Generating then Refining for Reliable Knowledge Base Question Answering

Jianqi Gao[1], Hang Yu[1], Jian Cao[2,*], Ranran Bu[2], Jinghua Tang[2], Nengjun Zhu[1], Yonggang Zhang[3]

[1]Shanghai University, [2]Shanghai Jiao Tong University
[3]The Hong Kong University of Science and Technology

{jianqi_gao,zhu_nj,yuhang}@shu.edu.cn
{cao-jian,buranran,Tang_Tony}@sjtu.edu.cn, zhangyg@ust.hk

Abstract

Knowledge Base Question Answering (KBQA) aims to retrieve accurate answers to natural language queries by retrieving and reasoning over large-scale structured knowledge bases (KBs). Advanced semantic parsing-based methods promoted by large language models (LLMs) demonstrate superior performance by transforming questions into structured queries, i.e., logical forms (LFs). However, LFs generated by LLMs could be non-executable due to the inherent semantic hallucination issue of LLMs and the complex graph retrieval characteristics of the KBQA task. To address this challenge, we propose a novel "generate-verify-refine" framework, termed Action-Reflection-Integrated KBQA (ARI-KBQA) for reliable LF generation. ARI-KBQA introduces a dual-module cooperative architecture: First, an action generator is trained to produce initial query paths based on a hop-by-hop reasoning strategy. Then a reflection verifier dynamically validates path feasibility by interacting with the KBs. Consequently, ARI-KBQA filters out invalid LFs and provides semantic correction feedback to the action generator for iteratively refining LFs. Evaluations on standard KBQA benchmarks show that the proposed ARI-KBQA significantly enhances model performance with a reduced search space, especially in complex multi-hop query scenarios. Our codes are publicly available at https://github.com/dependeonface/ARI-KBQA.

1 Introduction

Knowledge Base Question Answering (KBQA) plays a crucial role in the fields of natural language processing and information retrieval (Cui et al., 2019; Xiong et al., 2024). It aims to retrieve structured knowledge from large-scale knowledge bases (KB) to answer user queries (Miller et al., 2016). Knowledge Graphs (KGs) (Bollacker et al., 2008; Lehmann et al., 2015) store vast amounts of world knowledge in the form of triples, where nodes represent entities and edges represent relationships between entities, enabling multi-hop path reasoning based on user queries.

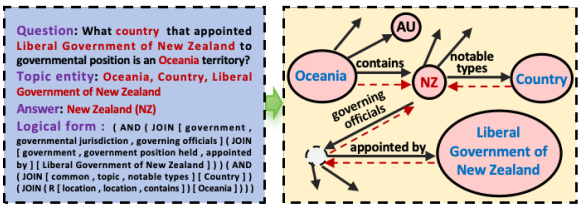

Figure 1: Illustration of logical form search: solid black and red dashed lines denote the relation directions in the KB and the search path, respectively.

Existing KBQA methods can be categorized into information retrieval (IR)-based methods (Sun et al., 2019; Saxena et al., 2022) and semantic parsing (SP)-based methods (Lan et al., 2019; Oguz et al., 2020; Huang et al., 2021). IR-based methods first retrieve information relevant to the question and then process it to generate answers, while SP-based methods parse the question into specific forms (e.g., logical forms) similar to natural language understanding and execute them using external executors (e.g., SPARQL executors) to obtain answers. Although IR-based methods are more intuitive and always able to obtain answers, the vast number of entities and relationships in KGs may introduce significant noise due to irrelevant reasoning paths, thereby greatly reducing its performance (Sui et al., 2024; Sun et al., 2024; Luo et al., 2024b). In contrast, SP-based methods have shown better performance on multiple benchmark datasets (Talmor and Berant, 2018a; Gu et al., 2021, 2022), but they struggle to effectively utilize knowledge from the KB, and the KBQA task has complex semantic matching and graph retrieval characteristics, which may cause

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 32070–32087

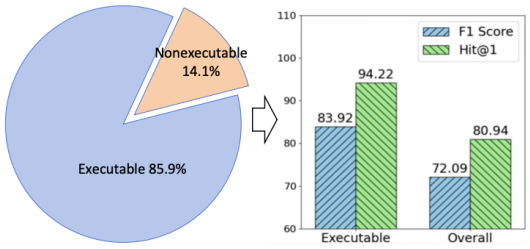

Figure 2: The impact of non-executable logical forms. Non-executable logical forms resulted in an overall performance decline on the CWQ dataset, with F1 decreasing by 11.83% and Hit@1 decreasing by 13.28%.

the model to generate non-executable LF, thus limiting performance improvement (Yu et al., 2023; Shu et al., 2022; Gao et al., 2025).

Recently, large language models (LLMs) have achieved remarkable success in various tasks (Chang et al., 2024; Wu et al., 2024; Jin et al., 2024), making their application to KBQA a promising direction. Studies have shown that using LLMs to generate LF for KBQA has yielded promising results (Yu et al., 2023; Luo et al., 2024a; Gao et al., 2025). Despite the remarkable performance of LLMs, they still suffer from inherent hallucination issues and exhibit notable limitations when handling complex reasoning tasks. For example, DecAF (Yu et al., 2023) and FM-KBQA (Gao et al., 2025) adopt a "retrieve-then-generate" approach, retrieving relevant information from the KB to assist LLMs in generating LF. However, similar to IR-based methods, this approach may introduce additional noise, limiting reasoning performance of LLMs. To reduce noise interference, ChatKBQA (Luo et al., 2024a) and TFS-KBQA (Wang and Qin, 2024) employ a "generate-then-retrieve" approach, directly generating LF and then retrieving answers from the KB, achieving state-of-the-art (SOTA) performance.

Nevertheless, as shown in Figure 1, LF can be viewed as semantic searches starting from multiple topic entities in the query, traversing relational paths in the KG to retrieve the final answer. The search direction (forward or backward) depends on the specific query. Obviously, KBQA involves complex semantic matching and graph retrieval processes. Furthermore, as illustrated in Figure 2, we fine-tuned the Llama-2-7B model on the CWQ dataset using a direct LF generation strategy. The experimental results indicate that when the LFs generated by the LLMs are executable, the predictions demonstrate high accuracy. However, due to the inherent hallucination problem of LLMs and the complexity of KBQA tasks, the generated LFs may sometimes be non-executable, leading to a significant degradation in the overall performance of SP-based KBQA methods.

To address the above challenges, we propose a novel "generate-verify-refine" framework, termed Action-Reflection-Integrated KBQA (ARI-KBQA). As illustrated in Figure 3, the ARI-KBQA framework innovatively proposes a dual-module interaction mechanism. This mechanism first employs an action generator that adopts a hop-by-hop reasoning strategy to generate preliminary search paths for the topic entity, thereby reducing the complexity of LF generation. Subsequently, the reflection verifier dynamically validates the feasibility of the current path and filters the optimal solution through real-time interaction with the KB, while simultaneously providing semantic correction feedback to the action generator to optimize the LF of the current hop. Through this iterative hop-by-hop optimization process, the action generator can produce faithful and refined LFs. Ultimately, the framework achieves accurate answer retrieval by integrating the optimized LFs from multiple rounds of refinement. Our contributions can be summarized as follows.

- We propose a novel learning framework, ARI-KBQA, which integrates iterative LF generation with dynamic verification and feedback mechanisms. This approach effectively mitigates the non-executable LFs resulting from semantic hallucinations in LLMs for complex KBQA tasks, enhancing the reliability and executability of the generated results.

- We propose a dual-module collaborative architecture: the action generator generates query paths based on hop-by-hop reasoning, and the reflection verifier verifies the path feasibility in real time by interacting with the Knowledge Base (KB), filtering invalid LFs and providing semantic correction feedback to form a closed-loop optimization.

- Results on the standard KBQA benchmark demonstrate that ARI-KBQA achieves improved performance with a smaller search space, particularly in complex multi-hop query scenarios.

2 Related Work

In this section, we provide a comprehensive introduction to KBQA, focusing on two key methodologies: Information Retrieval (IR)-based KBQA and Semantic Parsing (SP)-based KBQA.

IR-based KBQA typically consists of two stages: Information Retrieval (Sun et al., 2018; He et al., 2021) and Knowledge Reasoning (Miller et al., 2016; Sun et al., 2018, 2019; Zhang et al., 2021). During the Information Retrieval stage, the system extracts relations from the KG that are semantically relevant to the query using retrievers such as BM25 (Robertson et al., 2009), Dense Passage Retrieval (DPR) (Karpukhin et al., 2020), and SentenceBERT. During the Knowledge Reasoning stage, the system constructs structured reasoning paths by leveraging the retrieved relations and entities, and subsequently derives the final answer based on these paths (Sun et al., 2019; Jiang et al., 2022). In the domain of LLMs, ToG (Sun et al., 2024) leverages the reasoning capabilities of open-source LLMs to iteratively explore reasoning paths in the knowledge graph until the LLMs determine that it can answer the question based on the current path. PoG (Chen et al., 2024) enhances LLMs' reasoning over knowledge graphs through a self-correcting adaptive planning mechanism to enhance LLM-based reasoning over KG. While these methods ensure answer generation, the retrieval process often introduces irrelevant paths, compromising reasoning precision and overall system performance.

