Which bird does not have wings: Negative-constrained KGQA with Schema-guided Semantic Matching and Self-directed Refinement

Midan Shim, Seokju Hwang, Kaehyun Um, Kyong-Ho Lee*
Department of Computer Science, Yonsei University
{midans26, hsjtjrwn, khyun33, khlee89}@yonsei.ac.kr

Abstract

Large language models still struggle with faithfulness and hallucinations despite their remarkable reasoning abilities. In Knowledge Graph Question Answering (KGQA), semantic parsing-based approaches address the limitations by understanding constraints in a user's question and converting them into a logical form to execute on a knowledge graph. However, existing KGQA benchmarks and methods are biased toward positive and calculation constraints. Negative constraints are neglected, although they frequently appear in real-world questions. In this paper, we introduce a new task, NEgative-conSTrained (NEST) KGQA, where each question contains at least one negative constraint, and a corresponding dataset, NestKGQA. We also design PyLF, a Python-formatted logical form, since existing logical forms are hardly suitable to express negation clearly while maintaining readability. Furthermore, NEST questions naturally contain multiple constraints. To mitigate their semantic complexity, we present a novel framework named CUCKOO, specialized to multiple-constrained questions and ensuring semantic executability. CUCKOO first generates a constraint-aware logical form draft and performs schema-guided semantic matching. It then selectively applies self-directed refinement only when executing improper logical forms yields an empty result, reducing cost while improving robustness. Experimental results demonstrate that CUCKOO consistently outperforms baselines on both conventional KGQA and NEST-KGQA benchmarks under few-shot settings.

1 Introduction

With an enormous number of parameters, large language models (LLMs) have demonstrated remarkable reasoning capabilities (Wei et al., 2022; Shinn et al., 2023). Nevertheless, LLMs suffer from hallucinations and faithfulness concerns (Huang et al., 2025; Muneeswaran et al., 2024). To overcome the limitations, research utilizing external knowledge has been actively conducted (Lewis et al., 2020; Kaiser et al., 2024; Wan et al., 2024; Taffa and Usbeck, 2025). Knowledge Graph Question Answering (KGQA) is one of the promising approaches because a Knowledge Graph (KG) provides rich semantics (Ye et al., 2022; Li et al., 2023; Atif et al., 2023; Sun et al., 2024). In particular, Semantic Parsing (SP)-based approaches map a natural language question into a logical form, which is later converted into a graph query (e.g., SPARQL) to execute on a KG. Such graph queries specify an answer derivation process by leveraging graph patterns, leading to an explainable and faithful answer.

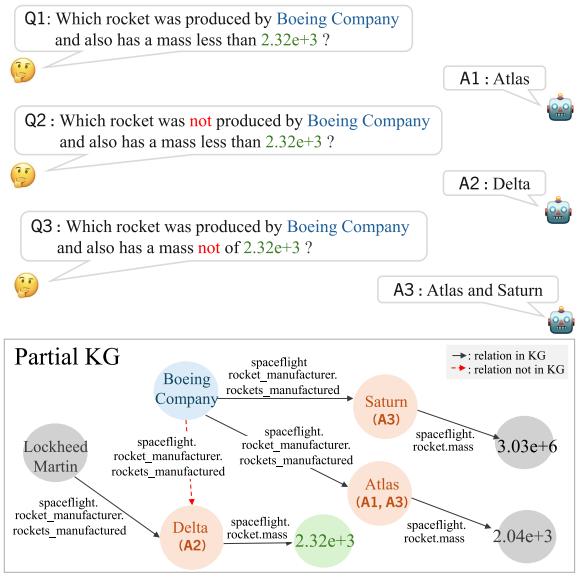

Figure 1: Motivating Example. Partial KG is the union of the subgraphs for Q1-Q3. Q2 requires reasoning over a relation that is not present in the KG unlike Q1 and Q3.

KGQA requires identifying constraints in a question and providing an answer set that satisfies them from a KG. Figure 1 depicts KGQA examples. Q1 contains a positive constraint that is produced by 'Boeing Company', and a comparison of a mass to

* Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 22547–22561

2.32e+3. Although Q2 and Q3 superficially resemble Q1, their constraints are entirely different. Q2 requires a negative constraint on being produced by 'Boeing Company' and a less-than comparison against 2.32e+3. 'Delta', produced by 'Lockheed Martin' rather than 'Boeing Company', is the answer to Q2. Although Q3 contains "not", it essentially involves a comparison of whether the rocket mass equals 2.32e+3 or not. A3 is 'Saturn', which is produced by 'Boeing Company' and has a mass of 3.03e+6.

However, existing KGQA studies tend to be biased towards positive and calculation constraints. To the best of our knowledge, there is no KGQA benchmark focusing on negative-constrained questions such as Q2. Although some datasets (Cao et al., 2022; Usbeck et al., 2024) appear to involve negation words (e.g., 'not') in a question, they actually include comparison exemplified by Q3. Since LLMs are vulnerable to negation reasoning (García-Ferrero et al., 2023; Truong et al., 2023), it is not guaranteed that previous LLM-based methods answer negative-constrained questions as well. In addition, existing logical forms are hardly suitable for negative questions. Only some logical forms based on λ calculus (Krivine, 1993) can express negations (Liang, 2013; Wang et al., 2015), but their application scope is limited. This is because the λ calculus is less readable than a symbolic expression (Ye et al., 2022). To tackle the limitations, we formulate a new KGQA task, NEgative-conSTrained (NEST) KGQA, to answer a question containing at least one negative constraint. We also propose a NestKGQA dataset by extending previous benchmarks. To enhance both expressiveness and readability, we design PyLF, a Python-formatted logical form based on a symbolic expression.

Furthermore, it is challenging to provide accurate answers to NEST questions. This is not only because negation reasoning itself is difficult, but also because NEST questions naturally contain multiple constraints[1]. Therefore, NEST KGQA requires interpreting all of the constraints in a question and converting them into an executable query. In practice, NEST questions are structurally more complex than non-NEST questions, which substantially increases the risk of generating unexecutable queries. This is because existing SP-based studies (Li et al., 2023; Nie et al., 2024; Luo et al., 2024a; Agarwal et al., 2025) rarely consider semantics of a KG. To alleviate this issue, we propose a novel framework, Code generation and schema-gUided matChing for Knowledge graph-based questiOn cOnstraints (CUCKOO). A constraint-aware draft generation module first extracts constraints in a given question and generates a logical form draft in an in-context manner. Then a schema-guided semantic matching module converts the draft into a list of logical forms. Since the matching is grounded on a KG schema, the logical forms are semantically executable. Nevertheless, some flawed drafts with formal defects trigger empty prediction results. We propose a self-directed refinement module that diagnoses errors in the draft and regenerates it without additional parameter tuning. The experimental results reveal that CUCKOO achieves the best or secondary best performance on both NEST and non-NEST benchmarks. Our contributions are as follows[2]:

- We propose a novel task, NEST KGQA and a corresponding benchmark, NestKGQA. To effectively express negative constraints, PyLF is introduced. We also reveal that answering NEST questions is challenging for LLMs.
- We design an SP-based framework named CUCKOO that explicitly models constraint elements in draft generation and conducts self-directed refinement. Experimental results on KGQA benchmarks demonstrate that CUCKOO achieves outstanding generalization performance.
- We present a schema-guided semantic matching method that alleviates the unexecutability of a logical form. It addresses a chronic challenge faced by SP-based approaches.

