Triad: A Framework Leveraging a Multi-Role LLM-based Agent to Solve Knowledge Base Question Answering

Chang Zong[1], **Yuchen Yan**[1], **Weiming Lu**[1†], **Jian Shao**[1]
Yongfeng Huang[2], **Heng Chang**[3], **Yueting Zhuang**[1†]
[1]College of Computer Science and Technology, Zhejiang University
[2]The Chinese University of Hong Kong
[3]Tsinghua University
{zongchang, luwm, yzhuang}@zju.edu.cn

Abstract

Recent progress with LLM-based agents has shown promising results across various tasks. However, their use in answering questions from knowledge bases remains largely unexplored. Implementing a KBQA system using traditional methods is challenging due to the shortage of task-specific training data and the complexity of creating task-focused model structures. In this paper, we present **Triad**, a unified framework that utilizes an LLM-based agent with multiple roles for KBQA tasks. The agent is assigned three roles to tackle different KBQA subtasks: agent as a generalist for mastering various subtasks, as a decision maker for the selection of candidates, and as an advisor for answering questions with knowledge. Our KBQA framework is executed in four phases, involving the collaboration of the agent's multiple roles. We evaluated the performance of our framework using three benchmark datasets, and the results show that our framework outperforms state-of-the-art systems on the LC-QuAD and YAGO-QA benchmarks, yielding F1 scores of 11.8% and 20.7%, respectively.

1 Introduction

A question-answering system is designed to extract information by converting a natural language question into a structured query that can retrieve precise information from an existing knowledge base (Omar et al., 2023a). The resolution of Knowledge Base Question Answering (KBQA) typically involves phases including question understanding, URI linking, and query execution. Traditional KBQA systems require the use of specialized models trained with domain datasets for question parsing and entity linking (Hu et al., 2018; Omar et al., 2023a; Hu et al., 2021). Large language models (LLMs), however, have shown promising competencies in in-context learning using task-specific demonstrations (Dong et al., 2022). LLMs have recently been employed as agents in the execution of complex problems. A framework that employs LLM-augmented agents can generate actions or coordinate multiple agents, thus improving the capacity to handle complex situations (Liu et al., 2023). Despite the remarkable performance of LLMs in various tasks as evidenced in previous studies, a comprehensive qualitative and quantitative evaluation of KBQA frameworks empowered with an LLM-based agent remains insufficiently explored.

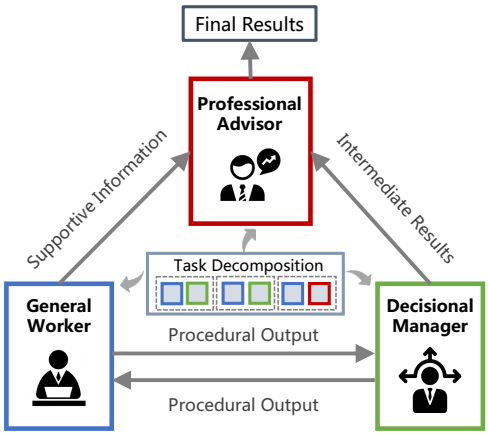

Figure 1: A system with multiple roles who focus on sub-problems of each phase to solve a complex task.

Studies on KBQA with LLMs has attracted considerable attention. Some works focus primarily on highlighting the inability of LLMs to generate complete factoid results (Hu et al., 2023b; Tan et al., 2023c) or demonstrating their potential efficacy in future research (Omar et al., 2023b; Tan et al., 2023b). Other works concentrates on generating answers by prompt learning and incorporating external knowledge bases (Baek et al., 2023; Tan et al., 2023a). Concurrently, LLMs can be deployed to address each phase within Text2SQL challenges(Li et al., 2023, 2024) or theorem proof tasks(Dong et al., 2023). However, each phase of

†Corresponding authors.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 1698–1710
November 12-16, 2024

KBQA can be further decomposed into subtasks and completed through an agentic approach that provides feedback and cooperation. Additionally, decomposing the task reduces the complexity of cooperative working by allowing each role to concentrate on smaller sub-problems(Wang et al., 2020). As illustrated in Figure 1, three roles in an organization work together to provide the final answer for the overall task. The above observations spur our exploration into the following question: **How does an LLM-based agent solve KBQA tasks by serving as multiple roles, and its performance is comparable to systems trained specifically?**

In this study, we introduce **Triad**, a unified framework that leverages an LLM-based agent with three roles to address KBQA tasks. Specifically, we implement the agent consisting of an LLM as the core, supplemented by various task-specific modules such as memory and executing functions. The agent is assigned three distinct roles: a generalist (G-Agent) adept at mastering numerous small tasks by the given examples, a decision maker (D-Agent) proficient at identifying options and selecting candidates, and an advisor (A-Agent) skilled at providing answers using internal and external knowledge. The cooperation of these agent roles composes a KBQA process containing four phases: question parsing, URI linking, query construction, and answer generation. We evaluate our framework on three benchmark datasets in various difficulties. The results show that our framework outperforms state-of-the-art systems, demonstrated by 11.8% and 20.7% F1 scores on the LC-QuAD and YAGO-QA benchmarks, respectively[1].

The contributions of this study can be summarized as follows:

- We propose Triad, the first framework that leverages an LLM-based agent to solve KBQA tasks in all its four phases, without specialized training models.
- We implement an LLM-based agent with various task-specific modules that can act as three roles, including a generalist, a decision maker, and an advisor, to collaboratively solve KBQA via focusing on subtasks.
- We evaluate the performance of Triad. The results show a competitive ability compared to both state-of-the-art KBQA systems and pure LLM methods.

[1]Code and data are available at https://github.com/ZJU-DCDLab/Triad.

