Self-VEQA Agent: Self-Verification Enhanced Question Answering Agent

Anonymous ACL submission

Abstract

The task of Knowledge Graph Question Answering (KGQA) involves using information stored in a knowledge graph (KG) to answer questions by identifying the relation path between the subject entity and the answer. Traditional KGQA methods require extensive training data and are time-consuming. Recent advancements in Large Language Models (LLMs) have shown potential in various tasks. However, methods leveraging LLMs for KGQA face challenges such as inference errors and excessive reliance on prompt design. To address these issues, we propose the Self-VEQA Agent, which utilizes two agents: a QA Agent for initial answers based on KG and a Verification Agent to iteratively refine these answers, improving accuracy over time. Additionally, our model features a memory mechanism that enables dynamic evolution. As the Self-VEQA Agent performs tasks and accumulates experience, the overall performance improves over time. Evaluated on two KGQA benchmarks, Self-VEQA Agent outperforms most traditional and LLM-based methods, demonstrating its effectiveness.

1 Introduction

KGQA task aims to respond to NLP questions using information stored in a knowledge graph (KG) by discerning the relation path between the subject entity and the answer. Traditional methods (He et al., 2021; Jiang et al., 2022; Xu et al., 2019; Saxena et al., 2020) for KGQA involve training models on a substantial amount of training data specific to a dataset to perform question-answering tasks. However, these methods face challenges such as high time costs and the need for extensive training data.

Recently, LLMs have excelled across various tasks (Anil et al., 2023; Bai et al., 2023; Touvron et al., 2023), primarily attributed to their training on extensive datasets and parameter scales reaching billions to trillions. Many studies lean towards leveraging prompt engineering with LLMs, as it eliminates the need for fine-tuning while harnessing their inference capabilities to achieve satisfactory performance.

Using LLMs without fine-tuning for KGQA tasks mainly falls into two categories. One (Wu et al., 2023; Kim et al., 2023; Guo et al., 2023) involves retrieving relevant knowledge to serve as external knowledge for the LLMs, essentially knowledge augmentation, enabling the LLM to answer questions. Methods based on knowledge augmentation primarily utilize retrieved knowledge for knowledge enhancement. However, the lengthy input text provided to the LLM for inference has compromised its ability to make accurate inferences, resulting in reduced effectiveness and occasional reasoning errors. The other (Taffa and Usbeck, 2023; Madani et al., 2023) involves generating SPARQL statements using LLMs. However, a significant issue arises due to insufficient understanding ability of LLM, resulting in formatting issues with SPARQL or errors in parsing relationship chains, leading to unsuccessful KG queries or inability to find answers. Besides, both categories rely excessively on the design of prompts and human experience.

To address the aforementioned challenges, we have introduced Self-Verification Enhanced Question Answering Agent (Self-VEQA Agent). Instead of filtering fact triples, our method only generate inference chains based on KG to address the problem of lengthy tokens. Besides, our method uses the concept of automatic self-verification to address the inadequacies in LLM reasoning abilities. Building upon a foundational LLM, our approach employs two agents: QA Agent and Verification Agent(Ver Agent).

QA Agent serves as a basic question answering module. The Ver Agent then verifies the answers

