RSTAR: RELATIONAL AND SEMANTIC TOPOLOGY-AWARE REFINEMENT FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SQL aims to translate natural language questions into executable Structured Query Language (SQL) statements, allowing users to access relational databases through plain language instead of formal query syntax. Recent advances in Large Language Models (LLMs) have established prompt-based reasoning and retrieval-augmented generation as dominant paradigms for Text-to-SQL. However, these approaches often treat schema linking, example retrieval, and result verification as isolated stages guided primarily by surface-level semantics, leading to schema topology ambiguity, retrieval mismatch, and execution-level semantic blind spots when handling complex and heterogeneous schemas of enterprise databases. To address these issues, this paper proposes a Relational and Semantic Topology-Aware Refinement (RSTAR) framework for Text-to-SQL. First, a relational topology-aware schema linker is designed to constrain LLM reasoning to valid foreign-key navigation paths for eliminating hallucinated intermediate-table joins under schema topology ambiguity. Second, a semantic topology graph retriever is proposed to disentangle structural topologies from domain entities for resolving retrieval mismatch via structurally isomorphic few-shot retrieval across heterogeneous databases. Third, an execution-feedback refinement agent is built to convert sandbox execution signals into localized value-aware feedback for repairing execution-level semantic errors. Extensive experiments on five benchmark datasets across several mainstream LLMs demonstrate that RSTAR significantly outperforms state-of-the-art Text-to-SQL competitors, attaining 95.4% execution accuracy on Spider and 79.2% on BIRD and surpassing the strongest baseline by 6.9 points on UniSQL. The source code and datasets are submitted as Supplementary Materials for Reproducibility.

1 INTRODUCTION

Text-to-SQL aims to translate natural language questions into executable Structured Query Language (SQL) statements, allowing users to access relational databases through plain language instead of formal query syntax (Shi et al., 2025). With the emergence of Large Language Models (LLMs), LLM-based semantic parsing has become a dominant paradigm for complex cross-domain SQL generation (Hong et al., 2025). However, as modern enterprise databases continue to expand in scale and complexity, existing pipelines often struggle to accurately understand heterogeneous schemas, reason over densely connected relational structures, and maintain semantic consistency during SQL generation, limiting their robustness in real-world deployment scenarios (Glass et al., 2025).

Mainstream solutions have evolved from encoder–decoder parsers with relation-aware, graph-enhanced schema encoding (Huang et al., 2021) to LLM-centric paradigms: prompt engineering and task decomposition (Dai et al., 2025; Li et al., 2023c), reasoning-driven generation (Guo et al., 2025), retrieval-augmented generation (RAG) (Gao et al., 2023b; Tan et al., 2026), and, most recently, structure-aware retrieval and agentic execution feedback (Mai et al., 2024; Zeng et al., 2025; Ma et al., 2024).

Despite these advances, existing methodologies remain insufficient against the deep complexity of enterprise-scale databases due to three persistent challenges, as illustrated in Figure 1.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

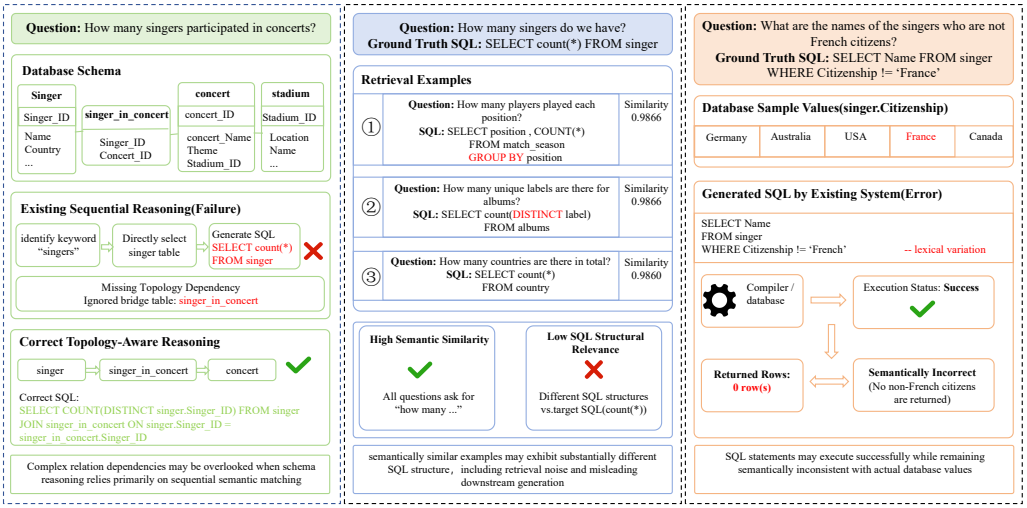


Figure 1: Persistent challenges in enterprise-scale Text-to-SQL systems.

First, as shown in Figure 1(a), schema understanding and linking over large, highly interconnected databases remains challenging. CoT prompting decomposes reasoning sequentially, whereas relational schemas are inherently graph-structured, governed by foreign-key constraints and multi-hop table interactions; implicit linear reasoning therefore misses long-range relational paths, causing hallucinated intermediate-table selection and unstable multi-table joins.

Second, as shown in Figure 1(b), retrieval mismatch persists because representations remain sensitive to database-specific identifiers: retrievers recall examples with similar surface wording but fundamentally different SQL skeletons, while overlooking structurally relevant examples from heterogeneous databases.

Third, as illustrated in Figure 1(c), execution-level semantic blind spots amplify reasoning inconsistency in open-loop generation. Syntactically valid SQL frequently returns empty or semantically wrong results due to value mismatches or synonym variations, and such failures emit no explicit error signal, so models may mistake execution success for correctness.

Motivated by these limitations, we propose RSTAR, a closed-loop Text-to-SQL framework built on a single principle: Relational and Semantic Topology-Aware Refinement. RSTAR grounds schema linking, example retrieval, and execution verification in one structural substrate, the database topology, and closes the generation loop through execution feedback. As illustrated in Figure 2, three core modules operate at the linking, grounding, and verification stages of the pipeline. First, a relational topology-aware schema linker constrains reasoning to valid foreign-key navigation paths over a directed attribute graph abstraction of the schema. Second, a semantic topology graph learning schema-augmented retriever retrieves few-shot examples that are structurally isomorphic to the target query across heterogeneous databases. Third, an execution-feedback refinement agent converts sandbox execution signals, such as error stacks and empty result sets, into localized value-aware feedback for iterative repair. As shown in Figure 4 (Appendix A.1), existing frameworks treat these capabilities as isolated heuristics and thus only partially alleviate individual issues; in contrast, RSTAR addresses all three challenges within a single closed loop.

The main contributions of this paper are summarized as follows:

1. We propose RSTAR, a closed-loop Text-to-SQL architecture that unifies topology-constrained linking, structure-isomorphic retrieval, and execution-feedback refinement in a single generation loop, simultaneously addressing the three challenges above.

2. **Relational Topology-Aware Schema Linker**: an explicit topology-aware reasoning process for schema linking and multi-table relation modeling, which abstracts the database schema as a directed attribute graph and constrains LLM inference within valid foreign-key navigation paths through a progressive SFT–MTL–GRPO training paradigm.

3. **Semantic Topology Graph Learning Schema-Augmented Retriever**: a two-stage retriever that combines relational graph attention with SQL-skeleton contrastive learning to disentangle textual semantics from structural topologies, reducing sensitivity to database-specific lexical variations and enabling structurally isomorphic example selection across heterogeneous databases.

4. **Execution-Feedback Refinement Agent**: a ReAct-based agent that converts sandbox execution signals (error stacks, empty result sets) into localized, value-aware feedback through AST-level fuzzy probing of real data, enabling dynamic self-repair of execution-level semantic inconsistencies.

2 Related works

Text-to-SQL has evolved from encoder–decoder parsers with graph-enhanced schema alignment (Tang et al., 2025) to LLM-centric paradigms built on prompt engineering and task decomposition (Qu et al., 2024), exemplified by subtask decomposition in DIN-SQL (Pourreza & Rafiei, 2023), automatic Chain-of-Thought generation in ACT-SQL (Zhang et al., 2023), zero-shot bias calibration in C3-SQL (Dong et al., 2023), and pseudo-SQL task alignment in TA-SQL (Qu et al., 2024). Retrieval-Augmented Generation further grounds generation with in-context demonstrations (Zhu et al., 2025), progressing from dense lexical–semantic retrieval (Shi et al., 2021) to structure-aware selection in DAIL-SQL, RESDSQL, RAT-SQL, and SchemaRAG (Gao et al., 2023a; Li et al., 2023a; Wang et al., 2021; Wu et al., 2026). Most recently, reinforcement learning and agentic execution feedback have enabled dynamic self-alignment, via memory-efficient policy optimization such as GRPO (Shrivastava et al., 2025), multi-turn interactive refinement (Hua et al., 2026; Zhang et al., 2026), execution-grounded rewards (Yao et al., 2026; Liu et al., 2026), role-specialized multi-agent verification in MAC-SQL (Wang et al., 2025a), and preference-optimized multi-path refinement in CHASE-SQL (Pourreza et al., 2024).

However, these advances remain architecturally isolated. Concurrent efforts alleviate individual symptoms only in part: SchemaGraphSQL performs schema linking through training-free pathfinding over foreign-key graphs (Safdarian et al., 2026), and LinkAlign scales schema linking to multi-database environments via multi-round retrieval (Wang et al., 2025b); yet, the former treats all join paths as equally weighted heuristics without learned reasoning, and the latter targets retrieval scale rather than generation quality, leaving schema linking, retrieval, and verification disconnected. Prompt-centric methods reason over schemas through flat linear semantics, which drifts on interconnected enterprise databases (Singh & Bedathur, 2025; Sheng et al., 2025); retrieval representations stay sensitive to domain-specific identifiers and fail to reflect physical join topologies; and execution feedback largely reacts to syntax errors, while empty result sets caused by value mismatches pass silently. RSTAR targets exactly these three gaps—schema topology ambiguity, retrieval mismatch, and execution-level semantic blind spots—through tightly integrated topological navigation, structure-isomorphic retrieval, and value-aware execution refinement.

3 Framework

3.1 The Overall Structure

To overcome the vulnerabilities of conventional open-loop Text-to-SQL architectures, such as topological path hallucinations during multi-table joins and unverified execution-level errors, we propose RSTAR, a closed-loop, collaborative framework. As illustrated in Figure 2, our architecture establishes a self-calibrating generation pipeline by seamlessly integrating schema graph constraints, topology-aware example retrieval, and dynamic execution feedback.

The framework operates across three core modules at different generation stages. First, the **Relational Topology-Aware Schema Linker**, deployed during semantic alignment, produces the aligned schema subset. Second, the **Semantic Topology Graph Learning Schema-Augmented Retriever**, deployed during context grounding, selects few-shot examples that are structurally isomorphic to the

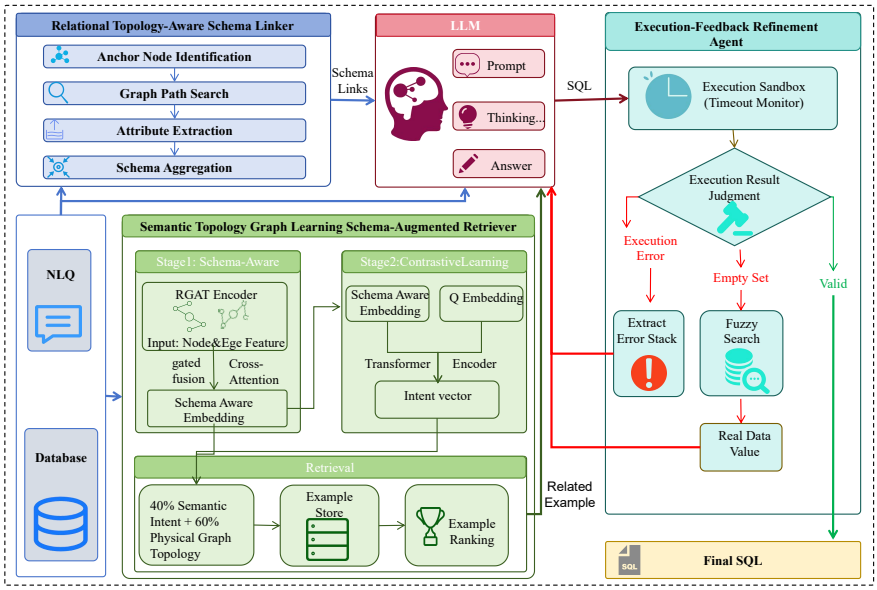


Figure 2: The overall architectural pipeline of RSTAR with integrated structure-aware retrieval and execution feedback.

target query. Third, the **Execution-Feedback Refinement Agent**, deployed during post-verification, executes candidate SQL in a sandbox and repairs it until a valid non-empty result is returned. The schema links and the retrieved examples jointly condition the LLM generation, while the execution outcome feeds back into the refinement loop, closing the pipeline.

