Middleware for LLMs:
Tools Are Instrumental for Language Agents in Complex Environments

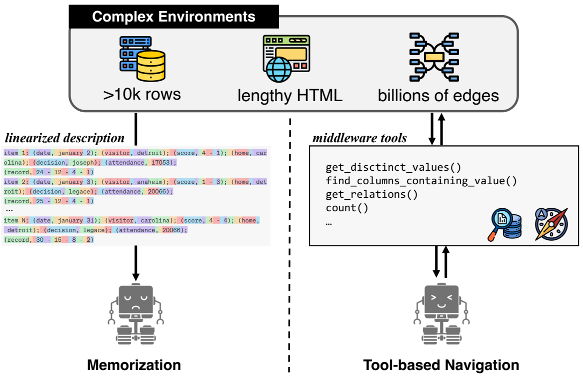
Anonymous ACL submission

Abstract

The applications of large language models (LLMs) have expanded well beyond the confines of text processing, signaling a new era where LLMs are envisioned as generalist language agents capable of operating within complex real-world environments. These environments are often highly expansive, making it impossible for the LLM to process them within its short-term memory. Motivated by recent research on extending the capabilities of LLMs with tools, this paper investigates the intriguing potential of tools to augment LLMs in handling such complexity. To this end, we design customized tools to aid in the proactive exploration within these massive environments. Such tools can serve as a *middleware* layer shielding the LLM from environmental complexity. In two representative complex environments—knowledge bases (KBs) and databases—we demonstrate the significant potential of augmenting language agents with tools in complex environments. Notably, equipped with these tools, GPT-4 achieves **2.8**× the performance of the best baseline in tasks requiring access to database content and **2.2**× in KB tasks. Our findings illuminate the path for advancing language agents in complex real-world applications.

1 Introduction

Large language models (LLMs) have demonstrated revolutionary language capabilities, demonstrating a human-like mastery over text (OpenAI, 2023a,b; Touvron et al., 2023; Jiang et al., 2024). However, the true ambition of AI extends well beyond the realm of text. The goal is to ultimately empower LLMs to act as generalist language agents that can aid humans across the multitude of complex real-world tasks (Yao et al., 2022; Schick et al., 2023; Gu et al., 2023), which often involve handling complex environments, whether it be browsing intricate webpages (Deng et al., 2023) or managing vast databases with millions of entries (Li et al., 2023a).



```
item 1: (date, january 2); (visitor, detroit); (score, 4 - 1); (home, car
olina); (decision, joseph); (attendance, 17053);
(record, 24 - 12 - 4 - 1)
item 2: (date, january 3); (visitor, anaheim); (score, 1 - 3); (home, det
roit); (decision, legace); (attendance, 20066);
(record, 25 - 12 - 4 - 1)
…
item N: (date, january 31); (visitor, carolina); (score, 4 - 4); (home,
 detroit); (decision, legace); (attendance, 20066);
(record, 30 - 15 - 8 - 2)
```

```
get_disctinct_values()
find_columns_containing_value()
get_relations()
count()
…
```

Figure 1: (*left*) When an LLM engages with a complex environment, it can develop an understanding by fitting the environment's description (*i.e.*, linearized tokens) into its short-term memory (*i.e.*, the LLM's input window). However, this method encounters drastic scalability issues as the complexity of the environment grows. (*right*) Another option is to furnish the LLM with a set of tools that assist it in actively engaging with the environment and acquiring the necessary information.

For LLMs to effectively serve as agents that ground human instructions into accurate actions within the environment, they must develop a robust understanding of the environment. The most direct method to achieve it is to linearize the environment into a sequence of tokens that fit into the LLM's short-term memory (*i.e.*, its input window) and have the LLM process the environment based on the linearized description (Tai et al., 2023; Shridhar et al., 2021; Liu et al., 2023). However, such a method faces steep challenges in scaling to more complex environments, primarily due to the input size limitations of LLMs. Also, discrete token descriptions may not reflect the most natural perception of the environment. Recent work has explored using tools to extend the boundary of the LLM's capacity (Li et al., 2023b; Qin et al., 2023b; Schick et al., 2023). The core idea is that LLMs can actively decide a proper tool to use, using language as a powerful vehicle of thought (Su, 2023). For example, the LLM may invoke a calculator when facing a computationally intensive math task. Intuitively, we can also equip the LLM with tools that enable navigating complex environments, so that the LLM can proactively invoke different tools to explore the environment, thus circumventing limitations posed by its short-term memory (Figure 1). However, this promising paradigm has been thus far underexplored. In this paper, we aim to delve into this paradigm and answer an intriguing question: ***How effectively can LLMs handle complex environments with the aid of tools?***

Answering this question requires equipping the LLM with a suite of tools designed to meet a wide range of needs within the target environment. In this paper, we carefully develop such tailored tools for two exemplar complex environments, *i.e.*, databases and knowledge bases (KBs). Unlike readily available Web APIs (Qin et al., 2023b) used in prior research, our tools have to be manually invented from scratch. In crafting these tools, we capitalize on the intuition of human information-gathering behaviors—such as performing keyword searches to identify a relevant database column or investigating the connections of a KB entity—to fulfill complex tasks in these intricate environments (Section 3.1). Ideally, these tools are designed to function as a *middleware* layer between the LLM and the environment, shielding the LLM from environmental complexity. With these specialized tools in place, we adapt ReAct (Yao et al., 2022), a standard framework that enables the LLM to synergistically combine reasoning with tool usage, as our reasoning algorithm to allow the LLM to effectively leverage the provided tools (Section 3.2). The combination of the crafted tools and the reasoning algorithm allows the LLM to actively explore the environment and ground human instructions into accurate actions. We call this framework Fuxi (*i.e.*, **f**lexible gro**u**nding with e**x**plorat**i**on).

With Fuxi, we evaluate different LLMs on benchmarks featuring complex tasks over the target environments, including a newly curated benchmark for the KB. The outcomes of our experiments are revealing: *LLMs equipped with customized tools demonstrate a significant enhancement in their ability to engage with complex environments, markedly surpassing the prior art.* In particular, despite its simplicity, Fuxi allows GPT-4 (OpenAI, 2023a) to achieve **2.8**× the performance of the best baseline in tasks requiring access to database content and **2.2**× in KB tasks. Our findings underscore the integral role of tool augmentation in enabling LLMs to handle complex environments.