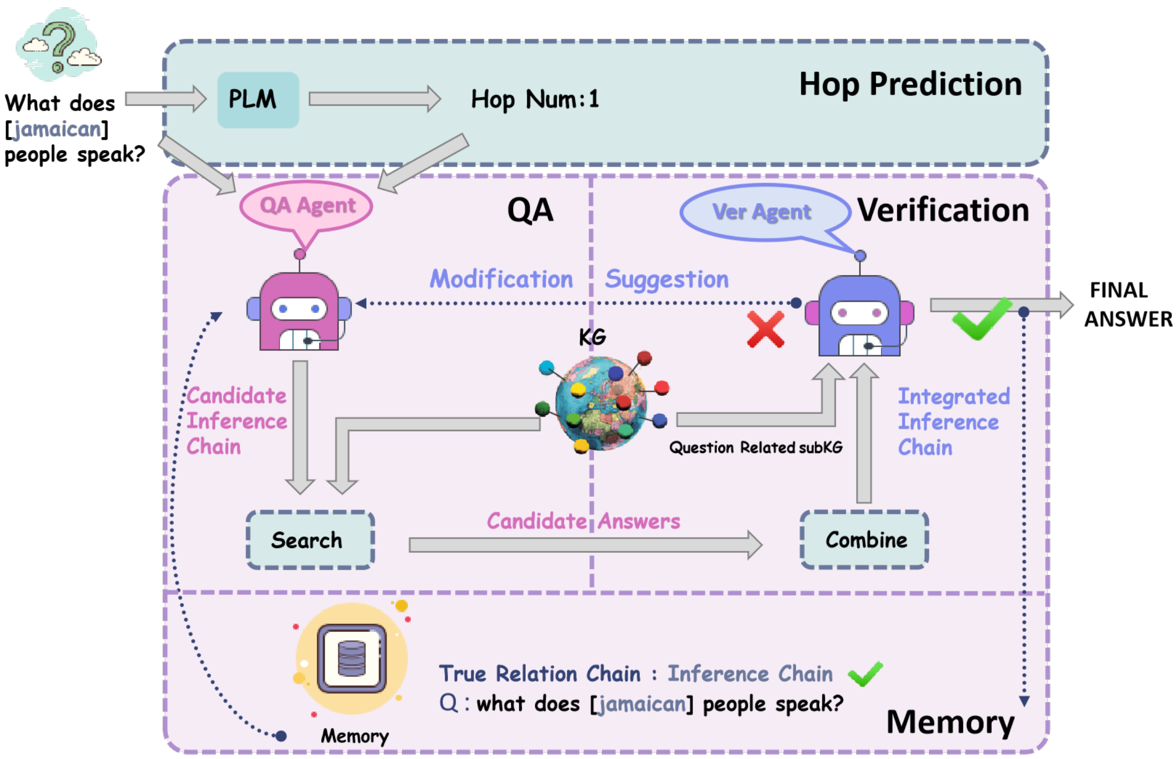

Figure 1: The structure of Self-VEQA Agent.

provided by the QA Agent to resolve incorrect reasoning problems. The two agents iterate through rounds of checking and adjusting answers until reaching a final satisfactory result or the maximum number of iterations is reached. This iterative process improves the accuracy of answers over time. Furthermore, our model has a memory mechanism, which enables it to possess dynamic evolution capabilities. This means that as the Self-VEQA Agent continues to perform tasks and accumulate experience, the accuracy of the QA Agent improves towards later stages of tasks.

Main contributions:

(1) We propose Self-VEQA Agent, which introduces agents to KGQA task. Self-VEQA Agent incorporates the Ver Agent to enhance the autonomy of the LLMs. This guidance from the Ver Agent helps the QA Agent generate more accurate answers.

(2) Self-VEQA Agent includes a memory mechanism that enables the model to evolve dynamically over time. As the Self-VEQA Agent performs tasks and gains experience, the QA Agent's accuracy improves progressively with each subsequent task. Experiments demonstrate that possessing a memory mechanism enhances the performance of the Self-VEQA Agent in subsequent tasks.

(3) Self-VEQA Agent, without the need for fine-tuning, outperforms traditional methods and most LLM-based approaches on the MetaQA and WebQSP datasets in KGQA task.

2 Related Work

2.1 PLM for Knowledge Graph Reasoning.

The KV-mem approach mainly adopts the idea of Traditional Key-value Memory Neural Networks (Xu et al., 2019), treating the answer-question pairs as key-value pairs and training them accordingly. This enables it to conduct interpretable reasoning for complex questions. EmbedKGQA (Saxena et al., 2020) utilizes knowledge graph embeddings to answer multi-hop natural language questions by training. Firstly, it learns the representation of the knowledge graph in the embedding space. Then, given a question, it learns a question embedding. Finally, it combines these embeddings to predict the answer. UniKGQA (Jiang et al., 2022) integrates retrieval and reasoning with a semantic matching module leveraging a pre-trained language model (PLM) for question-relation semantic matching.

While the mentioned methods have improved performance in KGQA tasks, they come with high time and resource costs for model training and have stringent requirements on datasets. Additionally,

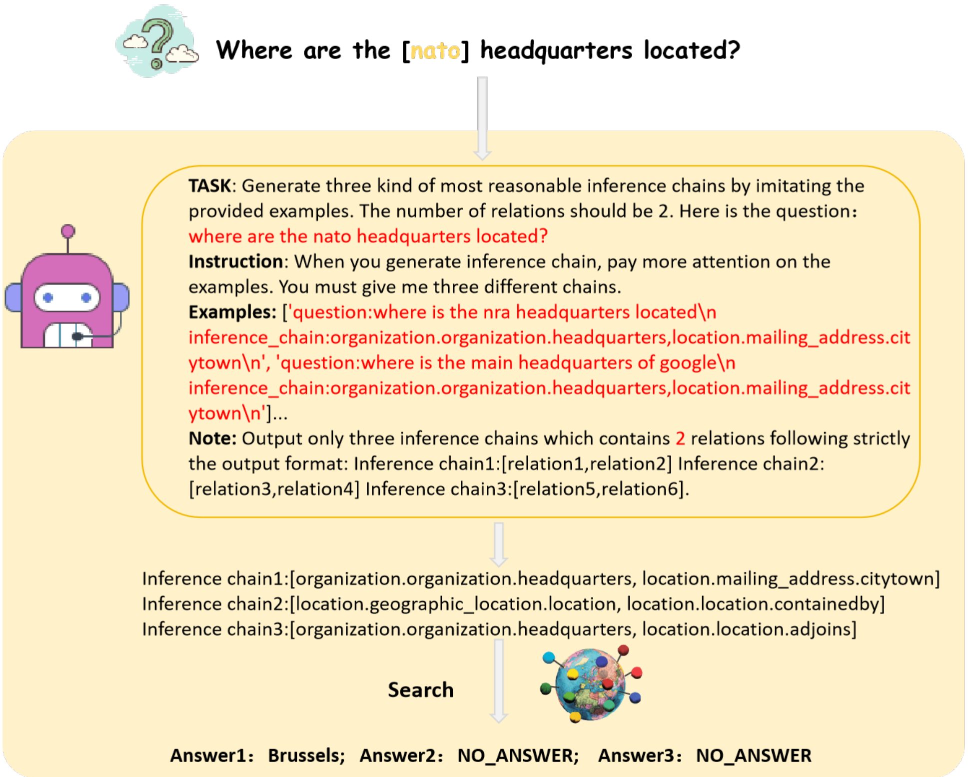

Figure 2: An example of QA Agent.

the trained models tend to lack generalizability.