3.2 Relational Topology-Aware Schema Linker

Traditional schema linking treats the schema as a flat text sequence, leaving LLMs prone to topological path hallucinations during complex multi-table joins. Relational Topology-Aware Schema Linker instead abstracts the schema into a directed attribute graph $G = (V_T, V_C, \mathcal{E})$, where V_T and V_C denote table and column nodes and $\mathcal{E}$ denotes edges induced by foreign-key (FK) constraints, and constrains LLM reasoning within explicit FK navigation paths.

3.2.1 Relational Topology Navigation Chain-of-Thought

Given an NLQ Q, schema linker decomposes schema linking into four graph-constrained stages:

- **Anchor identification:** map core business concepts in Q to table nodes, $V_{\text{anchor}} = f_{\text{anchor}}(Q, V_T) \subseteq V_T$, directly locating the underlying data source (e.g., the "budget" table) while rejecting spurious mappings without FK associations;
- **Graph path search:** derive join paths $P = f_{\text{path}}(V_{\text{anchor}}, \mathcal{E})$ along which every hop must follow actual FK edges, eliminating illusory cross-table associations;
- **Attribute extraction:** locate the SELECT and WHERE columns on the validated paths, $V_{\text{target}} = f_{\text{attr}}(Q, P, V_C) \subseteq V_C$;
- **Schema aggregation:** normalize and merge the retrieved nodes into the final aligned schema subset S_{final}, which serves as a rigid constraint for subsequent SQL generation.

3.2.2 Progressive Three-Stage Training

To internalize schema linker reasoning and acquire fault tolerance, we adopt a progressive training paradigm (pipeline illustrated in Appendix A.2.1). **Stage 1 (SFT):** full fine-tuning on navigation trajectories—each sample pairs the question, schema, and gold schema links with a four-step CoT chain and the final answer—under the standard autoregressive cross-entropy loss (Eq. 7). **Stage 2**

(MTL): joint optimization over error detection, error correction, and standard generation on GPT-constructed erroneous trajectories; inverse-frequency sampling weights $(0.3, 0.3, 1.0)$ counteract severe task imbalance (Eq. 8). **Stage 3 (GRPO):** for each question we sample $G = 4$ responses and use standardized intra-group returns as advantages, avoiding the critic-induced OOM of PPO at 7B scale (Shao et al., 2024); a hybrid reward (Eq. 12) combines a schema-overlap content reward—true positives $+2.0$, false positives -0.5, false negatives -3.0, and an F1 bonus $+0.5$—with a format reward that penalizes non-standard output structures, jointly stabilizing generation quality and reasoning structure.

3.3 Semantic Topology Graph Learning Schema-Augmented Retriever

Traditional dense retrieval overfits to surface-level lexical similarity, so recalled examples often diverge in SQL skeletons and join topologies. Semantic topology graph learning schema-augmented retriever disentangles textual semantics from structural topologies through a two-stage representation learning scheme and a hybrid retrieval mechanism (architecture in Appendix A.2.2).

3.3.1 Schema-Aware Representation via Relational Graph Attention

Phase I parses the schema into a directed heterogeneous graph with node set $V = V_T \cup V_C$ and edge set $\mathcal{E} = \mathcal{E}_{TC} \cup \mathcal{E}_{FK}$, comprising table–column and primary-key–foreign-key relations. Starting from PLM text embeddings, relation-specific RGAT layers Wang et al. (2020) propagate messages along $\mathcal{E}_{TC}$ and $\mathcal{E}_{FK}$ (Eq. 13–15); a dynamic gate then fuses semantic and topological features to counteract over-smoothing, and the question vector attends over the fused nodes via multi-head cross-attention, yielding the pattern-aware representation z^{stage1} supervised by SQL skeletons.

3.3.2 Domain-Agnostic Contrastive Learning

Phase II severs the dependence on specific data domains by stripping each gold SQL into an AST skeleton: string and numeric constants are replaced by `<VAL>`, table and column identifiers by merged `<ID>` tags, retaining only relational algebra operations (`SELECT`, `JOIN`, `GROUP BY`, ...). The concatenation of z^{stage1} and the question embedding is encoded by a causal Transformer and projected to a 1024-dimensional intent vector. Training combines InfoNCE instance discrimination (van den Oord et al., 2018) with an auxiliary local similarity loss (Eq. 19):

$$\mathcal{L}_{\text{stage2}} = \mathcal{L}_{\text{CL}} + \omega\, \mathcal{L}_{\text{sim}}. \tag{1}$$

3.3.3 Topological Waveform Fingerprint Retrieval

At inference, the retriever recalls examples through a dual-channel (intent + topology) mechanism. The cross-attention weights $a \in \mathbb{R}^{|V|}$ over G are decoupled into table and column intervals and sorted in descending order, discarding node identities to form a fingerprint invariant to database size and naming:

$$S_{\text{topo}} = \left[\text{sort}_{\text{desc}}(a_{\text{table}}) \parallel \text{sort}_{\text{desc}}(a_{\text{column}})\right]. \tag{2}$$

Candidates are ranked by a hybrid similarity that jointly scores the semantic intent space and the physical topological space:

$$S_{\text{final}} = \alpha\ \text{sim}(z_{\text{test}}, z_{\text{ret}}) + (1-\alpha)\ \text{sim}(S^{\text{test}}_{\text{topo}}, S^{\text{ret}}_{\text{topo}}), \tag{3}$$

where we set $\alpha = 0.4$, assigning a dominant 60% weight to the topological fingerprint and achieving robust cross-domain, structure-isomorphic recall.

3.4 Execution-Feedback Refinement Agent

Syntactically valid SQL can still yield runtime errors or empty result sets due to value mismatches and implicit logical discrepancies. The Execution-Feedback Refinement Agent treats the database compiler as an interactive sandbox and establishes a dynamic closed-loop calibration mechanism through multi-turn interaction following the ReAct paradigm (Yao et al., 2023): sandbox execution actions and reflective reasoning over their outcomes are interleaved to guide iterative repair. (Figure 3).

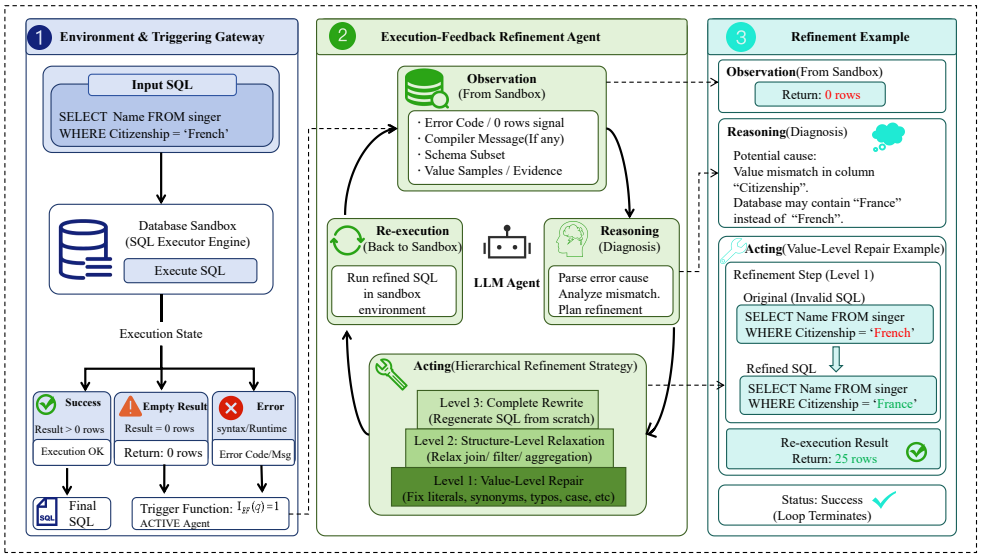


Figure 3: The adaptive control flow and hierarchical calibration mechanism of the Execution-Feedback Refinement Agent.

3.4.1 Execution State Modeling

The refinement agent models each sandbox execution by a state function $\mathcal{E}(q) \in \{\text{Success}, \text{Empty}, \text{Error}\}$: `Success` returns a non-empty result set; `Empty` executes without error but returns zero rows (typically from mismatched string values or overly strict constraints); `Error` aborts on syntax errors, non-existent schema references, or runtime exceptions. To guard against computationally explosive queries (e.g., accidental cross-joins), the sandbox enforces a 10-second timeout through native SQLite interrupts, returning a dedicated `EXECUTION_TIMEOUT` stack to the LLM; implementation details are given in Appendix A.2.3.

3.4.2 Value-Aware Reflection

Rather than asking the LLM to blindly guess the cause of an `Empty` state, the refinement agent bridges generated strings with actual data distributions. An AST parser (sqlglot) extracts each WHERE/HAVING predicate as a tuple $C_x = (\text{col}_k, \text{op}_k, \text{val}_k)$, and a fuzzy probe searches the database for near-matches:

$$\mathcal{F}(\text{col}_k, \text{val}_k) = \{v \mid \text{sim}(v, \text{val}_k) > \tau\}, \tag{4}$$

where $\text{sim}(\cdot, \cdot)$ is the string similarity implemented via Python's `difflib`. A non-empty $\mathcal{F}$ proves that the empty result stems from a lexical mismatch (e.g., querying `'T. Swift'` where the database stores `'Taylor Swift'`); the retrieved actual values are then packaged as factual evidence and injected into the feedback prompt, forcing the LLM to align its conditions with exact database semantics.

3.4.3 ReAct-Based Multi-Turn Refinement

the refinement agent is activated only when needed, $I_{\text{EF}}(q) = 1 \Leftrightarrow \mathcal{E}(q) \in \{\text{Empty}, \text{Error}\}$, and iterates

$$q^{(t+1)} = \text{LLM}\left(q^{(t)}, \mathcal{E}(q^{(t)}), \mathcal{F}\right) \tag{5}$$

with an explicit diagnosis-then-repair step at each turn, terminated at $\mathcal{E}(q) = \text{Success}$ or a strict budget of $T_{\max} = 3$ iterations. Repairs follow a hierarchy that minimizes token cost: *value-level* fixes of literals in WHERE conditions (guided by $\mathcal{F}$), *structure-level* relaxation of joins and filters, and

complete rewrite only when lightweight strategies fail. The overall inference cost is thus bounded by $O(K \cdot (C_{\text{LLM}} + C_{\text{Exec}}))$ with $K \leq T_{\max}$, guaranteeing reliability without exponential overhead.

3.5 Algorithm Design

The complete inference pipeline is formalized in Algorithm 1 (Appendix A.2.4), where its computational complexity is also analyzed.