2 Related Work

2.1 Negation Reasoning on LLMs

Recent studies indicate that LLMs often misinterpret negative statements on various tasks. In natural language inference settings, LLMs often fail to reason on a sentence containing negation words (She et al., 2023). LLMs also tend to ignore "not" and generate answers equivalent to those for affirmative statements (Truong et al., 2023). Ye et al. (2023)

[1]For example, the number of answers to "who did not write Harry Potter?" is tremendous. It is rarely meaningful in realistic QA scenarios.

[2]Dataset and code are available at https://github.com/midannii/CUCKOO

Function	s-expression	Description	Class of output
JOIN(r, t, $neg = bool$)	(JOIN r t)	Find entities that are joined with t by r	c_{domain_r}
JOIN(R_r, h, $neg = bool$)	(JOIN (R r) h)	Find entities that are joined with h by reversed r	c_{range_r}
AND($e1$, $e2$)	(AND $e1$ $e2$)	Find intersection of $e1$ and $e2$	Same as $e1$ and $e2$
COUNT(e)	(COUNT e)	Return the number of e	Integer
ARG(m_a, h, r)	(ARGMIN h r), (ARGMAX h r)	Among entities h, find the entity with the smallest (argmin) or largest (argmax) value of a linked by r	c_{domain_r}
CMP(m_c, r, n)	(lt r n), (le r n), (gt r n), (ge r n)	Find entities that are linked with a by r, where a is less than (lt) / less than or equal to (le) / greater than (gt) / greater than or equal (ge) to n	c_{domain_r}
START(i)	-	Start a logical form with i	Same as i
STOP(e)	-	Stop a logical form	-

Table 1: Description of PyLF functions. Note that a relation $r \in R$, entities $\{e, e1, e2, h, t\} \subset E$, attributes $\{a, n\} \subset A$, and an instance $i \in E \cup A$. m_a and m_c are parameters to determine which operator applies for an answer. $m_a = \{$'ARGMIN', 'ARGMAX'$\}$ and $m_c = \{$'>', '>=', '<', '<='$\}$.

report that LLMs exhibit limited robustness to lexical negation during complex reasoning. Moreover, knowledge-intensive tasks show a significant performance drop when negation operations are involved (Rezaei and Blanco, 2024; Varshney et al., 2025). Negation reasoning poses a significant challenge for LLMs and often magnifies hallucinations. In this paper, we introduce a KGQA task that requires negation reasoning with LLMs.

2.2 KGQA with LLM

Retrieval-augmented approaches are composed of a subgraph retriever and an LLM-based reasoner (Luo et al., 2024b). Some approaches aim to refine knowledge for LLMs (Wu et al., 2024; Xu et al., 2025). Nevertheless, there remain risks of hallucinations, as the reasoning depends on LLMs.
Agent-based approaches treat an LLM as a KG-exploring agent to find the best reasoning paths (Sun et al., 2024; Tan et al., 2025). LLMs are employed to use APIs (Xiong et al., 2024) or tools (Jiang et al., 2023) to interact with KGs. However, multiple-constrained questions require increased LLM calls, raising computational cost and latency.
SP-based approaches leverage an LLM as a logical form generator, exploiting its strong generative abilities (Li et al., 2023). Some approaches reformulate logical form generation as code generation, utilizing programming languages familiar to LLMs (Agarwal et al., 2024; Nie et al., 2024). Meanwhile, a self-correction module is proposed to refine code-like logical forms (Agarwal et al., 2025) with error messages of program execution.

3 Negative-constrained KGQA

3.1 Preliminaries

KG Let E, A, and R denote the sets of entities, attribute values, and relations, respectively. We define a KG $G = \{(h, r, t) \mid h \in E,\ r \in R,\ t \in E \cup A\}$, where a triple ($h$, r, t) indicates a relation r from a head entity h to a tail entity or attribute t. Note that the KG is under closed-world assumptions like conventional KGQA studies to focus on scenarios that rely solely on knowledge within the KG since precision is important (Álvez et al., 2020).
Logical Form l is a machine-interpretable meaning representation that captures semantics of a given question. We use PyLF as a logical form.
KG schema Each r constrains h to belong to $c_{domain_r} \in \mathcal{C}$, and t to belong to $c_{range_r} \in \mathcal{C} \cup \mathcal{D}$. Here, $\mathcal{C}$ denotes classes, each defined as a set of entities, and $\mathcal{D}$ denotes a set of data types. We define (c_{domain_r}, r, c_{range_r}) as a schema-level triple and (h, r, t) as an instance-level triple.
Reasoning path is a linear chain from a topic entity in a question to answer entities. The path is decomposed into one or multiple instance-level triples.
Constraint $S = S_{pos} \cup S_{neg} \cup S_{cal}$. A positive constraint, S_{pos}, is satisfied when all triples in a reasoning path exist in G. S_{neg} is a negative constraint that is satisfied when at least one triple (h, r, t) in a reasoning path is not present in G, where h is an instance of c_{domain_r} and t is an instance of c_{range_r}, as marked by a red dashed line in Figure 1. A calculation constraint, S_{cal}, requires calculations such as comparison and counting. The total number of constraints included in each question is the sum of the number of reasoning paths and the number of calculation constraints.
KGQA Given a natural language question q_i containing a non-empty constraint set $S_i \subset S_{pos} \cup S_{neg} \cup S_{cal}$, KGQA aims to generate l_i to return a non-empty answer set satisfying S_i based on G.
NEST KGQA Among KGQA, NEST KGQA aims to answer q_i containing constraint set S_i where $n(S_i) > 1$ and at least one constraint is in S_{neg}.

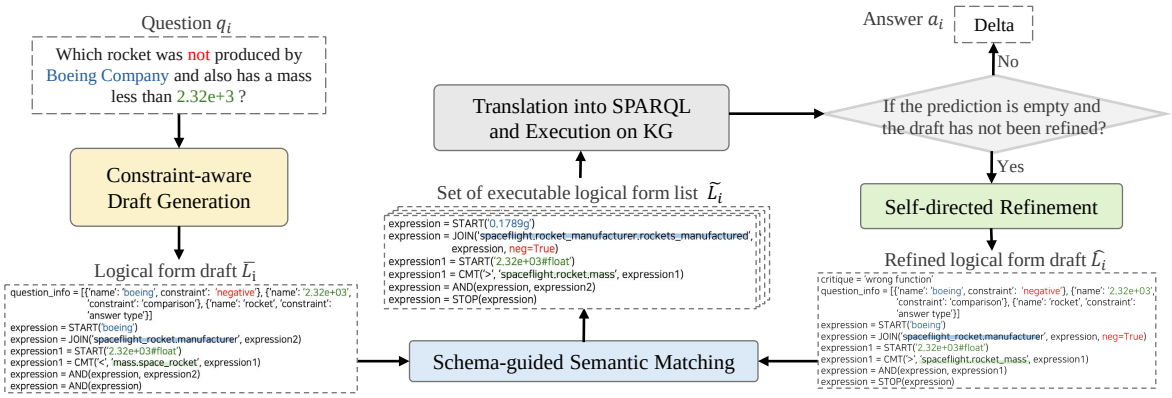

Figure 2: An illustration of CUCKOO framework. A constraint-aware draft generation module first isolates relevant constraint elements in a given question and generates a logical form draft in an in-context learning manner. The draft is then fed into a schema-guided semantic matching module to ground semantically valid KG components. The matched logical form list is converted into SPARQL queries and executed on a KG. Only when the query returns an empty answer set does a self-directed refinement module revise the original draft based on few-shot demonstrations.