Our main contributions are as follows: a) We develop Fuxi, a new framework with customized tools for two complex environments, to investigate the role of tools in handling complex environments with LLMs; b) We extensively evaluate six different LLMs on our carefully chosen benchmarks; c) Our analysis highlights a critical takeaway: augmenting LLMs with tools is crucial for successfully tackling complex environments, opening new possibilities to progress LLMs as generalist language agents for practical applications.

2 Related Work

Interface Complex Environments with LLMs. Existing methods that feed the environment directly into the LLM for grounding (Chandu et al., 2021) would fail in complex environments due to scalability issues. Specifically, these methods process the environment by linearizing it into discrete tokens (Hwang et al., 2019; Shridhar et al., 2021; Yu et al., 2023; Liu et al., 2023; Tai et al., 2023; Song et al., 2023). However, linearizing expansive environments like databases with millions of entries (Li et al., 2023a) or lengthy webpage HTML code (Deng et al., 2023) can often exceed an LLM's input length constraints. Alternative studies bypass the LLM's direct interaction with complex environments by generating ungrounded draft plans for post-processing grounding (Li et al., 2023c; Nie et al., 2023) or by using the LLM to assess grounded plans created via predefined rules (Gu et al., 2023). Such strategies do not fully utilize the LLMs' innate reasoning potential in actively navigating complex environments. In this paper, we explore a new paradigm where we can bypass these issues by equipping LLMs with a suite of comprehensive tools to actively gather necessary information about the environment upon demand, leveraging the LLMs' inherent reasoning capabilities. The most closely related work to ours is StructGPT (Jiang et al., 2023b). However, the narrow tool selection of StructGPT (*i.e.*, only two tools for KBs and three schema-level tools for databases) largely constrains its flexibility in perceiving the complex environment when handling diverse tasks.

Tool Learning. Tools are essential for enhancing the capabilities of LLMs (Schick et al., 2023; Qin et al., 2023a; Mialon et al., 2023; Hao et al., 2023). Existing research, such as ToolLLM (Qin et al., 2023b) and API-Bank (Li et al., 2023b), focuses on open-domain applications with a wide array of readily available RESTful APIs. In contrast, this paper specifically aims to study the potential of tools in augmenting LLMs to effectively execute tasks within complex environments, where we carefully craft the specialized tools for different environments by ourselves. In addition, research focusing on RESTful APIs typically displays shallow reasoning, while practical tasks within a complex environment typically entail a long sequence of actions (*e.g.*, querying a KB or browsing a webpage). To enable tool use in more intricate settings within a more specific complex environment, StructGPT (Jiang et al., 2023b) employs a predefined sequence of tool invocations; Chameleon (Lu et al., 2023) functions in an open-loop setting where the LLM directly produces a sequence for tool usage before any execution occurs. Both of them fail to seamlessly integrate the reasoning capacity of the LLM with the use of tools. In this paper, we build on ReAct to tightly synergize the generation of a reasoning step and corresponding tool use. Additionally, we introduce two simple yet effective strategies aimed at improving the accuracy of action prediction.

3 FUXI

FUXI equips LLMs with a suite of tools specifically tailored to support an extensive variety of operations and cater to the diverse needs within a complex environment $\mathcal{E}$. These tools can serve as a feature-rich *middleware* layer between the LLM and $\mathcal{E}$, abstracting the LLM from having to directly interact with all of its intricacies (Section 3.1). Furthermore, to fully unleash the inherent planning capabilities of LLMs in invoking proper tools, FUXI builds on ReAct (Yao et al., 2022) to seamlessly integrate chain-of-thought (CoT) reasoning (Wei et al., 2022) with tool use, with novel strategies to enhance action accuracy (Section 3.2). This unified framework allows us to reliably investigate the potential of LLMs in handling complex environments with the aid of tools.

3.1 Tools for Complex Environments

To evaluate the potential of LLMs in handling complex environments when equipped with tools, we need to first carefully craft the necessary tools for the environments. These tools should meet two essential criteria: 1) They should offer comprehensiveness, encompassing a broad spectrum of operations and needs. Broad coverage of tools is crucial for maximizing the potential of LLMs in planning. 2) The tools should prioritize ease of use, enabling the LLM to invoke them mostly with straightforward slot filling, thus shielding the LLM from the implementation details of the tools.

Databases In production scenarios, databases typically feature dozens of tables, with each table containing thousands of rows or more. A key task in such environments is performing data analysis through SQL queries. To bridge the gap between natural language instructions and SQL, LLMs are employed to automate the generation of SQL queries (*i.e.*, text-to-SQL parsing (Yu et al., 2018; Li et al., 2023a)). To support the LLM in crafting complex SQL queries, we introduce a set of specialized tools designed for interaction with intricate databases. These tools are divided into two main categories: navigational and functional. Navigational tools help the LLM to explore the environment (*e.g.*, `get_distinct_values()` and `find_columns_containing_value()`), while functional tools help check each SQL clause composed by the LLM. For example, `where()` verifies the legality of the WHERE clause and determines if the specified conditions can match any entries in the database. In total, we craft 12 tools for databases (Appendix A). The development of these tools is grounded in our domain expertise in SQL and databases.

