Pandora: A Code-Driven Large Language Model Agent for Unified Reasoning Across Diverse Structured Knowledge

Anonymous ACL submission

Abstract

Unified Structured Knowledge Reasoning (USKR) aims to answer natural language questions (NLQs) by using structured sources such as tables, databases, and knowledge graphs in a unified way. Existing USKR methods either rely on employing task-specific strategies or custom-defined representations, which struggle to leverage the knowledge transfer between different SKR tasks or align with the prior of LLMs, thereby limiting their performance. This paper proposes a novel USKR framework named PANDORA, which takes advantage of PYTHON's PANDAS API to construct a unified knowledge representation for alignment with LLM pre-training. It employs an LLM to generate textual reasoning steps and executable Python code for each question. Demonstrations are drawn from a memory of training examples that cover various SKR tasks, facilitating knowledge transfer. Extensive experiments on four benchmarks involving three SKR tasks demonstrate that PANDORA outperforms existing unified frameworks and competes effectively with task-specific methods.

1 Introduction

Structured knowledge, such as tables, databases (DBs), and knowledge graphs (KGs), forms the foundation for many of today's intelligence applications, including legal judgment (Cui et al., 2023), disease diagnosis (Li et al., 2020), and investment analysis (Zhang et al., 2024a). As the core technology of these applications, Structured Knowledge Reasoning (SKR) has been a longstanding research focus in NLP, as demonstrated by tasks such as TableQA (Pasupat and Liang, 2015), Text-to-SQL (Yu et al., 2018), and KGQA (Yih et al., 2016). Using the powerful generation capabilities of Large Language Models (LLMs), recent works (Ye et al., 2023; Li et al., 2024; Nie et al., 2024) have made significant progress in reasoning tasks that involve structured single-type knowledge.

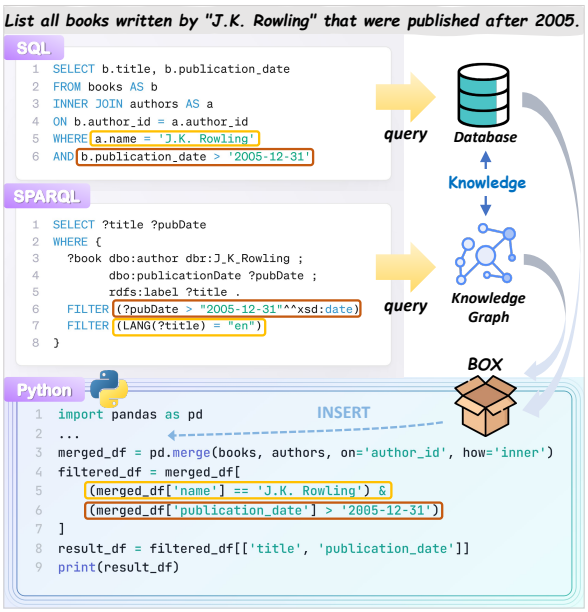

Figure 1: SQL, SPARQL, and PANDAS code derived from an NLQ, with matching colors highlighting corresponding query logic for clarity.

However, a complicated real-world application often integrates various types of structured knowledge. For instance, a medical decision support system (Antoniadi et al., 2021) may need to reason over both patient DBs and drug KGs. This requires the ability to handle various SKR tasks in a unified manner. Unfortunately, most existing methods struggle to bridge the gap between different SKR tasks due to task-specific designs (Pourreza and Rafiei, 2024; Nie et al., 2024).

Building on LLMs as the foundation, recent studies have proposed several unified SKR frameworks, such as StructGPT (Jiang et al., 2023), Readi (Cheng et al., 2024), and TrustUQA (Zhang et al., 2024b). Although these methods achieve uniformity by relying on task-specific strategies (StructGPT, Readi) or custom-defined representations (TrustUQA), their performance is limited. In particular, Readi and TrustUQA suffer from insuf-

ficient coverage of the reasoning over DB.

We believe that an ideal unified SKR framework should have two key characteristics: a) Facilitating knowledge transfer across diverse structured knowledge sources. For instance, as shown in Figure 1, the given SQL and SPARQL queries may differ in external syntax but share equivalent meanings. Transforming these queries into a unified representation can help LLMs leverage knowledge from other SKR tasks, enhancing target task performance. b) Representing different structured knowledge in a unified format familiar to LLMs. Code, being structured and compositional, is an ideal choice as LLMs excel in understanding, generating, and reasoning with code due to extensive pre-training on programming languages (Dubey et al., 2024). Converting diverse knowledge into code reduces the gap between input representations and the LLM's inherent understanding.

In this paper, we propose a new unified SKR framework, named PANDas cOde-dRiven Agent (PANDORA). It is composed of three key components: a well-aligned LLM, a reasoning memory, and a PYTHON interpreter. We start by transforming tables, DBs, and KGs into a unified representation built on the PANDAS library, referred to as PANDORA's BOX. For each NLQ, The PANDORA agent leverages the LLM to first generate textual reasoning steps, followed by executable PYTHON code. The generated code is then executed to derive the answer from the BOXes. The memory is constructed from the training examples and provides annotated demonstrations for *in-context learning* (ICL), enabling the LLM to learn the mapping from NLQs to PANDAS APIs. To leverage knowledge transfer across different SKR tasks, the demonstrations can be collected from any SKR task. In addition, the feedback from the code execution given by the interpreter further motivates the model to refine its reasoning steps and correct its code. We conducted extensive experiments on four widely-used datasets across three structured knowledge reasoning tasks, namely Text-to-SQL, TableQA, and KGQA. Experimental results demonstrate that our method outperforms all existing unified structured knowledge reasoning frameworks and matches the performance of task-specific methods. In summary, the contributions of this paper include:

- We propose a novel framework that utilizes LLMs to generate code-driven reasoning steps for diverse structural knowledge. To the best of our knowledge, this is the first time to leverage code as a unifying mechanism for SKR.
- We propose facilitating knowledge transfer across different structured knowledge sources by sharing demonstrations, thereby enhancing the performance of a unified framework.
- We conduct comprehensive experiments on multiple mainstream benchmarks, and our method achieves state-of-the-art performance in unified structured knowledge reasoning.

2 Preliminary

2.1 Structured Knowledge

Following Jiang et al. (2023), we focus on the following three types of structured knowledge:

Data Table A table can be regarded as $\mathcal{T} = (\{c_i\}_{i=1}^{C}, \{r_j\}_{j=1}^{R}, \{v_{i,j}\}_{i=1,j=1}^{C,R})$, where c_i denotes the i-th column name and r_j denotes a data record indexed by columns. $v_{i,j}$ denotes the content of the cell located at the intersection of c_i and r_j.

Database A database $\mathcal{D}$ consists of multiple tables, represented as $\mathcal{D} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T\}$. Besides the column names, the foreign keys across all tables are also available to link the data from two tables, denoted as $\{(c_i^p, c_j^q)\}$, where c_i^p and c_j^q denote the i-th and j-th columns in the $\mathcal{T}_p$ and$\mathcal{T}_q$, respectively.

Knowledge Graph A knowledge graph (KG) is typically a collection of subject-predicate-object triples, denoted by $\mathcal{K} = \{\langle s, p, o\rangle \,|\, s \in \mathcal{E}, p \in \mathcal{R}, o \in \mathcal{E} \cup \Gamma\}$, where $\mathcal{E}$, $\mathcal{R}$, and Γ denote the entity set, relation set, and type set respectively.

2.2 Problem Formulation

Given an NLQ $\mathcal{Q}$ and accessible structured knowledge $\mathcal{S}$ (e.g., a table $\mathcal{T}$, a database $\mathcal{D}$, or a KG $\mathcal{K}$), the objective is to generate an executable query that retrieves the desired answer $\mathcal{A}$ from $\mathcal{S}$.

2.3 BOX & Pandas Code Representation

To facilitate the transfer of knowledge across different SKR tasks, we propose a unified structure of knowledge representation, BOX.

