# CONTRASTIVE SELF-REFINEMENT FOR LOW-COST ADAPTATION IN REAL-WORLD TEXT-TO-SQL

**Yuyang Wu**[1,2], **Xin Shi**[1,2], **Wei Long** [1], **Cam-Tu Nguyen**[1], **Xiaoliang Wang**[1],
[1]Nanjing University
[2]Guodian Nanjing Automation CO.,LTD.

## ABSTRACT

Deploying Text-to-SQL systems in real-world industries faces a "last-mile" bottleneck: the high maintenance cost of adapting to frequent schema changes and the lack of domain-specific knowledge in raw databases. Traditional fine-tuning methods struggle to keep pace with evolving business logic, while standard in-context learning lacks the robustness to bridge the semantic knowledge gap. To address these challenges, we propose SEEK-SQL, a training-free, self-evolutionary multi-agent framework designed for low-cost adaptation. Our framework introduces a novel Contrastive Self-Refinement (CRF) strategy that mimics human learning: by contrasting positive and negative execution trajectories, the system generates reusable guidelines to "learn" new schema rules on the fly without parameter updates. Additionally, we integrate a Sniffer agent to dynamically retrieve external knowledge, ensuring robustness against ambiguous queries. Experiments on Spider and BIRD show that SEEK-SQL achieves state-of-the-art performance among in-context learning methods while reducing token consumption by significantly compared to other multi-agent frameworks. These results demonstrate SEEK-SQL's potential as a cost-efficient, agile solution for industrial Text-to-SQL deployment.

## 1 INTRODUCTION

The translation of natural language into SQL (Text-to-SQL) holds immense promise for democratizing data access in enterprise environments Zhong et al. (2020); Wang et al. (2020). However, despite recent advances in Large Language Models (LLMs), a significant gap remains between academic benchmarks and reliable industrial deployment. Practitioners often face two prohibitive barriers that prevent current methods from functioning effectively in real-world scenarios.

The first and most critical hurdle is the high cost of adaptation during schema migration. Real-world databases are dynamic environments where business logic evolves and schemas change frequently. Previous approaches relying on fine-tuning Sun et al. (2024) or complex prompt engineering Pourreza & Rafiei (2023) suffer from a "cold start" problem when transferred to new domains. Retraining models for every schema update is computationally expensive and operationally impractical. Consequently, industry demands a system that is training-free and capable of rapidly adapting to unfamiliar environments through interaction, similar to how a human engineer learns a new codebase.

Compounding this challenge is the issue of robustness caused by the "Knowledge Gap." Industrial databases often lack self-contained semantic information, containing cryptic column names or implicit domain logic. As analyzed in our problem formulation, a substantial portion of generation errors stems not from incorrect SQL syntax, but from missing domain context—for instance, knowing that a specific name refers to an ID column rather than a text field. Standard agents often hallucinate in these scenarios without external grounding Wang et al. (2025), leading to unreliable execution results.

To overcome these barriers, we propose SEEK-SQL, a knowledge-enhanced, self-optimizing multi-agent framework. Unlike static pipelines, SEEK-SQL operates as a self-evolutionary system designed to minimize adaptation costs. It employs a Sniffer agent to actively retrieve external knowl-

edge to bridge the semantic gap, ensuring robustness against ambiguous queries. More critically, we introduce a Contrastive Self-Refinement (CRF) strategy to address the adaptation challenge. Instead of merely fixing a single SQL error, our system analyzes the contrast between failed and successful trajectories to synthesize generalizable guidelines. These guidelines are stored in a memory bank, allowing the system to avoid repeating the same logic errors—effectively "learning" the nuances of a new database without requiring parameter updates.

Our contributions are summarized as follows:

- We identify the dual challenges of Knowledge Gap and Adaptation Cost in industrial Text-to-SQL and propose SEEK-SQL, a training-free framework designed for these constraints.
- We introduce the Contrastive Self-Refinement (CRF) strategy. This mechanism transforms error correction into system evolution, significantly improving adaptability and reducing token costs by curbing repetitive failures.
- We achieve state-of-the-art results on the challenging BIRD Li et al. (2024) and Spider Yu et al. (2018) benchmarks among in-context learning methods, demonstrating superior cost-efficiency suitable for high-concurrency real-world scenarios.

## 2 Problem Formulation & Challenges

We formulate the robust Text-to-SQL task as a tuple $\mathcal{X} = (\mathcal{Q}, \mathcal{S}, \mathcal{D})$, where $\mathcal{Q}$ is the user query, $\mathcal{S} = \{\mathcal{T}, \mathcal{C}\}$ represents the database schema consisting of tables and columns, and $\mathcal{D}$ represents an external knowledge corpus such as data dictionaries or web content. The goal is to generate an executable SQL $\mathcal{Y}$ that accurately answers $\mathcal{Q}$ by leveraging both the schema structure and retrieved knowledge.

A major impediment to this goal is the prevalence of "Knowledge-gap Errors", which we provide detailed definition and example in Appendix A. Existing methods often operate under the assumption that the schema $\mathcal{S}$ contains all necessary semantics for reasoning. However, in practice, failures frequently occur due to the loss of essential domain knowledge not explicitly present in $\mathcal{S}$. As shown in Appendix A, our analysis of error cases from GPT-4o on Spider and BIRD benchmarks reveals that nearly 40% of failures fall into this category, details of this experiment can be found in Appendix A. This necessitates a shift from passive generation to a system capable of active retrieval, denoted as $\mathcal{K} \leftarrow \text{Retrieve}(\mathcal{Q}, \mathcal{D})$, to ground the generation process.

Furthermore, we define "Adaptation Cost" as the computational resources—specifically token consumption and time—and human effort required for a system to achieve peak accuracy on a new schema. While traditional fine-tuning incurs high human and time costs, multi-turn prompting often leads to excessive token usage. Our objective is to minimize this cost via efficient self-correction and memory reuse, enabling the system to reach high performance on new schemas with minimal overhead.

## 3 Methodology

### 3.1 Framework Overview

To address the dual challenges of knowledge gaps and adaptation costs, we propose SEEK-SQL. As shown in Figure 1, the framework operates on a collaborative architecture comprising four specialized agents: a Manager for task decomposition and planning, a Selector for schema reduction, a Generator for SQL synthesis, and a Sniffer for external knowledge retrieval. While the Manager, Selector, and Generator follow standard multi-agent patterns, our distinct contribution lies in the integration of the Sniffer and the Contrastive Self-Refinement (CRF) strategy. For more detailed information about our framework, please refer to Appendix C.

