SARANK: STRUCTURE-AWARE DEMONSTRATION RERANKING FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SQL translates natural-language questions into executable SQL queries. Existing retrieval-augmented methods usually rank demonstrations by semantic or schema similarity. However, semantically similar questions may require different SQL operations, causing structurally relevant demonstrations to fall outside the limited Top-K context. We call this problem the demonstration exposure bottleneck. To address it, we propose SARank, a structure-aware demonstration reranking method. SARank first uses a schema-aware retriever to construct a high-recall candidate pool. It then abstracts SQL literals and learns question–SQL compatibility with a shared encoder trained by a candidate-aware contrastive objective using retrieval-based, same-database, and random negatives. The resulting scores reorder the candidates before prompt construction, without requiring a predicted target SQL or execution feedback. Experiments across Spider, BIRD, BEAVER, and Dr.Spider, including evaluations with multiple LLMs, show that SARank improves BestSim @3, Recall @3(0.8), and downstream execution accuracy. Candidate-budget analysis and case studies further show that SARank promotes lower-ranked but structurally aligned demonstrations into the Top-K, providing more effective guidance for SQL generation.

1 INTRODUCTION

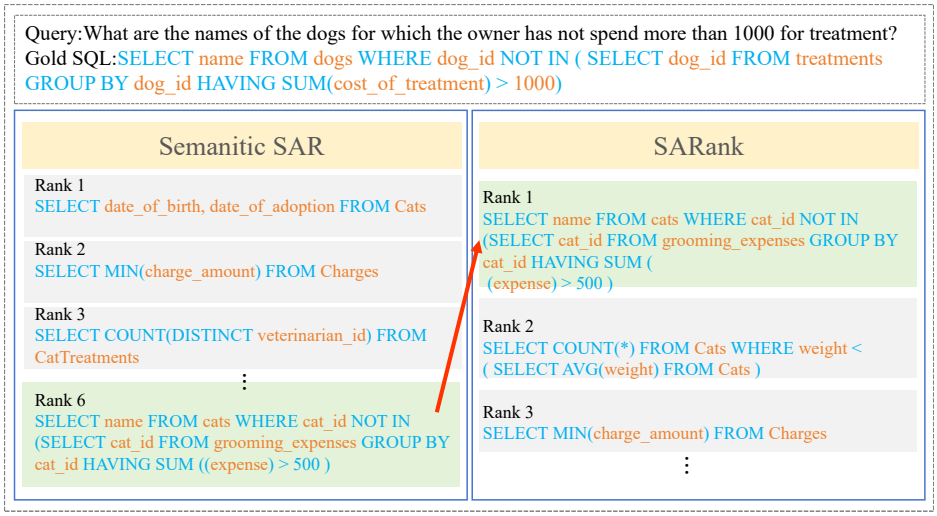


Figure 1: The demonstration exposure bottleneck: a structurally aligned example is already retrieved but falls below the Top-3 cut-off. SARank promotes it from rank 6 to rank 1.

Text-to-SQL translates natural-language questions into executable database queries. Large language models have improved this task through few-shot prompting and retrieval-augmented generation (Hong et al., 2024; Mohammadjafari et al., 2024; Gao et al., 2023). In these systems, demonstrations show how a question can be expressed in SQL. A useful demonstration should therefore be relevant to the question and illustrate the operations needed to answer it.

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.

Existing demonstration retrievers use semantic, schema, or SQL-structure information (Lee et al., 2025a; Shen et al., 2024; Wu et al., 2026; 2025). However, similar questions can require different joins, aggregations, ordering operations, or nested queries. A useful candidate may therefore be retrieved but ranked too low to enter a short prompt. We call this the *demonstration exposure bottleneck*: the candidate pool contains a relevant query pattern, but the final Top-K demonstrations do not expose it to the LLM. This motivates a focused question: how much can we improve demonstration selection by changing the order of a fixed candidate pool?

Figure 1 gives a concrete example. A demonstration with the required nested aggregation pattern appears at rank 6, outside the Top-3 prompt. SARank moves it to rank 1. A second case asks for the most frequent nationality (Figure 6). Examples from the same database appear relevant, but they do not show grouping and count-based ordering. Here, promoting a structurally aligned example from rank 10 to rank 1 is accompanied by a correct generated query. These cases show why retrieving an example and selecting it for the prompt are separate decisions.

We propose SARank, a structure-aware reranker placed between retrieval and generation. A schema-aware retriever first builds a candidate pool. SARank then masks literal values in each candidate SQL and uses a shared encoder to score its compatibility with the target question. The highest-scoring demonstrations enter the prompt. Because scoring uses the question and candidate SQL directly, it requires neither a predicted target SQL nor execution feedback.

To train for this setting, we combine three negative sources. Retrieved negatives resemble the candidates that the reranker must distinguish at inference. Same-database negatives encourage it to look beyond a shared schema, while random negatives provide a wider range of queries. Gold SQL provides the positive examples during training. At inference, only the target question and retrieved candidates are needed.

Our contributions are threefold. First, we identify the demonstration exposure bottleneck and study reranking within a fixed retrieved pool. Second, we propose SARank, which learns question–SQL compatibility using literal masking and mixed negatives without requiring a predicted target SQL or execution feedback. Third, controlled comparisons, ablations, and budget experiments show that these design choices improve structural selection and end-to-end accuracy over SchemaRAG across three LLMs on Spider and BIRD. We fix the retrieved pool to isolate ordering; improving candidate recall remains a separate task.

2 Related Work

Text-to-SQL pipelines. Systems such as CHESS (Talaei et al., 2024), RSL-SQL (Cao et al., 2024), and ADEPT-SQL (Chen et al., 2025) combine schema selection, retrieval, generation, or refinement. LitE-SQL (Piao et al., 2026) and ReSQL (Park et al., 2026) also study retrieval and error-driven refinement. These systems make several pipeline choices together. SARank focuses on one step: selecting demonstrations before SQL generation.

Demonstration retrieval and ranking. DCG-SQL (Lee et al., 2025a) and SchemaRAG (Wu et al., 2026) use schema-aware representations for retrieval, while ASTReS (Shen et al., 2024) and UCS-SQL (Wu et al., 2025) use SQL structures. ASTReS ranks examples using an approximated target SQL. SARank instead scores candidate SQL against the question itself, so it does not require that intermediate query. R^3-SQL (Han et al., 2026) addresses a different stage by ranking SQL answers after generation. We use the retriever from SchemaRAG and hold its candidate pool fixed when comparing ranking strategies. This isolates ranking from candidate recall, but does not establish an advantage over DCG-SQL or ASTReS. Appendix G provides further discussion.

3 Method

3.1 Candidate Retrieval and Overall Pipeline

Given a question q, schema Σ, and demonstration bank $\mathcal{D} = \{(q_i, \Sigma_i, y_i)\}$, we first retrieve M examples using the schema-aware retriever from SchemaRAG (Wu et al., 2026). Its question-conditioned schema representation is fused with the question embedding to produce $\mathbf{h}_q$; demon-

strations are encoded in the same way. The candidate set is

$$\mathcal{C}_M = \mathrm{TopM}_{d_i \in \mathcal{D}} \cos(\mathbf{h}_q, \mathbf{h}_i). \tag{1}$$

SARank preserves this pool and changes its order before selecting K demonstrations. Unless otherwise stated, $M = 10$ and $K = 3$. The inherited retrieval equations are provided in Appendix A, and Algorithm 1 summarizes the complete inference procedure.