4 Experiments

In the subsequent experiments, we aim at answering the following research questions (RQs):

- **RQ1 (Performance Comparison):** Does our proposed model achieve the best performance among the baseline models considered in this study?
- **RQ2 (Ablation Experiments):** Are all the modules in our proposed framework necessary for the model to function effectively?
- **RQ3 (Error Analysis):** What types of errors occur in our proposed model?
- **RQ4 (Self-Consistency & Efficiency):** How does Self-Consistency impact schema linking in our schema linker, and what are the key factors affecting maintainability and efficiency?
- **RQ5 (The Retriever Evaluation):** What factors influence the performance of the retriever, and to what extent is the model itself effective?

Organization of Experimental Results. Due to strict main-text page constraints, our experimental results are structured as follows:

- **Primary Results in Main Text:** Section 4.1 presents the main benchmark performance comparisons (**RQ1**), and Section 4.2 details the core module ablation studies (**RQ2**).
- **Extended Analysis in Supplementary Material:** Supplementary details for **RQ1** (including non-parametric statistical significance testing and fine-grained difficulty breakdowns) and extended ablation variations for **RQ2** are provided in Appendix A.4.1 and Appendix A.4.2, respectively. Complete empirical investigations for **RQ3** (Error Analysis), **RQ4** (Self-Consistency & Efficiency), and **RQ5** (The Retriever Detailed Evaluation) are deferred in full to Appendix A.4.3, A.4.4, and A.4.5. Full experimental setup details (datasets, metrics, baselines, and hyperparameter implementations) are documented in Appendix A.3.

4.1 Performance Comparison(RQ.1)

Table 1 summarizes the performance comparison across five benchmark datasets. Our framework consistently achieves top-tier performance across all underlying backbone LLMs, demonstrating strong robustness and cross-domain generalization. Detailed numerical analysis, non-parametric statistical significance tests (Loss/Win, Wilcoxon signed-rank, and Friedman ranking), and fine-grained difficulty breakdowns against fine-tuned baselines are deferred to Appendix A.4.1.

4.2 Ablation Experiments (RQ.2)

Table 2 summarizes the performance impact of incrementally ablating each core module across five benchmark datasets. The results demonstrate that removing any single component leads to consistent performance degradation across all underlying backbone LLMs, confirming that our architecture's effectiveness stems from the tight, synergistic integration of topological navigation, structure-aware retrieval, and execution feedback. Detailed numerical breakdowns, non-parametric statistical significance tests (Loss/Win counts, Wilcoxon signed-rank, and Friedman ranking) are deferred to Appendix A.4.2.

Table 1: Model performance on benchmark datasets. The last two columns under the ⋆ model report results for SAR trained with BIRD-based (left) and Spider-based (right) contrastive learning data.

Model	LLM	Spider		BIRD		Spider 2.0	Beaver	UniSQL	
		EX	EM	EX	VES	EX	EX	EX	
TA-SQL	GPT-4o	85.00	68.70	52.43	23.21	1.65	0	35.16	
	Deepseek-v3	84.60	75.80	56.24	25.66	1.46	0	37.01	
	Qwen-7B	76.40	19.70	33.11	12.92	0	0	24.32	
	XiYanSQL-14B	76.60	33.60	37.74	17.42	1.65	2.40	26.95	
	Llama-8B	72.00	12.60	32.03	11.89	0	0	23.34	
DIN-SQL	GPT-4o	74.20	60.10	48.86	20.11	1.46	0	31.25	
	Deepseek-v3	80.30	43.70	47.11	18.81	1.46	0	32.03	
	Qwen-7B	73.80	27.30	27.50	9.67	0	0	21.29	
	XiYanSQL-14B	85.10	77.00	44.18	22.58	0.91	1.90	30.96	
	Llama-8B	73.50	27.90	26.98	10.67	0	0	20.90	
ACT-SQL	GPT-4o	73.40	45.10	47.51	22.15	1.46	0	31.45	
	Deepseek-v3	75.50	25.20	45.98	18.80	1.65	0	30.96	
	Qwen-7B	76.60	23.50	23.67	9.43	0	0	20.90	
	XiYanSQL-14B	75.80	23.70	43.16	20.35	1.46	1.44	29.49	
	Llama-8B	76.50	22.80	26.98	9.12	0	0	21.29	
C3-SQL	GPT-4o	81.90	46.90	44.46	18.86	1.46	0	30.76	
	Deepseek-v3	76.50	23.50	44.85	18.50	1.46	0	30.47	
	Qwen-7B	72.30	44.70	28.51	9.67	0	0	21.58	
	XiYanSQL-14B	78.30	48.80	42.34	22.10	1.83	1.44	29.49	
	Llama-8B	71.80	44.40	27.08	9.66	0.55	0	20.90	
Chase-SQL	GPT-4o	85.20	24.20	69.75	30.51	3.33	3.34	43.52	
	Deepseek-v3	85.10	30.80	70.01	30.30	5.83	2.87	44.00	
	Qwen-7B	80.50	31.00	65.91	26.38	0	3.34	43.58	
	XiYanSQL-14B	93.00	84.50	72.23	30.37	4.17	3.82	51.93	
	Llama-8B	70.20	17.70	55.48	19.99	0	4.31	37.23	
SchemaRAG⋆	GPT-4o	85.60	66.80	68.90	31.98	5.90	7.18	48.05	38.48
	Deepseek-v3	89.90	75.90	68.38	31.05	10.79	6.67	46.88	36.91
	Qwen-7B	80.40	53.30	54.50	25.10	7.50	2.87	37.60	28.13
	XiYanSQL-14B	93.30	88.00	65.25	34.54	19.20	10.05	48.34	38.28
	Llama-8B	80.80	64.50	60.37	29.92	7.31	2.39	40.43	30.76
RSTAR⋆	GPT-4o	**89.60**	**84.30**	**75.10**	**33.20**	**16.67**	**8.13**	**50.96**	**46.61**
	Deepseek-v3	**90.20**	**88.00**	**79.20**	**34.35**	**11.37**	**7.17**	**53.57**	**49.90**
	Qwen-7B	**85.20**	**67.20**	**76.27**	**33.60**	**8.33**	**6.21**	**52.61**	**47.67**
	XiYanSQL-14B	**95.40**	**93.70**	**76.66**	**34.55**	**20.80**	**11.57**	**55.22**	**51.64**
	Llama-8B	**86.40**	**70.80**	**71.32**	**30.46**	**7.50**	**4.78**	**49.86**	**43.71**

5 Limitations and Future Works

Although RSTAR significantly improves the execution accuracy of Text-to-SQL systems in complex scenarios, it still faces theoretical and engineering limitations when deployed in future highly concurrent, latency-sensitive real-world enterprise environments. These include excessive test-time compute overhead, restricted topological generalization on sparse databases lacking explicit foreign keys, and bounded active exploration capabilities of the reactive feedback agent. The future Text-to-SQL paradigm must evolve from a "monolithic heuristic pipeline" to an "autonomous multi-agent ecosystem." Subsequent research should focus on dynamically scaling test-time compute using Monte Carlo Tree Search (MCTS) and preference optimization strategies to break the latency bot-

Table 2: Ablation study. • indicates a result that is higher than the corresponding value of RSTAR, ∗ Indicates that the result is equal to the corresponding value in RSTAR

Model	LLM	Spider		BIRD		Spider 2.0	Beaver	UniSQL
		EX	EM	EX	VES	EX	EX	EX
w/o schema linker	GPT-4o	85.60	67.90	73.79	31.31	16.32	8.28•	50.58
	Deepseek-v3	89.90	86.20	76.34	32.52	10.88	6.88	52.51
	Qwen-7B	84.90	64.10	74.64	31.98	8.12	6.18	51.86
	XiYanSQL-14B	94.10	93.10	75.42	31.69	19.73	10.73	54.15
	Llama-8B	86.20	69.20	71.32∗	30.10	6.67	4.23	48.51
w/o retriever	GPT-4o	79.50	36.00	60.63	23.62	15.48	7.36	44.58
	Deepseek-v3	82.00	59.80	62.58	24.04	9.17	6.24	47.39
	Qwen-7B	81.20	48.60	57.56	22.18	6.58	5.53	44.68
	XiYanSQL-14B	91.70	89.70	65.06	26.46	18.79	9.85	52.42
	Llama-8B	72.00	29.90	49.93	18.64	5.87	3.71	39.94
w/o refinement agent	GPT-4o	82.00	52.20	72.10	30.80	15.84	7.84	49.83
	Deepseek-v3	85.50	65.10	78.23	33.28	10.86	6.67	52.47
	Qwen-7B	84.90	65.10	70.14	30.86	7.47	5.72	51.25
	XiYanSQL-14B	95.30	94.70•	73.34	32.03	19.15	11.13	53.76
	Llama-8B	84.00	65.10	63.49	27.12	7.18	4.14	49.32
RSTAR	GPT-4o	**89.60**	**84.30**	**75.10**	**33.20**	**16.67**	8.13	**50.96**
	Deepseek-v3	**90.20**	**88.00**	**79.20**	**34.35**	**11.37**	**7.17**	**53.57**
	Qwen-7B	**85.20**	**67.20**	**76.27**	**33.60**	**8.33**	**6.21**	**52.61**
	XiYanSQL-14B	**95.40**	93.70	**76.66**	**34.55**	**20.80**	**11.57**	**55.22**
	Llama-8B	**86.40**	**70.80**	71.32	**30.46**	**7.50**	**4.78**	**49.86**

tleneck of brute-force multi-path reasoning; combining LLM intent recognition with classical graph pathfinding algorithms (such as Dijkstra) to achieve deterministic zero-shot schema alignment with minimal overhead; adopting trajectory-based Multi-Agent Reinforcement Learning (MARL) to provide dense intermediate state feedback; and developing proactive SQL Probes to pre-interact with real data before deployment, thereby completely eliminating data prior blind spots and enabling the transition to truly dynamic, interactive next-generation autonomous database systems.

6 Conclusion

Addressing the vulnerabilities caused by open-loop generation and flat linear semantic matching in traditional Text-to-SQL systems, this paper proposes RSTAR, a closed-loop Text-to-SQL framework built on relational and semantic topology-aware refinement. RSTAR abstracts the database schema as a directed attribute graph and links questions to schema items through a relational topology-aware schema linker; it grounds generation with a semantic topology graph learning schema-augmented retriever that disentangles structural topology from domain entities; and it calibrates generated SQL through an execution-feedback refinement agent that repairs value-level semantic errors via sandbox feedback. Together, these components achieve full-lifecycle structural intervention from semantic alignment and few-shot retrieval to code execution and repair. Extensive experiments on five benchmarks demonstrate that RSTAR eliminates topological hallucinations, alleviates retrieval mismatch, and repairs execution-level semantic blind spots, setting a new performance benchmark for reliable Text-to-SQL over complex enterprise databases.

AI USE STATEMENT

Large language models were used in this work for multiple purposes: (1) polishing and refining the English writing of the manuscript; (2) assisting retrieval and discovery of related references; (3) supporting research ideation and parts of the experimental workflow; (4) generating synthetic datasets used in our experiments. All AI-generated outputs were carefully reviewed, verified, and revised by the authors. The authors retain full responsibility for the content of this paper.

ETHICS STATEMENT

Our experiments are conducted entirely on publicly available benchmark datasets (Spider, BIRD, Spider 2.0, Beaver, and UniSQL), which contain no personally identifiable information. We did not recruit human subjects or access proprietary production databases. Because Text-to-SQL systems may emit incorrect or unsafe queries, we recommend that generated SQL be reviewed and constrained by database access controls before execution on live systems. We do not anticipate direct negative societal impacts, and we release our resources solely for research reproducibility.

REPRODUCIBILITY STATEMENT

To support the reproducibility of our work, we have submitted the source code, datasets, implementation details, and necessary instructions as the Supplementary Materials. We believe that these resources provide sufficient information for reproducing the main experimental results presented in this paper.

REFERENCES

Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. Beaver: An enterprise benchmark for text-to-sql, 2026. URL https://arxiv.org/abs/2409.02038.