SP-based KBQA converts user queries into structured representations similar to natural language (e.g., S-expressions or logical forms) and transforms them into SPARQL queries to retrieve final answers from the KB (Zhang et al., 2023; Lan et al., 2019; Huang et al., 2021; Gu et al., 2021). For example, HGNet (Chen et al., 2022) proposes a two-stage SPARQL query generation method: first ranking candidate instances and then constructing query graphs through autoregressive decoding. In the domain of LLMs, DecAF (Yu et al., 2023) and Interactive-KBQA (Xiong et al., 2024) adopt a "retrieve-then-generate" strategy, retrieving relevant information from the KB to assist LLMs in generating LF. However, similar to IR-based methods, this approach may introduce additional noise, limiting the reasoning performance of LLMs. To mitigate noise interference, TFS-KBQA (Wang and Qin, 2024) and ChatKBQA (Luo et al., 2024a) employ a "generate-then-retrieve" approach, directly generating LF before retrieving answers from the KB, achieving SOTA results. However, due to the inherent semantic hallucination issue in LLMs and the complex graph retrieval characteristics of KBQA tasks, the LFs generated by LLMs may be non-executable.

3 Proposed Method

In this section, we present the fine-tuning process of the ARI-KBQA framework, as illustrated in Figure 3. The framework comprises two core components: an action generator and a reflection verifier. Our approach unfolds in two sequential stages: fine-tuning and reasoning. In the fine-tuning stage, the action generator and the reflection verifier are trained independently. In the reasoning stage, we introduce a dual-module architecture to facilitate knowledge reasoning. For definitions of key terminology used in the KBQA task, please refer to Appendix A for a detailed explanation.

3.1 Fine-Tuning on LLMs

3.1.1 Objective of Action Generator

The action generator employs a hop-by-hop generation mechanism to incrementally construct each component of the LF, where the generation of the j-th hop LF explicitly depends on the previously generated LF from the first $j-1$ hops.

Assume that the input question is q and the i-th topic entity involved in the question q is e_q^i, where $e_q^i \in E_q$ and E_q represents the set of all topic entities related to the question q. In the hop-wise LF generation process, the generation of the j-th hop LF depends not only on the current topic entity e_q^i, the question q, and the knowledge graph $\mathcal{G}$, but also explicitly on the LF l_{j-1}^i of the $(j-1)$-th hop. Besides, the LF l_{j-1}^i already encapsulates all information generated in the previous $j-1$ hops (as detailed in Figure 3). Accordingly, we formulate the hop-by-hop generation process of the action generator as an optimization problem, aiming to maximize the following probability,

$$P_\theta(L_p|q,\mathcal{G}) = \prod_{i=1}^{|E_q|} P_\theta(l_j^i|q, e_q^i, l_{j-1}^i, \mathcal{G}), \quad e_q^i \in E_q, l_j^i \in L_p, \tag{1}$$

where $l_j^i \in L_p$ is the ground truth LF of the i-th topic entity e_q^i at the j-th hop, θ denotes the parameters of LLMs, L_p is the set composed

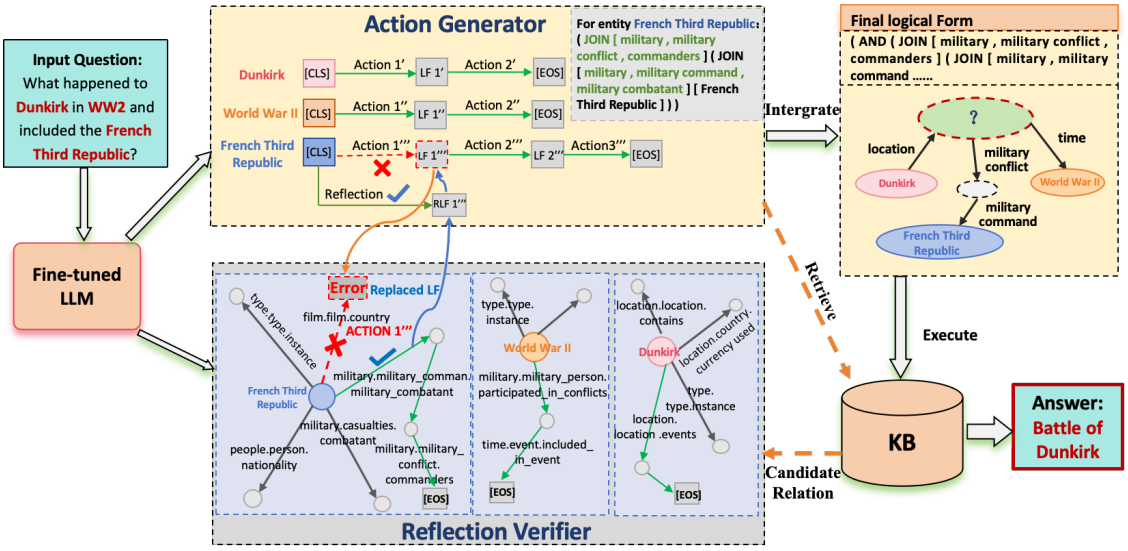

Figure 3: Overview of ARI-KBQA. ARI-KBQA uses a dual-module framework: an action generator produces query paths via hop-by-hop reasoning, while a reflection verifier validates them against the KB, filtering invalid LFs and refining outputs iteratively.

of the LFs of all topic entities. $|E_q|$ denotes the number of topic entities in the question q. The term $P_\theta(l_j^i|q, e_q^i, l_{j-1}^i, \mathcal{G})$ represents the conditional probability of generating the LF l_j^i, conditioned on the question q, i-th topic entity e_q^i, LF l_{j-1}^i, and KG $\mathcal{G}$. Additionally, it is important to note that we introduce the initial state token `[CLS]` and the termination state token `[EOS]` for the LF generation process targeting the topic entity e_q, which are used to denote the start and end of the generation, respectively.

After generating the preliminary logical form (PLF) for each topic entity, the action generator comprehensively analyzes all initial LFs and produces a complete and refined logical form (RLF) through integration, thereby retrieving answers from the KB. The integration process of the action generator can be formally described as follows,

$$P_\theta(l_r|q, \mathcal{G}) = P_\theta(l_r|L_{\text{PLF}})P_\theta(L_{\text{PLF}}|q, \mathcal{G}), \quad (2)$$

where l_r is the final ground truth RLF after considering all the PLFs $L_{\text{PLF}} \in L_p$ for each topic entity. Thus, the final objective function of the action generator is formulated as follows.

$$\mathcal{L}_a = \log P_\theta(L_p|q, \mathcal{G}) + \log P_\theta(l_r|q, \mathcal{G}). \quad (3)$$

where the above optimization process comprises two components: (1) hop-by-hop LF generation to produce PLF for each topic entity (Eqn. 1), and (2) the aggregation of PLFs across all topic entities to construct the RLF (Eqn. 2).

3.1.2 Objective of Reflection Verifier

After the action generator produces the LF for the i-th entity at the j-th hop, the reflection verifier receives the previous action (i.e., the $(j-1)$-th hop) from the action generator along with the retrieved candidate relations from the KB. Through a reflection mechanism, the verifier selects the correct relation from the candidate set and rectifies potential relation errors generated by the action generator at the j-th hop. This process ensures that the action generator is guided hop-by-hop to produce faithful retrieval paths.

Specifically, when the action generator produces the LF for the j-th hop, the reflection verifier determines the search direction based on its structure: forward relation search is performed when the current hop takes the form $(\texttt{JOIN}(\ R\ r)\ e)$, whereas backward relation search is executed for forms $(\texttt{JOIN}\ r\ e)$. To mitigate noise introduced by excessive candidate relations, we select the top-k candidate relations based on the relations generated by the action generator at the j-th hop, with the specific calculation as follows,

$$\Phi(r_c) = \underset{r_c \in C}{\text{argtopk}} \left(\sum_{i=1}^{n} p_i \cdot \text{sim}(r_i, r_c) \right), \quad (4)$$

where $\text{sim}(r_i, r_c)$ computes the similarity between generated relation r_i and candidate relation r_c, r_i is the relation in the i-th LF and p_i represents the probability of the i-th LF, argtopk selects the k candidates with highest weighted similarity scores.

After obtaining the top-K candidate relations, the reflection verifier takes the question q, the i-th topic entity e_i, the $(j-1)$-th hop LF l^i_{j-1} generated from e_i, and the candidate relation set C as input, then outputs the most plausible relation for the j-th hop. Accordingly, we formulate the reflection process as an optimization problem, aiming to maximize the following probability.

$$\mathcal{L}_r = \log P_\theta(r|q, e_i, l^i_{j-1}, C, \mathcal{G}). \quad (5)$$

where C is the top-K candidate relation set, r is the gold relation for the j-th hop, $P_\theta(r|q, e_i, l^i_{j-1}, C, \mathcal{G})$ is the models predicted probability for relation r.

3.1.3 Fine-tuning with LoRA

To achieve the objectives in Eqn. 3 and Eqn. 5, we use LoRA (Hu et al., 2021) to perform parameter-efficient fine-tuning (PEFT) on the training dataset. Given the training dataset of context-target pairs $\mathcal{Z} = \{(x_i, y_i)\}_{i=1,\ldots,N}$, where x_i and y_i are sequences of tokens. The instruction tuning objective function can be expressed as.

$$\max_{\Delta\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log(P_{\Phi+\Delta\Phi}(y_t|x, y_{<t})). \quad (6)$$

where Φ represents the pre-trained weight of LLMs, $\Delta\Phi$ is a smaller set of additional trainable parameters for PEFT with $\Delta\Phi \ll \Phi$, and $\theta = \Phi + \Delta\Phi$ denotes the task-specific fine-tuning process for generating LF after PEFT.