3.2 PyLF

In order to improve the expressiveness of negative constraints while maintaining readability, we design a novel logical form, PyLF. Table 1 is a summary of PyLF. In detail, we extend meta-functions (Nie et al., 2024), which are defined as a Python version of s-expression (Gu et al., 2021). This is because LLMs are exposed to substantial amounts of Python code during pre-training, and the functions have been reported to have low syntax errors (Nie et al., 2024).

To indicate negative constraints, we add a boolean argument **neg** in JOIN(). For example, "not producing Saturn" is expressed as JOIN('producing', 'Saturn', *neg = True*), whereas its positive counterpart is expressed as JOIN('producing', 'Saturn', *neg = False*).

We also add a prefix **R_** to clarify whether the answer should be drawn from the head or the tail of a triple. For instance, "Which rocket is produced by Boeing Company?" requires querying a tail entity of (Boeing Company, producing, Saturn). Otherwise, "Which company produces Saturn?" asks a head entity of the same triple. The existing function expresses the questions as JOIN('producing', 'Boeing Company') and JOIN('producing', 'Saturn'), respectively. The difference between the second arguments of them is insufficient to distinguish the semantic difference between the questions. In contrast, PyLF represents the former as JOIN('R_producing', 'Boeing Company'). It allows us not only to perform an elaborate semantic parsing, but also to identify each expression's return type. Given a schema-level triple (rocket manufacturer, producing, rocket)[3], an output of JOIN('producing', 'Saturn') is guaranteed to be an instance of rocket manufacturer. Both modifications are used to filter out invalid functions in schema-guided semantic matching (Section 4.2).

3.3 Data Construction

We construct NestKGQA by modifying existing KGQA datasets, GrailQA (Gu et al., 2021) and GraphQ (Su et al., 2016), since they capture semantic structures of each question explicitly. From the source datasets, we select questions containing at least 2 constraints. Among them, 10 randomly selected examples are annotated by 4 graduate students with background knowledge in KGQA. We employ `GPT-4o` to annotate the rest of the examples in an in-context manner. Given a question, a logical form, and a negative version of the logical form, the model is instructed to generate a new question. The new logical form is made by changing one positive constraint of the question with a negative one (details in Appendix A). Ten demonstrations from the annotators are also included in the prompt for guidance. The generated outputs are cross-checked and refined by the annotators for logical validity. Questions without answers are excluded to ensure the reliability of NestKGQA.

4 CUCKOO

Figure 2 is an illustration of CUCKOO, a generation-and-matching paradigm for KGQA.

[3]The original schema is (spacefight.rocket_manufacturer, spaceflight.rocket_manufacturer.rockets_manufactured, spaceflight.rocket), but it is simplified for a clear explanation.

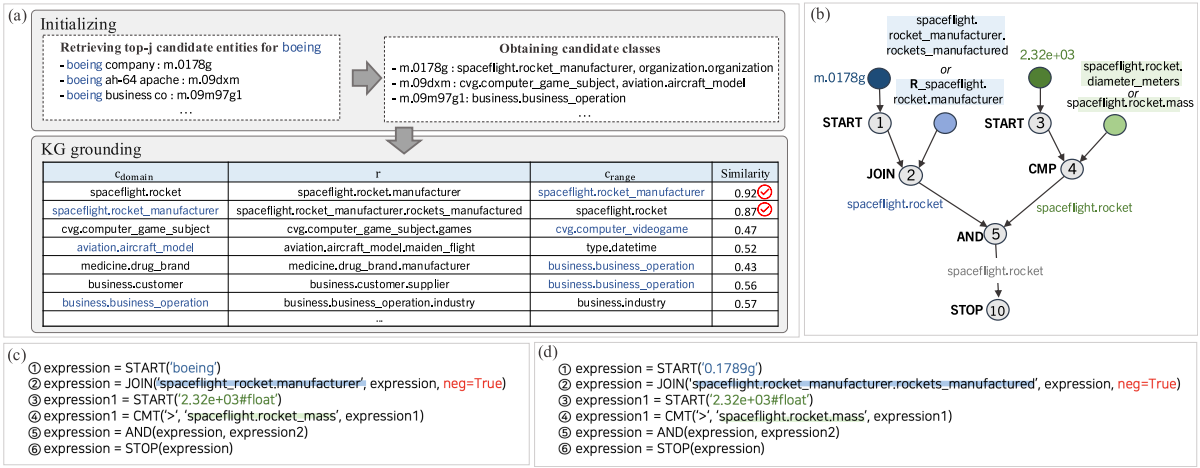

Figure 3: Schema-guided semantic matching. (a) Partial example of schema-guided semantic matching. (b) Logical tree of (d). (c) Example of a draft to be matched. (d) One of the candidate logical form lists after the matching. Each logical form may be converted into a logical tree because PyLF originates from s-expression, which have nested structures. The semantic matching aims to determine which KG item corresponds to each leaf node.

4.1 Constraint-aware Draft Generation

To generate a logical form draft of each question, conventional in-context learning paradigm (Brown et al., 2020) is adopted. Given a question q_i, a logical form draft $\bar{L}^{draft}$ is derived as:

$$\bar{L}^{draft} = f(I^{draft}, F, D_i^{draft}, q_i) \quad (1)$$

with an LLM f, an instruction I_{draft}, PyLF signatures F, and k illustrative demonstrations D^{draft}. The detailed prompt is present in Appendix B. As shown in the lower-left of Figure 2, f is prompted to decompose the question in order to identify the constituent constraints. All topic entities and literals with corresponding constraints in the question are enumerated before generating a logical form draft. This process urges the LLM to explicitly consider all constraints in a question when generating the draft.

4.2 Schema-guided Semantic Matching

Previous studies (Li et al., 2023; Nie et al., 2024) conduct semantic matching in a brute-force manner. For each entity mention, the top K_e entities are matched solely based on cosine similarity to generate logical form candidates. Likewise, for each relation, top K_r relations are selected. Then the candidates are executed one by one on a KG until the execution result is not null. As the logical form grows longer, the number of executions increases exponentially. To avoid this, we propose a schema-guided semantic matching method that exploits KG schema as shown in Figure 3.

Initializing It starts with START() function, which takes a topic entity mention or a numeric value in a question as an input argument. When a mention is a numeric value, we assign a proper data type such as float for '2.32e+03' in Figure 3(c) ③. Cosine similarity between the embeddings of each mention and surface names of entities in a KG is computed to retrieve the top j candidate entities. As shown in Figure 3(a), a mention 'boeing' in Figure 3(c) ① derives multiple candidate entities. They belong to various classes, candidate classes, which are clues to match other mentions in posterior functions.

KG Grounding It first extracts all schema-level triples containing the candidate classes. Then, a comparison of cosine similarity between the embeddings of a relation name in each triple and a relation mention in the draft is conducted. For example, 'spaceflight_rocket.manufacturer', the first argument in Figure 3(c) ②, is compared with elements of r column of the table in Figure 3(a). Among them, relations with a similarity greater than θ are matched to the mention. In this example, spaceflight.rocket_manufacturer.rockets_manufactured and spaceflight.rocket.manufacturer are matched. This leads to 'boeing' in Figure 3(c) ① matched into `m.0178g`, an instance of spaceflight.rocket_manufacturer. According to Table 1, an output of Figure 3(c) ② is an instance of spaceflight.rocket.