2 Preliminaries

2.1 Phases of KBQA

A typical KBQA system has a process that encompasses four phases:

Question parsing involves converting natural language questions into a structured format that incorporates references to entities and relations.

URI linking entails associating and replacing these entity and relation mentions with their corresponding URIs within a knowledge base.

Query construction involves creating executable queries in a standard format to extract answers from knowledge bases.

Answer generation seeks to obtain the ultimate answers either by performing queries within knowledge bases or by directly querying an agent.

2.2 Roles of LLM-based Agent

Drawing an analogy to a software development scenario, where coders complete small development tasks, with the process and plan being decided by the manager, and ultimately the outcome inspected by the leader, we assign the following three roles to an LLM-based agent to solve the KBQA task:

Agent as a generalist (G-Agent) is capable of mastering various small tasks by providing a few examples.

Agent as a decision-maker (D-Agent) adepts at analyzing options and providing candidate results as procedural feedback.

Agent as an advisor (A-Agent) is skilled in providing final answers with the aid of both external and its own knowledge.

2.3 Task Formulation

A KBQA task refers to the process of solving a set of subtasks S. Each subtask $S_t \in S$ contributes to one phase of the whole process. An LLM-based agent $Agent_r$ with a role r can be used to resolve a type of subtasks by its task-specific components, including a language model LLM, a memory Mem_t, a function F_t, a prompt Pmt_t and a set of parameters θ_t, using the set of role-related hyperparameters σ_r. The task can be formulated as follows:

$$f(KBQA) = \bigoplus_{t=1}^{T} f(S_t)$$
$$f(S_t) = Agent_r(LLM, Mem_t, F_t, Pmt_t, \theta_t, \sigma_r) \quad (1)$$

, where T is the total number of subtasks, $\bigoplus$ is the way to coordinate subtasks to solve the whole.

3 Triad Framework

The overall architecture of **Triad** is shown in Figure 2. Each role of the LLM-based agent, along with its associated subtasks, is illustrated as follows.

3.1 G-Agent as a Generalized Solver

A generalized agent (G-Agent) proficiently manages numerous tasks by leveraging learning from limited examples through an LLM. In our framework, a G-Agent can perform question parsing, query template generation, or answer type classification as actions solely utilizing an LLM. These three subtasks are illustrated as follows:

Triplet mention extraction: The process of extracting triplet mentions in question parsing involves the conversion of a naturally phrased question, denoted as Q, into formatted triplets of entities and relations. This subtask is executed employing an LLM, which is guided by a prompt with a set of prerequisites and a selection of examples. This subtask can be represented as follows:

$$\begin{aligned} f(S_{tri}) &= Agent_g(LLM, Pmt_{tri}, Q, \mathcal{N}) \\ Pmt_{tri} &= [Ins_{tri}, Shot_{tri}, CoT_{tri}] \end{aligned} \quad (2)$$

, where $Agent_g$ is the agent as a generalist to perform the triplet extraction subtask with $\mathcal{N}$ examples. Pmt_{tri} is the prompt to guide LLM to generate triplets from the question Q, which consists of instruction Ins_{tri}, examples $Shot_{tri}$, and chain-of-thought prompt CoT_{tri} (Kojima et al., 2022).

SPARQL template generation: The generation of SPARQL templates in query construction involves the use of an LLM to create a SPARQL template that articulates the question using standard SPARQL syntax, replacing URI identifiers with entity and relation variables. To derive precise and comprehensive answers from the knowledge base using SPARQL queries, there are two potential strategies. One approach involves the direct generation of an executable SPARQL using an LLM, though this method may significantly increase LLM call times and error rates when numerous candidate queries are in play. Alternatively, a SPARQL template can initially be generated with entity and relation variables, which are subsequently replaced with linked URIs. For the sake of stability and efficiency, we opt for the second strategy. This subtask can be denoted as:

$$\begin{aligned} f(S_{qt}) &= Agent_g(LLM, Pmt_{qt}, \theta_{qt}, \mathcal{N}), \\ Pmt_{qt} &= [Ins_{qt}, Shot_{qt}, CoT_{qt}], \\ \theta_{qt} &= [Q, f(S_{tri})] \end{aligned} \quad (3)$$

, where $Agent_g$ is the agent as generalist to perform SPARQL template generation with $\mathcal{N}$ examples, $f(S_{tri})$ is the triplets derived from the previous subtask, Pmt_{qt} is the prompt for LLM to generate SPARQL template.

Answer type classification: In the phase of answer generation, the answer type classification subtask refers to the process of assigning a specific category to a response according to the question. This process serves as a guiding mechanism for the framework to generate comprehensive and accurate answers. This classification subtask is denoted as:

$$\begin{aligned} f(S_{cls}) &= Agent_g(LLM, Pmt_{cls}, Q, \mathcal{N}), \\ Pmt_{cls} &= [Ins_{cls}, Shot_{cls}, CoT_{cls}] \end{aligned} \quad (4)$$

, where $Agent_g$ is the agent as a generalist to perform type classification subtask with $\mathcal{N}$ examples, Pmt_{cls} is the prompt for LLM.