3.2 Dual-module Architecture for Knowledge Reasoning

Given the input question q, the topic entity e_i, and the initial start token `[CLS]`, the action generator generate LF hop-by-hop until the termination token `[EOS]` is encountered, at which point the LF generation process for the topic entity e_i concludes. During the generation process for topic entity e_i, assuming a beam size of b, the action generator produce n LFs at the j-th hop, typically satisfying $n \leq b$. To reduce the search space and minimize interaction costs with the reflection verifier, if there is an executable instance of the LF generated by the current hop, it is considered to be consistent with the facts and has a high accuracy (See Figure 2), and the next hop's LF is generated directly. Otherwise, the reflection verifier is invoked to replace the relations in the current hop. The above operation can be expressed as follows,

$$l^i_{j+1} = \begin{cases} \text{GenNext}(l^i_j) & \text{executable } l^i_j, \\ \text{GenNext}(\text{Replace}(l^i_j)) & \text{otherwise}, \end{cases} \quad (7)$$

where $\text{GenNext}(\cdot)$ denotes generate next $(j+1)$-hop LF. $\text{Replace}(\cdot)$ is to replace the relation of j-th hop LF via reflection verifier. Additionally, if all LFs generated at the j-th hop contain the `[EOS]` token, we normalize their probabilities using the softmax function as follows,

$$p(l^i_j|q, e_i, l^i_{j-1}, \mathcal{G}) = \frac{\exp(s(l^i_j))}{\sum_{l'\in L_{\text{retained}}} \exp(s(l'))}, \quad (8)$$

where $s(l^i_j)$ denotes the raw score of the LF l^i_j output by the action generator, and L_{retained} is the set of all retained LF (including `[EOS]`). If the overall probability of the LF is greater than the probability of the termination token `[EOS]`, (i.e., $1 - P(\texttt{[EOS]}) > P(\texttt{[EOS]})$), the LLMs continues to generate the LF for the next hop; otherwise, it terminates the LF generation process for the topic entity e_i. After generating the LF for each topic entity, we select the PLFs for each topic entity from the generated LF and guide the action generator to further generate the final RLF based on these PLFs. Then the overall answer retrieval process is summarized below.

$$\mathcal{A}_o = \begin{cases} \mathcal{A}_{ex} & \text{executable } l_{\text{RLF}} \\ \text{None} & \text{otherwise} \end{cases}. \quad (9)$$

If an executable RLF l_{RLF} exists, the answers $\mathcal{A}_o$ are directly derived from the execution results $\mathcal{A}_{ex}$. If none of the RLF l_{RLF} are executable, it will have no answer for the given question. More implementation details and prompt templates can be found in Appendix B.

4 Experiments

In this section, we provide a comprehensive overview of the experimental details, including the experimental setup and the main experimental results and analysis.

Datasets & evaluation metrics. We evaluate our method on four standard KBQA benchmarks. For Freebase-based evaluation, we adopt three widely-used datasets: WebQuestionSP (WQSP) (Yih et al., 2016), ComplexWebQuestions (CWQ) (Talmor and Berant, 2018b), and FreebaseQA (Jiang et al., 2019). For WQSP and CWQ, we follow prior work (Sun et al., 2018; Jiang et al., 2023)

Method	Backbone	WQSP			CWQ			FreebaseQA
		F1	Hit@1	ACC	F1	Hit@1	ACC	Hit@1
Non-LLMs methods								
Rigel (Sen et al., 2021)	-	-	73.3	-	-	48.7	-	-
NSM(He et al., 2021)	-	-	74.31	-	-	53.92	-	-
TIARA (Shu et al., 2022)	-	78.9	75.2	-	-	-	-	-
FILM (Verga et al., 2021)	-	-	54.7	-	-	-	-	63.3
RnG-KBQA (Ye et al., 2022)	-	75.6	71.1	-	-	-	-	-
UniK-QA (Oguz et al., 2020)	-	79.1	-	-	-	-	-	-
UniKGQA (Jiang et al., 2023)	-	72.2	77.2	-	49.4	51.2	-	-
HGNet (Chen et al., 2022)	-	76.6	76.9	70.7	68.5	68.9	57.8	-
Prompting-LLMs Only								
Zero-shot	GPT-4	59.71	62.32	-	37.93	42.71	-	-
Few-shot	GPT-4	62.71	68.75	-	43.70	51.52	-	-
CoT	GPT-4	65.37	72.11	-	44.76	53.51	-	-
Prompting-LLMs + KG								
ToG (Sun et al., 2024)	GPT-3.5	-	76.2	-	-	58.9	-	-
ToG (Sun et al., 2024)	GPT-4	-	82.6	-	-	72.5	-	-
FiDeLiS (Sui et al., 2024)	GPT-3.5	76.78	79.32	-	63.12	61.78	-	-
FiDeLiS (Sui et al., 2024)	GPT-4	78.32	84.39	-	64.32	71.47	-	-
PoG (Chen et al., 2024)	GPT-3.5	-	82.0	-	-	63.2	-	-
PoG (Chen et al., 2024)	GPT-4.0	-	87.3	-	-	75.0	-	-
Fine-tuning-LLMs + KG								
DecAF (Yu et al., 2023)	T5-11B	-	82.1	-	-	70.42	-	79.0
KD-CoT (Wang et al., 2023)	Llama-2-7B	50.2	73.7	-	-	50.5	-	-
ChatKBQA (Luo et al., 2024a)	Llama-2-7B/13B	83.5	86.4	-	**81.3**	86.0	-	85.77
TFS-KBQA (Wang and Qin, 2024)	Llama-2-13B	79.9	79.8	-	63.6	-	-	-
RoG (Luo et al., 2024b)	Llama-2-7B	69.81	83.15	-	56.17	61.39	-	-
RGR-KBQA (Feng and He, 2025)	Llama-3-8B	80.7	84.5	72.1	76.6	82.0	72.2	-
AMAR (Xu et al., 2025)	Llama-2-7B/13B	81.2	84.3	75.2	78.5	83.1	74.5	-
FM-KBQA (Gao et al., 2025)	Llama-2-7B	84.24	87.34	-	68.68	79.50	-	-
ARI-KBQA	Llama-2-7B	83.4	90.1	**79.21**	80.96	88.28	75.51	89.24
ARI-KBQA	Llama-3-8B	**86.31**	**91.65**	78.28	79.61	**89.63**	**76.59**	**89.77**

Table 1: Comparison results with baseline methods, where WebQSP, CWQ, and FreebaseQA are datasets constructed based on the Freebase knowledge graph.

Method	Backbone	LC-QuAD	
		Hits@1	F1
HGNet (Chen et al., 2022)	–	78.7	**78.1**
ChatKBQA (Luo et al., 2024a)	Llama-3-8B	68.1	56.9
ARI-KBQA	Llama-3-8B	**83.6**	77.5

Table 2: Comparison with baseline methods, where LC-QuAD are datasets constructed based on the DBpedia knowledge graph.

and use the same train/test splits to ensure fair comparison. Details on dataset statistics and the additional preprocessing steps introduced for training the action generator and reflection verifier are provided in Appendices C and D. Following (Yu et al., 2023; Luo et al., 2024b; Xu et al., 2025), we report Hits@1, F1, and Accuracy (Acc) as primary metrics, which respectively measure the precision of the top-ranked prediction, the coverage across all candidate answers, and strict exact-match performance. FreebaseQA, characterized by its complex linguistic structures, serves as an additional benchmark for robustness evaluation, on which we report Hits@1 following (Yu et al., 2023; Verga et al., 2021). To further assess the generalizability of ARI-KBQA on different KG, we conduct experiments on LC-QuAD, reporting Hits@1 and F1 following (Chen et al., 2022; Luo et al., 2024a).

Baselines. We compare our method with existing classical KBQA methods, including Non-LLMs methods, Prompting-LLMs Only methods, Prompting-LLMs + KG methods, and Fine-tuning-LLMs + KG methods. For **Non-LLMs methods**, we select NSM(He et al., 2021), Rigel (Sen et al., 2021), UniKGQA (Jiang et al., 2023), FILM (Verga et al., 2021) and UniK-QA (Oguz et al., 2020) as information retrieval baselines, and HGNet (Chen et al., 2022), RnG-KBQA (Ye et al., 2022) and TIARA(Shu et al., 2022) as semantic parsing baselines. For **Prompting-LLMs Only methods**, we show the results of zero-shot, few-shot, and CoT methods which are obtained in FiDeLis (Sui et al., 2024). For **Prompting-LLMs + KG methods**, we employ ToG (Sun et al., 2024), FiDeLis (Sui et al., 2024), and PoG (Chen et al., 2024) as baselines, while considering the results of GPT-3.5 and GPT-4. For **Fine-tuning-LLMs + KG methods**, we compare our approach with DecAF(Yu et al., 2023), KD-CoT(Wang et al., 2023), ChatKBQA (Luo et al., 2024a), TFS-KBQA (Wang and Qin, 2024), RoG(Luo et al., 2024b), FM-KBQA (Gao et al., 2025), AMAR (Xu et al., 2025), and RGR-KBQA (Feng and He, 2025). The results of these comparison methods are directly obtained from their respective papers. More details about the above baselines can be found in Appendix E.

Backbone LLMs. The open-source Llama-2-7B and Llama-3-8B models are employed (Luo et al., 2024a) as backbone to build the action generator and reflection verifier (Note: Both use the same base model). Additionally, to validate the robustness of the proposed method across different backbone LLMs, we further applied ARI-KBQA to two LLMs including Qwen-2.5-7B (Hui et al., 2024) and DeepSeek-LLM-7B (Bi et al., 2024).

Hyperparameters and environment. The model is fine-tuned using LoRA (Hu et al., 2021), and all experiments are conducted on a single NVIDIA A40 GPU. For the reasoning process of ARI-KBQA, the beam size of the action generator is empirically selected from $\{3, 5, 8\}$, while the beam size of the reflection verifier remains fixed at 3. For additional hyperparameter settings of fine-tuning using LoRA, please refer to Appendix F.

