Call Me When Necessary: LLMs can Efficiently and Faithfully Reason over Structured Environments

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have shown potential in reasoning over structured environments, *e.g.,* knowledge graphs and tables. Such tasks typically require multi-hop reasoning, *i.e.,* match natural language utterance with instances in the environment. Previous works adopt LLMs to incrementally build a reasoning path, where LLMs either invoke tools or pick up items by step-by-step interacting with the environment. We propose **Re**asoning-**P**ath-E**di**ting (Readi), a novel framework where LLMs can efficiently and faithfully reason over structured environments. In Readi, LLMs initially generate a reasoning path given a query, and edit the path only when necessary. We instantiate the path on structured environments and provide feedback to edit the path if anything goes wrong. Experimental results on three KGQA and two TableQA datasets show the effectiveness of Readi, significantly surpassing previous LLM-based methods (by Hit@1 9.1% on WebQSP, 12.4% on MQA-3H and 9.5% on WTQ), comparable with state-of-the-art fine-tuned methods (67% on CWQ and 74.7% on WebQSP) and substantially boosting the vanilla LLMs (by 14.9% on CWQ). Our code will be available upon publication.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance in NLP fields (Ouyang et al., 2022; OpenAI, 2023; Liang et al., 2023). To further unleash their reasoning ability in complex scenarios, delicate strategies are proposed to equip LLMs with human-like thought process (*e.g.,* Chain-of-thought Wei et al., 2022) or leverage them as autonomous agents capable of planning, reflection and execution (Yao et al., 2023; Liu et al., 2024). One compelling scenario where LLMs showcase their potential is reasoning over **structured environments (SEs)** (Jiang et al., 2023a; Sun et al., 2024). With

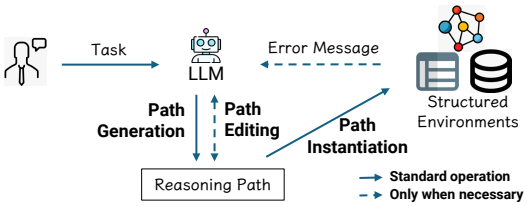

Figure 1: An illustration of our proposed framework, Readi, where LLMs initially generate a reasoning path, and when necessary, edit this path. We instantiate the path on structured environments and invoke editing if the instantiation gets stuck.

dedicate schemas, SEs (*e.g.,* knowledge graphs, tables) abstract real-world semantics for representing, storing, and querying data with relational structures. For instance, Freebase (Bollacker et al., 2008) captures 45M entities and 3B facts over 100 domains, organized in triple patterns. The crux of successful reasoning lies in bridging the gap between natural language and the mechanism of how the SEs are represented and operated (Gu et al., 2023; Li et al., 2023).

While LLMs exhibit promising capabilities, their performance often falls short when faced with multi-hop reasoning involving large-scale SEs. To faithfully reason, prior works adopt an iterative way that start from certain elements (*e.g.,* entity, relation in KG, column in Table), instantiate on SEs and then gradually expand the reasoning path while consuming just relevant portions instead of the entire environment (Sun et al., 2024; Jiang et al., 2023a). This step-by-step interaction could mitigate hallucination of LLMs to some extent. However, the reasoning efficiency is sacrificed and thus hinder the practical feasibility. For a simple constraint, say "`the daughter of Obama`", such methods probably require two LLM-calls to first query the relations around "`Obama`" and then select "`father_of`" from returned candidates. Moreover, at each step, LLMs make one choice based on history, making it prone to error propagation. Alternatively, fine-tuned methods inject environments into model

parameters by tuning with human-labeled supervision. During inference, they recall schema patterns to build reasoning paths (Zhang et al., 2022; Saxena et al., 2020; Luo et al., 2024; Ding et al., 2024) without interaction with SEs. This end-to-end paradigm is efficient, however, it is never ensured that the model output can be grounded on SEs. Study shows that 50% paths of RoG (Luo et al., 2024) failed to yield faithful results. In addition, they heavily rely on annotations, which are difficult to obtain for large-scale SEs. Further research is thus needed to achieve efficient (*i.e.,* less LLM-calls) and faithful (*i.e.,* grounded on SEs) reasoning over massive SEs.

Therefore, we seek to propose an interaction paradigm that leverage LLMs to support complex reasoning on large-scale SEs faithfully and efficiently. Conducting empirical study on KGQA task, we find that 46%-60% of reasoning paths initially generated by LLMs can be well instantiated, which inspires us to fully exploit LLM's intrinsic planning ability in complex reasoning. While the idea of plan-and-refine is straightforward and applied in various real world tasks (Qiao et al., 2023; Collaboration, 2023; Ahn et al., 2022), it's worth noting that there exists few research on application in SEs when the initial plan encounters obstacles.

In this paper, we propose **Re**asoning-P**a**th-E**di**ting (Readi), a novel framework that leverages the intrinsic planning ability of LLMs (Figure 1). In Readi, LLMs initially generate a reasoning path, which is then instantiated on SEs to facilitate faithful reasoning. Path editing is triggered only if corrections are necessary. This way, we alleviate the burden of step-by-step interaction for LLMs, resulting in improved overall efficiency. To harness the information of large-scale SEs, instead of injecting the entire static SEs into the model, we collect reasoning log as immediate feedback which includes details such as the position of stuck points, associated relations, half-way done instances, etc. This dynamic guidance refines the reasoning path more targeted and further enhances faithfulness. Our experiments on Question Answering over Knowledge Graphs (KGQA) and Tables (TableQA) demonstrate that Readi surpasses existing solutions in terms of both LLM-calls and accuracy.

We summerize our contributions as follows:

• We introduce Readi, a novel framework where LLMs reason efficiently and faithfully over large-scale structured environments. Notably, Readi is the first to fully harness LLMs intrinsic planning ability for reasoning in such contexts.

• In comprehensive experiments across five multi-hop reasoning tasks in KGQA and TableQA, Readi outperforms other LLM-based solutions and surpasses most fine-tuned methods. Specifically, it achieves 67.0% Hit@1 on CWQ, 78.7% on WebQSP and state-of-the-art results on MQA-1H.

• We give detailed analysis which highlights the performance of Readi's reasoning path generation and editing modules. Experiments demonstrate that Readi, with an average of 1.55 calls for editing, significantly reduces the number of LLM-calls compared to the step-by-step interaction paradigm (which costs 4 to 8 calls). Furthermore, the reasoning log reveals that Readi exhibits characteristics akin to human thought process.

2 Related Works

Step-by-step reasoning over structured environments by LLMs. Introducing massive SEs (*e.g.,* Freebase (Bollacker et al., 2008) captures 45M entities and 3B facts over 100 domains) into LLMs context windows is not practical. Existing works break the task down to incrementally construct a reasoning path. They either treat LLMs as an agent to invoke tools based on history states and observations (Liu et al., 2024; Qin et al., 2023), or design iterative procedures where LLMs are responsible for picking up items on SEs (Gu et al., 2023; Jiang et al., 2023a; Sun et al., 2024). These works reach faithfulness by leveraging the reasoning ability of LLMs for *tool or item selection*. However, their performance is concerned with three shortcomings: 1) The iterative interaction with SEs is cumbersome, requiring quite a few LLM-calls, which is especially not efficient for complex reasoning task. 2) The greedy step-by-step decision lacks a global view of the path, making it prone to error propagation. 3) The accumulated prompts are lengthy where LLMs may lose attention of history or candidates. To ease these problems, we propose to directly generate a reasoning path, and edit it with feedback when the instantiation gets stuck.

End-to-End reasoning over structured environments by fine-tuning. Fine-tuned models *memorize the environments* through training over annotations. They either directly generate a path and then ground it on SEs (Luo et al., 2024; Huang et al., 2023b; Shu et al., 2022; Hu et al., 2022), or retrieve relevant items to build a path (Zhang et al., 2022; Saxena et al., 2020; Ding et al., 2024). Such end-to-

end reasoning shows efficiency with no interaction with SEs. However, they have three limitations: 1) The grounding of reasoning path only depends on model outputs, without ensuring faithfulness on SEs. To remedy this issue, they rely on a wider beam at the expense of more retrieved instances. 2) They rely heavily on annotations, which are expensive for massive environments. 3) The performance drops substantially on data unseen during training (Gu et al., 2021; Huang et al., 2023a), which is common in real-world scenarios. To alleviate these problems, instead of fine-tuning, we propose to instantiate LLMs reasoning path. If anything goes wrong, we call LLMs to edit the path.