3.2 D-Agent as a Decision-Maker

An agent as a decision maker (D-Agent) is capable of making candidate selections step by step through filtering and choosing from given options, harnessing the capabilities of an LLM and KB as memory. The D-Agent can effectively handle three subtasks, which are delineated as follows:

Candidate entity selection: The selection of candidate entities in URI linking is pivotal to the ultimate efficacy of KBQA. Prior research has focused primarily on developing a semantic similarity model to address this linking challenge. However, the linking task requires numerous iterations of searching within the knowledge base, which poses a compatibility issue for LLM-oriented methods. In our framework, an agent as a decision maker is utilized initially to filter all potential entity URIs from the knowledge base, subsequently deploying an LLM to select candidate URIs from a pool of

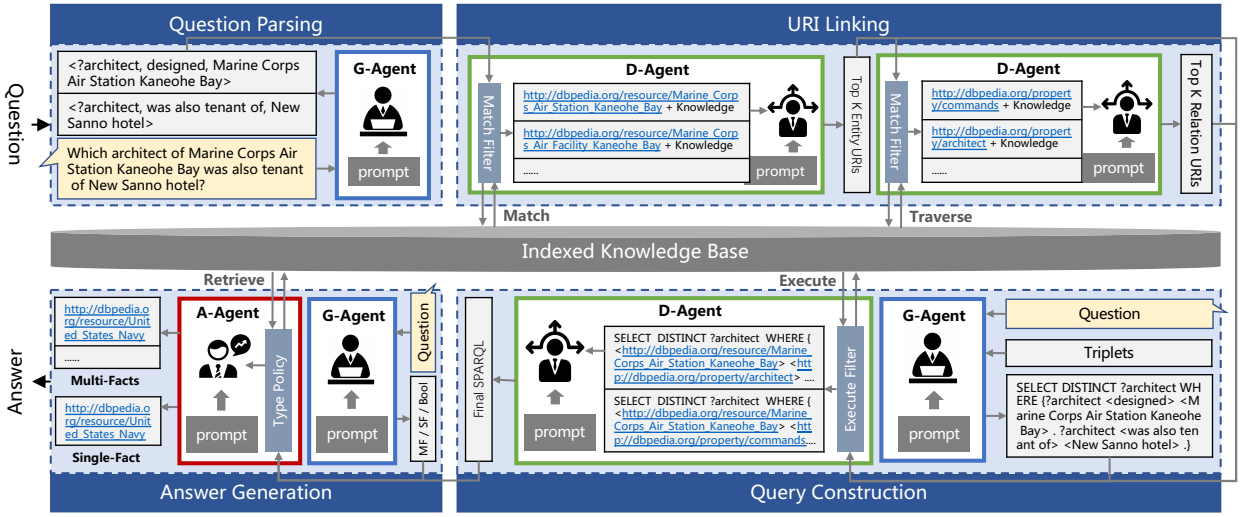

Figure 2: Our Triad framework leverages an LLM-based agent with three different roles including a generalist, a decision-maker, and an advisor to cooperatively handle a series of subtasks in the four phases of a KBQA process.

potential identifiers. For each entity, our aim is to find the $\mathcal{K}$ most possible entity URIs which can be used to traverse over the KB to get the final answer. The entity selection subtask can be denoted as:

$$\begin{aligned} f(S_{es}) =& Agent_d(LLM, Mem_{es}, F_{es}, \\ & Pmt_{es}, \theta_{es}, \mathcal{K}), \\ Mem_{es} =& [KB, List_{es}], \theta_{es} = [Q, f(S_{tri})] \end{aligned} \tag{5}$$

, where $Agent_d$ is the agent as a decision maker to perform the entity selection subtask with question Q, extracted triplets $f(S_{tri})$ and memory Mem_{es}, Mem_{es} is composed of a knowledge base KB and a list of entity URIs $List_{es}$ filtered from KB using a text similarity matching function F_{es}, Pmt_{es} is the prompt for LLM to perform the subtask, $\mathcal{K}$ is the hyperparameter of $Agent_d$, indicating the number of candidates selected by LLM.

Candidate relation selection: The task of candidate relation selection in URI linking presents considerable challenges due to the discrepancies between word forms and meanings. Nevertheless, the existence of reasoning paths in the knowledge base can be utilized to allow for a significant reduction of the search space in relation linking. In our framework, an agent as a decision maker endeavors to sieve through all potential relation URIs by navigating the knowledge base with candidate entity URIs generated from the previous subtask. Subsequently, an LLM is used to select the top $\mathcal{K}$ most probable relation URIs for output. The relation selection subtask can be denoted as:

$$\begin{aligned} f(S_{rs}) =& Agent_d(LLM, Mem_{rs}, F_{rs}, \\ & Pmt_{rs}, \theta_{rs}, \mathcal{K}), \\ Mem_{rs} =& [KB, List_{rs}], \theta_{rs} = [Q, f(S_{es})] \end{aligned} \tag{6}$$

, where memory Mem_{rs} is composed of the knowledge base KB and a list of possible relation URIs $List_{rs}$ filtered from KB using a one-order traversing function F_{rs}. Pmt_{rs} is the prompt for LLM to perform relation selection. $\mathcal{K}$ is the number of relation URIs selected by LLM.

Candidate SPARQL selection: The subtask of candidate SPARQL selection in query construction involves determining the appropriate SPARQL queries to obtain the final answers. Given a SPARQL template generated by the G-Agent, along with multiple candidate URIs selected from the D-Agent in previous subtasks, our D-Agent is targeted to identify the most plausible query. To further reduce the difficulty of selection, an executor function is applied to eliminate queries that cannot retrieve any results from the knowledge base. In conclusion, our aim in this subtask is to use D-Agent to construct executable SPARQLs and find the most possible one given a query candidate list with supported information. The SPARQL selection subtask can be denoted as:

$$\begin{aligned} f(S_{qs}) =& Agent_d(LLM, Mem_{qs}, F_{qs}, \\ & Pmt_{qs}, \theta_{qs}, \mathcal{K}), \\ Mem_{qs} =& [KB, List_{qs}], \\ \theta_{qs} =& [Q, f(S_{es}), f(S_{rs}), f(S_{qt})] \end{aligned} \tag{7}$$

, where memory Mem_{qs} is composed of a knowledge base KB and a list of possible SPARQLs $List_{qs}$ constructed with SPARQL template $f(S_{qt})$, entity URIs $f(S_{es})$, and relation URIs $f(S_{rs})$ by the function F_{qs}, Pmt_{qs} is the prompt for LLM to perform query selection, $\mathcal{K} = 1$ is the number of queries selected by LLM.