The matched entity and relations are marked with blue nodes of a logical tree in the upper left corner of Figure 3(b). The same process is applied to other nodes to complete all nodes in the logical

tree. As a result, expression ① is matched with one entity. Expressions ② and ④ are matched with 2 and 2 relations, respectively. The logical form draft yields 1×2×2=4 executable logical form candidates. Whereas existing methods require $K_e^1 K_r^2$ executions. Our method achieves a considerable reduction in computation cost. Figure 3(d) is an example of the candidates. Each candidate is converted into SPARQL and executed on a KG. As in the previous studies (Li et al., 2023; Nie et al., 2024), the first executed query is considered the final prediction.

4.3 Self-directed Refinement

Although the generated logical form is semantically executable by schema-guided semantic matching methods, some drafts may possess the wrong format, be grammatically incorrect, or fail to understand the constraints. To mitigate this, we invoke a self-directed refinement stage, triggered only once when the query execution result is empty.

As depicted in the lower-right part of Figure 2, CUCKOO first identifies a critique category for refinement from predefined error cases: an absence of constraint decomposition (no question_info), incorrect formatting of constraint decomposition (wrong question_info), incorrect PyLF function syntax (wrong expression), and incorrect formatting of both components (wrong format). Then CUCKOO generates a new draft. Unlike previous code generation approaches (Agarwal et al., 2025; Zheng et al., 2024), CUCKOO is independent of access to external execution feedback and additional LLM calls, alleviating cost and latency. Formally, the refined draft for q_i is obtained by:

$$\hat{L}_i = f(I^{refine}, F, D_i^{refine}, q_i, \bar{L}_i) \quad (2)$$

where I^{refine} denotes a refinement-specific instruction and D_i^{refine} is a set of demonstrations that include training questions, an initial draft, and a gold logical form. F and $\bar{L}_i$ are the same as in Equation (1). The details are in Appendix B.

5 Experiments

All experiments were conducted to answer the following research questions (RQ): (1) Are LLMs skillful at answering questions that require negation reasoning on KGQA? (2) Is CUCKOO superior to previous SP-based methods in terms of both performance and efficiency? (3) How does each module of CUCKOO contribute? (4) How well does CUCKOO perform on challenging questions, such as multiple constraints and calculations?

	Method	GrailQA	WebQSP	GraphQ	NestKGQA
LLM I/O	Llama3-8B-Instruct	15.8	39.1	35.9	12.0
	QWEN 2.5-7B	25.7	39.7	28.5	20.1
	DeepSeek-v2-Lite	11.5	25.0	16.2	3.0
	DeepSeek-R1-QWEN-7B	18.4	20.1	13.6	8.3
	GPT-3.5-turbo	24.0	63.3	21.6	5.4
	GPT-4o-mini	25.9	49.4	28.2	9.7
LLM w/CoT	Llama3-8B-Instruct	27.7	45.7	35.2	12.8
	QWEN 2.5-7B	26.9	39.7	31.2	20.3
	DeepSeek-v2-Lite	12.5	27.6	19.1	4.5
	DeepSeek-R1-QWEN-7B	20.6	20.1	15.4	10.2
	GPT-3.5-turbo	27.0	62.2	21.8	7.7
	GPT-4o-mini	32.9	61.7	28.6	11.8

Table 2: Exact Match result on Zero-shot LLM.

5.1 Experimental Setup

Dataset and Evaluation Metrics The overall experimental setup follows previous SP-based KGQA studies (Li et al., 2023; Nie et al., 2024). WebQSP (Yih et al., 2016), GraphQ (Su et al., 2016), GrailQA (Gu et al., 2021), and our proposed NestKGQA are used to evaluate CUCKOO. The details of each dataset are given in Appendix C.1. We reported results using Exact Match (EM) and F1 score (percentage).

Baselines To evaluate the performance of LLMs on NEST KGQA, we set various LLMs as baselines: GPT-3.5-turbo, GPT-4o-mini (Achiam et al., 2023), Llama3-8B-Instruct (Grattafiori et al., 2024), DeepSeek-v2-Lite (16B) (Liu et al., 2024), QWEN 2.5-7B (Hui et al., 2024) and DeepSeek-R1-QWEN-7B (Guo et al., 2025). We also conducted experiments using the Chain-of-Thought (CoT) (Wei et al., 2022), which is known to improve reasoning ability. Meanwhile, we compared CUCKOO with SP-based KGQA models, KB-BINDER (Li et al., 2023) and KB-Coder (Nie et al., 2024) based on in-context learning. We also included fully supervised methods in baselines: DecAF (Yu et al., 2023), TIARA (Shu et al., 2022), ArcaneQA (Gu and Su, 2022) and SG-KBQA (Gao et al., 2025). The details of each baseline are in Appendix C.2

Implementation Details The overall implementation follows KB-BINDER (Li et al., 2023) and KB-Coder (Nie et al., 2024). In the draft generation module, we selected the top k demos from the training data for each test example based on the cosine similarity of SimCSE embedding (Gao et al., 2021). We set k as 40 for GrailQA/NestKGQA and 100 for WebQSP/GraphQ. For self-directed refinement, top 10 demonstrations are selected for

Methods	I.I.D.		Compositional		Zero-shot		Overall	
	EM	F1	EM	F1	EM	F1	EM	F1
Full Supervised on the Entire Training set								
DecAF	88.7	92.4	71.5	79.8	65.9	74.7	72.5	81.4
TIARA	88.4	91.2	66.4	74.8	73.3	77.3	75.3	81.9
SG-KBQA	93.1	94.6	78.4	83.6	84.4	87.9	85.1	88.5
In-Context Learning (Training-Free)								
KB-BINDER (1)	71.8	72.4	40.8	47.6	37.1	40.8	46.1	49.8
KB-Coder (1)	72.0	72.6	44.6	48.5	43.2	48.3	50.3	54.1
CUCKOO (1)	75.4	76.8	53.2	56.9	54.3	57.0	59.0	61.6
KB-BINDER (6)	73.4	74.8	46.3	46.9	45.9	48.6	52.5	54.5
KB-Coder (6)	74.8	72.9	45.5	50.8	46.7	51.6	51.2	56.3
CUCKOO (6)	77.8	78.9	56.5	59.2	57.5	59.8	62.1	64.2

Table 3: EM and F1 results on GrailQA.

Methods	WebQSP	GraphQ	NestKGQA
Full Supervised on the Entire Training set			
ArcaneQA	75.6	31.8	7.8
TIARA	78.7	37.9	0.0
SG-KBQA	80.3	43.5	2.3
In-Context Learning (Training-Free)			
KB-BINDER (1)	68.9	26.7	3.7
KB-Coder (1)	72.2	30.0	21.7
CUCKOO (1)	69.0	36.7	20.3
KB-BINDER (6)	70.4	32.7	4.6
KB-Coder (6)	75.2	35.8	24.4
CUCKOO (6)	70.8	40.8	26.2

Table 4: F1 results on WebQSP, GraphQ and NestKGQA

draft generation. GPT-3.5-turbo is employed as f to generate 1 and 6 candidates denoted as (1) and (6), respectively. The final prediction was determined by majority voting (Wang et al., 2023). The temperature is set to 0.9. In schema-guided semantic matching, we set j=10 to retrieve entity candidates via cosine similarity between SimCSE embeddings. To enhance efficiency, faiss indexing (Johnson et al., 2019) is used for all relations and entity names. A similarity threshold is applied to $\theta = 0.7$ for GrailQA/NestKGQA and $\theta = 0.8$ for WebQSP/GraphQ. All experiments are conducted on a single NVIDIA RTX 3090 Ti (24 GB). We reported fully reproduced results to ensure strict comparability.