The Sniffer agent serves as the system's robustness base, designed to bridge the knowledge gap by acting as a dynamic information bridge. Unlike static prompt augmentation, it retrieves external context $\mathcal{K}$ based on sub-query requirements. To accommodate diverse industrial security constraints, the Sniffer supports three retrieval modes: Local mode utilizing RAG via LlamaIndex for private knowledge bases, Open World mode using search engines for public data, and a Hybrid mode. This

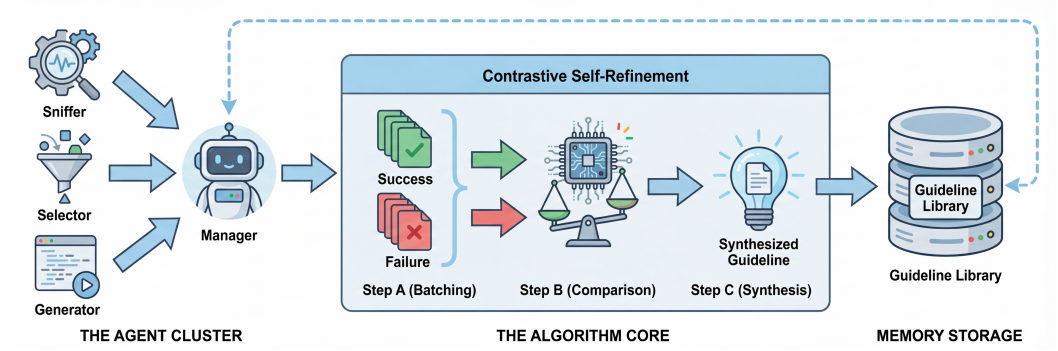



Figure 1: The overall structure of SEEK-SQL

flexibility ensures the system can resolve semantic ambiguities, such as specific domain abbreviations, that typically cause hallucinations in standard Text-to-SQL systems.

### 3.2 Contrastive Self-Refinement (CRF) Strategy

Standard error correction methods often fix bugs in isolation, leading to repetitive failures on similar logic. We propose a system-level evolution strategy, CRF, which transforms transient errors into persistent, retrievable guidelines. The process consists of three sequential phases: Backward Diagnosis, Contrastive Evolution, and Dynamic Guideline Retrieval. We provide a detailed pseudocode algorithm in Appendix B.

#### 3.2.1 Backward Diagnosis

We adopt a backward fixing mechanism that aligns with the logical dependency of SQL generation. The process begins with a syntax check performed locally by the Generator. Since syntax errors are typically low-level coding mistakes, resolving them locally reduces the overhead on the Manager. If the SQL syntax is correct but the execution result is erroneous (e.g., empty sets or incorrect answers), the process escalates to a semantic repair phase. Here, the Manager initiates a debugging process by tracing the generated SQL against the sub-query intention line-by-line. Upon detecting a mismatch—such as missing a filter condition or misinterpreting a column—the Manager dispatches the specific sub-task back to the Sniffer to check for missing knowledge or to the Selector to correct schema hallucinations.

#### 3.2.2 Contrastive Evolution

Once a difficult case is resolved, the system performs a post-mortem analysis to extract generalizable rules. We collect pairs of trajectories: the initial failed attempt (Negative Sample) and the final successful execution (Positive Sample). Following a batched training strategy, the Manager compares these positive and negative trajectories to identify the root cause of the divergence. For instance, the system might identify that table A should always be joined with table B using a specific foreign key. This logic is then distilled into a natural language guideline, transforming a specific error instance into a general rule.

#### 3.2.3 Context-Aware Adaptation

A key innovation of SEEK-SQL is the utilization of these synthesized guidelines through a Dynamic Guideline Retrieval mechanism. Simply concatenating all historical guidelines into the prompt would exceed context limits and introduce noise. Instead, we maintain a Guideline Library $\mathcal{L}_G = \{(g_i, q_i)\}_{i=1}^{N}$, where each guideline $g_i$ is indexed by the specific query $q_i$ that triggered its creation. When a new user query $q_{new}$ arrives, the system retrieves the top-$k$ most relevant guidelines based on vector similarity:

$$\mathcal{G}^* = \{g_i \mid \text{TopK}_{g_i \in \mathcal{L}_G}(\text{Sim}(\text{Emb}(q_{new}), \text{Emb}(q_i)))\} \tag{1}$$

This retrieval mechanism ensures that agents receive only the most contextually relevant advice. For example, logic derived from financial reporting queries is retrieved only when the system processes new financial queries. This effectively allows the system to "remember" how to handle specific scenarios without retraining.

Complementing the retrieved guidelines, we also employ a Memory Bank to serve as a rolling buffer of the system's latest experiences, following previous work Wu et al. (2024). By retaining the full trajectories of the most recent successful executions, it allows agents to 'warm up' via in-context learning, ensuring adherence to the correct output format. It is important to distinguish this from the Guideline Library: whereas the Memory Bank acts as a short-term working memory for format stabilization, the Guideline Library serves as a long-term semantic memory for logical correction.

Due to space constraints, we provide a detailed discussion on the long-term stability of this evolutionary mechanism—specifically addressing potential risks in Appendix D.

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL SETUP

**Benchmarks.** We evaluate SEEK-SQL on two widely adopted benchmarks: **Spider** Yu et al. (2018), representing diverse cross-domain schemas, and **BIRD** Li et al. (2024), a challenging large-scale dataset emphasizing external knowledge reasoning and efficiency. BIRD is particularly relevant to our industrial focus due to its massive database size (33.4GB) and noisy real-world content.

**Metrics.** Following standard protocols Zhong et al. (2020); Li et al. (2024), we report *Execution Accuracy (EX)* to gauge correctness and *Valid Efficiency Score (VES)* for SQL efficiency. *Exact Match (EM)* is also reported for Spider to facilitate comparison with prior works.

**Implementation.** We employ Deepseek-V3-671B Liu et al. (2024a) and GPT-4 OpenAI et al. (2024) as backbone LLMs. For fair comparison in token efficiency, we also benchmark using GPT-3.5-turbo. All experiments run on a single NVIDIA A100 GPU.

