A Multi-Agent Framework with Multi-Step Self-Correction for Knowledge Base Question Answering

WentaoQiu 30920241154570[1], KunYang 36920241153270[2]
XiangyuWu 30920241154577[2], KunpengLiao 36920241153230[2], ZhenyuShao 36920241153245[2]
Class of School of Informatics Xiamen University[1]
Class of Institute of Artificial Intelligence[2]
wtqiu@stu.xmu.edu.cn

Abstract

Knowledge Base Question Answering (KBQA) combines large language models (LLMs) with knowledge bases (KBs) to solve complex problems, demonstrating significant potential in addressing intricate queries. Traditional step-by-step reasoning and correction frameworks, while improving reliability and efficiency, often suffer from high computational overhead and vulnerability to errors in the initial stages, leading to increased correction costs. To tackle these challenges, we propose a novel KBQA framework that integrates subgraph-based reasoning with a multi-agent system to enhance accuracy, robustness, and efficiency.Our approach employs three specialized agents—CandidateFinder, PathFinder, and SelfCorrector—that collaboratively handle subtasks such as entity recognition, relation extraction, evidence retrieval, and iterative error correction. By leveraging subgraph information and a dynamic feedback mechanism, the framework mitigates logical deviations and error propagation while reducing computational demands. Experimental results demonstrate that this method significantly improves reasoning depth, recall, and answer consistency, effectively handling complex multi-hop queries across diverse KBQA tasks.

Introduction

Knowledge Base Question Answering (KBQA) has become a crucial field in natural language processing (NLP), aiming to interpret and respond to natural language questions by leveraging large structured knowledge bases (KBs) such as Wikidata, DBpedia, and Freebase. KBQA is widely applied in digital assistants, search engines, and specialized knowledge retrieval systems across domains like healthcare and finance. However, traditional KBQA methods, which rely on predefined question templates and rule-based approaches, often struggle with the complexity and variability of natural language queries. Early frameworks such as KBQA successfully demonstrated that mapping queries to templates could enhance retrieval efficiency and accuracy in structured KBs, although these methods faced challenges with adaptability in handling dynamically evolving and diverse question forms (Cui et al. 2019).

The rise of large language models (LLMs) has led to more sophisticated KBQA frameworks capable of contextual learning and adapting to new queries with minimal training data. Few-shot In-Context Learning (ICL) frameworks allow KBQA models to generate logical forms on the fly by learning from a limited set of examples, making them suitable for low-data scenarios. Despite their success, these frameworks are limited by their context length and may struggle with longer or more complex queries (Li et al. 2023). To further enhance KBQA frameworks' effectiveness, AgentBench was developed as a comprehensive evaluation tool, assessing LLMs' multi-turn reasoning and decision-making abilities across diverse real-world tasks. Findings from AgentBench highlight the strengths of LLMs in practical reasoning tasks, yet reveal significant gaps in multi-turn task management and sustained accuracy, pointing to areas where KBQA models require refinement (Liu et al. 2023).

In addition, Few-shot Transfer Learning introduces a hybrid approach that integrates supervised models with in-context learning, allowing KBQA systems to perform cross-domain question answering. This framework demonstrates strong performance in domains with sparse labeled data but suffers from increased computational requirements due to its complex architecture (Patidar et al. 2024). FlexKBQA, meanwhile, mitigates the data scarcity issue by synthesizing training data, thus reducing reliance on labeled datasets and achieving commendable results in zero-shot settings, although the model's accuracy can be influenced by discrepancies between synthetic and real data distributions (Li et al. 2024).

For more adaptable KBQA, KS-LLM combines in-context learning with knowledge selection mechanisms that employ relevant supporting documents to aid logical form generation. This design improves accuracy by drawing on external evidence, yet it still depends heavily on initial entity selection, which may not always align with the desired context (Zheng et al. 2024). Addressing this, QueryAgent implements an environmental feedback mechanism for iterative self-correction, allowing for dynamic response adjustments. This method significantly reduces logical errors in multi-hop reasoning, although the process introduces latency in response times due to multiple correction stages (Huang et al. 2024).

Despite these advancements, KBQA still faces notable challenges. Many frameworks are hampered by their inabil-

ity to maintain high efficiency in computationally intensive multi-role or iterative systems. Additionally, issues with error propagation remain pervasive, particularly in multi-hop reasoning tasks where each inference step builds upon the last. Furthermore, most current models lack a flexible feedback mechanism to dynamically adjust to logical deviations, making them less robust in complex real-world applications.

To overcome the challenges in existing KBQA methods, we propose a novel framework based on a **Multi-Agent System (MAS)**. This framework leverages a modular architecture where multiple specialized agents—**CandidateFinder**, **PathFinder**, and **SelfCorrector**—collaboratively handle distinct subtasks, including entity recognition, relation extraction, and evidence retrieval. Each agent focuses on a specific aspect of the KBQA process, enhancing reasoning accuracy and reducing error rates. By incorporating **dynamic feedback mechanisms** and a **multi-step self-correction process**, the MAS framework addresses logical inconsistencies, minimizes error propagation, and ensures robust performance in complex multi-hop reasoning scenarios. Furthermore, the use of **subgraph-based reasoning** enhances efficiency, enabling the framework to handle diverse and intricate queries with lower computational costs.