5.2 Main Results

To verify RQ1, we reported zero-shot performance on NestKGQA for various LLMs. As in Table 2, EM scores of all LLMs are significantly lower than those of existing KGQA datasets. The result suggests that NEST KGQA, which requires both negation reasoning and understanding multiple constraints within a question, is challenging for LLMs.

Tables 3 and 4 demonstrate performance on KGQA benchmarks of CUCKOO compared to the baselines. CUCKOO achieves state-of-the-art or secondary best in training-free settings. Our framework especially demonstrates significant improvements on challenging questions, such as Compositional and Zero-shot splits of GrailQA. CUCKOO also achieves the secondary best after KB-Coder on WebQSP, which consists entirely of I.I.D. According to the definition of I.I.D. setting, the same example is assured to exist in k-shot demonstrations. Since KB-Coder simply mimics a ground-truth draft in the demonstration, it is more likely to answer the I.I.D. questions. In this case, enumerated constraint elements in CUCKOO's draft generation may act as noise. This tendency is also observed in the I.I.D. split of GrailQA. On the other hand, CUCKOO outperforms the baselines on GraphQ, which is constructed solely under compositional settings. The result indicates that CUCKOO's strength lies not merely in superficial matching between questions and logical forms, but in its generalization ability.

Moreover, CUCKOO achieves the highest F1 score in NestKGQA among all baselines. KB-Coder ranks second with a substantial gap from KB-BINDER. The result suggests that a Python-formatted logical form offers an advantage in handling complex questions. As a result, CUCKOO consistently excels in handling not only conventional KGQA but also NEST KGQA (RQ2).

Meanwhile, supervised fine-tuning methods perform poorly on NestKGQA. In particular, TIARA and SG-KBQA achieve F1 scores below 5. This

Method	GrailQA	NestKGQA
CUCKOO	64.2	26.2
w/o Self-directed Refinement	63.2	25.8
w/o Constraint Elements	61.3	24.4
w/o Schema-guided Semantic Matching	56.6	16.3

Table 5: Ablation Study of CUCKOO.

Method	input tokens	Inference time	CPU Memory
KB-Coder	5598.08	23.76s	4.7 GB
CUCKOO	6289.06	38.45s	4.4 GB
w/o Refinement	6256.80	32.25s	4.4 GB

Table 6: Comparison of efficiency per question.

suggests that the models fail to encourage logical-form generation fundamentally, but may lead them to induce surface-level mimicry.

5.3 Ablation Study

We attempted to prove whether CUCKOO's components are necessary (RQ3). Table 5 reports F1 scores for the ablation study.

Since self-improvement occurs only in specific situations, it contributes slightly to performance improvement. Constraint elements also contribute to understanding constraints in a question. Eliminating the elements in both draft generation and refinement reduces performance. The decline confirms that surfacing constraints is substantial to help the model comprehend the question. Most of all, the schema-guided semantic matching module is essential. Replacing the module with a canonical brute-force matching noticeably lowers all metrics for both datasets. Since the brute-force manner considers all possible cases, it is expected to achieve higher performance. Nevertheless, it actually increases the possibility of wrong but executable queries as a final prediction. In contrast, our semantic matching achieves good performance while reducing the number of execution cases.

5.4 Further Analysis

For RQ2 and RQ4, we examine CUCKOO with respect to efficiency and robustness on challenging questions. We also conducted an error analysis and a case study, in Appendix D and E, respectively.

Analysis of the number of constraints Figure 4(a) demonstrates CUCKOO's performance with respect to the number of constraints in a question. CUCKOO maintains a consistent margin over KB-Coder across all levels, especially on complex questions. For questions with three constraints, CUCKOO achieves the highest EM score. This indicates that CUCKOO predominates in interpreting multiple constraints and deriving a precise answer.

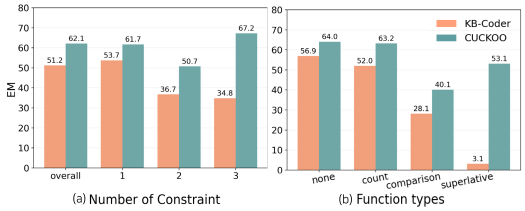

Figure 4: In-depth analysis on GrailQA.

Analysis of function types Figure 4(b) shows whether CUCKOO is fluent in handling various calculations, annotated as 'function' in GrailQA benchmark. Performance of CUCKOO has improved not only for relatively simple calculations such as none and count, but also for comparisons and superlatives. In particular, there is a remarkable improvement in performance from 3.1 to 53.1 for superlative questions, which are the most difficult for the baseline. The result reveals that CUCKOO maintains robustness regardless of calculation type.

Analysis of Efficiency Table 6 shows the efficiency of CUCKOO compared to the existing method. The average number of LLM input tokens per question is about 12.3% higher for CUCKOO than KB-Coder. This is because constraints are enumerated together during the draft generation stage. Also, CUCKOO's inference time is about 1.6 times higher than KB-Coder due to two factors: extra constraint enumeration during draft generation, and additional lookup time for loading entity classes and schema information in schema-guided semantic matching. In addition, self-directed refinement affects execution time as it requires not only calling the LLM for draft regeneration but also performing semantic matching once again. However, in terms of CPU memory usage, CUCKOO achieved a 4.7% reduction relative to the baseline. It is due to schema-based candidate pruning rather than brute-force search in semantic matching.

6 Conclusion

This paper introduces a novel KGQA task incorporating negative-constrained questions that is under-explored in prior studies. To express and evaluate negative constraints, we constructed NestKGQA dataset and PyLF. Furthermore, CUCKOO, an in-context learning framework, is designed to explicitly model constraint elements within questions. To address unexecutability, which is a long-standing issue in SP-based methods, we proposed a schema-

guided semantic matching method. A self-directed refinement module is employed only when execution yields empty results, improving robustness without additional parameter tuning. Experiments demonstrate that CUCKOO achieves outstanding performance across various KGQA benchmarks.

Limitations

Despite the promising results, our study has several limitations. First, our method assumes closed-world assumptions, which may be limited in applicability to open-world scenarios. Second, the size of NestKGQA dataset remains relatively small, as it is constructed by extending the existing benchmark. Third, our study assumes the availability of a complete schema. When the schema is implicit or only partially specified, the proposed approach may require an additional schema extraction model. Lastly, since CUCKOO is based on in-context learning, its performance is influenced by the backbone LLM. Future work includes relaxing closed-world assumptions, leveraging reasoning ability of LLMs while alleviating hallucinations, and investigating model-agnostic strategies to reduce dependency on specific LLMs.

Ethical Considerations

This paper aims to address negative-constrained questions on KGQA by designing new benchmark and models. During the data construction, we confirm the copyright licenses of seed datasets. We employ LLMs' generalization and reasoning abilities on draft generation and self-directed refinement of the proposed model. We also perceive that LLMs may generate inappropriate outputs due to inherent biases in their parametric knowledge. To alleviate this, the model is designed to filter out any invalid PyLF drafts before semantic matching. Meanwhile, we employed an LLM to generate a sketch of NestKGQA. The generated outputs are subsequently cross-checked and refined by graduate student annotators with relevant knowledge, who were appropriately compensated for their time.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Prerna Agarwal, Nishant Kumar, and Srikanta Bedathur. 2024. Symkgqa: Few-shot knowledge graph question answering via symbolic program generation and execution. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10119–10140.

Prerna Agarwal, Nishant Kumar, and Srikanta Bedathur Jagannath. 2025. Aligning complex knowledge graph question answering as knowledge-aware constrained code generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3952–3978.

