Toward Structured Knowledge Reasoning:
Contrastive Retrieval-Augmented Generation on Experience

Jiawei Gu[1,2], Ziting Xian[1], Yuanzhen Xie[2], Ye Liu[2],
Enjie Liu[2], Ruichao Zhong[2], Mochi Gao[2], Yunzhi Tan[2,†], Bo Hu[2,†], Zang Li[2]
[1]Sun Yat-sen University [2]Platform and Content Group, Tencent

Abstract

Large language models (LLMs) achieve strong performance on plain text tasks but underperform on structured data like tables and databases. Potential challenges arise from their underexposure during pre-training and rigid text-to-structure transfer mechanisms. Unlike humans who seamlessly apply learned patterns across data modalities, LLMs struggle to infer implicit relationships embedded in tabular formats, especially in the absence of explicit structural guidance. To bridge this cognitive gap, we introduce **Co**ntrastive **R**etrieval-Augmented Generation on **E**xperience (**CoRE**), a framework that builds experience memory representations and enhances generalization through contrastive in-context learning (ICL) to simulate human-like knowledge transfer. Experiments on Text-to-SQL and TableQA show CoRE significantly improves performance, achieving average gains of 3.44% and 4.24%, with up to 17.2% on challenging tasks. Our Monte Carlo Tree Search (MCTS)-generated Experience Memory expands training data 8-9×, enhancing diversity and domain coverage. This training-free and continual method propels LLMs toward structured knowledge expertise. Code is available at: https://github.com/Kuvvius/CoRE.

1 Introduction

Large language models (LLMs) have shown remarkable capabilities on plain text but struggle with low-frequency, domain-specific concepts (Ling et al., 2023; Gu et al., 2024b) and structured data (Zhuang et al., 2024; Jiang et al., 2023; Sui et al., 2024) like tables and databases. This arises from two key issues: (1) limited exposure of LLMs to structural relationships during pre-training (Kandpal et al., 2023; Xu et al., 2024), and (2) difficulty in transferring reasoning from text to structured data (Xie et al., 2023; Ye et al., 2023). These issues are especially evident when reasoning with structured knowledge, where understanding the connections between structured data pieces and query text is crucial for accurate conclusions.

To reduce reliance on pre-training data, Retrieval-Augmented Generation (RAG) (Lewis et al., 2020a; Gao et al., 2023b) has become a widely adopted technique for injecting external knowledge into LLMs. While effective in text-based tasks (Ling et al., 2023), naïve RAG shown in Figure 1 (a), fails to improve performance due to the lack of specialized mechanisms for structured knowledge reasoning. Adding disconnected structured components (e.g. schemas and sub-tables) without grasping their interrelationships can disrupt the reasoning chain and hinder performance (Wang et al., 2023; Gao et al., 2023a).

While RAG struggles with structured knowledge reasoning, humans excel at leveraging and transferring past experiences to solve complex, specialized problems, even those they have less exposure to. According to *Deliberate Practice* (Ericsson and Pool, 2016), human experts deliberately expose themselves to diverse schema cases, building mental representations that compensate for initial knowledge gaps (more explanation in Appendix A.1). This enables them to solve unfamiliar problems by reflecting on both successful and failed attempts (e.g., joining database tables correctly or incorrectly). Through self-monitoring, they identify mistakes, refine strategies, and focus on transferring patterns to structural reasoning rather than just memorizing surface-level facts.

To adapt this human-like learning process to LLMs, the focus shifts to building dynamic experience and leveraging contrastive reflection. Previous works on contrastive learning (Khosla et al.,
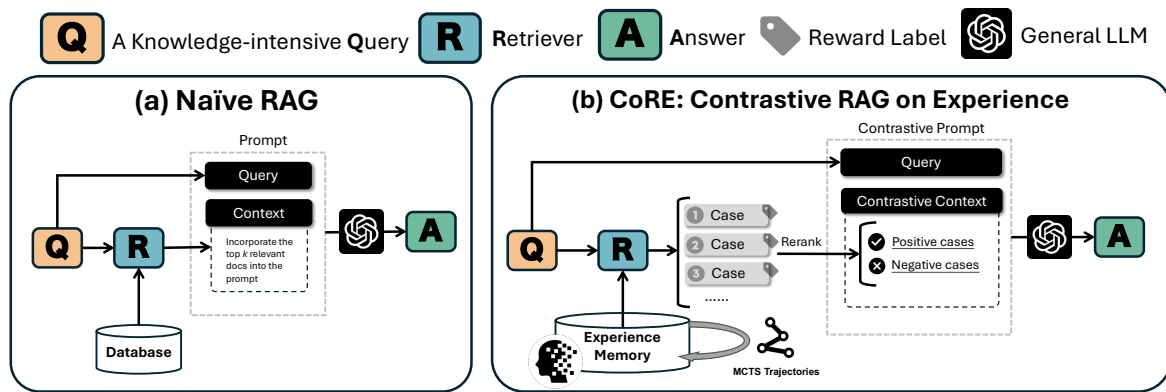
Figure 1: Comparison between naïve RAG and our CoRE.

2020), alignment (Ouyang et al., 2022; Lightman et al., 2023), and in-context learning (Chia et al., 2023) have shown that LLMs can learn more effectively by both positive and negative examples. This idea has been further supported by studies on reasoning (Putta et al., 2024), which improve the generalization of contrastive trajectories in complex, multi-step reasoning tasks.

Therefore, we introduce **Co**ntrastive **R**etrieval-Augmented Generation on **E**xperience (**CoRE**). *CoRE builds experience memory representations and incorporates contrasted examples*—both successful and failed cases—*into the context* to solve structured knowledge reasoning problems. As shown in Figure 1 (b), CoRE simplifies intricate structured reasoning problems into decomposable ones. It employs Monte Carlo Tree Search (MCTS) (Silver et al., 2017; Hao et al., 2023) along with a self-evaluation mechanism (Gu et al., 2024a) to mimic human *Deliberate Practice*, heuristically guiding the collection of successful and failed trajectories. Through *selection*, *expansion*, *simulation*, and *back-propagation*, question-answer pairs are generated with reward labels (representing the probability of success in intermediate steps). Subsequently, when a new problem arises, CoRE retrieves relevant cases and re-ranks them based on reward labels to present both successful (positive) and failed (negative) examples for contrastive reflection, enabling more accurate and knowledge-intensive answers.

Extensive experiments on structured knowledge reasoning tasks, such as Text-to-SQL and TableQA, show that CoRE significantly enhances both performance and efficiency, yielding average gains of 3.44% and 4.24% over baseline methods. In particular, CoRE achieves greater improvements on more challenging problems, with increases of 17.2% and 8.2% (Figure 4), respectively. The Experience Memory Dataset built from MCTS trajec-

tories expands the original training dataset by 8-9×, demonstrating increased diversity and domain coverage with diluted, decomposed high-density knowledge (Figure 3). Moreover, compared to few-shot approaches using only fixed examples, CoRE underscores the effectiveness of contrastive learning in context, achieving a 1.76% gain (Table 3). In summary, our main contributions are as follows:

1. **Experience Memory Construction**: We introduce a method to build experience memory using MCTS trajectories, capturing diverse reasoning experiences (both successful and failed), enabling better generalization in complex tasks with minimal initial exposure.

2. **Contrastive Reflection Mechanism**: CoRE introduces a contrastive learning method, which obtains two ranks through re-ranking in the RAG process and then incorporating retrieved content with specialized mechanisms for structured knowledge reasoning.

3. **Scalable Knowledge Retrieval**: CoRE offers a plug-and-play, training-free solution, allowing for continuous learning and retrieval in evolving environments by dynamic updates to the Experience Memory.

2 Preliminary

In this section, we present our main Research Questions and the formulation of **Structured Knowledge Reasoning**, which uses LLMs to solve complex reasoning tasks based on structured data.

> **Research Questions**
>
> (1) Can MCTS effectively imitate human's practice for heuristic data augmentation? (2) Can RAG enhance structured knowledge reasoning? (3) Does contrastive ICL outperform fixed examples?

Existing methods for structured knowledge reasoning seldom employ RAG because retrievable, domain-specific data is scarce. Instead, retrieval from general documents often introduces noise, leading to a negative impact (Yan et al., 2024a; Liu et al., 2024). For instance, DAIL-SQL (Gao et al., 2023a) restricts retrieval to training data, limiting diversity and coverage. Second, agent-based methods (Wang et al., 2023; Jiang et al., 2023; Pourreza and Rafiei, 2024) rely on fixed few-shot examples without incorporating dynamic contrastive learning for human-like adaptation. Third, most approaches lack systematic mechanisms to evaluate positive/negative signal impacts during reasoning. These gaps drive our investigation into MCTS-based augmentation, RAG optimization, and contrastive learning for structured knowledge reasoning with LLMs.