**Sniffer Configuration** To strictly evaluate the contribution of our framework versus external knowledge, we configure the system with two distinct operational settings. In **Close World Mode**, we intentionally disable all RAG and web search tools, forcing the agents to rely solely on the LLM's internal parametric knowledge. This setting serves as a fair baseline to compare with other baselines and verify self-refinement capabilities of the CRF mechanism. In contrast, the **Open World Mode** empowers the Sniffer agent with Google Search Tool to address knowledge-intensive queries.

**Baselines.** We compare against two categories of state-of-the-art methods: (1) *LLM-based Prompting methods* including DIN-SQL Pourreza & Rafiei (2023), DAIL-SQL Gao et al. (2023),ACT-SQL Zhang et al. (2023) and C3-SQL Dong et al. (2023); and (2) *Multi-Agent frameworks* including MAC-SQL Wang et al. (2025), SQLFixAgent Cen et al. (2025), and CHASE-SQL Pourreza et al. (2025).

## 4.2 OVERALL PERFORMANCE

We first evaluate SEEK-SQL against state-of-the-art baselines in the Close World setting to isolate the architectural contribution from external knowledge benefits. As shown in Tables 1 and 2, SEEK-SQL establishes a new performance standard even without web retrieval. On the challenging BIRD benchmark, our GPT-4o-powered Close World model achieves an execution accuracy (EX) of 74.49%, outperforming the strongest previous multi-agent frameworks, MAC-SQL (59.39%) and SQLFixAgent (58.67%), by a substantial margin of over 15%. It also marginally surpasses the recent CHASE-SQL (74.46%), which utilizes the larger context window of Gemini-1.5-Pro. Similarly, on Spider, SEEK-SQL maintains a lead with 87.1% EX compared to MAC-SQL's 86.8%. These results emphatically demonstrate that our performance gains are primarily driven by the Contrastive Self-Refinement (CRF) strategy rather than mere tool access. By synthesizing domain-specific guidelines from execution feedback, SEEK-SQL effectively "learns" to navigate complex schemas where

| Methods | Dev | | Test | |
|---|---|---|---|---|
| | EX% | EM% | EX% | EM% |
| *LLM-based Prompting* | | | | |
| DAIL-SQL (GPT-4) | 83.6 | 68.7 | 86.6 | 66.0 |
| C3-SQL (ChatGPT) | 81.8 | 71.4 | 82.3 | - |
| ACT-SQL (GPT-4) | 82.9 | 61.7 | - | - |
| DIN-SQL (GPT-4) | 85.3 | 60.0 | - | - |
| *Multi-Agent Frameworks* | | | | |
| MAC-SQL (GPT-4) | 86.8 | 23.5 | 82.8 | 19.3 |
| SQLFixAgent (GPT-3.5) | 84.8 | 77.9 | - | - |
| CHASE-SQL (Gemini-1.5 Pro) | 86.9 | - | 87.6 | 71.2 |
| *Ours (SEEK-SQL)* | | | | |
| Base (Close World) + GPT-4 | 87.1 | 77.9 | 87.7 | 71.6 |
| Base (Close World) + DeepSeek-V3 | 87.4 | 77.2 | 87.4 | 72.4 |
| **Full (Open World) + DeepSeek-V3** | **88.2** | **78.3** | **87.9** | **74.3** |

Table 1: Performance comparison on Spider Dev and Test sets. *Close World* Mode means Sniffer agent only uses its own knowledge inside LLM.

| Methods | EX% | VES% |
|---|---|---|
| *LLM-based Prompting* | | |
| GPT-4 (Zero-shot) | 46.35 | 49.77 |
| DIN-SQL (GPT-4) | 50.72 | 59.79 |
| DAIL-SQL (GPT-4) | 54.76 | 56.08 |
| *Multi-Agent Frameworks* | | |
| MAC-SQL (GPT-4) | 59.39 | 66.39 |
| SQLFixAgent (GPT-3.5) | 58.67 | 62.19 |
| CHASE-SQL (Gemini-1.5 Pro) | 74.46 | - |
| *Ours (SEEK-SQL)* | | |
| Base (Close World) + GPT-4 | 74.49 | 68.81 |
| Base (Close World) + DeepSeek-V3 | 72.15 | 68.85 |
| **Full (Open World) + DeepSeek-V3** | **74.53** | **69.72** |

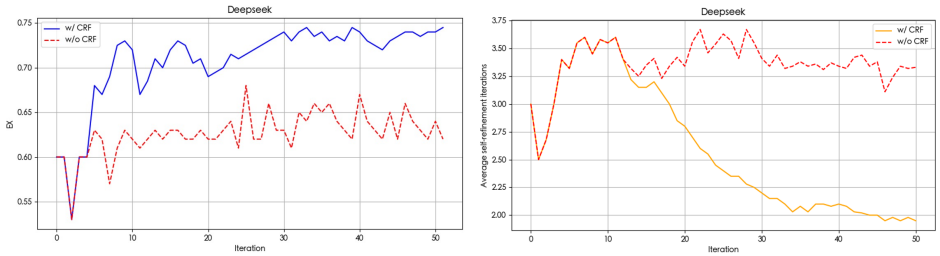
Table 2: Performance evaluation on BIRD Dev set. *Close World* Mode means Sniffer agent only uses its own knowledge inside LLM.

traditional self-correction agents fail, proving its architectural superiority in fair, apples-to-apples comparisons.

Comparing our Open World and Close World variants reveals the distinct value of active knowledge retrieval. Enabling the Sniffer's web search capability pushes the performance boundary further, reaching an EX of 88.2% on Spider and 74.53% on BIRD. While the Close World performance confirms that our reasoning architecture is robust, the additional gain from the Open World mode highlights its ability to resolve "last-mile" semantic ambiguities—such as identifying specific entity names or domain abbreviations—that are inaccessible via reasoning alone. Notably, the consistency of these gains across both benchmarks verifies our hypothesis that the "Knowledge Gap" is a pervasive issue in real-world databases. The combination of CRF for logic adaptation and the Sniffer for semantic grounding allows SEEK-SQL to achieve robust performance across diverse industrial scenarios.