KBs Modern KBs, such as FREEBASE (Bollacker et al., 2008), are vast repositories storing billions of facts as triples $\langle h, r, t \rangle$. These KBs cover a wide array of domains and support complex information-seeking tasks, including answering questions that require multi-hop reasoning. To support the LLM in engaging the extremely massive KB environments, we also devise a toolset tailored for KBs. Similarly, tools for KBs also include navigational tools and functional tools. The navigational tools facilitate efficient exploration of the KB by the LLM (*e.g.*, `get_relations()` and `get_attributes()`), while the functional tools support the LLM in executing precise operations, such as counting and intersecting two sets (*e.g.*, `intersection()` and `count()`). Both are critical for completing complex reasoning tasks on KB. A key concept in tools for KBs is a *vari-*

```
What is the website that belongs to the category of Pixiv and has a web API of developers.facebook.com?
```

```
Thought 1: I will first query the KB to find relations connected to the entity 'Pixiv'.
Act 1: get_relations(Pixiv)
Obs 1: [website.owner_new, website_category, website_owner]

Thought 2: I might use internet.website.category to find websites that belong to the category of Pixiv.
Act 2: get_neighbors(Pixiv, website_category)
Obs 2: variable #0

Thought 3: Now I need to find websites that have a web API of developers.facebook.com. To achieve this, I will
query the KB to find relations connected to it.
Act 3: get_relations(developers.facebook.com)
Obs 3: [api.site, api.protocols]
```

```
Thought 4: I might use internet.api.site to find websites that have a web API of developers.facebook.com.
Act 4: get_neighbors(#0, api.site)
Obs 4: api.site is not a relation from #0. Please retry.

Thought 5: Apologize for the mistake, I should use developers.facebook.com as the argument for get_neighbors.
Act 5: get_neighbors(developers.facebook.com, api.site)
Obs 5: variable #1
```

```
Thought 4: I might use internet.api.site to find websites that have a web API of developers.facebook.com.
```

```
Given the thought: I might use internet.api.site to find websites that have a web API of
developers.facebook.com.
Please make a choice from the options below. Specify your choice with an option letter.
Permissible actions:
a. get_neighbors(developers.facebook.com, api.site)
b. get_neighbors(developers.facebook.com, api.protocols)
c. Return #0 as the final answer
d. None of the actions corresponds to the thought.
Choose from a, b, c, d

My choice: a
```

Figure 2: The LLM is equipped with an array of tools to facilitate its engagement with complex environments (*e.g.*, a KB here). (a) The LLM may produce invalid actions (marked in pink). This can be mitigated by prompting it with an error message that encourages a reattempt (corrected action marked in green). (b) Alternatively, we can have the LLM first generate a thought, then predict an action based on it in a separate context (marked in blue), and finally insert the action back to the original context. Text marked in yellow are input from the environment.

able, representing a set of entities and typically generated as an intermediate result through the execution of functions like `get_neighbors()` or `intersection()`. The use of variables facilitates multi-hop reasoning across KBs, as it enables the natural linkage of a sequence of tool executions. In total, we implement 7 tools for KBs (Appendix A). Our design of KB tools tightly adheres to the common needs in knowledge base question answering (KBQA) (Gu et al., 2021; Cao et al., 2022).

3.2 Reasoning with Tools

We leverage ReAct to enable the LLM to effectively invoke our crafted tools. Unlike existing methods relying on rigid, human-defined workflows that follow fixed tool usage sequences (Jiang et al., 2023b), ReAct allows the LLM autonomy in proactively determining tool selection using CoT. Thus, ReAct allows us to tap into the full potential of the LLM's reasoning capabilities.

Formally, at each step t, the LLM makes predictions following a policy that maps a current context to an output: $\pi : c_t \rightarrow \hat{a}_t$, where

$$c_t = (\hat{a}_1, o_1 \cdots, \hat{a}_{t-1}, o_{t-1})$$

$$\hat{a}_t = r_t \oplus a_t$$

$\hat{a}_t$ is the concatenation of a rationale r_t (*i.e.*, a thought in CoT) and a concrete tool use a_t (*e.g.*, in Figure 2, $\hat{a}_1$ is the concatenation of **`Thought 1`** and **`Act 1`**), while o_t is an observation from the environment (*i.e.*, the execution result of a_t). In ReAct, the LLM jointly decodes $\hat{a}_t$ based on c_t for each step. However, originally designed for simpler tools like the Wikipedia Search API, ReAct is more susceptible to producing an invalid a_t that is unfaithful to r_t when applied to more nuanced tool usage. We propose two simple strategies to remedy this issue. The first strategy is to simply amplify

ReAct by providing detailed *error feedback* in case of incorrect tool usage by the LLM, followed by a prompt to retry based on these messages (see Figure 2(a)).[1] This relies on the LLM's capacity for self-correction through feedback (Gou et al., 2023; Chen et al., 2023), which may not always be reliable when the underpinning LLM is weak, potentially leading to the repetition of the same mistakes (Guan et al., 2023). Additionally, we present *decoupled generation*, where the LLM's policy π is split into two sequential phases (*i.e.*, $\pi \propto \pi_1 \circ \pi_2$), allowing for more nuanced control of its actions. Initially, the LLM only decodes a thought r_t following $\pi_1(r_t|c_t)$. Subsequently, the LLM predicts an action a_t in a *separate context*, incorporating both the thought r_t and a set of simple rules $\mathcal{M}$ that determines permissible actions of this step. This is further guided by π_2, formulated as $a_t \sim \pi_2(a_t|r_t, \mathcal{M})$. $\mathcal{M}$ encapsulates the governing rules of the environment (*e.g.*, the relation argument for `get_neighbors()` must be derived from the output of `get_relations()`, which is applied to the specified entity argument in prior steps), infusing prior knowledge into the LLM's decision-making process (see Figure 2(b)). The concrete prompts used by us are shown in Appendix C.

4 Benchmarks

The predominant tasks for databases and KBs are text-to-SQL parsing and KBQA. However, *popular benchmarks for them may fall short for evaluating language agents out-of-box.* Specifically, the majority of questions in popular KBQA datasets like WEBQSP (Berant et al., 2013; Yih et al., 2016) are one-hop or two-hop questions, for which we can effectively handle with existing semantic parsing methods (Gu et al., 2022). Additionally, the databases featured in SPIDER (Yu et al., 2018) and WIKISQL (Zhong et al., 2017) have limited complexity in terms of both schema design and the number of rows in the tables. This over-simplification enables the direct feeding of the database schema to the LLM, achieving strong performance without the need to access the actual content of the database (Rajkumar et al., 2022). Therefore, we need different benchmarks with complex environments and instructions that better mirror the real-world situations language agents must handle (see statistics of our benchmarks in Appendix B).