2.2 Reasoning with Large Language Models

Although initially designed for text generation, LLM has demonstrated remarkable performance when applied to other subfields of natural language processing (Cheng et al., 2022). Particularly, the reasoning capability of LLM has garnered widespread attention in the field of artificial intelligence research (Arora et al., 2022; Sun et al., 2022). Some studies have explored LLM's performance in various reasoning skills, including arithmetic, logical, and commonsense reasoning. These outstanding performances make LLM an ideal reasoning tool for tasks in other domains (Clusmann et al., 2023).

2.3 LLM Agents

The utilization of LLMs for real-world tasks has emerged as an intriguing research area due to their human-like intelligence. While some studies leverage their linguistic capabilities, exploring LLMs as autonomous agents in specific scenarios offers diverse and promising applications. This approach aims to address issues like reliance on parameter settings and lack of adaptability in traditional agent-based simulations using rules or reinforcement learning. For instance, (Park et al., 2023) pioneered an LLM-powered agent framework to simulate human behavior in interactive scenarios, highlighting LLMs' potential to model complex social interactions and decision-making. BabyAGI[1] is a language model that interacts with a task list to automatically generate, prioritize, and execute tasks based on predefined objectives. Auto-GPT[2] uses GPT-4 to bridge AI "thinking" and autonomously attempts to achieve specified objectives by executing commands, pushing the boundaries of AI capabilities. In this paper, we adopt the agent concept to enhance LLM decision-making for improving KGQA task performance.

3 Method

Self-VEQA Agent contains four modules: Hop Prediction module, QA module, Verification module and Memory module as shown in Figure 1.

3.1 Hop Prediction Module

Hop Prediction involves estimating the number of hops needed to answer a question, guiding subsequent inference path prediction. Our work refers

[1] https://github.com/yoheinakajima/babyagi
[2] https://github.com/Significant-Gravitas/AutoGPT

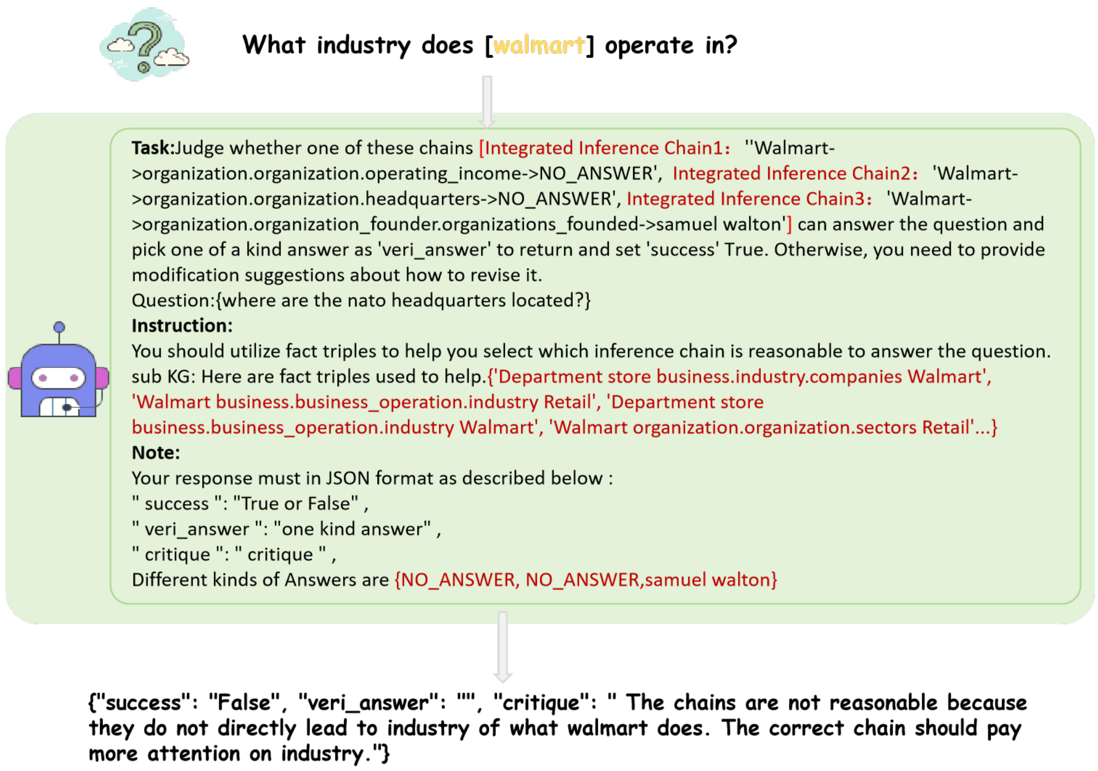

Figure 3: An example of Ver Agent. The prompt here only shows the crucial part.

to the work of (Wu et al., 2023). This process is framed as a classification task leveraging a Pre-trained Language Models (PLM) and a simple linear classifier. Through providing the number of hops, it will help QA Agent to determine inference chains more accurate. We fine-tune bert-base-uncased (Devlin et al., 2018) and a linear classifier on the training set of the datasets for hop prediction of WebQSP.