Yaxun Dai, Haiqin Yang, Mou Hao, and Pingfu Chao. PARSQL: Enhancing text-to-SQL through SQL parsing and reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 661–681, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.37. URL https://aclanthology.org/2025.findings-acl.37/.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, et al. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot text-to-sql with chatgpt, 2023. URL https://arxiv.org/abs/2307.07306.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation, 2023a. URL https://arxiv.org/abs/2308.15363.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, et al. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql, 2025. URL https://arxiv.org/abs/2411.08599.

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

Michael Glass, Mustafa Eyceoz, Dharmashankar Subramanian, Gaetano Rossiello, Long Vu, and Alfio Gliozzo. Extractive schema linking for text-to-SQL. *arXiv preprint arXiv:2501.17174*, 2025.

Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024. URL https://arxiv.org/abs/2406.12793.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*, 37(12):7328–7345, 2025. doi: 10.1109/TKDE.2025.3609486.

Harper Hua, Zhen Han, Zhengyuan Shen, Jeremy Lee, Patrick Guan, Qi Zhu, Sullam Jeoung, Yueyan Chen, Yunfei Bai, Shuai Wang, Vassilis Ioannidis, and Huzefa Rangwala. SQL-Trail: Multi-turn reinforcement learning with interleaved feedback for text-to-SQL. *arXiv preprint arXiv:2601.17699*, 2026.

Junyang Huang, Yongbo Wang, Yongliang Wang, Yang Dong, and Yanghua Xiao. Relation aware semi-autoregressive semantic parsing for nl2sql, 2021. URL https://arxiv.org/abs/2108.00804.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows, 2025. URL https://arxiv.org/abs/2411.07763.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *AAAI*, 2023a.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3), May 2024. doi: 10.1145/3654930. URL https://doi.org/10.1145/3654930.

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

Xinze Li, Zhenghao Liu, Chenyan Xiong, Shi Yu, Yu Gu, Zhiyuan Liu, and Ge Yu. Structure-aware language model pretraining improves dense retrieval on structured data. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 11560–11574, Toronto, Canada, July 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.734. URL https://aclanthology.org/2023.findings-acl.734/.

Tao Liu, Tao Feng, Xiangheng Li, Jinwang Song, Yifan Li, Xiaoqing Cheng, Dixuan Zhang, Siquan Li, Lin Lan, Hongying Zan, Kunli Zhang, and Chao Wu. SERL-SQL: Selective hindsight distillation for text-to-SQL reinforcement agentic learning. *arXiv preprint arXiv:2608.00485*, 2026.

Limin Ma, Ken Pu, and Ying Zhu. Evaluating llms for text-to-sql generation with complex sql workload, 2024. URL https://arxiv.org/abs/2407.19517.

Chuhong Mai, Ro ee Tal, and Thahir Mohamed. Learning metadata-agnostic representations for text-to-sql in-context example selection, 2024. URL https://arxiv.org/abs/2410.14049.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, et al. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction, 2023. URL https://arxiv.org/abs/2304.11015.

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, 2024. URL https://arxiv.org/abs/2410.01943.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation, 2024. URL https://arxiv.org/abs/2405.15307.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

AmirHossein Safdarian, Milad Mohammadi, Ehsan Jahanbakhsh, Mona Shahamat Naderi, and Heshaam Faili. SchemaGraphSQL: Efficient schema linking with pathfinding graph algorithms for text-to-SQL on large-scale databases. In *Findings of the Association for Computational Linguistics: EACL 2026*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowen Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Lei Sheng, Shuai-Shuai Xu, and Wei Xie. BASE-SQL: A powerful open source text-to-SQL baseline approach. *arXiv preprint arXiv:2502.10739*, 2025.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2), September 2025. ISSN 0360-0300. doi: 10.1145/3737873. URL https://doi.org/10.1145/3737873.

Peng Shi, Rui Zhang, He Bai, and Jimmy Lin. Cross-lingual training of dense retrievers for document retrieval. In Duygu Ataman, Alexandra Birch, Alexis Conneau, Orhan Firat, Sebastian Ruder, and Gozde Gul Sahin (eds.), *Proceedings of the 1st Workshop on Multilingual Representation Learning*, pp. 251–253, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.mrl-1.24. URL https://aclanthology.org/2021.mrl-1.24/.

Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

Rajat Singh and Srikanta Bedathur. Can the rookies cut the tough cookie? exploring the use of llms for sql equivalence checking, 2025. URL https://arxiv.org/abs/2412.05561.

Zhiwen Tan, Jiaming Huang, Qintong Wu, Hongxuan Zhang, Chenyi Zhuang, and Jinjie Gu. Rag-r1: Incentivizing the search and reasoning capabilities of llms through multi-query parallelism, 2026. URL https://arxiv.org/abs/2507.02962.

Zetong Tang, Qian Ma, and Di Wu. Auto prompt sql: a resource-efficient architecture for text-to-sql translation in constrained environments, 2025. URL https://arxiv.org/abs/2506.03598.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers, 2021. URL https://arxiv.org/abs/1911.04942.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. Mac-sql: A multi-agent collaborative framework for text-to-sql, 2025a. URL https://arxiv.org/abs/2312.11242.

Kai Wang, Weizhou Shen, Yunyi Yang, Xiaojun Quan, and Rui Wang. Relational graph attention network for aspect-based sentiment analysis. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3229–3238, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.295. URL https://aclanthology.org/2020.acl-main.295/.

Yihan Wang, Peiyu Liu, and Xin Yang. LinkAlign: Scalable schema linking for real-world large-scale multi-database text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 977–991, 2025b.

Di Wu, Zetong Tang, Yi He, and Xin Luo. Schemarag: A schema-aware retrieval-augmented generation framework for text-to-sql. *Proc. ACM Manag. Data*, 4(1), April 2026. doi: 10.1145/3786696. URL https://doi.org/10.1145/3786696.

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

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-Text2SQL-R1: Simple rewards, strong reasoning in text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

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 Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

Qingyun Zeng, Simin Ma, Arash Niknafs, Ashish Basran, and Carol Szabo. Taming sql complexity: Llm-based equivalence evaluation for text-to-sql, 2025. URL https://arxiv.org/abs/2506.09359.

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, 2023. URL https://arxiv.org/abs/2310.17342.

Shihao Zhang, X. Wang, Y. Liu, Y. Lan, and Weining Qian. Progress-SQL: Improving reinforcement learning for text-to-SQL via progressive rewards. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. Large language models for information retrieval: A survey. *ACM Transactions on Information Systems*, 44(1):1–54, 2025. doi: 10.1145/3748304.

A SUPPLEMENTARY MATERIAL

A.1 CAPABILITY COMPARISON WITH REPRESENTATIVE METHODS

To systematically benchmark the three challenges discussed in Section 1, Figure 4 compares representative state-of-the-art Text-to-SQL methodologies with our proposed framework across the adopted key technologies and the extent to which they address schema topology ambiguity, retrieval mismatch, and execution-level semantic blind spots. Although recent frameworks integrate specific techniques, such as structurally aware retrieval in SchemaRAG and test-time execution verification via multi-agent collaboration in CHASE-SQL, they intrinsically treat these components as isolated heuristics: execution-guided agents remain bottlenecked by implicit topological reasoning, generating massive redundant candidates, while retrieval enhancements lack the dynamic semantic

Method	Schema Linking	Similar Case Retrieval	Agent Technology	Schema Topology Ambiguity	Retrieval Mismatch	Execution-Level Semantic Blind Spot
TA-SQL	✔	✘	✘	○	○	○
DIN-SQL	✔	✘	✘	○	○	○
ACT-SQL	✔	✔	✘	◐	◐	○
C3-SQL	✔	✘	✘	◐	○	○
CHASE-SQL	✔	✔	✔	◐	◐	●
SchemaRAG	✔	✔	✘	◐	●	○
RSTAR	✔	✔	✔	●	●	●

✔ Supported ✘ Not Supported ◐ Partial

Figure 4: Capability comparison of representative state-of-the-art Text-to-SQL methodologies against the proposed framework across key technologies and enterprise-scale challenges.

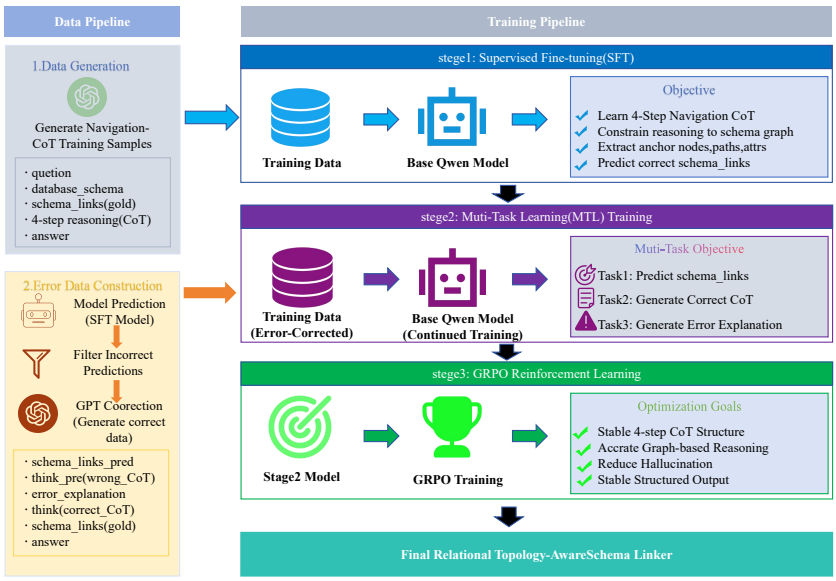


Figure 5: The multi-stage progressive training paradigm and optimization pipeline for the relational topology-aware schema linker.

validation provided by execution feedback. Our framework achieves architectural synergy by fusing explicit graph-path navigation, contrastive structural grounding, and feedback-driven calibration, addressing all three challenges simultaneously within a closed loop.

A.2 Model Details

A.2.1 Schema Linker Training Pipeline and Objectives

This section details the progressive three-stage training paradigm of the schema linker, with the full pipeline illustrated in Figure 5 and the corresponding objectives derived below.

Phase I: Supervised Fine-Tuning (SFT) for CoT Elicitation. The first stage aims to inject the underlying distribution of schema linker into the model through full fine-tuning. Given a training sample set:

$$D_{\text{SFT}} = \left\{ \left(X^{(i)}, Y_{\text{CoT}}^{(i)} \right) \right\}_{i=1}^{N}, \tag{6}$$

where Y_{CoT} contains a four-step inference chain tightly wrapped by ¡think¿ and ¡/think¿, along with the final answer. The optimization objective is the standard autoregressive cross-entropy loss:

$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{i=1}^{N}\sum_{t=1}^{|Y^{(i)}|}\log P_\theta\left(y_t^{(i)} \mid y_{<t}^{(i)}, X^{(i)}\right), \tag{7}$$

Phase II: Multi-Task Learning (MTL) for Reflection. To prevent path deviations in complex topology queries, a reflection and exploration mechanism is introduced in the second stage. We utilize GPT to automatically construct data containing erroneous inference trajectories and conduct multi-task learning (MTL). The model needs to optimize three dimensions of tasks simultaneously: (1) Error Detection ($\mathrm{T_{err}}$): identifies intermediate table illusions and foreign key errors; (2) Error Correction ($\mathrm{T_{corr}}$): outputs the correct schema linker based on the detection results; (3) Standard Generation ($\mathrm{T_{gen}}$): maintains the positive generation capability of the SFT stage. To address the issue of severe task imbalance, we employ a weighted random sampling strategy based on inverse task frequency in the DataLoader. The sampling weight multipliers for the three tasks are set to $\omega_{\mathrm{err}} = 0.3$, $\omega_{\mathrm{corr}} = 0.3$, and $\omega_{\mathrm{gen}} = 1.0$. Under this sampling strategy, the expected multi-task training loss is defined as:

$$\mathcal{L}_{\mathrm{MTL}} = \mathbb{E}_{\tau\sim D_{\mathrm{MTL}}}\left[-\sum_{k\in\{\mathrm{err},\,\mathrm{corr},\,\mathrm{gen}\}}\omega_k\log P_\theta\left(Y_k \mid X_k, \tau\right)\right], \tag{8}$$