The key contributions of our work are as follows:

1. **Multi-Agent KBQA Framework**: We introduce a novel multi-agent KBQA framework that combines *CandidateFinder*, *PathFinder*, and *SelfCorrector*. These agents collaboratively address the challenges of multi-hop reasoning, ensuring a structured and scalable approach to KBQA.
2. **Subgraph-Driven Reasoning**: By utilizing a subgraph-based reasoning strategy, the framework improves query efficiency and answer precision. This approach effectively narrows the search space, enabling the model to focus on relevant entities and paths.
3. **Dynamic Feedback Mechanism**: The integration of dynamic feedback allows the agents to adjust their reasoning strategies dynamically, reducing logical errors and improving overall consistency.

Preliminary

Definition 1: Knowledge Base (KB). A Knowledge Base (KB) is defined as $K = \{(s, r, o) \mid s \in E, r \in R, o \in E \cup L\}$, where s represents an entity, r represents a relation, and o can be either an entity or a literal(Luo et al. 2023a). The set E contains all entities, while the set R contains all relations. Each entity $e \in E$ is identified by a unique ID, for example, e.id $=$ "$m.123abc$", and can be queried to retrieve its label, e.g., e.label $=$ "$Albert\ Einstein$". Each relation $r \in R$ has a hierarchical label, for instance, $r =$ "$person.education.university$". This structure allows entities and their relationships to be queried and analyzed.

Definition 2: Logical Form. A logical form is a formal representation of a natural language question, typically structured to perform operations on a KB(Luo et al. 2023a). For instance, in the S-expression format, a logical form usually includes projection operations and other logical operators. A projection refers to a one-step query over a triple (s, r, o). For example, querying for an object given a relation is represented as $(?, r, o)$, which is denoted as (JOIN r o). Conversely, querying for a subject given a relation is represented as $(s, r, ?)$, denoted as (JOIN (R r) s). Other common operators such as `AND`, `COUNT`, and `ARGMAX` are described in more detail in Appendix A.

Definition 3: LLM-based Agents. Large language model-based agents have gained increasing attention due to the remarkable emergent capabilities of LLMs, leading researchers to leverage these models for building AI agents (Xi et al. 2023). The framework proposed by (Xi et al. 2023) consists of three components: brain, perception, and action, adaptable to various applications. The brain, powered by an LLM, serves as the core of the agent's intelligence, handling memory storage, information processing, decision-making, reasoning, and planning. The perception module broadens the agent's input to a multimodal space, incorporating sensory modalities like sound and visuals, which enhances environmental awareness. Finally, the action module extends the agent's ability to generate outputs, perform actions, and use tools, enabling it to interact with and influence its environment.

Semantic Parsing (SP) methods. SP methods learn to transform the given question Q into a logical form $F = Sp(Q)$, where $Sp(.)$ is the semantic parsing function(Sun et al. 2020). The logical form F is then converted into an equivalent SPARQL query $q = \text{Convert}(F)$, using the conversion function Convert$(.)$. Finally, the query q is executed against the knowledge base K, yielding the final set of answers $A = \text{Execute}(q \mid K)$, where Execute$(.)$ represents the query execution function.

Information Retrieval (IR) methods. In contrast to semantic parsing methods, Information Retrieval (IR) methods focus on extracting relevant knowledge graph (KG) information directly, without requiring the transformation of questions into logical forms(Zhang et al. 2022). Given a natural language question Q, IR methods retrieve a KG subgraph G_Q that is most relevant to the query. This is achieved through a retrieval function $R(Q, K)$, where K is the entire knowledge graph, and $G_Q \subset K$. The retrieved subgraph G_Q serves as the input for reasoning models to infer the correct answer.

Formally, IR methods involve two steps:

1. **Retrieval:** Identify $G_Q = R(Q, K)$, where $R(.)$ selects entities, relations, and paths in the KG based on their relevance to Q.
2. **Reasoning:** Apply reasoning models f to predict the answer $A = f(Q, G_Q)$.

Related Work

In the domain of KBQA, advancements in Large Language Models (LLMs) have significantly enhanced the ability to parse complex queries. Many KBQA systems adopt generation-retrieval frameworks, where logical form generation and retrieval are tightly integrated to improve accuracy and interpretability. For example, FlexKBQA and ChatKBQA employ this approach by combining candidate response generation with knowledge base retrieval, enabling more precise question interpretation and answer generation (Li et al. 2024; Luo et al. 2023a). Similarly, Few-shot In-context Learning and Code-Style In-Context Learning optimize logical form generation, albeit using different techniques. Few-shot In-context Learning leverages small datasets to iteratively refine logical forms, while Code-Style In-Context Learning translates these forms into code to reduce syntax errors and improve learning efficiency, though both face limitations in handling complex, multi-hop reasoning tasks (Li et al. 2023; Nie et al. 2024).

Recent innovations address challenges in handling intricate queries. The Triad framework employs multi-agent collaboration to decompose question-answering tasks into stages, improving multi-hop reasoning by dividing the workload among specialized agents, albeit with increased complexity (Zong et al. 2024). Similarly, the "Make a Choice" framework introduces in-context learning tailored for decision-making in constrained-choice scenarios, enhancing adaptability but remaining limited by parsing accuracy (Tan et al. 2023). These approaches exemplify the growing emphasis on task-specific frameworks to address complex KBQA challenges.