In the formula, Q represents the question; V_Q represents the embedding of the question; h_Q represents the predicted number of hops.

$$V_Q = PLM(Q) \tag{1}$$

$$h_Q = \underset{h}{argmax}\, P(h|V_Q), h \in 1, 2, ..., H \tag{2}$$

3.2 QA Module

Our method designs a QA Agent, which is responsible for generating candidate inference chains upon knowledge graph. To empower the QA Agent to generate more precise inference chains, few-shot selects from Memory module which is initialized by training set. And then based on candidate inference chains, we can obtain answers from knowledge graph.

The generated candidate inference chains and answers undergo verification by the Ver Agent. If it can select a reasonable one from the n-chains, the process ends. And the correct inference chain will be stored in Memory module. Otherwise, QA Agent will modify inference chains according to the modification suggestion from Ver Agent. The Memory module provides the QA Agent with experience, continuously enriching and improving its accuracy through usage. This demonstrates the QA Agent's dynamic evolution capability, as its accuracy steadily improves with each completed task and gained experience.

To delve deeper into the details in Figure 2, we start by chunking the various parts of the QA Agent prompt, which mainly composed of three parts: Task, Instruction, Note, shows below:

Task part primarily aims to clearly convey to the QA Agent what its main task is. Here in Figure 2 the number of hops refers to 2. The question, along with its corresponding number of hops, is fed into the QA Agent as external input.

For instruction part, here, examples are sourced from the Memory module, with the training set used for initialization purposes. And examples are composed of question and inference chain. Further-

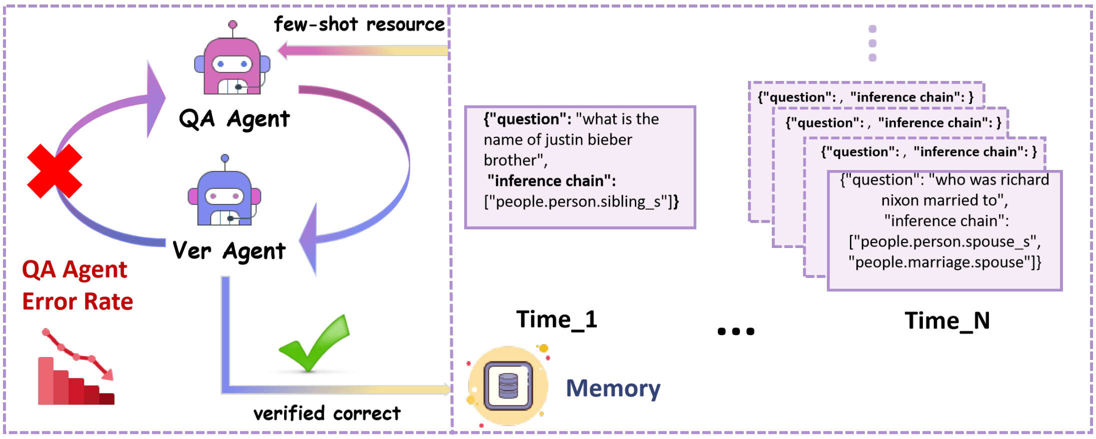

Figure 4: Memory Module.

more, this setup demonstrates the model's dynamic evolution capability, as it continuously learns from accumulated memory.

Note section imposes strict requirements on the output format as well as the specific output criteria. Then, the QA Agent generates n-inference chains. Combined with the head entity, we can identify n-candidate answers through searching the KG. Here head entity can be obtained from datasets.

And then candidate inference chains and corresponding answers will be combined together which are integrated inference chains. Based on these, we can derive n-integrated inference chains and sets of answers. An example of integrated inference chain is shown as Appendix A.1. These n-integrated inference chains, along with the set of answers and a relevant subKG related to the question will be input into the Ver Agent for answer selection. Here, the term "relevant subKG" refers to a subset of the knowledge graph (KG) that specifically pertains to the current question. This subset is determined through similarity calculations [3] with the questions, which help identify nodes and relationships within the KG that are closely related to the query at hand. By utilizing this relevant subKG, our method can effectively narrow down the scope of information needed for processing verification, thus reducing the overall number of input tokens and enhancing efficiency.

Complete prompt can be found in Appendix B.1.

3.3 Verification Module

To enhance the accuracy of answers that generated by the QA Agent, we have designed a Ver Agent. This agent primarily leverages the reasoning capabilities of LLMs to assess the coherence of the previously generated integrated inference chains from the preceding step. In other words, it determines whether the generated chain can effectively answer the given question. If it can, Ver Agent selects the final answer. If not, it provides modification suggestions explaining why it cannot answer, and this modification suggestions are passed to the QA Agent for regeneration. This process is repeated until the Ver Agent assesses the reasonableness of the answer or reach the maximum number of rounds.

To delve deeper into the details in Figure 3, we start by chunking the various parts of the QA Agent prompt, which mainly composed of three parts: Task, Instruction, Note.