4.1 Results and Discussion

Overall results. To demonstrate the effectiveness of our approach, we conduct a comprehensive comparison between ARI-KBQA and four categories of baseline methods. The experimental results are summarized in Table 1 and Table 2, ARI-KBQA significantly outperforms all baselines on WQSP, CWQ, and FreebaseQA

Method	Llama-2-7B				Llama-3-8B			
	WQSP		CWQ		WQSP		CWQ	
	F1	Hits@1	F1	Hits@1	F1	Hits@1	F1	Hits@1
ARI-KBQA (DR)	74.93	82.06	72.09	80.94	75.31	82.51	72.11	81.17
ARI-KBQA (only AC)	81.78	87.31	79.51	86.96	81.68	86.82	80.75	86.9
ARI-KBQA	**83.4**	**90.1**	**80.96**	**88.28**	**85.77**	**90.53**	**81.92**	**88.05**

Table 3: Ablation results of ARI-KBQA on WQSP and CWQ datasets with beam size of 5. ARI-KBQA (DR) directly generates the final LF, while ARI-KBQA (only AC) generates LF using only the action generator.

Method	Beam Size	Llama-2-7B		Llama-3-8B	
		WQSP	CWQ	WQSP	CWQ
ARI-KBQA (DR)	3	14.60	18.03	13.57	17.85
ARI-KBQA (only AC)		8.23	11.48	7.61	11.13
ARI-KBQA		4.08	11.29	3.40	8.78
ARI-KBQA (DR)	5	11.26	14.10	10.41	13.88
ARI-KBQA (only AC)		6.57	9.87	6.31	10.21
ARI-KBQA		3.71	7.54	2.54	7.75
ARI-KBQA (DR)	8	9.22	11.01	7.95	10.89
ARI-KBQA (only AC)		5.51	5.99	4.95	6.78
ARI-KBQA		3.83	5.27	1.55	6.02

Table 4: The non-executable rate results of ARI-KBQA based on different backbone models.

datasets. Compared to prompting-LLMs + KG methods, ARI-KBQA achieves Hits@1 improvements of 4.35%/14.63% over PoG (GPT-4.0) and 5.25%/3.63% over ChatKBQA on WQSP/CWQ respectively. On FreebaseQA, it outperforms ChatKBQA by 4% and DecAF by 10.77%. Notably, while ChatKBQA and AMAR require Llama-2-13B and RGR-KBQA uses Llama-3-8B, ARI-KBQA achieves superior results using only Llama-2-7B. Among baselines, Non-LLM methods exhibit limited reasoning capabilities despite specialized architectures. Prompting-LLMs + KG approaches suffer from noise introduced by excessive path retrieval. ChatKBQA and AMAR directly generate LF but struggle with their semantic complexity. In contrast, ARI-KBQA employs hop-by-hop LF generation via its action generator, effectively reducing the complexity of LF generation, while its reflection verifier dynamically corrects errors through KB interaction, effectively mitigating hallucinations of LLMs and achieving SOTA performance.

To further validate the generalizability of the proposed method, we conduct additional experiments on the LC-QuAD dataset, a question answering benchmark built upon the DBpedia knowledge base. As shown in the results, ARI-KBQA consistently outperforms both conventional baseline methods and LLM-based baselines on this entirely different knowledge graph, demonstrating its strong generalization capability.

Model	WQSP(LLM Call/Time (s))	CWQ(LLM Call/Time (s))
ToG (Sun et al., 2024)	15.9 / 63.1	22.6 / 96.5
PoG (Chen et al., 2024)	9.0 / 16.8	12.3 / 23.3
FiDeLiS	10.7 / -	15.2 / -
ChatKBQA	1.0 / 3.8	1.0 / 11.7
ARI-KBQA (Ours)	1.9 / 3.4	3.7 / 11.4

Table 5: Efficiency comparison of different models on WQSP and CWQ datasets.

Model	WQSP			CWQ		
	F1	Hits@1	ACC	F1	Hits@1	ACC
Llama-2-7B	83.40	90.1	79.21	80.96	88.28	75.51
Llama-3-8B	86.31	91.65	78.28	79.61	89.63	76.59
Qwen-2.5-7B	82.85	89.11	75.19	78.93	88.62	75.43
Deepseek-LLM-7B	83.68	90.97	76.05	79.13	89.35	76.37

Table 6: Comparison with other LLMs on WQSP and CWQ datasets.

Ablation study. To evaluate the contribution of each component in ARI-KBQA, we conduct an ablation study (as shown in Table 3). Experimental results demonstrate that generating the LF only use the action generator outperforms direct generation. The model's performance further improves with the introduction of the reflection verifier, indicating that each component of ARI-KBQA contributes to enhancing its capability on the KBQA task. Notably, as illustrated in Figure 5 in Appendix G, ARI-KBQA maintains robust performance even under constrained search spaces for the action generator, which strongly validates the effectiveness of its dual-module architecture.

Comparison of non-executable rates. To evaluate ARI-KBQA's effectiveness in reducing the non-executable rate of LFs, we present results in Table 4. Compared to the direct generation ARI-KBQA(DR), ARI-KBQA significantly reduces the non-executable rate across all beam search sizes. This is primarily attributed to: (1) the hop-wise LF generation strategy that simplifies generation complexity, and (2) the dynamic "generate-verify-refine" mechanism that guides faithful reasoning and improves executable rate of LFs.

Efficiency analysis. To evaluate the time efficiency of ARI-KBQA, we conduct experiments as shown in Table 5. The results demonstrate that retrieval-based methods such as ToG, PoG, and FiDeLiS require multiple LLM calls per question, with ToG averaging 22.6 model calls and 96.5 seconds per question on CWQ. In contrast, ARI-KBQA achieves substantial performance gains over the SP-based ChatKBQA while maintaining competitive efficiency. Although ChatKBQA requires only a single LLM calls, its subsequent logical form correction introduces additional compu-

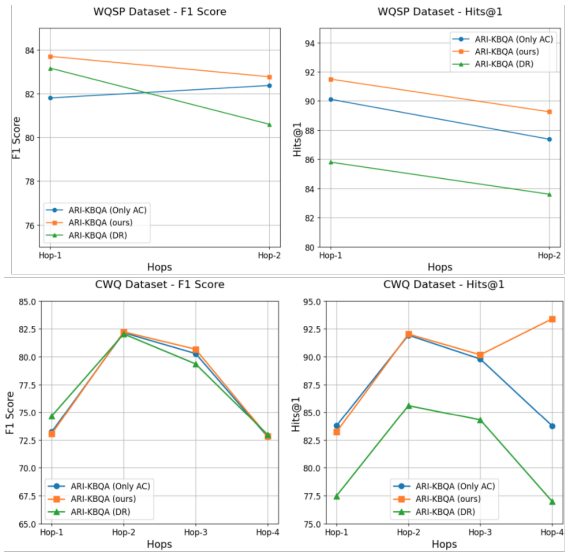

Figure 4: Model performance across different hops.

Model	WQSP(Error rate)	CWQ(Error rate)
Llama-2-7B	5.1%	9.2%
Llama-3-8B	4.3%	6.7%

Table 7: Error rate of reflection verifier.

tational overhead and error accumulation. ARI-KBQA instead adopts a dynamic "generate-verify-refine" strategy, wherein the action generator exhibits a low initial non-executable rate (6.31% on WQSP and 10.21% on CWQ; see Table 4), meaning the verifier remains idle in over 90% of reasoning steps, thereby substantially reducing LLM call and the associated computational time overhead..

Robustness analysis. To assess the robustness of ARI-KBQA across different LLMs, we conduct experiments as shown in Table 6. Beyond Llama-2-7B and Llama-3-8B, we evaluate additional models including Qwen-2.5-7B and DeepSeek-LLM-7B. ARI-KBQA demonstrates consistent performance across all four backbone LLMs, with optimal results on Llama-3-8B. These findings confirm the strong robustness and generalization capability of our method across diverse foundation models.

Model performance analysis under different hop counts. In KBQA tasks, higher reasoning hop counts indicate greater complexity. To evaluate ARI-KBQA's performance across varying hop counts, we conduct experiments on WQSP (max hops = 2) and CWQ (max hops = 4) datasets (Figure 4). Compared to direct generation (ARI-KBQA(DR)) and hop-wise generation using only action generator (ARI-KBQA(Only AC)), ARI-

Question:
What party with organization named Free Soil Party was Andrew Jackson in?
Ground Truth Logical Form:
(AND (JOIN [organization , organization , child] (JOIN [organization , organization relationship , child] [Free Soil Party])) (JOIN (R [government , political party tenure , party]) (JOIN (R [government , politician , party]) [Andrew Jackson])))
Ground Truth Answer: Democratic Party

GPT+CoT: Think step by step, I have a quest1on: (What party with organization named Free Soil Party was Andrew Jackson in?) and a seres of candidate reasoning path:
1: Andrew Jackson government.politician.party m.03l63p8 government.political_party_tenure.party Jacksonian Democratic Party **2:** Andrew Jackson government.politician.party m.03gjg09 government.political_party_tenure.party Democratic Party **3:**..... **Retrieve the answer from the candidates.**
Final Answers: Jacksonian Democratic Party

Direct Logical Form Generation: ARI-KBQA (DR)
Logical Form 1:
(AND (JOIN [government , political party , party ideology] [Free soil]) (JOIN (R [government , political party tenure , party]) (JOIN (R [government , politician , party]) [Andrew Jackson]))) Non-executable, discard
Final Answers: (None)