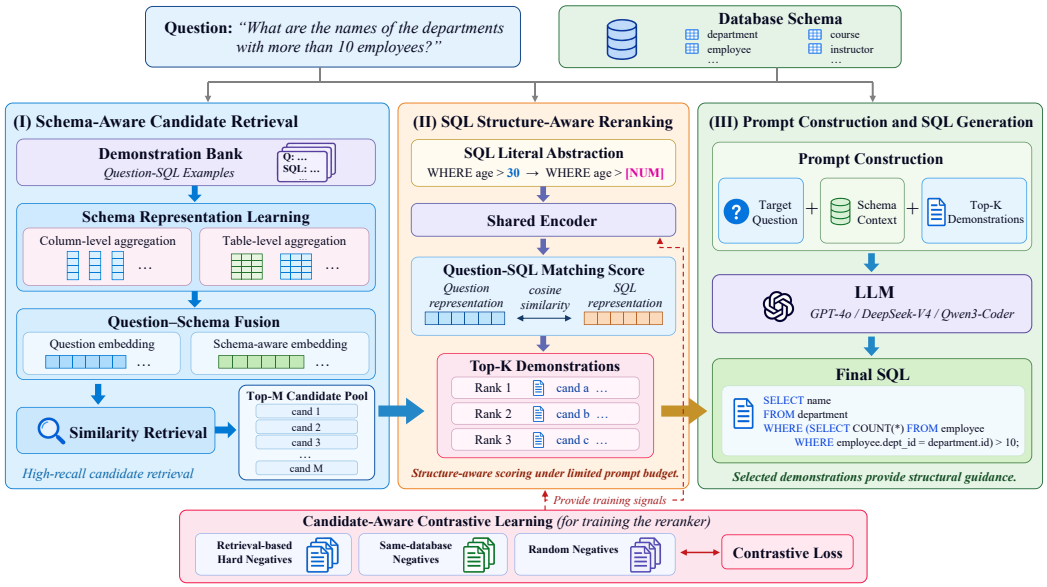


Figure 2: SARank's retrieve–rerank–generate framework. Schema-aware retrieval supplies the candidate pool; learned question–SQL compatibility selects the prompt demonstrations. Candidate-aware contrastive learning trains the reranker and is not an additional inference stage. Figure 3 details the shared encoder.

3.2 SQL Literal Abstraction and Compatibility Scoring

Literal values can make two queries appear similar because they mention the same value rather than use the same operations. We therefore replace SQL literals with placeholders, giving $\widetilde{y}_i = \mathrm{Mask}(y_i)$. We retain operators, clauses, nesting relationships, and table and column names, and leave the question unchanged. A shared encoder E maps the question and masked candidate SQL into the same space. We then score each candidate using cosine similarity:

$$r(q, y_i) = \frac{E(q)^\top E(\widetilde{y}_i)}{\|E(q)\|_2 \, \|E(\widetilde{y}_i)\|_2}. \tag{2}$$

Candidates are sorted by this score, and the Top-K examples are inserted into the generation prompt. This learned question–SQL score is distinct from the gold-SQL structural similarity used only for evaluation in Section 4.2.

The retriever uses the question and schema to find a relevant pool. The reranker then compares the question with each candidate SQL to decide which examples enter the prompt. Masking removes literal-specific information, but table and column names remain, so the representation still carries database information.

3.3 CANDIDATE-AWARE CONTRASTIVE LEARNING

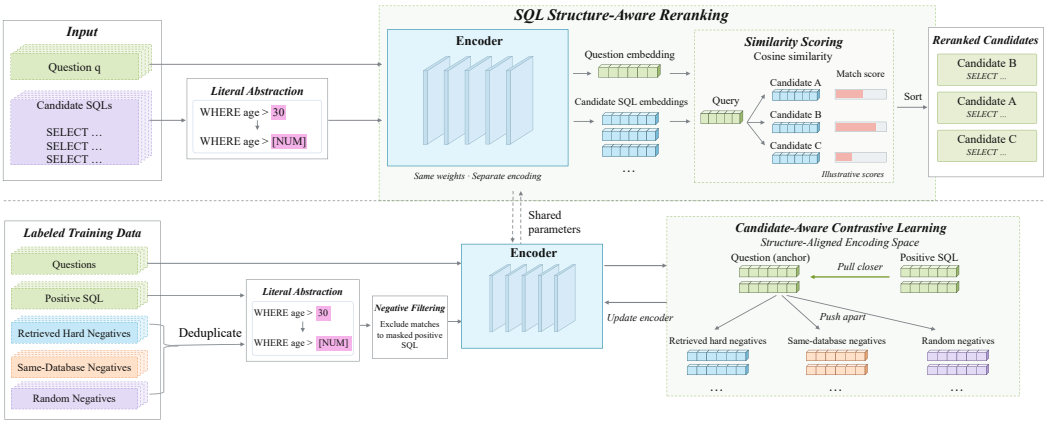


Figure 3: Architecture of the shared encoder for SQL structure-aware reranking and candidate-aware contrastive learning. The upper branch illustrates inference-time candidate reranking, while the lower branch shows mixed-negative contrastive training.

For each training question, the paired SQL y^+ is the positive. We sample 30 negatives: eight unpaired SQL queries retrieved by the first-stage retriever, eight queries from other demonstrations in the same database, and fourteen random queries from the training bank. These sources provide both closely related alternatives and more distant queries.

Retrieved negatives can match the question semantically or share relevant schema information while requiring different SQL operations. They therefore resemble the mistakes that the reranker must correct. Same-database negatives also share database context, which encourages finer comparisons between questions and queries. Random negatives broaden the training examples beyond these local alternatives. We combine the sources to learn from all three comparisons, and test their individual and combined effects in Section 4.5.

We exclude the current positive, remove duplicate demonstration IDs across sources, and discard queries identical to the positive after literal masking. Queries with partial structural overlap remain eligible negatives. Some may still be useful demonstrations, so distinguishing paired SQL from unpaired SQL is only a training proxy for demonstration selection. We evaluate its usefulness with both retrieval and generation results. Appendix B gives the sampling and replenishment rules.

Writing $r^+ = r(q, y^+)$ and $r_j^- = r(q, y_j^-)$, the objective is

$$\mathcal{L} = -\log \frac{\exp(r^+/\tau_{\text{cl}})}{\exp(r^+/\tau_{\text{cl}}) + \sum_{j=1}^{N_{\text{neg}}} \exp(r_j^-/\tau_{\text{cl}})}, \qquad N_{\text{neg}} = 30. \tag{3}$$

We average the loss over questions in a training batch, initialize E with Qwen3-Embedding-0.6B (Zhang et al., 2025), and set $\tau_{\text{cl}} = 0.1$. The first-stage retriever supplies both inference candidates and training negatives. Training therefore includes the kinds of candidates that the model will later rank. Figure 3 shows the shared encoder, and Appendix D gives the full optimization settings.