Definition 1 (BOX) *A BOX is a data structure, denoted by $\mathcal{B} = (b, \Phi, \Psi)$, where b represents its textual name, $\Phi = \{\phi_i\}_{i=1}^{N}$, and $\Psi = \{[\psi_j^{\phi_i}]_{j=1}^{M}\}_{i=1}^{N}$. ϕ_i denotes a field that can be a column name in a table, or a KG relation. ψ_i^j represents the j-th value associated with the field ϕ_i. A value ψ_i^j can be a table cell content or a KG entity.*

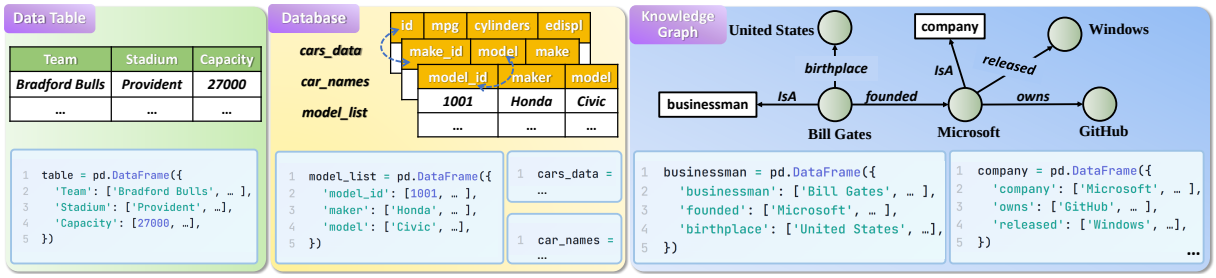

Figure 2: Examples of converting a table (a), a database (b), and a KG subgraph (c) into their corresponding BOX codes. For brevity, only a subset of the fields and values is shown. The blue arrows indicate the foreign key.

A BOX can be considered a dynamic table that is both understandable and operable by PYTHON. In our experiments, BOX is implemented using PANDAS[1], a PYTHON library designed for manipulating relational data. Specifically, a BOX $\mathcal{B} = (b, \Phi, \Psi)$ is represented in PANDAS code as:

```
import pandas as pd
b = pd.DataFrame({phi_1: [psi_1_1, ...],
                  phi_2: [psi_2_1, ...], ...})
```

where `phi_1` and `psi_2_1` are code rewritings of ϕ_1 and $\psi_1^{\phi_2}$, respectively. PANDAS provides versatile methods for manipulating BOX. For instance, in Figure 1, `pd.merge` is first used to join BOX `author` and BOX `book` to form a new BOX `merged_df`. Then, a filtering operation is applied to `merged_df` using a boolean index as follows:

```
merged_df[(merged_df['name'] == 'J.K.Rowling') &
          (merged_df['publication_date'] > 2005)]
```

PANDAS offers additional powerful tools like grouping, ordering, and aggregation, enabling it to handle a wide variety of query logic found in NLQs. More examples are listed in Appendix A.

3 Structured Knowledge to BOX

Figure 2 illustrates examples of converting structured knowledge $\mathcal{S}$ to their corresponding BOXes.

3.1 Table-to-BOX

As shown in Figure 2(a), a data table, denoted by $\mathcal{T} = (\{c_i\}_{i=1}^{C}, \{r_j\}_{j=1}^{R}, \{v_{i,j}\}_{i=1,j=1}^{C,R})$, can be seamlessly transformed into one box $\mathcal{B} = (b, \{\phi_i\}_{i=1}^{C}, \{[\psi_j^{\phi_i}]_{j=1}^{R}\}_{i=1}^{C})$ by treating each column name c_i as a field name ϕ_i and the content of each table cell $v_{i,j}$ as a field value $\psi_j^{\phi_i}$.

3.2 DB-to-BOX

As illustrated in Figure 2(b), for a database $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^{T}$, each table $\mathcal{T}_i \in \mathcal{D}$ is converted to a box $\mathcal{B}_i$ following the procedure described in Section 3.1. Meanwhile, foreign key information $\{(\phi_i^p, \phi_j^q)\}$ is retained, where ϕ_i^p and ϕ_j^q represent the i-th field in $\mathcal{B}_p$ and the j-th field in $\mathcal{B}_q$, respectively.

3.3 KG-to-BOX

Figure 2(c) shows an example of KG-to-BOX. Since the KG $\mathcal{K} = \{\langle s, p, o\rangle \mid s \in \mathcal{E}, p \in \mathcal{R}, o \in \mathcal{E} \cup \Gamma\}$ is too large, it is necessary to extract a subgraph for each NLQ $\mathcal{Q}$. Concretely, a depth-first search is initially performed to extract the H-hop subgraph $\mathcal{K}^* \subset \mathcal{K}$ for each topic entity mentioned in $\mathcal{Q}$. Here, $\mathcal{E}^* \subset \mathcal{E}$ and $\Gamma^* \subset \Gamma$ denote the entity set and type set of $\mathcal{K}^*$, respectively. To further narrow down the search space, the processed data from Xie et al. (2022) is utilized by retaining only the relations $\mathcal{R}^* \subseteq \mathcal{R}$ that demonstrate high embedding similarity to $\mathcal{Q}$. Subsequently, for each entity type $\gamma \in \Gamma^*$ and its corresponding entity set $\mathcal{E}_\gamma = \{e \mid \exists\langle e, \text{IsA}, \gamma\rangle \in \mathcal{K}^*\}$, a BOX $\mathcal{B}_\gamma = (\gamma, \Phi_\gamma^{1:N}, \Psi_\gamma^{1:N})$ is constructed. Specifically, the field names $\Phi_\gamma^{1:N} = \Phi_\gamma^1 \cup \Phi_\gamma^{2:N}$, where $\Phi_\gamma^1 = \{\gamma\}$, and $\Phi_\gamma^{2:N} = \{\phi_i \mid \phi_i \in \mathcal{R}^*, \exists\langle s, p, o\rangle \in \mathcal{K}^*, s \in \mathcal{E}_\gamma\}_{i=2}^{N}$ consists of 1-hop relations originating from the entities in $\mathcal{E}_\gamma$. Similarly, the field values $\Psi_\gamma = \Psi_\gamma^1 \cup \Psi_\gamma^{2:N}$, where $\Psi_\gamma^1 = \{[\psi_{1,j} \mid \psi_{1,j} \in \mathcal{E}_\gamma]_{j=1}^{M}\}_{i=1}^{1}$ corresponds to Φ_γ^1 and contains the entities of type γ. $\Psi_\gamma^{2:N} = \{[\psi_{i,j} \mid \exists\langle s, p, \psi_{i,j}\rangle \in \mathcal{K}^*, s \in \mathcal{E}_\gamma, p \in \mathcal{R}^*]_{j=1}^{M}\}_{i=2}^{N}$ corresponds to $\Phi_\gamma^{2:N}$ and consists of the 1-hop neighbors of the entities in $\mathcal{E}_\gamma$ through the relations in $\mathcal{R}^*$. From the perspective of the KG, for a box $\mathcal{B}_\gamma$, $\psi_{1,j}$ serves as the subject, ϕ_i represents the predicate, and $\psi_{i,j}$ acts as the object. In this way, multi-hop reasoning over the KG can be implemented by joining BOXes using `pandas.merge`, as shown in Figure 1. After all BOXes are built, the foreign key information is defined as $\{(\phi_i^p, \phi_j^q)\}$, where ϕ_i^p and ϕ_j^q share at least one common entity. The detailed KG-to-BOX algorithm is provided in Appendix B.3.