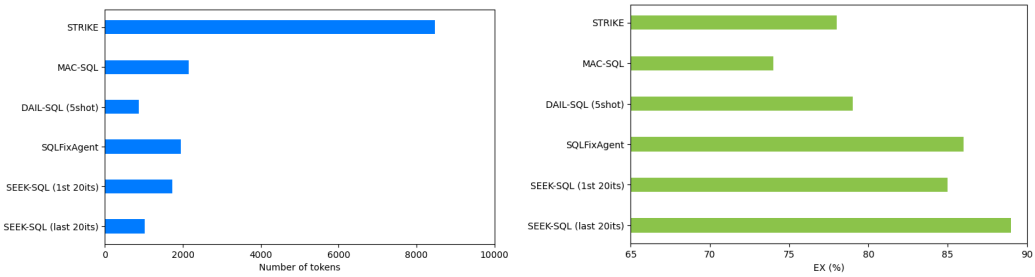
### 4.3 Adaptation and Efficiency Analysis

This section presents the core evidence for our "Low-Cost Adaptation" claim, which is critical for industrial deployment.




(a) Dynamics curve of Execution Accuracy (EX) (b) Dynamics curve of average self-refinement iterations needed for each query

Figure 2: Optimization dynamics of SEEK-SQL agents on Spider dev set. Deepseek-V3 as backbone LLM.To ensure statistical stability and mitigate the impact of query ordering, the reported curves are the fitted average over 5 random permutations of the query sequence.




(a) Average token consumption (b) Average EX value

Figure 3: Token consumption and execution accuracy of Text-to-SQL methods on Spider's dev set sample. ChatGPT-3.5-Turbo as backbone LLM as some baselines don't share their code, we use their data and the same backbone LLM.

#### 4.3.1 Cold-Start Analysis

To quantify the system's ability to "learn" a new environment, we track the evolution of Execution Accuracy (EX) and average self-refinement iterations over a sequence of 100 queries. we conducted the experiment using 5 different random samples to permute the query sequence. As shown in Figure 2, SEEK-SQL exhibits a clear "Fast Adaptation" trajectory. Initially, the system behaves like a novice, requiring an average of 3.7 refinement turns per query. However, as the Guideline Library accumulates rules from early failures, the iteration count drops rapidly to 1.8, while EX climbs from 53% to 74%. In contrast, the baseline without CRF shows a flat trend, repeating the same errors. This curve serves as empirical proof of In-Context Reinforcement Learning: the system effectively converges to a domain-adapted state within just 50 queries, eliminating the need for expensive fine-tuning.

#### 4.3.2 Token Cost Efficiency

For large-scale deployment, token consumption is a direct proxy for operating cost. Figure 3 compares the average token usage of SEEK-SQL against leading baselines. Despite being a multi-agent framework, SEEK-SQL achieves token efficiency comparable to single-agent methods like DAIL-SQL and consumes significantly fewer tokens than MAC-SQL and SQLFixAgent. This efficiency is a direct byproduct of the adaptation curve described above. While early queries may be token-heavy due to multiple refinement rounds, the system quickly "learns" to generate correct SQLs in fewer shots. Specifically, in the last 20 iterations of our experiment, SEEK-SQL consumed 45% fewer tokens than SQLFixAgent while achieving higher accuracy. This "amortized cost" model makes SEEK-SQL uniquely suitable for high-concurrency industrial applications.

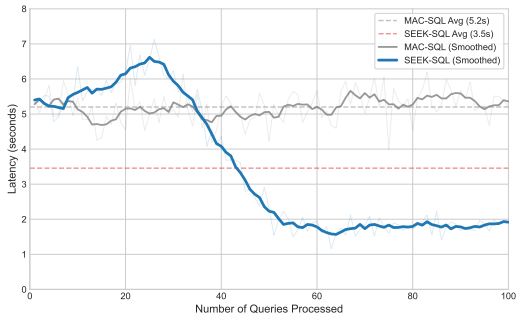



Figure 4: Real-time latency dynamics on Spider dev. sample set. Deepseek-V3 as backbone LLM.

| Configuration | Spider | | BIRD | |
|---|---|---|---|---|
| | EM | EX | EX | VES |
| **SEEK-SQL (Full)** | **78.3** | **88.2** | **74.53** | **69.72** |
| w/o Sniffer | 76.9 | 86.7 | 72.07 | 67.25 |
| w/o Memory Bank | 77.3 | 86.9 | 72.46 | 67.13 |
| w/o CRF (Strategy) | 74.4 | 85.1 | 71.25 | 65.07 |
| *Base Model Analysis* | | | | |
| DeepSeek-V3 w/o CRF | 35.4 | 70.1 | 47.39 | 55.20 |
| DeepSeek-V3 w/ CRF | 41.2 | 77.3 | 50.04 | 62.70 |

Table 3: Ablation study demonstrating the contribution of each component. CRF and Sniffer are critical for performance.

#### 4.3.3 LATENCY ANALYSIS

A persistent critique of multi-agent systems in industrial settings is the latency overhead introduced by inter-agent communication. However, unlike static pipelines that incur a constant computational cost per query, SEEK-SQL functions as a self-evolutionary system where efficiency improves intrinsically with usage. Figure 4 visualizes this "Efficiency Flywheel" effect by tracking the real-time latency dynamics of our system compared to the MAC-SQL baseline over a continuous stream of 100 queries. The visualization reveals a distinct "inverted-U" trajectory that characterizes the system's transition from exploration to exploitation.

In the initial phase, the system operates in a cold-start mode, exhibiting latency metrics similar to the baseline ($\sim$5.2s) as agents rely on standard trial-and-error interactions without historical context. As the complexity of queries increases, we observe a deliberate latency peak ($\sim$6.5s). This surge represents the computational investment required for active learning: the `Sniffer` agent performs open-world retrieval to bridge knowledge gaps, and the `Manager` executes Contrastive Self-Refinement to diagnose root causes of failures. While momentarily expensive, this active exploration is critical for synthesizing high-quality guidelines that static prompts fail to capture.

The return on this investment becomes evident in the subsequent phase, where the latency curve drops precipitously and stabilizes at approximately 1.8s—a 65% reduction compared to the baseline. This shift occurs because the system begins to "hit" the cache in the Guideline Library. By retrieving proven logic plans for recurring patterns, the agents bypass the expensive retrieval and refinement loops, effectively short-circuiting the generation process. Consequently, although SEEK-SQL incurs transient overheads during learning, its amortized average latency (3.5s) remains significantly lower than the baseline (5.2s), confirming that the evolutionary strategy successfully converts short-term computational costs into long-term operational efficiency suitable for high-concurrency environments.

### 4.4 Ablation Study

Table 3 dissects the contribution of each module. Removing the **Sniffer** leads to a notable drop in BIRD, confirming that external knowledge is foundational for handling dirty data. However, the most critical insight comes from removing the **CRF strategy** and the **Memory Bank**. Disabling these components degrades performance across all metrics, proving that the system's "intelligence" stems from its ability to reflect and remember. We also applied our CRF strategy to a standalone Deepseek-V3 model (without the full agent framework), which yielded substantial improvements, demonstrating that our contrastive refinement logic is a generalizable paradigm that can enhance various base models.