Databases For databases, we leverage BIRD (Li et al., 2023a), which is a recent dataset notable for its complexity, featuring intricate instructions over highly complex databases. There are originally two different settings in BIRD: with and without oracle knowledge, where the oracle knowledge supplies specific information about the target database needed to fulfill each task. For instance, *"Exclusively virtual refers to Virtual = 'F'"*. With such oracle knowledge, the complexity of the environments is substantially mitigated; it offers a shortcut for the task and eliminates the necessity for deep engagement with the database. This cheating setting is also unrealistic for practical applications. As a result, we stick to the setting without oracle knowledge. For each of the 1534 questions in BIRD's dev set, we manually label whether accessing the database content is necessary to compile the SQL queries, noting that access is unnecessary if all mentioned values in a question exactly match database cells. This facilitates decomposing the language agent's performance based on questions that require deeper database engagement (496 questions) versus not (1038 questions) and enables fine-grained insights into the LLM's performance. In addition to execution accuracy (**EX**) used in BIRD, which determines if the execution results of the predicted SQL match those of the ground truth SQL, we also evaluate whether the predicted SQL is a valid SQL query (**VA**).

KBs We curate KBQA-AGENT, a new test set sourcing from existing KBQA datasets that contain complex questions. In particular, we selected 500 diverse questions that involve at least three relations, or two relations coupled with an aggregation function (*i.e.*, **Counting** or **Superlative**). For each question, we annotate it with a ground truth sequence of actions based on the toolset defined by us. Specifically, KBQA-AGENT comprises questions from three KBQA datasets on FREEBASE: GRAILQA (Gu et al., 2021), COMPLEXWEBQ (Talmor and Berant, 2018), and GRAPHQ (Su et al., 2016), ensuring a wide range of question types and sources. KBQA-AGENT is designed to be more representative of challenging, real-world scenarios compared to existing benchmarks (Appendix B). It offers an ideal testbed for evaluating language agents in interacting with massive KBs. We assess this through two metrics: **F1** of answer entities and Validity (**VA**), a binary met-

---

[1] For databases, we directly use the error message from sqlite3. For KBs, we manually define several simple templates for error feedback along with each tool.

Model	Req. Cont. (N)		Req. Cont. (Y)		Overall	
	EX	VA	EX	VA	EX	VA
w/ Oracle Knowledge						
API Docs Prompt (Rajkumar et al., 2022)						
w/ GPT-3.5-turbo	38.1	78.4	32.1	74.6	36.1	77.2
w/ GPT-4	49.5	95.5	41.7	89.9	46.9	93.7
w/o Oracle Knowledge						
API Docs Prompt (Rajkumar et al., 2022)						
w/ GPT-3.5-turbo†	30.9	82.9	10.9	80.0	24.4	82.0
w/ GPT-4	38.2	91.6	13.8	93.1	30.4	92.1
StructGPT (Jiang et al., 2023b)						
w/ GPT-3.5-turbo	36.2	86.5	8.7	80.8	27.3	84.7
w/ GPT-4	40.7	93.4	13.5	91.1	31.8	92.6
Fuxi (*error feedback*)						
w/ GPT-3.5-turbo	**38.8**	**95.7**	**19.8**	**94.7**	**32.7**	**95.4**
w/ GPT-4	**45.1**	**98.8**	**38.3**	**97.2**	**42.9**	**98.3**

Table 1: Results on Bird's dev set. Performance of all baselines is obtained under a *zero-shot* setting. † denotes the best method *w/o* oracle knowledge on Bird's official leaderboard. The predictions with API Docs Prompt are directly supplied by the authors of Bird.

Model	Counting		Superlative		None		Overall	
	F1	VA	F1	VA	F1	VA	F1	VA
Pangu$^{\diamondsuit}$ (Gu et al., 2023)								
w/ GPT-3.5-turbo	10.1	**100.0**	9.0	**100.0**	23.4	**100.0**	18.1	**100.0**
w/ GPT-4	12.3	**100.0**	14.2	**100.0**	35.6	**100.0**	27.1	**100.0**
KB-Binder (Li et al., 2023c)								
w/ GPT-3.5-turbo (20-shot)	0.0	33.7	0.2	19.4	6.7	37.0	4.2	32.8
w/ GPT-4 (20-shot)	7.9	48.3	0.4	28.2	6.0	45.8	5.2	42.6
StructGPT (Jiang et al., 2023b)								
w/ GPT-3.5-turbo	4.5	50.6	3.9	51.5	11.4	57.1	8.6	54.8
w/ GPT-4	2.2	37.1	3.9	30.1	11.7	26.3	8.4	29.0
Fuxi (*error feedback*)								
w/ GPT-3.5-turbo	33.7	70.7	22.0	64.1	23.9	56.8	25.3	60.8
w/ GPT-4	70.7	96.6	39.9	74.5	55.8	74.0	55.1	78.0
Fuxi (*decoupled generation*)								
w/ GPT-3.5-turbo	**48.9**	97.7	**29.5**	88.0	**32.1**	77.3	**34.3**	83.0
w/ GPT-4	**74.1**	98.9	**42.6**	85.1	**61.0**	83.6	**59.3**	85.8

Table 2: Results on KBQA-Agent. All models are provided with *one-shot* demonstration except for KB-Binder, where we provide 20-shot demonstrations for optimal performance. ♢ indicates our reimplementation of Pangu, as the original code lacks support for chat models.

ric evaluating the LLM's ability to find an answer, whether correct or not.

5 Experiments

5.1 Setup

Implementation To concretely instantiate our tools for the two environments, we employ standard query interfaces for databases and KBs, specifically SQLite for databases and Virtuoso for KBs. We then prompt the LLM with the tool descriptions together with the input task instructions (Appendix C). Each environment exhibits its own unique characteristics and challenges. In KBQA, the arguments for each function are either a variable or an item from the KB schema (*i.e.*, a relation or an attribute). In contrast, in text-to-SQL parsing, the arguments can be more varied, ranging from a part of a SQL query to a complete query. This makes listing potential actions, as needed in *decoupled generation*, much more complex for text-to-SQL parsing. Therefore, we implement *error feedback* solely for text-to-SQL parsing.