3.4 SELECTION WITHIN A FIXED CANDIDATE POOL

The reranker chooses which retrieved examples reach the generator. Given the pool $\mathcal{C}_M$ in Eq. (1), it selects

$$\mathcal{D}_K(q) = \text{TopK}_{d_i \in \mathcal{C}_M} r(q, y_i), \qquad K \le M. \tag{4}$$

The selected demonstrations are placed in the prompt in score order. This changes their membership and order, while the candidate pool remains the same. A useful SQL pattern can therefore enter the prompt even if the original retriever ranked it below the cut-off. However, a pattern missing from the pool cannot be recovered by reranking. This distinction also helps interpret the structural metrics. For any fixed question, the best structural match among K selected examples cannot exceed the

best match in the full pool. Reranking can reduce the loss caused by selecting a small subset, but it cannot improve this pool-level upper bound. Gold SQL is used only for evaluation; it is not available to the reranker at inference.

Inference work. At the default budget, scoring involves the question and ten masked candidate SQL strings. The reranker computes their embeddings, compares them by cosine similarity, and sorts the ten scores. It does not make an additional generation call or execute SQL. For a fixed embedding dimension, the similarity calculations grow linearly with M, while a full sort takes $O(M \log M)$ comparisons. Encoding cost also depends on sequence length and implementation. These steps describe the added computation; without latency measurements, we do not infer a wall-clock speedup or negligible overhead.

3.5 Algorithm Design

Based on the preceding analysis, we formulate the overall SARank procedure, as summarized in Algorithm 1.

Algorithm 1. Overall workflow of SARank.

Input: question q, schema Σ, demonstration bank $\mathcal{D}$, budgets M, K
Output: generated SQL y_{final}
$\mathbf{h}_q \leftarrow \text{SchemaAwareEncoder}(q, \Sigma)$
$\mathcal{C}_M \leftarrow \text{TopM}(\mathbf{h}_q, \mathcal{D}, M)$
$\mathbf{u} \leftarrow E(q)$
for each $d_i \in \mathcal{C}_M$ **do**
  $\widetilde{y}_i \leftarrow \text{Mask}(y_i)$
  $\mathbf{v}_i \leftarrow E(\widetilde{y}_i)$
  $r_i \leftarrow \cos(\mathbf{u}, \mathbf{v}_i)$
end for
$\mathcal{C}_{\text{rerank}} \leftarrow \text{SortDescending}(\mathcal{C}_M, \mathbf{r})$
$\mathcal{D}_K \leftarrow \text{TopK}(\mathcal{C}_{\text{rerank}}, K)$
$\text{prompt} \leftarrow \text{BuildPrompt}(q, \Sigma, \mathcal{D}_K)$
$y_{\text{final}} \leftarrow \text{LLM}(\text{prompt})$

4 Experiments

4.1 Experimental Setup

Benchmarks and comparisons. We evaluate on Spider (Yu et al., 2018) and BIRD (Li et al., 2023), and use BEAVER (Chen et al., 2026) and Dr.Spider (Chang et al., 2023) for additional enterprise and robustness evaluations. The Spider/BIRD end-to-end comparison includes RSL-SQL (Cao et al., 2024), CHESS (Talaei et al., 2024), and SchemaRAG (Wu et al., 2026), using GPT-4o, DeepSeek-V4, and Qwen3-Coder. The four-dataset comparison uses DeepSeek-V4. Separately, we compare Original SAR scores, BM25s (Lü, 2024; Robertson & Zaragoza, 2009), BGE embeddings (Xiao et al., 2024), and SARank to assess the quality of candidate ranking.

Configuration. We use $M = 10$ candidates and select $K = 3$ prompt demonstrations by default. Ranking strategies receive the same SAR Top-10 pool, so neither the retriever nor M changes in this comparison. For generation comparisons between the original ordering and SARank, we also keep the prompt template and downstream LLM the same. This protocol isolates the effect of ordering.

The reranker uses native last-token pooling and AdamW. We train for four epochs with a learning rate of 6×10^{-6}, a per-device batch size of four, and two gradient-accumulation steps. The contrastive temperature is 0.1, and SQLite is used for database execution. The appendix retains the full hyperparameters, while Appendix E gives the shared generation prompt.

Zero-shot evaluation. We apply the trained reranker to BEAVER and Dr.Spider without target-dataset fine-tuning. Here, zero-shot refers to reranker transfer: the downstream LLM still receives the selected demonstrations in its prompt. The comparison therefore asks whether the learned question–SQL scoring function remains useful on these evaluation datasets without adapting its parameters to them. We use the same candidate pool for the original ordering and SARank, so any difference between their selected contexts comes from reranking.

Table 1: End-to-end results on Spider and BIRD across language models.

Model	LLM	Spider		BIRD	
		EX	EM	EX	VES
RSL-SQL (Cao et al., 2024)	GPT-4o	**87.40**	66.10	63.49	25.21
	DeepSeek-V4	87.93	74.20	67.21	27.60
	Qwen3-Coder	79.10	52.00	58.38	17.42
CHESS (Talaei et al., 2024)	GPT-4o	86.84	67.98	65.98	21.05
	DeepSeek-V4	87.20	75.80	68.33	28.80
	Qwen3-Coder	81.60	53.70	62.40	19.81
SchemaRAG (Wu et al., 2026)	GPT-4o	84.86	66.80	66.33	31.68
	DeepSeek-V4	88.90	76.60	67.02	31.05
	Qwen3-Coder	83.42	60.40	61.64	26.10
SARank	GPT-4o	86.48	**69.63**	**67.05**	**33.54**
	DeepSeek-V4	**90.18**	**79.91**	**69.74**	**32.15**
	Qwen3-Coder	**84.70**	**63.68**	**64.26**	**30.66**

4.2 Evaluation Metrics

We report execution accuracy (EX), which compares predicted and gold SQL execution results, together with Spider exact match (EM) and BIRD valid efficiency score (VES) (Yu et al., 2018; Li et al., 2023). EM measures normalized structural agreement; VES additionally accounts for execution efficiency for correct predictions.

For demonstration quality, let $\mathcal{F}(\widetilde{y})$ be the SQL structural feature set, including operators and clauses, extracted after literal masking. For a nonempty feature union,

$$\mathrm{sim}(\widetilde{y}^{\star}, \widetilde{y}) = \frac{|\mathcal{F}(\widetilde{y}^{\star}) \cap \mathcal{F}(\widetilde{y})|}{|\mathcal{F}(\widetilde{y}^{\star}) \cup \mathcal{F}(\widetilde{y})|}. \tag{5}$$

This measures feature overlap between two SQL queries, rather than the learned question–SQL score in Eq. (2). Given the selected set $\widetilde{\mathcal{Y}}_i^{(K)}$, define $m_i^{(K)} = \max_{\widetilde{y} \in \widetilde{\mathcal{Y}}_i^{(K)}} \mathrm{sim}(\widetilde{y}_i^{\star}, \widetilde{y})$. We aggregate over n evaluation questions as

$$\mathrm{BestSim}\,@K = \frac{1}{n}\sum_{i=1}^{n} m_i^{(K)}, \qquad \mathrm{Recall}\,@K(\delta) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(m_i^{(K)} \geq \delta). \tag{6}$$

We use $\delta = 0.8$. BestSim measures the closest structural match among the selected examples. Recall measures how often at least one example reaches the similarity threshold. Since neither metric checks whether the generated answer is correct, we also report EX.