## 5 Related Work

**Text-to-SQL Systems.** The field has rapidly evolved from single-turn prompting Zhang et al. (2023); Gao et al. (2023); Tai et al. (2023) to sophisticated multi-stage frameworks Pourreza & Rafiei (2023); Dong et al. (2023); Xie et al. (2024); Jiang et al. (2023) that decompose complex queries into manageable sub-tasks. To further enhance reasoning, recent works like MAC-SQL Wang et al. (2025) and SQLFixAgent Cen et al. (2025) adopt multi-agent collaboration to verify and refine SQLs interactively. However, these approaches largely remain static: decomposition pipelines struggle with the industrial "Knowledge Gap" where schemas lack semantic context, while multi-agent frameworks treat each query as an isolated event. Without a mechanism for persistent learning, these systems suffer from "amnesia," repeating the same logical errors and incurring high token costs.

**LLM Agent Optimization.** LLMs have demonstrated strong language understanding and reasoning skills in various downstream tasksWei et al. (2022); Yao et al. (2023a;b). Beyond specific SQL tasks, optimizing autonomous agents generally focuses on two paradigms: fine-tuning LLM backbones Chen et al. (2023); Huang et al. (2023); Le et al. (2022) to internalize capabilities, or iterative prompt refinement He et al. (2024); Khattab et al. (2022) to guide behavior. While effective, fine-tuning incurs prohibitive maintenance costs during frequent schema migrations, and standard prompt tuning often lacks explicit optimization for complex tool usage. In contrast, SEEK-SQL proposes a *training-free* self-evolution paradigm. Unlike parameter-heavy optimization, we introduce a Contrastive Self-Refinement mechanism that synthesizes "guidelines" from execution trajectories. This allows the system to bridge the knowledge gap via a Sniffer agent and achieve rapid, low-cost adaptation to new environments by accumulating experience rather than updating weights.

## 6 Conclusion

In this paper, we address the critical barriers impeding the industrial deployment of Text-to-SQL systems: the semantic opacity of real-world databases (Knowledge Gap) and the prohibitive maintenance costs associated with frequent schema migrations. We propose SEEK-SQL, a self-evolutionary multi-agent framework that fundamentally shifts the paradigm from static model training to dynamic, in-context adaptation. By integrating a retrieval-augmented Sniffer, our system ensures robustness against ambiguous queries in dirty data environments. More importantly, our novel Contrastive Self-Refinement (CRF) strategy enables the framework to autonomously learn from execution failures, synthesizing reusable guidelines that drive continuous improvement without human intervention.

Extensive evaluations on the BIRD and Spider benchmarks demonstrate that SEEK-SQL not only achieves state-of-the-art performance among in-context learning methods but also significantly reduces operational costs through efficient error convergence. By eliminating the need for expensive fine-tuning while maintaining high adaptability, SEEK-SQL offers a practical, scalable solution for next-generation enterprise data interfaces. Future work will explore applying this self-evolution mechanism to more complex tasks, such as autonomous data analysis and report generation.

## REFERENCES

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. Sqlfixagent: Towards semantic-accurate text-to-sql parsing via consistency-enhanced multi-agent collaboration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25:1–53, 2024.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*, 2023.

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

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *Proceedings of The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*, 2023.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 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*, pp. 9237–9251, 2023.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 2022.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 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*, 36:42330–42357, 2024.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

Jerry Liu. LlamaIndex, 11 2022. URL `https://github.com/jerryjliu/llama_index`.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*, 2024b.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, and Adam Perelman et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348, 2023.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *Proceedings of The Thirteenth International Conference on Learning Representations*, 2025.

Ruoxi Sun, Sercan O Arik, Alexandre Muzio, Lesly Miculicich, Satya Kesav Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. SQL-paLM: Improved large language model adaptation for text-to-SQL. *Transactions on Machine Learning Research*, 2024.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. Exploring chain of thought style prompting for text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5376–5393, 2023.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 7567–7578, 2020.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 540–557, 2025.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. Tool-assisted agent on sql inspection and refinement in real-world scenarios. *arXiv preprint arXiv:2408.16991*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 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*, 2022.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37: 25981–26010, 2024.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023a.

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

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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*, 2018.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3501–3532, 2023.

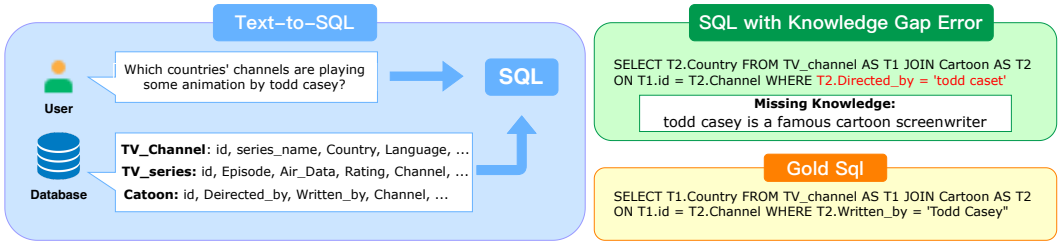



Figure 5: A real-world example of the Text-to-SQL task and knowledge gap error SQL

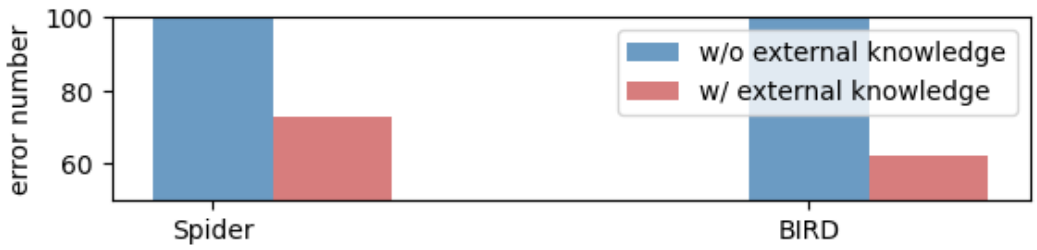



Figure 6: Error number comparison before/after given external knowledge

Ruiqi Zhong, Tao Yu, and Dan Klein. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 396–411, 2020.