For the underlying LLMs, we primarily compare Fuxi with baseline methods using two of the most advanced LLMs—GPT-3.5-turbo (OpenAI, 2023b) and GPT-4 (OpenAI, 2023a)—since our goal is investigating the full potential of tool-enhanced LLMs operating within complex environments. In addition, we also explore four open-source LLMs to more comprehensively evaluate

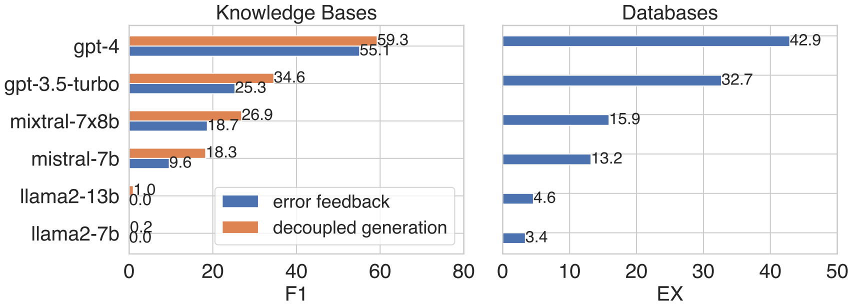


Figure 3: The open-source LLMs still largely lag behind GPT-3.5-turbo and GPT-4 in both environments.

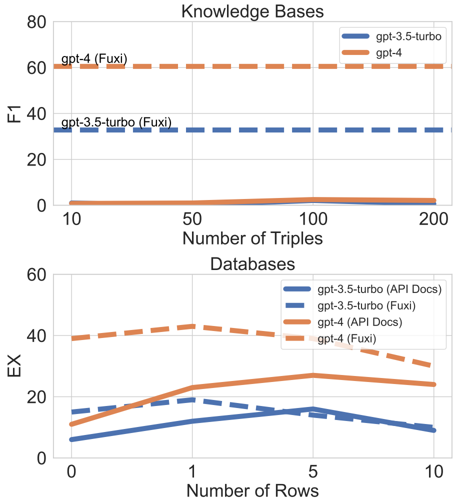


Figure 4: The customized tools can serve as effective *middleware* between the LLM and the environment.

our framework: Llama2-7B-Chat, Llama2-13B-Chat (Touvron et al., 2023), Mistral-7B-Instruct-v0.2 (Jiang et al., 2023a), and Mixtral 8×7B-Instruct-v0.1 (Jiang et al., 2024).

Baselines To fully understand the potential of tool augmentation for assisting LLMs in handling complex environments, we compare FUXI against an array of strong baselines. For text-to-SQL parsing, LLMs demonstrate a strong ability to compose SQL queries when properly prompted with the database schema (*i.e.*, API docs prompting (Rajkumar et al., 2022)). This also represents the current state-of-the-art prompting-based method when oracle knowledge is not available on BIRD's leaderboard.[2] In addition, we experiment with StructGPT (Jiang et al., 2023b), which represents an advanced text-to-SQL parsing method leveraging tools. For all methods on text-to-SQL parsing, we adopt the *zero-shot* setting. Unlike text-to-SQL parsing, directly prompting LLMs does not generate reasonable outputs for KBQA due to the massive size of the KB schema. Instead, existing KBQA methods based on LLMs typically follow two paradigms: either first generating an ungrounded program and then grounding the program to the KB schema afterwards (Li et al., 2023c; Nie et al., 2023), or gradually constructing a complex program and grounding it step by step (Gu et al., 2023). We compare FUXI with the most representative work from each paradigm, namely KB-Binder (Li et al., 2023c) and Pangu (Gu et al., 2023). We also include StructGPT as an additional baseline for tool use. For all KBQA methods except KB-Binder, we provide a *one-shot* demo to obtain more meaningful results.

5.2 Main Results

As shown in Tables 1 and 2, equipping LLMs with customized tools leads to significant improvement over previous standards, almost doubling or tripling the performance under multiple metrics. Specifically, API docs prompting can only feed the schema information to the LLM due to the vast amount of database content. As a result, it fails catastrophically on examples that require database content to compose the SQL query. In contrast, FUXI equips the agent with tools to actively navigate the database to collect relevant information for composing a SQL query. As a result, FUXI significantly closes the gap between performance on questions requiring database content and questions not requiring it when using GPT-4 (*i.e.*, 45.1% vs. 38.3%). Additionally, we notice that FUXI minimizes the gap between with and without oracle knowledge from 15.5% to 4.0% using GPT-4 and 11.7% to 3.3% using GPT-3.5-turbo. Finally, StructGPT demonstrates a similar trend to API docs prompting because its tools do not provide any information about the database content. For KBQA, FUXI demonstrates uniformly superior performance across different question types and significantly outperforms Pangu with both GPT-3.5-turbo and GPT-4. In particular, when equipped with GPT-4, FUXI + *decoupled generation* outper-

---

[2] https://bird-bench.github.io

forms Pangu by 32.2% in F1. As for the other two baselines, KB-Binder and StructGPT, both fail miserably on our challenging setting. On the one hand, KB-Binder only retrieves relations within two hops from the entities for grounding. However, most questions in KBQA-AGENT involve more than two relations. As a result, many of its drafted programs are unable to ground, which explains its low VA. On the other hand, StructGPT is heavily limited by its constrained toolset and cannot handle complex questions in KBQA-AGENT. Therefore, StructGPT frequently refuses to provide an answer (as revealed by its low VA) due to insufficient information. The strong performance of FUXI underscores that tools are instrumental for language agents in complex environments.

5.3 Experiments with Open-Source LLMs

To gain a more thorough insight, we also include experiments with four open-source LLMs (Figure 3). Our findings indicate that Llama2 models generally underperform compared to other LLMs, aligning with trends observed in other LLM leaderboards, such as Chatbot Arena (Zheng et al., 2023). Specifically, we find Llama2 models struggle with even generating grammatical tool use following our instruction. On the other hand, Mistral and Mixtral demonstrate much better performance than Llama2. In particular, Mixtral represents an advanced mixture-of-experts model that has demonstrated superior performance and even surpasses GPT-3.5-turbo on Chatbot Arena (Zheng et al., 2023). However, different from answering open-ended questions featured in Chatbot Arena, properly engaging with the complex environment demands the language agent to produce more precise actions that strictly conform to the task specification. There is still a performance gap between Mixtral and GPT-3.5-turbo in terms of predicting valid actions over intricate environments. Compared to GPT-3.5-turbo, Mixtral tends to output invalid actions more frequently. This also explains why *decoupled generation*, where the output space is strictly constrained to a list of valid actions, helps weaker models more. With FUXI + *decoupled generation*, using Mistral can almost match the best baseline performance with GPT-3.5-turbo, and using Mixtral can even match the best baseline with GPT-4. While stronger models like GPT-4 can effectively recover the mistake via *error feedback*, weaker models tend to benefit more from *decoupled generation*.