Hop-wise Logical Form Generation: ARI-KBQA (Only AC)
For 1st Entity Andrew Jackson: (JOIN (R [government , politician , party]) [Andrew Jackson]) **Executable** → (JOIN (R [government, political party tenure, party]) (JOIN (R [government, politician, party]) [Andrew Jackson])) **Executable** → [EOS]
For 2nd Entity Free Soil Party: (JOIN [organization, organization spin off, child company] [Free Soil Party]) Non-executable → (JOIN [government, political party, periods in office] (JOIN [organization, organization spin off, child company] [Free Soil Party])) Non-executable → [EOS]
Logical Form Integration:
(AND (JOIN [government, political party, periods in office] (JOIN [organization, organization spin off, child company] [Free Soil Party])) (JOIN (R [government, political party tenure, party]) (JOIN (R [government, politician, party]) [Andrew Jackson]))) Non-executable, discard
Final Answers: (None)

Proposed ARI-KBQA:
For 1st Entity Andrew Jackson:
(JOIN (R [government , politician , party]) [Andrew Jackson]) **Executable** → (JOIN (R [government , political party tenure , party]) (JOIN (R [government , politician , party]) [Andrew Jackson])) **Executable** → [EOS]
For 2nd Entity Free Soil Party:
(JOIN [organization , organization spin off , child company] [Free Soil Party]) Non-executable
Reflection and select from candidate relations:
[government.political_party_tenure.party, common.image.appears_in_topic_gallery, organization.organization_founder.organizations_founded, organization.organization_relationship.child . . .]
Reflected Substitution Relation: **organization.organization_relationship.child**
Replace and Continue: (JOIN [**organization.organization_relationship.child**] [Free Soil Party])**Executable** → (JOIN [organization , organization , child] (JOIN [organization , organization relationship , child] [Free Soil Party])) **Executable** → [EOS]
Logical Form Integration:
(AND (JOIN [organization , organization , child] (JOIN [organization , organization relationship , child] [Free Soil Party])) (JOIN (R [government , political party tenure , party]) (JOIN (R [government , politician , party]) [Andrew Jackson]))) **Executable**
Final Answers: Democratic Party

Table 8: Case study.

KBQA achieves superior performance across all hop counts, particularly for complex multi-hop scenarios. This improvement stems from: (1) hop-wise generation strategy that simplifies graph reasoning for LLMs, and (2) the dynamic "generate-verify-refine" mechanism that mitigates error accumulation and hallucinations during LF generation, thereby enhancing generation quality.

Refinement error rate of the reflection verifier. We evaluate the verifier using Llama-2-7B and Llama-3-8B, computing error rate as incorrect refinements over total refinements. Table 7 shows that Llama-3-8B achieves 95.7% accuracy on WQSP and 93.3% on CWQ, validating the effectiveness of the proposed reflection verifier. This is further corroborated by the case study in Table 8 and Table 9 (Appendix), where the reflection verifier successfully identifies and corrects erroneous relations generated by the action generator through KB interaction. For instance, the initially generated relation "organization.organization spin

off.child_company" is corrected to "organization.organization_relationship.child", after which the generator produces accurate relation paths in subsequent hops.

Case study. To demonstrate ARI-KBQA's effectiveness, we present a case study in Table 8 comparing it with GPT+CoT, ARI-KBQA(DR), and ARI-KBQA(Only AC). Only ARI-KBQA retrieves the correct answer. Two key observations emerge: (1) While ARI-KBQA(Only AC) fails to generate executable LFs, its hop-wise predictions outperform direct generation (ARI-KBQA(DR)), validating hop-wise generation's superiority. (2) When the action generator produces an invalid first-hop LF for entity `Free Soil Party`, ARI-KBQA dynamically corrects it via the reflection verifier, enabling valid second-hop LF generation, while ARI-KBQA(Only AC) propagates errors across hops. This confirms that ARI-KBQA mitigates LLM hallucinations and enhances both LF quality and answer accuracy.

Additional experimental analysis on various hyperparameter analysis and direction-based retrieval are detailed in Appendix G.

5 Conclusion

In this paper, we propose ARI-KBQA, a novel generate-verify-refine framework that enhances LF generation through iterative refinement. The framework comprises: (1) an action generator that constructs query paths via hop-wise reasoning, reducing generation complexity, and (2) a reflection verifier that validates LF executability against the KB, filtering invalid candidates and guiding faithful reasoning. Experimental results demonstrate that ARI-KBQA achieves superior performance with a compact search space, mitigates LLM hallucinations, and significantly improves LF executability, validating the effectiveness of our dual-module architecture.

Limitations

Since our proposed method operates within the generate-then-retrieve paradigm, a key direction for future research is to adapt it to smaller open-source LLMs with further reduced or eliminated training costs while maintaining strong generalization performance across diverse knowledge graphs. Additionally, although ARI-KBQA achieves state-of-the-art performance with lower computational overhead compared to baseline methods, further improving reasoning efficiency remains critical for real-time KBQA applications.

Acknowledgments

This research is supported by Voicecomm. It is also supported by the Smart Medical Special Research Fund of the Shanghai Municipal Health Commission (Grant No. 2025ZHYL003) and the Interdisciplinary Program of Shanghai Jiao Tong University (project number YG2024QNB05).

References

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and 1 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and 1 others. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yongrui Chen, Huiying Li, Guilin Qi, Tianxing Wu, and Tenggou Wang. 2022. Outlining and filling: hierarchical query graph generation for answering complex questions over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8343–8357.

Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. 2019. Kbqa: learning question answering over qa corpora and knowledge bases. *arXiv preprint arXiv:1903.02419*.

Tengfei Feng and Liang He. 2025. Rgr-kbqa: Generating logical forms for question answering using knowledge-graph-enhanced large language model. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3057–3070.

Jianqi Gao, Jian Cao, Ranran Bu, Nengjun Zhu, Wei Guan, and Hang Yu. 2025. Promoting knowledge base question answering by directing llms to generate task-relevant logical forms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23914–23922.

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, Vardaan Pahuja, Gong Cheng, and Yu Su. 2022. Knowledge base question answering: A semantic parsing perspective. *arXiv preprint arXiv:2209.04994*.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 553–561.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Xin Huang, Jung Jae Kim, and Bowei Zou. 2021. Unseen entity handling in complex question answering over knowledge base via language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 547–557.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Great truths are always simple: A rather simple knowledge encoder for enhancing the commonsense reasoning capacity of pre-trained models. *arXiv preprint arXiv:2205.01841*.

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

Kelvin Jiang, Dekun Wu, and Hui Jiang. 2019. Freebaseqa: A new factoid qa data set matching trivia-style question-answer pairs with freebase. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 318–323.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. 2024. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019. Knowledge base question answering with topic units.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and 1 others. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, Wei Lin, Yifan Zhu, and Anh Tuan Luu. 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, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 2039–2056. Association for Computational Linguistics.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024b. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Barlas Oguz, Xilun Chen, Vladimir Karpukhin, Stan Peshterliev, Dmytro Okhonko, Michael Schlichtkrull, Sonal Gupta, Yashar Mehdad, and Scott Yih. 2020. Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering. *arXiv preprint arXiv:2012.14610*.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. *arXiv preprint arXiv:2203.10321*.

Priyanka Sen, Amir Saffari, and Armin Oliya. 2021. Expanding end-to-end question answering on differentiable knowledge graphs with intersection. *arXiv preprint arXiv:2109.05808*.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. Tiara: Multi-grained retrieval for robust question answering over large knowledge bases. *arXiv preprint arXiv:2210.12925*.

Yuan Sui, Yufei He, Nian Liu, Xiaoxin He, Kun Wang, and Bryan Hooi. 2024. Fidelis: Faithful reasoning in large language model for knowledge graph question answering. *arXiv preprint arXiv:2405.13873*.

Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. *arXiv preprint arXiv:1904.09537*.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. *arXiv preprint arXiv:1809.00782*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. 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, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Alon Talmor and Jonathan Berant. 2018a. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*.

Alon Talmor and Jonathan Berant. 2018b. The web as a knowledge-base for answering complex questions.

Pat Verga, Haitian Sun, Livio Baldini Soares, and William Cohen. 2021. Adaptable and interpretable neural memoryover symbolic knowledge. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 3678–3691.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*.

Shouhui Wang and Biao Qin. 2024. No need for large-scale search: Exploring large language models in complex knowledge base question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 12288–12299.

Likang Wu, Zhi Zheng, Zhaopeng Qiu, Hao Wang, Hongchao Gu, Tingjia Shen, Chuan Qin, Chen Zhu, Hengshu Zhu, Qi Liu, and 1 others. 2024. A survey on large language models for recommendation. *World Wide Web*, 27(5):60.

Guanming Xiong, Junwei Bao, and Wen Zhao. 2024. Interactive-kbqa: Multi-turn interactions for knowledge base question answering with large language models. *arXiv preprint arXiv:2402.15131*.

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.

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, Chris Meek, Ming Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Lingxi Zhang, Jing Zhang, Yanling Wang, Shulin Cao, Xinmei Huang, Cuiping Li, Hong Chen, and Juanzi Li. 2023. Fc-kbqa: A fine-to-coarse composition framework for knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1002–1017.

Yonggang Zhang, Mingming Gong, Tongliang Liu, Gang Niu, Xinmei Tian, Bo Han, Bernhard Schölkopf, and Kun Zhang. 2021. Causaladv: Adversarial robustness through the lens of causality. *arXiv preprint arXiv:2106.06196*.