Phase III: Group Relative Policy Optimization (GRPO) Alignment. Traditional PPO algorithms, due to the need to deploy additional Critic models, are prone to memory overflow (OOM) on large models of 7B scale. This paper adopts Group Relative Policy Optimization (GRPO) proposed by Shao et al. (2024) for final reinforcement learning alignment. GRPO generates multiple response groups ($\mathrm{Num_generations}\ G = 4$) for the same question Q, and uses the relative returns within the group to replace the external baseline, thereby achieving lightweight policy optimization. Its objective function is:

$$\begin{aligned} J_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{\substack{q\sim P(Q),\\ \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\min\Bigg(&\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\mathrm{old}}}(o_i \mid q)}\hat{A}_i,\\ &\mathrm{clip}\Bigg(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\mathrm{old}}}(o_i \mid q)}, 1-\epsilon, 1+\epsilon\Bigg)\hat{A}_i\Bigg)\\ &-\beta D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\Bigg]. \end{aligned} \tag{9}$$

The advantage function $\hat{A}_i$ depends on the standardization of intra-group rewards:

$$\hat{A}_i = \frac{r_i - \mathrm{mean}(r)}{\mathrm{std}(r)}, \tag{10}$$

To accurately penalize non-standard output frameworks, this paper designs a fine-grained hybrid reward function:

$$r_i = R_{\mathrm{content}} + R_{\mathrm{format}}, \tag{11}$$

The content reward R_{content} is computed based on the overlap between the predicted schema element set S_{pred} and the ground-truth set S_{true}. Specifically, the reward assigns a true-positive weight of $\lambda_{\mathrm{TP}} = 2.0$, a false-positive penalty of $\lambda_{\mathrm{FP}} = -0.5$, a false-negative penalty of $\lambda_{\mathrm{FN}} = -3.0$, and an F1 score bonus of $\lambda_{\mathrm{F1}} = 0.5$:

$$\begin{aligned} R_{\mathrm{content}} = \lambda_{\mathrm{TP}}\left|S_{\mathrm{pred}} \cap S_{\mathrm{true}}\right| + \lambda_{\mathrm{FP}}\left|S_{\mathrm{pred}} \setminus S_{\mathrm{true}}\right|\\ + \lambda_{\mathrm{FN}}\left|S_{\mathrm{true}} \setminus S_{\mathrm{pred}}\right| + \lambda_{\mathrm{F1}}\left(\frac{2PR}{P+R}\right), \end{aligned} \tag{12}$$

A.2.2 The Retriever Encoding and Contrastive Objectives

This section details the schema-aware representation learning and the contrastive objectives of the retriever, with the full architecture illustrated in Figure 6.

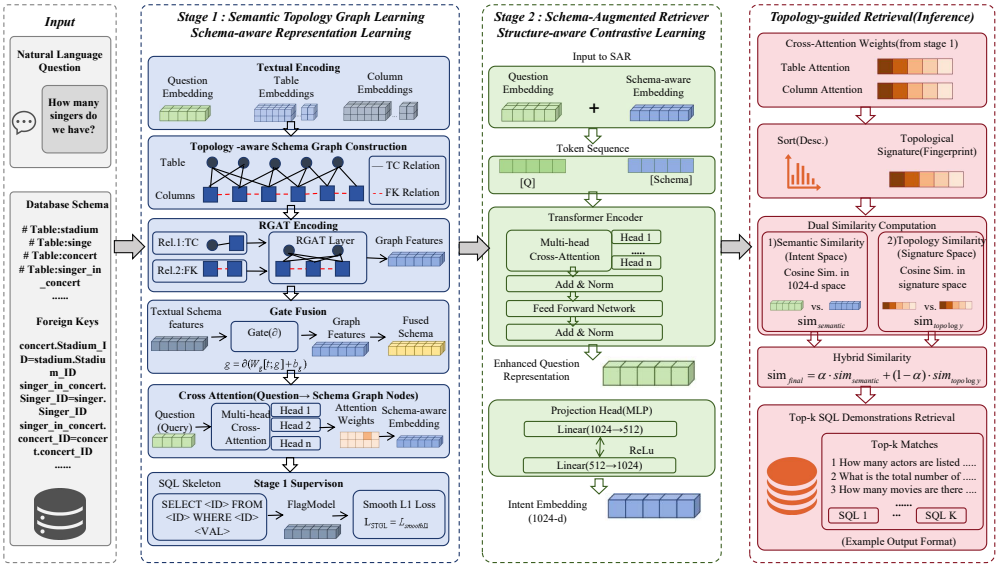


Figure 6: The dual-phase representation learning and generative-contrastive architecture of the Semantic Topology Graph Learning Schema-Augmented Retriever.

Phase I: Explicit Physical Topology Modeling via RGAT. To enable the model to perceive the rigid physical layout of the database from the ground up, the retriever first parses the database schema into a directed heterogeneous graph $G = (V, \mathcal{E})$. The node set $V = V_T \cup V_C$ consists of table nodes V_T and column nodes V_C, while the edge set $\mathcal{E} = \mathcal{E}_{TC} \cup \mathcal{E}_{FK}$ comprises table-column association edges ($\mathcal{E}_{TC}$) and primary key–foreign key relationship edges ($\mathcal{E}_{FK}$).

Given the initial text node embeddings $X \in \mathbb{R}^{|V| \times d}$ extracted by a pre-trained language model (PLM), in the RGAT (Relational Graph Attention Network) layer, for different relations $r \in \{\mathrm{TC}, \mathrm{FK}\}$, the transformation and message passing process of node features is defined as (Wang et al., 2020):

$$h_i^{(r)} = W_r x_i, \tag{13}$$

where $W_r \in \mathbb{R}^{d \times d}$ is the transformation matrix specific to relation r. The attention weight $\alpha_{i,j}^{(r)}$ of node i to its neighbor node j is calculated as follows:

$$\alpha_{i,j}^{(r)} = \frac{\exp\left(\mathrm{LeakyReLU}\left((a_r)^\top \left[h_i^{(r)} \parallel h_j^{(r)}\right]\right)\right)}{\sum_{k \in N_r(i)} \exp\left(\mathrm{LeakyReLU}\left((a_r)^\top \left[h_i^{(r)} \parallel h_k^{(r)}\right]\right)\right)}, \tag{14}$$

After aggregating messages of all relation types, the graph node features $\tilde{x}_i$ containing the physical structure are obtained through the ELU activation function:

$$\tilde{x}_i = \mathrm{ELU}\left(\sum_{r \in \{\mathrm{TC},\mathrm{FK}\}} \sum_{j \in N_r(i)} \alpha_{i,j}^{(r)} h_j^{(r)}\right), \tag{15}$$

Dynamic Gated Fusion and Cross-Attention. To avoid feature smoothing effects during graph convolution, a dynamic gating mechanism is introduced to calculate the fusion rate λ_i, decoupling and fusing the original semantic features with the physical topological features:

$$\lambda_i = \sigma\left(W_{\mathrm{gate}}\left[x_i \parallel \tilde{x}_i\right] + b_{\mathrm{gate}}\right), \tag{16}$$

$$x_i^{\mathrm{fused}} = \lambda_i \odot x_i + (1 - \lambda_i) \odot \tilde{x}_i, \tag{17}$$

Subsequently, the natural language question vector $Q \in \mathbb{R}^{1 \times d}$ is used as the Query to initiate multi-head cross-attention computation on the fused graph node matrix X^{fused}, resulting in the final representation z^{stage1} that includes pattern awareness.

Phase II: SQL Skeletonization and Contrastive Objectives. The core task of Phase Two is to sever the model's dependence on specific data domains. In this phase, the actual target SQL is forcibly transformed into a generalized abstract syntax tree (AST) skeleton template. The specific stripping rules are as follows. Use a regularization mask to replace all string text (e.g., `' .*? '`) and specific numeric constants (e.g., `\b\d+\b`) with the uniform tag `<VAL>`. Replace all non-SQL algebraic keywords (e.g., specific table names and column name identifiers) with the tag `<ID>`, and merge consecutively repeated `<ID>` tags. The resulting skeleton retains only relational algebra operations such as `SELECT`, `JOIN`, and `GROUP BY`.

The pattern-aware vector z^{stage1} output from Stage 1 is concatenated with the question vector Q and input into a causal Transformer encoder. This concatenation is then mapped to the intent latent space via an MLP projector, resulting in an enhanced intent representation z. Within this feature space, InfoNCE (van den Oord et al., 2018) is used for instance-discriminatory level contrastive learning:

$$\mathcal{L}_{\text{CL}} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{\exp\big(\text{sim}(z_i, z_i^+)/\tau\big)}{\sum_{j=1}^{B}\exp(\text{sim}(z_i, z_j)/\tau)}, \tag{18}$$

where $\text{sim}(\cdot,\cdot)$ is the cosine similarity, B is the batch size, and τ is the temperature hyperparameter. To further align similar syntactic queries, an auxiliary local similarity loss is introduced:

$$\mathcal{L}_{\text{sim}} = \frac{1}{BK}\sum_{i=1}^{B}\sum_{k=1}^{K}\max\left(0,\, 1-\text{sim}\left(z_i, z_{i,k}^{\text{sim}}\right)\right), \tag{19}$$

which forces the representation space to cluster into an abstract topological reasoning structure (the combined objective $\mathcal{L}_{\text{stage2}}$ is defined in Eq. 1).

A.2.3 The Refinement Agent Sandbox Implementation

This section details the concurrency anti-freeze sandbox of the refinement agent summarized in Section 3.4.1. To prevent the system from crashing due to malicious or computationally explosive queries (e.g., accidental cross-joins causing massive Cartesian products), the refinement agent implements a concurrency anti-Freeze sandbox built on a lightweight multi-threading architecture. Each candidate query is dispatched to a worker thread holding a dedicated SQLite connection, and a monitor thread tracks elapsed execution time. Once the query exceeds the strict temporal threshold of 10 seconds, the monitor invokes the native SQLite interrupt signal (`sqlite3.interrupt()`), which physically terminates the statement at the C layer of SQLite rather than at the Python wrapper. The sandbox then forcefully halts the worker, reclaims the connection, and returns a dedicated `EXECUTION_TIMEOUT` error stack to the LLM in the same format as ordinary runtime errors, so that the agent can reason about timeouts within the same ReAct loop. This design effectively prevents a single explosive query from blocking the sandbox or triggering cascading system failures.

A.2.4 Inference Algorithm and Complexity Analysis

We analyze Algorithm 1 with the following notation: C_{LLM} denotes the cost of one LLM call (prompt encoding plus decoding), C_{Exec} the cost of one sandbox execution, P the number of literal predicates extracted from the WHERE/HAVING clauses ($P = 0$ for `Error` states, where probing is skipped), $N_{\max}$ the largest number of rows scanned in a probed column, $M_{\max}$ the largest number of distinct values held in memory during fuzzy matching, and L the average string length of a literal value. Recall that $T_{\max} = 3$ is a small constant.

Time complexity. The pipeline issues exactly one LLM call for the initial generation, at most $T_{\max}$ sandbox executions (one per refinement agent turn), and at most $T_{\max}$ repair LLM calls; each `Empty`-turn additionally performs AST parsing and, per predicate, one `SELECT DISTINCT` scan plus a difflib fuzzy match against $M_{\max}$ candidates, costing $O(M_{\max}L)$ per predicate. Hence

$$T_{\text{total}} = O\Big((1{+}T_{\max})\,C_{\text{LLM}} + T_{\max}\,C_{\text{Exec}} + T_{\max}\,P\,(N_{\max} + M_{\max}L)\Big), \tag{20}$$

which, since $T_{\max}$ is constant, is linear in the number of LLM calls, sandbox runs, and probed column size. Treating the value probe as part of the per-turn sandbox overhead, this refines the coarse bound $O\big(K\cdot(C_{\text{LLM}}+C_{\text{Exec}})\big)$ of Sec. 3.4.5 while preserving its conclusion: no exponential blow-up occurs, and queries that execute successfully incur only a single LLM call and no probing at all.