5.4 Tools as A Middleware Layer

To deepen our understanding of the integral roles of tools in aiding LLMs in accessing complex environments (*i.e.*, KB triples and database rows in our setup), we conduct further analysis by comparing FUXI with prompting baselines with different amounts of data items directly sampled from the environment (Figure 4). For the KB, we sample 10, 50, 100, and 200 triples from FREEBASE based on the three-hop neighborhood of each entity in a question. These triples are the top-ranked ones using a sentence-BERT retriever (Reimers and Gurevych, 2019) based on their similarity with the input question. We prompt the LLM directly with these sampled triples and request it to generate an answer to the given question. Given the extensive size of FREEBASE, accurately representing the environment with a mere subset of samples proves to be exceedingly difficult. Consequently, both GPT-3.5 Turbo and GPT-4 consistently yield an F1 score close to 0. For the database, we similarly augment API docs prompting with 1, 5, and 10 sampled rows for each table and evaluate on 100 random questions from BIRD that require accessing database content. Additionally, we also augment FUXI with the same sampled rows in the database setting. We observe that including more database rows initially boosts baseline performance but eventually decreases it. With FUXI, prompting the LLM with sampled rows yields minimal gain, and the standard setting without sampled rows already significantly outperforms all baselines. These results further confirm that the LLM, when augmented with tools, can effectively engage with complex environments, flexibly gathering the necessary information on demand and bypassing the limitations on the amount of data it can handle (*e.g.*, around 200 triples or 10 rows per table).

6 Conclusion

A pioneering vision is for language agents to assist humans in tackling intricate real-world tasks. This paper demonstrates that with meticulously-crafted tools acting as *middleware* between LLMs and complex environments, LLMs can substantially exceed current solutions. Our results spotlight these specialized tools' indispensable role in unlocking the potential of LLMs within complex real-world tasks previously posing immense challenges.

Limitations

In this paper, we aim to address the compelling question we posed: how effectively can LLMs handle complex environments with the aid of tools? We investigate this through evaluations in two exemplary environments: KBs and databases. While we achieve notable results in these environments, it is important to acknowledge that implementing customized tools for KBs and databases presents fewer challenges compared to environments without a straightforward query interface, such as a webpage or a physical environment. In future work, we plan to extend FUXI across a broader range of environments, aiming to fully realize the potential of language agents in complex environments through the integration of customized middleware tools.

Furthermore, the tools developed in this study are soley grounded in our experience. Despite this, our results already demonstrate the significant potential of augmenting LLMs with customized tools in complex environments, aligning with the primary objective of this paper. Nonetheless, to enhance performance further, adopting a more principled strategy for tool design is essential.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022. KQA Pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6101–6119. Association for Computational Linguistics.

Khyathi Raghavi Chandu, Yonatan Bisk, and Alan W. Black. 2021. Grounding 'grounding' in NLP. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 4283–4305. Association for Computational Linguistics.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *CoRR*, abs/2304.05128.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *CoRR*, abs/2306.06070.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. CRITIC: large language models can self-correct with tool-interactive critiquing. *CoRR*, abs/2305.11738.

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), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4928–4949. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: three levels of generalization for question answering on knowledge bases. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3477–3488. ACM / IW3C2.

Yu Gu, Vardaan Pahuja, Gong Cheng, and Yu Su. 2022. Knowledge base question answering: A semantic parsing perspective. In *4th Conference on Automated Knowledge Base Construction*.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *CoRR*, abs/2305.14909.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *CoRR*, abs/2305.11554.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *CoRR*, abs/1902.01069.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023a. Mistral 7b. *CoRR*, abs/2310.06825.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L'elio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *CoRR*.

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

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023a. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. *CoRR*, abs/2305.03111.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. API-Bank: A benchmark for tool-augmented llms. *CoRR*, abs/2304.08244.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023c. Few-shot in-context learning on knowledge base question answering. In *Annual Meeting of the 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. 2023. AgentBench: Evaluating llms as agents. *CoRR*, abs/2308.03688.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *CoRR*, abs/2304.09842.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey. *CoRR*, abs/2302.07842.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2023. Code-style in-context learning for knowledge-based question answering. *CoRR*, abs/2309.04695.

OpenAI. 2023a. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2023b. Models - OpenAI API. `https://platform.openai.com/docs/models/gpt-3-5`.

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. 2023a. Tool learning with foundation models. *CoRR*, abs/2304.08354.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. ToolLLM: Facilitating large language models to master 16000+ real-world apis. *CoRR*, abs/2307.16789.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *CoRR*, abs/2204.00498.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *CoRR*, abs/2302.04761.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. LLM-Planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Yu Su. 2023. Language agents: a critical evolutionary step of artificial intelligence. *yusu.substack.com*.

Yu Su, Huan Sun, Brian M. Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 562–572. The Association for Computational Linguistics.

Shuo Sun, Yuchen Zhang, Jiahuan Yan, Yuze Gao, Donovan Ong, Bin Chen, and Jian Su. 2023. Battle of the large language models: Dolly vs LLaMA vs vicuna vs guanaco vs bard vs ChatGPT - a text-to-SQL parsing comparison. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11225–11238, Singapore. Association for Computational Linguistics.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5376–5393. Association for Computational Linguistics.

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, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 641–651. Association for Computational Linguistics.

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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6069–6076. AAAI Press.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Appendices

In this supplementary material, we provide further details as follows:

- Appendix A: Detailed Tool Definitions
- Appendix B: Benchmark Statistics
- Appendix C: Prompts

A Detailed Tool Definitions

In this section, we detail the descriptions of our customized tools for both environments. Specifically, we implement 12 different tools for databases and 7 different tools for KBs. The tool selection is carefully made based on our domain knowledge of these environments. Note that, for databases, we direct prompt the LLM with the DB schema information in API docs format (Rajkumar et al., 2022), as a result, our tools focus on helping the LLM better engage with the database content.