Retrieval-focused advancements have also played a crucial role. KS-LLM combines triple-based knowledge with evidence retrieval to enhance QA performance while mitigating hallucinations, though noise remains a challenge when retrieval fails (Zheng et al. 2024). Few-shot Transfer Learning demonstrates efficacy in low-data scenarios by combining supervised models with LLM-based re-ranking to refine logical form generation, though frequent API calls significantly increase computational costs (Patidar et al. 2024). SEMQA integrates multi-source information for answer verification, but its reliance on diverse data sources amplifies noise and hallucination issues (Schuster et al. 2023).

Subgraph Retriever (SR) and ReaRev have pushed the boundaries of reasoning efficiency. SR decouples subgraph retrieval from reasoning, reducing noise and excelling in multi-hop QA, though it struggles with incomplete KGs and complex queries (Zhang et al. 2022). ReaRev builds on SR by incorporating adaptive reasoning mechanisms that dynamically adjust reasoning paths using Breadth-First Search (BFS) strategies and graph-aware updates, achieving high precision in incomplete KG scenarios (Mavromatis and Karypis 2024).

For large-scale KB reasoning, Reasoning on Graphs (RoG) integrates LLMs' natural language capabilities with KG structural information through a plan-retrieve-reason framework. RoG enhances reasoning by generating relation paths, addressing hallucinations and knowledge gaps, and stands as the current state-of-the-art (SOTA) method due to its balance of efficiency and interpretability (Luo et al. 2023b). Similarly, GNN-RAG combines Graph Neural Networks (GNNs) with Retrieval-Augmented Generation (RAG) to extract candidate paths from dense subgraphs and complements this with LLM reasoning for multi-hop tasks. Despite its effectiveness, GNN-RAG's computational demands limit its scalability (Mavromatis and Karypis 2024).

Among these, RoG is recognized as the leading SOTA method in KBQA, offering strong interpretability and performance while leaving room for future advancements to optimize computational efficiency.

Methodology

Overview

Our overall framework, as shown in Figure 1, consists of three agents—CandidateFinder, PathFinder, and SelfCorrector—and three supporting modules—SubgraphRetriever, ToolFunctions, and ThoughtBeforeThought. The SubgraphRetriever module facilitates the extraction and refinement of relevant entities and relations from the knowledge base, reducing the search space and eliminating the need for SPARQL query generation. Within this module, ToolFunctions such as get_relations(entity) and get_neighbors(entity, relation) are employed to retrieve relationships and neighbors of entities, forming the basis of the subgraph. The ThoughtBeforeThought module pre-processes the question, assisting the agents in identifying initial reasoning steps and prioritizing relevant entities. The three agents work collaboratively: CandidateFinder identifies potential answer candidates by analyzing the subgraph, PathFinder explores reasoning paths linking the question to these candidates, and SelfCorrector iteratively validates and refines intermediate results to ensure reasoning accuracy and robustness. This integrated framework combines the strengths of efficient subgraph retrieval, advanced reasoning through fine-tuned Large Language Models (LLMs), and error correction via the SelfCorrector, resulting in a robust pipeline capable of delivering precise answers for multi-hop KBQA tasks.

SubgraphRetriever

Our methodology leverages the SubgraphRetriever proposed by Zhang et al. (Zhang et al. 2022), which is designed to retrieve relevant subgraphs for multi-hop KBQA. This approach decouples the subgraph retrieval process from reasoning, enabling more accurate and efficient reasoning over large knowledge bases.

Path Expansion and Subgraph Induction The SubgraphRetriever retrieves subgraphs through a two-step process:

- **Path Expansion**: Starting from the topic entity, relevant paths are expanded iteratively by scoring relations based on their relevance to the question using embedding-based similarity measures. The top-K paths are selected using a beam search.
- **Subgraph Induction**: The expanded paths are merged into a unified subgraph by combining overlapping enti-