Task part primarily aims to clearly convey to the Ver Agent what its main task is. Here, n-integrated inference chains will be passed to the Ver Agent.

Instruction part specifies the detailed aspects of the task and provides the reasoning basis for the Ver Agent and represents the core of this agent's function.

Note section imposes strict requirements on the output format as well as the specific output criteria.

If the Ver Agent deems any of these n-inference chains as reasonable, it will output the answer. Otherwise, it will generate revision suggestions and send them back to the QA Agent to regenerate inference chains. In this example, there are not reasonable inference chains present, so Ver Agent will give feedback about how to revise it.

A more specific example that interation between Ver Agent and QA agent can be found in Appendix A.2. Complete prompt can be found in Appendix B.2.

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Table 1: The Performance of Self-VEQA Agent and Baselines on MetaQA and WebQSP. The best result in each block is in bold. Δ refers to the performance decrease from two hops to three hops on MetaQA.

Type	Methods	MetaQA 2-HOP	MetaQA 3-HOP	Δ	WebQSP	
		Hit@1	Hit@1	Hit@1	F1	Hit@1
Traditional Method	KV-Mem	82.7	48.9	-33.8	34.5	46.7
	EmbedKGQA	98.8	94.8	-4.0	-	66.6
	NSM	**99.9**	98.9	-1.0	62.8	68.7
	UniKGQA	99.0	99.1	+0.1	70.2	75.1
LLM-based Method	ChatGPT	31.0	43.2	-12.2	-	61.2
	StructGPT(ChatGPT)	97.3	87.0	-10.3	-	72.6
	KnowledgeNavigator	99.5	95	-4.5	-	**83.5**
Autonomous Agent	Self-VEQA Agent	99.7	**99.6**	-0.1	**70.5**	82

3.4 Memory Module

Due to the transient memory mechanism of LLMs, we propose Memory module shown as Figure 4 as the storage space for their experiences.

Its primary function is to store inference chains and questions validated as correct by the Ver Agent. This accumulation enriches the Memory module, serving as an experience repository for subsequent few-shot selections by the QA Agent. Memory module will be continuously updated with new data, thus ensuring a continuous update process. As Memory module accumulates and is continuously updated with new data, the QA Agent's capabilities gradually improve over time.

4 Experiment

4.1 Dataset

We conducted evaluations on both representative small- and large-scale graphs along with their corresponding KGQA datasets.

MetaQA: (Zhang et al., 2018) is a substantial KGQA dataset in the movie domain, featuring a knowledge graph with 43,000 entities, 9 relations, and 135,000 triples. It includes 407,000 questions requiring 1-hop to 3-hop reasoning from head entities. Each question consists of a head entity, a relation reasoning path, and answer entities. To evaluate Self-VEQA Agent's multi-hop reasoning capabilities, we focus on the 2-hop and 3-hop datasets within MetaQA for our experiments.

WebQSP: is a benchmark with a smaller set of questions but a large-scale knowledge graph. It includes up to 2-hop questions on Freebase(Bollacker et al., 2008), each with a topic entity, constraints, inferential chains, and SPARQL queries to find answers. We used the latest Freebase data dumps from Google[4], containing 3.12 billion triples as of 2023. WebQSP has 4,737 questions, but we excluded 11 without gold answers.

4.2 Evaluation Metric

Consistent with prior studies, we employ Hits@1 and F1 as the evaluation metrics. Hits@1 measures the percentage of questions for which the top-1 predicted answer is accurate. Recognizing that a question might have multiple correct answers, F1 takes into account the coverage of all answers, striking a balance between the precision and recall of the predicted responses.

4.3 Baselines

To assess the effectiveness of Self-VEQA Agent, we conduct a comparative analysis against a collection of established baseline models in the KGQA field on WebQSP and MetaQA. These baselines can be categorized into traditional methods, which do not incorporate LLM, and LLM-based methods. Traditional methods' baselines include KV-Mem (Xu et al., 2019), EmbedKGQA (Saxena et al., 2020), NSM (He et al., 2021), UniKGQA (Jiang et al., 2022). All of these baselines were evaluated on both MetaQA and WebQSP. In addition, we add StructGPT (Jiang et al., 2023) and KnowledgeNavigator (Jiang et al., 2023) as the baseline models for KGQA tasks leverage LLM. Both of these frameworks rely on un-fine-tuned LLM for knowledge retrieval and question reasoning. The current LLM-based approaches involve a one-time prompt without a process for autonomous decision-making.

[4] https://developers.google.com/freebase/data

4.4 Implementation Details

We use the closed-source GPT-3.5-turbo model via the OpenAI API. The temperature parameter is set to 0.0 for reproducibility, with a context length of 4096 and a maximum of 2000 tokens per output sequence. In all experiments, the maximum number of interactions between the QA Agent and Ver Agent is set to 2.

4.5 Main Results

Table 1 shows the performance of Self-VEQA Agent and baseline on KGQA datasets.

MetaQA 2-Hop dataset is relatively simple, so both traditional methods and LLM-based methods can achieve relatively optimal performance. Many of them have reached 99+. Self-VEQA Agent outperforms most traditional methods and LLM-based methods.