Algorithm 1 The overall algorithm workflow of our framework

Require: Q, $\mathcal{D}$, L (schema links by schema linker, offline), $\mathcal{E}_k$ (retrieved examples by retriever, offline)
Ensure: final SQL S

$P \leftarrow \mathrm{PromptBuilder}(\mathcal{D}, L, \mathrm{Top}_3(\mathcal{E}_k), Q)$
$q \leftarrow \mathrm{LLM}(P)$
$S \leftarrow \textsc{Agent}(Q, q, \mathcal{D})$
return S
procedure $\textsc{Agent}(Q, q, \mathcal{D})$
  for $t = 1$ to $T_{\max}$ **do**
    $(s, r) \leftarrow \mathrm{SandboxExecute}(q, \mathcal{D})$ ▷ 10s timeout
    if $s = \mathrm{Success}$ **then break**
    end if
    if $s = \mathrm{Empty}$ **then**
      $F \leftarrow \mathrm{ValueProbe}(q, \mathcal{D})$ ▷ AST + fuzzy match
    else
      $F \leftarrow r$ ▷ error trace
    end if
    $q' \leftarrow \mathrm{LLM}(\mathrm{RepairPrompt}(Q, \mathcal{D}, q, F))$
    if $q' = \emptyset$ **or** $q' = q$ **then break**
    end if
    $q \leftarrow q'$
  end for
  return q
end procedure

Space complexity. The persistent context consists of the schema string $\mathcal{S}$ and the top-3 retrieved examples, requiring $O\big(|\mathcal{S}| + k\,(|Q| + |\mathrm{SQL}|)\big)$ with $k \leq 3$. During each refinement agent turn the transient storage is dominated by the sandbox result buffer and the distinct-value list of the probed column, $O(R_{\max} + M_{\max})$, where $R_{\max}$ is the number of returned rows (status evaluation itself needs only $O(1)$ state: the current SQL and the last execution state). Across turns only one SQL string is carried over, so

$$S_{\mathrm{total}} = O\Big(|\mathcal{S}| + k\,(|Q| + |\mathrm{SQL}|) + R_{\max} + M_{\max}\Big), \tag{21}$$

i.e., memory grows linearly in the input and database column sizes and is independent of the number of refinement turns. In practice $M_{\max}$ is bounded by the `SELECT DISTINCT` result (with cutoff-based pruning), and $R_{\max}$ by the timeout-guarded execution, keeping the per-query memory footprint small and constant across the $T_{\max}$ turns.

A.3 Experiment Setup

In this section, we present comprehensive details regarding datasets, metrics, baseline configurations, and implementation parameters to support reproducible research.

A.3.1 Datasets

The core experiments of this study were conducted on five authoritative benchmark datasets: Spider (Yu et al., 2018), BIRD (Li et al., 2023b), Spider 2.0 snow (Lei et al., 2025), Beaver (Chen et al., 2026), and UniSQL, selected to ensure robustgeneralization across diverse query types and database structures.While datasets like Spider emphasize complex cross-domain semantic parsing, BIRD focuses on large-scale database grounded evaluation in big data environments, Spider 2.0 snow highlights real-world enterprise workflows in cloud systems, and Beaver concentrates on private enterprise data warehouses with intricate schemas and business-oriented queries, UniSQL is specifically utilized as a comprehensive generalization benchmark. It is constructed via stratified sampling across four datasets, and is carefully balanced to ensure diversity in query and database complexity, providing a robust evauation of generalization across various SQL query types and database structures.

A.3.2 Evaluation Metrics

Evaluation Metrics. Evaluation metrics center on Execution Accuracy (EX), which measures semantic correctness by comparing the execution results of predicted and ground-truth SQL across all datasets. For the Spider dataset, we report Exact Match (EM), requiring strict string-level consistency to assess syntactic accuracy. For BIRD, Beaver, UniSQL, and Spider 2.0 snow, we rely primarily on EX, as their large-scale databases minimize false positives and ensure reliable evaluation. The Valid Efficiency Score (VES) is also used for BIRD to assess efficiency and validity of generated queries.

A.3.3 Baselines

This study adopts a range of foundation models,including open-source models such as CodeS (Li et al., 2024) (PACMMOD 2024),XiYanSQL (Gao et al., 2025) (Alibaba 2025), Qwen Qwen et al. (2025) (Alibaba 2025), and Llama (Grattafiori et al., 2024) (Meta AI 2024), as well as powerful proprietary models like GPT-4o-mini and GPT-4o (OpenAI et al., 2024) (OpenAI 2024), DeepSeek-V3 (DeepSeek-AI et al., 2025) (DeepSeek2024) and GLM-4 (GLM et al., 2024) (Zhipu AI 2024). These models are comparedagainst competitive Text-to-SQL methods, including TA-SQL(ACL 2024), Din-SQL (NeurIPS 2023), ACT-SQL (EMNLP 2023), C3-SQL (arXiv 2023), CHASE-SQL and SchemaRAG.

A.3.4 Implementation Details

Our experiments were conducted on a server equipped with a single NVIDIA RTX PRO 6000 GPU (96GB), 110GB of RAM, and an Intel(R) Xeon(R) Platinum 8470Q CPU (22 vCPUs). We utilized SQLite as the underlying database management system and the primary external execution sandbox. During the contextual grounding phase, our framework employs the Semantic Topology Graph Learning Schema-Augmented Retriever rather than a standard dense retriever. For cross-domain example retrieval, the dual-similarity fusion mechanism is configured with an intent space weight of $\alpha = 0.4$, allocating a dominant 60% weight to the physical topological graph fingerprints. For the SQL generation and verification phase, instead of generating multiple static candidates for Pareto selection, we deploy the Execution-Feedback Refinement Agent. The generated SQL queries are dynamically evaluated in a concurrency anti-freeze sandbox with a strict execution timeout threshold set to 10 seconds. Within the refinement agent module, sqlglot is utilized for Abstract Syntax Tree (AST) parsing of conditional predicates, and Python's difflib is employed for fuzzy value string matching. To optimize inference efficiency while preventing infinite feedback loops, the maximum number of adaptive refinement iterations is strictly bounded to $\mathrm{T}_{\max} = 3$.

A.4 Experiment

This supplementary document provides complete experimental setups, supplementary evaluations for RQ1 and RQ2, and the full empirical investigations for RQ3, RQ4, and RQ5 as referenced in Section 4

A.4.1 Performance Comparison (RQ1 Extended Analysis)

Complementing the main benchmark results in Section 4.1, this section provides: (1) Statistical significance testing including Loss/Win tallies, Wilcoxon signed-rank tests, and Friedman ranking tests; and (2) Fine-grained performance breakdown across difficulty strata (Simple, Moderate, Challenging) on the BIRD dataset.

(1) a Loss/Win analysis, counting cases where RSTAR outperforms baselines across EX, EM, and VES metrics; (2) the Wilcoxon signed-rank test to assess pairwise statistical significance; and (3) the Friedman test for overall model ranking, where a lower F-rank indicates better performance.

RSTAR achieves substantial gains over all baseline models (Table 1), with statistical validation confirming the significance of these improvements (Table 3). It attains a dominant win/loss ratio of 210/0, with all pairwise comparisons showing p-values below 0.05 under the Wilcoxon signed-rank test. On the UniSQL dataset, RSTAR(GPT-4o), trained with a schema linker based solely on BIRD-based contrastive learning data, achieves an EX accuracy of 50.96%, surpassing the best baseline

Table 3: Loss/Win counts, Wilcoxon signed-rank test, and the Friedman test among models

Model	loss/win	p-value	F-rank
TA-SQL	35/0	5.82×10^{-11}	4.84
DIN-SQL	35/0	5.82×10^{-11}	5.20
ACT-SQL	35/0	5.82×10^{-11}	5.80
C3-SQL	35/0	5.82×10^{-11}	5.60
Chase-SQL	35/0	5.82×10^{-11}	3.24
SchemaRAG	35/0	5.82×10^{-11}	2.31
RSTAR	0/210	-	1.00

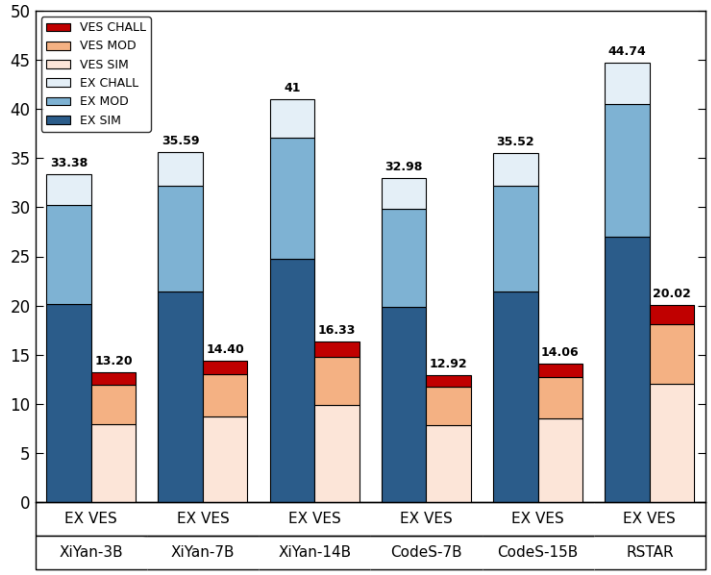


Figure 7: A comparison with fine-tuned large models on the BIRD dataset without evidence.

SchemaRAG (48.05%). These results highlight RSTAR's strong robustness and generalization in complex, cross-domain Text-to-SQL tasks.

Figure 7shows the performance of various models, including the XiYanSQL series, CodeS series, and RSTAR (GLM-4-Plus), on the BIRD dataset without evidence, evaluated using both EX and VES metrics. We compare model performance across three difficulty levels—Simple (SIM), Moderate (MOD), and Challenging (CHALL)—under both metrics. RSTAR consistently achieves the highest scores across all categories: for EX, 51.41 (SIM), 33.55 (MOD), 37.93 (CHALL), and 44.74 (TOTAL); for VES, 21.32 (SIM), 18.60 (MOD), 16.21 (CHALL), and 20.02 (TOTAL). XiYanSQL-14B ranks second under the EX metric.

In summary, RSTAR(GLM-4-Plus) demonstrates superior performance overall and across all difficulty levels, significantly outperforming the fine-tuned models.

A.4.2 Ablation Experiments (RQ2 Extended Analysis)

Complementing the primary ablation findings in Section 4.2, this section reports supplementary statistical significance tests and in-depth discussion for our main ablation results.

Table 4: Loss/Win counts, Wilcoxon signed-rank test, and the Friedman test among models

Model	loss/win	p-value	F-rank
w/o schema linker	33/1	4.37×10^{-7}	2.33
w/o retriever	35/0	5.82×10^{-11}	4.00
w/o refinement agent	34/1	6.40×10^{-9}	2.70
RSTAR	2/102	-	1.07

The quantitative results are detailed in Table 2, and the corresponding statistical significance tests (Wilcoxon signed-rank and Friedman tests) are summarized in Table 4. The analysis reveals the following critical insights:

The Dominant Impact of retriever: The retrieval module constitutes the most critical component of the architecture. Removing retriever induces the most severe and consistent performance collapse across all evaluated LLMs and datasets. For example, the execution accuracy (EX) of Deepseek-v3 on the highly complex BIRD dataset drops precipitously from 79.20% to 62.58%. Furthermore, the exact match (EM) score of Llama-8B on Spider plunges from 70.80% to 29.90%. This catastrophic decline confirms that relying on standard semantic retrieval without explicit topological graph learning fails to provide structurally isomorphic examples, leading to severe downstream syntactic hallucinations. Statistical validation in Table 4 cements this finding, showing that w/o retriever records 35 losses against 0 wins, yielding a p-value of 5.82×10^{-11} and the worst Friedman rank of 4.00.