## A Definition of Knowledge-gap Errors

Previous studies on SQL-fixing tasks have focused on defining SQL errors based on their superficial manifestations rather than the deep-seated causes that lead to the errors, such as mismatch errorsWang et al. (2024) or semantic errors Cen et al. (2025). Our experience in practice has revealed that a significant portion of such errors stem from a loss of essential knowledge. Fig 5 shows one example from Wang et al. (2024):The system first gives the wrong answer as it doesn't know the identity of Todd Casey. When provided with external knowledge 'Todd Casey is a famous director', the system can revise the SQL to the correct answer. So we assume the existence of a *"knowledge-gap" error*, defined as *wrong SQL sentences because of a loss of certain essential knowledge*.

To confirm the existence of such error, we sampled 100 error cases generated by GPT4o OpenAI et al. (2024) from 2 widely used benchmarks, Spider Yu et al. (2018) and BIRD Li et al. (2024), respectively. After adding additional useful information to the query, we ask GPT4o to regenerate the SQL again. Figure 6 shows the result, where we find nearly 40% errors are refined, proving the hypothesis of knowledge-gap error. Under this premise, we focus on the research problem of "how to retrieve and introduce external knowledge into real-world Text-to-SQL generation".

## B Algorithms

Algorithm 1 outlines the overall SQL generation workflow of SEEK-SQL, incorporating the retrieval of external knowledge and guidelines. Algorithm 2 details our core *Contrastive Self-refinement Strategy*, demonstrating how the system evolves by updating its Guideline Library through batch-wise reflection.

## C Implementation Details of Agents

We share details of agents of SEEK-SQL. For detailed generation progress, we provide a example in Fig 7.

**Algorithm 1:** SQL Generation Process of SEEK-SQL

**Input:** Query $q$; Database $S$; Knowledge Corpus $D$; Guideline Library $\mathcal{L}_G$
**Output:** SQL answer $SQL$

subQs = Manager.decompose(q, S);
SQLs = [];
**for** *subQ* ***in*** *subQs* **do**
  /* Retrieving context from both Knowledge and Rule memories */
  $G_{relevant}$ = GuidelineLibrary.retrieve(subQ, $\mathcal{L}_G$);
  $k$ = Sniffer.search(subQ, S, D);
  subS = Selector.select(subQ, S, k);
  subSQL = Generator.generate(subQ, subS, k, $G_{relevant}$);
  subSQL = Generator.selfCheck(subSQL);
  SQLs.append(subSQL)
**end**
sql = SQLs.gather();
ok, err = Execute(sql, S);
**if** *ok* **then**
  **return** *sql*
**else**
  /* Triggering Self-Evolution Process */
  sql = SelfRefine(err, sql, q); // See Algorithm 2
  **return** *sql*
**end**

**Algorithm 2:** Contrastive Self-refinement Strategy (CRF)

**Input:** Query $q$; Database $S$; Wrong SQL $SQL_{error}$; Guideline Library $\mathcal{L}_G$
**Output:** Refined SQL answer $SQL$

/* Phase 1: Backward SQL Fixing (Diagnosis) */
**for** *count* ***in*** *[0, MaxTryTimes]* **do**
  err = SyntaxCheck(Generator, $SQL_{error}$);
  **if** $err \neq \emptyset$ **then**
    sql = SyntaxRefine(Generator, $SQL_{error}$, err);
    ok, err = Execute(sql, S);
    **if** *ok* **then** break;
  **end**
  Solution = SemanticCheck(Manager, err, $SQL_{error}$);
  sql = SemanticRefine(Solution, Manager, err, $SQL_{error}$);
  ok, err = Execute(sql, S);
  **if** *ok* **then** break;
**end**
/* Phase 2: Contrastive Evolution (Learning) */
$T_{pos} \leftarrow$ Current successful trajectory;
$T_{neg} \leftarrow$ Previous failed trajectory;
ReflectionBatch.append($[T_{pos}, T_{neg}]$);
**if** $ReflectionBatch.size == SettledSize$ **then**
  Guideline $g$ = Manager.contrast(ReflectionBatch);
  Vector $v_q$ = EmbeddingModel(q);
  /* Update Long-term Rule Memory */
  GuidelineLibrary.insert($g, v_q$);
**end**
/* Update Short-term Case Memory */
MemoryBankUpdate($T_{pos}$);
**return** *sql*

### C.1 Sniffer Agent Configuration

To address the diverse security and data availability constraints in industrial environments, we implement the Sniffer agent with three distinct modes. The specific technical stack for each mode is as follows:

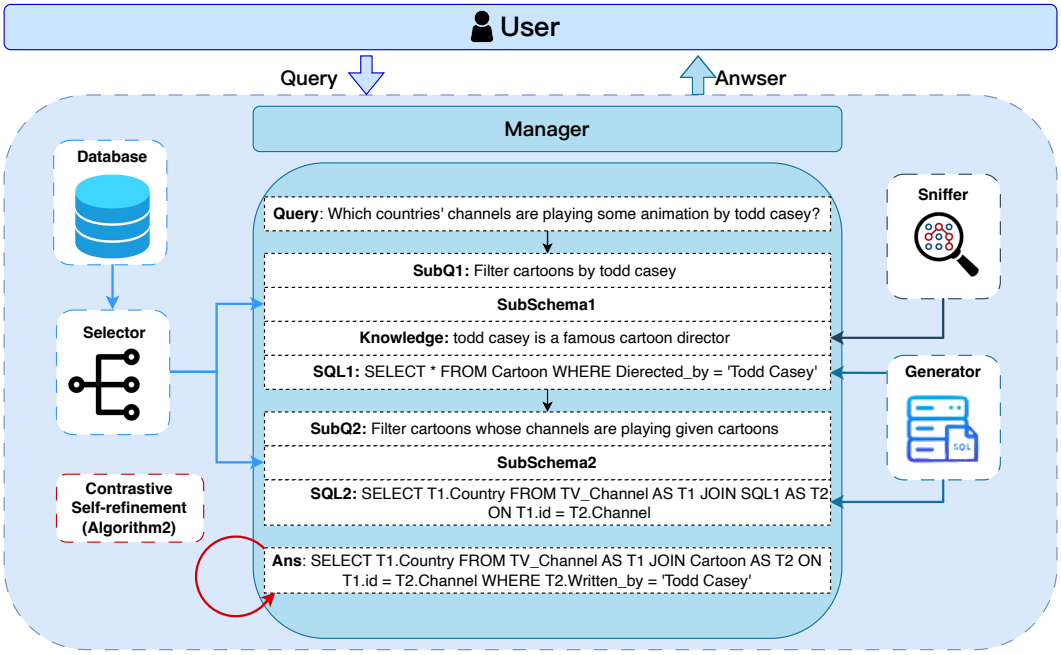



Figure 7: SQL generation progress of SEEK-SQL framework

**Local Mode (Privacy-First).** This mode is designed for scenarios where data cannot leave the local intranet. We implement a lightweight Retrieval-Augmented Generation (RAG) system using LlamaIndex Liu (2022). We use `E5-v2-base` Wang et al. (2022) to vectorize the external knowledge corpus (e.g., data dictionaries, business manuals). We employ `Flan-T5-base` Chung et al. (2024) to synthesize the retrieved chunks into concise knowledge summaries. This configuration ensures that the Sniffer can bridge the knowledge gap without transmitting sensitive schema data to external APIs.