Javier Álvez, Itziar Gonzalez-Dios, and German Rigau. 2020. Applying the closed world assumption to sumo-based fol ontologies for effective commonsense reasoning. In *ECAI 2020*, pages 585–592. IOS Press.

Farah Atif, Ola El Khatib, and Djellel Difallah. 2023. Beamqa: Multi-hop knowledge graph question answering with sequence-to-sequence prediction and beam search. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 781–790.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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.

Shengxiang Gao, Jey Han Lau, and Jianzhong Qi. 2025. Beyond seen data: Improving KBQA generalization through schema-guided logical form generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8764–8783, Suzhou, China. Association for Computational Linguistics.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910.

Iker García-Ferrero, Begoña Altuna, Javier Álvez, Itziar Gonzalez-Dios, and German Rigau. 2023. This is not a dataset: A large negation benchmark to challenge large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8596–8615.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, pages 3477–3488.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne 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.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Magdalena Kaiser, Rishiraj Saha Roy, and Gerhard Weikum. 2024. Robust training for conversational question answering models with reinforced reformulation generation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 322–331.

Jean Louis Krivine. 1993. *Lambda-calculus, types and models*. Ellis Horwood.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. 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.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *arXiv e-prints*, pages arXiv–1309.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Haoran Luo, E Haihong, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, et al. 2024a. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2039–2056.

Linhao Luo, Yuan-Fang Li, Reza Haf, and Shirui Pan. 2024b. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*.

I Muneeswaran, Advaith Shankar, V Varun, Saisubramaniam Gopalakrishnan, and Vishal Vaddina. 2024. Mitigating factual inconsistency and hallucination in large language models. In *WSDM*, pages 1169–1170.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2024. Code-style in-context learning for knowledge-based question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18833–18841.

MohammadHossein Rezaei and Eduardo Blanco. 2024. Paraphrasing in affirmative terms improves negation understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 602–615, Bangkok, Thailand. Association for Computational Linguistics.

Jingyuan S She, Christopher Potts, Samuel Bowman, and Atticus Geiger. 2023. Scone: Benchmarking negation reasoning in language models with fine-tuning and in-context learning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1803–1821.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

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 *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*.

Tilahun Abedissa Taffa and Ricardo Usbeck. 2025. Bridge-generate: Scholarly hybrid question answering. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 1321–1325.

Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, and Wenjie Zhang. 2025. Paths-over-graph: Knowledge graph empowered large language model reasoning. In *Proceedings of the ACM on Web Conference 2025*, pages 3505–3522.

Thinh Hung Truong, Timothy Baldwin, Karin Verspoor, and Trevor Cohn. 2023. Language models are not naysayers: an analysis of language models on negation benchmarks. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 101–114, Toronto, Canada. Association for Computational Linguistics.

Ricardo Usbeck, Xi Yan, Aleksandr Perevalov, Longquan Jiang, Julius Schulz, Angelie Kraft, Cedric Möller, Junbo Huang, Jan Reineke, Axel-Cyrille Ngonga Ngomo, et al. 2024. Qald-10–the 10th challenge on question answering over linked data: Shifting from dbpedia to wikidata as a kg for kgqa. *Semantic Web*, 15(6):2193–2207.

Neeraj Varshney, Satyam Raj, Venkatesh Mishra, Agneet Chatterjee, Amir Saeidi, Ritika Sarkar, and Chitta Baral. 2025. Investigating and addressing hallucinations of LLMs in tasks involving negation. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pages 580–598, Albuquerque, New Mexico. Association for Computational Linguistics.

Fanqi Wan, Xinting Huang, Leyang Cui, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Mitigating hallucinations of large language models via knowledge consistent alignment. *CoRR*.

Xingyao Wang, Sha Li, and Heng Ji. 2023. Code4struct: Code generation for few-shot event structure prediction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3640–3663.

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 (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yike Wu, Yi Huang, Nan Hu, Yuncheng Hua, Guilin Qi, Jiaoyan Chen, and Jeff Pan. 2024. Cotkr: Chain-of-thought enhanced knowledge rewriting for complex knowledge graph question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3501–3520.

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10561–10582.

Derong Xu, Xinhang Li, Ziheng Zhang, Zhenxi Lin, Zhihong Zhu, Zhi Zheng, Xian Wu, Xiangyu Zhao, Tong Xu, and Enhong Chen. 2025. Harnessing large language models for knowledge graph question answering via adaptive multi-aspect retrieval-augmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25570–25578.

Mengyu Ye, Tatsuki Kuribayashi, Jun Suzuki, Goro Kobayashi, and Hiroaki Funayama. 2023. Assessing step-by-step reasoning against lexical negation: A case study on syllogism. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14753–14773.

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)*, pages 6032–6043, Dublin, Ireland. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher 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.

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

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12834–12859.

```
// Instruction
Rewrite the given question into negative question.
Referring to the data example. Try to use the same words you would have used in the original sentence.
// Example
--Example 1--
Original question: 'who does christopher walken play in batman returns?'
Original codes:
        expression = START('christopher walken')
        expression = JOIN('film.actor.film', expression)
        expression1 = START('batman returns')
        expression1 = JOIN('film.performance.film', expression1)
        expression = AND(expression, expression1)
        expression = JOIN('film.performance.character', expression)
        expression = STOP(expression)
New codes:
        expression = START('christopher walken')
        expression = JOIN('film.actor.film', expression)
        expression1 = START('batman returns')
        expression1 = JOIN('film.performance.film', expression1, neg = True)
        expression = AND(expression, expression1)
        expression = JOIN('film.performance.character', expression)
        expression = STOP(expression)
New questions: 'who does christopher walken play not in batman returns?'

// There are 9 examples omitted here.

// New example
question = 'how many soundtracks are there in the albums of hip house musical genre?'
-- Test Example--
Original question: 'Which rocket manufacturer collaborate with boeing company on making a rocket with mass over 2.1e+06?'
Original codes:
        expression = START('boeing')
        expression = JOIN('spaceflight.rocket.manufacturer', expression)
        expression1 = START('2.916e+06^^http://www.w3.org/2001/XMLSchema#float')
        expression1 = CMP('ge', 'spaceflight.rocket.mass', expression1)
        expression = AND(expression, expression1)
        expression = JOIN('spaceflight.rocket_manufacturer.rockets_manufactured', expression)
        expression2 = START('Saturn v')
        expression2 = JOIN('spaceflight.rocket.manufacturer', expression2)
        expression = AND(expression, expression2)
        expression = STOP(expression)
New codes:
        expression = START('boeing')
        expression = JOIN('spaceflight.rocket.manufacturer', expression)
        expression1 = START('2.916e+06^^http://www.w3.org/2001/XMLSchema#float')
        expression1 = CMP('ge', 'spaceflight.rocket.mass', expression1)
        expression = AND(expression, expression1)
        expression = JOIN('spaceflight.rocket_manufacturer.rockets_manufactured', expression)
        expression2 = START('Saturn v')
        expression2 = JOIN('spaceflight.rocket.manufacturer', expression2, neg=True)
        expression = AND(expression, expression2)
        expression = STOP(expression)
New questions:
```

Figure 5: Prompt for data construction

A Details of NestKGQA Construction

Existing SP-based KGQA datasets (Gu et al., 2021; Su et al., 2016) are generally constructed through template-based automatic generation, followed by careful human refinement. In line with this standard pipeline, we automatically produce initial question drafts and subsequently subject them to detailed human review. The detailed generation procedure is as follows.