[1] https://pandas.pydata.org/

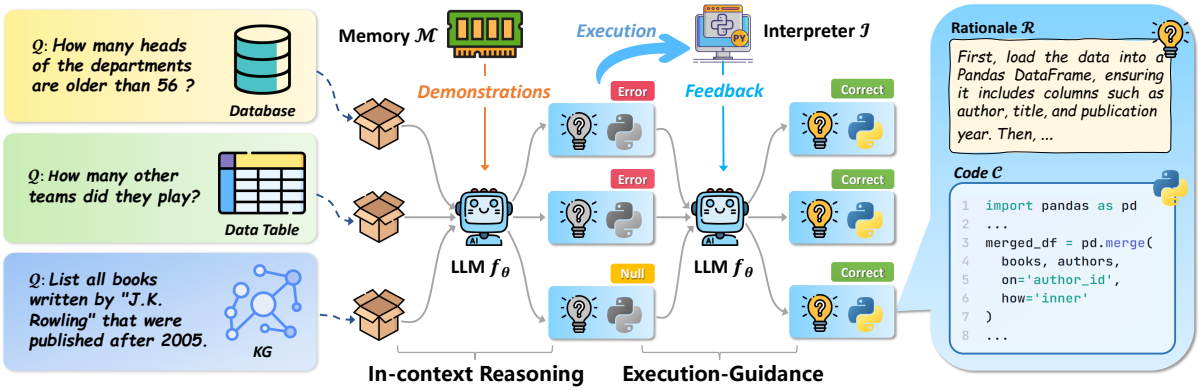

Figure 3: The inference process of our proposed agent PANDORA. PANDORA initially leverages the LLM f_θ to perform in-context reasoning, assisted by $\mathcal{M}$, to generate preliminary reasoning steps $\mathcal{R}$ and executable code $\mathcal{C}$. Subsequently, PANDORA provides the executed results back to f_θ for self-correction.

4 Pandora

4.1 Overview

PANDORA is an agent comprising three main components: a well-aligned LLM, f_θ, responsible for generating code-based reasoning steps; a memory, $\mathcal{M}$, which stores pairs of NLQs and valid reasoning steps for in-context learning; and a PYTHON interpreter, $\mathcal{I}$, used to execute the generated code. In general, PANDORA operates through two primary actions: *code-driven reasoning*, and *code execution*, which interact with an environment consisting of BOXes defined in PYTHON using $\mathcal{I}$.

4.2 Pandora Reasoning

Figure 3 illustrates the reasoning process of PANDORA. Initially, the structured knowledge $\mathcal{S}$ is transformed into a collection of BOXes $\mathcal{B}^*$. Next, $\mathcal{B}^*$ and $\mathcal{Q}$ are integrated into a prompt, $\mathcal{X}$, which is fed into f_θ. f_θ then generates the code-based reasoning steps $\mathcal{Y} = \mathcal{R}, \mathcal{C}$, where $\mathcal{R}$ represents the natural language rationale and $\mathcal{C}$ corresponds to the executable PYTHON code. Finally, the answer $\mathcal{A}$ is derived by executing code $\mathcal{C}$ on $\mathcal{B}^*$ using $\mathcal{I}$.

In-context Reasoning To help f_θ understand the mapping from NLQ to various PANDAS APIs, we leverage *in-context learning* (ICL) (Brown et al., 2020). Specifically, the prompt $\mathcal{X}$ is structured as:

$$\mathcal{X} = \mathcal{P}, \mathcal{Q}_1, \mathcal{B}^*_1, \mathcal{Y}_1, \ldots \mathcal{Q}_K, \mathcal{B}^*_K, \mathcal{Y}_K, \mathcal{Q}, \mathcal{B}^*$$

Here, $\mathcal{P}$ denotes the natural language instruction that guides f_θ to first generate $\mathcal{R}$ and subsequently $\mathcal{C}$. This adopts the concept of *chain of thought* (COT) (Wei et al., 2022). Notably, to manage the input length, all the values Ψ within $\mathcal{B}^*$ are excluded from all the prompts. $(\mathcal{Q}_k, \mathcal{B}^*_k, \mathcal{Y}_k)$ $(1 \leq k \leq K)$ constitute a demonstration retrieved from the memory $\mathcal{M}$. The complete prompt is provided in Appendix C.2. Then, f_θ generate $\mathcal{Y}$ by estimating

$$P(\mathcal{Y}|\mathcal{X}, \theta) = \prod_{j=1}^{|\mathcal{Y}|} P(y_j|\mathcal{X}, y_{<j}, \theta), \quad (1)$$

where y_j denotes the j-th token of $\mathcal{Y}$.

Shared Demonstration Retrieval Within the unified BOX representation, we assume that reasoning over structured knowledge $\mathcal{S}_a$ can potentially support f_θ in reasoning over another type of structured knowledge $\mathcal{S}_b$, as both share PANDAS APIs. Consequently, when retrieving $(\mathcal{Q}_k, \mathcal{B}^*_k, \mathcal{Y}_k)$ from $\mathcal{M}$, we do not require $\mathcal{Q}_k$ and $\mathcal{Q}$ to originate from the same SKG task. The K demonstrations of $\mathcal{Q}$ are selected based on the highest semantic similarity,

$$s(\mathcal{Q}_k, \mathcal{Q}) = \cos(g_\theta(\mathcal{Q}_k), g_\theta(\mathcal{Q})) \quad (2)$$

where $g(\mathcal{Q}) \in \mathbb{R}^d$ represents the embedding of $\mathcal{Q}$ obtained by an encoding-only LLM g_θ.

Execution Guidance To alleviate the *hallucination problem* (Huang et al., 2023) of generated code $\mathcal{C}$, we leverage the results of code execution as feedback to prompt f_θ to correct $\mathcal{C}$. In particular, when $\mathcal{C}$ is executed by the interpreter $\mathcal{I}$, if the result $\mathcal{A}$ satisfies the following two conditions, it is considered invalid and is fed back to f_θ: a) The execution of $\mathcal{C}$ raises an error. b) $\mathcal{A}$ is empty. The error information from $\mathcal{I}$ is recorded as $\mathcal{F}$. The prompt template for the execution guide is as follows:

$$\mathcal{X}_\mathcal{F} = \mathcal{P}_{\text{EG}}|\mathcal{Q}, \mathcal{B}^*, \mathcal{R}, \mathcal{C}|\mathcal{F}$$

Here, $\mathcal{P}_{\text{EG}}$ represents the natural language instruction. Subsequently, $\mathcal{X}_\mathcal{F}$ is fed to f_θ, which provides the corrected $\mathcal{Y}_\mathcal{F}$. This process continues until $\mathcal{Y}_\mathcal{F}$ is valid or exceeds the upper limit L we set.

4.3 Pandora Learning

The learning process of PANDORA mainly involves annotating the NLQs from the training data with PYTHON code and storing them in the memory $\mathcal{M}$. This can be divided into two stages.

4.3.1 Reasoning Memory Initialization

In the first stage, the training NLQs of the DB SKR task are annotated. Typically, the DB SKR task, like Spider (Yu et al., 2018), provides reliable human-written SQL labels, which can help reduce the difficulty of code annotation even in the absence of available demonstrations. In particular, given a training example $(\tilde{\mathcal{Q}}, \tilde{\mathcal{S}}, \tilde{\mathcal{Z}}, \tilde{\mathcal{A}})$, where $\tilde{\mathcal{Z}}$ represents the SQL label and $\tilde{\mathcal{A}}$ is the gold answer set, f_θ is employed to generate the code-based label $\tilde{\mathcal{Y}} = \tilde{\mathcal{R}}, \tilde{\mathcal{C}}$. The prompt format is structured as:

$$\mathcal{X} = \mathcal{P}_{\text{train}} | \tilde{\mathcal{Q}}, \tilde{\mathcal{S}}, \tilde{\mathcal{Z}}, \tilde{\mathcal{B}}^*$$

Where $\mathcal{P}_{\text{train}}$ is the instruction, and $\tilde{\mathcal{B}}^*$ is the converted BOX set derived from $\tilde{\mathcal{S}}$. To ensure the quality of $\tilde{\mathcal{Y}}$, the execution guidance (EG) strategy is employed. Moreover, the retrieved answers $\mathcal{A}$, obtained by executing $\tilde{\mathcal{C}}$, are compared with $\tilde{\mathcal{A}}$. The result of this comparison is fed back into f_θ to enable further self-correction. Ultimately, Finally, all samples with correct $\tilde{\mathcal{C}}$ are retained to form $\mathcal{M}_0$.

4.3.2 Multi-Task Adaptation

In the second stage, examples from $\mathcal{M}_0$ (DB SKR) are utilized as demonstrations to annotate the training NLQs for the KG and Table SKR tasks, instead of employing their specific labels. There are two main reasons: a) The examples in the Table SKR task consist only of NLQ-answer pairs and lack logical queries to describe the reasoning steps. b) Our experiments show that f_θ has a better understanding of SQL compared to SPARQL (see Table 4 for details). The EG strategy is also applied here. It should be emphasized that, for the three SKR tasks, only a small amount of data is selected for annotation, ultimately resulting in the memory $\mathcal{M}$.