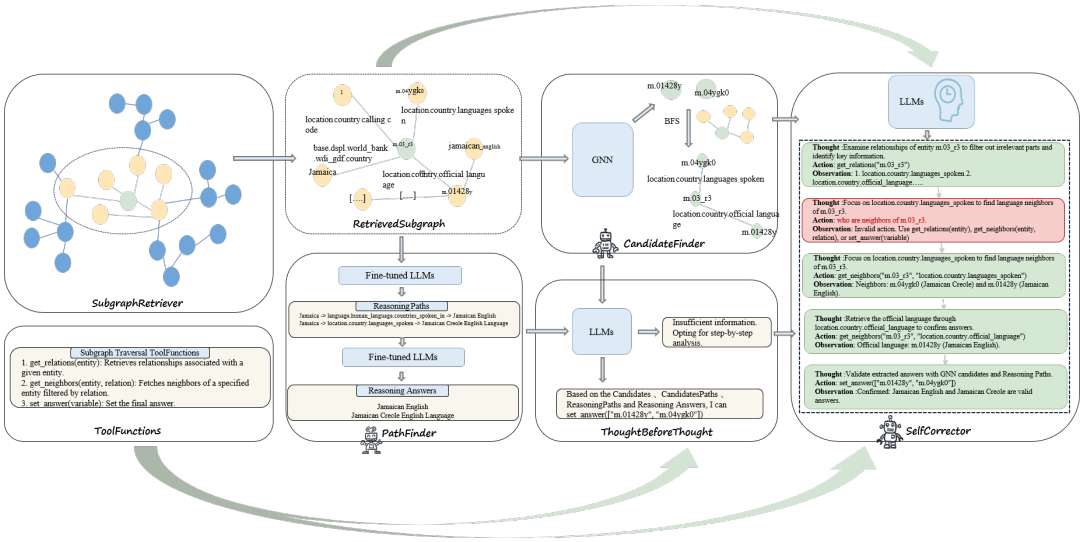

Figure 1: The figure illustrates the proposed KBQA methodology framework, which integrates subgraph traversal, Graph Neural Networks (GNNs), and fine-tuned Large Language Models (LLMs). Subgraph traversal functions extract relationships and neighbors from the knowledge base, forming the initial subgraph. The GNN processes the subgraph to capture structural dependencies, while the LLMs perform iterative reasoning to identify reasoning paths and validate answers. This pipeline ensures efficient and accurate multi-hop question answering by combining knowledge retrieval, graph-based reasoning, and language model capabilities.

ties and retaining relations connecting the topic entities to candidate answers.

Training Strategies The retriever utilizes a combination of training strategies, including weakly supervised pre-training based on shortest paths, unsupervised pre-training with pseudo labels, and end-to-end fine-tuning that incorporates feedback from the reasoning process.

Advantages By retrieving smaller and more focused subgraphs, this method reduces reasoning complexity while maintaining high answer coverage. It has been shown to achieve state-of-the-art results in embedding-based KBQA models (Zhang et al. 2022).

In this work, we adopt SubgraphRetriever (Zhang et al. 2022) as a core component of our KBQA pipeline to ensure efficient and accurate subgraph retrieval for multi-hop reasoning. By leveraging the retrieved subgraphs as the knowledge source for question answering, our approach eliminates the need for generating SPARQL queries, simplifying the KBQA process. This information retrieval-based method significantly reduces the complexity of the reasoning task while maintaining high answer coverage and accuracy.

CandidateFinder: Iterative Adaptive Reasoning Candidates Finding Agent

Inspired by the ReaRev approach (Mavromatis and Karypis 2022), our CandidateFinder leverages adaptive reasoning techniques with GNNs to identify candidate answer entities for multi-hop KBQA. The module incorporates adaptive instruction decoding and execution to iteratively refine the reasoning process and accurately locate potential answers.

Instruction Decoding and Execution CandidateFinder begins by decomposing the question into a set of dense vector representations, termed instructions $\{i_k\}_{k=1}^{K}$. These instructions are dynamically updated using KG-aware information derived from iterative reasoning steps. The reasoning process emulates a breadth-first search (BFS) strategy by treating instructions as a set and determining their execution order adaptively. At each GNN layer l, the node representations are updated as:

$$h_v^{(l)} = \psi\left(h_v^{(l-1)}, \phi\left(\{m_{v'v}^{(l)} : v' \in N(v) | i_k\}\right)\right), \quad (1)$$

where $h_v^{(l)}$ represents the node embedding at layer l, ϕ aggregates messages from neighboring nodes v', and ψ fuses these representations with previous node embeddings. This mechanism ensures that only question-relevant facts are aggregated.

Adaptive Updates To improve reasoning over complex questions, CandidateFinder employs an adaptive mechanism that iteratively updates instructions and node embeddings. At each reasoning stage t, the final node representations H_{out} are used to adjust the initial instructions $\{i_k\}_{k=1}^{K}$ and seed representations, ensuring alignment with the KG semantics:

$$i_k^{(t+1)} = (1 - g_k) \odot i_k^{(t)} + g_k \odot W_q \left(i_k^{(t)} \| h_e \| \cdots\right), \quad (2)$$

where h_e represents the KG-aware information from seed entities, g_k is a gating mechanism, and W_q is a learnable transformation matrix.

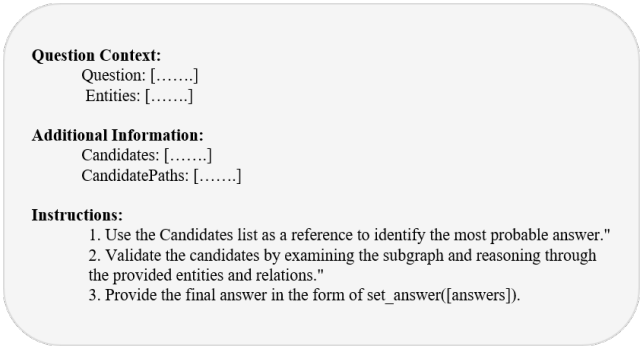

Figure 2: The prompt template incorporating candidate entities and candidate paths provided by CandidateFinder.

Candidate Entities The final node representations are processed to classify nodes as candidate answers or non-answers. This is achieved through a softmax operation over the probability scores of all nodes in the subgraph. Nodes with the highest probabilities are ranked as the top candidates.