4.3 End-to-End Results

Table 1 compares end-to-end results on Spider and BIRD. SARank improves over SchemaRAG in all 12 LLM–metric combinations. Averaged across the three LLMs, the gains are 1.39 EX and 3.14 EM points on Spider, and 2.02 EX and 2.51 VES points on BIRD. The improvement is therefore consistent across the tested backbones and metrics, rather than limited to one model or one best score.

Across all evaluated systems, SARank has the highest reported score in 11 of the 12 settings. With DeepSeek-V4, it reaches 90.18 EX and 79.91 EM on Spider, and 69.74 EX and 32.15 VES on BIRD. The exception is Spider EX with GPT-4o: SARank scores 86.48, below RSL-SQL at 87.40, although its EM is higher (69.63 versus 66.10). It still improves on SchemaRAG in this setting. These results support consistent gains over the retrieval baseline, without a claim of winning every system comparison.

Figure 4 extends the comparison to four datasets using DeepSeek-V4. Reranking the same Top-10 pool improves both structural metrics and EX on each dataset. BestSim gains range from 0.039 to 0.120, and Recall gains range from 0.123 to 0.318. Thus, better structural selection occurs alongside better generation in these experiments. BEAVER and Dr.Spider also extend this observation to the reported enterprise and perturbation settings.

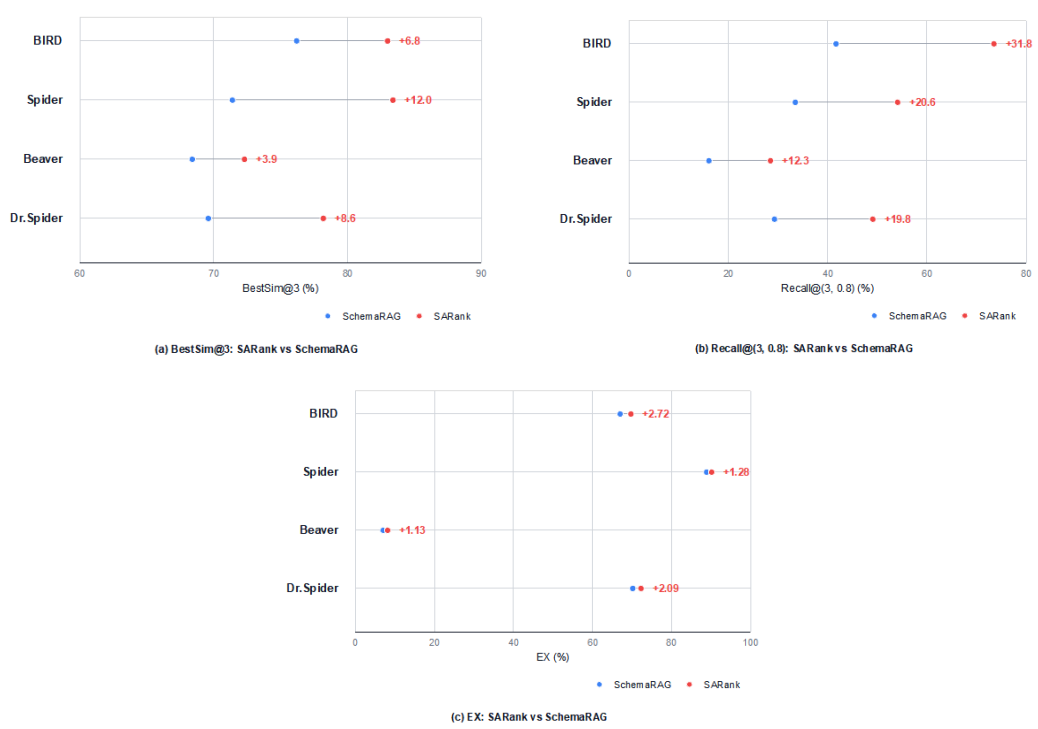


Figure 4: Retrieval-quality and execution-accuracy comparison between SchemaRAG and SARank on four datasets: (a) BestSim @3 (%), (b) Recall @3 ($\delta = 0.8$) (%), and (c) EX (%).

Table 2: Comparison of candidate ranking strategies on Spider.

Method	BestSim @3	Recall @3 $\delta = 0.8$
Original SAR score (Wu et al., 2026)	0.714	0.335
BM25s (Lü, 2024)	0.722	0.359
BGE (Xiao et al., 2024) embedding rerank	0.746	0.404
SARank	0.834	0.541

Zero-shot results on BEAVER and Dr.Spider. On BEAVER, SARank raises BestSim by 0.039, Recall by 0.123, and EX by 1.13 percentage points. On Dr.Spider, BestSim increases by 0.086 and Recall by 0.198, while EX improves by 2.09 percentage points. The BestSim and Recall differences are on the 0–1 scale, whereas EX is reported as a percentage. Both datasets therefore show improved structural selection together with higher execution accuracy, even though the reranker is not fine-tuned on either target dataset.

Both datasets show higher structural-match scores and EX. These results support transfer to the evaluated settings; they do not establish robustness across individual perturbation types.

4.4 Controlled Comparison of Ranking Strategies

Table 2 compares four ways to order the same SAR pool. BM25s and BGE both improve on Original SAR, while SARank reaches the highest BestSim and Recall, at 0.834 and 0.541. Within the same Top-10 pool, SARank exceeds BGE by 0.088 BestSim@3 and 0.137 Recall@3. Table 2 evaluates structural selection quality; generation accuracy is not evaluated here.

Table 3: Effect of negative sampling strategies on Spider and BIRD.

Negative strategy	BestSim @3		Recall @3 ($\delta = 0.8$)	
	Spider	**BIRD**	**Spider**	**BIRD**
Original	0.714	0.762	0.335	0.416
Random Neg.	0.789	0.806	0.502	0.702
Same-DB Neg.	0.791	0.809	0.511	0.709
SAR Hard Neg.	0.794	0.813	0.514	0.711
Mixed Neg.	0.834	0.830	0.541	0.735

Table 4: Effect of SQL literal masking on Spider.

Variant	BestSim @3	Recall @3 ($\delta = 0.8$)	**EX**
SARank w/o SQL Masking	0.765	0.432	89.10
SARank w/ SQL Masking	0.834	0.541	90.18

4.5 Component Ablations

Negative sources. We keep the SARank architecture and training settings fixed and change the negative source. Original SAR provides a reference retrieval result rather than a SARank ablation. In Table 3, retrieved hard negatives perform best among the single-source choices. Mixed negatives then improve all four measurements: BestSim/Recall rise by 0.040/0.027 on Spider and 0.017/0.024 on BIRD. This supports using the sources together, since their combination performs better than any tested source alone.

SQL literal abstraction. Table 4 compares SARank with and without SQL literal masking on Spider. Masking raises BestSim from 0.765 to 0.834 and Recall from 0.432 to 0.541. EX also rises from 89.10 to 90.18, a gain of 1.08 points. The benefit therefore extends beyond the structural retrieval metrics to generation accuracy in this setting.