However, it is worth noting that Self-VEQA Agent's performance on MetaQA 2-Hop questions falls short compared to NSM. This is primarily because the MetaQA dataset comprises fewer than 300 template questions but includes over 100,000 training instances, enabling models to be extensively trained, thus achieving relatively high performance. However, our approach mainly relies on LLMs and does not involve specific fine-tuning for the particular downstream task.

From the experimental results, we can observe that for most models, including traditional ones and those based on LLMs, the performance on the MetaQA dataset tends to decrease when transitioning from 2-hop to 3-hop questions. This is primarily because, for the MetaQA dataset, correctly reasoning through 3-hop questions is more challenging than 2-hop questions. The heightened challenge stems from 3-hop questions, which entail reasoning chains comprised of three relationships. This complexity makes it more challenging to deduce the accurate sequence and relationships. For LLM-based methods without fine-tuning, such as KnowledgeNavigator, the performance tends to decrease as the number of hops increases. This is because KnowledgeNavigator primarily relies on knowledge enhancement methods. As the number of hops increases, the necessary knowledge also grows exponentially. Consequently, this amplifies the difficulty of LLM-based reasoning, resulting in a decline in performance compared to 2-hop questions. Unlike other methods that treat retrieval and reasoning as separate stages, UniKGQA proposes a unified model for both processes, effectively transferring relevant information from the retrieval stage to the reasoning stage. Therefore, its performance remains consistent even with three hops.

In contrast, our method shows almost no change in performance on the MetaQA dataset when transitioning from 2-hop to 3-hop questions. The reason why our method's performance on 3-hop questions does not decrease significantly compared to 2-hop questions is mainly due to the design of our verification agent, which validates the rationality of generated reasoning chains and answers. Additionally, with accumulated task experience, the error rate of generated reasoning chains decreases over time, leading to overall performance improvement.

Compared to MetaQA, the WebQSP dataset presents more complex relationship chains and includes questions with constraints just as described earlier. However, the maximum hop for questions in this dataset is 2. Therefore, the main challenge for this dataset lies in identifying the correct relationship chains and corresponding constraints. Our method outperforms traditional models by a significant margin, primarily because the WebQSP dataset encompasses more template questions but includes less training instances compared to MetaQA, and some relationship chains have similar and diverse names, making it challenging for traditional models to adequately learn. Compared to methods based on LLMs, our approach also achieves better performance than most models. This is mainly because we not only simply utilize internal knowledge from LLMs or simply input relevant external knowledge but also propose two agents to interact with each other, thereby enhancing performance.

However, our method lags behind KnowledgeNavigator by approximately 1.5% in performance on the WebQSP dataset. This is mainly because KnowledgeNavigator employs knowledge enhancement techniques, providing more accurate factual triplets, which leads to better answers for constrained questions and thus slightly better overall QA performance than ours. In contrast, our approach primarily relies on a Ver Agent to enhance the accuracy of the QA Agent, leveraging its reasoning capability, but there is room for improvement in addressing constrained questions.

4.6 Ablation Experiment

Here, we conducted an ablation study on Self-VEQA Agent to assess the influence of the Ver

Table 2: Ablation experiment of Self-VEQA Agent on MetaQA and WebQSP.

Models	**MQA 2-HOP**	**MQA 3-HOP**	**WebQSP**	
	Hit@1	Hit@1	F1	Hit@1
Self-VEQA Agent	99.7	99.6	70.5	82
-mem module	99.0	99.4	70.2	79.9
-ver module	98.9	98.7	63.9	74.3

Table 3: The Influence of Interaction Rounds on WebQSP.

the Number of Rounds	**WebQSP**	
	F1	Hit@1
Rounds=1	69.8	80.4
Rounds=2	70.5	82
Rounds=3	70.1	81.1
Rounds=4	69.8	80.4
Rounds=5	69.6	80.2

Agent and Memory module by removing each one individually. Table 2 shows the results of the ablation study, in which -ver module and -mem module stand for removing verification module and removing Memory module. All these variants underperform the complete Self-VEQA Agent, which indicates that the two strategies are both important for enhancing KGQA performance. But -ver module is more significant for Self-VEQA Agent. The Ver Agent has a greater impact on the WebQSP dataset compared to MetaQA, because the inference chains in the WebQSP dataset are relatively challenging. It is difficult to generate the correct inference chain correctly on the first attempt.

4.7 Other Analysis

4.7.1 Interaction Rounds Analysis

We also analyzed the impact of setting different numbers of interaction rounds between the QA Agent and Ver Agent on the performance of the Self-VEQA Agent. Experimental results indicate that increasing the number of interaction rounds does not lead to better performance. For the WebQSP dataset, the best performance was observed when the round was set to 2. This is because for problems with fewer hops, interacting twice is usually sufficient to obtain the correct result. Excessive interaction rounds can instead lead to a decrease in performance. In this context, considering the amount of data and the complexity of the two datasets, WebQSP is representative, thus we exclusively carry out this experiment using this dataset.

Table 4: The Impact of Inference Chain Quantity on WebQSP.