Candidate Paths Starting from the question entity and candidate entities mentioned in the question, we apply the BFS algorithm to find all shortest paths in the subgraph connecting the question entity to the candidate entities.

PathFinder: LLM-Driven Reasoning Paths Generation and Utilization Agent

To facilitate faithful and interpretable reasoning for KBQA, our framework integrates a PathFinder agent inspired by the Reasoning on Graphs (RoG) method (Luo et al. 2023b). PathFinder leverages LLMs to generate reasoning plans (relation paths) and retrieve reasoning paths from knowledge base, ultimately synthesizing these paths to derive accurate and interpretable answers.

Relation Path Generation PathFinder initiates reasoning by prompting the LLM to generate relation paths grounded in KG structures. A relation path consists of a sequence of semantic relations between entities, serving as a faithful reasoning plan. For example, to answer the question, "Who is the child of Alice?", a generated relation path might be:

$$z = \texttt{marry\_to} \rightarrow \texttt{father\_of}.$$

This path corresponds to the plan: first, identify the spouse of Alice, and second, identify the child of the spouse.

Retrieval of Reasoning Paths Using the relation paths as a guide, PathFinder employs a constrained breadth-first search (BFS) on the KG to retrieve corresponding reasoning paths. These reasoning paths are grounded instances of the relation path, linking specific entities. For example, based on the relation path above, a reasoning path could be:

$$w_z = \texttt{Alice} \xrightarrow{\texttt{marry\_to}} \texttt{Bob} \xrightarrow{\texttt{father\_of}} \texttt{Charlie}.$$

Reasoning with Retrieved Paths The retrieved reasoning paths are then processed by the LLM to generate the final answer. The LLM reasons over multiple paths, identifying the most relevant ones and synthesizing the results into an interpretable answer. For instance, the LLM would conclude from the example above that "Charlie" is the child of Alice.

Optimization of Reasoning PathFinder optimizes reasoning through two key objectives:

1. **Planning Optimization**: Ensures that the generated relation paths are faithful to the KG structure by aligning LLM outputs with valid KG-based paths.
2. **Retrieval-Reasoning Optimization**: Maximizes the probability of generating accurate answers based on the retrieved reasoning paths.

SelfCorrector: Step-by-Step Multi-round Self-correction Agent

To address the challenges of error accumulation and inefficiency in multi-step reasoning, our methodology incorporates a SelfCorrector agent inspired by the QueryAgent framework (Huang et al. 2024). The SelfCorrector employs an self-correction mechanism to ensure reliable reasoning over knowledge graphs.

Step-by-Step Reasoning and Action The SelfCorrector frames the KBQA process as an iterative, multi-turn reasoning task. At each step, the model generates a thought and executes a corresponding action to refine its query strategy on the subgraph. These actions interact with the KB and other environments, with their execution outcomes serving as observations to inform and adjust subsequent steps. Leveraging the reasoning capabilities and language understanding of large models, the SelfCorrector dynamically adjusts its strategy based on observations, correcting errors and iteratively improving the query until the final answer is determined.

Error Detection and Correction Based on the environmental feedback, the system performs adaptive self-correction tailored to the detected error type. For example:

- When the KB returns no results, the system analyzes the relation used in the query and provides alternative suggestions or revisions. For instance, it might observe: *"Error during execution: No results found for the given relation"*.
- If the Python Interpreter detects a syntax error or invalid function call, the system offers corrective feedback. For example, when the action `set_answer["final_answer1']` has an incorrect format, it generates feedback such as: *"Invalid set_answer action format. Correct format: set_answer(variable)"*.
- Logical inconsistencies identified by the Reasoning Memory lead the system to revise previous steps. For example, if an action like `get_neighbors("entity1", "relation1","entity2")` is formatted incorrectly, the system responds with: *"Invalid get_neighbors*

You are asked to answer this question by using some predefined functions. Here is three functions you can use.

1.get_relations([entities]): Retrieves all unique relationships directly associated with a given entity. [entities]: A list of entity identifiers (strings or numbers) enclosed in square brackets (e.g., ["entity1", "entity2"]).

2. get_neighbors([entities], relation): Fetches the neighbors of a specified entity, optionally filtered by a specific relationship. [entities]: A list of entity identifiers, formatted as a comma-separated string enclosed in square brackets (e.g., ["entity1", "entity2"]).

3. set_answer(variable): This is used when you think the answer was already complete . Calling this Action will set the variable as final answer.If the final answer consists of multiple items, present them in a list format [ans1, ans2, ...]. If it is a single item, represent it as a string : [ans].

Please follow this instruction: Your answer should have two parts:
- Thought: Explain your reasoning or analysis that leads to the action.
- Action: Call one of the predefined functions in the format: Action {number}: function

Note: Your answer should prioritize brevity and ensure that the action is given as soon as possible.

Example: [.....]

Figure 3: The prompt template for FewShot, omitting self-correction through Thought, Action, and Observation.