Formalization Structured knowledge reasoning can be described as a question-answering task. Given a triple $\mathcal{X} = (\mathcal{Q}, \mathcal{K}, \mathcal{A})$, where $\mathcal{Q}$ represents a natural language question, $\mathcal{K}$ is a structured knowledge representation, and $\mathcal{A}$ is an evidence-supported answer. Based on the natural language question $\mathcal{Q}$ and the accessible structured data $\mathcal{K}$ (e.g., a table or database), the first step is to extract useful evidence $\mathcal{E}$ from $\mathcal{K}$ (i.e., structured knowledge grounding). Notably, $\mathcal{E}$ remains a structured, high-density knowledge representation. Then, the LLM generator $\mathcal{G}$ uses this evidence to generate the expected answer $\mathcal{A}$ in response to the question $\mathcal{Q}$. According to the task requirement, the generated answer $\mathcal{A}$ must be evidence-supported, but either free-form answers in natural language or structured expressions (e.g. SQL statements) to be executed for obtaining the answer from $\mathcal{K}$. This process can be formalized as:

$$\mathcal{A} \overset{\text{reasoning}}{\sim} \mathcal{G}\left(\mathcal{Q}, \left(\mathcal{E} \overset{\text{grounding}}{\leftarrow} (\mathcal{Q}, \mathcal{K})\right)\right) \quad (1)$$

In our work, **we focus on improving reasoning over extracted structured evidence, excluding the grounding phase**. To better illustrate our task, we detail two types of reasoning formulation on databases and data tables:

Text-to-SQL We define a database $\mathcal{K} = \{\mathcal{T}, \mathcal{C}\}$, where $\mathcal{T} = \{t_1, t_2, \ldots, t_{|\mathcal{T}|}\}$ denotes tables and $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$ denotes columns, with foreign keys $\{(c_i^{(k)}, c_j^{(h)})\}$ linking them; consequently, the grounding phase selects from $\mathcal{K}$ the relevant tables and columns for $\mathcal{Q}$, forming evidence $\mathcal{E} = \{(t_k, c_i) \mid t_k \in \mathcal{T}_\mathcal{E}, c_i \in \mathcal{C}_\mathcal{E}\}$ that the generator

then uses to produce the answer $\mathcal{A}$. The reasoning process employs $\mathcal{G}$ to derive the final answer by sampling from its generative distribution. For more specific examples, refer to Appendix C.

TableQA We model a data table $\mathcal{K} = \{\mathcal{C}, \mathcal{R}\}$, where $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$ are columns and $\mathcal{R} = \{r_1, r_2, \ldots, r_{|\mathcal{R}|}\}$ are rows, with each cell value denoted by $v_{i,j}$; accordingly, the grounding process identifies from $\mathcal{K}$ the pertinent columns $\mathcal{C}_\mathcal{E}$ and rows $\mathcal{R}_\mathcal{E}$ based on $\mathcal{Q}$, forming evidence $\mathcal{E} = \{v_{i,j} \mid c_i \in \mathcal{C}_\mathcal{E}, r_j \in \mathcal{R}_\mathcal{E}\}$ that supports $\mathcal{G}$ deriving $\mathcal{A}$.

3 CoRE

We present our <u>CoRE</u> framework in this section. As illustrated in Figure 2, it consists of three key modules: <u>C</u>ontrastive Thinker (§ 3.3), <u>R</u>etriever (§ 3.2) and <u>E</u>xperience Memory Buildier (§ 3.1).

3.1 Experience Memory Builder

To tackle complex reasoning tasks involving questions $\mathcal{Q}$ with structured knowledge $\mathcal{E}$, humans and agents decompose problems into simpler sub-questions. Each sub-question and answer pair, $\langle q, a \rangle$, represents an intermediate stage in a reasoning trajectory $\tau = \{\langle q_1, a_1 \rangle, \langle q_2, a_2 \rangle, \ldots, \langle q_n, a_n \rangle\}$, leading to the final answer $\mathcal{A}$. This decomposition transforms reasoning into a planning problem, where the goal is to find the optimal trajectory: $\tau^* = \{\langle q_1^*, a_1^* \rangle, \langle q_2^*, a_2^* \rangle, \ldots, \langle q_n^*, a_n^* \rangle\}$. We employ MCTS with balanced exploration and exploitation to construct trajectories (see Figure 2(c) and Algorithm 1) involving four phases:

Selection. Starting from the root node (representing $\mathcal{Q}$ and $\mathcal{E}$), the algorithm recursively selects child nodes based on the Upper Confidence Bound for Trees (UCT):

$$q_k^* = \arg\max_{q_k \in \text{sub}(q_{k-1})}$$

$$\left[Q_{\text{value}}(s_{k-1}, q) + w\sqrt{\frac{\ln N(s_{k-1})}{N(c(s_{k-1}, q))}} \right], \quad (2)$$

where $s_{k-1} = \langle q_{k-1}, a_{k-1} \rangle$, $N(s_{k-1})$ is the visit count for node s_{k-1}, and $c(s_{k-1}, q_k)$ is the corresponding child node. The weight w tunes the balance between exploration (less visited nodes) and exploitation (high Q_{value} nodes).

Expansion. Upon reaching a leaf node, we generate d candidate sub-questions using $\mathcal{G}$: $\text{sub}(q_k) \sim \mathcal{G}(q \mid s_{k-1}, \ldots, s_0, \mathcal{E})$. If the leaf is terminal (i.e., the end of a reasoning chain), expansion is skipped.
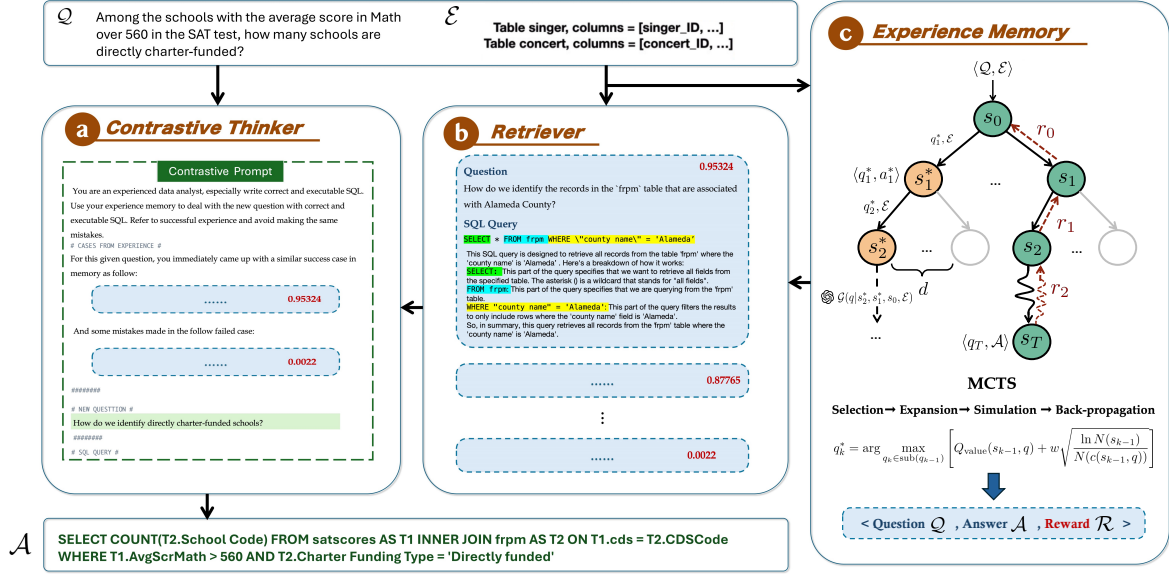
Figure 2: Overall framework of CoRE.

Simulation To estimate the expected future rewards (Q_{value}), this phase simulates future scenarios of the current node using the world model. Starting from the current node, at each state s, a sub-question is generated according to a *roll-out policy*, and the world model predicts the next state. This roll-out continues until a terminal state is reached. In CoRE, to simplify the process and reduce noise, we adopt a similar approach as in the expansion phase, generating d candidate sub-questions and selecting the one with the highest local reward, $q_k^* = \max_{q_k} f_r(s_{k-1}, q_k)$. For practical efficiency, we use a simplified reward function f_r to select sub-questions during simulation: $f_r(s_{k-1}, q_k) = r_1^\alpha \cdot r_2^{1-\alpha}$, where r_1 is the consistency reward for q_k, calculated by counting the frequency of the d sub-questions $\text{sub}(q_{k-1})$, and r_2 is the self-evaluation reward from the generator, defined as $r_2 = \mathcal{G}(\text{"Yes"} \mid q_k, \text{prompt})$, with prompt = "Is this reasoning step correct?".