Inference Chain Quantity	**WebQSP**	
	F1	Hit@1
num=1	65.5	76.9
num=3	70.5	82
num=5	69.8	81.2
num=10	68.9	79.4

4.7.2 Inference Chain Quantity Analysis

We also analyzed the impact of setting different numbers of inference chain generated by QA Agent. The experimental results indicate that setting the number of inference chains generated by QA to 3 achieves the best performance. Increasing the number of generated chains, for example, generating five or ten, actually decreases the overall task performance. This is because an excessive number of generated inference chains introduces too much noise when the Ver Agent makes rational judgments, thereby impacting its performance. Conversely, from the experimental results, it can be observed that the fewer inference chains generated by the QA Agent, the poorer its performance. Given the volume of data and the complexity of the two datasets, we have selected WebQSP as our representative dataset for this experiment.

5 Conclusion

This paper introduces the Self-VEQA Agent for Knowledge Graph Question Answering, addressing limitations of traditional methods and recent approaches involving large language models. Our method generates inference chains to avoid lengthy token issues and employs automatic self-verification to enhance LLM reasoning accuracy. The Self-VEQA Agent, comprising a QA Agent and a Verification Agent, iterates through verifying and adjusting answers, leading to improved accuracy. Additionally, the model features a memory mechanism for dynamic evolution, enhancing performance over time.

Limitations

We validated the effectiveness of each module through ablation experiments. However, there is still room for improvement in the performance of the Ver Agent. While we don't have a mature solution yet, future work could focus on futher enhancing the Ver Agent's performance. This could involve using more advanced large language models and further exploring their reasoning capabilities to achieve better results.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, and Christopher Re. 2022. Ask me anything: A simple strategy for prompting language models. In *The Eleventh International Conference on Learning Representations*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

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.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

Jan Clusmann, Fiona R Kolbinger, Hannah Sophie Muti, Zunamys I Carrero, Jan-Niklas Eckardt, Narmin Ghaffari Laleh, Chiara Maria Lavinia Löffler, Sophie-Caroline Schwarzkopf, Michaela Unger, Gregory P Veldhuizen, et al. 2023. The future landscape of large language models in medicine. *Communications medicine*, 3(1):141.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Tiezheng Guo, Qingwen Yang, Chen Wang, Yanyi Liu, Pan Li, Jiawei Tang, Dapeng Li, and Yingyou Wen. 2023. Knowledgenavigator: Leveraging large language models for enhanced reasoning over knowledge graph. *arXiv preprint arXiv:2312.15880*.

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.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*.

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023. Kg-gpt: A general framework for reasoning on knowledge graphs using large language models. *arXiv preprint arXiv:2310.11220*.

Navid Madani, Rohini K Srihari, and Kenneth Joseph. 2023. Domain specific question answering over knowledge graphs using logical programming and large language models. *arXiv preprint arXiv:2303.02206*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4498–4507.

Tian-Xiang Sun, Xiang-Yang Liu, Xi-Peng Qiu, and Xuan-Jing Huang. 2022. Paradigm shift in natural language processing. *Machine Intelligence Research*, 19(3):169–183.

Tilahun Abedissa Taffa and Ricardo Usbeck. 2023. Leveraging llms in scholarly knowledge graph question answering. *arXiv preprint arXiv:2311.09841*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Yike Wu, Nan Hu, Guilin Qi, Sheng Bi, Jie Ren, Anhuan Xie, and Wei Song. 2023. Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering. *arXiv preprint arXiv:2309.11206*.

Kun Xu, Yuxuan Lai, Yansong Feng, and Zhiguo Wang. 2019. Enhancing key-value memory neural networks for knowledge based question answering. 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 2937–2947.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

A Example Appendix

A.1 Example of Integrated Inference Chain

It is shown as Figure 5.

A.2 Example of Self-VEQA Agent

In cases where the QA Agent generates an incorrect inference chain, the Ver Agent plays a crucial role in identifying and flagging the error as shown in Figure 6. It provides reasons for considering the inference chain incorrect. In this example, Ver Agent points out instances where the second relationship fails to explicitly indicate the ultimate relationship necessary for answering the question. Subsequently, the QA Agent is tasked with modifying and correcting the relationship chain to ensure accuracy.

B Prompt Appendix

B.1 Prompt of QA Agent

TASK: Generate three kind of most reasonable inference chains by imitating the provided examples. The number of relations should be {the number of hops}. Here is the question:{question}

Instruction: When you generate inference chain, pay more attention on the examples examples. You must give me three different chains.

Examples: {examples}

Note: Output only three inference chains which contains {the number of hops} relation following strictly the output format:

Inference chain1 [relation1,relation2]

Inference chain2: [relation3,relation4]

Inference chain3:[relation5,relation6].

B.2 Prompt of Ver Agent

TASK: Judge whether one of these chains {chain_list} can answer the question and pick one of a kind answer as 'veri_answer' to return and set 'success' True.

Question:{question}

Instruction: Pay more attention on the inference chains. You should utilize fact triples to help you select which inference chain is reasonable to answer the question. Pick one and put only answer in "veri response" without any useless word such as 'and', and just use semicolon to separate different answers, otherwise directly set "success" false and give reasons why the relation chains cannot answer the question. There is no need to answer the question utilizing your internal knowledge. But you can judge whether the answer is correct to answer the question utilizing your internal knowledge. Don't apology!

