QueryAgent: A Reliable and Efficient Reasoning Framework with Environmental Feedback based Self-Correction

Anonymous ACL submission

Abstract

Employing Large Language Models (LLMs) for semantic parsing has achieved remarkable success. However, we find existing methods fall short in terms of reliability and efficiency when hallucinations are encountered. In this paper, we address these challenges with a framework called QueryAgent, which solves a question step-by-step and performs step-wise self-correction. We introduce an environmental feedback-based self-correction method called ERASER. Unlike traditional approaches, ERASER leverages rich environmental feedback in the intermediate steps to perform selective and differentiated self-correction only when necessary. Experimental results demonstrate that QueryAgent notably outperforms all previous few-shot methods using only one example on GrailQA and GraphQ by 7.0 and 15.0 points. Furthermore, our approach exhibits superiority in terms of efficiency, including runtime, query overhead, and API invocation costs. By leveraging ERASER, we further improve another baseline (*i.e.,* AgentBench) by approximately 10 points, validating the strong transferability of our approach.

1 Introduction

Recent advances in employing Large language models (LLMs) on various tasks have exhibited impressive performance (Brown et al., 2020; OpenAI, 2023). Among these tasks, Knowledge Base Question Answering (KBQA), which aims to answer questions over KB, has emerged as a critical and complex challenge, serving as an ideal testbed for assessing the reasoning capabilities of LLMs over structured data (Gu et al., 2023).

However, despite their remarkable achievements, we find that existing LLM-backend KBQA methods fall short in both reliability (the credibility of results) and efficiency (*i.e.,* running time, query times, and API invocation cost). Following the popular In-Context Learning (ICL) paradigm, Li et al. (2023); Nie et al. (2023) generates the target query with few-shot demonstrations. They consider LLMs as a black box and complete a complex task in one go. As a result, it lacks interpretability and is prone to hallucination (Yao et al., 2023), leading to lower accuracy of the top-1 candidate. To alleviate these issues, they employ beam search and self-consistency (Wang et al., 2023). However, these also result in numerous unreliable candidates, thus increasing the running time and query times. Typically, it requires querying thousands of SPARQL and several minutes to obtain the answer.

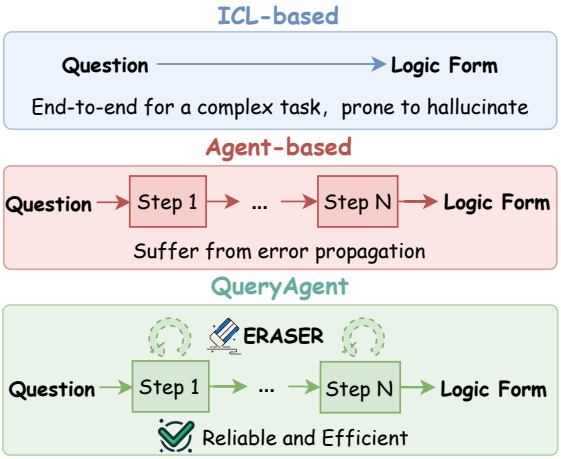

Figure 1: QueryAgent compared with two mainstream KBQA paradigms employing LLMs.

For a complex task, solving it step-by-step has emerged as a promising solution (Wei et al., 2022; Zhou et al., 2023). AgentBench (Liu et al., 2024) implements an Agent-based (Yao et al., 2023) KBQA system by progressively invoking tools to build the target query. However, its iterative nature dictates that each step strictly relies on the previous steps. When hallucination occurs, subsequent reasoning processes would be built upon erroneous foundations, resulting in unreliable candidates and meaningless resource wastage. Additionally, the necessity to invoke an LLM at each step renders

beam search unaffordable, placing a high demand on the accuracy of the top-1 results of each step. In our preliminary experiments, we observed that 35% of the questions in AgentBench suffer from various hallucinations. As a result, AgentBench achieves unsatisfactory performance, only 57% of the state-of-the-art ICL-based methods on GrailQA.