4.6 Demonstration Budget and Qualitative Analysis

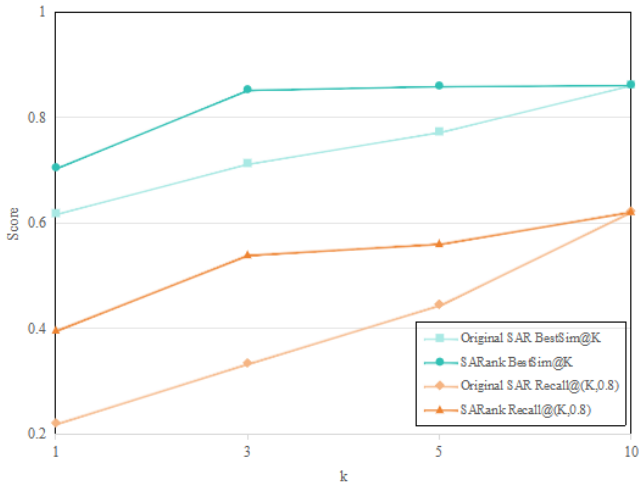


Figure 5: Selection quality on Spider as K varies within the same Top-10 SAR pool. BestSim and Recall use the left-axis 0–1 scale.

Figure 5 varies the number of selected demonstrations while keeping the initial pool at $M = 10$. SARank improves both metrics for $K \in \{1, 3, 5\}$. At $K = 1$, BestSim rises from 0.619 to 0.726 and Recall from 0.221 to 0.398. Among the tested budgets, $K = 3$ yields the largest gains in BestSim and Recall: 0.120 BestSim and 0.206 Recall. At $K = 5$, the gains narrow to 0.077 and 0.116.

At $K = 10$, both methods select the full pool and therefore have identical set-based metrics. The gains at smaller K show the benefit of selecting a subset; candidate-pool size M is held fixed in this experiment.

Figure 6 shows how the selected examples can affect generation. The question asks for the most frequent nationality, which requires `GROUP BY`, `ORDER BY COUNT(*) DESC`, and `LIMIT 1`. Original SAR selects field-selection queries, and the generated SQL includes only `LIMIT 1`. SARank moves a structurally aligned example from rank 10 to rank 1. With this example in the prompt, the generated SQL includes the missing aggregation operations and executes correctly. Similarly, Figure 1 shows a nested aggregation example moving from rank 6 to rank 1. These cases help explain the selection behavior; the comparisons and ablations measure its broader effect.

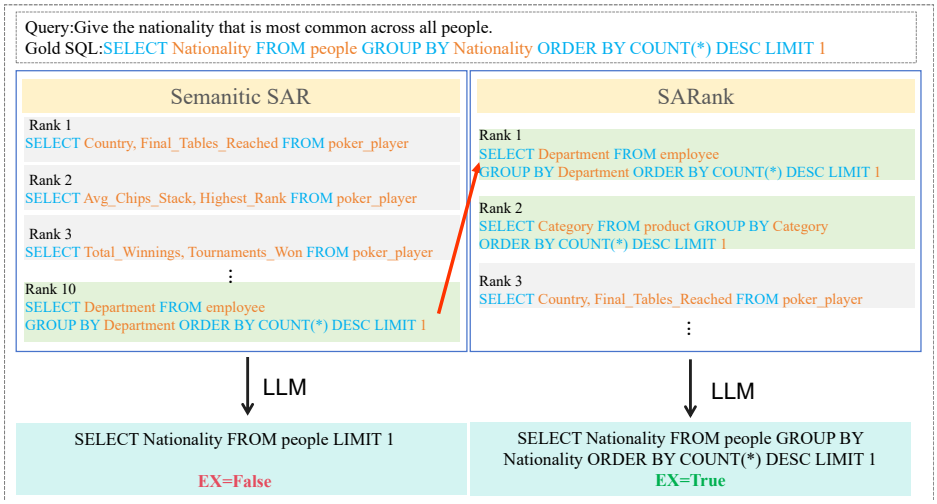


Figure 6: A grouping and count-based ordering case comparing demonstrations selected by Original SAR and SARank and the resulting generated SQL.

5 Limitations and Future Work

SARank depends on the first-stage candidate pool. It can promote useful demonstrations that have been retrieved, but cannot recover examples missing from the Top-M candidates. Our experiments use one first-stage retriever. Although the zero-shot results on BEAVER and Dr.Spider support transfer across datasets, they do not establish that the gains hold with other retrievers. Future work will study joint retrieval–reranking optimization and evaluate combinations with other retrieval methods. Training and evaluation also have limitations. Unpaired SQL queries are used as negatives, although some may still be useful demonstrations. In addition, structural overlap does not fully capture demonstration usefulness, so we report execution accuracy alongside the structural metrics. Further evaluation across SQL dialects and real-world settings would help clarify where SARank is most effective.

6 Conclusion

We presented SARank, a structure-aware demonstration reranker for Text-to-SQL. It addresses the demonstration exposure bottleneck by selecting examples from a fixed candidate pool before prompt construction. Trained with literal masking and mixed negatives, SARank scores each candidate SQL directly against the question, without requiring a predicted target SQL or execution feedback.

SARank improves the reported SchemaRAG results by selecting demonstrations from the same retrieved pool. The masking and negative-sampling ablations support these design choices, while performance remains bounded by the examples available to the reranker. On Spider and BIRD, the gains are consistent across three LLMs.

AI USE STATEMENT

AI tools were used only to polish the language of the manuscript. The research ideas, experimental design and execution, and overall manuscript structure were developed by the authors. The authors reviewed all AI-assisted edits and take full responsibility for the final manuscript.

REFERENCES

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, et al. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *arXiv:2411.00073*. https://arxiv.org/abs/2411.00073

Shuaichen Chang, Jun Wang, Mingwen Dong, et al. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. *In International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2301.08881

Peter Baile Chen, Devin Yang, Weiyue Li, et al. 2026. BEAVER: An Enterprise Benchmark for Text-to-SQL. *arXiv:2409.02038 (revised May 2026)*. https://arxiv.org/abs/2409.02038

Yongnan Chen, Zhuo Chang, Shijia Gu, et al. 2025. ADEPT-SQL: A High-performance Text-to-SQL Application for Real-World Enterprise-Level Databases. *In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations). 275–283*. https://aclanthology.org/2025.acl-demo.27/

Dawei Gao, Haibin Wang, Yaliang Li, et al. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv:2308.15363*. https://arxiv.org/abs/2308.15363

Jiawei Gu, Ziting Xian, Yuanzhen Xie, et al. 2025. Toward Structured Knowledge Reasoning: Contrastive Retrieval-Augmented Generation on Experience. *In Findings of the Association for Computational Linguistics: ACL 2025. 23891–23910*. https://aclanthology.org/2025.findings-acl.1224/

Hojae Han, Yeonseok Jeong, Seung-won Hwang, et al. 2026. R^3-SQL: Ranking Reward and Resampling for Text-to-SQL. *arXiv:2604.25325. Accepted to Findings of ACL 2026*. https://arxiv.org/abs/2604.25325

Zijin Hong, Zheng Yuan, Qinggang Zhang, et al. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. *arXiv:2406.08426*. https://arxiv.org/abs/2406.08426

Fengrui Kang, Mingxi Tan, Xianying Huang, et al. 2025. Self-Contrastive Loop of Thought Method for Text-to-SQL Based on Large Language Model. *In Proceedings of the 1st Joint Workshop on Large Language Models and Structure Modeling (XLLM 2025). 71–85*. https://aclanthology.org/2025.xllm-1.8/

Donguk Kwon, Jaewan Moon, and Jongwook Lee. 2025. SEG-SQL: Structure-aware Example Generation for Text-to-SQL Method with In-context Learning. *Journal of KIISE 52, 11 (2025), 992–1001*. https://doi.org/10.5626/JOK.2025.52.11.992