A Additional Terminology Details

Problem definition: KBQA retrieves answers from a Knowledge Graph (KG) to resolve user queries. Formally, given a natural language question q, a topic entity e mentioned in q, and a KG $\mathcal{G}$, the objective of KBQA is to design a mapping function f that leverages the relational information r embedded in $\mathcal{G}$ to accurately predict an answer a, where $a \in \mathcal{A}_q$ and $\mathcal{A}_q$ denotes the set of candidate answers for query q.

Non-executable logical form and question: Given a LF $l \in L$ associated with a question q, where L denotes the set of all possible LFs, if l is converted into a SPARQL query and yields no answers upon execution, then l is a **non-executable LF**. Furthermore, if none of the LFs in L can produce a valid answer, the question q is considered a **non-executable question**.

Logical forms (LF) is a structured intermediate semantic representation similar to natural language understanding. Its core function is to establish a bidirectional mapping bridge between natural language questions and underlying KB query languages (e.g., SPARQL). The transformation process from SPARQL to LF involves several operations, including projection and operator arrangement. Take single-hop retrieval as an example, given an entity e and a relation r, the SPARQL query for retrieving an entity x is represented as "$?x\ r\ e\ .$", which is transformed into the corresponding LF "*(JOIN r e)*". For commonly used operators, including the logical AND operator "AND", the argument maximum operator "ARGMAX", and the argument minimum operator "ARGMIN", etc.

Projection: projection retrieve entities or values from a KB by following relations (properties) from a given entity or set of entities. A projection can be either forward (from subject to object) or backward (from object to subject).

- For forward projection, given an entity e and a relation r, a forward projection retrieves all objects linked to e via r in the KB. Formally: $(\texttt{JOIN}(Rr)e) \rightarrow \{x|(e, r, ?x) \in \mathcal{K}\}$.
- For backward projection, given an entity e and a relation r, a backward projection retrieves all subjects that point to e via r. Formally: $(\texttt{JOIN}re) \rightarrow \{x|(?x, r, e) \in \mathcal{K}\}$.

Operators: Operators apply constraints or transformations to entity sets or values retrieved via projections. They include set operations (intersection: `AND`), comparative operations (literal constraints `GT`, `LT`, etc., temporal constraints: `TC`), and aggregations (extremum: `ARGMAX` and `ARGMIN`, counting: `COUNT`).

B Additional Implementation Details and Template

To provide a more intuitive illustration of the implementation process of ARI-KBQA, we present a detailed case study, including the prompt template and the input-output results for each step, as shown in Table 9 and Algorithm 1.

ARI-KBQA framework primarily consists of three key steps: hop-wise LF generation, reflection and replacement, and logical form integration. Their templates are as follows.

Hop-wise logical form generation

You're an expert in Knowledge Graph Question Answering.
Now I have a question: {},
the topic entity: {},
and the premise is: `(Previous Hop Content)`.
You need to perform the following task.
Task 1: You are asked to generate logical form based on the question, the topic entity, and the given premise.

Reflection and replacement

You are a KBQA validator, please perform Logical Form Validation & Correction Task.
Given the following input:
Question: {}
Topic entity: {}
Previous Logical Form: {}
Valid KB Relationships: {}
Please select the correct relation from valid KB relationships

Logical form integration

You're an expert in Knowledge Graph Question Answering.
Now I have a question: {},
the topic entities: `(All the Topic Entities)`,
and the premises: `(Executable PLF for Each Topic Entity)`.
You need to perform the following task.
Task 1: You are asked to generate the final LF based on the question, the topic entities, and the given premises.

Algorithm 1 The dual-module interactive reasoning process of ARI-KBQA.

Require: Question q, topic entities set $\mathcal{E}$, knowledge base $\mathcal{K}$, beam size b, maximum hops H, top-k value k
Ensure: Final answer $\mathcal{A}_o$

1: Identify topic entities: $E_q \leftarrow \{e_q^1, e_q^2, \ldots, e_q^{|E_q|}\}$
2: Initialize entity-specific LFs: $\mathcal{L}^i \leftarrow \emptyset$
3: **for** each entity $e_q^i \in E_q$ **do**
4: Initialize: $l_j^i \leftarrow$ [CLS], $j \leftarrow 1$ for each $e_q^i \in E_q$
5: **while** $j \leq H$ **do**
6: Generate beam-sized b LF candidates for hop j using Eq.(7)
7: Compute probabilities via Eq. 8
8: **if** [EOS] $\in L_{\text{retained}}$ and $P([\text{EOS}]) > 1-P([\text{EOS}])$ **then**
9: **break** {Terminate reasoning for current entity}
10: **end if**
11: Initialize executable_LFs $\leftarrow \emptyset$
12: **for** each l_j^i in candidates **do**
13: Execute LF: $\mathcal{A}_{ex} \leftarrow$ execute(convert(l_j^i))
14: **if** $\mathcal{A}_{ex} \neq \emptyset$ **then**
15: executable_LFs $\leftarrow$ executable_LFs $\cup \{l_j^i\}$
16: **end if**
17: **end for**
18: **if** executable_LFs $= \emptyset$ **then**
19: Select highest-probability LF: $l_j^i \leftarrow \arg\max P(l_j^i)$
20: Determine search direction via pattern matching:
21: Forward case: $(\texttt{JOIN}(R\ r)\ e)$
22: Backward case: $(\texttt{JOIN}\ r\ e)$
23: Retrieve candidate relation set: $\mathcal{C}$ by Eq.(4)
24: Select corrected relation r^* using reflection verifier
25: Replace relation in l_j^i with r^*
26: **end if**
{Continue with all executable LFs for next hop}
27: $j \leftarrow j + 1$
28: **end while**
29: $\mathcal{L}^i \leftarrow \mathcal{L}^i \cup \{l_j^i\}$
30: **end for**
31: Aggregate LFs using action generator: $l_{\text{RLF}} \leftarrow$ Aggregate($\{l^i\}_{i=1}^{|E_q|}$)
32: Convert l_{RLF} into SPARQL and execute: $\mathcal{A}_{ex} \leftarrow$ execute(convert(l_{RLF}))
33: **return** $\mathcal{A}_{\text{ex}}$

In the hop-wise logical form generation phase of ARI-KBQA, we guide the action generator to incrementally generate LF for each topic entity through the prompt. The generation process starts with the `[CLS]` token and proceeds to generate LF hop-by-hop until the `[EOS]` token is encountered (See line 3-10 in Algorithm 1). For the LF generated at the current hop, we convert it into a SPARQL query and execute it in the knowledge base (KB). If there exists at least one executable LF among the n LFs generated in the current hop when sorted by probability in descending order, the executable LF is retained, and the action generator proceeds to generate the next hop (See line 12-17 in Algorithm 1). Otherwise, it indicates that the LFs generated by the action generator in the current hop suffer from hallucination, and the reflection verifier is then invoked to correct the erroneous LFs.

In the reflection and replacement phase of ARI-KBQA, it employs a two-stage verification mechanism: (a) When the logical form (LF) generated in the current hop yields executable instances, it indicates that the LF aligns with the facts in the KB, triggering the direct generation of LFs for subsequent hops; (b) If none of the LFs in the current hop are executable, ARI-KBQA activates a reflection verifier to dynamically replace the relation within the current hop (See line 18-26 in Algorithm 1). The reflection verifier takes the LF of the previous hop, the topic entity e_j^i, the question q, and the candidate relation set C retrieved based on Eqn. 4 as input, it reflects to select correct relations from the candidate relation set C and replace erroneous ones. The action generator then generates the next-hop LF based on the corrected LF.

As shown in Table 9, it can be seen for the topic entity `Gábor Piroch`, the action generator initially generate an non-executable logical form `(JOIN [film, film, film art direction by] [Gábor Piroch])` during the first reasoning step. Through the optimization by the reflection verifier, the reflection verifier accurately identified the correct relation `film, performance, actor` from the candidate relation set and effectively replaced the erroneous relation `film, film, film art direction by` produced by the action generator. After this correction, the action generator successfully produce an executable logical form at the second reasoning step.

In the logical form integration phase of ARI-KBQA, we combine preliminary logical forms

(PLF) from the final executable sets of LF for each entity, the action generator synthesize each PLF to generate the final refined logical form (RLF). Subsequently, the RLF is converted into SPARQL queries and executed on the KB to obtain the final answer (See line 31-33 in Algorithm 1).

C Dataset Statistics

The statistical information of the three KBQA datasets, WQSP, CWQ, and FreebaseQA, is presented in Table 10.

The WQSP dataset (Yih et al., 2016) is primarily designed to evaluate the effectiveness of semantic parsing methods in KBQA tasks. It comprises 4,737 high-quality question-SPARQL query pairs, covering 34 logical form templates and 2,461 entities. The dataset defines 628 relations. It is a simple KBQA dataset containing one-hop and two-hop questions. WQSP uses Freebase (Bollacker et al., 2008) as its KB and is specifically designed for developing systems capable of processing natural language questions and answering them using structured data.

The CWQ dataset (Talmor and Berant, 2018b) is designed to answer complex questions requiring reasoning over multiple web snippets, with 20.75% of the questions involving more than two hops. CWQ is larger in scale, comprising 34,689 question-SPARQL query pairs supported by 174 logical form templates. It covers 11,422 entities and 845 relations. CWQ also utilizes Freebase as its KB and is specifically tailored for complex question-answering tasks that require interpreting and synthesizing information from multiple sources.

The FreebaseQA dataset (Jiang et al., 2019) is similar to the WebQuestionsSP (WQSP) dataset, primarily consisting of simple question-answer pairs with reasoning paths of no more than two hops. However, the questions in the FreebaseQA dataset exhibit more complex linguistic characteristics, such as more intricate syntactic structures. Therefore, we employ it as an auxiliary benchmark dataset to further validate the effectiveness of the proposed method.