Plan-and-Refine Reasoning with LLMs. For faithfulness of LLMs reasoning, previous works adopt LLMs to refine the output (Pan et al., 2023). Some require LLMs to self-correct, prompting them to identify and correct errors (Madaan et al., 2023; Pourreza and Rafiei, 2023). Such methods achieve limited improvement, since they rely only on the intrinsic knowledge of LLMs, without any access to the environment. Alternatively, other works require LLMs to refine the previous plan based on environmental feedback (*e.g.* code error messages) (Chen et al., 2024; Qiao et al., 2023). The feedback provides the execution results and possible errors, which are more purposeful and thus effective. However, how to collect feedback for large-scale structured environments remains an open question. In Readi, we collect immediate reasoning log through the instantiation of reasoning path, including the position of error, half-way done instances and associated relations.

3 Task Definition

Reasoning over structured environments (SEs), *e.g.,* question answering over Knowledge Graphs (KGQA) and Tables (TableQA), typically requires matching a question with instances in SEs to constrain the answer. An intermediate reasoning path is **a structural representation of the question**, as a bridge between the question and SEs. Figure 2 exemplifies some reasoning paths and their instances on KG. Note that a KG is a set of triple patterns, *i.e.,* $\{(e, r, e')|e, e' \in \mathcal{E}, r \in \mathcal{R}\}$, where $\mathcal{E}$ and $\mathcal{R}$ refers to the set of entity and relation, respectively.

Formally, given a question Q and n topic entities E, the reasoning path P is conditioned by several constraints in Q. It is worth noting that P can represent complex constraints, (*i.e.,* P can be single-constrained or multi-constrained). A single-constrained P is from only one topic entity. For example, a sequence of relations from the entity to the answer (Example $Q1$) or a Chain-of-thought (Wei et al., 2022). A multi-constrained P is from multiple topic entities, consisting of multiple paths to constrain the answer (Example $Q2$). Correspondingly, we instantiate P on KG to obtain instances. For Example $Q1$, the reasoning path "`father_of→university`" from "`Obama`" is instantiated to "`Obama` $\xrightarrow{father\_of}$ `Malia` $\xrightarrow{college}$ `UCMC`" and "`Obama` $\xrightarrow{father\_of}$ `Sasha` $\xrightarrow{college}$ `UCMC`".

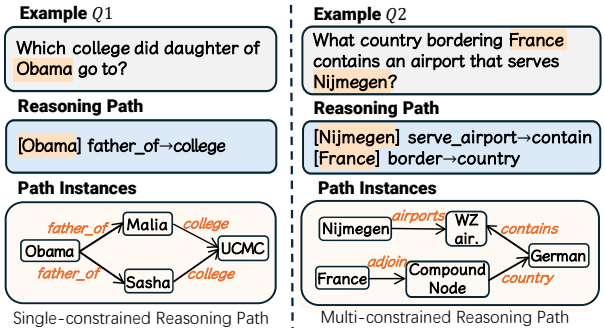

Figure 2: Examples of the question, reasoning path, and corresponding path instances on knowledge graph.

Following Sun et al. (2018), we model reasoning over structured environments as an retrieve-then-reason task, where we leverage LLMs to build the reasoning path given Q. Then we reason over the path instances the obtain the answer.

4 Methodology

4.1 Overview

For better illustration, we adopt KGQA, a challenging scenario of multi-hop reasoning over massive environments, to showcase concrete implementation (Refer to Appendix A for TableQA). A running example is in Figure 3. Given a question and topic entities, we leverage the planning ability of LLMs to generate the initial reasoning path P (Section 4.2). Then we instantiate P on KG (Section 4.3). If the entire P is successfully instantiated, we take the intersection of all instances then generate the answer. If any path in P gets stuck, we collect error messages to guide further refinement (Section 4.4). Such process runs until P is fully instantiated or a maximum edit time is reached. Please refer to Appendix G for concrete prompts of each module.

4.2 Reasoning Path Generation

Inspired by Li et al. (2023), we leverage in-context learning (ICL) to generate the initial reasoning path,

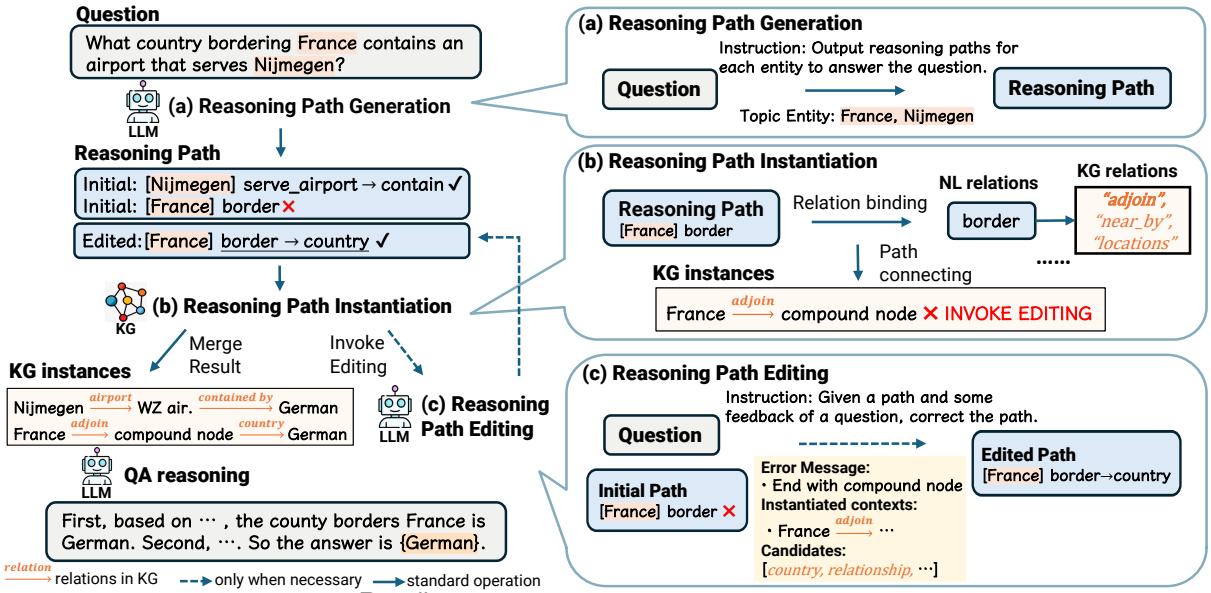

Figure 3: A running example of Readi on KGQA. An LLM initially generate an reasoning path for a question. Then, we instantiate it on KG. If anything goes wrong (the path from "France"), we collect some error messages and call an LLM to edit the path. Finally, an LLM answers the question based on the KG instances.

as shown in Figure 3(a). Given a question and n topic entities, we utilize Chain-of-Thought (CoT) to generate the initial path, consisting of n constraints starting from each topic entity. For the example in Figure 3, we have a two-constrained path (from "`France`" to constrain the bordering countries and from "`Nijmegen`" to airports serving it and then to countries containing these airports).

4.3 Reasoning Path Instantiation

We instantiate the reasoning path P on KG and merge the instances. The main difficulty lies in how to sequentially match the natural language (NL) relations in P with relations in KG. Our instantiation involves two steps: relation-binding and path-connecting, as shown in Figure 3(b).

For relation-binding, given a path consisting of some NL relations (*e.g.,* for "`[France] border`", we have one relation "`border`" expressed in NL), we bind them to relation schemas in KG. Since a relation $\boldsymbol{r}$ in NL may have similar relation schemas in KG with analogical format and semantic meanings (Li et al., 2023), we leverage BM25 and Contriever (Izacard et al., 2022) to retrieve similar relations $\hat{\boldsymbol{r}}$ as candidates for $\boldsymbol{r}$. For example, we bind "`border`" with 3 candidate relations "*adjoin, near_by, locations*" in KG (the orange italicized relations in Figure 3(b). This way, we obtain candidate relations for all relations in P.

For path-connecting, note that the starting entity of each path is given. For the example in Figure 3(b), we need to check if there exists any path instance in KG from "`France`" where relations in this instance sequentially match the bound candidate relations of all relations in P. Specifically, if any relations in candidates "*adjoin, near_by, locations*" connects to "`France`" (the orange bold "***adjoin***"), we obtain entities satisfying the constraint "`[France] border`", so the NL relation "`border`" is instantiated to "***adjoin***" on KG. Then, we repeat the same process for the remaining relations in P to finally instantiate the whole path.

If any path in P is not successfully instantiated, *e.g.,* none of the candidate relations $\hat{\boldsymbol{r}}$ connects to the current entity, this is the *necessary time* to **invoke editing** (Section 4.4). If the entire P is successfully instantiated or the maximum edit time is reached, we merge all KG instances by intersection to answer the question (Section 4.5). Please refer to Appendix B.1 for concrete implementation.

4.4 Reasoning Path Editing

The goal of editing is to help LLMs identify the error of previous reasoning path and provide some error messages, consisting of 2 steps: summarization and preparation, similar to an error traceback of coding, as shown in Figure 3(c).