In view of these challenges, we introduce a framework called **QueryAgent** to explore more reliable and efficient reasoning in complex environments. Specifically, QueryAgent models KBQA as a multi-turn generation task to step-by-step construct the target query with tools and perform stepwise self-correction. To mitigate the error accumulation issue of multi-step reasoning, we propose an environmental feedback-based method called **ERASER** (**E**nvi**R**onmental feedb**A**ck **SE**lf-co**R**rection). For each LLM generated text, ERASER infers whether it is erroneous and analyzes the possible causes based on the feedback from environments (*e.g.,* KB execution results, Python interpreter execution status, previous reasoning history) in the intermediate steps. Upon analyzing this feedback, ERASER provides the system with potential causes of errors and general guidelines for correction. Based on the guidelines, LLM can reconsider and correct the erroneous result.

Unlike previous self-correction methods (Pourreza and Rafiei, 2023; Chen et al., 2023) which purposelessly correct every generated result with the same few-shot demonstrations, the idea of ERASER is to actively identify and differentiate different errors based on the rich environmental feedback in the intermediate reasoning steps and then provide tailored guidelines for the distinct error type. With the help of environmental feedback, ERASER has a more solid basis for more precise detection, analysis, and correction, rather than relying solely on the final answer. Moreover, ERASER distinguishes between different types of errors, allowing it to provide guidelines specifically tailored to each error scenario. This targeted approach makes ERASER more purposeful and scalable. In situations where there are numerous potential error scenarios, the guideline of different errors can be independently developed without the need to encode all possible error cases to the prompt.

We conduct extensive experiments to evaluate the effectiveness of QueryAgent and ERASER. With only 1 example, QueryAgent notablely surpasses all few-shot methods, which require up to 100 shots, on GrailQA (+7.0), GraphQ (+15.0), WebQSP (+3.4), and MetaQA (+2.0). While achieving superior performance, our approach exhibits significant efficiency improvements. Compared to ICL-based methods, QueryAgent reduces runtime and query overhead to several orders. Compared to Agent-based methods, QueryAgent allows for approximately a 50% reduction in API invocation costs and runtime. These results highlight our proposed method is both reliable and efficient. We also evaluate QueryAgent on a Text2SQL dataset (WikiSQL), and adapt ERASER to another system (AgentBench), to demonstrate their versatility. Results reveal that QueryAgent outperforms the previous 32-shot method by 6.9 points. Besides, ERASER relatively yields an additional improvement for AgentBench by 26% and 42% in F1 on the GrailQA and GraphQ, respectively [1].

2 Related Work

2.1 Few-shot KBQA

Recent advances in adopting LLMs for few-shot KBQA can be broadly categorized into 3 groups:

1) **ICL-based** KB-BINDER (Li et al., 2023) and KB-Coder (Nie et al., 2023) implement an ICL-based system by taking dozens of annotated examples into the prompt. Since they model this complex task as a simple end-to-end generation process, LLMs are directly confronted with a large search space and thus more likely to generate unreliable results. Although they incorporate beam search and self-consistency to increase the likelihood of encompassing the correct logic form, these also introduce the need to process a large number of candidates. On average, to solve a question, it takes executing thousands of candidate queries and several minutes to obtain the final answer

2) **IR-based** Starting from an entity, StructGPT (Jiang et al., 2023), and ToG (Sun et al., 2023) iteratively walk on the graph, selecting the next neighboring entity to jump to, until finding the answer. Compared with the methods that generate an executable query, these methods can only solve questions whose reasoning process can be modeled as a single, non-branching chain. They cannot model questions with multi-constraints whose reasoning process is a tree or graph. As they traverse in the KG to obtain the answer, they have limitations on questions whose answer is not an entity in the KG (*e.g.,* aggregation or boolean question).

[1] Our code will be released after acceptance

3) **Agent-based** AgentBench (Liu et al., 2024) utilizes some pre-defined SPARQL templates to solve the question step-by-step, including acquiring the one-hop relation, merging two reasoning paths, adding aggregation, and so on. For a complex task, solving it step by step aligns with human intuition and helps reduce the potential search space. However, at each step, AgentBench heavily relies on the previous results, hence demanding high precision. We observe that AgentBench encounters various unexpected outputs during reasoning, leading to serious error accumulation. When hallucinations arise in the preceding steps, the subsequent become meaningless or unreliable. These factors contribute to inferior performance, which is only half as effective as the ICL-based methods.