5 Experiments

5.1 Datasets & Evaluation Metrics

We evaluated the methods on three SKR tasks:

DB SKR We use Spider (Yu et al., 2018), a human-annotated dataset designed for complex and cross-domain Text-to-SQL generation. The dataset contains diverse databases and intricate NLQs that require multi-step reasoning and a deep understanding of database schemas to construct accurate SQL.

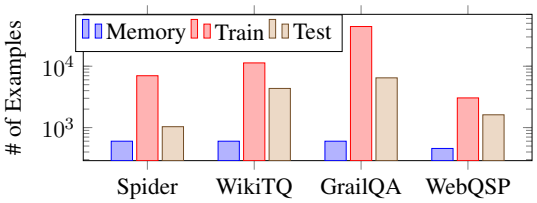

Figure 4: Statistics (Logarithmic y-axis) of $\mathcal{M}$.

Table SKR We use WikiTableQuestions (WikiTQ) (Pasupat and Liang, 2015), a dataset designed for question answering over real-world tables. This dataset requires performing operations such as aggregation, comparison, and filtering.

KG SKR We utilize GrailQA (Gu et al., 2021) and WebQSP (Yih et al., 2016), which feature NLQs that require up to multi-hop reasoning over the Freebase knowledge graph. These tasks involve entities, relations, and complex logical structures.

Following Jiang et al. (2023), we use Execution Accuracy (EX) and Denotation Accuracy (DA) to evaluate Spider and WikiTQ. For GrailQA and WebQSP, we use Hit@1 as the evaluation metric. In addition, we calculate the F1-score between the predicted answer set and the gold answer set.

5.2 Implementation Details

We utilized `gpt-4o-mini-2024-07-18` and `bge-large-en-v1.5` as f_θ and g_θ, respectively. The number of demonstrations for all in-context reasoning, K, was set to 10. The hop count H for the KG subgraph was set to 3 for GrailQA and 2 for WebQSP. For each NLQ, EG is executed up to $L = 3$ times. The statistics of the used datasets and our memory $\mathcal{M}$ are shown in Figure 4.

5.3 Compared Methods

1) **competitive baselines for single-type SKR task**. KG SKR: RnG-KBQA (Ye et al., 2022), TIARA (Shu et al., 2022), DecAF (Yu et al., 2023), KB-Binder (Li et al., 2023), and KB-Coder (Nie et al., 2024); DB SKR: DIN-SQL (Pourreza and Rafiei, 2023), DAIL (Gao et al., 2024), CodeS (Li et al., 2024), and DTS-SQL (Pourreza and Rafiei, 2024). Table SKR: TAPEX (Liu et al., 2022), Binder (Cheng et al., 2023), DATER (Ye et al., 2023), and STR (Kojima, 2024). 2) **Pure LLMs or unified SKG methods**. CodeLlama (Rozière et al., 2023), DeepSeek-Coder (Guo et al., 2024), UnifiedSKG (Xie et al., 2022), StructLM (Zhuang et al., 2024), StructGPT (Jiang et al., 2023), Readi (Cheng et al., 2024), TrustUQA (Zhang et al.,

Table 1: Results of the DB SKR task.

Method	Spider
	EX (%)
Task Specific Method	
DIN-SQL (Pourreza and Rafiei, 2023)	74.2
DAIL (Gao et al., 2024)	78.1
CodeS♠ (Li et al., 2024)	73.4
DTS-SQL♠ (Pourreza and Rafiei, 2024)	**85.5**
Unified Method	
CodeLlama-7B♠ (Rozière et al., 2023)	72.9
DeepSeek-Coder-7B♠ (Guo et al., 2024)	77.5
StructLM-7B♠ (Zhuang et al., 2024)	79.6
StructGPT (Jiang et al., 2023)	77.8
PANDORA	**81.3**

Table 2: Results of the Table SKR task.

Method	WikiTQ
	DA (%)
Task Specific Method	
TAPEX♠ (Liu et al., 2022)	57.5
Binder (Cheng et al., 2023)	65.0
DATER♠ (Ye et al., 2023)	**65.9**
STR♠ (Kojima, 2024)	65.7
Unified Method	
UnifiedSKG♠ (Xie et al., 2022)	49.3
StructLM-7B♠ (Zhuang et al., 2024)	50.1
StructGPT (Jiang et al., 2023)	52.2
Readi (Cheng et al., 2024)	66.7
TrustUQA (Zhang et al., 2024b)	66.2
PANDORA	**68.9**

2024b). We use ♠ to denote the fine-tuning using the target dataset.

5.4 Results for DB SKR

The results in Table 1 highlight the superiority of PANDORA, which outperforms all unified methods, including fine-tuned models such as StructLM-7B. Specifically, PANDORA achieves a 1.7% improvement over StructLM-7B, demonstrating its ability to align with LLM pre-training and transfer knowledge effectively across tasks. While task-specific fine-tuned methods like DTS-SQL continue to lead in performance, PANDORA narrows the gap, showcasing its advanced capabilities in DB SKR tasks without requiring task-specific fine-tuning.

5.5 Results for Table SKR

In Table 2, PANDORA establishes itself as the top-performing unified model, surpassing the best unified fine-tuned method, Readi, by 2.2%. Moreover, PANDORA slightly outperforms DATER, demonstrating that it can achieve competitive results without relying on task-specific fine-tuning.

Table 3: Results of the KG SKR task.

Method	GrailQA	WebQSP
	F1 (%)	Hit@1 (%)
Task Specific Method		
RnG-KBQA♠ (Ye et al., 2022)	76.9	-
DecAF♠ (Yu et al., 2023)	81.4	**78.7**
TIARA♠ (Shu et al., 2022)	**81.9**	76.7
KB-Binder (Li et al., 2023)	51.7	68.9
KB-Coder (Nie et al., 2024)	61.3	72.2
Unified Method		
UnifiedSKG♠ (Xie et al., 2022)	-	80.7
StructGPT (Jiang et al., 2023)	-	69.6
Readi (Cheng et al., 2024)	-	74.3
TrustUQA (Zhang et al., 2024b)	-	**83.5**
PANDORA	**77.3**	82.8

5.6 Results for KG SKR

In Table 3, PANDORA demonstrates its capability to tackle the more challenging GrailQA benchmark, surpassing existing unified approaches and outperforming the best non-fine-tuning method, KB-Coder, by 16%. On WebQSP, PANDORA delivers competitive results, trailing the existing top unified SKR method, TrustUQA in Hit@1 by just 0.7%.

5.7 Impact of different backbone LLMs

Figure 5 depicts the performance of PANDORA when employing `gpt-4o` and `gpt-4o-mini` as f_θ. To minimize the cost of using `gpt-4o`, we randomly sampled the same set of 200 NLQs from each dataset for evaluation in both settings. `gpt-4o` consistently outperforms `gpt-4o-mini`, especially on WikiTQ which lacks labels of logical form.

5.8 Ablation Study

We evaluated the performance of the proposed PANDORA by sequentially removing the following components: **a)** − **Execution-Guidance** (− **EG**): Generate reasoning steps $\mathcal{R}$ and codes $\mathcal{C}$ without receiving any feedback from the interpreter $\mathcal{I}$. **b)** − **Shared Demonstration** (− **SD**): Use only examples from the same dataset as the test NLQ as demonstrations. **c)** − **Similarity Retrieval** (− **SR**): Select demonstrations randomly, disregarding the process described in Equation (2). **d)** − **In-context Reasoning** (− **ICR**): Perform zero-shot inference to generate $\mathcal{R}$ and $\mathcal{C}$ without using any demonstrations. **e)** − **Code Style** (− **CS**): Directly generate labels for the original task label (e.g., SQL or SPARQL) instead of producing PANDAS code.