Back-propagation Upon reaching a terminal state s_T (i.e., determining the final answer) in the previous phases, we obtain a reasoning path from the root node to the terminal node. At this point, we back-propagate the rewards along the path to update the Q_{value} for each question-answer pair. Specifically, $Q_{\text{value}}(s_{k-1}, q_k)$ is updated by aggregating the rewards from all future steps of node s_k:

$$Q_{\text{value}}(s_{k-1}, q_k) = \max_{s_{k-1}, q_k, r_k, \ldots, s_l, q_l, r_l, s_{l+1}, \ldots} \text{avg}(r_{k-1}, \ldots, r_l, \ldots). \quad (3)$$

After completing the predetermined MCTS iter-ations, the algorithm terminates and extracts the final reasoning trace by starting at the root and iteratively selecting the action with the highest Q_{value} until a terminal node is reached.

3.2 Retriever

The Retriever in the CoRE framework identifies both positive and negative cases for the new question $\mathcal{Q}_{\text{current}}$. As showed in Figure 2 (b), it searches the Experience Memory database $\mathcal{D}_{\text{experience}}$ by measuring semantic similarity $\text{Sim}(\mathcal{Q}_{\text{current}}, \mathcal{Q}_{e_i})$ between $\mathcal{Q}_{\text{current}}$ and stored experience entries $\{e_1, e_2, \ldots, e_k\}$. The top k entries are retrieved and ranked by similarity, denoted as $\text{rank\_sim}(e_i)$. These entries, which include structured knowledge and natural language descriptions, enhance retrieval accuracy. For example, SQL queries are accompanied by natural language descriptions of their Abstract Syntax Trees (AST) (Example in the Figure 2), improving the semantic understanding of retrieval and aligning the entries more closely with the query semantics. Next, the top k entries are re-ranked based on their reward labels $r(e_i)$, producing $\text{rank\_reward}(e_i)$. The final ranking combines both similarity and reward ranks: $\text{rank\_final}^{\text{positive}}(e_i)$ merges $\text{rank\_sim}(e_i)$ and $\text{rank\_reward}(e_i)$, while $\text{rank\_final}^{\text{negative}}(e_i)$ merges $\text{rank\_sim}(e_i)$ with the reverse of $\text{rank\_reward}(e_i)$. The final ranking is obtained by averaging these components. Finally, based on the n-shot setting, the top n positive and negative entries are selected for the next stage.
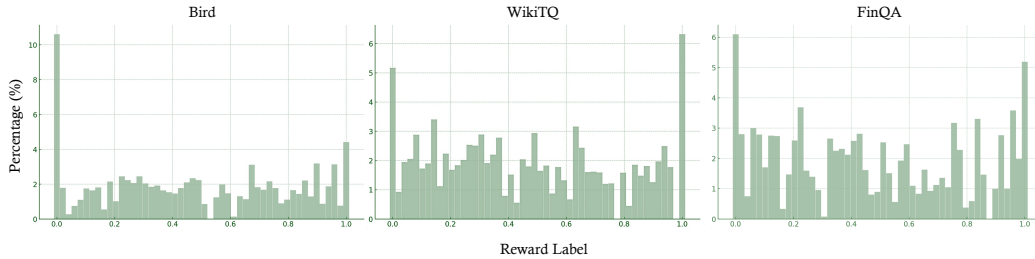
Figure 3: Reward label distribution on *Bird*, *WikiTQ* and *FinQA*

3.3 Contrastive Thinker

The Contrastive Thinker is built upon a general LLM that is specifically prompted to generate the necessary text based on the given contrastive context. Upon receiving positive and negative cases from the Retriever, the Contrastive Thinker employs deliberate practice by treating positive cases as past successes and negative cases as past failures. Through In-Context Learning, it can monitor its own performance, identify mistakes, and make adjustments accordingly. This process is encapsulated in the directive: "Solve the «<NEW CASE »>using the successful experience and avoid repeating the errors from failed experience...", as illustrated in Figure 2 (a). The examples of prompts constructed through the collaboration of Retriever and Contrastive Thinker can be found in the Appendix C.

4 Experiment

In this section, we first outline the experiment setup in § 4.1 and describe the Experience Memory dataset we constructed in § 4.2. Then, we present experiments and results for two structured knowledge reasoning tasks in §4.3, including TableQA and Text-to-SQL. Finally, the ablation study (§ 4.4) and strategy analysis (§ **??**) are presented.

4.1 Experiment setup

We use LlaMA-3 series (Dubey et al., 2024) and OpenAI's GPT-4 (Achiam et al., 2023) as feneral LLMs in different experiment settings. The embedding models we used for retrieval is `bge-large-en-v1.5`. More experiment details in Appendix B.

Datasets. We evaluate our CoRE on the dev sets on the training sets of different structured knowledge reasoning datasets showed in Table 5, including WikiTQ (Pasupat and Liang, 2015a), FinQA (Chen et al., 2021a) and Bird (Li et al., 2024). The details of these two datasets are provided in Appendix B.3.

Evaluation. For Text-to-SQL, we use Execution Accuracy (EX), which measures the correctness of the SQL execution, to evaluate our method on Bird (Li et al., 2024). For TableQA, accuracy is used to verify whether predicted answers match the gold ones on WikiTQ (Pasupat and Liang, 2015a) and FinQA (Chen et al., 2021a).

Baselines. CoRE is a plug-and-play framework, so we selected prompt-based agent reasoning frameworks as baselines (Pourreza and Rafiei, 2024; Gao et al., 2023a; Wang et al., 2023; Jiang et al., 2023; Ye et al., 2023). We integrate CoRE into their components or replace their few-shot approaches. Note that we have not altered the grounding method but only integrated CoRE into the reasoning component. A detailed introduction to various methods is provided in Appendix B.4, and differences between their approaches and ours are discussed below.

4.2 Experience Memory Dataset

The method we used to build the Experience Memory Dataset is detailed in § 3.1 and Algorithm 1. For cost considerations, we used LlaMA-3-70b as backbone to produce experience memory data. During the MCTS process to imitate human thinking, we set the capacity of candidate sub-questions $K = 4$, roll-out times in simulation $N = 10$, depth limit $L = 5$, the balance weight $w = 0.5$, and reward weight $\alpha = 0.5$. To encourage more diverse reasoning paths, we set the sampling temperature to 0.8. On average, each structure knowledge reasoning question can be decomposed into 3-5 $\langle q, a \rangle$ pairs in each iteration. To obtain more reliable reward labels for each $\langle q, a \rangle$, in the search tree we built after 10 rolls-outs, the Q_{value} of the nodes with more than or equal to 3 visits is stored as the reward label of $\langle q, a, r \rangle$. Moreover, we add overall successful cases with reward labels "1.00" and failed cases with "0.00" to the dataset as experience.

Finally, we construct the Experience Memory Dataset on the training set (Table 5), expanding the

original data by 8-9×. The reward label distribution is shown in Figure 3, demonstrating **increased diversity and domain coverage** with diluted and decomposed high-density knowledge (Figure 3). The $\langle q, a, r \rangle$ with $r \in (0, 1)$ is not included in the original training dataset, and they have difficulty difference after decomposition. In cases of errors, data entries with lower reward values usually come from nodes close to the error endpoint. On the other hand, data entries with higher rewards are more likely to lead to the correct endpoint after undergoing multiple verifications.

4.3 Result and Analysis

Table 1: Results by LlaMA3-70b or GPT-4, evaluated by Execution Accuracy (EX) on Bird Dev.

Method	EX (%)
DIN-SQL (Pourreza and Rafiei, 2024)	30.5
+ our CoRE	34.0 (↑ 3.5)
DAIL-SQL (Gao et al., 2023a)	31.6
+ our CoRE	35.2 (↑ 3.6)
DAIL-SQL (SC) (Gao et al., 2023a)	37.5
+ our CoRE	36.7 (↓ 0.8)
MAC-SQL (Wang et al., 2023)	34.9
+ our CoRE	40.8 (↑ 5.9)
MAC-SQL (GPT-4) (Wang et al., 2023)	46.6
+ our CoRE	51.6 (↑ 5.0)
Δ CoRE	3.44

Text-to-SQL. Table 1 shows overall performance of all methods. To ensure consistency with CoRE, baselines were run in a 2-shot setting. On average, CoRE improves performance by 3.44% across all models. Specifically, traditional methods like DIN-SQL and DAIL-SQL gain 3.5% and 3.6% in accuracy, respectively. Note that DAIL-SQL uses few-shot examples extracted from the training set without retrieval, while MAC-SQL and DIN-SQL employ fixed examples in the prompt template without additional positive or negative signals to indicate relevance. Moreover, CoRE provides substantial improvements for advanced models. It enhances MAC-SQL and its GPT-4 version by 5.9% and 5.0%, respectively, as the more powerful language model effectively captures contrastive contexts.

