manufacturer.cameras, ?v0)* *answer(?v1)* **Rank3** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *answer(?v1)* ..... **Rank71** *triplet([canon], digicams.digital_camera_manufacturer.cameras, ?v0)* *triplet(?v0, digicams.camera_storage_type.compatible_cameras, ?v1)* *answer(?v1)*

Pipeline	Details
Question Generation	**Query1** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.camera_sensor_manufacturer.cameras, ?v0)* *answer(?v1)* **Generated Question** what is the viewfinder type of the digital camera that uses the jpeg (exif 2.21) compressed format and is manufactured by canon? **Query2** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.digital_camera_manufacturer.cameras, ?v0)* *answer(?v1)* **Generated Question** what is the viewfinder type of the canon camera with jpeg (exif 2.21) format? ..... **Query10** *triplet([jpeg (exif 2.21), digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.sensor_resolution, ?v1)* *triplet([canon], digicams.digital_camera_manufacturer.cameras, ?v0)* *answer(?v1)* **Generated Question** what is the sensor resolution of the canon digital camera with jpeg (exif 2.21) format?
LLM Output	*triplet([jpeg (exif 2.21)], digicams.camera_compressed_format.cameras, ?v0)* *triplet(?v0, digicams.digital_camera.viewfinder_type, ?v1)* *triplet([canon], digicams.camera_sensor_manufacturer.cameras, ?v0)* *answer(?v1)*
Parsed SPARQL	*SELECT DISTINCT ?v1 WHERE {* *:m.03h4lt3 :digicams.camera_compressed_format.cameras ?v0 .* *?v0 :digicams.digital_camera.viewfinder_type ?v1 .* *:m.01bvx1 :digicams.camera_sensor_manufacturer.cameras ?v0 .* *}*
Execution Result	{m.01xrg1f: Optical, m.01xrg6z: TTL}
F1 Score	**1.0**