The Necessity of Execution Feedback: Unlike conventional open-loop generation pipelines, the removal of the refinement agent results in a substantial performance drop, validating the theoretical concern regarding "execution-level semantic blind spots." Without the multi-turn sandbox interaction, models fail to correct latent value-mismatches and logical operator errors. This leads to degraded execution accuracy on stringent benchmarks like Spider 2.0 and UniSQL. While a marginal EM score anomaly was observed for XiYanSQL-14B on Spider—where the ablated model (94.70%) slightly outperformed the full framework (93.70%)—this represents a surface-level syntax match artifact. It did not translate to superior functional correctness, as the corresponding EX score still dropped from 95.40% to 95.30%. The w/o refinement agent configuration ultimately registered a high Friedman rank of 2.70.

The Role of the schema linker: Eliminating the explicit schema linker also leads to consistent systemic degradation (Friedman rank 2.33, 33 losses), underscoring its necessity in constraining the initial topological search space. Without schema linker, models struggle with accurate schema routing in massive databases. A rare exception occurred with GPT-4o on the Beaver dataset, where the ablated model marginally outperformed the full framework (8.28% vs. 8.13%). This suggests that for certain specific schemas, the exceptionally robust parametric knowledge of a frontier model like GPT-4o might occasionally bypass the need for explicit graph routing. However, this isolated anomaly does not offset the widespread deterioration across all other metrics and models.

Ultimately, the complete framework achieves overwhelming superiority, recording 102 wins against merely 2 losses and attaining the optimal Friedman rank of 1.07. The Wilcoxon signed-rank test confirms that removing any single module results in a statistically significant performance decline (p ¡ 0.05). This definitively proves that the competitive performance of the model relies on the tight, synergistic integration of topological path navigation, structure-aware contrastive retrieval, and dynamic execution feedback.

A.4.3 Error Analysis(RQ.3)

To better understand its behavior and identify remaining challenges, we now present a detailed error analysis based on its implementation on GPT-4o. This analysis examines a random sample of 300 incorrect predictions generated by the model on the UniSQL dataset.

As illustrated in Figure 8, although our schema linker mitigates schema-level issues and the retriever for SQL-aware question embeddings addresses higher-level semantic and logical reasoning challenges, the persistence of errors across schema linking, semantic understanding, and logical

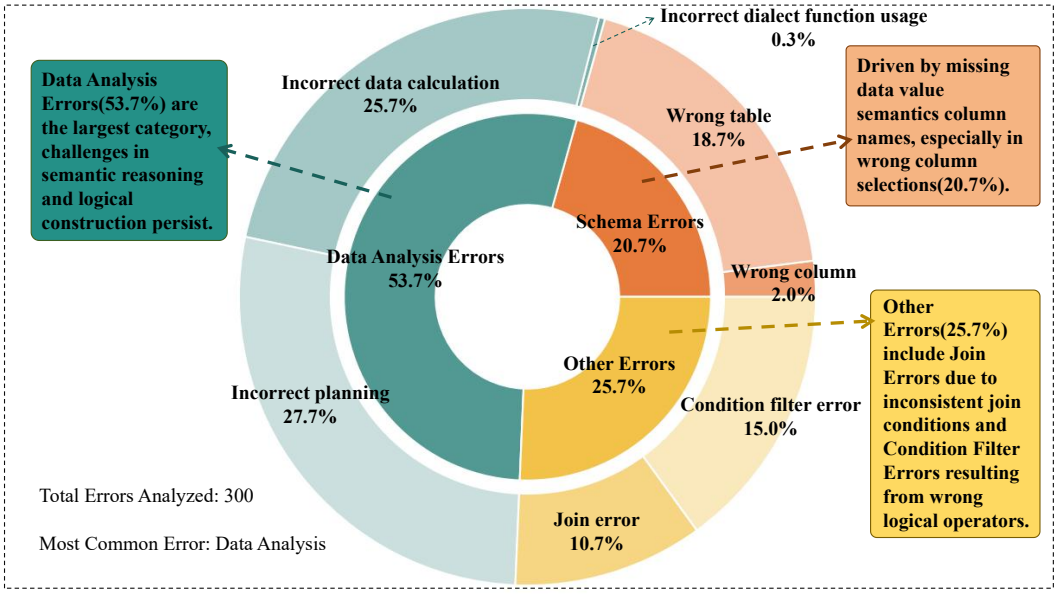


Figure 8: Error statistics.

query construction still highlights the limitations of our approach in comprehending large, ambiguous databases.

A.4.4 Self-Consistency Effects on Schema Linker and RSTAR Efficiency(RQ.4)

Impact of Self-Consistency on Schema Linking in Schema Linker To investigate the specific impact of the self-consistency strategy on the schema linker, we conducted independent evaluations on a challenging subset of 300 samples extracted from the BIRD dataset. This subset comprises queries where the model failed to predict key schema items (e.g., necessary table joins or columns) without the self-consistency strategy enabled. For each query, the schema linker model was executed k times, and the final schema linking result was derived by taking the union of the outputs generated across these k iterations. The experimental results are illustrated in Figure 9, with the following detailed observations:

Overall Performance Improvement Trend: As the number of self-consistency runs, k, increases, all four core evaluation metrics (Table F1, Table Recall, Column MCC, and Column Recall) exhibit a significant upward trend. This indicates that taking the union of multiple independently generated topological reasoning paths effectively compensates for topological breaks or entity omissions that occur in a single inference run, thereby substantially enhancing the schema linking performance of schema linker.

Trade-off Between Marginal Benefits and Computational Costs: The performance metrics experience the most dramatic improvement as k increases from 1 to 4. When k reaches 4 to 5, the growth rates of Table F1 (stabilizing around 82%) and Column MCC (stabilizing around 70%) decelerate significantly and begin to plateau. Although the recall metrics (Table Recall and Column Recall) continue to rise slowly for $k > 5$, the comprehensive accuracy indicators have essentially hit a ceiling. This phenomenon suggests that in practical deployments, blindly increasing the value of k (e.g., $k > 5$) yields limited marginal gains, necessitating a rational trade-off between inference accuracy and computational overhead.

Significant Optimization in Column-Level Feature Extraction: Among all the metrics, the absolute improvement in Column Recall is the most striking (surging from approximately 48% at k=1 to nearly 79% at k=10). Because column selection in complex enterprise-scale databases involves a vastly larger search space and lower initial accuracy, the union mechanism of the self-consistency strategy perfectly demonstrates its advantage. Multi-path exploration effectively captures fine-

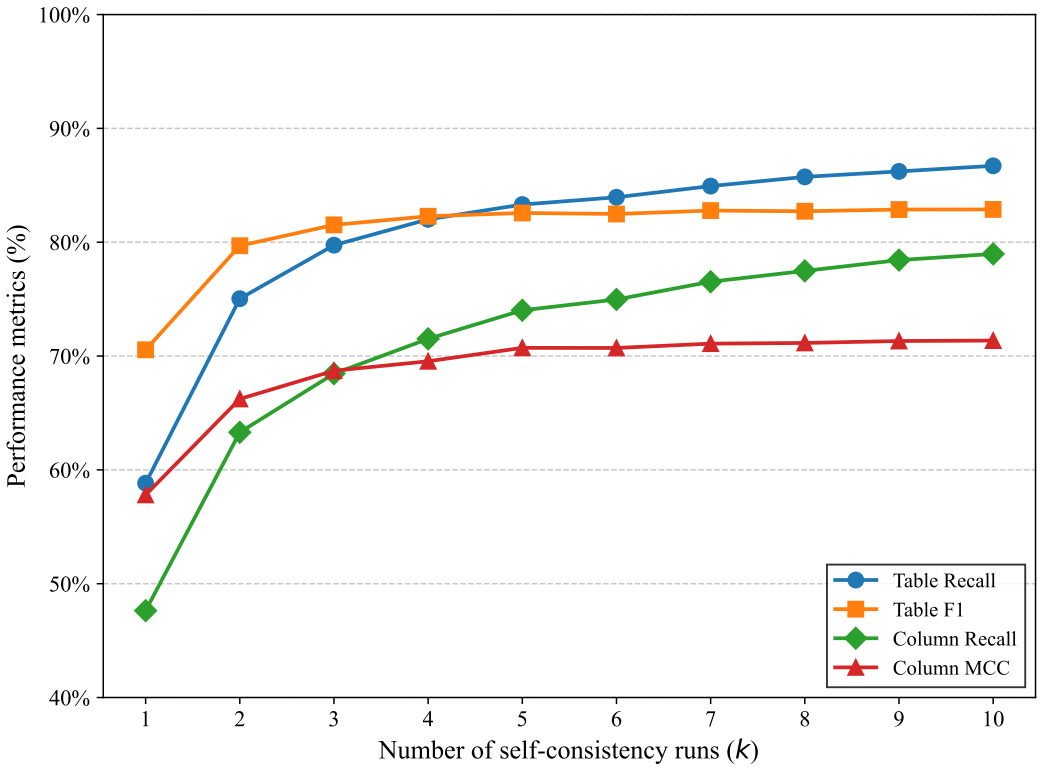


Figure 9: Performance metrics (Table F1, Table Recall, Column MCC, and Column Recall) as a function of the number of self-consistency runs (k).

grained attribute nodes that might be missed in a single schema linker generation, providing a highly robust recall guarantee for the downstream generation modules.

The experimental results confirm that the self-consistency method can significantly enhance the feature extraction capability and robustness of the schema linker model when processing highly difficult samples with complex topologies.

Table 5: Maintainability and Efficiency analysis of different components on the UniSQL benchmark. Green cells indicate components with significantly increased context size or latency

Configuration	**Context Size (tok/req)** $L + \sigma$	**Avg Latency (s/req)** $T + \sigma$	**P95 Latency (s/req)** $T + \sigma$
GPT	151 (-)	3.44 (-)	6.38 (-)
+Schema Linker	—	—	—
+PromptSchema	1253 (↑ 1102)	4.24 (↑ 0.8)	9.24 (↑ 2.86)
+fine-tuned	3057 (↑ 1804)	4.49 (↑ 0.25)	12.79 (↑ 3.55)
+self-consistency(5)	12778 (↑ 9721)	12.95 (↑ 8.46)	29.30 (↑ 16.51)
+retriever	12856 (↑ 78)	12.99 (↑ 0.04)	30.82 (↑ 1.52)
+refinement agent	16025 (↑ 3169)	27.99 (↑ 15.0)	103.20 (↑ 72.38)

Maintainability and Efficiency Analysis of RSTAR Our efficiency and maintainability analysis on the UniSQL benchmark, evaluated through incremental modular additions to the base LLM,

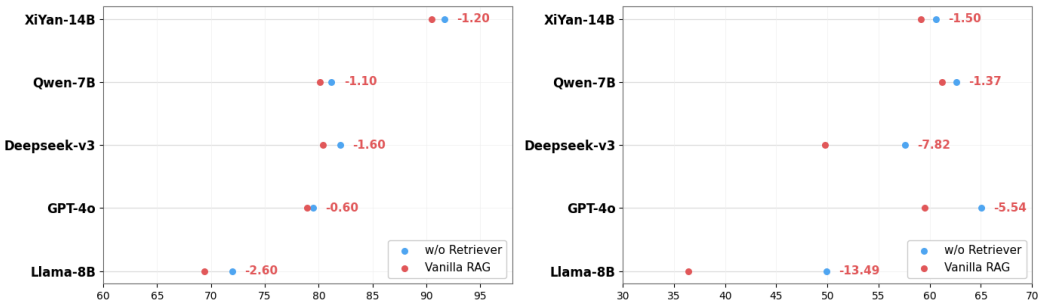


Figure 10: Performance comparison of RSTAR without retrieval augmentation versus Vanilla RAG on Spider (left) and BIRD (right).

demonstrates that the proposed framework effectively balances complex reasoning capabilities with computational constraints,as summarized in Table 5. Early components in the pipeline, such as PromptSchema and the fine-tuned schema linker, introduce a moderate increase in context size and a slight latency overhead. However, these foundational modules provide the essential semantic and structural grounding required for precise topological schema linking.