The example selection in DAIL-SQL masks domain-specific words in both target and candidate questions, ranking candidates by cosine similarity of their embeddings to pick the most relevant examples for ICL. In contrast, **our CoRE retrieves a wider range of data, providing diversity and broader coverage** (Figure 3) — *answer to RQ-(1)*.

However, it is noteworthy that CoRE slightly reduces the accuracy for DAIL-SQL (SC) by 0.2%. This result may be due to the Contrastive Thinker in CoRE, which includes a large amount of contextual learning information. In contrast, Self-Consistency (SC) uses voting on five answers generated with a temperature of 1.0. This high-temperature setting can amplify the impact of noise from excessive contextual information in the generated content. As a result, it leads to negative outcomes. This also suggests that **combining different prompting methods is not always better**, as CoRE can be viewed as a dynamic few-shot approach for prompt engineering. Moreover, CoRE shows significant improvement for challenging Text-to-SQL tasks as shown in Figure 4. It nearly doubles the execution accuracy, reaching 40%, compared to MAC-SQL. This also indicates that **CoRE is useful for challenging knowledge-intensive reasoning questions.**
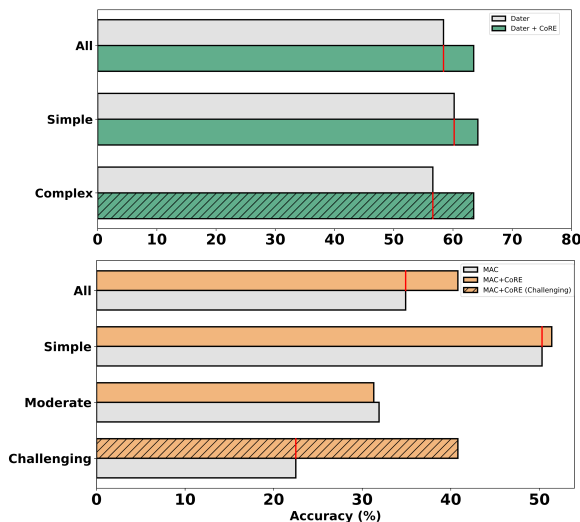

Figure 4: Performance under different difficulty.

Table 2: Results by GPT-3.5, evaluated by Accuracy on WikiTQ and FinQA Dev.

Dataset	Method	Accuracy (%)
WikiTQ	StructGPT (Jiang et al., 2023)	64.4
	+ our CoRE	66.1 (↑ 1.7)
	Dater (Ye et al., 2023)	58.4
	+ our CoRE	63.5 (↑ 5.1)
Δ CoRE		3.4
FinQA	StructGPT (Jiang et al., 2023)	51.2
	+ our CoRE	53.1 (↑ 1.9)
	Dater (Ye et al., 2023)	52.4
	+ our CoRE	59.0 (↑ 6.6)
Δ CoRE		4.25

Table 3: Ablation Experiment by GPT-4 (2-shot results.)

Method	2-shot				
	😐😐	😊😊	😡😡	😊😡	😊|😡
Generic Reasoning					
End-to-End QA + our CoRE	67.68	67.89 (↑ 0.21)	67.33 (↓ 0.35)	68.20 (↑ 0.52)	68.14 (↑ 0.46)
CoT QA (Wei et al., 2022) + our CoRE	68.09	68.07 (↓ 0.02)	67.73 (↓ 0.36)	68.35 (↑ 0.26)	68.61 (↑ 0.52)
Δ CoRE	–	–	+0.39	+0.49	
Structure Knowledge Reasoning					
MAC-SQL (Wang et al., 2023) + our CoRE	56.40	55.74 (↓ 0.66)	45.20 (↓ 11.20)	**58.92** (↑ 2.52)	58.08 (↑ 1.68)
Dater (Ye et al., 2023) + our CoRE	64.88	66.20 (↑ 1.32)	65.27 (↑ 0.39)	68.03 (↑ 3.15)	**69.72** (↑ 4.84)
Δ CoRE	–	–	+2.84	+3.26	

Table 4: Ablation Study with GPT-4 (0/1-shot Settings)

Method	0-shot	1-shot		
	😐	😐	😊	😡
End-to-End QA				
+ our CoRE	66.79	66.83	66.60 (↓0.23)	66.04 (↓0.79)
CoT QA				
+ our CoRE	67.68	67.85	68.71 (↑0.86)	66.69 (↓1.16)
MAC-SQL				
+ our CoRE	48.77	54.36	54.45 (↑0.09)	40.10 (↓14.26)
Dater				
+ our CoRE	60.19	64.90	66.38 (↑1.48)	64.82 (↓0.08)

TableQA Consistent with the Text-to-SQ, we employed the 2-shot setting. Table 2 shows that when applied to TableQA baselines, CoRE boosts Struct-GPT's accuracy by 1.7% and improves Dater's accuracy from 58.4% to 63.5% (↑ 5.1%). Both Struct-GPT and Dater use fixed few-shot examples in their *Iterative Reading-then-Reasoning* and *Jointly Reasoning* methods. It can be observed that using the CoRE plugin on Dater results in greater improvement. **This difference stems from the grounding phase in reasoning**. Dater leverages its Evidence Decomposer to break down and filter high-density structured knowledge, producing a cleaner reasoning input $\mathcal{E}$ with reduced noise. Consequently, with this refined $\mathcal{E}$, CoRE performs Contrastive In-Context Learning more effectively, resulting in a larger boost in reasoning performance. Overall, CoRE yields an average improvement of 3.4% on the WikiTQ dataset, and as shown in Figure 4, gains on complex issues dominate the overall enhancement.

On FinQA, CoRE increases StructGPT's accuracy by 1.9% and Dater's by 6.6%, achieving an average gain of 4.25%. As FinQA contains much less structured knowledge than WikiTQ, grounding and reasoning play a stronger role. Moreover, the higher plain-text content in FinQA simplifies ICL, allowing CoRE to enhance relevant information

more effectively and attain a higher success rate.

4.4 Ablation Study

We analyzed the differences between existing methods and our CoRE, then proposed the our research questions (§2). The *Answer to RQ-(1)*: our MCTS-generated Experience Memory Dataset offers greater diversity and coverage than datasets relying solely on training data by comparing with DAIL-SQL in § 4.3. To answer the remaining RQs, we further conducted ablation studies on MAC-SQL and Dater using GPT-4, demonstrating that CoRE benefits both structured and generic reasoning. To confirm its effectiveness, we compared the fixed few-shot method used in baselines (Wang et al., 2023; Ye et al., 2023) with our Retriever-provided positive and negative examples across 0-shot to 2-shot settings. As shown in Table 4 and Table 3, 😐 denotes fixed examples without any additional signals, while 😊 and 😡 represent positive and negative cases for the new question. These three types of examples are provided as N-shot inputs for reasoning in a single round. However, "😊 | 😡" uses a two-round approach, with positive cases guiding the first and negatives correcting errors in the second. The results lead to the following conclusions:

RAG helps effective knowledge augmentation. — *Answer to RQ-(2)*. For structure knowledge (high-density) reasoning, RAG proves to be an effective approach for diluting intensive knowledge with effective examples, to guide LLM to interpret complex knowledge. This is consistent with the effectiveness demonstrated by DAIL-SQL. Compared with the fixed example without specifically associated signals, CoRE can improve performance through dynamically recalled positive examples (without comparison), which can be seen in both 1-shot (↑ 1.48%) and 2-shot (↑ 1.32%). The

reasoning task in the more general FinQA leans towards a strong LLM background, showing similar performance without much corresponding increase. **Contrastive ICL brings consistent improvement** — *Answer to RQ-(3)*. Leveraging CoRE's dynamic retrieval to obtain relevant examples, contrastive ICL outperforms both positive-positive (😃 😃) and negative-negative (☹️ ☹️) combinations. In particular, positive-negative combinations (😃 ☹️ or 😃 | ☹️) yield the highest gains—up to ↑ 2.835% in a single round or ↑ 3.260% across two rounds. This consistent improvement in both generic and structured knowledge reasoning demonstrates that contrastive ICL effectively stabilizes and enhances performance on challenging tasks.