Jihyung Lee, Jin-Seop Lee, Jaehoon Lee, et al. 2025a. DCG-SQL: Enhancing In-Context Learning for Text-to-SQL with Deep Contextual Schema Link Graph. *In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 15397–15412*. https://aclanthology.org/2025.acl-long.748/

Jimin Lee, Ingeol Baek, Byeongjeong Kim, et al. 2025b. SAFE-SQL: Self-Augmented In-Context Learning with Fine-grained Example Selection for Text-to-SQL. *In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing. 19023–19035*. https://aclanthology.org/2025.emnlp-main.962/

Fangyu Lei, Jixuan Chen, Yuxiao Ye, et al. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. *In International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2411.07763

Jinyang Li, Binyuan Hui, Ge Qu, et al. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. *arXiv:2305.03111*. https://arxiv.org/abs/2305.03111

Geling Liu, Yunzhi Tan, Ruichao Zhong, et al. 2024. Solid-SQL: Enhanced Schema-linking based In-context Learning for Robust Text-to-SQL. *arXiv:2412.12522*. https://arxiv.org/abs/2412.12522

Xing Han Lü. 2024. BM25S: Orders of magnitude faster lexical search via eager sparse scoring. *arXiv:2407.03618*. https://arxiv.org/abs/2407.03618

Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. 2024. From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. *arXiv:2410.01066*. https://arxiv.org/abs/2410.01066

Minjun Park, Yongju Seong, Myoseop Sim, et al. 2026. ReSQL: Self-Improving Framework for Reasoning-Aware Text-to-SQL Dataset Generation. *In Findings of the Association for Computational Linguistics: ACL 2026. 33582–33602*. https://aclanthology.org/2026.findings-acl.1677/

Shengmin Piao, Jieun Lee, and Sanghyun Park. 2026. LitE-SQL: A Lightweight and Efficient Text-to-SQL Framework with Vector-based Schema Linking and Execution-Guided Self-Correction. *In Findings of the Association for Computational Linguistics: EACL 2026. 3593–3608*. https://aclanthology.org/2026.findings-eacl.186/

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, et al. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. *In International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2410.01943

Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval 3, 4 (2009), 333–389*. https://doi.org/10.1561/1500000019

Chen Shen, Jin Wang, Sajjadur Rahman, et al. 2025. MageSQL: Enhancing In-context Learning for Text-to-SQL Applications with Large Language Models. *arXiv:2504.02055*. https://arxiv.org/abs/2504.02055

Zhili Shen, Pavlos Vougiouklis, Chenxin Diao, et al. 2024. Improving Retrieval-augmented Text-to-SQL with AST-based Ranking and Schema Pruning. *In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. 7865–7879*. https://aclanthology.org/2024.emnlp-main.449/

Xiaodong Su, Yang Gu, Peng Wang, et al. 2026. A robust natural language text-to-SQL generation framework with dynamic strategies based on LLMs. *Scientific Reports 16, 1, Article 7892 (2026)*. https://www.nature.com/articles/s41598-026-39128-9

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, et al. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *arXiv:2405.16755*. https://arxiv.org/abs/2405.16755

Sameeksh Thakur, Saurabh Chandra, Shekhar Sahu, et al. 2025. Adaptive Demonstration Selection with Reinforcement Feedback for Enhanced Text-to-SQL Generation. *International Journal of Research Publication and Reviews 6, 4 (2025), 13198–13202*. https://ijrpr.com/uploads/V6ISSUE4/IJRPR43457.pdf

Di Wu, Zetong Tang, Yi He, and Xin Luo. 2026. SchemaRAG: A Schema-aware Retrieval-Augmented Generation Framework for Text-to-SQL. *Proceedings of the ACM on Management of Data 4, 1, Article 82 (2026), 26 pages*. https://doi.org/10.1145/3786696

Zhenhe Wu, Zhongqiu Li, Jie Zhang, et al. 2025. UCS-SQL: Uniting Content and Structure for Enhanced Semantic Bridging In Text-to-SQL. *In Findings of the Association for Computational Linguistics: ACL 2025. 8156–8168*. https://aclanthology.org/2025.findings-acl.427/

Shitao Xiao, Zheng Liu, Peitian Zhang, et al. 2024. C-Pack: Packed Resources For General Chinese Embeddings. *In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. 641–649.* https://arxiv.org/abs/2309.07597

Xindi Xu, Xiaoning Jiang, Ao Wang, et al. 2026. PPR-SQL: a robust text-to-SQL system via probabilistic graph pruning and multi-view reasoning. *Journal of Intelligent Information Systems 64, 4 (2026), 1829–1853.* https://link.springer.com/article/10.1007/s10844-026-01049-4

Tao Yu, Rui Zhang, Kai Yang, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 3911–3921.* https://aclanthology.org/D18-1425/

Yanzhao Zhang, Mingxin Li, Dingkun Long, et al. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. *arXiv:2506.05176.* https://arxiv.org/abs/2506.05176

2026. Structure-Aware RAG: Boosting Accuracy in LLM-Based SQL Dialect Translation via AST-Guided Retrieval. *CSITec 2026.*

A Retrieval and Inference Details

A.1 Inherited Schema-Aware Retriever

The candidate retrieval stage combines question semantics and database schema information to retrieve an initial set of demonstrations for subsequent reranking (Wu et al., 2026). Given the target schema Σ, its question-conditioned representation is computed as

$$\mathbf{a}_q = A(q, \Sigma). \tag{7}$$

In Eq. (7), A denotes hierarchical attention aggregation. Each table first attends to its columns to obtain a column-aware representation. The question then attends to these table representations, yielding the schema representation $\mathbf{a}_q$.

Next, we fuse the question embedding with the schema representation to obtain an enhanced representation for retrieval:

$$\mathbf{h}_q = F\big(E_{\text{ret}}(q), \mathbf{a}_q\big). \tag{8}$$

Here, $E_{\text{ret}}(q)$ is the question embedding produced by the text encoder, and F is a masked Transformer that incorporates schema information into the question representation. The resulting vector $\mathbf{h}_q$ in Eq. (8) is used to compute similarity to candidate demonstrations.

For each demonstration $d_i = (q_i, \Sigma_i, y_i)$ in the demonstration bank $\mathcal{D}$, we compute its enhanced representation $\mathbf{h}_i$ using the same procedure, where y_i denotes its SQL query. The candidate set is defined as the M demonstrations with the highest cosine similarities to $\mathbf{h}_q$:

$$\mathcal{C}_M = \text{TopM}_{d_i \in \mathcal{D}}\big(\cos(\mathbf{h}_q, \mathbf{h}_i)\big). \tag{9}$$

Equation (9) supplies the first-stage candidate pool for reranking.

A.2 Detailed Architecture Diagrams

The shared encoder architecture is shown in Figure 3 in the main text.

B NEGATIVE SAMPLING DETAILS

The three negative sources used by the training objective are described below.

> **(1) Retrieval-based hard negatives.** We select unpaired SQL queries from the candidates retrieved for each training question in the first stage. These candidates may be semantically related to the question yet use different query operations, providing training examples similar to those encountered during inference. They help the model distinguish retrieval relevance from a match to the question's query requirements.
>
> **(2) Same-database negatives.** We sample SQL queries from other demonstrations associated with the same database. Because these examples share the same schema, the model must distinguish question requirements through query operations and their composition, reducing reliance on database-related information alone.
>
> **(3) Random negatives.** We sample SQL queries from the training bank to provide broader contrasts beyond retrieved and same-database examples.