Note: Your response must in JSON format as described below:

"success": "True or False",

"veri_answer": "one kind answer or answer it utilizing KnowledgeGraph",

"critique": "critique",

Ensure the response can be parsed by Python 'json . loads' , e.g.: no trailing commas, no single quotes, all contents should be strings etc. Pay more attention on the output format, which is important, otherwise, it will influence latter work.

When you pick one kind of answers from different kinds of Answers, you should choose all answers from only one kind you pick.

sub KG: Here are fact triples used to help. {fact_triples}.

Different kinds of Answers: {answer1, answer2, answer3}.

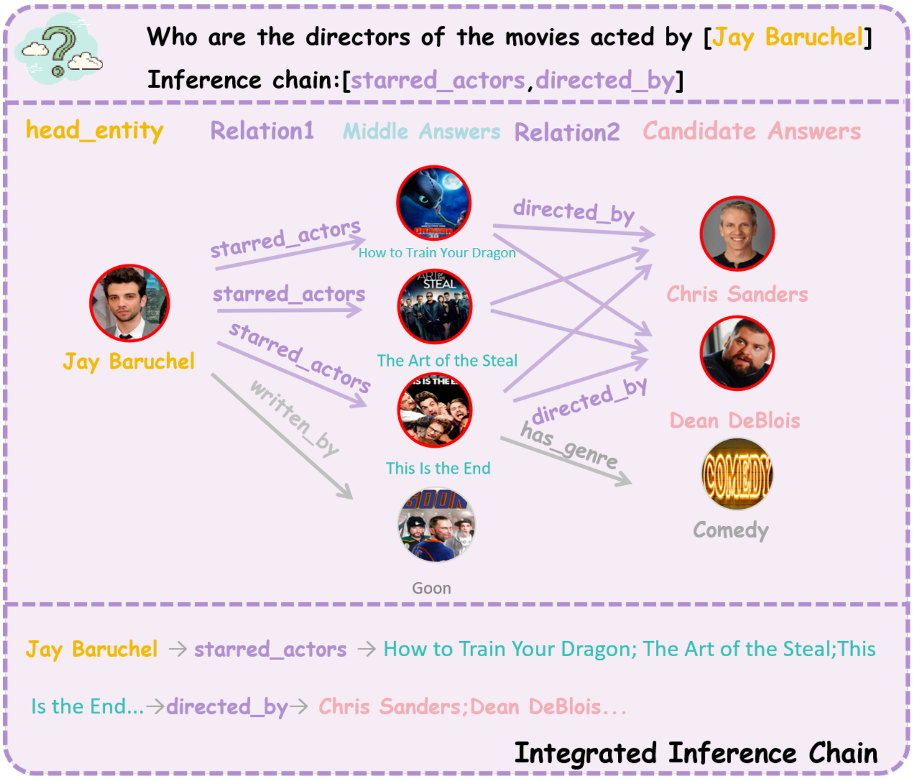

Figure 5: an example of integrated inference chain.

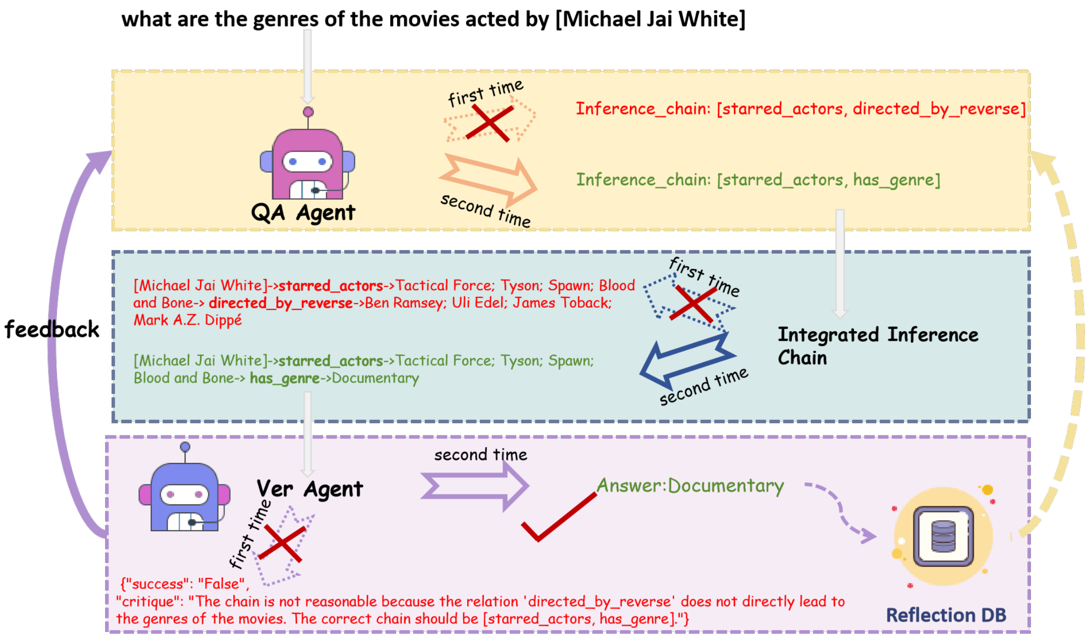

Figure 6: Ver Agent provide revision suggestion and QA Agent revises inference chain.