Only using negative examples without pointing out the mistakes would harm performance The data indicates that using positive examples before generation yields better outcomes, while negative examples can have adverse effects. In the **2-shot** columns of the table, positive-positive combinations (😃 😃) show superior performance compared to fix examples (😐 😐). Conversely, negative-negative combinations (☹️ ☹️) degrade model performance, showing that negative examples introduce noise and negatively affect results. It seems that only using negative examples without pointing out the mistakes would harm performance, as generative models will involuntarily imitate incorrect cases even if they have been informed that these are wrong cases.

Different ways of comparison in the dialogue yield similar effects. In the results of using Contrastive In-Context Learning, the effect of comparing in single-turn dialogues (😃 ☹️) and comparing before and after in two-turn dialogues (😃 | ☹️) is competitive. This means that our CoRE can be implemented in a more flexible way, and even with limited context length, similar improvements can still be achieved through multi-turn dialogues.

We experimented with various strategies in prompt engineering for our CoRE. In the Text-to-SQL task, we integrated CoRE into the Self-Consistency (SC) baseline method but observed negative effects (§ 4.3), leading us to discontinue discussing this strategy. In the Text-to-SQL task, we used LlaMA-3 to apply CoRE on MAC-SQL multiple times for iterative optimization in multi-turn dialogues, a process known as bootstrapping.
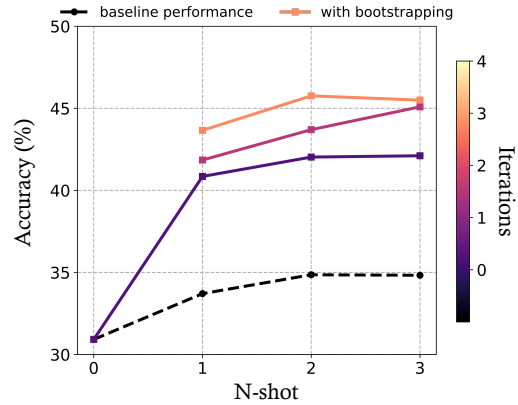

Figure 5: Bootstrapping with CoRE by LlaMA-3-70b.

Bootstrapping. We repeatedly apply CoRE in multi-turn dialogues under various few-shot settings—one iteration equals one use of CoRE and N-shot indicates the number of contrastive examples in context. Due to context length limits, structured knowledge $\mathcal{E}$ is included only in the first round, with subsequent iterations updating the current answer solely based on retrieved examples. As shown in Figure 5, while the baseline improves modestly from 30.92% (0-shot) to 34.83% (3-shot), bootstrapping achieves 42.11% at 3-shot without iterations, rising further to 45.09% at 3-shot after one iteration and 45.76% at 2-shot after two iterations. This shows that using bootstrapping can lead to more accurate results when there is a sufficient budget and low requirements for delay restrictions.

Grounding and Reasoning. In our work, we concentrate on the reasoning aspect of structured knowledge question-answering by following the grounding method of each baseline. We wonder how the grounding results affect our CoRE reasoning, so we also experimented with using a golden schema for CoRE with MAC-SQL. The results in Figure 6 show that the grounding process will significantly affect the results of CoRE reasoning. This observation is consistent with the phenomenon observed on two different baselines in TableQA. Using CoRE as a dynamic few-shot strategy on more fine-grained structured knowledge $\mathcal{E}$ can lead to higher returns.
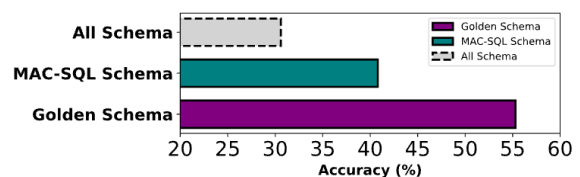

Figure 6: Different Schema for CoRE by LlaMA-3-70b

Computational Analysis CoRE's one-time offline cost is amortized across tasks, its inference-time overhead is minimal, and it remains practical under strict token constraints. We analyze three aspects here. ***1. Offline Investment:*** We run Monte Carlo Tree Search once per training example to build the Experience Memory of (question, context, answer) triplets. This step produces a reusable repository without further model training and resolves cold-start issues. After construction, no additional offline computation is required for any downstream task. ***2. Inference-Time Retrieval Overhead:*** At inference, each query is mapped to an embedding and compared against stored entries. Retrieving the top-k candidates requires computing similarity scores for all entries (complexity $O(N \times d)$) and re-ranking the best matches (complexity $O(N \log k)$). In practice, retrieval plus prompt construction takes under 60 ms per query, which is negligible compared to typical model generation time. ***3. Scalability under Token Limits (Few-Shot Adaptability):*** CoRE supports multi-turn in-context learning (like 😊 | 😠), allowing us to present contrastive examples across several shorter prompts rather than one long prompt. This design reduces the impact of strict token limits while preserving few-shot performance. Bootstrapping further refines examples over turns, so even a small set of positive and negative instances yields consistent improvements without excessively long context.

5 Related Work

5.1 Contrastive learning

Contrastive learning (Khosla et al., 2020) is a well-known technique that encourages models to differentiate between "positive" and "negative" samples, commonly used in areas like RLHF, where reward models are trained on human preference data (Lightman et al., 2023; Ouyang et al., 2022). However, contrastive methods are less explored in prompt engineering. It remains unclear whether contrastive demonstrations can enhance the reasoning process (Wang et al., 2022), though recent work shows that contrastive CoT consistently improves In-context learning (Chia et al., 2023).

5.2 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) enhances LLMs by fetching relevant documents from external databases and integrating them into the generation process (Gao et al., 2023b; Lewis et al., 2020b; Khandelwal et al., 2020; Izacard and Grave, 2021;

Luo et al., 2023) (see Figure 1(a)). Recent work has focused on helping LLMs decide when and what to retrieve (Schick et al., 2024) or on better exploiting context (Sarthi et al., 2024; Kim et al., 2024). While Self-Reflective RAG (Asai et al., 2023) uses reflection tokens to guide retrieval and Corrective RAG (Yan et al., 2024b) proposes a lightweight evaluator (albeit lacking high-level reasoning), CoRE instead emphasizes effective utilization of retrieved content and contexts by contrastive learning.

5.3 Structured Knowledge Reasoning

Structured knowledge, such as tables, knowledge graphs, and databases, has been central to knowledge grounding (Liu et al., 2025) and reasoning tasks. Due to heterogeneous data formats, methods (Xie et al., 2024) have emerged that leverage specific training setups to learn these representations. However, recent prompt-based methods, capitalizing on LLMs' capabilities, aim to process structured knowledge without re-training. Approaches like MAC-SQL (Wang et al., 2023), StructGPT (Jiang et al., 2023), Dater (Ye et al., 2023), DIN-SQL (Pourreza and Rafiei, 2024), and DAIL-SQL (Gao et al., 2023a) enhance LLMs' reasoning through multi-agent frameworks, decomposition, and schema linking, improving performance with techniques like evidence decomposition and self-correction.

6 Conclusion

We have introduced CoRE, a training-free contrastive RAG framework that uses Monte Carlo Tree Search to build an Experience Memory for structured knowledge reasoning. Our experiments on Text-to-SQL and TableQA tasks demonstrate that CoRE improves performance substantially on challenging queries by retrieving evidence with greater diversity and coverage than prior approaches. By incorporating both positive and negative examples through contrastive in-context learning, CoRE enhances reasoning robustness across different grounding methods and remains scalable under strict token limits through multi-turn prompting. The offline memory construction incurs a one-time cost that is amortized across tasks, and inference-time retrieval adds negligible overhead. These results establish CoRE as a flexible, efficient solution that extends the generalization of NLP models to complex structured reasoning scenarios.

Limitation

While CoRE provides a promising solution for the cold start problem in RAG tasks and offers a tailored approach for structured knowledge reasoning, its performance may vary depending on the coverage and quality of the Experience Memory dataset. The effectiveness of the method relies heavily on the breadth of the data used to generate this memory, and when transferring the approach to a new task, the Experience Memory must be rebuilt as per the outlined method for stable results. Additionally, CoRE's performance is highly dependent on the accuracy of the initial grounding phase. The grounding phase identifies relevant data from structured sources, and inaccuracies in this process may hinder the model's ability to reason effectively. This aspect requires careful attention, especially when applied to real-world scenarios where structured knowledge might be incomplete or ambiguous.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Ding Chen, Shichao Song, Qingchen Yu, Zhiyu Li, Wenjin Wang, Feiyu Xiong, and Bo Tang. 2024. Grimoire is all you need for enhancing large language models. *Preprint*, arXiv:2401.03385.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021a. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of EMNLP 2021*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021b. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of EMNLP 2021*.

Yew Ken Chia, Guizhen Chen, Luu Anh Tuan, Soujanya Poria, and Lidong Bing. 2023. Contrastive chain-of-thought prompting. *arXiv preprint arXiv:2311.09277*.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023.