**Open World Mode (Performance-First).** For non-sensitive scenarios requiring broad common sense (e.g., "Who is Todd Casey?"), we utilize the Web-search-pro tool Liu et al. (2024b) powered by the Google Search Engine. The Sniffer automatically generates search queries based on the semantic ambiguity detected in the user query and summarizes the top-5 search results.

**Hybrid Mode.** This mode comprehensively compares information from both local knowledge bases and online search results, prioritizing local definitions when conflicts arise.

### C.2 Forward-Backward Decomposition (FBD)

Standard Chain-of-Thought (CoT) decomposition often suffers from "context accumulation," where redundant information from early steps propagates to later steps, increasing token costs and confusing the LLM. To mitigate this, we utilize a Forward-Backward Decomposition strategy for the Manager agent.

The process involves three steps:

1. **Forward Generation:** The LLM initially decomposes the complex query $\mathcal{Q}$ into a sequence of sub-tasks $(t_1, t_2, \ldots, t_n)$ from front to back.
2. **Backward Pruning:** A secondary verification pass scans the sequence from back to front ($t_n \rightarrow t_1$). It removes redundant constraints in step $t_i$ that are already fully encapsulated in step $t_{i-1}$ or subsequent steps. For example, if $t_2$ filters a table that is already filtered in $t_1$, the redundant condition in $t_2$ is stripped.
3. **Executability Check:** The finalized sequence is validated forward to ensure logical continuity.

This strategy ensures that each sub-agent receives only the *minimum necessary context*, thereby reducing the likelihood of hallucination and saving tokens.

### C.3 Configuration of Standard Agents

This section details the specific roles and optimization strategies of the standard agents within our framework, emphasizing their design for industrial efficiency.

**Manager.** The Manager acts as the central brain, responsible for task decomposition and agent dispatching. Unlike rigid pipelines that execute a fixed sequence of steps for every query, the Manager employs a Dynamic Planning mechanism based on Chain-of-Thought (CoT) reasoning. It assesses the complexity of the user query $\mathcal{Q}$:

- For simple queries (e.g., "Show all users"), it bypasses complex decomposition and instructs the Generator directly.
- For complex multi-hop queries, it invokes the Forward-Backward Decomposition (Appendix C.2) to create a minimal sufficient plan.

This dynamic behavior ensures that computational resources are allocated proportionally to task difficulty, avoiding "over-reasoning" on simple tasks.

**Selector.** Directly feeding the entire database schema into the LLM context is prohibitively expensive and prone to distraction errors for large industrial databases. The Selector agent functions as a pre-processor, extracting only the minimum necessary sub-schema $\mathcal{S}'$ relevant to the current sub-query. This Schema Pruning mechanism reduces the input token count by approximately 70% in our experiments on BIRD, significantly lowering API costs per query.

**Generator.** Generator is responsible for generating SQL sentences for the current sub-query based on the sub-query and the SQL sentences as. To relieve the burden on the central Manager, the Generator includes a lightweight, local self-correction loop. Before returning a result, it executes a syntax validity check. Since syntax errors (e.g., missing brackets) are deterministic and do not require complex reasoning, resolving them locally at the Generator level prevents unnecessary and expensive round-trips to the Manager, further optimizing latency.

## D Discussion on Potential Risks

In this section, we address two critical aspects regarding the long-term stability and quality assurance of the self-evolutionary mechanism, which are crucial for real-world industrial deployment.

### D.1 Guideline Library Maintenance

In real-world deployments, maintaining the purity and compactness of the Guideline Library is essential to prevent "Guideline Pollution." We implement a two-stage lifecycle management strategy to ensure the knowledge base evolves effectively:

**1. Ingestion: Semantic De-duplication.** To prevent the accumulation of redundant rules, we enforce a strict similarity check before any new guideline is committed to the library. When the Manager synthesizes a new candidate guideline $g_{new}$, we calculate its cosine similarity against all existing guidelines $\{g_i\}$ in the vector space.

$$\text{If } \max(\text{Sim}(g_{new}, g_i)) > \tau_{dup}, \text{ discard } g_{new} \tag{2}$$

We set the threshold $\tau_{dup} = 0.92$ in pratice, this may vary according to specific requirements. This ensures that the system does not store multiple variations of the same logical constraint, keeping the retrieval space sparse and distinct.

| Configuration | Spider | | Private Env. | |
|---|---|---|---|---|
| | EM% | EX% | EX% | EM% |
| SEEK-SQL w/ G.L.M | 87.3 | 77.1 | **96.1** | **85.2** |
| SEEK-SQL w/o G.L.M | **87.4** | **77.2** | 88.3 | 71.3 |

Table 4: Ablation study on Guideline Library Maintenance (G.L.M). While maintenance shows negligible impact on static benchmarks (Spider), it effectively prevents "Knowledge Pollution" in a Private Environment (80K+ tables) running for over 1 month.

**2. Eviction: Feedback-Weighted LRU.** We impose a maximum capacity on the library to bound storage costs. When the limit is reached, we employ a Feedback-Weighted Least Recently Used (LRU) eviction policy. Unlike standard LRU which updates usage statistics upon any retrieval, we introduce a *Quality Verification* gate:

- A guideline's "Last Active Timestamp" and "Frequency Count" are updated **only if** the downstream SQL generation leads to a successful execution (High EX).
- If a guideline is retrieved but leads to a failure, its freshness score is **not** updated, causing it to decay faster.