In this work, based on the agent paradigm, we propose a reliable and efficient framework called QueryAgent, and alleviate LLM's hallucination by introducing a self-correction method.

2.2 Self-Correction

As the concern persists in the accuracy and appropriateness of LLM's generated content, self-correction has been proposed as a remedy to these issues (Pan et al., 2023). DIN-SQL (Pourreza and Rafiei, 2023) utilizes a zero-shot prompt to rectify errors in the generated SQL queries. The prompt asks the LLMs to examine the generated SQL queries for potential errors and correct them, while skipping those that are deemed error-free. Such intrinsic self-correction, which is solely based on LLMs' inherent capabilities without the crutch of external feedback, fails to achieve significant improvement and is unreliable (Huang et al., 2023a). An intuitive improvement would be to incorporate few-shot demonstrations in the prompt (Chen et al., 2023). However, this would result in longer prompts, and can only cover a limited number of scenarios. Since they indiscriminately apply the same prompt to all cases, LLMs may be confused about which example fits the current situation. Some works like SALAM (Wang and Li, 2023) train a model to retrieve the most similar error case. Even so, it still can not ensure precise error discrimination and is heavyweight. Besides, the above methods overlook the rich feedback that the environment (*e.g.,* KB, DB) can provide for error correction. These approaches rely solely on the final output as the basis for error correction, presenting substantial challenges for LLMs to make accurate judgments.

To address the above issues, we propose ERASER, an environmental feedback based self-correction method. Based on the feedback from the environment in the intermediate steps, ERASER proactively identifies when errors arise and provide tailored guidelines.

3 Method

3.1 Overview

In this work, we model KBQA as a semantic parsing task. We propose an LLM-backed symbolic agent called QueryAgent which step-by-step constructs formal queries with tools and performs stepwise self-correction. The process of QueryAgent can be divided into two parts: Query construction and Self-Correction (ERASER). At each step, QueryAgent first generates the action to be taken in this step, then leverages environmental feedback to identify and distinguish potential errors. If an error is detected, the system provides tailored guidelines to help LLMs perform error correction.

The reliability and efficiency of QueryAgent are reflected in the following aspects. For reliability: 1) It solves questions progressively rather than directly outputting the complete answer. 2) We introduce a correction process, namely ERASER, during reasoning. 3) ERASER is purposeful and more precise. For efficiency: 1) The high accuracy of our top-1 candidate eliminates the need for beam search and self-consistency. 2) Self-correction reduces meaningless reasoning along erroneous paths. 3) We perform self-correction only when necessary and only incorporate related guidance to the prompt.

3.2 Query Construction

To interact with KB and step-by-step build a target query, we leverage PyQL (Huang et al., 2023b) to systematically construct the workflow. PyQL is a management toolset designed for query building over knowledge bases, including various tools to incorporate clauses to the final executable query (*i.e.,* SPARQL) , such as adding triple patterns, filters, aggregations, etc. As the final query can be transformed from a sequence of PyQL functions, our objective is to generate these functions incrementally during the multi-turn interaction.

As shown in Figure 2, at each step, the LLM provides its *thoughts* over the current step and suggests the next *action* to be taken. The *action* is a PyQL function, we execute it to get the execu-