Everything of thoughts: Defying the law of penrose triangle for thought generation. *arXiv preprint arXiv:2311.04254*.

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*.

Anders Ericsson and Robert Pool. 2016. *Peak: Secrets from the new science of expertise*. Random House.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023a. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024a. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Jiawei Gu, Zacc Yang, Chuanghao Ding, Rui Zhao, and Fei Tan. 2024b. Cmr scaling law: Predicting critical mixture ratios for continual pre-training of language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16143–16162.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*.

Gautier Izacard and Edouard Grave. 2021. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

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

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. In *International Conference on Machine Learning*, pages 15696–15707. PMLR.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through Memorization: Nearest Neighbor Language

Models. In *International Conference on Learning Representations (ICLR)*.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673.

Jaehyung Kim, Jaehyun Nam, Sangwoo Mo, Jongjin Park, Sang-Woo Lee, Minjoon Seo, Jung-Woo Ha, and Jinwoo Shin. 2024. Sure: Summarizing retrievals using answer candidates for open-domain qa of llms. *arXiv preprint arXiv:2404.13081*.

Aobo Kong, Shiwan Zhao, Hao Chen, Qicheng Li, Yong Qin, Ruiqi Sun, and Xin Zhou. 2023. Better zero-shot reasoning with role-play prompting. *Preprint*, arXiv:2308.07702.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020a. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Preprint*, arXiv:2305.03111.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, et al. 2023. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *arXiv preprint arXiv:2305.18703*.

Geling Liu, Yunzhi Tan, Ruichao Zhong, Yuanzhen Xie, Lingchen Zhao, Qian Wang, Bo Hu, and Zang Li. 2025. Solid-SQL: Enhanced schema-linking based in-context learning for robust text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9793–9803, Abu Dhabi, UAE. Association for Computational Linguistics.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12.

Hongyin Luo, Yung-Sung Chuang, Yuan Gong, Tianhua Zhang, Yoon Kim, Xixin Wu, Danny Fox, Helen Meng, and James Glass. 2023. Sail: Search-augmented instruction learning. *arXiv preprint arXiv:2305.15225*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

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

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

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Boshi Wang, Sewon Min, Xiang Deng, Jiaming Shen, You Wu, Luke Zettlemoyer, and Huan Sun. 2022. Towards understanding chain-of-thought prompting: An empirical study of what matters. *arXiv preprint arXiv:2212.10001*.

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

Yuanzhen Xie, Xinzhou Jin, Tao Xie, Matrixmxlin Matrixmxlin, Liang Chen, Chenyun Yu, Cheng Lei, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving LLM-based text-to-SQL through workflow paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10796–10816, Bangkok, Thailand. Association for Computational Linguistics.

Yuanzhen Xie, Tao Xie, Mingxiong Lin, WenTao Wei, Chenglin Li, Beibei Kong, Lei Chen, Chengxiang Zhuo, Bo Hu, and Zang Li. 2023. Olagpt: Empowering llms with human-like problem-solving abilities. *arXiv preprint arXiv:2305.16334*.

Zhengzhuo Xu, Bowen Qu, Yiyan Qi, Sinan Du, Chengjin Xu, Chun Yuan, and Jian Guo. 2024. Chartmoe: Mixture of expert connector for advanced chart understanding. *arXiv preprint arXiv:2409.03277*.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024a. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024b. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *arXiv preprint arXiv:2301.13808*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *Preprint*, arXiv:2308.10144.

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. *Preprint*, arXiv:2402.16671.

A More Related Work

A.1 Mental Representations & Memory Representations

This concept of *Mental Representations* (Ericsson and Pool, 2016) on human cognition refers to a mental structure that corresponds to an object, an idea, or a collection of information, whether concrete or abstract, that the brain is contemplating. *Mental Representations* plays a crucial role across various domains, facilitating effective planning and problem-solving, which can be figuratively stated as:

> *While the Mental Representations give masters a view of the forest when everyone else sees only trees, they also allow masters to zero in on the trees when necessary.*

> ——*Ericsson and Pool*

Moreover, from the perspective of natural language development, the meaning of language is not right over there to see, to grasp or to learn, but rather constructed during a communicative event through interactive processes among interlocutors. Hence, intuitively, developing structures similar to *Mental Representations* within LLMs' dialogue contexts naturally corresponds to the internal cognitive mechanisms of language, while also capturing the essential features of *Mental Representations*, including its dynamic formation and continual refinement.

Thus, intuitively, constructing structures akin to Mental Representations within the context of dialogue, significantly enhances the understanding and reasoning of natural language content. Consequently, we borrow the concept of *Mental Representations* and propose **Memory Representations** for LLMs, which enhances their abilities in comprehending, memorizing, organizing, analyzing, and ultimately making decisions with interactive information. Many paradigms in reasoning and planning incorporate components with functionalities similar to **Memory Representations**, albeit in naive forms, such as long-short-term memory (Shinn et al., 2023), task-specific heuristic function and policy networks (Ding et al., 2023). However, they are more than just storage memory in different forms or functions; indeed, they should encapsulate valid experiences that can guide efficient practices in subsequent instances. They optimizes decision-making policies through policy gradient methods relying on binary or scalar feedback, which significantly differs from the human-like decision-making process. However, quantifying the reward from natural language text is particularly challenging due to its inherent ambiguity and the subjective nature of language interpretation. Additionally, many existing methods (Shinn et al., 2023; Ding et al., 2023) equip LLMs with crucial human capabilities like memory, thereby regarding them as autonomous agents to complete human-like reasoning and planning. In this paper, we concentrate on the task of decision-making through natural language text, which encapsulates a comprehensive range of information, nuances, and potential actions.

A.2 Experiential Learning

To improve the performance of LLMs, researchers provide textual experience to LLMs through prompts. Early studies primarily involve manually crafting such experiential prompts (Wei et al., 2022; Kong et al., 2023), while more recent work focuses on utilizing the LLMs themselves to obtain task-solving experience automatically. Zhao et al. (2023) and Chen et al. (2024) leveraged LLMs to automatically summarize experience from manually annotated NLP datasets. Zhao et al. (2023) employed Reflexion (Shinn et al., 2023) to generate reasoning chains for each question. Then, the experience is summarized from the questions, chains, and human-annotated labels by LLMs. Chen et al. (2024) analyzed the impact of different examples and prompts on the quality of the summarized experience. However, these methods still require human labor to obtain experience and determine which experience to employ for the current question. Our framework autonomously learns and selects experience by RAG, which is continual and more consistency.

Algorithm 1 MCTS with LLMs

Require: Overall question $\mathcal{Q}$, structured evidence $\mathcal{E}$, large language model Generator, action generator $\mathcal{G}(\cdot)$, number of generated sub-questions d, depth limit L, number of roll-outs N, and exploration weight w, reward function to return specific reward $f_r(s_{k-1}, q_k)$

1: **for** $n \leftarrow 0, \ldots, N-1$ **do**
2:      $k \leftarrow 1$
3:      **while** $N(s_{k-1}) > 0$ **do**   ▷ Selection
4:          $N(s_{k-1}) \leftarrow N(s_{k-1}) + 1$
5:          $q_k \leftarrow \arg\max_{q \in \text{sub}(q_{k-1})} \left[Q_{\text{value}}(s_{k-1}, q) + w\sqrt{\frac{\ln N(s_{k-1})}{N(c(s_{k-1}, q))}} \right]$
6:          $r_k = f_r(s_{k-1}, q_k), s_k \leftarrow c(s_{k-1}, q_k)$
7:          $k \leftarrow k + 1$
8:      **end while**
9:      **while** s_k is not a terminal state $\wedge \, k \leq L$ **do**
10:         **for** $i \leftarrow 1, \ldots, d$ **do**   ▷ Expansion
11:            sample $\text{sub}(q_{k-1}) \leftarrow \text{sub}(q_{k-1}) \cup \{\mathcal{G}(q | s_{k-1}^*, \ldots, s_0, \mathcal{E})\}$,
12:   ▷ *Invoking LLM*
13:         **end for**
14:         $q_k^* \leftarrow \arg\max_{q \in \text{sub}(q_{k-1})} f_r(s_{k-1}, q_k)$ ▷ Simulation
15:         $r_k \leftarrow r(s_{k-1}, q_k), s_k \leftarrow c(s_{k-1}, q_k)$
16:         $k \leftarrow k + 1$
17:      **end while**
18:      **for** $k' \leftarrow (k-1), \ldots, 0$ **do**   ▷ Back propagation
19:         Update $Q_{\text{value}}(s_{(k-1)'}, a_{k'})$ with $\{r_{k'}, r_{k'+1}, \ldots, r_{k-1}\}$
20:      **end for**
21: **end for**

B   Experiment Details

B.1   Pseudo-code

B.2   Experience Memory Dataset Details

Finally, we build the Experience Memory Dataset on training part (Table 5) and finally obtain 85,956 cases with Bird (Li et al., 2024), 98,586 cases with WikiTQ (Pasupat and Liang, 2015a) and 55,399 cases with FinQA (Chen et al., 2021a), which has expanded the original training dataset by 8-9 $\times$.