This mechanism serves as a natural selection process: guidelines that repeatedly contribute to correct answers are preserved ("Survival of the Fittest"), while "spurious" or "toxic" guidelines naturally drift to the bottom of the LRU stack and are eventually purged.

It is important to acknowledge that within the standard experimental benchmarks used in Section 4.2 (Spider and BIRD), the limited scale and short session duration are insufficient to trigger the pollution issues mentioned above. Therefore, we did not enforce these maintenance protocols in the primary experiments reported in the main text. We document these strategies and the following ablation study specifically to address potential long-term deployment challenges that readers and practitioners may encounter in future industrial applications.

**Empirical Validation of Maintenance Strategy.** To demonstrate the necessity of these protocols in such scenarios, we conducted a comparative ablation study (Table 4). On the static Spider benchmark, the impact of Guideline Library Maintenance (G.L.M) is indeed negligible (77.1% vs. 77.2%), confirming our decision to exclude it from the main experimental setup. However, the value of G.L.M becomes decisive in a real-world Private Industrial Environment (electric power sector domain, 80K+ tables) monitored over 30 days. In the absence of maintenance ('SEEK-SQL w/o G.L.M'), the accumulation of conflicting rules caused Execution Accuracy to degrade to 88.3%. By enabling our lifecycle strategies ('SEEK-SQL w/ G.L.M'), the system pruned obsolete knowledge and sustained a high accuracy of 96.1%. This comparison validates that while maintenance is optional for academic benchmarks, it is a prerequisite for sustainable industrial deployment.

### D.2 Mitigating Spurious Correctness

Since guidelines is automatically generated and managed during runtime, there is a risk that false guidelines may be mistakenly created and 'luckily' added to the library. Thus guidelines may serve as a "bad teacher" and worsen our system's performance. To empirically assess the risk of the "Bad Teacher" problem, we conducted a rigorous manual audit of the synthesized Guideline Library.

We randomly sampled 200 synthesized guidelines generated during the BIRD and Spider evaluation phases. Two human experts annotated these guidelines to classify them into Valid Logic (capturing true domain constraints) or Spurious Patterns (hacky shortcuts that coincidentally pass execution). As shown in Table 5, the prevalence of spurious correctness is extremely low, averaging only 3.5% across both datasets. This high purity is attributed to the Manager's *Dual-Verification* mechanism during the diagnosis phase, which validates not only the execution result but also the semantic alignment between the SQL logic and the sub-query intent.

Furthermore, the Feedback-Weighted LRU mechanism (Section D.1) serves as a secondary defense against these rare "leakage" cases in real-world environment. Spurious guidelines typically suffer

| Source | Sampled | Valid Rules | Spurious Rules | Spurious Rate |
|---|---|---|---|---|
| Spider | 100 | 97 | 3 | 3.0% |
| BIRD | 100 | 96 | 4 | 4.0% |
| **Total** | **200** | **193** | **7** | **3.5%** |

Table 5: Manual audit statistics of the Guideline Library. "Spurious Rules" refer to guidelines that yield correct execution results but contain logically flawed or non-generalizable constraints. The low spurious rate confirms the reliability of our semantic diagnosis mechanism.

from poor generalization—they function as "overfitted" solutions to specific past queries. Consequently, when retrieved for novel, unseen queries, they are statistically likely to fail execution checks. This failure prevents their "freshness" score from being updated, causing these bad rules to naturally drift to the bottom of the eviction queue and be purged rapidly, ensuring the long-term health of the knowledge base.

## E CASE STUDY

To demonstrate the efficacy of our Contrastive Self-Refinement (CRF) strategy, Table 6 details the complete lifecycle of a complex query from the BIRD dataset. This example illustrates how the system transitions from a "Knowledge Gap" failure to a successful execution, and ultimately synthesizes a reusable guideline to prevent future errors.

Table 6: A step-by-step case study illustrating the evolution from a cold-start failure to a generated guideline. The narrative highlights the interaction between the Sniffer's knowledge retrieval and the Manager's contrastive reflection.

| Process Phase | System Narrative & Execution Details |
|---|---|
| *1. Input & Scenario* | The user initiates a query: *"Which countries' channels are playing some animation by todd casey?"*. The underlying database schema contains tables for `TV_Channel` and `Cartoon`, but lacks explicit metadata indicating whether "Todd Casey" is a person or a show title. |
| *2. Cold-Start Failure (Negative Trajectory)* | Lacking external context, the Generator mistakenly infers that "Todd Casey" represents the title of an animation. It generates a SQL query with the condition `WHERE Title = 'Todd Casey'`. When executed against the database, this query returns an Empty Set, as no such title exists. This represents a typical semantic hallucination caused by the Knowledge Gap. |
| *3. Diagnosis & Repair (Positive Trajectory)* | Upon detecting the execution anomaly, the Manager triggers a semantic check. The Sniffer agent is activated in Open World mode and retrieves external knowledge stating that *"Todd Casey is a famous cartoon director and writer"*. Leveraging this context, the Selector re-maps the intent to the `Written_by` and `Directed_by` columns. The Generator then synthesizes the corrected SQL:<br>`SELECT T1.Country ... WHERE T2.Written_by = 'Todd Casey' OR T2.Directed_by = 'Todd Casey'.`<br>This refined query executes successfully, returning valid results such as 'USA' and 'France'. |
| *4. Contrastive Evolution (Guideline Generation)* | In the final phase, the Manager performs a contrastive analysis between the failed trajectory (Title match) and the successful trajectory (Role match). It identifies that the preposition "by" followed by a proper noun in this context implies authorship rather than a title. Consequently, the system synthesizes and stores the following guideline into the Guideline Library for future retrieval:<br>*"Rule: When a query references a creative work followed by a proper noun (e.g., 'by X'), the system must verify if X is a creator (Director/Writer) using the Sniffer, rather than defaulting to a Title match."* |

## F ETHICAL CONSIDERATIONS

To prioritize data privacy and reproducibility, we utilize open-source benchmarks, while supplementary industrial tests (Appendix D) employ strictly sanitized, PII-free data. In "Open World" mode, the Sniffer restricts retrieval to public knowledge and is architecturally barred from uploading local schemas to external servers. Regarding operational risks, despite improved alignment, we advocate deploying SEEK-SQL as a supervised decision-support tool. Production environments must enforce safeguards—including read-only permissions, timeouts, and sandboxing—to prevent Denial-of-Service scenarios or data corruption. Additionally, we certify that all engineers involved in the system deployment and data monitoring processes underwent rigorous privacy training and received fair compensation.