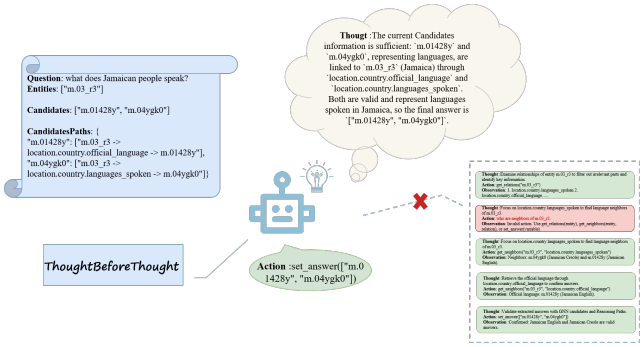

Figure 4: The figure illustrates how the ThoughtBeforeThought module evaluates available candidates and reasoning paths before initiating further reasoning. With sufficient evidence from m.03_r3 (Jamaica) linking to valid answers m.01428y (Jamaican English) and m.04ygk0 (Jamaican Creole), the module directly sets the answer, avoiding additional steps and reducing computational overhead.

action format. Correct format is get_neighbors(entity, relation)".

These feedback observations are dynamically analyzed, enabling the system to refine its reasoning process iteratively. Additionally, the system ensures robustness by suggesting valid actions when an invalid one is detected, such as: *"Invalid action. Use 'get_relations(entity)', 'get_neighbors(entity, relation)', or 'set_answer(variable)'."*

Through this feedback-driven approach, the system not only corrects errors but also optimizes the reasoning process, ensuring precise and efficient resolution of complex queries.

ThoughtBeforeThought

The ThoughtBeforeThought module is designed to optimize the reasoning process by leveraging information provided by PathFinder and CandidateFinder to directly determine the answer when sufficient evidence is available. By analyzing candidates and reasoning paths early in the process, ThoughtBeforeThought can often bypass the need for multi-step reasoning, reducing computational resource consumption and minimizing calls to large language models (LLMs).

Efficient Early Reasoning ThoughtBeforeThought evaluates whether the combined outputs of CandidateFinder and PathFinder provide enough information to confidently answer the question. It consolidates:

- Candidate entities identified by CandidateFinder.
- Reasoning paths generated by PathFinder that directly link the question entity to potential answers.

When these components form a clear and unambiguous reasoning chain, the module sets the answer directly without invoking further reasoning steps.

Examples of Early Answer Determination In cases where the evidence is sufficient, ThoughtBeforeThought enables direct answer determination,as shown in Figure4.

Experimental Settings

Dataset and Evaluation Metrics

We evaluated our model on two benchmark datasets, WebQuestionsSP (WebQSP) (Yih et al. 2016) and Complex WebQuestions (CWQ) (Talmor and Berant 2018), both based on Freebase (Bollacker et al. 2008). These datasets contain questions requiring up to 2-hop and 4-hop reasoning, respectively, as shown in Table 1.

To evaluate the performance of our model, we use standard metrics including Precision, Recall, F1-score, and Hit@1. These metrics are computed by comparing the predicted answers (`pred`) with the ground truth answers (`gt_answer`):

- **Precision**: Defined as the ratio of correctly predicted answers to the total number of predicted answers, i.e.,

$$\text{Precision} = \frac{|\text{pred\_set} \cap \text{gt\_set}|}{|\text{pred\_set}|}.$$

- **Recall**: Defined as the ratio of correctly predicted answers to the total number of ground truth answers, i.e.,

$$\text{Recall} = \frac{|\text{pred\_set} \cap \text{gt\_set}|}{|\text{gt\_set}|}.$$

- **F1-score**: The harmonic mean of Precision and Recall, calculated as:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

- **Hit@1**: A binary metric indicating whether at least one ground truth answer is present in the predicted set:

$$\text{Hit@1} = \begin{cases} 1.0 & \text{if } \exists x \in \text{gt\_set such that } x \in \text{pred\_set}, \\ 0.0 & \text{otherwise.} \end{cases}$$

The ground truth set (`gt_set`) includes both text answers and knowledge base IDs, while the predicted set (`pred_set`) is constructed by normalizing and matching strings in the predictions.

Table 1: Dataset Statistics

Datasets	Train	Dev	Test
WebQSP	2,848	250	1,639
CWQ	27,639	3,519	3,531

GNN Configuration in CandidateFinder

For CandidateFinder agent, GNNs are employed to model the candidate answer generation process, following the methodology outlined in (Mavromatis and Karypis 2024). The GNN configurations for different datasets are as follows:

- **WebQSP Dataset**:
  - GNN Model: ReaRev
  - Language Model: Sentence-BERT (SBERT)
- **CWQ Dataset**:
  - GNN Model: ReaRev
  - Language Model: LMsr

LLMs for PathFinder

We utilize the fine-tuned large language model (RoG) described in (Luo et al. 2023b) as the backbone for the PathFinder agent. Specifically, we use LLaMA2-Chat-7B (Touvron et al. 2023), which has been instruction fine-tuned on the training splits of WebQSP, CWQ, and Freebase for 3 epochs. This fine-tuning ensures the model is optimized for generating reasoning paths in KB. For each question, PathFinder employs beam search to generate the top-3 relation paths, ensuring a diverse and relevant set of reasoning paths for accurate answer derivation.