For summarization, we categorize the reasons of error as follows: *i.* "irrelevant NL relation $\boldsymbol{r}_{err}$": none of the KG relation candidates of $\boldsymbol{r}_{err}$ connects to current entity; *ii.* "empty reasoning path"; *iii.* "path ends with compound nodes"[1]. Thus, if the

[1]Compound value node, *i.e.,* blank node, is typical in KG to express some complex entities, such as an event.

instantiation goes wrong, we detect error position $\boldsymbol{err}$, current NL relation $\boldsymbol{r_{err}}$ (none for reason *iii*) and current ending entity $\hat{\boldsymbol{e}}_{\boldsymbol{err}}$ (before which the NL relations are successfully instantiated).

For preparation, we collect some useful messages: 1) the reason of error. 2) the currently halfway-done instances. 3) relations around $\hat{\boldsymbol{e}}_{\boldsymbol{err}}$ is accessible, which might be the candidates to amend the halfway-done instances. Finally, an LLM is called to edit the previous path based on Q and these error messages by ICL.

For the example in Figure 3, we have the NL relation "`border`" instantiated to *"adjoin"* and obtain a compound node at the end of instance, which triggers reason *iii* to invoke LLMs for editing. Therefore, we concatenate reason *iii*, halfway-done instance "`France` $\xrightarrow{adjoin}$ `Compound Node`" and candidate relations around the "`Compound Node`" (*e.g.,* "*country*" and "*relationship*") as error messages. More details are in Appendix B.2.

4.5 QA Reasoning

Upon obtaining the merged KG instances $\mathcal{S}_q$, *i.e.,* the intersection of instances for each path (constraint), we build an LLM-based reasoning module to answer the question Q. We concatenate Q and $\mathcal{S}_q$ as input and prompt an LLM to generate the answer. Note that the form of $\mathcal{S}_q$ is a set of triple patterns $(entity, relation, entity')$. For entities, we convert the mid in KG to the corresponding friendly name. For relations and compound nodes, we keep their original forms in KG. Specifically, we ask LLMs to pick the knowledge triples used to answer Q in a CoT manner. For the example KG instances from "`France`" in Figure 3, we have 2 knowledge triples "`(France, adjoin, compound node), (compound node, country, German)`", based on which, the LLM can reason that "`the country bordering France`" is "`German`".

5 Experiments

To comprehensively evaluate the reasoning ability of Readi on large-scale structured environments, we experiment on two complex tasks, KGQA and TableQA (implementation in Appendix A).

5.1 Datasets

We evaluate on three KGQA and two TableQA datasets. Detailed statistics are in Appendix C.1.

WebQuestionsSP (WebQSP) (Yih et al., 2016) contains KGQA questions from Google query logs with up to 2-hop reasoning on Freebase, mostly requiring a single-constrained reasoning path.

ComplexWebQuestions (CWQ) (Talmor and Berant, 2018) is a complex KGQA benchmark, challenging for up to 4-hop reasoning on Freebase, with 55% multi-constrained questions.

MetaQA (Zhang et al., 2018) is a KGQA dataset from movie domain, with 3 levels of difficulty, denoted as MQA-1H, MQA-2H and MQA-3H.

WikiTableQuestions (WTQ) (Pasupat and Liang, 2015) contains questions over 421 tables, challenging for complex aggregation operations, *e.g.,* count, argmax, and sorting.

WikiSQL (Zhong et al., 2017) is a large-scale complex dataset based on Wikipedia tables, requiring comparison, aggregation and arithmetic operations.

5.2 Baselines

KGQA Baselines. *Training-based methods* fine-tune pre-tained language models (PLMs): EmbedKGQA (Saxena et al., 2020), NSM (He et al., 2021), TransferNet (Shi et al., 2021), SR+NSM+E2E (Zhang et al., 2022), UniKGQA (Jiang et al., 2023c), ReasoningLM (Jiang et al., 2023b) and RoG (Luo et al., 2024). *Inference-based methods* call LLM-APIs: Davinci-003 (Ouyang et al., 2022), GPT3.5, GPT4 (OpenAI, 2023), AgentBench (Liu et al., 2024), StructGPT (Jiang et al., 2023a). All baselines assume golden topic entities are given.

TableQA Baselines. *Training-based methods* find-tune PLMs: TAPAS (Herzig et al., 2020), UnifiedSKG (Xie et al., 2022), TAPEX (Liu et al., 2022). *Inference-based methods* call LLM-APIs: Davinci-003 (Ouyang et al., 2022), GPT3.5, GPT4 (OpenAI, 2023), StructGPT (Jiang et al., 2023a). Note that *Inference-based methods* model TableQA as an information retrieval task.

Following Tan et al. (2023), we adopt Hit@1, assessing whether the predicted entity is correct, to evaluate KGQA. We adopt denotation accuracy, assessing whether the prediction exactly matches the golden, to evaluate TableQA. We elaborate the baselines in Appendix C.3. We discuss in-depth comparison with ToG (Sun et al., 2024) and DATER (Ye et al., 2023) in Appendix F.

5.3 Implementation Details

We adopt `gpt-3.5-turbo` (OpenAI, 2022) and GPT4 (OpenAI, 2023) as LLMs, denoted as Readi-GPT3.5 and Readi-GPT4. Temperature is 0.3 for all modules. For relation-binding, we deploy a

Methods	WebQSP	CWQ	MQA-1H	MQA-2H	MQA-3H
Training-based Method					
EmbedKGQA (Saxena et al., 2020)	66.6	-	**97.5**	98.8	94.8
NSM (He et al., 2021)	67.7	47.6	97.1	99.9	98.9
TransferNet (Shi et al., 2021)	71.4	48.6	97.5	**100***	**100***
SR+NSM+E2E (Zhang et al., 2022)	69.5	49.3	-	-	-
UniKGQA (Jiang et al., 2023c)	75.1	50.7	**97.5**	99.0	99.1
ReasoningLM (Jiang et al., 2023b)	78.5	**69.0***	96.5	98.3	92.7
RoG (Luo et al., 2024)	**85.7***	62.6	-	-	84.8
Inference-based Method					
Davinci-003 (Ouyang et al., 2022)	48.7	-	52.1	25.3	42.5
GPT3.5 (OpenAI, 2022)	65.7	44.7	61.9	31.0	43.2
GPT4 (OpenAI, 2023)	70.7	52.1	71.8	52.5	49.2
AgentBench (Liu et al., 2024)	47.8	24.8	-	-	-
StructGPT (Jiang et al., 2023a)	69.6	-	97.1	97.3	87.0
Readi-GPT3.5	74.3	55.6	98.4	**99.9**	**99.4**
Readi-GPT4	**78.7**	**67.0**	**98.5***	**99.9**	99.2

Table 1: QA performance (Hit@1) of Readi on KGQA datasets. Results of GPT3.5, GPT4 (OpenAI, 2023) and AgentBench (Liu et al., 2024) are run by ourselves, others are from the origin paper. Bold and underline fonts denotes the best and second-best for two types of methods, respectively. $\star$ denotes the overall state-of-the-art result.

Pyserini as a hybrid searcher with BM25 and Contriever (Izacard et al., 2022). For each relation generated by LLMs, we retrieve top 5 similar relations on Freebase. For instantiation, we deploy a Virtuoso server following the instructions[2]. More details can be found in Appendix C.

5.4 Main Results

Results for KGQA As shown in Table 1, overall, Readi significantly outperforms all Inference-based methods, the vanilla LLMs and most training-based methods on all datasets, demonstrating the effectiveness of Readi. Compared with inference-based methods, Readi substantially boosts the results of the vanilla LLM (by 8.6% on WebQSP and 14.9% on CWQ), demonstrating that Readi enables LLMs to practically interact with structured environments. Moreover, Readi-GPT3.5 already significantly surpasses state-of-the-art results with LLM-APIs (by 4.7% on WebQSP and 12.4% on MQA-3H), and Readi-GPT4 further enhances the performance. With fewer LLM-calls, our directly-generated reasoning path and editing framework still achieves better performance. Compared with Training-based methods, without large-scale supervision and cost of beam search, Readi achieves comparable performance (*e.g.* 67.0 Hit@1 on CWQ) by some demonstration examples, showing the effectiveness of Readi. Additionally, Readi sets new state-of-the-art results MQA-1H, showing the effectiveness of editing which provides pertinent feedback upon instantiation errors.