Step 1: Selection of source examples GrailQA (Gu et al., 2021) and GraphQ (Su et al., 2016) were designed to include diverse reasoning hops and function types. Since NestKGQA builds upon these datasets, it naturally preserves their structural diversity with respect to reasoning complexity and functional scope. From the source datasets, we selected all questions containing two or more constraints in their executable queries, resulting in 3,468 examples from GrailQA and 483 examples from GraphQ.

I^{draft}

```
'''
You have to understand a given question and generate corresponding expressions.
First, Given a question, generate question_info as a list of components. Each component should include:
'name': the key item or concept mentioned in the question.
'constraint': the role or restriction it represents (e.g. "positive" for the target entity, "comparison" for the comparison, "answer type" for what is being asked).
Then, use the functions defined below to generate the expression corresponding to the question step by step. You may refer question_info. Test example is analogous to Training examples, so apply a similar solution approach.
'''
```

F

```
def START(entity:str):
  return entity

def JOIN(relation:str, expression: str, neg: bool):
  return '(JOIN {} {})'.format(relation, expression)

def AND(expression:str, sub_expression: str):
  return '(AND {} {})'.format(expression, sub_expression)

def ARG(operator:str, expression: str, relation: str):
  assert operator in ['ARGMAX', 'ARGMIN']
  return '({} {} {})'.format(operator, expression, relation)

def CMP(operator: str, relation: str, expression: str):
  assert operator in ['<', '<=', '>', '>=']
  return '({} {} {})'.format(operator, relation, expression)

def COUNT(expression: str):
  return '(COUNT {})'.format(expression)

def STOP(expression: str):
  return expression
```

D_i^{draft}

```
question = 'who was the person that produced idol?'
question_info = [{'name': 'americal idol', 'constraint': 'positive'}, {'name': 'person', 'constraint': 'answer type'}]
expression = START('american idol')
expression = JOIN('broadcast.producer.produces', expression)
expression = STOP(expression)

## There are k-1 examples omitted here.
```

q_i

```
question = 'how many soundtracks are there in the albums of hip house musical genre?'
```

Figure 6: Prompts for constraint-aware draft generation.

Step 2: Draft generation For each selected instance, an initial negative draft was generated using GPT-4o via carefully designed prompts as depicted in Figure 5. The prompts were structured to preserve the original semantic intent while explicitly modifying or negating the relevant constraints. The parts highlighted in blue in the figure show where the original logical form was modified to generate the negative version.

Step 3: Human verification and refinement All generated drafts were reviewed by four independent human annotators. Each question was manually evaluated for (i) linguistic naturalness, (ii) plausibility, (iii) correctness of the modified constraints, and (iv) the existence of at least one valid answer. Questions without at least one valid answer were removed. Based on criteria (i)-(iii), questions were manually revised to achieve natural phrasing while maintaining the intended semantics and constraints. Inclusion in the final dataset required agreement from at least two annotators.

Step 4: Final statistics After this quality control process, 3,252 examples from GrailQA and 476 from GraphQ were retained in the final dataset. The resulting dataset was split into 2,514 training examples and 1,214 test examples.

B Details for CUCKOO

Constraint-aware draft generation Figure 6 is an entire prompt for draft generation. I^{draft} instruct the LLM to generate drafts for q_i referring F to follow syntax of the logical forms. Just before generating the drafts, the LLM has to enumerate constraints in q_i.

Self-directed refinement Based on our preceding error analysis[4], we found that most incorrect drafts fail to produce valid SPARQL queries, primarily due to formatting errors or invalid function syntax. Therefore, rather than focusing on localized corrections, we design the refinement stage to encourage the model to introspect on and revise the draft as a whole. The entire prompt for self-directed refinement is shown in Figure 7. The prompt partially shares the same components as the prompt in Figure 6. However, I^{refine} first asks the LLM to identify the critique most relevant to refinement.

The critique candidates correspond to error types observed during draft generation: an absence of constraint decomposition (`no question_info`), wrong constraint decomposition (`wrong question_info`), wrong PyLF function (`wrong expression`), and wrong formatting (`wrong format`). According to the critique, the LLM is employed to revise incorrect drafts into a desired draft. D^{refine} contains fixed 10 examples, which are 1, 2, 3, and 4 examples for `no question_info`, `wrong question_info`, `wrong expression`, and `wrong format`, respectively.

I^{refine}

```
'''
You have to understand a given question and regenerate corresponding expressions.
Before regenerate question_info and expression of a given question,
you have to review a previous output for a given question by selecting critique among ['no
question_info', 'wrong question_info', 'wrong expressions', 'wrong format']
and then regenerate the question_info and expression based on the feedback.
Use the functions defined below to generate the expression corresponding to the question step by
step.
Test example is analogous to Training examples, so apply a similar solution approach.
After writing the expression, double-check that it fully addresses the question. If you find any
mistake or missing step, correct it before giving the final code.
'''
```

F

```
def START(entity:str):
    ...
```

D_i^{refine}

```
question = 'how many soundtracks are there in the albums of hip house musical genre?'
## 1. Here's the previous output
calculation = 'count'
numeric = 'none'
entities = ['hip house']
expression = START('hip house')
expression = JOIN('music.genre.albums', expression)
expression = COUNT(expression)
expression = STOP(expression)

## 2. select critique and regenerate question_info and expressions of the question based on critique
critique = 'wrong format'
Question_info = [{'name': 'hip house', 'constraint': 'positive'}, {'name': 'soundtrack', 'constraint':
                 'answer type'}]
expression = START('hip house')
expression = JOIN('music.genre.parent_genre', expression)
expression = JOIN('music.genre.albums', expression)
expression = COUNT(expression)
expression = STOP(expression)

## There are 9 examples omitted here.
```

q_i

```
question = 'what are the options of manufacturer for bipropellant rocket engines with at least
2.361e+06 newton sea level trust?'
```

$\overline{L_i}$

```
## Here's the previous output
['2.361e+06', 'rocket engines', 'newton sea']

## 2. select critique and regenerate question_info and expressions of the question based on critique
```

Figure 7: Prompts for self-directed refinement.

Dataset	Generalization Level	Train	Dev	Test
WebQSP	I.I.D.	3,098	-	1,639
GraphQ	Compositional	2,381	-	2,395
GrailQA	I.I.D., Zero-shot, Compositional	44,337	6,763	13,231
NestKGQA	Zero-shot, Compositional	2,514	-	1,214

Table 7: Data Statistics.

C Experiment Details

C.1 Datasets

Table 7 is a summary of the statistics of all the benchmarks in this paper. All of the datasets use Freebase as a KG.

WebQSP (Yih et al., 2016) is a dataset constructed under a fully I.I.D. setting. The dataset mainly consists of non-calculation questions involving 1 to 2 hops. **GraphQ** (Su et al., 2016) is focused on questions with complex semantic structures. It includes questions with subgraphs of up to 3 hops and handles various calculations such as counting, comparison, and superlatives. Note that GraphQ is a fully compositional setting. **GrailQA** (Gu et al., 2021) is similar to GraphQ but is a dataset that aims to improve the generalization ability of KGQA systems. It evaluates data that includes not only i.i.d. or compositional data but also zero-shot entities and schemas. **NestKGQA** proposed in this paper shares similarities with the source datasets GR and GQ. To incorporate negative constraints, which include two or more constraints, the dataset only includes multi-hop and multi-entity questions.