Each question is assigned 30 negatives: 8 retrieval-based hard negatives, 8 same-database negatives, and 14 random negatives. During sampling, we exclude the current positive example and remove duplicates across sources by demonstration ID. Examples shared with the hard-negative pool are retained in that pool, and additional examples are sampled from the other sources to meet their respective quotas.

We also exclude SQL queries that become identical to the positive SQL after literal abstraction, preventing the same input from receiving conflicting supervision. Queries with only partial structural similarity are retained as negatives to help the model distinguish differences in query operations. Although these queries are not reference answers to the target question, they may still contain reusable query patterns.

C EXPERIMENTAL CONFIGURATION AND BENCHMARKS

C.1 BENCHMARK DESCRIPTIONS

Spider (Yu et al., 2018) is a widely used cross-domain text-to-SQL benchmark containing 10,181 questions and 5,693 unique SQL queries over 200 databases from 138 domains. The dataset focuses on evaluating standard cross-domain text-to-SQL performance.

BIRD (Li et al., 2023) contains 12,751 question-SQL pairs over 95 large databases from 37 professional domains, with a total database size of 33.4 GB. It features large and noisy database contents, external knowledge, complex SQL queries, and query-efficiency requirements.

BEAVER (Chen et al., 2026) is an enterprise text-to-SQL benchmark constructed from three anonymized private data warehouses. It contains 9,128 queries spanning 812 tables across 19 domains. Its complex schemas and business-oriented queries frequently require multi-table joins, aggregations, nested queries, and domain knowledge.

Dr.Spider (Chang et al., 2023) is a diagnostic robustness benchmark derived from the Spider development set. It focuses on evaluating the robustness of demonstration selection under schema, lexical, and query variations.

C.2 COMPUTE AND SOFTWARE

All experiments were conducted on a server with two NVIDIA Tesla V100 GPUs, each with 32 GB of memory. The server also had 1,024 GB of RAM and an Intel Xeon Gold 5218 CPU at 2.30 GHz, and ran Ubuntu 22.04 LTS with PyTorch 2.7.0. SQLite was used for database execution and evaluation.

C.3 Model Versions

Table 5: Exact model versions used in the experiments.

Model	Exact Version	Parameters	Inference
GPT-4o	`gpt-4o` (snapshot not pinned)	Proprietary / undisclosed	API
DeepSeek-V4-Flash	`DeepSeek-V4-Flash-0731`	284B total / 13B active	API
Qwen3-Coder-Flash	`qwen3-coder-flash-2025-07-28`	Undisclosed	API
Qwen3-Embedding-0.6B	`Qwen/Qwen3-Embedding-0.6B`	0.6B	Local

C.4 Dataset Statistics

Table 6: Dataset split and evaluation statistics.

Dataset	Official Train	Official Dev / Public Eval	Official Test / Robustness Eval
Spider	8,659	1,034	2,147 hidden test
BIRD	9,428	1,534	1,789 hidden test
BEAVER	No canonical train/dev split like Spider	7,978 public queries in latest release	1,150 private test queries
Dr.Spider	No independent train split	Derived from Spider dev	17 perturbation test sets, 15,269 paired cases

D Reranker Training Details

D.1 Implementation Details

Table 7: Hyperparameters used to train the reranker.

Hyperparameter	Value
Base encoder	Qwen3-Embedding-0.6B
Training objective	InfoNCE
Optimizer	AdamW
Learning rate	6×10^{-6}
Weight decay	0.1
Adam (β_1, β_2)	0.9, 0.95
LR scheduler	Cosine
Warmup steps	0
Training epochs	4
Per-device batch size	4
Gradient accumulation	2
Maximum gradient norm	1.0
Precision	BF16
Random seed	42

D.2 Input Representation and Training Objective

Both natural-language questions and literal-abstracted SQL queries are tokenized using the tokenizer associated with Qwen3-Embedding-0.6B. The maximum input length is set to 32,768 tokens; samples exceeding this limit are discarded during preprocessing. We adopt the native last-token pooling strategy of Qwen3-Embedding-0.6B to obtain fixed-dimensional representations. Candidate compatibility is measured using cosine similarity between the question and SQL embeddings.

The reranker is optimized with the InfoNCE objective using a contrastive-learning temperature of $\tau_{\text{cl}} = 0.1$.

E Prompt Template for SQL Generation

We use a unified prompt template for downstream SQL generation across all evaluated methods. The prompt contains the database schema, optional evidence, predicted relevant schema elements, retrieved demonstrations, and the target question. The schema elements provided by the schema-linking stage are explicitly constrained to be used during generation, while retrieved demonstrations are provided only as structural references. The same prompt format is used across different reranking strategies to ensure a controlled comparison.

E.1 System Prompt

```
You are an NL2SQL expert.

I will give you the database structure and the most likely
table columns.
You can use this information to perform the NL2SQL task.

Please read and understand the database schema carefully,
and generate an executable SQL query. The generated SQL
should be enclosed by ```sql and ```.
```

E.2 User Prompt

```
### Database Schema
{database}

### Evidence
{evidence}

### Relevant Schema Elements (MUST use these)
{links_str}

### Reference Examples
(Follow SQL structure only, DO NOT copy entities)

{demonstration_1}
{demonstration_2}
{demonstration_3}

### Current Question
{question}

### Your Task
Generate the correct SQLite SQL query.
Use only the schema elements above.
Follow the SQL ...
```

F Qualitative Case Studies

To provide a qualitative understanding of how SARank affects demonstration selection and downstream SQL generation, Figures 1 and 6 present two representative examples from the Spider development set. As shown in Fig. 1, the target SQL contains a NOT IN subquery together with a GROUP BY–HAVING SUM structure. The top three demonstrations selected by Original SAR (Wu

et al., 2026) involve simple projection, minimum aggregation, and distinct counting, offering limited structural guidance for the target query. A demonstration with an analogous SQL skeleton appears at rank 6, differing mainly in database entities and literal values. SARank promotes this demonstration to rank 1. This example shows that SARank can reduce the influence of instance-specific content and prioritize correspondences among nested queries, grouped aggregation, and filtering conditions.

Fig. 6 further illustrates how improved demonstration exposure can benefit SQL generation. The target query asks for the most frequent nationality and therefore requires GROUP BY, ORDER BY COUNT(∗) DESC, and LIMIT 1. The Original SAR (Wu et al., 2026) Top-3 candidates mainly contain field-selection queries from the same database and provide insufficient guidance for the required aggregation pattern. Consequently, the generated SQL retains only LIMIT 1 and produces an incorrect execution result. SARank promotes a structurally aligned demonstration from rank 10 to rank 1. With this demonstration included in the prompt, the LLM generates the complete grouping, count-based ordering, and limiting operations, changing the execution outcome from incorrect to correct.

Together, these examples illustrate that useful demonstrations may already exist in the retrieved SAR candidate pool, while their positions determine whether they become available under a limited context budget. By prioritizing structurally aligned candidates within the same pool, SARank improves Top-3 demonstration exposure and provides more informative structural guidance for downstream SQL generation.