3.3 A-Agent as a Comprehensive Advisor

An advisory agent (A-Agent) is capable of processing a question and a corresponding type of answer as input. Its response is generated by either extracting information from an external knowledge base or by utilizing its internal knowledge to provide a direct answer. This comprehensive answering subtask can be described as follows:

Comprehensive answering: The objective of comprehensive answering in the answer generation phase is to derive a definitive response based on an incoming question. Previous work (Omar et al., 2023b) has demonstrated that LLMs are more proficient in delivering single-fact responses and making Boolean judgments. Given this understanding, we implement an advisory agent that incorporates a simple policy to facilitate a comprehensive answering approach. Specifically, if a question yields a final SPARQL generated from the preceding steps, A-Agent extracts elements from the knowledge base to give the answer. Conversely, if the agent does not receive a feasible SPARQL, A-Agent provides a direct response with LLM's internal knowledge, following the prompt based on the type of the answer. Additionally, A-Agent will send a retry signal to previous phases if no result is generated. The subtask can be formulated as below:

$$\begin{aligned} f(S_{ca}) =& Agent_a(LLM, Mem_{ca}, F_{ca}, \\ & Pmt_{ca}, \theta_{ca}, \mathcal{T}), \\ Mem_{ca} =& [KB], \theta_{ca} = [Q, f(S_{qs}), f(S_{cls})] \end{aligned} \tag{8}$$

, where $Agent_a$ is the agent as an advisor to perform a comprehensive answering for the question Q with a memory Mem_{ca} of knowledge base, Pmt_{ca} is the prompt for LLM to perform a direct response according to the type of the answer, $f(S_{qs})$ is the final query and $f(S_{cls})$ is the answer type, $\mathcal{T}$ is the maximum times to retry for previous phases if no result is returned from KB.

4 Performance Evaluation

4.1 Experimental Settings

Indexed Knowledge Bases: The efficacy of our framework is assessed through the collection of two real knowledge bases, specifically DBpedia and YAGO. DBpedia (Auer et al., 2007) serves as an accessible knowledge base extracted from Wikipedia, while YAGO (Pellissier Tanon et al., 2020) is a large knowledge base that includes individuals, cities, nations, and organizations. We index the triples and the mentions of entities and relations in a Virtuoso endpoint and an Elasticsearch server, respectively.

KBQA Benchmark Datasets: We evaluate our framework on datasets including YAGO-QA, LC-QuAD 1.0, and QALD-9, which have various difficulties in interpreting the questions. These datasets contain questions in English, paired with their respective SPARQL queries, and accurate responses derived from a specific knowledge base. QALD-9 (Usbeck et al., 2018) and LC-QuAD 1.0 (Trivedi et al., 2017) are frequently used to evaluate QA systems with DBpedia. The recently published YAGO-QA in (Omar et al., 2023a), features questions accompanied by annotated SPARQL queries sourced from YAGO. The statistics for three benchmarks, along with their associated knowledge bases, are depicted in Table 1.

Baseline Methods: We evaluate Triad against traditional KBQA systems such as KGQAN (Omar et al., 2023a), EDGQA (Hu et al., 2021) and gAnswer (Hu et al., 2018). This comparison shows how our LLM-based agent framework can rival full-shot systems with just a few examples. Additionally, we contrast our framework with pure GPT models like GPT-3.5 Turbo and GPT-4 [2] to exhibit Triad's architectural performance. We treat these foundation models as few-shot methods to answer the questions referring to some examples.

Implementation Details: Triad is implemented with Python 3.9. We incorporate LLM capabilities to our multi-role agent via OpenAI's API services. The names of entities and relations from knowledge bases are indexed in an ElasticSearch 7.5.2 server for text matching. All triples are imported into an SPARQL endpoint of Virtuoso 07.20.3237 for retrieval. Triad requires four hyperparameters: the number of examples G-Agent uses for subtask

[2] https://platform.openai.com/docs/models

Benchmarks	Benchmark Statistics				
	#Questions	KB	#Triples	Virtuoso Size	ES size
LC-QuAD 1.0	1000	DBpedia-04	397M	35.40G	1.56G
QALD-9	150	DBpedia-10	374M	36.89G	1.57G
YAGO-QA	100	YAGO-4	207M	24.85G	0.54G

Table 1: The statistics of KBQA benchmarks, including the number of questions number, the number of triples, the size of index in Virtuoso and Elasticsearch.