The LC-QuAD dataset contains 5,000 sample pairs consisting of natural language questions and corresponding SPARQL queries, using DBpedia as the knowledge base. The dataset covers a variety of query types, including SELECT (entity retrieval) and COUNT (quantity aggregation)

Question: What was the name of the movie where Angelina Jolie directed Gabor Piroch?
Topic Entities: 'm.0f4vbz': Angelina Jolie, 'g.122bg1tz': Gábor Piroch

Ground Truth Logical Form:
(AND (JOIN [film, film, starring] (JOIN [film, performance, actor] [Gábor Piroch])) (JOIN (R [film, director, film]) [Angelina Jolie]))
Ground Truth Answer: In the Land of Blood and Honey

Proposed ARI-KBQA:
For 1st Entity Angelina Jolie:
Prompt: You're an expert in Knowledge Graph Question Answering. Now I have a question: `What was the name of the movie where Angelina Jolie directed Gabor Piroch?`, *the topic entity*: `Angelina Jolie`, *and the premise is*: `(Previous Hop Content)`. *You need to perform the following task. Task 1: You are asked to generate logical form based on the question, the topic entity, and the given premise.*
Hop 1: (JOIN (R [film, director, film]) [Angelina Jolie]) **Executable**
Hop 2: `[EOS]`

For 2nd Entity Gábor Piroch:
Prompt: You're an expert in Knowledge Graph Question Answering. Now I have a question: `What was the name of the movie where Angelina Jolie directed Gabor Piroch?`, *the topic entities*: `Gábor Piroch`, *and the Premises*: `(Executable PLF for Each Topic Entity)`. *You need to perform the following task. Task 1: You are asked to generate the final LF based on the question, the topic entities, and the given premises.*
Hop 1: (JOIN [film, film, film art direction by] [Gábor Piroch]) Non-executable, reflection
candidate relations: [film.performance.actor, type.type.instance, people.profession.people_with_this_profession, location.location.people_born_here]
Reflect relations: [film, performance, actor]
Reflection LF: (JOIN [film, performance, actor] [Gábor Piroch])**Executable**
Hop 2: (JOIN [film, film, starring] (JOIN [film, performance, actor] [Gábor Piroch])) **Executable**
Hop 3: `[EOS]`
Logical Form Assembly:
(AND (JOIN [film, film, starring] (JOIN [film, performance, actor] [Gábor Piroch])) (JOIN (R [film, director, film])[Angelina Jolie])) **Executable**
Final Answers: In the Land of Blood and Honey

Table 9: Case study of ARI-KBQA.

Datasets	Train	Valid	Test	Max hop	1 hop	2 hop	> 2 hop
WQSP	3098	-	1639	2	65.49 %	34.51%	0.00%
CWQ	27,639	3519	3,531	4	40.91 %	38.34%	20.75%
FreebaseQA	20358	3994	3996	2	66.43 %	33.57%	0.00%
LC-QuAD	4000	-	1000	2	60.54 %	39.46%	0.00%

Table 10: Statistics of the datasets used.

queries, making it one of the important benchmark datasets for KBQA tasks.

D Further Processing of the Dataset

ARI-KBQA introduces no extra annotation burden, as the training data for both the action generator and the reflection verifier are derived entirely from existing SPARQL annotations in WQSP, CWQ, and FreebaseQA, without requiring any additional labeled resources.

Training data acquisition for action generator: When processing SPARQL queries, the subgraph formed by the known SPARQL query is first decomposed into multiple paths starting from each topic entity. This decomposition process involves parsing the triples line by line and merging paths hop-by-hop in a nested manner based on the topic entities in the query. The goal is to extract path information derived from each topic entity (including the relations and directions at each hop) and convert these paths into LF. Specifically, each hop within a path is represented by the `JOIN` operation, while the intersection between different paths is represented by the `AND` operation. Additionally, operators such as `ARGMAX` and `GREATER THAN` can be introduced to further constrain the answers.

For example, given the following triples: `?y relation1 entity1`, `?y relation2 ?x`, and `?y relation3 entity2`. The first two triples, `?y relationship1 entity1` and `?y relationship2 ?x` can be merged into a single path starting from the topic entity `entity1`. Here, `relation1` serves as the relation through which the topic entity can retrieve the intermediate entity `?y` in a backward direction. This process is represented in LF as `(JOIN [relation1] [entity1])`. Subsequently, a forward retrieval is performed on `?y` via `entity2` to obtain the final answer represented by `?x`, which is nested as `(JOIN (R [relation2]) (JOIN [relation1] [entity1]))`. The third triple, `?y relation3 entity2`, forms another path starting from the topic entity `entity2`, and its LF can be derived in the same manner. The two paths intersect at the intermediate entity `?y`, and the final LF can be represented as `(JOIN (R [relation2]) (AND (JOIN [relation3] [entity2])) (JOIN [relation1] [entity1]))`.

Training data acquisition for reflection verifier: After obtaining the hop-wise training data for fine-tuning the action generator, we perform the following steps for the topic entity e_i mentioned in question q.

- Retrieve the tail entity of the $(j-1)$-th hops logical form;
- Search for candidate relations along the direction specified by the j-th hops logical form and select the top-k candidate relations based on Eqn. 4;
- Use the question q, topic entity e_i, the logical form l^i_{j-1} of the previous $j-1$ hops, and the top-k candidate relations of the j-th hop as input, with the correct relation of the j-th hop serving as the labeled data.

The detailed example is as follows.

Case study of fine-tuning data in reflection verifier

You are a KBQA validator, please perform Logical Form Validation & Correction Task. Given the following input:
Question: `What state is home to the university that is represented in sports by George Washington Colonials men's basketball?`
Topic entity: `George Washington Colonials men's basketball`
Previous Logical Form: `( JOIN ( R [ location, mailing address, state province region ] ) ( JOIN ( R [ organization, organization, headquarters ] ) ( JOIN [ education, educational institution, sports teams ] [ George Washington Colonials men's basketball ] ) ) )`
Valid KB Relationships: `location, statistical region, co2 emissions total|location, hud foreclosure area, ofheo price change|type, object, name|location, location, time zones|base, aliens, ufo sighting location, ufo sighting s .....`
Please select the correct relation from valid KB relationships.
Label output: `[ EOS ]`

E Details of the Comparative Methods

In this section, we provide a detailed description of the comparative methods used in the experiments, including Non-LLMs methods, Prompting-LLMs Only methods, Prompting-LLMs + KG methods, and Fine-tuning-LLMs + KG methods.

Non-LLMs methods. We select NSM(He et al., 2021), Rigel (Sen et al., 2021), UniKGQA (Jiang et al., 2023), and UniK-QA (Oguz et al., 2020) as information retrieval baselines, and HGNet (Chen et al., 2022), RnG-KBQA (Ye et al., 2022) and TIARA(Shu et al., 2022) as semantic parsing baselines.

- **Rigel** (Sen et al., 2021) proposes a technique to improve end-to-end question answering by leveraging differentiable KG and adding an intersection operation to handle multiple-entity questions.
- **NSM** (He et al., 2021) uses a sequential model to replicate the multi-hop reasoning process.
- **FILM** (Verga et al., 2021) integrate an interpretable, vectorized knowledge base as external fact memory into the language model, enabling knowledge modification through memory updates without retraining.
- **TIARA** (Shu et al., 2022) enhances question answering over s by focusing on relevant contexts and using constrained decoding to reduce errors.
- **RnG-KBQA** (Ye et al., 2022) generates possible LF based on input entities and utilizes a ranking and generation framework to select the most appropriate LF.
- **UniK-QA** (Oguz et al., 2020) integrates structured, unstructured, and semi-structured knowledge sources by flattening them into text and applying a unified retriever-reader model.
- **UniKGQA** (Jiang et al., 2023) combines retrieval and reasoning by using a unified architecture with semantic matching and information propagation modules for multi-hop KBQA.
- **HGNet** (Chen et al., 2022) proposes a hierarchical query graph generation method, consisting of an outlining stage for structural constraints, followed by a filling stage for instance selection.

Prompting-LLMs Only methods. We show the results of zero-shot, few-shot, and CoT methods, which are obtained in FiDeLis (Sui et al., 2024).

Prompting- LLMs + KG methods. We adopt ToG (Sun et al., 2024), FiDeLis (Sui et al., 2024), and PoG (Chen et al., 2024) as baselines, while considering the results of GPT-3.5 and GPT-4.

- **ToG** (Sun et al., 2024) leverages LLMs to iteratively explore reasoning paths on the KG until the question can be answered based on the current path.
- **FiDeLiS** (Sui et al., 2024) proposes a retrieval exploration method that incorporates both the logical and common sense reasoning of LLMs and the topological connectivity of KG into KBQA.
- **PoG** enhances LLMs' reasoning over knowledge graphs through a self-correcting adaptive planning mechanism, integrating dynamic guidance, memory updating, and reflective revision to significantly improve the efficiency and accuracy of KGQA.

Fine-tuning-LLMs + KG methods. We compare our approach with DecAF(Yu et al., 2023), KD-CoT(Wang et al., 2023), **ChatKBQA** (Luo et al., 2024a), **TFS-KBQA** (Wang and Qin, 2024), and RoG(Luo et al., 2024b).