As we stated in § 3.1, structured knowledge reasoning can be solved by planning for decomposition. Figure 7 shows one trajectory for FinQA Dataset.

B.3   Datasets

Dataset	Overall Length		Train			Dev/Test		
	In. (avg)	Out. (avg)	Cnt	In. (max)	Out. (max)	Cnt	In. (max)	Out. (max)
WikiTQ	831.8	5.8	11321	2028	273	4344	2048	148
BIRD	439.8	63.3	9428	1992	347	1534	1214	386
FinQA	1230.3	21.0	6251	2040	72	1147	2048	61

Table 5: Token sequence length statistics for each dataset.

TableQA   We evaluate our proposed CoRE on two table-based reasoning datasets, including WikiTable-Question (Pasupat and Liang, 2015a), and FinQA (Chen et al., 2021a). The details of these two datasets are provided as follows.

- **WikiTableQuestion** contains complex questions annotated by crowd workers based on Wikipedia tables. The crowd workers are asked to write questions that involve several complex operations such as comparison, aggregation and arithmetic operations, which require compositional reasoning over a series of entries in the given table. We use the standard test set with 4,344 samples.

Financial Example

Question 5: Please answer the given financial question based on the context.\nContext: entergy corporation and subsidiaries management 2019s financial discussion and analysis a result of the entergy louisiana and entergy gulf states louisiana business combination , results of operations for 2015 also include two items that occurred in october 2015 : 1) a deferred tax asset and resulting net increase in tax basis of approximately $ 334 million and 2) a regulatory liability of $ 107 million ($ 66 million net-of-tax) as a result of customer credits to be realized by electric customers of entergy louisiana , consistent with the terms of the stipulated settlement in the business combination proceeding . see note 2 to the financial statements for further discussion of the business combination and customer credits . results of operations for 2015 also include the sale in december 2015 of the 583 mw rhode island state energy center for a realized gain of $ 154 million ($ 100 million net-of-tax) on the sale and the $ 77 million ($ 47 million net-of-tax) write-off and regulatory charges to recognize that a portion of the assets associated with the waterford 3 replacement steam generator project is no longer probable of recovery . see note 14 to the financial statements for further discussion of the rhode island state energy center sale . see note 2 to the financial statements for further discussion of the waterford 3 write-off . results of operations for 2014 include $ 154 million ($ 100 million net-of-tax) of charges related to vermont yankee primarily resulting from the effects of an updated decommissioning cost study completed in the third quarter 2014 along with reassessment of the assumptions regarding the timing of decommissioning cash flows and severance and employee retention costs . see note 14 to the financial statements for further discussion of the charges . results of operations for 2014 also include the $ 56.2 million ($ 36.7 million net-of-tax) write-off in 2014 of entergy mississippi 2019s regulatory asset associated with new nuclear generation development costs as a result of a joint stipulation entered into with the mississippi public utilities staff , subsequently approved by the mpsc , in which entergy mississippi agreed not to pursue recovery of the costs deferred by an mpsc order in the new nuclear generation docket . see note 2 to the financial statements for further discussion of the new nuclear generation development costs and the joint stipulation . net revenue utility following is an analysis of the change in net revenue comparing 2015 to 2014 . amount (in millions) .\n|amount (in millions)\n|2014 net revenue|$ 5735|\nretail electric price|187|\n|volume/weather|95|\n|waterford 3 replacement steam generator provision|-32 (32)|\n|miso deferral|-35 (35)|\nlouisiana business combination customer credits|-107 (107)|\nother|-14 (14)|\n|2015 net revenue|$ 5829|\nthe retail electric price variance is primarily due to : 2022 formula rate plan increases at entergy louisiana , as approved by the lpsc , effective december 2014 and january 2015 ; 2022 an increase in energy efficiency rider revenue primarily due to increases in the energy efficiency rider at entergy arkansas , as approved by the apsc , effective july 2015 and july 2014 , and new energy efficiency riders at entergy louisiana and entergy mississippi that began in the fourth quarter 2014 ; and 2022 an annual net rate increase at entergy mississippi of $ 16 million , effective february 2015 , as a result of the mpsc order in the june 2014 rate case . see note 2 to the financial statements for a discussion of rate and regulatory proceedings. .\nQuestion: what is the net change in net revenue during 2015 for entergy corporation?\nAnswer:\n

Question 5.1: What is the net revenue in 2014?

Answer 5.1: The net revenue in 2014 is $5735 million.The answer is $5735.

Question 5.2: What is the net revenue in 2015?

Answer 5.2: The net revenue in 2015 is $5829 million.

Question 5.3: Now we can answer the question: What is the net change in net revenue during 2015 for entergy corporation?

Answer 5.3: The net change in net revenue during 2015 for entergy corporation is $5829 - $5735 = $94 million.

The answer is $94.

Figure 7: A trajectory of Experience Memory Builder for FinQA.

- **FinQA** consists of complex questions curated by financial experts based on real-world financial reports from S&P 500 companies. These questions involve intricate numerical reasoning, requiring multi-step operations such as addition, subtraction, and ratio calculations across both structured tables and unstructured text. The dataset is split into a training set with 6,251 samples, a validation set with 883 samples, and a test set with 1,147 samples.

Text-to-SQL

- **Bird** dataset is extensive, with 12,751 unique question-SQL pairs from 95 large-scale databases totaling 33.4 GB. It encompasses a wide array of over 37 professional domains including blockchain, hockey, healthcare, and education. To assist models in generating accurate SQL queries, Bird incorporates external knowledge from four sources: numerical reasoning, domain-specific information, synonyms, and value examples.

B.4 Baselines

To ensure a stable generation, we set the temperature to 0 when reasoning with CoRE. The baselines we used are as follow:

- **MAC-SQL** MAC-SQL (Wang et al., 2023) introduces a novel multi-agent framework for the Text-to-SQL task, leveraging large language models (LLMs). The framework comprises three specialized agents: the Selector, responsible for condensing the database by retaining only relevant table schemas; the Decomposer, which breaks down complex questions into simpler sub-problems; and the Refiner, which validates and refines the generated SQL queries. The Decomposer first determines whether the question can be addressed using a simple SQL query. When the question requires a more complex approach, it applies a chain-of-thought (CoT) strategy to generate sub-questions and corresponding SQL queries, iterating until the final SQL query is constructed. Throughout this process—whether generating SQL directly or decomposing the question—few-shot learning techniques are used to ensure precise results.

23905

- **StructGPT** StructGPT (Jiang et al., 2023) employs an iterative reading-then-reasoning (IRR) framework to enhance the capability of large language models (LLMs) in reasoning over structured data, particularly tables. The method involves constructing specialized interfaces to access and filter relevant evidence from the table data efficiently. Through an iterative process of invoking these interfaces to extract information, linearizing the extracted data into a readable format for LLMs, and generating responses based on this linearized data, the framework progressively homes in on the target answers to given queries. This approach allows LLMs to leverage their reasoning abilities effectively by focusing on the reasoning task rather than the intricacies of understanding structured data formats.

- **Dater** Dater (Ye et al., 2023) leverages large language models (LLMs) to decompose table-based reasoning tasks into more manageable sub-tasks. DATER addresses two main challenges in table-based reasoning: dealing with large evidence, such as extensive tables with numerous rows and columns, and reasoning over complex questions that require multi-step logical deductions. The method involves an Evidence Decomposer to break down the vast table into relevant sub-evidence by predicting the indexes of rows and columns, and a Question Decomposer that simplifies complex questions into a series of simpler sub-questions using a "parsing-execution-filling" strategy with intermediate SQL queries. These decomposed elements are then used by the LLM to reason and derive the final answer, enhancing the performance and interpretability of table-based reasoning.

- **DIN-SQL** DIN-SQL (Pourreza and Rafiei, 2024) enhances reasoning capabilities by decomposing the text-to-SQL task into smaller sub-tasks, which are then solved sequentially and their solutions fed into the LLMs. This includes modules for schema linking, query classification and decomposition, SQL generation, and self-correction. DIN-SQL employs prompting strategies tailored to the complexity of the task, addressing challenges in schema linking and utilizing LLMs for self-correction of minor mistakes in generated SQL queries. Through extensive experiments on the Spider and BIRD datasets, DIN-SQL demonstrates significant performance improvements, pushing the accuracy of LLMs towards or even surpassing the state-of-the-art fine-tuned models.