Databases

Navigational tools for databases:

`find_columns_containing_value(value)`

This function can help to find columns that contain the given cell value, which can help you make better decisions in decoding the right column to use. Note that, the value here means cell value in the rows of the column, not the column name.
Prerequisite: n/a

`find_columns_containing_value_fuzzy(value)`

Sometimes find_columns_containing_cell_value may not find a column with the exact matched cell value. This function can help to find columns that potentially contain the target cell value with fuzzy matching. Note that, the value here means cell value in the rows of the column, not the column name.
Prerequisite: n/a

`get_distinct_values(table, column)`

Returns the distinct values in the given column. This may mainly help you make better decisions in decoding the right value to use.
Prerequisite: n/a

`is_value_in_column(table, column, value)`

Returns whether the given value is in the given column. You can use this function to better detect the right column to use.
Prerequisite: n/a

`get_date_format(table, column)`

Returns an example item of the given Date column. This may help you to better understand the date format in the column.
Prerequisite: n/a

`search_by_SQL(query)`

Executing a SQL query to search the table.
Prerequisite: n/a

Functional tools for databases:

`from(from_statement)`

This function specifies the FROM clause, e.g., from("FROM table1") or from("FROM table1 JOIN table2 ON table1.id = table2.id")
Prerequisite: n/a

`where(where_statement)`

This function specifies the WHERE clause, e.g., where("WHERE table1.id = 1").
Prerequisite: from

`select(select_statement)`

This function specifies the SELECT clause, e.g., select("SELECT table1.id").
Prerequisite: from, where

`group_by(group_by_statement)`

This function specifies the GROUP BY clause, e.g., group_by("GROUP BY table1.id").
Prerequisite: from, where, select

`having(having_statement)`

This function specifies the HAVING clause, e.g., having("HAVING table1.id = 1").
Prerequisite: from, where, select, group_by

`order_by(order_by_statement)`

This function specifies an additional constraint like ordering. For example, order_by("ORDER BY table1.id DESC LIMIT 3").
Prerequisite: from, where, select

Knowledge Bases

Navigational tools for KBs:

Dataset	# Table/DB	# Row/DB	% Require Cont.
WikiSQL (Zhong et al., 2017)	1	17	0.0
Spider (Yu et al., 2018)	5.1	2K	0.0
Bird (Li et al., 2023a)	7.3	549K	32.3

(a) Databases

Dataset	# Relations/KB	# Triples/KB	# Hops	% Have Aggr.
MetaQA (Zhang et al., 2018)	9	135K	2.1	0.0
WebQSP (Yih et al., 2016)	19K	3B	1.5	4.9
GrailQA (Gu et al., 2021)	19K	3B	1.4	18.5
KBQA-Agent (Ours)	19K	3B	2.9	38.4

(b) Knowledge Bases

Table A.1: Our curated benchmarks more accurately mirror real-world complexity, offering a more effective assessment of language agents. Aggr. denotes aggregation functions.

`get_relations(variable) -> list of relations`

A variable can be either an entity or a set of entities (i.e., the result of a previous query). This function helps to navigate all relations in the KB connected to the variable, so you can decide which relation is the most useful to find the answer to the question.
A simple use case can be 'get_relations(Barack Obama)', which finds all relations/edges starting from the entity Barack Obama.
The argument of get_relations should always be an entity or a variable (e.g., #0) and not anything else.
Prerequisite: n/a

`get_neighbors(v, r) -> variable`

Given a variable, this function returns all entities connected to the variable via the given relation. Note that, get_neighbors() can only be used after get_relations() is used to find a set of viable relations.
A simple use case can be 'get_neighbors(Barack Obama, people.person.profession)', which returns the profession of Obama in Freebase.
Prerequisite: get_relations

`get_attributes(v) -> list of attributes`

This function helps to find all numerical attributes of the variable. Please only use it if the question seeks for a superlative accumulation (i.e., argmax or argmin).
Prerequisite: get_neighbors

Functional tools for KBs:

`argmax(v, a) -> variable`

Given a variable, this function returns the entity with the maximum value of the given attribute. It can only be used after get_attributes() is used to find a set of viable attributes.
A simple use case can be 'argmax(variable, age)', which returns the oldest entity belonging to the variable.
Prerequisite: get_attributes

`argmin(v, a) -> variable`

Given a variable, this function returns the entity with the minimum value of the given attribute. It can only be used after get_attributes() is used to find a set of viable attributes.
A simple use case can be 'argmin(variable, age)', which returns the youngest entity belonging to the variable.
Prerequisite: get_attributes

`intersection(v1, v2) -> variable`

Given two variables, this function returns the intersection of the two variables. The two variables must be of the same type.
Prerequisite: get_neighbors

`count(v) -> int`

Given a variable, this function returns the number of entities belonging to the variable.
Prerequisite: get_neighbors

B Benchmark Statistics

In Table A.1, we present the statistics of Bird and KBQA-Agent, which we have chosen for our evaluation. Relative to established benchmarks in text-to-SQL parsing and KBQA, Bird and KBQA-Agent exhibit significantly greater complexity, making them more suitable for assessing the capabilities of language agents.

C Prompts

Instructions and demonstrations for using database tools are shown in Figure A.1. Note that, we also include the schema information of the database in API Docs in our prompt, which is not shown here. This design choice has been a common practice for text-to-SQL parsing with LLMs (Tai et al., 2023; Sun et al., 2023) Instructions and demonstrations

`Instruction:` You are an agent that answers questions based on the info in a database. To achieve this, you need to write the correct SQL queries step by step. The following functions can help you to better navigate the database.

```
1. find_columns_containing_cell_value(value: str) [...]
2. find_columns_containing_cell_value_fuzzy(value: str) [...]
3. get_distinct_values(table: str, column: str) [...]
4. is_value_in_column(table: str, column: str, value: str) [...]
5. get_date_format(table: str, column: str) [...]
6. search_by_SQL(query: str) [...]
```

In addition to these DB-navigation tools, to construct the target SQL query, you MUST use the following functions to construct the SQL query step by step.

```
7. from(from_statement: str) [...]
8. where(where_statement: str) [...]
9. select(select_statement: str) [...]
10. group_by(group_by_statement: str) [...]
11. having(having_statement: str) [...]
12. order_by(statement: str) [...]
```

You can only take ONE action at a time! For each step, you may first state your thought, then take an action following the format of 'Thought: ... Action: ...'.
Make sure that the specified action comes right after 'Action:'.