SelfCorrector Configuration

The SelfCorrector agent utilizes the OpenAI API(Achiam et al. 2023) to access the GPT-4o-mini large language model for iterative reasoning and self-correction. The maximum number of reasoning steps is set to 7, ensuring a balance between computational efficiency and reasoning depth. This configuration allows the SelfCorrector to effectively refine query strategies and correct errors dynamically while maintaining resource efficiency.

Hardware Setup

All experiments were conducted on a machine equipped with an Intel Core i9 processor, 32GB of RAM, and an NVIDIA RTX 4090 GPU with 24GB of VRAM.

Results and Discussion

Results and Analysis

As shown in Figure 5, our framework integrates multiple reasoning agents that collaborate seamlessly. The CandidateFinder narrows the search space by generating candidate answers and associated paths, while the PathFinder ensures logical consistency by reasoning over these paths. The SelfCorrector enhances robustness through iterative error detection and refinement, ensuring precise answers even in the face of ambiguous or incomplete information.

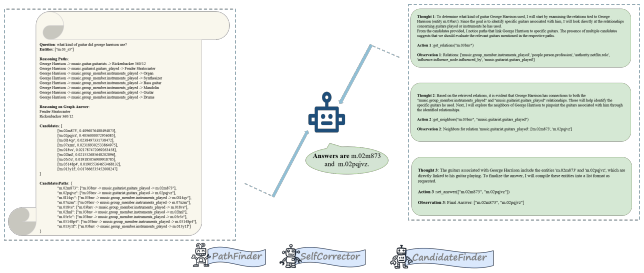

Figure 5: An example illustrating the full KBQA reasoning process using our proposed framework. The process includes CandidateFinder for identifying candidates and paths, PathFinder for exploring reasoning paths, and SelfCorrector for iterative answer refinement. This showcases how subgraph-based reasoning, intermediate thoughts, and feedback-driven corrections contribute to accurate and interpretable answers.

Table 2 presents the performance comparison of various methods on the WebQSP dataset. QueryAgent, as a Semantic Parsing-based approach focused on generating SPARQL queries, performs the weakest among the methods due to its limited ability to handle complex reasoning tasks and utilize subgraph information effectively. RoG, recognized as the current state-of-the-art (SOTA) method, achieves strong performance by leveraging fine-tuning on large-scale KBQA datasets, enabling it to handle diverse and complex queries effectively. Our method outperforms RoG in terms of overall reasoning and recall capabilities by integrating subgraph information, which enhances its ability to cover potential answers and tackle complex queries. SR+NSM w E2E and ReaRev deliver competitive results; however, their reliance on pre-defined pathways or specific reasoning structures limits their generalization compared to our approach. Overall, our method demonstrates balanced improvements across key metrics, showcasing the advantages of subgraph-based reasoning in improving reasoning depth and answer accuracy.

Table 2: Performance Comparison on WebQSP Dataset

Method	F1	Precision	Hit@1	Recall
RoG(Luo et al. 2023b)	0.7167	0.7816	**0.8498**	0.7165
QueryAgent(IR)(Huang et al. 2024)	0.4035	0.4723	0.5452	0.4000
ReaRev(Mavromatis and Karypis 2022)	0.7090	-	0.7640	-
SR+NSM w E2E(Zhang et al. 2022)	0.6410	-	0.6950	-
Ours	**0.7242**	**0.7823**	0.8278	**0.7365**

The experimental results on the CWQ dataset, presented in Table 3, highlight the challenges posed by the complexity of CWQ queries. Notably, we observed that setting the maximum retry attempts between 15 and 20 significantly improves the ability to derive the final answer. Additionally, part of the performance improvement stems from the model's inherent knowledge. When the model fails to infer an answer by reasoning through the subgraph, it sometimes leverages its internal knowledge to provide a response, which is occasionally correct. This phenomenon contributes to the observed improvements in both Hit@1 and F1 scores.

The combination of iterative subgraph reasoning and the model's internal knowledge results in more accurate and robust answers to the complex queries found in CWQ.

Table 3: Performance Comparison on CWQ Dataset

Method	Hit@1	F1
ReaRev	52.9	-
SR+NSM	50.2	47.1
RoG	57.8	56.2
GNN-RAG+RA(Mavromatis and Karypis 2024)	62.8	60.4
Ours	**64.9**	**61.8**

Ablation Study

Table 4 presents an ablation study on the impact of CandidateFinder on the WebQSP dataset. Excluding PathFinder, thought_before_thought, and SelfCorrecter, the experiment directly inputs subgraph information into the LLMs. Using CandidateFinder significantly improves all metrics by narrowing the search space and focusing on relevant entities. Without it, performance drops, underscoring its role in enhancing reasoning efficiency and accuracy.

Table 4: Ablation Study Results on WebQSP with and without CandidateFinder

Condition	F1	Precision	Recall	Hit@1
With CandidateFinder	0.5302	0.6078	0.5258	0.6795
Without CandidateFinder	0.3961	0.4627	0.3957	0.5409

Table 5 shows the impact of thought_before_thought with only SelfCorrecter and CandidateFinder. Incorporating thought_before_thought improves all metrics by enhancing reasoning steps and reducing errors. Without it, performance drops, highlighting its role in maintaining accuracy and consistency.

Table 5: Ablation Study: Effectiveness of thought_before_thought on WebQSP Dataset

Condition	F1	Precision	Recall	Hit@1
With thought_before_thought	0.6580	0.7672	0.6388	0.7979
Without thought_before_thought	0.6171	0.6965	0.6130	0.7466

Table 6 shows the impact of PathFinder with CandidateFinder, thought_before_thought, and SelfCorrecter. Including PathFinder, which provides reasoning paths and answers, improves F1, recall, and Hit@1, highlighting its role in guiding logical flow and refining results. Without PathFinder, performance declines, emphasizing its importance for reasoning accuracy.

Conclusion

In this paper, we propose a KBQA framework that combines subgraph-based reasoning with multi-agent collaboration to address the limitations of existing methods. By leveraging CandidateFinder, PathFinder, and SelfCorrector, our approach improves reasoning accuracy, recall, and robustness while handling complex queries effectively. The integration of subgraph information and dynamic feedback mechanisms enhances answer coverage and consistency. Future work will explore optimizing efficiency and extending the framework to larger, more diverse knowledge bases.

Table 6: Ablation Study: Effectiveness of PathFinder on WebQSP Dataset

Condition	F1	Precision	Recall	Hit@1
With PathFinder	0.6886	0.7585	0.6850	0.8163
Without PathFinder	0.6580	0.7672	0.6388	0.7979

Acknowledgments

Thank you to my teaching assistants and professors for their valuable guidance and support in my deep learning course.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 1247–1250.

Cui, W.; Xiao, Y.; Wang, H.; Song, Y.; Hwang, S.-w.; and Wang, W. 2019. KBQA: learning question answering over QA corpora and knowledge bases. *arXiv preprint arXiv:1903.02419*.

Huang, X.; Cheng, S.; Huang, S.; Shen, J.; Xu, Y.; Zhang, C.; and Qu, Y. 2024. QueryAgent: A Reliable and Efficient Reasoning Framework with Environmental Feedback based Self-Correction. *arXiv preprint arXiv:2403.11886*.

Li, T.; Ma, X.; Zhuang, A.; Gu, Y.; Su, Y.; and Chen, W. 2023. Few-shot in-context learning for knowledge base question answering. *arXiv preprint arXiv:2305.01750*.

Li, Z.; Fan, S.; Gu, Y.; Li, X.; Duan, Z.; Dong, B.; Liu, N.; and Wang, J. 2024. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 18608–18616.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Luo, H.; Tang, Z.; Peng, S.; Guo, Y.; Zhang, W.; Ma, C.; Dong, G.; Song, M.; Lin, W.; Zhu, Y.; et al. 2023a. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975*.

Luo, L.; Li, Y.-F.; Haffari, G.; and Pan, S. 2023b. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*.

Mavromatis, C.; and Karypis, G. 2022. Rearev: Adaptive reasoning for question answering over knowledge graphs. *arXiv preprint arXiv:2210.13650*.

Mavromatis, C.; and Karypis, G. 2024. GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning. *arXiv preprint arXiv:2405.20139*.

Nie, Z.; Zhang, R.; Wang, Z.; and Liu, X. 2024. Code-style in-context learning for knowledge-based question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 18833–18841.

Patidar, M.; Sawhney, R.; Singh, A.; Chatterjee, B.; Bhattacharya, I.; et al. 2024. Few-shot Transfer Learning for Knowledge Base Question Answering: Fusing Supervised Models with In-Context Learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 9147–9165.

Schuster, T.; Lelkes, A. D.; Sun, H.; Gupta, J.; Berant, J.; Cohen, W. W.; and Metzler, D. 2023. SEMQA: Semi-Extractive Multi-Source Question Answering. *arXiv preprint arXiv:2311.04886*.

Sun, Y.; Zhang, L.; Cheng, G.; and Qu, Y. 2020. SPARQA: skeleton-based semantic parsing for complex questions over knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 8952–8959.

Talmor, A.; and Berant, J. 2018. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*.

Tan, C.; Chen, Y.; Shao, W.; and Chen, W. 2023. Make a choice! knowledge base question answering with in-context learning. *arXiv preprint arXiv:2305.13972*.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xi, Z.; Chen, W.; Guo, X.; He, W.; Ding, Y.; Hong, B.; Zhang, M.; Wang, J.; Jin, S.; Zhou, E.; et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Yih, W.-t.; Richardson, M.; Meek, C.; Chang, M.-W.; and Suh, J. 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)*, 201–206.

Zhang, J.; Zhang, X.; Yu, J.; Tang, J.; Tang, J.; Li, C.; and Chen, H. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. *arXiv preprint arXiv:2202.13296*.

Zheng, X.; Che, F.; Wu, J.; Zhang, S.; Nie, S.; Liu, K.; and Tao, J. 2024. KS-LLM: Knowledge Selection of Large Language Models with Evidence Document for Question Answering. *arXiv preprint arXiv:2404.15660*.

Zong, C.; Yan, Y.; Lu, W.; Huang, E.; Shao, J.; and Zhuang, Y. 2024. Triad: A Framework Leveraging a Multi-Role LLM-based Agent to Solve Knowledge Base Question Answering. *arXiv preprint arXiv:2402.14320*.