Ablation Setting	DB SKR		Table SKR		KG SKR				Average
	Spider		WikiTQ		GrailQA		WebQSP		
	EX	F1	DA	F1	F1	Hit@1	F1	Hit@1	
PANDORA	**81.3**	**84.2**	**68.9**	**70.0**	**77.3**	**82.0**	**73.6**	**82.8**	**77.5**
– EG	78.1	80.5	64.2	65.2	71.1	75.3	66.6	75.2	72.0
– EG – CS	76.6	79.6	51.8	51.7	44.4	46.9	64.2	67.5	60.3
– EG – SD	65.7	69.5	52.2	53.5	70.1	74.3	69.9	77.5	66.6
– EG – SD – SR	64.4	68.3	50.1	51.7	71.4	75.4	55.8	61.8	62.4
– EG – SD – SR – ICR	62.2	66.0	45.2	47.3	64.5	70.0	36.4	46.0	54.7

Table 4: Experimental results (%) of the ablation studies (using `gpt-4o-mini`). Here – denotes removing.

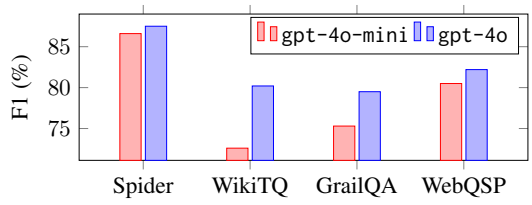

Figure 5: Results of PANDORA with different f_θ.

The ablation study in Table 4 highlights the contributions of each component in the proposed PANDORA framework. Removing EG results in an average performance drop of 5.5%, showing the value of interpreter feedback for refining reasoning and execution. Excluding CS further reduces performance by 17.2%, demonstrating the effectiveness of PANDORA code as a unified representation for structured reasoning tasks. Eliminating SD leads to a 5.4% drop, emphasizing the importance of knowledge transfer across datasets.

5.9 Impact of Demonstration Number

We investigated the impact of varying the number of demonstrations on the performance of the PANDORA. The experimental results are presented in Figure 6. Increasing the number of examples consistently improves Pandora's performance in different settings. On both the Spider and WikiTQ datasets, the removal of SD and SR results in a significant decline in performance. SD performs poorly on WebQSP, which could be attributed to the relatively simple SPARQL structures of the dataset. The diversity present in other datasets may introduce noise, reducing its effectiveness.

5.10 Error Analysis

To specifically evaluate the limitations of the proposed method, we randomly selected 50 samples from each of the three SKR tasks for error analysis. We have summarized the following types of errors: **a) Execution Failure**: The code fails to execute. **b) BOX Error**: The BOX in the code is wrong or missing. **c) Field Error**: Field error or missing in the code. **d) Reasoning Logic Error**: The relevant boxes and fields are correct, but the logic of the code is wrong. **e) Query Intent Error**: The intent of the query is misunderstood or incorrectly implemented. **f) Output Format Error**: The theoretical answer is correct, but the format returned does not match the annotated answer. Examples of cases d), e), and f) are provided below.

```
### SQL
SELECT T1.Id, T1.Maker
FROM CAR_MAKERS AS T1 JOIN MODEL_LIST AS T2 ON ...
HAVING count(*) >= 2
### Python
model_counts = model_list.groupby('maker').size()...
...
car_maker_ids = car_names.merge(model_counts, ...
```

```
### NLQ
what rail network does henley beach railway line
     belong to?
### Python
result = [['Henley␣Beach␣railway␣line',
rail_network.loc[..., 'rail_network'].values[0]]]
```

```
### Pandora Predicted Answer
["['Twenty-foot␣equivalent␣unit']"]
### Gold Answer
['Twenty-foot␣equivalent␣unit']
```

The distribution of various error types is shown in Figure 7. In the PANDAS reasoning environment, identifying fields (i.e., columns or KG relations) continues to be a major challenge. Furthermore, the wide range of PYTHON output methods contributes to errors in the output format.

6 Related Work

DB SKR. This task, often tackled via text-to-SQL, focus on converting NLQs into SQL queries. Conventional methods emphasize model architectures (Yu et al., 2021) and intermediate representations (Guo et al., 2019). Recent methods leverage LLMs with techniques like task decomposi-

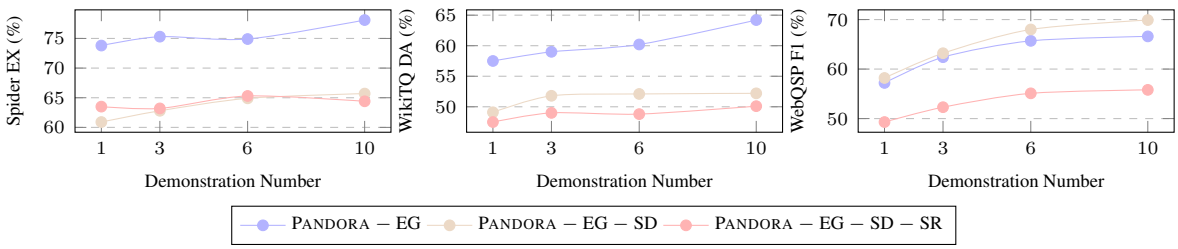

Figure 6: Impact of varying demonstration quantities on code-driven in-context reasoning.

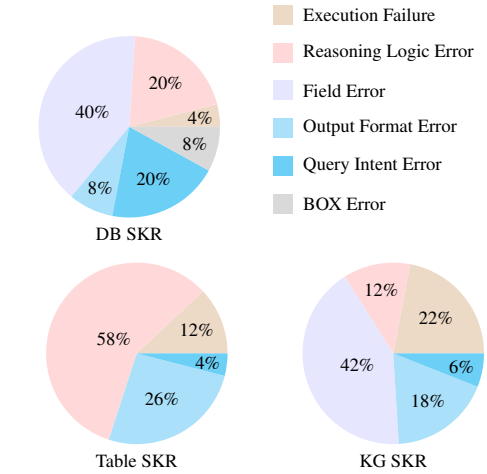

Figure 7: Distribution of various types of errors.

tion, chain of thought (Wei et al., 2022), and self-consistency (Wang et al., 2023), achieving improved results (Pourreza and Rafiei, 2023; Gao et al., 2024; Talaei et al., 2024; Pourreza et al., 2024). Other studies focus on fine-tuning open-source LLMs to match or surpass proprietary models (Li et al., 2024; Pourreza and Rafiei, 2024).

KG SKR. This task, addressed via KGQA, aim to resolve NLQs using KG. Traditional methods typically involve semantic parsing to generate executable logical forms (Berant et al., 2013; Yih et al., 2015) or embedding-based techniques for query matching (Das et al., 2018). Recent advancements with LLMs, such as DecAF (Yu et al., 2023), combine logical forms with direct answer generation, while KB-BINDER (Li et al., 2023) incorporates BM25 for improved performance. Similarly, KB-Coder (Nie et al., 2024) leverages ICL in a code-style paradigm, achieving better performance.

Table SKR. This task requires reasoning over NLQs and structured tabular data. Traditional methods rely on semantic parsing (Pasupat and Liang, 2015) or embedding-based methods for table-query matching (Yin et al., 2020; Deng et al., 2019). Modern LLM-powered models, such as TAPEX (Liu et al., 2022), Binder (Cheng et al., 2023), and DATER (Ye et al., 2023), excel by decomposing complex tables and NLQs into smaller components.

Unified Structured Knowledge Reasoning. Early unified frameworks, such as UnifiedSKG (Xie et al., 2022) and StructLM (Zhuang et al., 2024), integrate multiple structured knowledge datasets by fine-tuning models like T5 (Raffel et al., 2020) and CodeLlama (Rozière et al., 2023) to enhance structured knowledge understanding. More recent unified QA frameworks address diverse structured data types. For instance, StructGPT (Jiang et al., 2023) uses an iterative reading-then-reasoning strategy to retrieve evidence and generate answers, while Readi (Cheng et al., 2024) iteratively refines reasoning paths to extract evidence and produce answers. Despite these advancements, they rely on data-specific strategies, limiting uniformity. TrustUQA (Zhang et al., 2024b), the most related work, proposes a unified graph representation and generates explainable queries but redefines representations, creating gaps with LLMs' pre-trained knowledge. In contrast, our method, Pandora, adopts a code-based unified representation, inherently more aligned with LLMs' understanding.