[2]https://github.com/dki-lab/Freebase-Setup

Methods	WTQ	WikiSQL
Training-based Method		
TAPAS	48.8	83.6
UnifiedSKG (T5-3B)	49.3	86.0
TAPEX	**57.5**	**89.5**
Inference-based Method		
Davinci-003	34.8	49.1
GPT3.5	55.8	59.8
GPT4	57.0	59.9
StructGPT	52.2	65.6
Readi-GPT3.5	**61.7**	**66.2**
Readi-GPT4	61.3	66.0

Table 2: QA performance (denotation accuracy) of Readi on TableQA datasets. Bold fonts denotes the best results for Training-based and Inference-based methods.

Results for TableQA We experiment on TableQA scenario requiring multi-hop reasoning over tables to show the generalizability of Readi. As shown in Table 2, overall, Readi outperforms all Inference-based and most Training-based baselines, setting state-of-the-art results on WTQ, demonstrating the effectiveness of our framework. Compared with Inference-based methods, Readi sur-

Variance of Readi	Answer Coverage Rate (AC)				QA Performance (Hit@1)			
	Corrupt	*Empty*	GPT3.5	GPT4	*Corrupt*	*Empty*	GPT3.5	GPT4
w/o edit	-	-	56.7	62.7	-	-	51.0	57.2
w/ edit by GPT3.5	54.0	56.4	62.5	64.3	57.3	58.5	58.7	58.5
w/ edit by GPT4	55.6	63.9	68.6	65.8	58.2	59.9	58.1	59.3

Table 3: Answer Coverage Rate (AC) and QA Performance (Hit@1) of variance of Readi (GPT3.5 as reasoning module). Each column denotes a path generation method. *Corrupt* means a path with some randomly-sampled relations. *Empty* means empty path. w/o edit means we only leverage the initial reasoning path.

passes previous state-of-the-art methods by 9.5% and the vanilla LLM by 5.9% on WTQ, showing that Readi significantly improve the LLMs performance. Compared with Training-based methods, Readi, without massive annotations, is significantly superior on WTQ, while trailing behind on WikiSQL. This may be due to the i.i.d. distribution between the training and testing sets of WikiSQL favoring the results of fully-trained methods. Interestingly, Readi-GPT3.5 are comparable with Readi-GPT4. This might because we asks LLMs to reason (with aggregation operations) directly based on retrieved table items. Further analysis shows that Readi-GPT3.5 has more chance of invoking editing than Readi-GPT4, which offers more pertinent feedback from the environments. Extensive elaboration on effectiveness of editing is in Section 6.1, Section 6.2 and Appendix D.2.

6 Analysis

We further analyze Readi's modules, reasoning path and efficiency on 1000 test samples of CWQ. For fairness, we base all evaluation on our instantiation and reasoning modules. Please refer to Appendix D for more detailed analysis of Readi.

6.1 Ablation Study

Effectiveness of reasoning path generation and editing modules. As shown in Table 3, we adpot the answer coverage rate (AC, rate of instances containing the answer) and QA performance (Hit@1) for illustration. We also analyze the **robustness of editing module** with a *Corrupt* path and an *Empty* path. First, Readi establishes a plug-and-play nature for both modules, showing their effectiveness. With only an initial path (we instantiate the path and maintain the longest path if it goes wrong), Readi reaches comparable results (1st Row). With an *Empty* initial path (*Empty* columns), the editing modules still yield competitive results close to the full Readi. Second, generally, higher capacity of LLMs leads to better results for both modules,

Methods	Graph Quality		QA Perf.
	AC	#RK	Hit@1
SR			
- beam size 1	58.4	**26.3**	50.9
- beam size 3	67.2	47.1	54.6
RoG			
- beam size 1	57.0	69.5	52.2
- beam size 3	**77.5**	170.1	57.3
Readi initial path			
- GPT3.5	56.7	134.6	51.0
- GPT4	62.7	101.4	57.2
Readi full			
- GPT3.5	62.5	93.7	58.7
- GPT4	71.8	121.5	**59.3**

Table 4: Reasoning path evaluation of Readiand compared methods. AC and #RK denotes answer coverage rate and number of retrieved knowledge, respectively.

which meets our expectation. Third, Readi shows robustness for reasoning path editing, performing well even with an *Empty* or even a *Corrupt* path (*Corrupt* and *Empty* columns).

6.2 Reasoning Path Analysis

We compare reasoning path of Readi with representative fine-tuned methods, *i.e.,* Subgraph Retrieval (SR) (Zhang et al., 2022) trains an encoder to retrieve relations to build a path, RoG (Luo et al., 2024) tunes a Llama 2 (Touvron et al., 2023) to generate a path.

Quality of Readi's reasoning path. We adopt answer coverage rate (AC) and number of retrieved knowledge (#RK) as the quality of graph. Ideally, the higher AC and lower #RK, the better. Also, we analyze the QA performance (Hit@1), shown in Table 4. First, Readi's initial path is comparable with fine-tuned methods, with GPT4 surpassing them by a large margin (5.0% and 6.3% Hit@1 than RoG and SR, respectively), showing the effectiveness of our reasoning path. Second, with some necessary editing, Readi obtains substantially higher

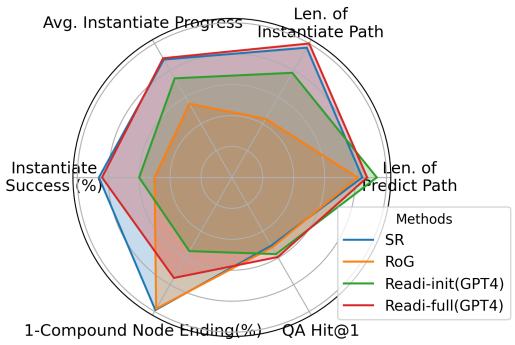

Figure 4: Extensive features of Readi's reasoning path, compared with fine-tuned methods and Golden.

AC, with a little higher #RK (GPT4) and even lower #RK (GPT3.5), and significantly higher QA performance than fine-tuned methods, showing the effectiveness of editing. Third, with wider beam size, fine-tuned methods gain higher AC, yet a drastically growing #RK, and lower Hit@1 than Readi, illustrating our superiority.

Extensive features of Readi's reasoning path. The quality of reasoning path is multi-dimensional. To further show insights of reasoning path by LLMs, we design some metrics, including the instantiation progress, error types, etc. We compare with SR, RoG and Golden (Outermost in Figure 4). More detailed analysis is in Appendix D.1.

Insights driven by Figure 4 are three-folded: 1) Readi's initial path already achieves better in QA results. More than a half (60%) of Readi's initial paths can be fully instantiated (LLMs called only once), showing that Readi effectively unleash the strong planning ability of LLMs. For each path, a large proportion (averagely 74%) can be instantiated, again indicating the possibility to safe LLM-calls. 2) With necessary editing, Readi gets significantly closer to golden (From green to red in Figure 4), exceeding compared methods in many metrics, especially the QA results. 3) Interestingly, the path by LLMs establish a human-like nature, for humans tend to get stuck at points they have never seen in real world. For Compound Node Ending Rate (special for KG), fine-tuned methods are close to Golden, showing that they memorize the structured information. However, they trail behind behind Readi on QA results, demonstrating that unfaithfulness still exits for fine-tuned methods.

6.3 Efficiency Evaluation

How many LLM-calls do we need? Due to unexpectable nature of LLMs output, we cannot give an exact number of LLM-calls for Readi. Instead, we present the distribution of number of LLM-calls

Question 1: What is the name of the money used in the country the Peruvian Paso breed originated?
Initial Reasoning Path: (from "Peruvian Paso")
biology.organism.breeds→biology.breed.originated_in→ location.country.currency_used
Error Message: irrelevant relation "biology...breeds".
Candidates: biology.breed.originated_in, ...
Edited Reasoning Path: (from "Peruvian Paso")
biology.breed.originated_in→location.country.currency_used

Question 2: What to see in the country that has Gozo?
Initial Reasoning Path: (from "Gozo")
location.location.containedby→location.country.attractions
Error Message: irrelevant relation "location...attractions".
Contexts: Gozo $\xrightarrow{location.location.containedby}$ Melta
Candidates: travel.travel_destination.tourist_attractions, ...
Edited Reasoning Path: (from "Gozo")
location.location.containedby→travel.travel_destination.tourist_attractions

Table 5: Cases of Readi's reasoning path editing.

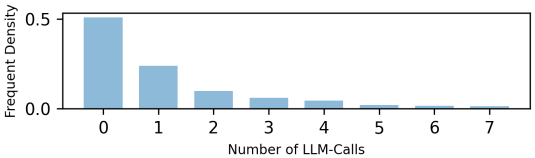

Figure 5: Distribution of number of LLM-Call for reasoning path editing of Readi-GPT4.

for editing in Figure 5 and some exemplar cases in Table 5. Note that there are at least 2-hop and up to 4-hop reasoning required for CWQ (theoretically 4-8 LLM-calls for iterative interaction). The instantiation success rate and average instantiate progress in Figure 4 already demonstrate that Readi can save a bunch of LLM-calls. In Figure 5, most of the time the initial reasoning path is already instantiable, no need for more LLM-calls. Averagely, the LLM is called 1.55 times to edit the path, saving more invocation than iterative interaction. Also, cases in Table 5 show that, with the necessary editing, LLMs can correct previous path based on some error messages during instantiation.