G Extended Related Work

G.1 LLM-based Text-to-SQL

Recent LLM-based Text-to-SQL systems increasingly adopt multi-stage pipelines that combine schema understanding, retrieval, generation, and refinement. Methods such as CHESS (Talaei et al., 2024) and RSL-SQL (Cao et al., 2024) improve complex query generation through schema selection, iterative reasoning, and self-correction, while ADEPT-SQL (Chen et al., 2025) extends this paradigm to enterprise databases with schema alignment and historical query retrieval. More recent approaches, including LitE-SQL (Piao et al., 2026) and ReSQL (Park et al., 2026), further explore lightweight retrieval, hard-negative learning, and error-driven refinement. These studies have substantially improved schema grounding and generation reliability, while the selection of effective few-shot demonstrations remains important for in-context SQL generation.

G.2 Retrieval-Augmented Demonstration Selection for Text-to-SQL

Retrieval-augmented Text-to-SQL commonly selects demonstrations according to their relevance to the target question. However, semantic similarity does not necessarily imply similar SQL operations. Recent methods therefore incorporate richer structural signals: DCG-SQL (Lee et al., 2025a) and SchemaRAG (Wu et al., 2026) introduce schema-aware representations for demonstration retrieval, while ASTReS (Shen et al., 2024) and UCS-SQL (Wu et al., 2025) further exploit SQL structures such as ASTs and query skeletons. These studies reflect a shift from semantic retrieval toward structure-aware demonstration selection. Our comparison isolates a second-stage scorer within a fixed SchemaRAG candidate pool. SARank scores candidate SQL directly against the question, without requiring an approximated target SQL. When the candidate pool is compressed to a small K, structurally relevant demonstrations may remain below the cut-off; SARank addresses this stage under a limited context budget.

G.3 Structure-Aware Reranking and Contrastive Learning

Structure-aware ranking has also been explored at different stages of Text-to-SQL. ASTReS (Shen et al., 2024) ranks demonstrations using AST similarity derived from an approximated SQL query, whereas methods such as R^3-SQL (Han et al., 2026) rank candidate SQL queries after generation. In contrast, SARank performs **pre-generation demonstration reranking** by directly modeling the compatibility between the target question and candidate SQL.

Contrastive learning provides another means of improving fine-grained discrimination. SchemaRAG (Wu et al., 2026) learns schema-aware representations through contrastive objectives, while LitE-SQL (Piao et al., 2026) introduces hard negatives for schema retrieval. SARank instead constructs hard negatives from the first-stage candidate pool and combines them with same-database and random negatives, aligning representation learning with the actual reranking scenario.

H NOTATION SUMMARY

Table 8: Notation used throughout the paper.

Notation	Explanation
	Common Variables
q	The target natural-language question.
Σ	The database schema associated with the target question.
$\mathcal{D}$	The demonstration bank from which candidate Question–SQL examples are retrieved.
$d_i = (q_i, \Sigma_i, y_i)$	The i-th demonstration in $\mathcal{D}$, consisting of a question, its database schema, and the corresponding SQL query.
q_i	The natural-language question in demonstration d_i.
Σ_i	The database schema associated with demonstration d_i.
y_i	The SQL query associated with demonstration d_i.
M	Size of the candidate demonstration pool retrieved in the first stage; $M = 10$ in the default setting.
K	Number of demonstrations retained after reranking and included in the final prompt; $K = 3$ by default.
	Schema-Aware Candidate Retrieval
$A(q, \Sigma)$	Hierarchical attention aggregation that produces a question-conditioned schema representation.
$\mathbf{a}_q$	Question-conditioned schema representation obtained from $A(q, \Sigma)$.
$E_{\text{ret}}(\cdot)$	Text encoder used by the first-stage schema-aware retriever.
$F(\cdot)$	Masked Transformer that fuses the question embedding with the schema representation.
$\mathbf{h}_q$	Schema-enhanced representation of the target question used for candidate retrieval.
$\mathbf{h}_i$	Schema-enhanced representation of demonstration d_i.
$\mathcal{C}_M$	Top-M candidate demonstration set returned by the schema-aware retriever.
$\text{TopM}(\cdot)$	Operator that selects the M demonstrations with the highest retrieval scores.
	SQL Structure-Aware Reranking
$\text{Mask}(\cdot)$	SQL literal-abstraction operation that replaces literal values with placeholders while preserving SQL structure.
$\widetilde{y}_i$	Literal-abstracted SQL query derived from y_i.
$E(\cdot)$	Shared reranking encoder that maps questions and abstracted SQL queries into a common representation space.
$\mathbf{u} = E(q)$	Embedding representation of the target question produced by the reranking encoder.

Continued on next page

Table 8 continued: notation summary.

Notation	Explanation
$\mathbf{v}_i = E(\widetilde{y}_i)$	Embedding representation of the abstracted candidate SQL.
$r(q, y_i)$	Question–SQL compatibility score computed using cosine similarity.
r_i	Reranking score of candidate demonstration d_i.
$\mathbf{r}$	Vector of reranking scores for all demonstrations in the candidate pool.
$\mathcal{C}_{\text{rerank}}$	Candidate set after sorting $\mathcal{C}_M$ according to the learned reranking scores.
$\mathcal{D}_K$	Final Top-K demonstrations selected from the reranked candidate set for prompt construction.
	Candidate-Aware Contrastive Learning
y^+	Positive SQL query paired with the training question.
y_j^-	The j-th negative SQL query used for contrastive learning.
N_{neg}	Number of negative SQL queries assigned to each training question; $N_{\text{neg}} = 30$.
r^+	Matching score between the training question and its positive SQL.
r_j^-	Matching score between the training question and the j-th negative SQL.
$\mathcal{L}$	InfoNCE contrastive loss used to optimize the shared reranking encoder.
τ_{cl}	Temperature coefficient of the InfoNCE contrastive objective.
	Retrieval Evaluation
$\widetilde{y}_i^\star$	Literal-abstracted gold SQL for the i-th evaluation example.
$\widetilde{\mathcal{Y}}_i^{(K)}$	Set of abstracted SQL queries from the Top-K retrieved demonstrations for evaluation example i.
$\mathcal{F}(\widetilde{y})$	Set of SQL structural features extracted from the literal-masked query $\widetilde{y}$.
$\text{sim}(\cdot, \cdot)$	Jaccard similarity between the structural feature sets of two literal-masked SQL queries.
$m_i^{(K)}$	Highest structural similarity between the gold SQL and any SQL among the Top-K demonstrations for example i.
n	Number of examples in the evaluation set.
$\text{BestSim}\,@K$	Mean best structural similarity over all evaluation examples.
δ	Structural-similarity threshold used by $\text{Recall}\,@K$; set to 0.8 by default.
$\text{Recall}\,@K(\delta)$	Fraction of evaluation examples whose Top-K demonstrations contain at least one SQL with structural similarity no lower than δ.
$\mathbb{I}(\cdot)$	Indicator function used in the definition of $\text{Recall}\,@K$.
	Generation and End-task Evaluation
y_{final}	Final SQL query generated by the downstream LLM.
EX	Execution Accuracy; percentage of predicted SQL queries whose execution results match those of the gold SQL.
EM	Exact Match; agreement between normalized predicted and gold SQL structures.
VES	Valid Efficiency Score; execution-correctness metric additionally accounting for query efficiency.