7 Conclusion

In this paper, we proposed PANDORA, a unified SKR agent that uses PANDAS APIs as a standardized representation format for structured knowledge. By combining the rationale of natural language with executable PYTHON code, PANDORA enables iterative refinement of reasoning steps, memory storage for reuse, and effective cross-task knowledge transfer. Comprehensive experiments show that PANDORA outperforms existing unified methods and remains competitive with task-specific methods. In the future, we plan to extend the framework to support additional structured knowledge and explore more advanced reasoning capabilities to address a wider variety of SKR tasks.

8 Limitations

While the use of PANDAS APIs provides a standardized and efficient way to represent structured knowledge, it also introduces several limitations. First, PANDAS APIs are primarily designed for tabular data manipulation, which may limit their adaptability for tasks requiring reasoning over non-tabular or hierarchical data structures, such as graphs or nested datasets. Second, the reliance on these APIs can make it challenging to handle domain-specific reasoning that involves specialized libraries or techniques outside the scope of PANDAS, reducing the framework's versatility in highly specialized applications.

References

Anna Markella Antoniadi, Yuhan Du, Yasmine Guendouz, Lan Wei, Claudia Mazo, Brett A Becker, and Catherine Mooney. 2021. Current challenges and future opportunities for xai in machine learning-based clinical decision support systems: a systematic review. *Applied Sciences*, 11(11):5088.

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, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1533–1544. ACL.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Sitao Cheng, Ziyuan Zhuang, Yong Xu, Fangkai Yang, Chaoyun Zhang, Xiaoting Qin, Xiang Huang, Ling Chen, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. 2024. Call me when necessary: Llms can efficiently and faithfully reason over structured environments. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 4275–4295. Association for Computational Linguistics.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Junyun Cui, Xiaoyu Shen, and Shaochun Wen. 2023. A survey on legal judgment prediction: Datasets, metrics, models and challenges. *IEEE Access*, 11:102050–102071.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Yang Deng, Yuexiang Xie, Yaliang Li, Min Yang, Nan Du, Wei Fan, Kai Lei, and Ying Shen. 2019. Multi-task learning with multi-view attention for answer selection and knowledge base question answering. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 6318–6325. AAAI Press.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

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.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database

with intermediate representation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4524–4535. Association for Computational Linguistics.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *CoRR*, abs/2311.05232.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251. Association for Computational Linguistics.

Atsushi Kojima. 2024. Sub-table rescorer for table question answering. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/-COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 15422–15427. ELRA and ICCL.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

Linfeng Li, Peng Wang, Jun Yan, Yao Wang, Simin Li, Jinpeng Jiang, Zhe Sun, Buzhou Tang, Tsung-Hui Chang, Shenghui Wang, and Yuting Liu. 2020. Real-world data medical knowledge graph: construction and applications. *Artif. Intell. Medicine*, 103:101817.

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), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 6966–6980. Association for Computational Linguistics.

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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2024. Code-style in-context learning for knowledge-based question answering. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 18833–18841. AAAI Press.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1470–1480. The Association for Computer Linguistics.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. 2024. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. *CoRR*, abs/2410.01943.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: decomposed text-to-sql with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 8212–8220. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje F. 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, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 8108–8121. Association for Computational Linguistics.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024.

CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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 *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 602–631. Association for Computational Linguistics.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: generation augmented iterative ranking for knowledge base question answering. 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 6032–6043. Association for Computational Linguistics.

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 2023, Taipei, Taiwan, July 23-27, 2023*, pages 174–184. ACM.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1321–1331. The Association for Computer Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher 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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8413–8426. 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, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 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.

Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024a. Finsql: Model-agnostic llms-based text-to-sql framework for financial analysis. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, pages 93–105. ACM.

Wen Zhang, Long Jin, Yushan Zhu, Jiaoyan Chen, Zhiwei Huang, Junjie Wang, Yin Hua, Lei Liang, and Huajun Chen. 2024b. Trustuqa: A trustful framework for unified structured data question answering. *CoRR*, abs/2406.18916.

Alex Zhuang, Ge Zhang, Tianyu Zheng, Xinrun Du, Junjie Wang, Weiming Ren, Stephen W. Huang, Jie Fu, Xiang Yue, and Wenhu Chen. 2024. Structlm: Towards building generalist models for structured knowledge grounding. *CoRR*, abs/2402.16671.

A More Examples of Pandas Code

A.1 Examples of Text-to-SQL

Below are examples of converting SQL queries into equivalent Pandas code, covering cases like filtering, ordering, grouping, and nested queries.

```
### NLQ
Show the name and number of employees for the
     departments managed by heads whose temporary
     acting value is 'Yes'?

### SQL
SELECT T1.name , T1.num_employees
FROM department AS T1 JOIN management AS T2
ON T1.department_id = T2.department_id
WHERE T2.temporary_acting = 'Yes'

### Python
merged_df = pd.merge(
    department, management,
    on='department_id'
)
result = merged_df[
    merged_df['temporary_acting'] == 'Yes'
][['name', 'num_employees']]
```

```
### NLQ
What are the ids of the students who registered for
     course 301 most recently?

### SQL
SELECT student_id FROM student_course_attendance
WHERE course_id = 301
ORDER BY date_of_attendance DESC LIMIT 1

### Python
filtered_df = student_course_attendance[
    student_course_attendance['course_id'] == 301
]
sorted_df = filtered_df.sort_values(
    by='date_of_attendance',
    ascending=False
)
result = sorted_df.iloc[0]['student_id']
```

```
### NLQ
For each zip code, what is the average mean
     temperature for all dates that start with '8'?

### SQL
SELECT zip_code, avg(mean_temperature_f)
FROM weather
WHERE date LIKE "8/%"
GROUP BY zip_code

### Python
filtered_df = weather[
    weather['date'].str.startswith('8/')
]
result = filtered_df.groupby('zip_code')
         ['mean_temperature_f'].mean().reset_index()
result.columns = [
'zip_code',
'avg_mean_temperature_f'
]
```

```
### NLQ
Which department has more than 1 head at a time?
     List the id, name and the number of heads.

### SQL
SELECT T1.department_id , T1.name , count(*)
FROM management AS T2 JOIN department AS T1
ON T1.department_id = T2.department_id
GROUP BY T1.department_id
HAVING count(*) > 1

### Python
merged_df = pd.merge(department, management,
                     on='department_id', how='inner')
grouped = merged_df.groupby(
    ['department_id', 'name']
).size().reset_index(name='count')
result = grouped[grouped['count'] > 1]
```

```
### NLQ
What is the average bike availability in stations
     that are not located in Palo Alto?

### SQL
SELECT avg(bikes_available)
FROM status
WHERE station_id NOT IN (
    SELECT id
    FROM station
    WHERE city = "Palo Alto"
)

### Python
palo_alto_stations = station[
    station['city'] == "Palo Alto"
]['id']
filtered_status = status[
    ~status['station_id'].isin(palo_alto_stations)
]
result = filtered_status['bikes_available'].mean()
```