7 Conclusion

In this paper, we propose a novel framework Readi where LLMs can reason over structured environments efficiently and faithfully. In Readi, LLMs initially generate a reasoning path which is then instantiated on environments to facilitate faithful reasoning. Path editing is triggered only if the instantiation gets stuck. We showcase the implementation of Readi on knowledge graph, evaluate the effectiveness on KGQA and TableQA and analyze extensive features of Readi. Our work also shed lights on practically interaction between natural language and structured environments, where LLMs plays a crucial role to bridge the gap.

8 Limitations

Although Readi achieves strong performance in complex reasoning task over structured environments, *e.g.,* knowledge graph and tables, there are still some limitations of our method. First, we leverage just 2 LLMs as backbone to evaluate Readi. Therefore, more experiments can be done to test performance of Readifor other LLMs (including the inference-based and training-based). Second, we adopt reasoning path as a representation of natural language utterance and model the task as a information-retrieval one. An interesting and more precise direction is semantic parsing, *e.g.,* text2SQL, which we will leave for future work. Moreover, we can also test Readi on question answering over database or other reasoning fields. Third, the instantiation in our current implementation of KGQA is intuitive but may induce many queries for SPARQL engine if encountered some "big" entities. For example, when constraining "locations in France", we may obtain hundreds of instances, for which we just set a threshold to restrict the number of current entities. This can be improved by meticulous query design. Forth, our proposal to fully harness intrinsic planning ability of LLMs performs well and saves some LLM-calls in experiments, which does not mean we should always depend on LLMs plans in every other task. And although the initial path by LLMs can be well instantiated, this just means the path by LLMs corresponds to some instances on KG, but not ensures the path exactly matches with the constraints in the question. Finally, by efficiency, we mean number of LLM-calls, or how LLMs can practically interact with structured environments. A promising direction is to reach for time and cost efficiency.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*.

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 SIGMOD*, pages 1247–1250.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

Embodiment Collaboration. 2023. Open x-embodiment: Robotic learning datasets and rt-x models.

Wentao Ding, Jinmao Li, Liangchuan Luo, and Yuzhong Qu. 2024. Enhancing complex question answering over knowledge graphs through evidence pattern retrieval. In *Proceedings of the ACM Web Conference 2024*, WWW '24.

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. In *Proceedings of the Web Conference 2021*, WWW '21. ACM.

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*, WSDM '21. ACM.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Xixin Hu, Xuan Wu, Yiheng Shu, and Yuzhong Qu. 2022. Logical form generation via multi-task learning for complex question answering over knowledge bases. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1687–1696, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Xiang Huang, Sitao Cheng, Yuheng Bao, Shanshan Huang, and Yuzhong Qu. 2023a. 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.

Xiang Huang, Sitao Cheng, Yiheng Shu, Yuheng Bao, and Yuzhong Qu. 2023b. Question decomposition tree for answering complex questions over knowledge bases. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):12924–12932.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023a. 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.

Jinhao Jiang, Kun Zhou, Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2023b. ReasoningLM: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3721–3735, Singapore. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023c. UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations*.

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.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023. Rethinking tabular data understanding with large language models. *arXiv preprint arXiv:2312.16702*.

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*.

Linhao Luo, Yuan-Fang Li, Reza Haf, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

OpenAI. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

OpenAI. 2022. Introducing chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

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.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

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*.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, Minghua Ma, Pu Zhao, Si Qin, Xiaoting Qin, Chao Du, Yong Xu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Tool learning with foundation models.

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, Online. Association for Computational Linguistics.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4149–4158, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, Brussels, Belgium. Association for Computational Linguistics.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

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.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

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

A Implementation of Readi for TableQA

Algorithm 1 demonstrates Readi's framework. Our implementation of Readi for TableQA strictly sticks to Algorithm 1, and is simpler than KGQA. The main difference between TableQA and KGQA is the reasoning path (mentioned in Section 3). Detailed prompts of all modules is in Appendix G.

Reasoning path for TableQA starts from a given table, to some columns and then to some rows, in order to constrain specific items in the table to answer the question. A sample of WTQ (Pasupat and Liang, 2015) dataset is shown in Table 6. The reasoning path of such question is from this table to column "`Name in English`" and "`Depth`", and then to row items whose "`Name in English`" is "`Lake Tuz`" and "`Lake Palas Tuzla`".

For **reasoning path generation**, we ask LLMs to generate a reasoning path, given a question and some table descriptions (*i.e.,* the header and a randomly-sampled row), by in-context learning. Specifically, we require the LLMs to pick up at least to rows for complex arithmetic and aggregation operations in TableQA. For the example in Table 6, ideally, the LLM would generate a dict indicating the chosen header is *"['Name in English', 'Depth']"*, and the row constrain is *"{"Name in English": ['Lake Tuz', 'Lake Palas Tuzla']}"*.

Algorithm 1: Readi

Data: task Q, entity set E, environment $\mathcal{G}$, max edit time T

$t \leftarrow 0$
$P_0 \leftarrow$ **Reasoning_Path_Generation**(Q, E)
while $t \leq T$ **do**
  $I_{\mathcal{G}}, Err_{msg} \leftarrow$ **Instantiate**$(P_t, \mathcal{G})$
  if $Err_{msg} \neq \emptyset$ **then**
    $P_{t+1} \leftarrow$ **Edit_Path**(Err_{msg}, P_t, Q)
    $t \leftarrow t + 1$
  else
    Go to Line 11
  end
end
$Instances_{\mathcal{G}} \leftarrow$ **Merge_Results**$(I_{\mathcal{G}})$

For **reasoning path instantiation**, since schemas of a table is not as massive as a KG, we don't need a relation binding module to retrieve candidate relations and connect them according to the reasoning path. Instead, we first filter out columns and then filter out the rows in the reasoning path. For the example in Table 6, we filter out the column `"Name in English"` and `"Depth"` and rows whose `"Name in English"` is `"Lake Tuz"` or `"Lake Palas Tuzla"`. If the selected columns or rows are not in the table, this is the *necessary time* invoke editing.

For **reasoning path editing**, we collect the currently instantiated columns and candidate columns when the instantiation goes wrong, as error messages. Note that most questions of WTQ (Pasupat and Liang, 2015) and WikiSQL (Zhong et al., 2017), so we only consider wrong columns as error. If a selected row in the reasoning path cannot be instantiated, we return all rows in the columns. Then we ask the LLMs to edit previous reasoning path. For example, if chosen header is *["English Name"]*, not matching the headers in Table 6. We provide all headers and a randomly-sampled row in Table 6 as error messages.

The **QA reasoning** is quite similar to KGQA, we concatenate the question and instantiated table items and ask the LLMs to answer the question.

B Detailed Implementation of Readi for KGQA

B.1 Reasoning Path Instantiation Details

In this part, we demonstrate more details of path-connecting mentioned in Section 4.3. Assume that we have a reasoning path starting from an entity, with m relations R. All relations are bound to candidates $\hat{R}$. For the example in Figure 3, we have a reasoning path "`[Nijmegen] serve_airport` $\rightarrow$ `contain`" and have "r_1 : `serve_airport`" bound to "$\hat{r}_1$: *airport, terminal, serving port*" in KG, and similarly have "r_2 : `contain`" bound to "$\hat{r}_2$: *contained by, contains, place of*". The purpose if path-connecting is to check if there exists an instance in KG where the starting point is entity "`Nijmegen`", then connect to some relations in $\hat{r}_1$, and then connect to some relations in $\hat{r}_2$.

The path-connecting is essentially an Breadth-first search (BFS) algorithm, with a queue que containing possible entities connected to current relations. Firstly, we push the starting point (*e.g.,* "`Nijmegen`") into que. Second, at each time, we pop out current head of que and check if current candidate relations (*e.g.,* "*adjoin, terminal, serving port*") can connect to it on KG. If so, we successfully instantiate "`serve_airport`" on KG, and we can obtain some entities which are "`serve_airport`" of "`Nijmegen`" (*e.g.,* "`WZ air.`" and "`NTA.`"). Then, we add these entities to que. Such BFS searching continues until all relations are instantiated or anything goes wrong.

Note that for a "big" entity, their may be a

Name in English	Name in Turkish	Area (km^2)	Depth	Location (Districts and/or provinces)
Lake Van	Van Gölü	3755 km^2	171 m	Van, Bitlis
Lake Tuz	Tuz Gölü	1500 km^2	2 m	Aksaray, Ankara, Konya
Lake Palas Tuzla	Palas Tuzla Gölü	106 km^2	15 m	Palas/Kayseri
		.....		