Type	Frameworks	LC-QuAD 1.0			QALD-9			YAGO-QA		
		P	R	F1	P	R	F1	P	R	F1
full-shot	gAnswer	-	-	-	0.293	0.327	0.298	0.585	0.341	0.430
	EDGQA	0.505	0.560	0.531	0.313	0.403	0.320	0.419	0.408	0.414
	KGQAN	**0.587**	0.461	0.516	**0.511**	0.387	**0.441**	0.485	0.652	0.556
few-shot	GPT-3.5	0.269	0.251	0.266	0.240	0.217	0.228	0.171	0.142	0.155
	GPT-4	0.336	0.344	0.340	0.250	0.249	0.249	0.193	0.190	0.191
	Triad-GPT3.5	0.490	0.519	0.504	0.293	0.302	0.297	0.660	0.639	0.649
	Triad-GPT4	0.561	**0.568**	**0.564**	0.408	**0.425**	0.416	**0.690**	**0.664**	**0.677**

Table 2: The performance of our proposed Triad on three benchmarks, comparing with traditional KBQA systems (full-shot) and pure LLM (few-shot) baselines. The optimal and suboptimal scores are highlighted with bold and underlined text, respectively.

learning, the number of candidates D-Agent selects for entity and relation linking, and the retry times for handling non-response SPARQLs. The optimal values for these parameters are 3, 2, 2, and 3, respectively. The framework and its variants are tested five times on each benchmark, with the average scores reported as the final results. For traditional systems, we report the results recorded in their papers. For pure LLM baselines, we write prompts to hire an LLM to answer questions directly referring to examples, and then link the mentions from the responses to the URIs in our indexed knowledge bases via built-in similarity search.

4.2 Performance Comparison

The performance of **Triad** compared to traditional KBQA systems and pure LLM generation methods is shown in Table 2. Evaluation metrics precision(P), recall(R), and F1-score(F1) are reported. We can observe from the experimental results that:

Few-shot can be competitive with full-shot. Our multi-role LLM-based agent framework, though executing a few-shot prompt learning, exhibits competitive performance with cutting-edge full-shot KBQA systems.

Underlying capability matters. The use of GPT-4 as the core in an LLM-based agent significantly outperforms GPT-3.5 on all benchmarks, demonstrating the importance of the underlying capabilities of an agent.

Explicit knowledge is necessary. Pure LLM models with GPT-3.5 and GPT-4 display deficiencies in generating accurate responses without an auxiliary knowledge base as a memory for intermediary steps such as URI linking.

Performance varies with complexity. Triad demonstrates superior results on the LC-QuAD and YAGO-QA benchmarks compared to QALD-9, due to an increasing failure in response to complex questions, which will be discussed later.

4.3 Study on Capabilities of Agent Roles

We assess the efficacy of G-Agent with various other language models as the core. The framework without **G-task** uses the text-davinci-002, which is not as powerful as GPT-3.5 and GPT-4 in solving many tasks, and the one without **G-chat** uses text-davinci-003 to eliminate the chat and alignment abilities. We test the ability of D-Agent without **D-uri** and **D-query** by replacing the URI selection and query selection with URI matching and query generation, respectively. We evaluate the contribution of A-Agent eliminating **A-llm** and **A-fact** by responding to questions without using LLM's assistance or use an LLM to answer Boolean questions

for auxiliary rather than single-fact questions. The F1 results of the role ablation experiments on two representative datasets are shown in Table 3. The results indicate that every component pertaining to each role contributes to the overall performance. More specifically, a G-Agent that employs a less powerful LLM as its core can drastically undermine performance. D-Agent assumes a more pivotal role during the linking phase compared to the query construction phase. A-Agent, on the other hand, proves to be an efficient solution for managing situations where SPARQL results are absent.

G-task	G-chat	LC-QuAD 1.0	QALD-9
✗	✗	0.343	0.159
✓	✗	0.443	0.248
✓	✓	0.564	0.416
D-uri	**D-query**	**LC-QuAD 1.0**	**QALD-9**
✗	✓	0.274	0.210
✓	✗	0.431	0.301
✓	✓	0.564	0.416
A-llm	**A-fact**	**LC-QuAD 1.0**	**QALD-9**
✗	✗	0.459	0.382
✓	✗	0.473	0.385
✓	✓	0.564	0.416

Table 3: Study on the roles of LLM-based agent by eliminating an element or downgrading the capability.

4.4 Analysis of Role Hyperparameters

We concentrate on three hyperparameters of roles, including the number of examples ($\mathcal{N} \in \{1, 2, 3\}$) provided for G-Agent to learn subtasks, the number of URI candidates ($\mathcal{K} \in \{(1, 1), (1, 2), (2, 2), (2, 3)\}$) selected by D-Agent for query construction, and the number of retry times ($\mathcal{T} \in \{1, 2, 3\}$) launched by A-Agent when there is no response. Table 4 presents the F1 results of Triad's performance, employing three hyperparameters on two benchmarks. We discover that:

Quality is more important than quantity. More examples provided to G-Agent do not always improve the performance. The efficacy of G-Agent is significantly influenced by the quality of examples.

More options may harm the result. Choosing more candidate URIs for entities and relations could potentially disrupt subsequent query phases, thus affecting overall performance.

Triad Variants	LC-QuAD 1.0	QALD-9
Triad-1-Shot	0.556	0.376
Triad-2-Shot	0.511	0.402
Triad-3-Shot	0.564	0.416
Triad-Top1-1	0.528	0.281
Triad-Top1-2	0.562	0.375
Triad-Top2-2	0.564	0.416
Triad-Top2-3	0.558	0.384
Triad-1-Try	0.529	0.375
Triad-2-Tries	0.561	0.407
Triad-3-Tries	0.564	0.416

Table 4: Performance evaluation on three hyperparameters that related to each role of an LLM-based agent.

More chances benefits the framework. Persistently attempting to construct and execute SPARQL queries is an effective strategy that improves the probability of obtaining accurate answers. Considering the efficiency of overall execution, we set the maximum retry times as 3 in practice.

4.5 Analysis of Linking Recall

The process of linking is a relatively complex subtask in both the Text2SQL and the KBQA process (Li et al., 2024). Calculating the recall ratio of accurate URIs using D-Agent provides clarity on which step most adversely impacts performance. In the entity linking phase, considering all URIs of entities in the testing set as the ground truth of the linking results, 80. 75% of the correct URIs are contained from the output of the entity matching filter in D-Agent and 70. 50% of the correct URIs are retained from the entity selection performed by the LLM in D-Agent. Whereas, in the relation linking phase, only 52. 54% of the correct relation URIs survive from the selection of LLM, which indicates a greater difficulty in relation linking.