A.2 Examples of KGQA

Here are examples of converting SPARQL queries into their equivalent Pandas code. These examples cover cases such as multi-hop queries, counting, filtering by type, and finding argmax/argmin.

```
### NLQ
which hotel grading authority awards servigroup papa
      luna hotel?

### SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax
     -ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://rdf.freebase.com/ns/>
SELECT (?x0 AS ?value) WHERE {
    SELECT DISTINCT ?x0  WHERE {
        ?x0 :type.object.type :travel.
             hotel_grading_authority .
        ?x1 :type.object.type :travel.hotel_grade .
        VALUES ?x2 { :m.011nyts3 }
        ?x1 :travel.hotel_grade.awarded_by ?x0 .
        ?x2 :travel.hotel.grade ?x1 .
        FILTER ( ?x0 != ?x1
                && ?x0 != ?x2
                && ?x1 != ?x2 )
    }
}

### Python
result = hotel_grade[
    hotel_grade['hotel'] == 'Servigroup␣Papa␣Luna␣
         Hotel'
]['awarded_by'].tolist()
```

```
### NLQ
who is prime minister of japan 2011

### SPARQL
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT DISTINCT ?x
WHERE {
FILTER (?x != ns:m.03_3d)
FILTER (!isLiteral(?x) OR lang(?x) = '' OR
     langMatches(lang(?x), 'en'))
ns:m.03_3d ns:government.governmental_jurisdiction.
     governing_officials ?y .
?y ns:government.government_position_held.
     office_holder ?x .
?y ns:government.government_position_held.
     basic_title ns:m.060bp .
FILTER(NOT EXISTS {?y ns:government.
     government_position_held.from ?sk0} ||
EXISTS {?y ns:government.government_position_held.
     from ?sk1 .
FILTER(xsd:datetime(?sk1) <= "2011-12-31"^^xsd:
     dateTime) })
FILTER(NOT EXISTS {?y ns:government.
     government_position_held.to ?sk2} ||
EXISTS {?y ns:government.government_position_held.to
      ?sk3 .
FILTER(xsd:datetime(?sk3) >= "2011-01-01"^^xsd:
     dateTime) })
}


### Python
merged_data = governmental_jurisdiction.merge(
     government_position_held, left_on='
     governing_officials', right_on='
     government_position_held')
result = merged_data.loc[
    (merged_data['governmental_jurisdiction'] == '
         Japan')  &
    (merged_data['basic_title'] == 'Prime␣minister')
          &
    (merged_data['from'] <= '2011-12-31') &
    (merged_data['to'] >= '2011-01-01'),
    'office_holder'
].unique().tolist()
```

```
### NLQ
which countries does russia border

### SPARQL
PREFIX ns: <http://rdf.freebase.com/ns/>
SELECT DISTINCT ?x
WHERE {
FILTER (?x != ns:m.06bnz)
FILTER (!isLiteral(?x) OR lang(?x) = '' OR
     langMatches(lang(?x), 'en'))
ns:m.06bnz ns:location.location.adjoin_s ?y .
?y ns:location.adjoining_relationship.adjoins ?x .
?x ns:common.topic.notable_types ns:m.01mp .
}

### Python
merged_data = location.merge(adjoining_relationship,
      left_on='adjoin_s', right_on='
     adjoining_relationship')
result = merged_data.loc[
    (merged_data['location'] == 'Russia'),
    'adjoins'
].unique().tolist()
result = [x for x in result if x != 'Russia']
```

```
### NLQ
which countries does russia border

### SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax
     -ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://rdf.freebase.com/ns/>
SELECT (COUNT(?x0) AS ?value) WHERE {
SELECT DISTINCT ?x0  WHERE {
?x0 :type.object.type :religion.religious_leader .
VALUES ?x1 { :m.04l_pt }
?x0 :people.person.ethnicity ?x1 .
FILTER ( ?x0 != ?x1  )
}
}

### Python
result = [[len(religious_leader[religious_leader['
     religious_leader'].isin(person['person'][person
     ['ethnicity'] == 'jew'])])]]
```

A.3 Examples of TableQA

```
### NLQ
in which county did brown receive the most votes?

### Python
brown_votes = df['Votes'].str.replace(',', '').
    astype(int)
max_votes_index = brown_votes.idxmax()
result = [[df.loc[max_votes_index, 'County']]]
```

```
### NLQ
who was the top position american driver?

### Python
df['Pos'] = pd.to_numeric(df['Pos'], errors='coerce'
    )
result = df.nsmallest(1, 'Pos')['Driver'].tolist()
```

```
### NLQ
what is the total number of points india has scored
    throughout the rivalry?

### Python
valid_results = df['Results'].dropna().str.replace('
    Draw', '0 - 0')
scores = valid_results.str.extract(r'(\d+)\s*-\s*(\d
    +)')
scores = scores.fillna(0).astype(int)
india_scores = scores[1]
total_points_india = india_scores.sum()
result = [[total_points_india]]
```

```
### NLQ
how long did they play before they won a game?

### Python
first_game_date = pd.to_datetime(df['Date'].iloc[0])
wins = df[df['Result'].str.startswith('W')]
if not wins.empty:
    first_win_date = pd.to_datetime(wins['Date'].
        iloc[0])
    days_before_first_win = (first_win_date -
        first_game_date).days
else:
    days_before_first_win = 0
result = [[days_before_first_win]]
```

```
### NLQ
how far removed were the times from the fifth place
    and the tenth place contestant?

### Python
print(df['Rank'].unique())

rank_5 = df['Rank'].astype(int).eq(5)  # Creating a
    boolean series for rank 10

fifth_place_time = df.loc[rank_5, 'Time'].values[0]
    if df.loc[rank_5, 'Time'].size > 0 else None

tenth_place_time = df.loc[rank_10, 'Time'].values[0]
    if df.loc[rank_10, 'Time'].size > 0 else None

if fifth_place_time is None or tenth_place_time is
    None:
    result = [['Times not found for specified ranks.
        ']]
else:
    def convert_to_seconds(time_str):
        minutes, seconds = time_str.split(':')
        return int(minutes) * 60 + float(seconds)

    fifth_place_seconds = convert_to_seconds(
        fifth_place_time)
    tenth_place_seconds = convert_to_seconds(
        tenth_place_time)

    difference = abs(fifth_place_seconds -
        tenth_place_seconds)
    result = [[difference]]
```

```
### NLQ
how many times did the cowboys win the nfc
    championship in the 1970s?

### Python
nfc_cowboys_70s = df[(df['Team'] == 'Dallas Cowboys'
    ) & (df['Season'] >= 1970) & (df['Season'] <
    1980)]
result = [[nfc_cowboys_70s.shape[0]]]
```

```
### NLQ
who was the only administrator to have just a b.s.?

### Python
filtered_df = df[df['Educational Background'].str.
    contains('B.S.') & ~df['Educational Background'
    ].str.contains('Ph.D|M.S|M.A|Master')]
result = [[filtered_df['Name'].iloc[0]]] if not
    filtered_df.empty else [[]]
```

B Details of BOX Construction

B.1 Table-to-BOX

Algorithm 1 outlines the procedure for converting a table into a BOX. If the table's name is not provided, `Table` is used as the variable name for the BOX in the Pandas code.

B.2 DB-to-BOX

Algorithm 2 outlines the procedure for converting a database into a set of BOXes. Here, the function TABLETOBOX refers to the process described in Algorithm 1, which converts a table into a single BOX. In addition, the foreign key information of the database is retained.

B.3 KG-to-BOX

Algorithm 3 details the procedure for converting a KG into a set of BOXes.

The function DEPTHFIRSTSEARCH aims to retrieve the field records (i.e., the related triples) in the H-hop subgraph of the topic entities.

C Prompt Template

C.1 Prompt of In-context Reasoning

Figure 8 shows a prompt template for in-context reasoning.

C.2 Prompt of Execution Guidance

Figure 9 shows a prompt template for the execution guidance strategy.

Examples
[EXAMPLE]

You are an excellent data scientist and Python programmer. You can capture the link between the question and corresponding database schema and perfectly generate valid Python Pandas program or S-Expression query to answer the question.

Follow the instructions below:
Step 1. Comprehend the Question: Begin by thoroughly reading and understanding the main objectives and specific details outlined in the question. Determine whether the question requires an entity, a list of entities, or a numerical value. Then, break it down into logical steps and walk through the question systematically.
Step 2. Examine the Database Schema: Review the structure of the database schema to understand how data is organized. Identify relevant tables, columns, and values that are pertinent to the question. Use these elements to understand the question better and to create a link between the question and the database schema.
Step 3. Analyze the S-Expression Query: Carefully study the S-Expression query to identify key operations, keywords, and how they interact with the database schema. The query is designed to direct attention toward certain elements relevant to answering the question. Extract any keywords, phrases, or named entities that could provide further clarity or direction in formulating an answer.
Step 4. Convert Question to Pandas: Translate the question into an equivalent Python script using the Pandas library. Learn from the examples provided in ## Examples, try to understand the query logic they apply to solve their questions, and determine which parts can effectively help you solve the current question in ### Question. Make sure the parentheses and brackets in the script are placed correct especially if the generated code includes mathematical expression. In addition, always write down your answers in the json format structured as follows:
```json{"reasoning": "# describe the correct step-by-step reasoning process of how you convert the ### Question to codes.", "code": "# present only the Python codes to answer the ### Question without any schemas stored in pd.DataFrame."}```

Notice:
In the Python code you generate, the question should be solved step by step, rather than writing all the steps in only one line of code. Most importantly, all the final results should be consolidated into a list and stored in the variable `result: List[List[str]]`. The answer type can be a single entity, multiple entities, or numeric values. For a question, each answer should be placed in a list and the final complete answer consists of these lists. In this format, the 3 types of answers should look like this:
```python
# 1. **Single Entity**: the youngest monarch is whom?
>>> result
[['Galba']]
# 2. **Multiple Entities**: republic of indonesia is the home of what lakes?
>>> result
[['Lake Poso'], ['Lake Ranau'], ['Lake Matano'], ['Lake Toba']]
# 3. **Numerical Value**: how many papers are published in the journal? How many of them are in the engineering category?
>>> result
[[88], [10]]
```

Now let's think step by step and generate the Pandas Code:

Database Schema:
[SCHEMA]

Foreign Keys:
[FOREIGN_KEYS]

Question:
[QUESTION]

Pandas Code:

Figure 8: Prompt template for in-context reasoning of in-context reasoning.
```