The self-consistency mechanism emerges as a primary computational bottleneck, leading to a substantial expansion in context scale and a noticeable rise in average latency due to its multi-path decoding nature. Nevertheless, its highly pluggable design allows it to be dynamically adjusted, offering an effective trade-off between inference speed and accuracy for latency-sensitive scenarios.

In contrast, the retriever module exhibits exceptional inference efficiency. By leveraging cross-domain topological fingerprint retrieval, it extends the model's structural awareness with almost negligible increases in both context length and processing time, highlighting the lightweight nature of the contrastive learning representations during the inference phase.

The Execution-Feedback Refinement Agent acts as the most significant source of latency within the framework, particularly driving up long-tail latency (P95) and token consumption. This overhead is inherently tied to the multi-turn ReAct diagnostic loops and direct interactions with the external database sandbox. Importantly, because the refinement agent functions as a conditionally triggered post-generation gateway—activating primarily upon syntax failures or empty-set returns—it isolates this heavy computational cost from the standard generation flow. Overall, this non-monolithic, modular architecture ensures that components can be independently tuned or updated, isolating maintenance costs and securing robust execution-level semantics without permanently compromising overall system efficiency.

A.4.5 The Retriever Evaluation(RQ.5)

Analysis of Vanilla RAG As illustrated in Figure 10, to rigorously investigate the impact of conventional retrieval mechanisms, we compared the performance of our framework operating without any retrieval augmentation against variants employing Vanilla RAAs illustrated in Figure 10, tated in the performance comparison, incorporating Vanilla RAG consistently yields counterproductive effects, resulting in execution accuracy (EX) degradation across all evaluated Large Language Models (LLMs) compared to the non-retrieval baseline. This negative impact is particularly pronounced in models such as Llama-8B (a 2.60 drop) and Deepseek-v3 (a 1.60 drop).The degradation is even more pronounced on BIRD (right), where Vanilla RAG costs up to 13.5 EX points (Llama-8B), indicating that retrieval mismatch becomes increasingly harmful as schema complexity grows.

The underlying cause of this performance degradation is that Vanilla RAG methods depend heavily on surface-level semantic similarities and lexical overlap. Consequently, they frequently retrieve demonstration examples that share similar domain vocabularies but possess fundamentally different SQL syntactic structures and multi-table topological routing. LLMs are highly susceptible to contextual confusion and structural hallucination when presented with these misleading exemplars. While advanced proprietary models like GPT-4o exhibit slightly better noise-filtering

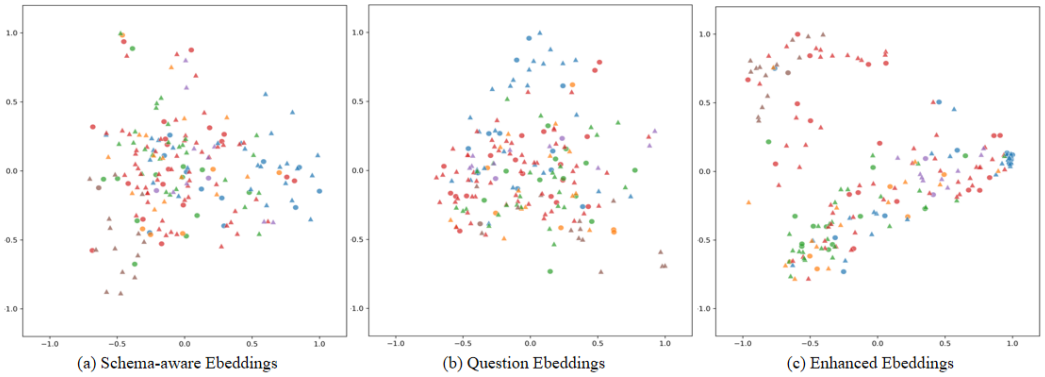


Figure 11: Visualization of different embedding spaces via Principal Component Analysis (PCA).Points of the same colorrepresent pairs with similar SQL syntactic structures. Shapes distinguish between input queries (circles) and candidate examples from our RAG datasets (triangles).

capabilities—experiencing a marginal 0.60 drop—they still fail to derive any positive gain from conventional dense retrieval.

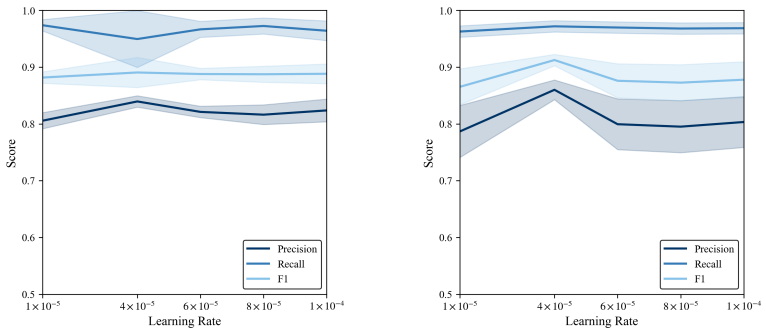
These empirical findings underscore the systemic limitations of relying solely on embedding similarity for Text-to-SQL tasks and highlight the critical necessity of our Semantic Topology Graph Learning Schema-Augmented Retriever. By explicitly disentangling underlying topological waveforms from database-specific lexical identifiers, the retriever effectively mitigates these retrieval mismatches and ensures rigorous syntactic alignment.

Analysis of Embedding Visualization As illustrated in Figure 11, the visualization confirms the profound impact of our contrastive learning module. The baseline question embeddings (Fig 11b) exhibit complete topological collapse, highlighting the inability of off-the-shelf PLMs to capture SQL syntactic boundaries. The schema-aware embeddings (Fig 11a) expand the feature variance but fail to disentangle the classes. In stark contrast, the enhanced embeddings (Fig 11c) successfully shatter this entanglement, pulling structurally isomorphic samples into distinct semantic regions.

Quantitatively, the Silhouette Score undergoes a critical phase transition from negative values in the baseline (-0.035) to positive separability (0.005) in the enhanced space. While preserving the natural intersection of complex SQL topologies (the overlapping core representing query sub-structures), the contrastive loss effectively extends distinct syntactic clusters outward. The precise geometric alignment between test queries (circles) and candidate examples (triangles) within these distinct regions robustly justifies the efficacy of the SAR retrieval mechanism.

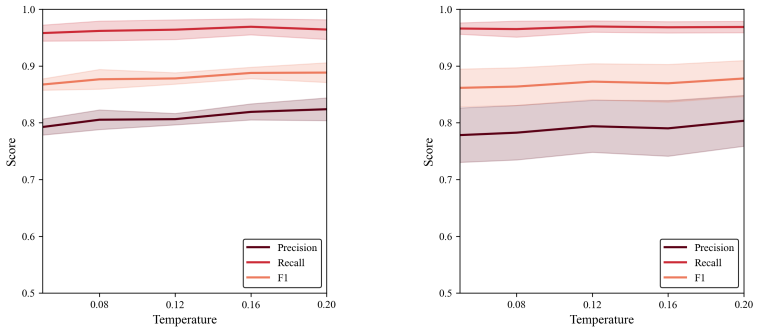
Hyperparameter Sensitivity Analysis As illustrated in Figure 12, we conducted a sensitivity analysis to determine the optimal learning rate and contrastive temperature using a 3-layer, 4-head Transformer baseline. As shown in Figure 12, a learning rate of 4×10^{-5} (with $T = 0.2$) achieved the best F_1-scores on RAG_Spider (0.8906) and RAG_BIRD (0.9125). Thus, we set LR $= 4 \times 10^{-5}$ and $T = 0.2$ for all subsequent experiments.

Impact of Model Architecture As illustrated in Figure 13, we further investigated the impact of the retriever module's architecture by varying the number of Transformer layers and attention heads. The evaluation metric heatmap reveals that the optimal architectural capacity is inherently dataset-dependent. For the RAG_Spider dataset, a leaner architecture configuration (2 layers, 2 heads) achieved the highest F1 score. Conversely, for the highly challenging RAG_BIRD dataset, which features massive enterprise-level databases and dense foreign-key topologies, a deeper and wider configuration (3 layers, 4 heads) yielded the optimal performance. This phenomenon indicates that explicitly disentangling topological waveforms in complex databases necessitates an expanded parameter space to accurately capture fine-grained cross-domain structural distinctions. However, scaling beyond these optimal configurations led to performance degradation across both



(a) The RAG_Spider score varies with learning rate.

(b) The RAG_BIRD score varies with learning rate.



(c) The RAG_Spider score varies with temperature.

(d) The RAG_BIRD score varies with temperature.

Figure 12: The performance of the retriever with different temperature parameters and learning rates on the RAG_Spider and RAG_BIRD datasets, averaged over 5 runs with mean and variance reported.

datasets, confirming that excessive depth or width induces overfitting and structural hallucination during contrastive alignment.

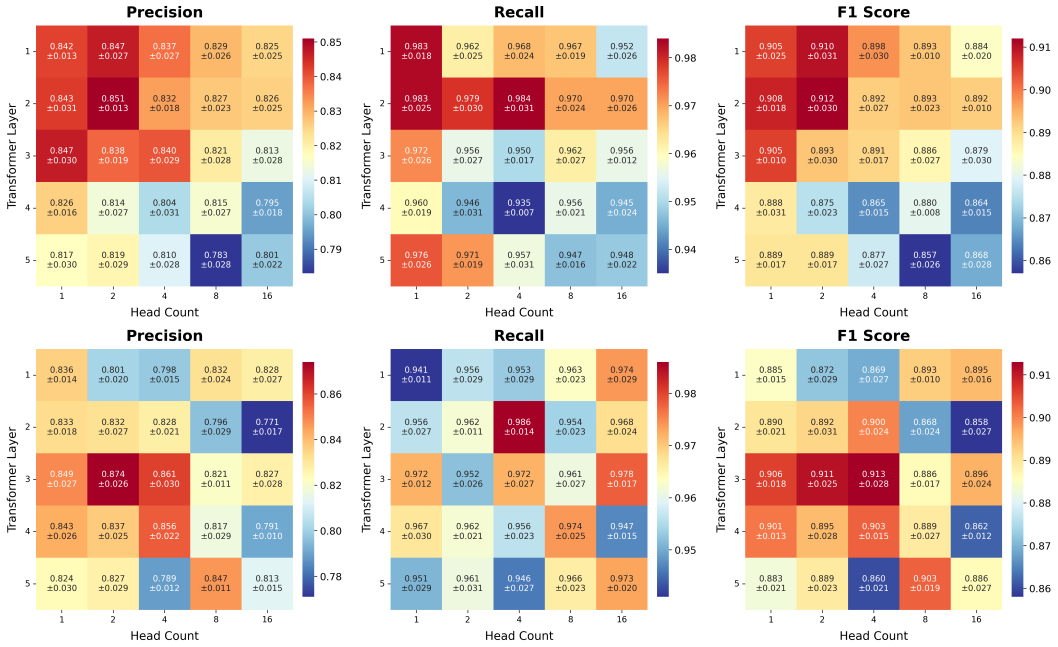


Figure 13: The performance of the retriever with different numbers of attention heads and Transformer layers on the RAG_Spider (top) and RAG_BIRD (bottom) datasets, averaged over 5 runs with mean and variance reported.