4.6 Study on Complex Cases

Despite the impressive performance of Triad in certain benchmarks, notable deficiencies remain in its ability to understand questions and generate queries for complex questions. A critical analysis of unsuccessful cases in QALD-9, which has the lowest F1 score, has revealed three primary reasons for this failure, as detailed below:

Complex Syntax signifies that advanced SPARQL queries incorporate keywords such as *GROUP BY* and *HAVING*. These terms augment the error propensity in the generation of SPARQL

Fail Reason	Ratio	Example
Complex Syntax	20%	Q42: Which countries have places with more than two caves?
Unexploited Semantics	17%	Q199: Give me all Argentine films.
Implicit Reasoning	5%	Q133: What are the names of the Teenage Mutant Ninja Turtles?

Table 5: The major reasons of complexity that result in failures, with their corresponding ratios of occurrence in failed cases.

templates such as the example: *Which frequent flyer program has the most airlines?*

Unexploited Semantics indicates that semantics of an implicit entity should be comprehended in order to exclude irrelevant URIs. In the example *Give me all Argentine films*, the meaning of *films* should be used to narrow down the scope of potential entities in order to eliminate unrelated answers.

Implicit Reasoning presents a challenge that requires a deeper level of traversal by the framework to deduce accurate results from the posed question. For example, another failure question, *How many grand-children did Jacques Cousteau have?*, the term *grand-children* must be interpreted to *son of son* to ensure an accurate response.

4.7 Cost Comparison and Analysis

According to our evaluation on the three datasets, the average cost of running a single case is 0.007 USD on average using Triad-GPT3.5 and 0.05 USD on average using Triad-GPT4. Specifically, most API calls occur in the phases of URL linking and comprehensive answering. Meanwhile, traditional KBQA baselines require a lot of training data and local training resources to achieve the SOTA performance, whereas Triad follows a zero- or few-shot manner to save computational cost locally. Furthermore, as shown in Section 4.4, in practice, adjusting the hyperparameters can make the cost as low as possible while preserving overall performance. As the cost of LLM services decreases, the value of Triad will increase accordingly.

5 Related Work

5.1 SPARQL-based and LLM-based KBQA

Traditional KBQA methods transform natural language queries into SPARQL requests for data extraction. Specific models are employed either for question understanding or URI linking, utilizing domain-based training datasets. Hu et al. (2018) introduces a semantic query graph to structurally represent the natural language query, thereby simplifying the task into a subgraph matching problem. Hu et al. (2021) proposes an entity description graph to represent natural language queries for question parsing and element linking. Omar et al. (2023a) restructures the question parsing task as a text generation issue using a sequence-to-sequence model. With the advent of LLMs, certain phases of KBQA can be enhanced with LLM-integrated methods. Baek et al. (2023) aims to augment LLM-based QA tasks with pertinent facts extracted from knowledge bases, offering a fully zero-shot architecture. Tan et al. (2023a) leverages the general applicability of LLMs to filter linking candidates by making selections via few-shot in-context learning. Omar et al. (2023b) provides a thorough comparison between LLMs and QA systems, recommending further studies to improve KBQA with LLM capabilities. However, apart from the above studies, our study proposes a complete framework incorporating both an LLM and few-shot learning across all KBQA phases from a systematic perspective.

5.2 LLM-based Agents for Complex Tasks

LLMs have recently gained significant attention due to their ability to approximate human-level intelligence. This has led to numerous studies focusing on LLM-based agents. A recent survey(Wang et al., 2023) proposes a unified architecture for LLM-based agents, which consists of four modules that include profile, memory, plan, and action. CHATDB(Hu et al., 2023a) employs an LLM controller to generate SQL instructions, which allows for symbolic memory and complex multi-hop reasoning. ART(Paranjape et al., 2023) uses a frozen LLM to generate reasoning steps and further integrates tools for new tasks with minimal human intervention. Toolformer(Schick et al., 2024) takes a different approach by training an LLM to plan and execute tools for the next token prediction by learning API calls generation. ReAct(Yao et al., 2023) focuses on overcoming LLM hallucination by interacting with external knowledge bases,

thus generating interpretable task-solving strategies. CodeAgent(Tang et al., 2024) designs a multi-agent collaboration system across four phases in a code review process. Divergent from the aforementioned studies, our framework concentrates on the solving KBQA tasks by introducing a multi-role LLM-based agent that specializes in various subtasks distributed across different phases.

6 Conclusion

In this study, we aim to bridge the gap between KBQA tasks and the investigation of LLM-based agents. We introduce **Triad**, a framework to address the KBQA task through an LLM-based agent acting as multiple roles, including a generalist capable of mastering diverse tasks given minimal examples, a decision-maker concentrating on option analysis and candidate selection, and an advisor skilled in answering questions with the aid of both external and internal knowledge. Triad achieves the best or competitive performance across three benchmark datasets compared to traditional KBQA systems and pure LLM models. In future research, we plan to broaden our framework to handle more intricate questions, such as multi-hop reasoning, and exploring the integration between our framework and retrieval-augmented generation.

Limitations

The limitation of our research lies in following aspects: (1) In terms of data, a broader range of QA datasets needs to be evaluated, encompassing datasets from different domains, languages, and difficulty levels. (2) In terms of model, more LLMs need to be evaluated, including open-source and commercial models from different organizations and on various scales. (3) In terms of framework, more types of agent collaboration methods can be explored to solve KBQA problems.