Table 6: An example from WTQ (Pasupat and Liang, 2015), with the question "`which is deeper, lake tuz or lake palas tuzla?`".

substantial number of tail entities for a relation. For example, there are hundreds of entities for "France$^{location}\longrightarrow$". This may end up with a longer instantiation time and shed lights on future improvements of Readi. In this case, we adopt a threshold to constrain the size of que.

B.2 Reasoning Path Editing Details

Section 4.4 has covered the basic process of reasoning path editing. Here are some implementation details for preparation step, the purpose is to help LLMs identify the error position of previous reasoning path and edit it according to some tips. For the half-way done instances in the Error Messages, we use "compound node" to indicate all cvt nodes in Freebase (Bollacker et al., 2008), and we just sample some instances of each instantiated relation to showcase the path instances in KG of the previous path. The loss of information from sampling is minor because the messages just inform the LLMs that the relations in the reasoning path can be grounded to certain instances on KG. For candidates, we use a threshold to constrain the size, if there are too many of them, we filter out using similarity search (same embedding as relation-binding) according to the original question.

For example, there are many compound node instances connected to "France$^{adjoin}\longrightarrow$", and we just use "France$\xrightarrow{adjoin}$compound node" to represent the instances. If their are hundreds of candidates connected to these compounds nodes, we use the question "`What country bordering France contains an airport that serves Nijmegen`" to filter out top 35 similar candidates.

C Detailed Experimental Setups

C.1 Dataset Statistics

We experiment Readi on test sets of all datasets. We adopt Hit@1 and denotation accuracy for KGQA and TableQA, respectively. We evaluate Readi on 3 KGQA and 2 TableQA datasets (Descriptions in Section 5.1). Statistics of datasets are shown in Table 7. Note that we model all datasets as an *information retrieval* task, instead of a *semantic parsing* one.

Dataset	Training	Dev	Test
KGQA Dataset			
WEBQSP	3,098	-	1,639
CWQ	27,639	3,519	3,531
MQA-1H	96,106	9,992	9,947
MQA-2H	118,980	14,872	14,872
MQA-3H	114,196	14,274	14,274
TableQA Dataset			
WTQ	11,321	2,831	4,344
WIKISQL	56,355	8,421	15,878

Table 7: Statistics of experiment datasets.

C.2 LLM API version

For CWQ and WebQSP, we use gpt-3.5-turbo-16k-0613 and gpt-4-32k for GPT3.5 and GPT4, respectively. For others, we adopt gpt-3.5-turbo-0613 and gpt-4 for GPT3.5 and GPT4, respectively.

C.3 Baselines

KGQA baselines. *Training-based methods:*

- EmbedKGQA (Saxena et al., 2020) adopts an encoder to retrieve relevant entities and generate the answer.
- NSM (He et al., 2021) adapts neural state machine for KGQA and rank entities in a retrieved subgraph.
- TransferNet (Shi et al., 2021) adopts a transparent framework to rank entities according to different parts of a question in a subgraph.
- SR+NSM+E2E (Zhang et al., 2022) trains an encoder to retrieve relevant relations and build a path from retrieved relations.

- UniKGQA (Jiang et al., 2023c) retrieves a sub-graph and rank the schemas in a unified way.
- ReasoningLM (Jiang et al., 2023b) designs an entity encoding and training framework to rank entities in a sub-graph.
- RoG (Luo et al., 2024) trains a LLama 2 (Touvron et al., 2023) to firstly generate a path, second ground this path to the knowledge graph, and then generate answer based on the grounded graph.

Inference-based methods:

- Davinci-003 (Ouyang et al., 2022), GPT3.5 and GPT4 (OpenAI, 2023) are based on LLM-APIs. We adopt few-shot in-context learning to ask the model to output the answer of question in order to test the models inherent knowledge of datasets.
- AgentBench (Liu et al., 2024) is an agent-based method asking LLMs to call tools based on tool-discription, history and observations.
- StructGPT (Jiang et al., 2023a) requires LLMs to iterative pick up relations and entities based on current returned candidates.

TableQA baselines. *Training-based methods:*

- TAPAS (Herzig et al., 2020) predicts denotation by selecting table cells and optionally applys an aggregation operator to the selection.
- UnifiedSKG (Xie et al., 2022) sequentializes the table and tunes a T5-3B model to answer the question.
- TAPEX (Liu et al., 2022) guides language models to mimic a SQL executor.

Inference-based methods:

- Davinci-003 (Ouyang et al., 2022), GPT3.5, GPT4 (OpenAI, 2023) are based on LLM-APIs. We adopt few-shot in-context learning to ask the model to output the answer, based on the question and the entire table.
- StructGPT (Jiang et al., 2023a) iteratively filter out columns and rows of table and ask the LLMs to answer the question.

Note that all Inference-based baselines are information retrieval ones.

Metrics	LPP	LIP	AIP	ISR	CER
Golden Path	3.2	3.2	1.0	1.0	0
SR (Zhang et al., 2022)	3.7	3.3	0.88	0.86	0.01
RoG (Luo et al., 2024)	2.6	1.4	0.55	0.50	0.02
GPT3.5	4.3	2.5	0.64	0.46	0.49
+ editing	3.3	2.8	0.86	0.80	0.22
ChatGPT4	3.4	2.5	0.74	0.60	0.45
+ editing	3.6	3.2	0.89	0.84	0.25

Table 8: Extensive features of Readi-GPT3.5 and Readi-ChatGPT4's reasoning path, compared with Golden reasoning path. LPP, LGP, AGP, GSR, IER means length of predict path, length of instantiated path, average instantiation progress, grounding success rate and ending with intermediate nodes rate, respectively.

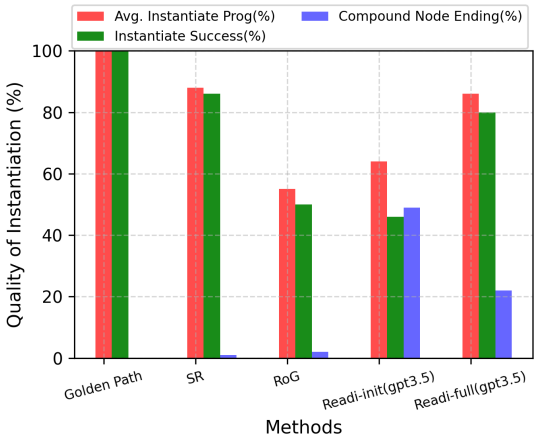

Figure 6: Extensive analysis of Readi-GPT3.5 reasoning path, compared with Golden reasoning path.

D Detailed Analysis

D.1 Elaboration of Reasoning Path Analysis

Note that quality of reasoning path is multi-dimensional. To further show some insights of reasoning path by LLMs, we meticulously design some metrics. We introduce average Length of predict path (LPP), average length of instantiated path (LIP), average instantiation progress (AIP, average of LIP/LPP for each question), instantiation success rate (ISR) and Compound nodes Ending Rate (CNR), where all length mentioned above refers to number of relations on path. We compare with fine-tuned SR (Zhang et al., 2022) and RoG (Luo et al., 2024) (beam=1). The Golden Path is obtained by extracting relations from golden logical forms in the dataset, wich is only used for analysis.

The absolute performance of different metrics of

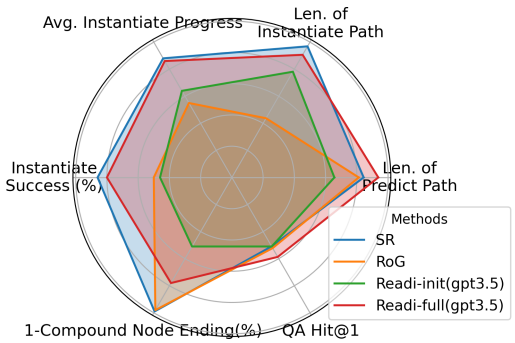

Figure 7: Features of Readi's reasoning path, compared with fine-tuned methods and Golden.

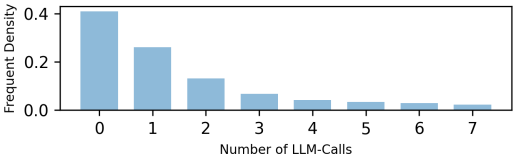

Figure 8: Distribution of number of LLM-Call for reasoning path editing of Readi-GPT3.5

Readi are shown in Table 8. Also, Figure 6 demonstrates some instantiate progress of Readi-GPT3.5. Figure 7 shows some relative results of different metrics for Readi-GPT3.5, compared with fine-tuned methods and the Golden. Additionally, we show the distribution of number of LLM-calls for editing for Readi-GPT3.5 in Figure 8.