Algorithm 1 Conversion from Table to BOX

Require: A data table $\mathcal{T} = (\{c_i\}_{i=1}^{C}, \{r_j\}_{j=1}^{R}, \{v_{i,j}\}_{i=1,j=1}^{C,R})$, where c_i denotes the i-th column name and each row r_j denotes a data record. $v_{i,j}$ denotes the content.

Initialize the BOX field set as $\Phi \leftarrow \emptyset$ and the BOX value set as $\Psi \leftarrow \emptyset$.
function TABLETOBOX($\mathcal{T}$)
    for $i = 1$ to C **do**
        $\Phi \leftarrow \Phi \cup \{c_i\}$ ▷ Treat each column as a field.
    end for
    for $i = 1$ to C **do**
        for $j = 1$ to R **do**
            $\Psi \leftarrow \Psi \cup \{v_{i,j}\}$ ▷ Treat each cell content as a field value.
        end for
    end for
    if $\mathcal{T}$ has a table name t **then** ▷ Name the each BOX in PANDAS code.
        $\mathcal{B} \leftarrow (t, \Phi, \Psi)$
    else
        $\mathcal{B} \leftarrow (\texttt{Table}, \Phi, \Psi)$
    end if
    return $\mathcal{B}$
end function
$\mathcal{B} = \text{TABLETOBOX}(\mathcal{T})$
return $\mathcal{B}$ ▷ A table can be converted into a single BOX.

Algorithm 2 Conversion from Database to BOX

Require: A database $\mathcal{D} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T\}$, where $\mathcal{T}_i$ denotes a table.

Initialize the BOX set as $\mathcal{B}^* \leftarrow \emptyset$.
for $i = 1$ to T **do**
    $\mathcal{B}_i = \text{TABLETOBOX}(\mathcal{T}_i)$ ▷ Follow Algorithm 1 to generate the BOX corresponding to each table.
    $\mathcal{B}^* \leftarrow \mathcal{B}^* \cup \{\mathcal{B}_i\}$
end for
return $\mathcal{B}^*$

Algorithm 3 Conversion from KG to BOX

Require: A knowledge graph $\mathcal{K} = \{\langle s, p, o\rangle \mid s \in \mathcal{E}, p \in \mathcal{R}, o \in \mathcal{E} \cup \Gamma\}$, where $\mathcal{E}$, $\mathcal{R}$, and Γ denote the entity set, relation set, and type set. A topic entity set $\mathcal{E}^* \subset \mathcal{E}$. A relevant relation set $\mathcal{R}^* \subset \mathcal{R}$.

Initialize the BOX set as $\mathcal{B}^* \leftarrow \emptyset$, the field record list $\Omega \leftarrow [\,]$, a visited entity set $\mathcal{V}$.
function DEPTHFIRSTSEARCH(e, ω, H)
  if $|\omega| = 2 \times H$ **then**
    Append ω to Ω
    return
  end if
  for $r \in \mathcal{R}^*$ **do**
    $\mathcal{E}^+ \leftarrow$ GetNeighborEntities$(e, r, +)$, $\mathcal{E}^- \leftarrow$ GetNeighborEntities$(e, r, -)$
    for $e^+ \in \mathcal{E}^+$ **do** ▷ Traverse the one-hop neighbor entities that start from e through r.
      if $e^+ \notin \mathcal{V}$ **then** ▷ Prune. Prevent passing through the same entity.
        Append $[\Gamma(e), \Gamma(e), e]$ to ω, Append $[\Gamma(e), r, e^+]$ to ω, $\mathcal{V} \leftarrow \mathcal{V} \cup \{e^+\}$
        DEPTHFIRSTSEARCH(e^+, ω, H)
        Pop(ω), Pop(ω), $\mathcal{V} \leftarrow \mathcal{V} \setminus \{e^+\}$
      end if
    end for
    for $e^- \in \mathcal{E}^-$ **do** ▷ Traverse the one-hop neighbor entities that end at e through r.
      if then$e^- \notin \mathcal{V}$ ▷ Prune. Prevent passing through the same entity.
        Append $[\Gamma(e^-), \Gamma(e^-), e^-]$ to ω, Append $[\Gamma(e^-), r, e]$ to ω, $\mathcal{V} \leftarrow \mathcal{V} \cup \{e^-\}$
        DEPTHFIRSTSEARCH(e^-, ω, H)
        Pop(ω), Pop(ω), $\mathcal{V} \leftarrow \mathcal{V} \setminus \{e^-\}$
      end if
    end for
  end for
end function
for $e \in \mathcal{E}^*$ **do**
  DepthFirstSearch$(e, [\,], H)$
end for
for $\omega \in \Omega$ **do** ▷ First, construct an empty BOX with only field names.
  for $(b, \phi, \psi) \in \omega$ **do**
    if $\nexists\, \Phi_b$ **then**
      $\Phi_b \leftarrow \emptyset$, $\Psi_b \leftarrow \emptyset$
    end if
    $\Phi_b \leftarrow \Phi_b \cup \{\phi\}$ ▷ Add field for each BOX.
  end for
end for
for $\omega \in \Omega$ **do** ▷ Second, fill values into each field.
  for $(b, \phi, \psi) \in \omega$ **do**
    if $\nexists\, \Psi_b^\phi$ **then**
      $\Psi_b^\phi \leftarrow [\,]$
    end if
    Append ψ to Ψ_b^ϕ ▷ Add value for each field.
    for $\tilde{\phi} \in \Phi \setminus \{\phi\}$ **do**
      Append "NA" to $\Psi_b^{\tilde{\phi}}$ ▷ Keep the number of rows the same for all columns.
    end for
  end for
end for
return $\mathcal{B}^*$

Task
A previous attempt to run a query did not yield the correct results, either due to errors in execution or because the result returned was empty or unexpected. Your role is to analyze the error based on the provided database schema and the details of the failed execution, and then provide a corrected version of the code.

Follow the instructions below:
Step 1. Review Database Schema: Examine the table schema and the provided foreign keys to understand the database structure. Identify relevant tables, columns, and values that are pertinent to the question. Use these elements to understand the question better and to create a link between the question and the database schema.
Step 2. Analyze Query Requirements: Consider what information the query is supposed to retrieve. Review the codes that was previously executed and led to an error or incorrect result. Analyze the outcome of the executed query to identify why it failed (e.g., syntax errors, incorrect column references, misuse of Python functions and logical mistakes).
Step 3. Correct the Code: Modify the code to address the identified issues, ensuring it correctly fetches the requested data according to the database schema and query requirements. Make sure the generated code should return all of the information asked in the question without any missing or extra information.
Step 4. Output Format: Present your respond in the json format structured as follows:
```json{"error": "# describe your analysis of the error and how to fix it", "reasoning": "# describe the correct step-by-step reasoning process of how you convert the ### Question to codes.", "code": "# present only the Python codes to answer the ### Question without any schemas stored in pd.DataFrame."}```

Execution Result
[EXECUTION]

Take a deep breath and try to think step by step about where your code is going wrong. Once you understand, find the correct Python code and rewrite your answer:

Figure 9: Prompt template for in-context reasoning of execution guidance.
```