- **DecAF** (Yu et al., 2023) combines LF and retrieved paths to generate answers using LLMs, achieving strong performance in KBQA tasks.
- **KD-CoT** (Wang et al., 2023) retrieves relevant knowledge from KG to generate faithful reasoning paths for LLMs.
- **ChatKBQA** (Luo et al., 2024a) adopts a generate-then-retrieve strategy, where LLMs are fine-tuned to directly generate a LF.
- **TFS-KBQA** (Wang and Qin, 2024) employs LLMs (e.g., Llama) to directly map natural language questions to structured knowledge representations via a three-step fine-tuning strategy.
- **RoG** (Luo et al., 2024b) uses a planning retrieval reasoning framework to combine LLMs with KG, enabling faithful and explainable reasoning.
- **RGR-KBQA** (Feng and He, 2025) proposes a Retrieve-Generate-Retrieve framework that

incorporates factual knowledge retrieval and unsupervised refinement to mitigate LLM hallucinations in KBQA.

- **AMAR** (Xu et al., 2025) proposes an Adaptive Multi-Aspect Retrieval framework that employs self-alignment and relevance gating modules to refine retrieved knowledge from KGs, effectively reducing noise and enhancing LLM reasoning accuracy.

- **FM-KBQA** (Gao et al., 2025) proposes a multi-task fine-tuning framework that retrieving relevant information from the KB to assist LLMs in generating LF.

F Fine-tuning Hyperparameter Settings

We fine-tuned Llama-2-7B and Llama-3-8B using the LoRA method, with a training batch size of 4 and a learning rate of 5e-5. The models were trained for 100 epochs on the WQSP dataset, 5 epochs on the CWQ dataset, 10 epochs on the FreebaseQA dataset, and 50 epochs on the LC-QuAD dataset. During the evaluation phase, to determine the optimal beam size parameter, we set the beam size for beam search to 3, 5, and 8 respectively.

G Additional Experimental Analysis

Hyperparameter analysis. To investigate the impact of different hyperparameters, we first conduct a comparative analysis of ARI-KBQA's performance on the WQSP and CWQ datasets under varying beam sizes, as shown in Figure 6. The results reveal that increasing the beam size of the action generator yields consistent improvements in Hits@1, while F1 scores exhibit a slight decline. This trade-off arises because larger beam sizes enable the generation of more diverse LFs, thereby enhancing answer coverage, but may simultaneously introduce noiseparticularly evident on the more complex CWQ dataset. Conversely, the beam size of the reflection verifier exhibits a relatively modest impact; ARI-KBQA maintains stable performance on both datasets as this parameter increases. Based on these observations, when employing Llama-2-7B as the backbone LLM, we set the action generator's beam size to 5 and the reflection verifier's beam size to 3 for both datasets.

We further investigate the impact of different top-K candidate relation thresholds on the performance of ARI-KBQA. As shown in Figure 7, ARI-KBQA demonstrates stable performance across various top-K thresholds. Considering both computational cost and model effectiveness, we set the threshold to 40 for all subsequent experiments.

Selection Strategy	WQSP		CWQ	
	F1	Hit@1	F1	Hit@1
Maximum Probability	85.72	90.95	79.26	89.48
Probabilistic Weighting	86.31	91.65	79.61	89.63

Table 11: Performance comparison of different relation selection strategies on the WQSP and CWQ datasets.

Model	WQSP	CWQ
Llama-2-7B	97.32%	94.16%
Llama-3-8B	98.80%	94.85%

Table 12: Accuracy of retrieval direction discrimination in logical forms generated by the action generator.

Furthermore, we investigate the impact of different relation selection strategies on model performance, including the Probabilistic Weighting Strategy (as shown in Eq.4) and the Maximum Probability Strategy ($\Phi(r_c) = \underset{r_c \in C}{\text{argtopk}} \left(\text{sim}(r_{i^*}, r_c)\right), i^* = \arg\max_{i \in 1,\ldots,n} p_i$). The Probabilistic Weighting Strategy selects the top-k candidate relations with the highest average similarity by considering the likelihood of each LF generated by the action generator. In contrast, the Maximum Probability Strategy selects the top-k relations with the highest similarity based solely on the most probable LF. As presented in Table 11, the results demonstrate that the Probabilistic Weighting Strategy slightly outperforms the Maximum Probability Strategy. This performance gap indicates that the Probabilistic Weighting Strategy achieves greater robustness by leveraging the entire probability distribution from the action generator. Instead of relying on a single, potentially noisy, maximum-probability prediction, it synthesizes information from all probable LFs, leading to more reliable and accurate relation selection.

Retrieval direction discriminant analysis of action generator. To demonstrate the effectiveness of using directions generated by the action generator for filtering candidate relations from the knowledge base. Specifically, performing forward relation search when the current hop takes the form (JOIN(R r) e) and backward relation search when it takes the form (JOIN r e). We

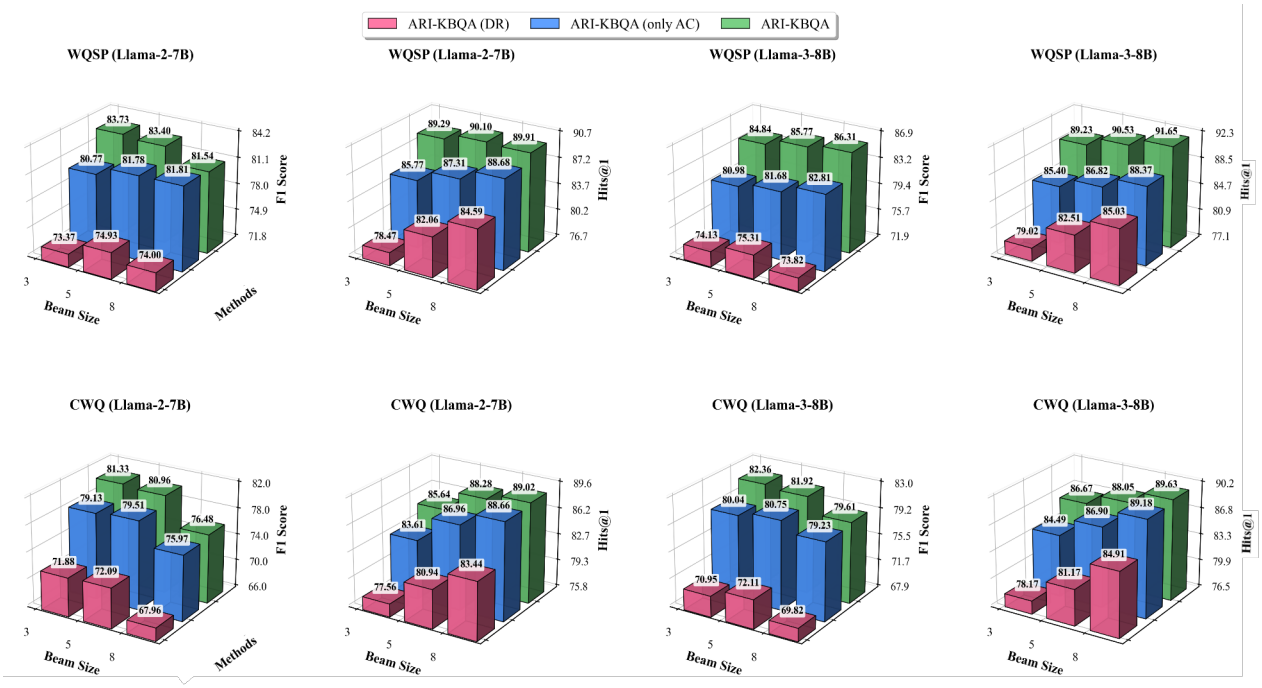

Figure 5: The results of ARI-KBQA with different beam sizes (action generator) and backbone LLMs on the WQSP and CWQ datasets. ARI-KBQA (DR) denotes directly generating the final LF, while ARI-KBQA (only AC) refers to generating the LF using only the action generator.

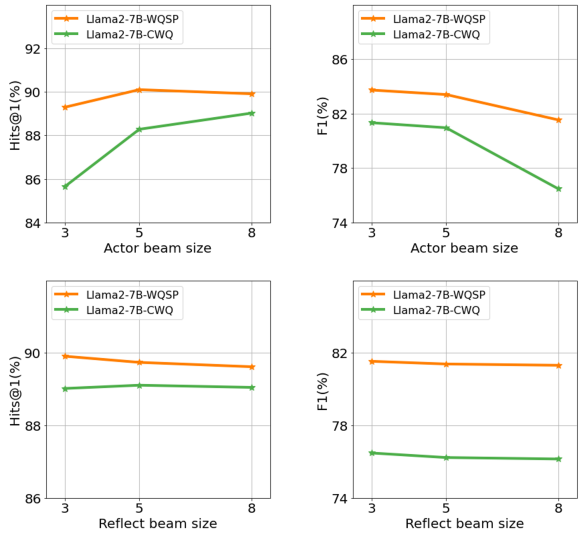

Figure 6: Hyperparameter analysis. Performance of ARI-KBQA, based on the Llama-2-7B, on the WQSP and CWQ datasets with beam sizes set to 3, 5, and 8, respectively.

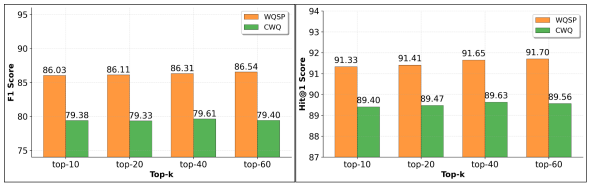

Figure 7: The impact of the top-k candidate relations threshold on model performance.

conduct experiments as shown in Table 12. The results indicate that the action generator achieves high accuracy in discriminating retrieval directions. For instance, when using Llama-3-8B as the base model, it achieves accuracies of 98.8% on WQSP and 94.85% on CWQ. Therefore, directly using the directions produced by the action generator as retrieval directions not only improves retrieval efficiency but also significantly mitigates the adverse impact of redundant and irrelevant relations on the reflection verifier.