It is shown that the initial reasoning path by Readi is already comparable with fine-tuned ones. With some necessary editing, Readi significantly gets closer to golden. In addition, without awareness of the environments, a large number of Readi's initial path (49%) get stuck at compound nodes, while fine-tuned methods well memorize the schemas in structured environments. However, with pertinent feedback upon instantiation errors, Readi substantially alleviates this problems and reaches higher QA performance than fine-tuned methods, again demonstrating that unfaithfulness still exits even with large-scale fine-tuning.

D.2 Performance of varied MAX_EDIT_TIME

We have reported the distribution of Editing times in Figure 5 (GPT4) and Figure 8 (GPT3.5). To some extent, the distribution showcases the quality of the Reasoning Path and Effectiveness of Editing module. Note that we set the MAX_EDIT_TIME to eight to test the quality, which can be set smaller to save resources. To further provide some insights, we experiment on different MAX_EDIT_TIME setting on CWQ subset in Table 9. Results show that with more chances to edit, the reasoning path gets better, further showing the effectiveness of editing. And four times of editing already achieves an comparable outcome.

MAX_EDIT_TIME	2	4	6	8
Hit@1	51.6	59.2	59.7	59.8

Table 9: Readi-GPT35's performance Performance for different MAX_EDIT_TIME on CWQ.

	GPT35	GPT4
CWQ	84.6	86.4
WebQSP	95.5	91.9

Table 10: Recall (%) of retriever for instantiation module on CWQ and WebQSP.

	CWQ	WebQSP
avg token (k)	52.9	37.6
avg cost ($)	0.090	0.064

Table 11: Average token cost of Readi (reasoning path generation and editing.

D.3 Performance of Retriever

The performance of retrieval may affect the instantiation. We calculate the recall of retriever in Table 10, showing that the performance of retrival is not the bottleneck of Readi. Moreover, our implementation is adapted from KB-BINDER (Li et al., 2023), which has shown effectiveness of binding relations to KG.

D.4 Token Cost

As indicated in our title, we discusses how LLMs can efficiently and faithfully interact with structured environments. By efficiency, we focus on *the way of interaction*, i.e. less LLM invocations (Section 1), to obtain pertinent information on large-scale structured environments for multi-hop reasoning. Some might concern the token cost of Readi as a measure of efficiency, though it is not our priority. Here, we report the average token cost per question in Table 11. We utilize Python library *tiktoken*, for GPT3.5 with six-shot for generation, 5-shot for editing and MAX_EDIT_TIME set to 4 for CWQ and 2 for WebQSP.

Note that it is non-trivial to directly compare the *exact token cost* with other methods, for the sake of number and format of examples, uncontrollable

Error Types	Instantiation Error	Answer not Covered	QA Error	False Negative
CWQ-GPT3.5	22%	28%	30%	20%
WebQSP-GPT3.5	6%	32%	32%	30%
CWQ-GPT4	18%	36%	20%	26%
WebQSP-GPT4	20%	26%	22%	32%

Table 12: Qualitative error analysis of Readi.

LLM-API traffic, and even the difficulty of the task. For example, StructGPT (Jiang et al., 2023a) claims to incorporate 32-shot examples, but the exact few-shot prompts are not provided. Readi's token cost can also be modulated by adjusting the MAX_EDIT_TIME. Furthermore, excessive calls or overly challenging tasks may cause the LLM-API to fail in providing the appropriate content or format, exceeding the MAX_TRY_TIMES, which also affects the token cost. That's also why we present the times of LLM-call to analyze to efficiency in the main body of our paper.

D.5 Qualitative Error Analysis

We analyze 200 randomly-sampled error cases of CWQ and WebQSP in Table 12. We divide the error into four categories (Section 4.4 and 6.2): *i.* "Instantiation Error": Even after Editing, the path is still not fully instantiated. *ii.* "Answer not Covered": Answer not in retrieved knowledge. *iii.* "QA Error": Even the answer is covered, the QA output is still wrong (e.g., hallucination). *iv.* "False Negative": For example, the ground truth of "`the two continents Turkey is in`" is "`Eurasia`", and the model output "`Europe and Asia`".

Based on Table 12, first, our reasoning path can be well instantiated and utilized to answer the multi-hop reasoning question. Second, there is still room for improvement of the retrieved knowledge, which we would like to focus in our future work. Third, the hallucination in QA reasoning still exits, with GPT4 performing better than GPT3.5.

E Generalizability

Note that our focus is not on comparative analysis of various LLMs. We propose Readi to enhance LLMs reasoning over structured environments. Therefore, we utilize LLMs with strong understanding and reasoning capabilities (*e.g.,* GPT3.5) to show their performance with Readi. Here we discuss generalizability of Readi framework.

Generalizability to domain-specific KG. The concept of reasoning path is a structured representation of a multi-hop reasoning process (Section 3). By the *intrinsic planning ability* in Section 1, we adopt LLMs strong question understanding and reasoning ability to directly generate the reasoning path. Here is an intuition of Readi: we humans can navigate multi-hop reasoning challenges across various domains by recognizing named entities and relations of the task. LLMs, trained on large-scale natural language corpus, may develop these abilities. Therefore, Readi is designed to operate without the need of domain-specific knowledge.

We have experimented on both large-scaled KG (CWQ and WebQSP) and domain-specific KG (MetaQA), where Readi shows impressive performance (Section 5). One concern is generalization to specific domain with massive relations. Here, we illustrate how Readi handles such cases with an example.

Consider a question "`What's the nationality of Lebron James`". Assume that the LLM has few domain knowledge and initially generates "`[Lebron James] nationality`" as the reasoning path, whereas the ground truth path in KG is "`[Lebron James] born_in - city_of`".

When instantiation (Section 4.3), we bind "`nationality`" to relations in the KG but find none of them connected to `[Lebron James]`, so we **invoke editing**. In the error message, we include relations around `[Lebron James]`, which is `[born_in, sex, father_of, . . . ]`, as a hint for Editing (Section 4.4 and Appendix B.2).

With the semantic understanding that "`born_in`" can relate to "`nationality of Lebron James`", the LLM can correct previous path. The ablation of Editing (Section 6.1) and analysis of Reasoning Path after Editing (Section 6.2) further demonstrate these. Note that relations around `[Lebron James]` is limited, compared with those in the whole KG.

The whole idea mirrors human cognitive strategies when conducting web searching for multi-hop reasoning tasks, where we flexibly adjust our subconscious plan based on information in browsers.

LLM	CWQ	WebQSP
Llama-2-70b-chat	49.2	76.9

Table 13: Readi's performance with other LLM-API on CWQ and WebQSP.

Method	CWQ	WebQSP
ToG-GPT3.5	57.1	76.2
Readi-GPT3.5	**57.9**	**77.5**
ToG-GPT4	67.6	**82.6**
Readi-GPT4	**69.2**	82.4

Table 14: Readi's performance compared with ToG over ToG's metric.

Genalizability to other LLMs. Here, we show Readi's geralizability to other LLMs. Due to API availability, we test Readi with Llama-2-70b-chat on CWQ and WebQSP samples. The MAX_EDIT_TIME is set to 4 for CWQ and 2 for WebQSP. For fair comparison, we adopt our GPT3.5-based QA reasoning module. Table 13 shows that Readi can generalize to other open-sourced LLMs.

We also discuss Readi's generalizability to fine-tuned models. RoG (Section 6.2) tunes a Llama 2 to generate a reasoning path, showing the feasibility of generating a path by fine-tuning. We've shown that Readi performs well with Editing by LLM to improve the path. One concern is about Editing with a fine-tuned model, depending on annotations, which we would go deeper in future works.

F Comparison with ToG and DATER

We further compare Readi with ToG (Sun et al., 2024) and DATER (Ye et al., 2023).

F.1 With ToG

Think-on-Graph (ToG) is an LLM-based KGQA method, which also reason the answer with an LLM in an information retrieval manner. Here, we discuss the differences between our Readi and ToG. As discussed in Related Works (Section 2), ToG adopts an LLM to step-by-step construct some Knowledge Graph (KG) instances, similar to beam search. Note that there is no reasoning path, *a structural representation of the question by LLMs* in ToG's methodology.