For example,
Thought: I need to check the distinct values of the column colB in table tabA to help me make better decisions.
Action:
`get_distinct_values(tabA, colB)`

Once you think you have gathered enough information, you can construct the SQL query and get the answer. Return your final SQL query by stating it right after 'Final Answer: ...'. Also, please do not include any linebreak (i.e., \n).
e.g., **Final Answer**: `SELECT x FROM tableA`

Figure A.1: Instructions for using database tools. Descriptions of tools are omitted.

for using KB tools are shown in Figure C.2. The instruction and demonstration for candidate selection in *decoupled generation* for KB is shown in Figure C.3.

Instruction: You are an agent that answers questions based on the knowledge stored in a knowledge base. To achieve this, you can use the following tools to query the KB.

```
1. get_relations(variable: var) -> list of relations [...]
2. get_neighbors(variable: var, relation: str) -> variable [...]
3. intersection(variable1: var, variable2: var) -> variable [...]
4. get_attributes(variable: var) -> list of attributes [...]
5. argmax(variable: var, attribute: str) -> variable [...]
6. argmin(variable: var, attribute: str) -> variable [...]
7. count(variable: var) -> int [...]
```

After a variable is produced along the process, you need to judge whether a variable is the final answer to the question. Each variable is represented as an id starting from 0. For example, #0 is the first variable, #1 is the second variable, and so on.

Once you find the answer, respond with '**Final Answer**: #id', where id is the id of the variable that you think is the final answer. For example, if you think #3 is the final answer, you MUST respond with '**Final Answer**: #3'.

You can only take ONE action at a time! After you get the observation from its execution, you can take another action. You can take at most 15 actions to find the answer to the question.

One-Shot Demonstration:

Question: the bipropellant rocket engine with kerosene and gas-generator cycle is designed by who?
Entities: [Gas-generator cycle, Kerosene]

Thought: I need to first find engines with gas-generator cycle. To achieve this, I will query the KB to find relations connected to the entity 'Gas-generator cycle' and see if any of them can help me find the answer.
Action: `get_relations(Gas-generator cycle)`
Observation: [spaceflight.rocket_engine_cycle.rocket_engines, spaceflight.satellite.orbiting, spaceflight.rocket_stage.fuel, spaceflight.satellite_manufacturer.spacecraft_manufactured, automotive.fuel.engines]

Thought: From the above relations, I might use spaceflight.rocket_engine_cycle.rocket_engines to find the engines of Gas-generator cycle.
Action: `get_neighbors(Gas-generator cycle, spaceflight.rocket_engine_cycle.rocket_engines)`
Observation: variable #0, which are instances of spaceflight.bipropellant_rocket_engine
Thought: I also need to find engines with kerosene. To achieve this, I will query the KB to find relations connected to the entity 'Kerosene'.
Action: `get_relations(Kerosene)`
Observation: [spaceflight.satellite_manufacturer.spacecraft_manufactured, automotive.fuel.engines, spaceflight.rocket_engine_cycle.rocket_engines]

Thought: From the above relations, I might use spaceflight.rocket_engine_cycle.rocket_engines to find engines with kerosene cycle.
Action: `get_neighbors(Kerosene, spaceflight.rocket_engine_cycle.rocket_engines)`
Observation: variable #1, which are instances of spaceflight.bipropellant_rocket_engine

Thought: The engines with both gas-generator cycle and kerosene cycle should be the intersection of variable #0 and variable #1.
Action: `intersection(#0, #1)`
Observation: variable #2, which are instances of spaceflight.bipropellant_rocket_engine

Thought: Now I need to find who designed such engines, which will be the final answer. To achieve this, I will query the KB to find relations connected to the variable #2.
Action: `get_relations(#2)`
Observation: [spaceflight.rocket_engine.manufactured_by, spaceflight.rocket_engine.designed_by, spaceflight.rocket_engine.design_period, spaceflight.rocket_engine.status]

Thought: From the above relations, I might use spaceflight.rocket_engine.designed_by to find the designer of the engines.
Action: `get_neighbors(#2, spaceflight.rocket_engine.designed_by)`
Observation: variable #3, which are instances of spaceflight.rocket_engine_designer

Thought: #3 is the final answer to the question, which represents the target rocket engine designer.
Final Answer: #3

Figure C.2: Instructions and a one-shot learning demonstration for using KB tools. Descriptions of tools are omitted.

Instruction: Given a thought and a set of candidate actions, your task is to select the most appropriate action that matches the thought.
Each action is indexed by a letter, and you need to respond with the letter of the selected action, not the action itself, i.e., output My choice: a, b, c, or d.

Demonstrations: Here are several examples,

Example 1

Thought: I need to first find engines with gas-generator cycle. To achieve this, I will query the KB to find relations connected to the entity 'Gas-generator cycle' and see if any of them can help me find the answer.
Candidate actions:
a. `get_relations(Gas-generator cycle)`
b. `get_relations(Kerosene)`
Make a choice from a, b.
My choice: a

Example 2

Thought: From the above relations, I might use spaceflight.rocket_engine_cycle.rocket_engines to find the engines of Gas-generator cycle.
Candidate actions:
a. `get_neighbors(Gas-generator cycle, spaceflight.satellite.orbiting)`
b. `get_neighbors(Gas-generator cycle, spaceflight.rocket_stage.fuel)`
c. `get_neighbors(Gas-generator cycle, spaceflight.satellite_manufacturer.spacecraft_manufactured)`
d. `get_neighbors(Gas-generator cycle, spaceflight.rocket_engine_cycle.rocket_engines)`
e. `get_neighbors(Gas-generator cycle, automotive.fuel.engines)`
Make a choice from a, b, c, d, e.
My choice: d

Example 3

Thought: The engines with both gas-generator cycle and kerosene cycle should be the intersection of variable #0 and variable #1.
Candidate actions:
a. `get_relations(#0)`
b. `get_relations(#1)`
c. `intersection(#0, #1)`
Make a choice from a, b, c.
My choice: c

Figure C.3: Candidate action selection in decoupled generation for KB.