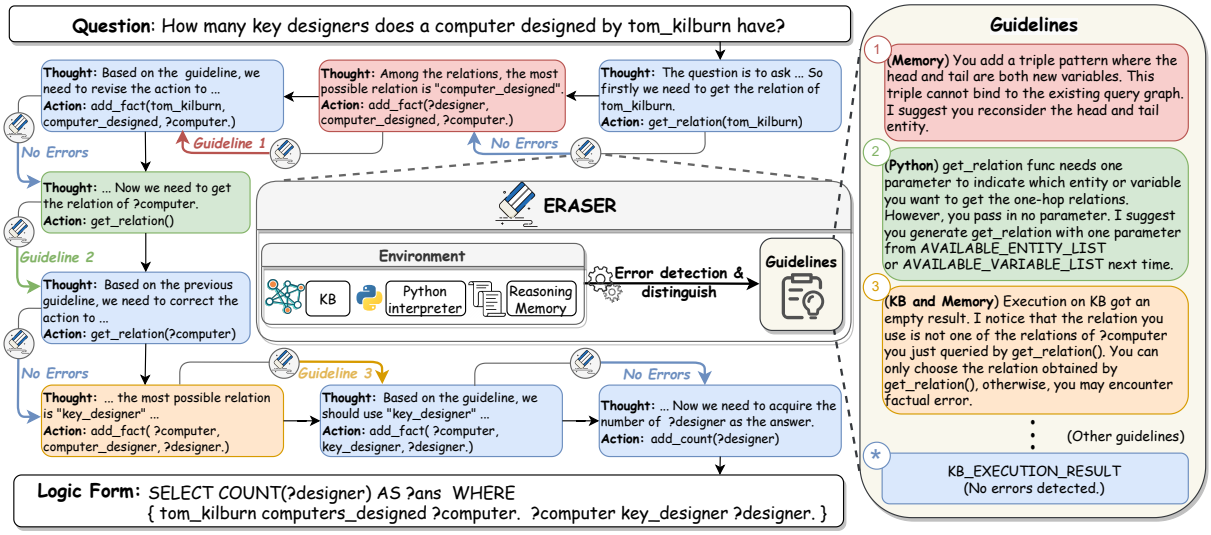

Figure 2: An example of QueryAgent and ERASER.

tion result as the *observation* from the environment. For the example in Figure 2, the LLM suggests firstly we need to obtain the one-hop relations of "tom kilburn" (*thought*) and the function `get_relation(tom_kilburn)` should be invoked at this step (*action*). By executing this function, we obtain relations around "tom kilburn" for the next step (*observation*). This process is iteratively repeated. When the reasoning process concludes we execute the generated query to obtain the answer. Given that each step corresponds to an executable query, we can easily observe the result of the current reasoning process, similar to how humans progressively write, execute, and validate a query.

The prompt consists of four parts: the task description, the document of available functions, a running example, and the new question. We first provide an overview of the task and the rules that must be followed. Then we provide a brief document of all available functions. Following that, we present a detailed step-by-step reasoning process for an example question. Finally, we concatenate the new question that needs to be solved at the end.

3.3 ERASER

In this section, we propose an environmental feedback based self-correction method (ERASER). The key ideas underlying ERASER are to let the environment "speak out" and distinguish different types of errors. We require the system to provide feedback on its current status and any encountered errors. Based on this feedback, we attempt to identify what types of errors arise and then provide targeted and valuable guidance.

The feedback mainly originates from three environments: Knowledge Base (KB), Python Interpreter, and Reasoning Memory. For example, KB can provide feedback such as: whether the executed result is empty, if the reasoning process ends with a blank node (CVT) or multiple variables, error messages from the query engine, and so on. For Python interpreters, it can provide error messages of various invalid function calls (*e.g.,* not enough values to unpack). For reasoning memory, we can access information including but not limited to: what steps have been taken, what variables have been created, and the executed result of the previous steps.

By analyzing the above feedback, we can detect some errors and determine the cause of them. As illustrated in Figure 2, an error is raised by the Python interpreter at the fourth step due to insufficient parameters in the generated action. In the sixth step, the query engine yields an empty result after a triple pattern constraint is added. Since we have acquired the relation of "?computer" but the used relation is not any of them, it is likely an incorrect relation was chosen in the previous steps. This example also showcases the importance of leveraging various feedback from different environments for error distinction. For instance, whether or not the system has obtained the relationship of the head/tail entity can be indicative of two distinct causes of error, but they both manifest as empty results in the execution. Compared with the previous methods only focus on the final answer, this rich environmental feedback in the intermediate steps can serve as crucial observational points for

detecting and distinguishing various errors.

The guidance in ERASER typically is some speculation about possible causes of error and general suggestions. Examples are shown in the right part of Figure 2. Compared to some code generation work which simply returns the original system error message (Chen et al., 2023), the guidance provided in the prompt can be seen as an intermediate language. It shields the LLM from directly considering the original error, instead focusing on easier-to-comprehend guidance, which ultimately contributes to a successful correction. Besides, by injecting the guidelines into the reasoning process, ERASER has no need for designing another specific module or agent to perform self-correction.

In this manner, the system designers only need to figure out how to identify potential errors from various environmental feedback and then provide modification suggestions for each type of error. To summarize, ERASER has the following advantages: 1) **Purposeful and Precise**: ERASER has the ability to detect errors. For each error, it provides tailored guidelines that relate to the current situation. 2) **Independent and Scalable**: The triggers for each type of error are independent of others. It provides convenience for incremental development without affecting the results of other questions. 3) **Lightweight and Economical**: Invocation of the LLM occurs exclusively when an error is detected. The correction prompt is a general guideline rather than lengthy few-shot examples.

4 Experiment

4.1 Datasets

We experiment with four KBQA datasets. The statistics can be found in Table 1. For GrailQA, we report the performance of the dev set to stay within our budget. For other datasets, we report the performance of the test set.

GRAILQA (Gu et al., 2021) is a large-scale complex dataset that evaluates three levels of generalization (*i.e., i.i.d., compositional*, and *zero-shot*)

GRAPHQ (Su et al., 2016) is a particularly challenging dataset given that it exclusively focuses on non-i.i.d. generalization. In this paper, we use the processed version by (Gu and Su, 2022).

WEBQSP (Yih et al., 2016) is a simple KBQA dataset with questions from Google query logs. It mainly tests i.i.d. generalization.

METAQA (Zhang et al., 2018) consists of 1-3 hops question based on Wiki-Movies KG. We experiment on the most difficult 3-hop subset (denoted as MetaQA-3Hop).

Dataset	Training	Dev	Test
GRAILQA	44,337	6,763	13,231
GRAPHQ	2,381	-	2,395
WEBQSP	3,098	-	1,639
METAQA-3HOP	114,196	14,274	14,274
WIKISQL	56,355	8,421	15,878

Table 1: Statistics of experiment datasets.

4.2 Baselines

We compare QueryAgent with fine-tuning and few-shot KBQA method. For simplicity, we mainly introduce the few-shot method here.

KB-BINDER (Li et al., 2023) is an ICL-based KBQA method utilizing dozens of (Question, S-expression) pairs as examples.

KB-Coder (Nie et al., 2023) converts the s-expression to a sequence of function calls thus reducing the format error rate.

Pangu (Gu et al., 2023) is a general framework with experiments on both fine-tuning and few-shot settings. For the few-shot setting, Pangu also adopts the ICL paradigm.

AgentBench (Liu et al., 2024) proposes an agent-based baseline by modeling the KBQA as a multi-turn open-ended generation task.

4.3 Experimental Setup

We use `gpt-3.5-turbo` for our experiment by default. All datasets use F1 as the evaluation metric. For baseline methods with the same setting, we report the performance from their original paper. KB-BINDER uses `Codex` which has been deprecated. For a fair comparison, We report the performance reproduced by KB-Coder with `gpt-3.5-turbo`. For KB-BINDER and KB-Coder, we compare the setting without similarity retrieval since it is not a strictly few-shot setting that requires the whole annotated training set can be accessed. AgentBench reports the performance on a mixed subset and uses golden linking results. We reproduce AgentBench with the same entity linking result as ours. We also implement the one-shot setting of KB-BINDER critically based on their public code.

4.4 Main Result

As shown in Table 2, with only one example, our method outperforms all few-shot methods that re-

Methods	GrailQA	GraphQ	WebQSP	MetaQA-3Hop
fine-tuning				
ArcaneQA (Gu and Su, 2022)	73.7	31.8	75.6	-
TIARA (Shu et al., 2022)	78.5	-	76.7	-
DecAF (Yu et al., 2023)	81.4	-	78.8	-
Pangu(T5-3B) (Gu et al., 2023)	83.4	57.7	79.6	-
few-shot				
Pangu (Gu et al., 2023)	53.5	35.4	48.6	-
KB-BINDER (Li et al., 2023)	50.8	34.5	56.6	96.5
KB-Coder (Nie et al., 2023)	51.7	35.8	60.5	-
one-shot				
KB-BINDER (Li et al., 2023)	16.8	4.8	9.0	65.3
AgentBench (Liu et al., 2024)	30.5	25.1	26.4	-
Ours	**60.5**	**50.8**	**63.9**	**98.5**
w/ GPT4	66.8	63.0	69.0	99.9

Table 2: Overall results on GrailQA, GraphQ, WebQSP, and MetaQA-3Hop. For the few-shot setting, Pangu uses 100-shot for all datasets. KB-BINDER and KB-Coder use 40-shot for GrailQA and 100-shot for GraphQ and WebQSP. KB-BINDER uses 5-shot for MetaQA.

Method	GrailQA	GraphQ
Ours	**60.5**	**50.8**
w/o ERASER	43.7	35.3
w/ zero-shot SC	38.5	30.2
w/ few-shot SC	48.0	40.1

Table 3: Ablation study of ERASER and a comparison with other methods. Zero-shot SC indicates the "generic" self-correction prompt of DIN-SQL (Pourreza and Rafiei, 2023). Few-shot SC indicates the "explanation feedback prompt" of Self-Debug (Chen et al., 2023). We follow and implement their ideas in our tasks.

quire up to 100 annotations on all four datasets. For GrailQA and GraphQ, our method notably surpasses the best few-shot methods by 7.0 and 15.0 points. On WebQSP, QueryAgent slightly surpasses 100-shot methods by 3.4 points. It is expected considering the inherent characteristics of the datasets. Since all WebQSP questions are under I.I.D. setting and this dataset is relatively small, few-shot methods have more opportunities to encounter similar questions within the prompts. In contrast, most of the questions of GrailQA are compositional and zero-shot questions, and 100% of GraphQ are compositional questions. Few-shot methods lose this advantage on such question types, which can reasonably explain why our approach exhibits a more pronounced advantage on GrailQA and GraphQ. Additionally, all few-shot methods incorporate beam search or self-consistency to further boost the performance. It also implies that there is still space for improvement in our method if we also choose a more costly setting.

Compared with the one-shot methods, the performance of QueryAgent approximately doubles that of Agentbench, elevating agent-based techniques and one-shot KBQA to a new level. We also reproduce the one-shot result of KB-BINDER. The dramatic decline in performance exposes some limitations of the ICL-based method in terms of example quantity.

5 Detailed Analysis

To gain more insights into QueryAgent's strong performance, we conduct some in-depth analysis.

5.1 Ablation Study

In this section, we analyze how ERASER contributes to reliable reasoning and compare it with other self-correction methods. The result is shown in Table 3. ERASER improves for 16.8 and 15.5 points for GrailQA and GraphQ, demonstrating the effectiveness of our method. For the baseline method, zero-shot SC failed to boost the performance further and even exhibited negative gains. The few-shot method has made some improvements but not that significant and its prompt is considerably longer than ERASER. It is expected since few-shot SC can only cover limited scenar-

Methods	GrailQA			GraphQ			WebQSP		
	TPQ	**QPQ**	**CPQ**	**TPQ**	**QPQ**	**CPQ**	**TPQ**	**QPQ**	**CPQ**
KB-BINDER	51.2 s	3297.7	**$ 0.010**	84.0 s	2113.8	$ 0.024	138.6 s	8145.1	$ 0.017
AgentBench	40.0 s	7.4	$ 0.034	65.1 s	7.2	$ 0.035	70.4 s	7.2	$0.038
Ours	**16.6 s**	**5.2**	$0.019	**15.3 s**	**6.2**	**$ 0.021**	**12.6 s**	**4.7**	**$ 0.014**

Table 4: Efficiency comparison with KB-BINDER. The TPQ, QPQ, and CPQ respectively represent the time cost, SPARQL query times, and `gpt-3.5-turbo` invocation cost per question.

ios and LLM needs to figure out which part in the prompt is related to the current situation. We also manually analyzed 200 questions of GrailQA to investigate how ERASER influences the reasoning process. We find that 43% of questions utilized ERASER in their reasoning processes. Among them, 30% questions were completely corrected. Given that our error detection strategy is conservative, all questions that triggered the ERASER were indeed found to contain errors during reasoning.

5.2 Efficiency Analysis

In this section, we evaluate the running efficiency. We conduct both horizontal and vertical comparisons by comparing KB-BINDER, which utilizes a different paradigm, and AgentBench, which is similar to ours. We analyzed the time cost per question (TPQ), query times per question (QPQ), and ChatGPT invocation cost (CPQ). All tests were conducted in the same network environment, with each experiment running independently.

As shown in Table 4, compared to KB-BINDER, our method exhibits overwhelming advantages in terms of TPQ and QPQ, while CPQ is a little higher on GrailQA. This outcome aligns with our expectations. KB-BINDER needs to conduct a beam search step by step to collect a large pool of candidates and then execute them one by one to find the first executable query, which requires querying numerous SPARQLs. Additionally, KB-BINDER uses self-consistency by repeating this paradigm for K times to boost the performance, leading to $(K-1)\times$ extra cost. To some extent, these also lead to a longer running time. Another thing worth noting is that more attempts also imply a lower accuracy of the top-1 candidate and a higher proportion of low-quality candidates. In contrast, our method only selects the top-1 candidate at a time, which means it requires the method to possess a high level of precision at each step. However, even under such extreme constraints, our approach still outperforms other methods.

As for the CPQ, our method incurs slightly higher costs in terms of LLM invocation compared to KB-BINDER. Our method is a step-by-step reasoning process, and while it has many advantages, we acknowledge that it also has an inevitable issue of requiring multiple requests to the LLM. However, on the flip side, KB-BINDER needs to concatenate many examples, which also faces the challenge of having a long prompt. In fact, on the 100-shot setting, the CPQ of using KB-BINDER has already exceeded that of our method.

On the other hand, compared to AgentBench, our method also surpasses it on all three criteria. It is noteworthy that our method is not only faster and cost-effective but also achieves approximately double the QA performance compared to AgentBench. At first glance, the incorporation of ERASER is a negative factor for efficiency evaluation since the prompt becomes longer than a regular reasoning process. Nonetheless, from a different perspective, timely and accurate error correction prevents the system from deviating further in the wrong direction and reduces the overhead caused by meaningless reasoning processes. Consequently, to some extent, a reliable reasoning process ultimately contributes to achieving efficient reasoning. Besides, by only performing corrections when necessary and distinguishing different types, we have managed to minimize the costs of ERASER.

5.3 Generalization Ability

In this section, we analyze the generalization ability of our method and ICL-based method from qualitative analysis and experimental comparisons.

Methodologically speaking, our method tackles the question step-by-step with atomic symbolic tools. By decomposing the problem into multiple reasoning steps, we bridge the semantic gap between different questions and datasets, as all questions can be represented using these limited

Methods	WikiSQL
few-shot(32 shot)	
Davinci-003	49.1
ChatGPT	51.6
StructGPT(Davinci-003)	64.6
StructGPT(ChatGPT)	65.6
one-shot	
AgentBench	57.6
Ours	**72.5**
w/o ERASER	67.0

Table 5: The results of QueryAgent on WikiSQL. We evaluate denotation accuracy.

tools. However, the combination of these steps can be numerous, posing challenges for compositional generalization. ICL-based methods learn and generate the complete query at once, directly facing and bearing the significantly larger search space.

From the perspective of the experiment, KB-BINDER is sensitive to whether similar examples appear in the prompt. If the most similar questions are retrieved as examples in the prompt, KB-BINDER can achieve up to 20 point improvement on WebQSP (100% i.i.d.) but a negative boost on GraphQ (100% non-i.i.d.). In contrast, our method uses the same example for all questions. Another observation is that, the higher the proportion of non-iid questions in the dataset, the greater the degree to which our approach exceeds the ICL-based approach. Compared to GrailQA (75% non-i.i.d.), QueryAgent demonstrates greater improvement on GraphQ (100% non-i.i.d.). This can also serve as evidence that QueryAgent has better generalization on unrelated examples.