C.2 Baselines

C.2.1 In-context Learning Methods

KB-BINDER (Li et al., 2023) let LLM generate a logical form draft for a given question based on few-shot demonstrations, then directly references the KB via BM25-based matching to bind entities/relations, converting it into an executable logical form. Meanwhile, to reduce format errors when the LLM generates unfamiliar logical form formats directly, **KB-Coder** (Nie et al., 2024) reformulate logical form generation as code generation. It aligns existing s-expressions into code style, generates drafts via in-context learning, and converts/validates the results into executable queries to enhance few-shot performance and stability.

[4]We found that 11% of the errors were due to the absence of constraint decomposition, 23% to incorrect constraint decomposition, 29% to incorrect PyLF functions, and 37% to formatting errors.

C.2.2 Fully Supervised Learning Methods

ArcaneQA (Gu and Su, 2022) is a candidate enumeration approach that progressively generates and expands the program (logical expression) step-by-step. At each step, dynamic program induction reduces the search space. Also **TIARA** (Shu et al., 2022) retrieves the question-related entity, exemplary logical form, and schema item separately, then uses all of them as input to generate a logical form through supervised learning. During the generation phase, constrained decoding controls the output space to reduce non-executable/non-syntactic programs. Rather than considering the schema, it compensates for generating incomplete class and relation names.

Instead of generating only a logical form (risking execution failure) or directly generating the answer (weak justification), **DecAF** (Yu et al., 2023) jointly generates the direct answer and logical form, then combines them to form the final answer. Unexecutable issues arising during semantic parsing are temporarily mitigated by generating the direct answer. **SG-KBQA** (Gao et al., 2025) aims to train a model to generate logical forms by incorporating schema information for each relation as input during logical form generation. Furthermore, during the generation phase, a fully-supervised LLM utilizes the schema to perform entity disambiguation. While actively leveraging schema like CUCKOO, SG-KBQA incurs the cost issue of requiring LLM training.

D Error Analysis

To understand the limitations of CUCKOO, we analyzed the errors in the GrailQA dataset and classified them into four main types.

- **Wrong function error (54.07%):** In this case, the model uses a pyLF function sequence that differs from the gold logical form. For example, a gold logical form requires a JOIN-JOIN-AND composition, but the model instead generates a JOIN-AND structure. Such errors predominantly occur in examples on the compositional setting, where the model must handle unseen logical form compositions.

- **Entity linking error (25.67%):** This error type primarily occurs when entity mentions are misrecognized during the constraint-aware draft generation stage. In some examples, errors propagate due to missing entity mentions in the FAISS index.

- **KB component linking error (20.19%):** This occurs when the generated PyLF contains incorrect components, relations and classes. Freebase components have long names, resulting in many components with similar names, such as broadcast.tv_station_owner.tv_stations and broadcast.tv_station.founded. Due to the nature of in-context learning, inference involves following training examples to make predictions. During this process, it often fails to generate unseen components accurately and instead generates components similar to those in the training examples. The issue propagates to the semantic matching stage, leading to downstream errors. Meanwhile, incorrect relation matching also occurs when the domain and range are identical, but the direction of the relation differs within the JOIN() function.

- **Wrong format error (0.07%):** Only a very small number of examples (two cases) fall into this category. Despite applying self-directed refinement, the LLM fails to follow the output format and instead produces answers in the CoT style.

E Case Study

Figure 8 shows a case study of CUCKOO with the baselines. Since both KB-Coder and CUCKOO employ GPT-3.5-turbo in an in-context learning, we conducted the case study under the GPT family.

A question in Figure 8(a) has two constraints: a negative constraint, 'not in the International System of Units,' and a calculation constraint, 'to measure less than 1e-15 seconds.' The gold answer is 'Planck Time'. The LLM-only and LLM-CoT return femtoseconds (1e-15), failing to understand any constraints. They focus narrowly on the numeric component while ignoring the negative constraint. GPT-4o-mini partially considers the calculation, but still violates the negation. This shows that LLMs struggle with negative constraints. Compared to in-context KB-Coder, which also misinterprets negation, CUCKOO produces a correct answer that satisfies both constraints. Likewise, a question in Figure 8(b) contains 'same temperament as the smooth fox terrier' as a positive constraint, 'not from Serbia' as a negative constraint,

(a)

question	what is the unit of time not in the international system of units used to measure less than 1e-15 seconds?
gold	m.017k5b (Planck Time)
LLM I/O	[GPT-3.5] The unit of time not in the International System of Units used to measure less than 1e-15 seconds is the "femtosecond" (fs). [GPT-4o-mini] A unit of time not in the International System of Units (SI) that is used to measure intervals shorter than (1 times 10^{-15}) seconds is the "femtosecond" (fs), which is equal to (10^{-15}) seconds. For even shorter time scales, physicists often use the "attosecond" (as), which is equal to (10^{-18}) seconds. If you are looking for a unit specifically for times less than (1 times 10^{-15}) seconds, attoseconds would be appropriate.
LLM w/CoT	[GPT-3.5] The unit of time not in the International System of Units (SI) used to measure less than 1e-15 seconds is the femtosecond (fs). To break it down step by step: 1. The SI unit of time is the second. … Therefore, the femtosecond is the unit of time not in the SI system used to measure less than 1e-15 seconds. [GPT-4o-mini] To measure time intervals less than (1 times 10^{-15}) seconds, we can use the unit called the "femtosecond" (fs). 1. Definition of a femtosecond: … Conclusion: Therefore, the unit of time not in the International System of Units (SI) that can be used to measure less than (1 times 10^{-15}) seconds is the femtosecond (fs)."
KB-Coder	m.05v2dfy (Zeptosecond), m.0kbjk(Attosecond), m.05v2dg0(Yoctosecond), m.017k5b (Planck Time)
CUCKOO	m.017k5b (Planck Time)

(b)

question	How many breeds of dogs that have the same temperament as the smooth fox terrier are not from Serbia?
gold	6
LLM I/O	[GPT-3.5] There are 4 breeds of dogs with the same temperament as the smooth fox terrier that are not from Serbia. [GPT-4o-mini] The smooth fox terrier is known for its lively, energetic, and intelligent temperament. Breeds that share similar traits include: 1. Parson Russell Terrier … 2. Jack Russell Terrier … 3. Border Collie … 4. Miniature Schnauzer … 5. Airedale Terrier …. All these breeds are not from Serbia. The smooth fox terrier itself is originally from England, and the other breeds listed have roots in different regions. Therefore, there are at least five breeds that exhibit a similar temperament to the smooth fox terrier and are not from Serbia.
LLM w/CoT	[GPT-3.5] To determine the number of breeds of dogs with the same temperament as the smooth fox terrier that are not from Serbia, we can follow these steps: 1. Identify the breeds that have a similar temperament to the smooth fox terrier … This process involves researching and comparing the temperament of different dog breeds to that of the smooth fox terrier, as well as determining the countries of origin for each breed. [GPT-4o-mini] To determine how many dog breeds have a temperament similar to that of the Smooth Fox Terrier and are not from Serbia, we can follow these steps: 1. Identify the Temperament of … Therefore, the answer is that there are 7 breeds of dogs that have the same temperament as the Smooth Fox Terrier and are not from Serbia.
KB-Coder	0
CUCKOO	6

Figure 8: Case study on questions with negative-constraint.

and ‘How many’ as a calculation constraint. The correct answer is ‘6’. All baselines fail to provide a correct answer. GPT-4o-mini even offers an ambiguous count “at least five". To make matters worse, GPT-3.5-turbo with CoT stops at an explanatory outline and never derives a final answer. CUCKOO correctly answers ‘6’, demonstrating strong handling of multiple and varied constraints (RQ2).