Interaction Paradigm. Our Readi is different in the following aspects: **1) Required Capability.** In ToG, the LLM traverses on the KG, similar to beam search, to filter out entities or relations by *scoring all candidates at each step*, which mainly requires discrimination ability. In contrast, our Readi requires the understanding and reasoning ability to *maintain a structural representation* of the entire question. **2) Way to Introduce the Environment.** To obtain structural information in large-scale KG, ToG queries for all relations around an entity or all tail entities around a relation as candidates to *score the distribution* by LLMs. Such process can be cumbersome because sometimes there are massive candidates on KG, *e.g., locations in France*. On the other hand, in Readi, we collect pertinent reasoning log only if the path instantiation goes wrong. Readi fully unleash the understanding ability of LLMs and ease the burden of multi-turn scoring for long candidate lists. **3) Grounding.** To obtain instances, in ToG, LLMs select items at each step, constrained by the beam size, hindering it from reasoning questions requiring logical operations (based on a set of instances for aggregation, comparison, etc). Conversely, for Readi, we introduce a novel instantiation module (Section 4.3) to obtain all instances based on constraints in the reasoning path.

Generalizability. Readi can generalize to more circumstances (Refer to Appendix E): **1) Supported Environments**, Think-on-Graph (ToG) only focuses on KG. However, Readi is a more general interaction framework for reasoning over structural environments. We showcase the concrete implementation of both KGQA (Section 4) and TableQA (Appendix A). **2) Supported Reasoning Tasks.** ToG, whose KG instances are restricted by the beam size, falls short for questions requiring logical operations, *e.g.,* aggregation, comparison, etc. However, Readi, adopting LLMs to maintain a structural representation of the question, is not restricted by any beam size, which can cover such reasoning cases by LLMs reasoning with retrieved instances.

Experimental Results. We did not cover the results of ToG in main results, for the sake of fair comparison. The evaluation metric from the official published code of ToG differs from the standard Hit@1. Conversely, Readi's evaluation strictly follows Tan et al. (2023), which is also used by all compared LLM-based baselines. While ToG does not provide any output files and the reported results are not reproducible, here we report Readi's re-

sults based on ToG's metrics in Table 14, where our Readi, with less LLM-calls, still outperforms ToG, overall.

F.2 With DATER

We have considered DATER but did not mention the result in Table 2 for the following reasons.

The base LLMs. DATER adopt Codex (OpenAI, 2021) as the backbone LLM, which is different from ours. Codex performs significantly better than GPT3.5 in TableQA (9.1% higher acc on WTQ (Liu et al., 2023)). However, Codex is close-soured and does not offer API-calls any more[3]. Therefore, it is not fair to directly compare Readi (based on GPT3.5) with DATER.

The way of reasoning over Tables. Our experiment is to show if our interaction framework, Readi, can adopt LLMs to effectively obtain useful information from structural environments and then reason the answer. Therefore, we compare with methods modeling TableQA as an *Information Retrieval* task (refer to Section 3 and Section 5.2). However, DATER uses a text2SQL model to obtain facts in table. This involves external tools (the SQL Interpreter) to handle the logical and numerical operations (e.g., aggregation, compare and calculation), which is not fair to compare with.

G Prompt List

Note that we do not modify prompts for baseline methods. Prompts for the vanilla LLMs is in Table 15, following (Sun et al., 2024). For TableQA, the prompts for vanilla LLMs is the same in Table 17. Detailed prompts for each module of Readi is in Table 16 (KGQA) and Table 17 (TableQA).

For number of few-shot demonstrations, on CWQ (Talmor and Berant, 2018) and WebQSP (Yih et al., 2016) we adopt 6 shots for reasoning path generation and 5 shots for other modules. For MQA(Zhang et al., 2018), the shot number for reasoning path generation follows (Li et al., 2023) and we adopt only 3 shots for editing and reasoning (we donnot design a reasoning module for MQA). For WTQ (Pasupat and Liang, 2015) and WikiSQL (Zhong et al., 2017), we use 7, 2, 7 shots for generation, editing and reasoning, respectively.

[3] https://platform.openai.com/docs/deprecations

Instruction Please answer the question:
Demonstration Example
Q: What state is home to the university that is represented in sports by George Washington Colonials men's basketball?
A: Washington, D.C..

Table 15: Prompts for vanilla LLMs for KGQA in our experiments.

Prompts for reasoning path generation
Given a question and some Topic Entities in the Question, output possible freebase Relation Paths starting from each Topic Entities in order to answer the question.
Demonstration Example
Question: Find the person who said "Taste cannot be controlled by law", where did this person die from?
Topic Entities: ["Taste cannot be controlled by law"]
Thought: There is only one topic entity, the answer is constrained by one path. For, the path from "Taste cannot be controlled by law", firstly, it should cover the person quote it. Second, it should cover the place where the person died.
Path: { "Taste cannot be controlled by law": [Taste cannot be controlled by law → people.person.quotations → people.deceased_person.place_of_death]}

Prompts for reasoning path editing
Task: Given an Inital Path and some feedback information of a Question, please correct the initial path.
Demonstration Example
Question: The movie featured Miley Cyrus and was produced by Tobin Armbrust?
Initial Path: Miley Cyrus→film.film.actor→film.film.producer Error Message
1. <compound node> in the end.
2. relation "film.film.producer" not instantiated.
Instantiation Context
Instantiate Paths: Miley Cyrus → film.actor.film → <compound node>
Candidate Relations
['film.director.film', 'film.performance.film', ...]
Corrected Path
Goal: The Initial Path starts from Miley Cyrus, which should cover the movies featured by Miley Cyrus.
Thought: In Instantiate Paths I know that Miley Cyrus acts some films, described by a compound node. In candidates, I find "film.performance.film" most relevant to get the films. Meanwhile, "film.film.producer" is not relevant to my Goal.
Final Path: Miley Cyrus→film.actor.film→ film.performance.film

Prompts for QA reasoning
Given a question and the associated retrieved knowledge graph triplets (entity, relation, entity), you are asked to answer the question with these triplets. If the given knowledge triples is not enough or missing, you can use your own knowledge. Use {} to enclose the answer! Please think step by step.
Demonstration Example
Q: The artist nominated for The Long Winter lived where?
Knowledge Triplets:
(The Long Winter, book.written_work.author, Laura Ingalls Wilder), (Laura Ingalls Wilder, people.person.places_lived, m.28e5697), (m.28e5697, people.place_lived.location, De Smet)
A: First, based on (The Long Winter, book.written_work.author, Laura Ingalls Wilder), the author of The Long Winter is Laura Ingalls Wilder. Second, based on (Laura Ingalls Wilder, people.person.places_lived, m.28e5697), (m.28e5697, people.place_lived.location, De Smet), Laura Ingalls Wilder lived in De Smet. So, the answer is {De Smet}.

Table 16: Detailed KGQA prompts for modules of Readi.

Prompts for reasoning path generation

You should predict the needed header and rows in a table for the question.

Demonstration Example

Question: what was the last year where this team was a part of the usl a-league?

| year | division | league | regular season | playoffs | open cup | avg. attendance |

| – | – | – | – | – | – | – |

| 2001 | 2 | USL A-League | 4th, Western | Quarterfinals | Did not qualify | 7,169 |

Thought:

First, according to headers and example rows, I need the years the team is in usl a-league league and return the latest year, so I need headers "year" and "league".

Second, I need to constrain "league" = "usl a-league" to know the years of this team as part of the "usl a-league", so I need "league": ["usl a-league"].

Chosen Headers: ["year", "league"]

Constrains: "league": ["usl a-league"]

Prompts for reasoning path editing

There are some mistakes in your previous header or constrains of a question. Follow the given feedback, fix your mistakes and give the correct header and constrains.

Demonstration Example

Question: what was the last year where this team was a part of the usl a-league?

| year | division | league | regular season | playoffs | open cup | avg. attendance |

| – | – | – | – | – | – | – |

| 2001 | 2 | USL A-League | 4th, Western | Quarterfinals | Did not qualify | 7,169 |

Wrong Answer:

Chosen Headers: ["year", "team"]

Constrains: "Team": ["usl a-league"]

Feedback:

1. Header ['team'] not in candidate Headers. You can only choose headers from ["year", ..."avg. attendance"].

Thought: First, previously I chose headers "year" and "yeam", but "team" is not in Header list. Following the feedback, I need the team in "league"="usl a-league" , so I need headers "year" and "league".

Second, I need to constrain "league" = "usl a-league".

Chosen Headers: ["year", "league"]

Constrains: "league": ["usl a-league"]

Prompts for QA reasoning

You should output the answer of question based on a table.

Output your answer in the last line as "Answer: ['your answer']"!

Demonstration Example

Question: what was the last year where this team was a part of the usl a-league?

Table:

Headers: league, year

item 1: (league, usl a-league); (year, 2001)

item 2: (league, usl a-league); (year, 2002)

item 3: (league, usl a-league); (year, 2003)

item 4: (league, usl a-league); (year, 2004)

Thought:

First, I know the years the teams is a part of usl a-league are 2001, 2002, 2003 and 2004 from the items in Table.

Second, I calculate the last year is 2004, so the answer is ['2004'].

Answer: ['2004']

Table 17: Detailed TableQA prompts for modules of Readi.