5.4 Transfer Experiment

In the previous sections, we choose KBQA as a representative testbed to instantiate QueryAgent and ERASER. To illustrate the versatility of our reasoning framework and ERASER, in this section, we conduct another two experiments: 1) we implement QueryAgent framework on another semantic parsing task, namely Text2SQL. 2) we adapt ERASER to AgentBench.

We choose the test set of WikiSQL (Zhong et al., 2017) as the experiment dataset. To acquire the execution feedback from the database environment, we implement a SQL-version PyQL to help LLM access the database and provide tools to construct the SQL query. We compare our method with StructGPT (Jiang et al., 2023). The baseline results of Dacinci-003 and ChatGPT also come from StructGPT. Our method outperforms the few-shot method with 32 examples. Besides, ERASER contributes to 7.6% of performance, indicating the generalization ability of our self-correction method.

Methods	GrailQA	GraphQ	WebQSP
AgentBench	30.5	25.1	26.4
w ERASER	38.5	35.6	32.0

Table 6: The performance of AgentBench with ERASER.

Another experiment (*i.e.,* AgentBench + ERASER) is to further verify that ERASER can enhance the existing agent-based KBQA system. Table 6 shows that ERASER further improves the performance of AgentBench by 8.0 and 10.5 points on GrailQA and GraphQ. By integrating ERASER, we have elevated the performance of another method to a new level, highlighting the versatility and plug-and-play nature of ERASER.

6 Conclusion

In this paper, we present a reliable and efficient framework called QueryAgent, which constructs the target query step-by-step with tools and performs stepwise self-correction. We also introduce a novel self-correction method called ERASER. It leverages rich environmental feedback to enable selective and differentiated self-correction, departing from the conventional approach which only uses the final result to conduct correction on every output with the same prompt. Experimental results demonstrate that QueryAgent notably outperforms all existing few-shot methods on four KBQA datasets with only a single example, especially on GrailQA (+7.0) and GraphQ (+15.0). Moreover, QueryAgent also exhibits superiority in efficiency with faster solving speed and lower resource utilization. Compared to ICL-based methods, our approach reduces runtime and query costs by a factor of tens, while compared to agent-based methods, it reduces time and API invocation costs by more than half. We also show the versatility of QueryAgent and ERASER by evaluating it on a Text2SQL dataset and applying ERASER on another system (AgentBench). QueryAgent outperforms previous few-shot method and ERASER further boosts the performance of QueryAgent.

Limitations

Here we would like to discuss several limitations of our method. Firstly, the various feedback is the basis to detect and distinguish different errors. If the feedback is unavailable or too simplistic, such as only providing the final answer, there is insufficient information to confidently conduct error detection and differentiate between various error types. Therefore, ERASER may have limited benefits in end-to-end approaches. Another limitation is that, while step-by-step solving is widely recognized as a promising way of addressing complex tasks, it inevitably leads to the issue of lengthy prompts. The cost can be further minimized by optimizing historical encodings and prompt engineering. However, these engineering techniques are not the primary focus of this study.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4928–4949, Toronto, Canada. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: Three levels of generalization for question answering on knowledge bases. *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, 2021:3477–3488.

Yu Gu and Yu Su. 2022. ArcaneQA: Dynamic program induction and contextualized encoding for knowledge base question answering. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1718–1731, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023a. Large language models cannot self-correct reasoning yet.

Xiang Huang, Sitao Cheng, Yuheng Bao, Shanshan Huang, and Yuzhong Qu. 2023b. MarkQA: A large scale KBQA dataset with numerical reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10241–10259, Singapore. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2023. Code-style in-context learning for knowledge-based question answering.

OpenAI. 2023. Gpt-4 technical report.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8108–8121, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Empirical Methods in Natural Language Processing (EMNLP)*, Austin, Texas, USA. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph.

Danqing Wang and Lei Li. 2023. Learning from mistakes via cooperative study assistant for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10667–10685.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

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

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

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206, Berlin, Germany. Association for Computational Linguistics.

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

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.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.