Ethics Considerations

All datasets utilized in this study are publicly available and we have adhered to ethical considerations by not introducing additional information as input during LLM text generation.

Acknowledgements

This work is supported by the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (Grant No. 2023C01152 and No. 2024C01034).

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In The Semantic Web, pages 722–735, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jinheon Baek, AlhamFikri Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. arXiv preprint arXiv:2306.04136.

Qingxiu Dong, Li Dong, Ke Xu, Guangyan Zhou, Yaru Hao, Zhifang Sui, and Furu Wei. 2023. Large language model for science: A study on p vs. np. arXiv preprint arXiv:2309.05689.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. arXiv preprint arXiv:2301.00234.

Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. 2023a. Chatdb: Augmenting llms with databases as their symbolic memory. arXiv preprint arXiv:2306.03901.

Nan Hu, Yike Wu, Guilin Qi, Dehai Min, Jiaoyan Chen, Jeff Z Pan, and Zafar Ali. 2023b. An empirical study of pre-trained language models in simple knowledge graph question answering. World Wide Web, pages 1–32.

Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2018. Answering natural language questions by subgraph matching over knowledge graphs. IEEE Transactions on Knowledge and Data Engineering, page 824–837.

Xixin Hu, Yiheng Shu, Xiang Huang, and Yuzhong Qu. 2021. Edg-based question decomposition for complex question answering over knowledge bases. In The Semantic Web – ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings, page 128–145, Berlin, Heidelberg. Springer-Verlag.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. Advances in neural information processing systems, 35:22199–22213.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 13076–13084.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale

database grounded text-to-sqls. Advances in Neural Information Processing Systems, 36.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. arXiv preprint arXiv:2308.05960.

Reham Omar, Ishika Dhall, Panos Kalnis, and Essam Mansour. 2023a. A universal question-answering platform for knowledge graphs. Proceedings of the ACM on Management of Data, 1(1):1–25.

Reham Omar, Omij Mangukiya, Panos Kalnis, and Essam Mansour. 2023b. Chatgpt versus traditional question answering for knowledge graphs: Current status and future directions towards knowledge graph chatbots. arXiv preprint arXiv:2302.06466.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and MarcoTulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. arXiv preprint arXiv:2303.09014.

Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. 2020. Yago 4: A reason-able knowledge base. In The Semantic Web, pages 583–596, Cham. Springer International Publishing.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36.

Chuanyuan Tan, Yuehe Chen, Wenbiao Shao, Wenliang Chen, Zhefeng Wang, Baoxing Huai, and Min Zhang. 2023a. Make a choice! knowledge base question answering with in-context learning. arXiv preprint arXiv:2305.13972.

Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023b. Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family. In International Semantic Web Conference, pages 348–367. Springer.

Yiming Tan, Dehai Min, Yu Li, Wenbo Li, Nan Hu, Yongrui Chen, and Guilin Qi. 2023c. Evaluation of chatgpt as a question answering system for answering complex questions. arXiv preprint arXiv:2303.07992.

Daniel Tang, Zhenghan Chen, Kisub Kim, Yewei Song, Haoye Tian, Saad Ezzini, Yongfeng Huang, and Jacques Klein Tegawende F Bissyande. 2024. Collaborative agents for software engineering. arXiv preprint arXiv:2402.02172.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. Lc-quad: A corpus for complex question answering over knowledge graphs. In The Semantic Web – ISWC 2017, pages 210–218, Cham. Springer International Publishing.

Ricardo Usbeck, Ria Hari Gusmita, Axel-Cyrille Ngonga Ngomo, and Muhammad Saleem. 2018. 9th challenge on question answering over linked data (qald-9) (invited paper). In Semdeep/NLIWoD@ISWC.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023. A survey on large language model based autonomous agents. arXiv preprint arXiv:2308.11432.

Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. 2020. Rode: Learning roles to decompose multi-agent tasks. arXiv preprint arXiv:2010.01523.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models.

A Response Time Analysis

We analyze various QA frameworks in response time to a question. The average latency of each phase including question parsing (QP), URI linking (UL), and answer generation (AG) for each knowledge base is reported. We randomly select 10 samples from each dataset for evaluation to obtain the average response times for Triad-1 and Triad-3, which represent retrying three times and generating an answer in one go, respectively, during the answer generation phase. The comparison between traditional QA systems and Triad is shown in Figure 3. Triad generally shows a competitive time-consuming performance to latest traditional QA systems. Specifically, compared to other phases, URL linking consumes more time due to the need to invoke LLM multiple times. Moreover, according to Section 4.4, with smaller retry times of A-Agent, Triad can significantly reduce time cost while only causing slight performance degradation, revealing the advantages of our framework in balancing performance and efficiency.

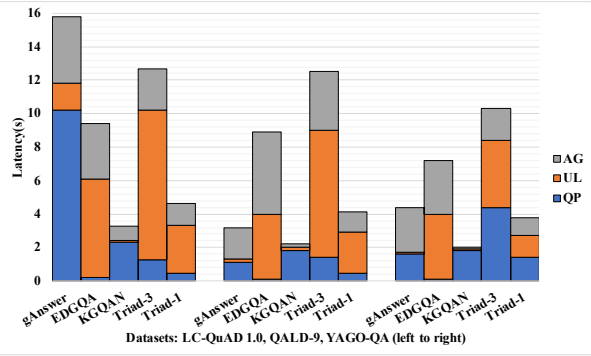

Figure 3: Response time of traditional KBQA systems and Triad on three datasets. Each bar shows average response time of a particular phase of KBQA.