- **DAIL-SQL** DAIL-SQL (Gao et al., 2023a) integrates structured knowledge as SQL statements, selects examples based on skeleton similarities, and removes cross-domain knowledge for token efficiency. DAIL also explores the potential of open-source LLMs through supervised fine-tuning, demonstrating significant performance improvements. In the context of Few-shot learning, the paper introduces DAIL Selection, a strategy that considers both question and query similarities for selecting the most helpful examples, and DAIL Organization, a method that presents both questions and corresponding SQL queries to retain the question-SQL mapping while improving token efficiency. These approaches collectively contribute to the state-of-the-art performance of DAIL-SQL in Text-to-SQL reasoning, especially in scenarios with limited examples.

C Prompt Details

We provide the prompt details in MAC-SQL + CoRE as an example.

[Instruction]

When executing SQL below, some errors occurred. Please fix SQL based on query and database info. Solve the task step by step if necessary. Use SQL format in the code block, and indicate script type in the code block. When you find an answer, verify it carefully. Include verifiable evidence in your response if possible.

[Constraints]

- In 'SELECT <column>', just select needed columns in the **[Question]** without any unnecessary column or value.

- In 'FROM <table>' or 'JOIN <table>', do not include unnecessary tables.

- If using max or min functions, 'JOIN <table>' FIRST, THEN use 'SELECT MAX(<column>)' or 'SELECT MIN(<column>)'.

- If [Value examples] of <column> has 'None' or None, use 'JOIN <table>' or 'WHERE <column> IS NOT NULL'.

- If using 'ORDER BY <column> ASC|DESC', add 'GROUP BY <column>' before selecting distinct values.

[Query]

{query}

[Evidence]

{evidence}

[Database info]

{desc_str}

[Foreign keys]

{fk_str}

[Old SQL]

\{sql\}

[SQLite Error]

{sqlite_error}

[Exception Class]

{exception_class}

Now please fix the old SQL and generate new SQL again.

[Correct SQL]

C.1 Retriever Result Examples

- **Score**: 0.6489822864532471

- **Reward**: 0.99

- **SQL Query**:

```
SELECT count(*)
FROM customer_address
INNER JOIN address_status
ON customer_address.status_id = address_status.status_id
WHERE address_status.address_status = 'Inactive'
```

- **Natural Language Explanation**: This SQL query is designed to count the number of customer addresses that have an 'Inactive' status. Here's a breakdown of how it works:

- **SELECT count(*)**: This part of the query specifies that we want to retrieve the total count of rows that meet the specified conditions. The asterisk (*) is a wildcard that stands for "all rows."
- **FROM customer_address**: This specifies that we are querying from the 'customer_address' table.
- **INNER JOIN address_status ON customer_address.status_id = address_status.status_id**: This performs an inner join between the 'customer_address' table and the 'address_status' table based on the 'status_id' field. For each row in the 'customer_address' table, it will find a corresponding row in the 'address_status' table where the 'status_id' values match.
- **WHERE address_status.address_status = 'Inactive'**: This filters the results to include only rows where the 'address_status' field in the 'address_status' table is 'Inactive'.

In summary, this query counts the number of customer addresses that are marked as 'Inactive' in the address status.

- **Score**: 0.6261888146400452

- **Reward**: 0.0

- **SQL Query**:
```
SELECT
    (COUNT(CASE WHEN e.product = 'Credit card' THEN 1 END) / COUNT(*)) * 1.0 AS percentage
FROM
    client c
LEFT JOIN
    events e ON c.client_id = e.client_id
WHERE
    c.sex = 'Male';
```

- **Natural Language Explanation**: This SQL query is designed to calculate the percentage of events related to 'Credit card' for male clients. Here's a breakdown of how it works:

  - **SELECT (COUNT(CASE WHEN e.product = 'Credit card' THEN 1 END) / COUNT(*)) * 1.0 AS percentage**: This part of the query calculates the percentage of events where the product is 'Credit card'. The CASE statement counts the number of rows where the product is 'Credit card', and this count is divided by the total number of rows (COUNT(*)) to get the proportion. Multiplying by 1.0 ensures that the result is a decimal number. The result is given the alias 'percentage'.
  - **FROM client c**: This specifies that we are querying from the client table (which has been aliased as c).
  - **LEFT JOIN events e ON c.client_id = e.client_id**: This joins the client table with the events table based on the client_id field. It ensures that all rows from the client table are included in the result, even if there is no matching row in the events table.
  - **WHERE c.sex = 'Male'**: This filters the results to only include rows where the sex field in the client table is 'Male'.

In summary, this query calculates the percentage of events related to 'Credit card' for male clients.

C.2 Contrastive Prompt

Given a [Database schema] description, a knowledge [Evidence] and the [Question], you need to use your ««<EXPERI-ENCE MEMORY»»> to deal with the ««<NEW CASE»»> with correct and executable SQL. Please refer to <Successful Experience> and avoid making the same mistakes as the <Failed Experience>.
When generating SQL, we should always consider constraints: **[Constraints]**

- In 'SELECT <column>', just select needed columns in the [Question] without any unnecessary column or value.

- In 'FROM <table>' or 'JOIN <table>', do not include unnecessary table.

- If using max or min functions, 'JOIN <table>' FIRST, THEN use 'SELECT MAX(<column>)' or 'SELECT MIN(<column>)'.

- If [Value examples] of <column> has 'None' or None, use 'JOIN <table>' or 'WHERE <column> IS NOT NULL' is better.

- If using 'ORDER BY <column> ASC|DESC', add 'GROUP BY <column>' before to select distinct values.

««<EXPERIENCE MEMORY»»>
<Successful Experience> For this given question, you immediately came up with a similar success case in memory as follows:
[Database schema]
```
CREATE TABLE address_status (
    "address_status" TEXT COMMENT address_status; VALUES: [Active,Inactive],
    "status_id" INTEGER COMMENT status_id; VALUES: [1,2]
    PRIMARY KEY ("status_id")
);

CREATE TABLE customer_address (
    "status_id" INTEGER COMMENT status_id; VALUES: [1,2],
    "address_id" INTEGER COMMENT address_id; VALUES: [606,266],
    "customer_id" INTEGER COMMENT customer_id; VALUES: [1,2]
    PRIMARY KEY ("address_id, customer_id")
);

CREATE TABLE customer (
    "customer_id" INTEGER COMMENT customer_id; VALUES: [1,2],
    "email" TEXT COMMENT email; VALUES: [upurdy0@cdbaby.com,rvatini1@fema.gov],
    "first_name" TEXT COMMENT first_name; VALUES: [Ursola,Ruthanne],
    "last_name" TEXT COMMENT last_name; VALUES: [Purdy,Vatini]
    PRIMARY KEY ("customer_id")
);

CREATE TABLE order_status (
    "status_id" INTEGER COMMENT status_id; VALUES: [1,2],
    "status_value" TEXT COMMENT status_value; VALUES: [Order Received,Pending Delivery]
    PRIMARY KEY ("status_id")
);
```
[Question] How many of the customer addresses are inactive?
[Successful SQL]
```
SELECT count(*)
FROM customer_address
INNER JOIN address_status
ON customer_address.status_id = address_status.status_id
WHERE address_status.address_status = 'Inactive';
```
<Failed Experience> Here is a case where the query wasn't quite correct: **[Failed SQL]**
```
SELECT
    (COUNT(CASE WHEN e.product = 'Credit card' THEN 1 END) / COUNT(*)) * 1.0 AS percentage
FROM
    client c
LEFT JOIN
    events e ON c.client_id = e.client_id
WHERE
    c.sex = 'Male';
```

==========
««<NEW CASE»»>
[Database schema]

```
CREATE TABLE cards (
    id INTEGER,
    power TEXT,
    uuid TEXT,
    artist TEXT,
    asciiName TEXT,
    availability TEXT
);

CREATE TABLE foreign_data (
    id INTEGER,
    language TEXT,
    uuid TEXT,
    flavorText TEXT,
    multiverseid TEXT,
    name TEXT
);
```
[Question] What percentage of cards without power are in French?
[NEW SQL]
```
SELECT
    (COUNT(CASE WHEN f.language = 'French' THEN 1 END) / COUNT(*)) * 100 AS percentage
FROM
    cards c
INNER JOIN
    foreign_data f ON c.uuid = f.uuid
WHERE
    (c.power IS NULL OR c.power = '*');
```