B Role Performance on YAGO-QA

We choose LC-QuAD 1.0 and QALD-9 as our two representative datasets in Section 4.3, as the questions among them vary in difficulty, and the tasks in these two datasets are relatively more challenging than YAGO-QA. We provide the performance of agent roles on YAGO-QA in Table 6, which shows a consistent result with other datasets in Table 3.

C Prompts Provided to LLMs of G-Agent for Solving Various Subtasks in KBQA

The prompt given to LLMs of $Agent_g$ to perform triplet extraction from the question Q is as follows:

G-task	G-chat	YAGO-QA
✗	✗	0.427
✓	✗	0.553
✓	✓	0.677
D-uri	**D-query**	**YAGO-QA**
✗	✓	0.346
✓	✗	0.534
✓	✓	0.677
A-llm	**A-fact**	**YAGO-QA**
✗	✗	0.626
✓	✗	0.647
✓	✓	0.677

Table 6: Performance of roles of LLM-based agent by eliminating an element or downgrading the capability.

You are an assistant to ***identify triples*** within a provided sentence. Please adhere to the following **guidelines:**

1. Triples should be structured in the format <entity1, relation, entity2>.
2. The sentence must contain at least one triple, so you should provide at least one.
3. Entities should represent the smallest semantic units and should not contain descriptive details.
4. Entities can take the form of explicit or implicit references. Explicit entities refer to specific named resources, whereas implicit entities are less certain.
5. When an entity is implicit, utilize a variable format such as '?variable' to denote it, for example, '?location' or '?person'.

Here are some examples:
Which city's founder is John Forbes? : <?city, foundeer, John Forbes>
How many races have the horses bred by Jacques Van't Hart participated in? : <?horse, participated in, ?race> <?horse, breeder, Jacques Van't Hart>
Is camel of the chordate phylum? : <camel, phylum, chordate>

Sentence: <Question Sentence>
Output:

The prompt given to LLMs of $Agent_g$ for SPARQL template generation is as follows:

You are an assistant to ***generate a SPARQL query*** to address a specific question. Here are the **guidelines to follow:**

1. Ensure that the resulting SPARQL query is designed to answer the provided question.
2. Adhere to the commonly accepted SPARQL standards when generating the query.
3. Make an effort to leverage the information provided to assist in the creation of the SPARQL query.
4. Strive to keep the generated SPARQL query as straightforward as possible.
5. Avoid including 'PREFIX' or ':' in the SPARQL query.
6. Enclose condition entities and predicates within angle brackets, such as <entity> or <predicate>.
7. Maintain the original order of the given triples without altering their sequence.

Question: <question sentence>
Triplets: <extracted triplets>
Output:

The prompt given to LLMs of $Agent_g$ for question type classification is as follows:

You are an assistant to ***determine the specific type*** of a given question according to the following guidelines:

1. You must determine the most probable question type for the input question.
2. The type of question should be enclosed within angle brackets, denoted as '<' and '>'.
3. Possible question types include: <count>, <select>, and <yes or no>.

Question: <question sentence>
Output:

D Prompts Provided to LLMs of D-Agent for Solving Selection Subtasks in KBQA

The prompt given to LLMs of $Agent_d$ for candidate entities selection is as follows:

You are an assistant to ***select <K> URIs*** from a provided list of possible URIs for a specified entity, following these **guidelines:**

1. Identify the <K> most appropriate URIs from the given list that best represent the entity in question.
2. Seek to understand the semantic information associated with the specified entity by examining the provided question.
3. The output should consist of <K> URIs chosen from the provided list of possible URIs.
4. Simply output these <K> target URIs, each on a separate line, without providing any additional explanations.

Sentence: <question sentence>
Entity: <entity mention>
Possible entity URIs: <Entity URI list>
Output:

The prompt given to LLMs of $Agent_d$ for candidate relation selection is as follows:

You are an assistant tasked with ***selecting the <K> relation URIs*** between entities mentioned in a sentence. Here are the **guidelines:**

1. The two entities are listed one after the other, without a specific order.
2. Use the provided sentence to discern the semantic meaning of these entities.
3. The potential relation URIs are listed one by one.
4. Your output should consist of a maximum of <K> possible relation URIs, although you may also output fewer if appropriate.
5. Ensure that your output is organized, prioritizing the most likely relationship first.
6. Provide a list of no more than <K> relation URIs (each on a separate line if there are multiple) without any additional descriptions.

Sentence: <question sentence>
Entities: <entity pair>
Possible relation URIs: <URI list>
Output:

The prompt given to LLMs of $Agent_d$ for the final SPARQL selection is as follows:

You are an assistant to ***select an appropriate SPARQL query*** from the provided list in order to respond to a specific question. Please adhere to the following **guidelines:**

1. Select the most suitable SPARQL query from the given query list to address the question.
2. Select a SPARQL query solely from the provided list; avoid crafting your own SPARQL query.
3. The selected SPARQL query must be applicable to answer the given question.

Sentence: <question sentence>
SPARQL candidates: <SPARQLs to choose>
Output:

E Prompts Provided to LLMs of A-Agent for Solving Answering Subtask in KBQA

The prompt given to LLMs of $Agent_a$ to generate a yes or no answer for the give question is as follows:

You are an assistant to ***answer a yes-or-no question***. Please adhere to the following **guidelines:**

1. If you believe that the answer is yes, provide an output of 'True'. If not, provide an output of 'False'.
2. Please do not include additional information or explanations in your response.

Sentence: <question sentence>
Output:

The prompt given to LLMs of $Agent_a$ to generate a single-fact answer for the give question is as follows:

You are an assistant to ***answer a question***. Please adhere to the following **guidelines:**

1. The answer to the question is a single entity.
2. You should just output the full expression of the answer without any punctuation.
3. Do not output any other description.

Sentence: <question sentence>
Output: