SchemaGraphSQL: Efficient Schema Linking with Pathfinding Graph Algorithms for Text-to-SQL on Large-Scale Databases

Anonymous ACL submission

Abstract

Text-to-SQL systems translate natural language questions into executable SQL queries, and recent progress with large language models (LLMs) has driven substantial improvements in this task. Schema linking remains a critical component in Text-to-SQL systems, reducing prompt size for models with narrow context windows and sharpening model focus even when the entire schema fits. We present a zero-shot, training-free schema linking approach that first constructs a schema graph based on foreign key relations, then uses a single prompt to Gemini 2.5 Flash to extract source and destination tables from the user query, followed by applying classical path-finding algorithms and post-processing to identify the optimal sequence of tables and columns that should be joined, enabling the LLM to generate more accurate SQL queries. Despite being simple, cost-effective, and highly scalable, our method achieves state-of-the-art results on the BIRD benchmark, outperforming previous specialized, fine-tuned, and complex multi-step LLM-based approaches. We conduct detailed ablation studies to examine the precision–recall trade-off in our framework. Additionally, we evaluate the execution accuracy of our schema filtering method compared to other approaches across various model sizes.

1 Introduction

Relational databases are foundational to modern data infrastructure, powering analytics, reporting, and decision-making across domains. Yet, querying these databases typically requires fluency in SQL—a barrier for many users. Text-to-SQL systems aim to democratize access by translating natural language (NL) questions into executable SQL queries (Zhu et al., 2024; Zhang et al., 2024). Enabled by large language models (LLMs), recent systems achieve impressive performance across complex cross-domain settings.

Algorithm 1: Graph-Based Schema Linking

```
Input: Question $q$; schema graph $G$;
Output: Relevant table set $\mathcal{T}^\star \subseteq \mathcal{T}$
1 Step 1: Identify source/destination tables;
2     $(\mathcal{T}_{src}, \mathcal{T}_{dst}) \leftarrow \text{LLM\_call}(q)$
3 Step 2: Build candidate path set;
4     $\mathcal{C} \leftarrow \varnothing$;
5     foreach $T_{src} \in \mathcal{T}_{src}, T_{dst} \in \mathcal{T}_{dst}$ do
6         $\mathcal{C} \leftarrow \mathcal{C} \cup \text{ShortestPaths}(T_{src}, T_{dst})$
7 Step 3: Build union path;
8     $U \leftarrow \bigcup_{p \in \mathcal{C}} p$;
9     return $U$;
```

However, bringing these systems to real-world applications introduces new challenges. Enterprise databases often contain hundreds of tables and thousands of columns—far beyond the scale of academic benchmarks. Supplying the entire schema to the model risks exceeding token limits and introduces considerable noise, which can hinder SQL generation and inflate inference cost (Cao et al., 2024; Li et al., 2023c). In practice, user queries typically touch only a small subset of the schema, making it crucial to identify and extract the relevant part—a process known as *schema linking* (Lei et al., 2020).

Schema linking aims to determine which tables or columns are needed to answer a user question. While early methods relied on exact string matches (Yu et al., 2018), recent work has proposed neural linkers (Gan et al., 2023), retrieval-based modules (Pourreza and Rafiei, 2024), and prompt-based systems (Wang and Liu, 2025). These can capture semantic signals beyond surface overlap, but typically require supervised training, complex multi-stage pipelines, or brittle prompt engineering. They also struggle with the core trade-off: being precise enough to reduce noise, yet broad enough not to miss critical context (Liu et al., 2024; Wang et al., 2025).

In this work, we ask: ***Can we perform effective schema linking without relying on specialized fine-tuned models or complex prompting strategies?*** Our answer is affirmative.

We introduce **SchemaGraphSQL**, a zero-shot schema linking framework that revisits classical algorithmic tools. Our key idea is to model schema linking as a graph search problem. We treat the database schema as a graph where nodes are tables and edges reflect foreign-key connections. Given a user query, we make a single LLM call to predict coarse-grained source and destination tables, then apply deterministic path-finding algorithms to enumerate all shortest join paths between them. The union of these paths forms a compact sub-schema—guaranteed to be connected and grounded in the query.

This perspective is both simple and surprisingly powerful. **To our knowledge, SchemaGraphSQL is the first Text-to-SQL system to rely exclusively on classical graph algorithms for schema linking, using LLMs only for coarse guidance.** It requires no training, incurs minimal inference cost, and integrates easily into any downstream parser or LLM-based SQL generator.

Empirical results on the BIRD benchmark show that SchemaGraphSQL achieves new state-of-the-art scores on recall-focused schema linking metrics and improves execution accuracy across multiple SQL generators. We also conduct ablations demonstrating that even this minimal linking method outperforms specialized neural or prompt-based systems in robustness and cost-efficiency.

Main Contributions:

- We introduce a zero-shot schema linking approach that models database schemas as graphs and applies classical path-finding algorithms. Our method achieves state-of-the-art performance without requiring any training—either for fine-tuning or inference—making it highly suitable for low-resource, real-world scenarios where training data is unavailable or difficult to obtain.

- Our system uses only a single lightweight LLM call (Gemini 2.5 Flash) per query, with minimal token usage (averaging 4593 input and 14 output tokens), significantly reducing inference cost while maintaining ease of integration and deployment.

- We conduct comprehensive empirical evaluations, demonstrating superior schema linking performance compared to fine-tuned and specialized methods. Additionally, we perform detailed ablation studies to examine precision–recall trade-offs and assess the downstream impact on Text-to-SQL execution accuracy across a range of open-source and closed-source models.

2 Related Work

Text-to-SQL systems aim to automatically translate natural language questions into executable SQL queries, thereby enabling non-experts to interact with relational databases. The advent of large language models (LLMs) has significantly advanced this task (Zhang et al., 2024; Zhu et al., 2024), with models like GPT-3.5/4, Gemini, and their open-source variants demonstrating impressive performance across benchmarks. However, as schema size increases, providing the entire schema as input may exceed the model's context window, especially in large-scale databases. Even when using recent LLMs with extended context lengths, supplying the full schema can introduce noise and hinder the model's ability to focus on relevant elements.

2.1 Schema Linking in Text-to-SQL

Schema linking—the process of aligning natural language mentions to corresponding tables and columns in a database—is a crucial component of Text-to-SQL systems (Lei et al., 2020; Liu et al., 2022; Li et al., 2023c). Early approaches relied on exact string matching or type-based heuristics (Yu et al., 2018), which struggled with synonyms, paraphrases, and complex cross-domain schemas. Recent methods have increasingly leveraged pretrained LLMs and neural encoders to improve linking accuracy (Gan et al., 2023; Glass et al., 2025). Schema linking has proven particularly important for LLM pipelines that operate on large or multi-database environments, where prompt space is limited and precision in schema filtering directly affects SQL generation quality (Cao et al., 2024; Liu et al., 2025).

2.2 Neural and Prompt-Based Linking Strategies

Numerous methods have been proposed to handle schema linking within LLM-based Text-to-SQL systems. Some decouple schema linking as a separate module before SQL generation (Pourreza and

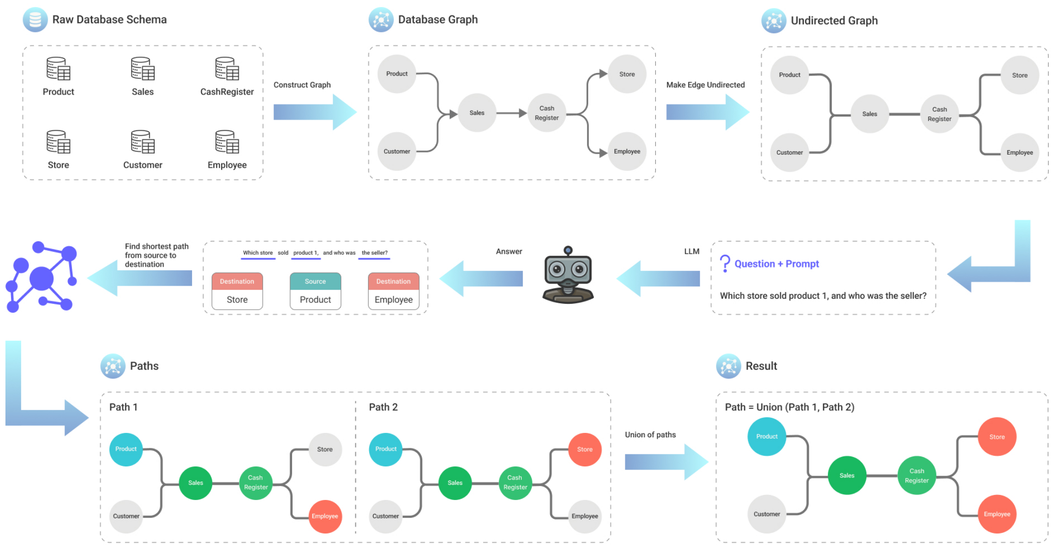

Figure 1: Overview of our graph-based schema linking pipeline.

Rafiei, 2024; Li et al., 2023a), while others incorporate schema selection as a prompt-driven or retrieval-augmented step (Wang and Liu, 2025). Extractive methods, such as Glass et al. (2025), directly prompt LLMs to list relevant schema items, trading generation flexibility for interpretability and control. RSL-SQL (Cao et al., 2024) proposes a bidirectional pruning mechanism with self-correction to boost recall, while Solid-SQL (Liu et al., 2025) augments training data to improve linking robustness. Despite variations in architecture, a common trend across these systems is the effort to balance schema coverage (recall) with relevance filtering (precision) to avoid overloading the LLM or omitting critical elements.

2.3 Graph-Based Approaches for Schema Linking

A parallel line of work models the database schema as a graph structure, where tables and columns are nodes, and foreign-key or semantic relations form edges. These methods primarily leverage graph neural networks (GNNs) or relation-aware transformers to propagate information across schema components. RAT-SQL (Wang et al., 2020) pioneered relation-aware attention over a joint question–schema graph, inspiring successors such as LGESQL (Cao et al., 2021) (line-graph encoding of meta-relations) and ShadowGNN (Chen et al., 2021) (delexicalised projection for cross-schema generalisation). Later hybrids integrate graph reasoning directly into pretrained LMs, e.g. Graphix-T5 (Li et al., 2023b) and GRL-SQL (Gong and Sun, 2024). Most recently, SQLformer (Bazaga et al., 2024) embeds schema structure as inductive bias in a Transformer encoder and autoregressively generates SQL ASTs as graphs. While graph-enhanced models capture rich global relations, they typically require substantial fine-tuning or architectural changes—an obstacle in low-resource, real-time deployments. Graph-based schema linking methods have recently declined in popularity as LLM-driven approaches have become dominant.

2.4 Classical Graph Algorithms in Schema Linking

In contrast to learned graph encoders, only a handful of systems reuse *classical* graph algorithms to aid LLMs. **DBCopilot** (Wang et al., 2025) constructs a directed schema graph and performs depth-first traversal to linearise the sub-schema passed to a lightweight "router" model. **Interactive-T2S** (Xiong et al., 2024) equips an LLM agent with a FINDSHORTESTPATH tool that runs breadth-first search over the foreign-key graph to supply valid join chains during multi-turn dialogue. These works demonstrate the practicality of DFS/BFS as auxiliary helpers, but the graph search remains peripheral—responsible only for join validation or routing—rather than serving as the *core* schema-linking engine.

2.5 Positioning Our Work

While prior literature has thoroughly explored neural and graph-enhanced architectures for schema linking, the explicit use of classical graph algorithms—particularly as the *core mechanism* for schema linking in LLM-based Text-to-SQL systems—remains rare. Our approach, SCHEMA-GRAPHSQL, revisits this paradigm by operationalizing schema linking as a path-selection problem on the schema graph. To our knowledge, this is the first work to systematically evaluate and ablate classic path-finding algorithms for schema linking in LLM-driven Text-to-SQL pipelines on real-world benchmarks.

3 Methodology

Notation

Databases. A *relational database* is represented as

$$\mathcal{D} = \langle \mathcal{T}, \mathcal{A}, \mathcal{K} \rangle,$$

where:

- $\mathcal{T} = \{T_1, \ldots, T_n\}$: set of tables.
- $A(T_i)$: attributes (columns) of table T_i; $\mathcal{A} = \bigcup_{T_i \in \mathcal{T}} A(T_i)$ is the global set of attributes.
- $\mathcal{K} \subseteq \mathcal{T} \times \mathcal{T}$: set of foreign key (FK) relations.

The *schema graph* is the undirected graph $G = (\mathcal{T}, \mathcal{K})$, with nodes as tables and edges as FK links. For sparse schemas (fewer than two edges), we further augment the schema graph by adding edges between tables that share a column containing "id" in its name, thus ensuring that the schema graph is sufficiently connected for path enumeration.

Languages.

- $\mathcal{L}$: set of well-formed natural language questions.
- $\mathcal{S}$: set of valid SQL queries.

Given $q \in \mathcal{L}$, the objective is to generate $Q \in \mathcal{S}$ that answers q over $\mathcal{D}$.

This section formalizes the schema linking problem and describes our graph-based, training-free approach for selecting minimal connected sub-schemas to facilitate Text-to-SQL generation. We begin by introducing notation and the problem formulation, then present our graph-based schema linking procedure, and finally detail the configuration space of our approach.

3.1 Problem Formulation

We first introduce the notation used throughout this paper:

Definition 3.1 (Text-to-SQL)**.** Given q and $\mathcal{D}$, **Text-to-SQL** seeks a function

$$\boxed{f_{\text{NL2SQL}} \ : \ \mathcal{L} \times \mathcal{D} \longrightarrow \mathcal{S}}$$

that returns an executable SQL query $Q = f_{\text{NL2SQL}}(q, \mathcal{D})$ that answers the user question q on the database $\mathcal{D}$.

Definition 3.2 (Schema Linking)**.** Let $G = (\mathcal{T}, \mathcal{K})$ be the schema graph of $\mathcal{D}$. **Schema linking** selects a connected sub-schema $S = \langle \mathcal{T}^\star, \mathcal{K}^\star \rangle$ with $\mathcal{T}^\star \subseteq \mathcal{T}$ and $\mathcal{K}^\star \subseteq \mathcal{K}$ sufficient to express the SQL query answering q. Formally,

$$\boxed{g_{\text{SL}} \ : \ \mathcal{L} \times G \longrightarrow \mathcal{P}(\mathcal{T}), \qquad \mathcal{T}^\star = g_{\text{SL}}(q, G)}$$

Here,

$$\boxed{\mathcal{K}^\star = \{(T_i, T_j) \in \mathcal{K} \mid T_i, T_j \in \mathcal{T}^\star\}}$$

The output sub-schema S defines the smallest set of tables and links needed to answer q while remaining connected within the schema graph.

3.2 Graph-Based Schema Linking as Path Selection

Step 1: Extracting Source and Destination Tables. A single LLM call extracts two subsets of tables from the schema:

- $\mathcal{T}_s$ (*sources*): tables whose columns appear in query conditions or filtering predicates;
- $\mathcal{T}_d$ (*destinations*): tables containing the columns requested as output.

Both sets are guaranteed to be non-empty and may overlap, reflecting cases where the same table is used for both filtering and output.

We operationalize schema linking as a path-selection task on the schema graph G, which enables systematic and efficient sub-schema identification:

This extraction is performed via a single call to Gemini 2.5 Flash, guided by a dedicated system prompt designed to elicit precise identification of source and destination tables from the question and schema. The full prompt is shown in Prompt 1.

Prompt 1: System prompt for source and destination extraction

ROLE & OBJECTIVE
You are a senior data engineer who analyses SQL schemas and maps user questions precisely to *source* tables (filtering) and *destination* tables (final result columns).

TASK
Identify:

- **Source table(s) (`src`):** contain columns used in filters/conditions.
- **Destination table(s) (`dst`):** contain columns returned in the answer.

INSTRUCTIONS

1. Internally inspect every table to determine
   - which tables participate in filtering, and
   - which tables supply the requested output columns.

   Briefly justify your choice *internally* but *do not* include that justification in the final answer.
2. Output exactly one line in the following format: `src=TableA,TableB, dst=TableC,TableD`

Step 2: Candidate Path Enumeration. For every pair $(T_s, T_d) \in \mathcal{T}_s \times \mathcal{T}_d$, we enumerate all shortest simple paths connecting them in G:

$$\boxed{\mathrm{SP}(T_s, T_d) = \Big\{ p \;\Big|\; p \text{ is a simple path } T_s \rightsquigarrow T_d,\ |p| = \mathrm{dist}_G(T_s, T_d) \Big\}}$$

This set $\mathrm{SP}(T_s, T_d)$ contains all minimal-length paths in the schema graph between each source and destination table pair.

The global candidate set and their union are defined as:

$$\boxed{\mathcal{C} = \bigcup_{T_s \in \mathcal{T}_s} \bigcup_{T_d \in \mathcal{T}_d} \mathrm{SP}(T_s, T_d), \qquad U = \bigcup_{p \in \mathcal{C}} p}$$

Here, $\mathcal{C}$ enumerates all candidate paths, and U is the union of all tables appearing in any candidate path—representing the maximal connected subgraph that could be relevant for the query.

Step 3: Path Selection and Sub-schema Construction. Depending on the configuration (detailed below), the set U is optionally appended to $\mathcal{C}$. A second LLM call (or a deterministic rule) selects a candidate path $p^\star \in \mathcal{C}$, and we set $\mathcal{T}^\star := p^\star$ as the chosen subset of relevant tables for downstream SQL generation.

3.3 Configurations

To provide flexibility and support empirical analysis, we define a family of selection strategies parameterized by the following flags: let $k_s = |\mathcal{T}_s| > 0$, $k_d = |\mathcal{T}_d| > 0$,

$$\textsc{Longest} \in \{\mathsf{false}, \mathsf{true}\},$$
$$\textsc{Union} \in \{\mathsf{false}, \mathsf{true}\}.$$

Table 3.3 summarizes the seven configurations we evaluate, spanning single-source/single-destination and union-based settings.

#	(k_s, k_d)	LONGEST	UNION
1	$(1,1)$	false	true
2	$(1,*)$	false	true
3	$(*,1)$	false	true
4	$(*,*)$	false	true
5	$(*,*)$	true	true
6	$(*,*)$	false	false
7	$(*,*)$	false	always select U

Here, $*$ means any positive integer. Mode 5 chooses the longest among the shortest paths; Mode 6 excludes U from $\mathcal{C}$; Mode 7 bypasses path selection and deterministically returns the union U. This design enables ablation studies to assess the effect of schema coverage and path selection criteria on final Text-to-SQL accuracy.

3.4 End-to-End Objective

Given configuration Θ, our full pipeline is:

$$\boxed{f^{\Theta}_{\mathrm{NL2SQL}}(q, \mathcal{D}) = h_{\mathrm{GEN}}\big(q,\ g^{\Theta}_{\mathrm{SL}}(q, G)\big)}$$

where g^{Θ}_{SL} is our graph-based schema linker and h_{GEN} is any downstream SQL generator, constrained to use only the filtered schema $\mathcal{T}^\star$. All pipeline steps operate in a single pass, are fully automatic, and require no training data or domain adaptation.

4 Experimental Setup

4.1 Dataset

All experiments are conducted on the BIRD development split, which comprises 1,534 natural-language questions over 11 heterogeneous relational databases. For schema linking precision, recall, and exact match rate, we use the BIRD dev set gold queries by extracting the referenced tables. For execution accuracy, we follow the official evaluation script provided by BIRD without modification.

4.2 Compared Methods

SchemaGraphSQL (Ours) Unless otherwise noted, results correspond to *Mode 7* in Table 3.3, i.e., we deterministically return the union U of all shortest paths connecting the LLM-identified source and destination tables (cf. Section 3.2). The `src/dst` extraction prompt (Prompt 1) is executed using `google/gemini-2.5-flash-preview` at temperature 0.2, while downstream SQL generation is performed at temperature 0.3.

LLM as Schema Linker (Baseline) A single Gemini 2.5 Flash call is prompted to list *all* tables that must appear in the `FROM/JOIN` clause given the user question. This mirrors prior "single-step" schema linking approaches while controlling for model and prompt length.

DENSE RETRIEVER We embed each table name (along with its column names) using the multilingual-E5-large-instruct encoder. For each question, the top-k tables ($k = 1 \dots 6$) retrieved via cosine similarity form the predicted schema.

For completeness, we also include published BIRD dev results from recent schema-linking systems such as Extractive Schema Linking for Text-to-SQL (Glass et al., 2025) and LINKALIGN. (Wang and Liu, 2025) We did *not* re-run these systems; hence, they are excluded from execution accuracy comparisons.

4.3 LLMs for SQL Generation

Following schema filtering, we evaluate four LLMs for SQL generation:

- `google/gemini-2.5-flash-preview`;
- `google/gemma-3-27b-it`;
- `google/gemma-3-12b-it`;
- `google/gemma-3-4b-it`.

All calls are made through the respective provider APIs using identical configurations and prompting templates.

4.4 Evaluation Metrics

Schema-level Metrics. Let G be the gold table set and P the predicted set.

- **Precision:** The percentage of predicted tables that are actually present in the gold SQL query:

$$\text{Precision} = \frac{|P \cap G|}{|P|}$$

- **Recall:** The percentage of gold tables that are successfully predicted:

$$\text{Recall} = \frac{|P \cap G|}{|G|}$$

- F_β **Score:** The generalized F-score that weights recall β times more than precision:

$$F_\beta = \frac{(1+\beta^2)\,|P \cap G|}{\beta^2|G| + |P|}, \qquad \beta \in \{1, 6\}$$

- **Exact Match Rate (EMR):** The percentage of examples where the predicted schema exactly matches the gold schema:

$$\text{EMR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[P_i = G_i]$$

End-to-End Metric *Execution accuracy* is computed using the official BIRD evaluation script: the generated SQL query is executed against the database, and its result must exactly match that of the reference query.

4.5 Implementation Notes

All experiments are conducted via hosted API endpoints; no on-premise hardware is used. Each query incurs (i) one Gemini 2.5 Flash call for schema linking, and (ii) one model call for SQL generation (Gemini2.5 or Gemma3). Code, prompts, and outputs will be released to support reproducibility.

5 Results

5.1 Schema Linking Evaluation

Table 1 shows that our primary configuration, **SchemaGraphSQL$_{\textbf{force-union}}$**, attains **Recall = 95.71 %** and an **F_6=95.43 %** on the BIRD development split—surpassing all published systems, including the previous recall-centric leader ExSL$_f$ (**F_6=93.92 %**). Prior work has argued that recall-weighted metrics such as F_6 are the most reliable indicator of downstream success, because omitting a relevant table is far more damaging than including extras (Glass et al., 2025). By pushing both recall and F_6 to new highs without any supervised training, **SchemaGraphSQL$_{\textbf{force-union}}$** establishes a new performance bar for zero-shot schema linking.

Table 1: Schema Linking Results in Dev Mode

Method	Exact Match Rate (%)	Precision (%)	Recall (%)	F1 (%)	F6 (%)
LLM as Schema Linker	75.88	91.79	89.90	90.83	89.95
Retrieval (Top1)	20.08	86.70	44.46	58.78	45.05
Retrieval (Top2)	26.79	66.59	67.80	67.19	67.77
Retrieval (Top3)	4.63	53.67	80.91	64.54	79.82
Retrieval (Top4)	1.24	45.79	87.64	60.15	85.52
Retrieval (Top5)	1.04	39.89	91.11	55.49	88.06
Retrieval (Top6)	1.04	35.43	93.31	51.36	89.37
DIN-SQL	-	79.90	55.70	65.64	56.16
PET-SQL	-	81.60	64.90	72.30	65.26
MAC-SQL	-	76.30	56.20	64.73	56.60
MCS-SQL	-	79.60	76.90	78.23	76.97
RSL-SQL	-	78.10	77.50	77.80	77.52
LinkAlign Agent	-	77.10	79.40	78.23	79.34
DTS-SQL	-	95.07	92.74	93.89	92.80
Gen	-	90.40	95.50	92.88	95.35
ExSL$_c$	-	95.86	93.94	94.89	93.99
ExSL$_f$	-	**96.35**	93.85	**95.08**	93.92
SchemaGraphSQL$_{1-1}$	71.06	94.89	84.02	89.12	84.28
SchemaGraphSQL$_{force-union}$	**76.60**	86.21	**95.71**	90.71	**95.43**

For users who require a tighter schema, our balanced **SchemaGraphSQL**$_{n\text{-}n}$ variant delivers the best F$_1$ (92.93 %) with only a modest drop in recall (95.10 %). Exact-match rate also improves over the single-step LLM baseline (75.88 %)—rising to 78.29 % for n-n and 76.60 % for force-union—demonstrating that classical graph search repairs connectivity errors that an LLM alone often misses.

5.2 Ablation Insights

The configuration sweep in Table 2 highlights two actionable lessons:

- **Union is essential.** Removing the union step (*no-union*) drops both F$_1$ and EMR, confirming that coverage matters more than compactness.
- **Avoid unnecessary hops.** Forcing the longest path (*force-longest*) harms all metrics, indicating that extra intermediate tables add noise without benefit.

Together, these results validate our design choice: merge all shortest paths for maximum recall, then optionally down-select (e.g., n-n) when higher precision is required.

5.3 End-to-End Execution Accuracy

Table 3 reports execution accuracy for four LLM generators. Across the board, SchemaGraphSQL yields gains of **6–12 %** over the single-step baseline. Using Gemini-2.5-Flash, **SchemaGraphSQL**$_{force-union}$ attains **62.91 %** total accuracy—only 1.5 % short of the oracle "ideal schema linking" setting, implying that most residual errors stem from SQL generation rather than linking.

Table 2: Schema-linking results across graph settings on BIRD-Dev.

Method	EMR (%)	Prec. (%)	Rec. (%)	F1 (%)	F6 (%)
SchemaGraphSQL$_{1-1}$	71.06	**94.89**	84.02	89.12	84.28
SchemaGraphSQL$_{1-n}$	78.16	93.29	91.55	92.41	91.60
SchemaGraphSQL$_{n-1}$	78.23	90.99	94.86	92.89	94.76
SchemaGraphSQL$_{n-n}$	**78.29**	90.87	95.10	**92.93**	94.98
SchemaGraphSQL$_{force\text{-}longest}$	71.64	89.47	88.45	88.96	88.47
SchemaGraphSQL$_{no\text{-}union}$	73.73	91.39	90.03	90.71	90.07
SchemaGraphSQL$_{force\text{-}union}$	76.60	86.21	**95.71**	90.71	**95.43**

Improvements concentrate on the Moderate and Challenging subsets: Gemini-2.5-Flash sees a +15 % boost on challenging questions, reflecting SchemaGraphSQL's advantage on multi-join queries.

For every generator, the high-recall *force-union* variant outperforms the high-precision 1-1 variant on execution accuracy by 2–7 % (Dev) and 4–12 % (MiniDev). This affirms that *omitting* a table is far more damaging than including extras—LLMs can ignore noise but cannot guess missing joins. Among schema metrics, F$_6$ correlates best with

Table 3: SQL Execution Accuracy Results - Dev

LLM	**Method**	**Simple (%)**	**Moderate (%)**	**Challenging (%)**	**Total (%)**
Gemma-3-4B	Ideal Schema Linking	42.49	21.94	16.67	33.83
	Baseline	30.05	13.76	7.64	23.01
	Retrieval	33.51	17.20	13.19	26.66
	SchemaGraphSQL$_{n-n}$	35.46	17.63	12.50	27.90
	SchemaGraphSQL$_{1-1}$	28.76	11.61	8.33	21.64
	SchemaGraphSQL$_{force-union}$	35.35	18.92	20.83	**29.01**
Gemma-3-12B	Ideal Schema Linking	58.38	41.08	29.86	50.46
	Baseline	42.59	22.15	16.67	33.96
	Retrieval	46.38	30.97	27.08	39.90
	SchemaGraphSQL$_{n-n}$	52.00	35.05	27.78	44.59
	SchemaGraphSQL$_{1-1}$	50.59	29.03	23.61	41.53
	SchemaGraphSQL$_{force-union}$	54.38	35.27	26.39	**45.96**
Gemma-3-27B	Ideal Schema Linking	63.14	47.96	38.19	56.19
	Baseline	49.41	31.40	25.69	41.72
	Retrieval	52.22	41.51	33.33	47.20
	SchemaGraphSQL$_{n-n}$	59.68	45.16	34.03	52.87
	SchemaGraphSQL$_{1-1}$	58.38	41.08	31.94	50.65
	SchemaGraphSQL$_{force-union}$	61.19	44.73	37.50	**53.98**
Gemini-2.5-Flash	Ideal Schema Linking	71.46	55.48	47.92	64.41
	Baseline	59.35	41.08	34.72	51.50
	Retrieval	64.11	50.97	45.83	58.41
	SchemaGraphSQL$_{n-n}$	68.22	53.33	44.44	61.47
	SchemaGraphSQL$_{1-1}$	66.81	51.61	43.06	59.97
	SchemaGraphSQL$_{force-union}$	68.32	56.13	50.00	**62.91**

end-to-end success: the highest-F_6 model is invariably the highest-accuracy model, whereas precision alone can be misleading.

5.4 Efficiency

Our pipeline adds negligible latency: one Gemini-Flash call consumes on average 4.6 K input and 14 output tokens, and the subsequent $O(|E|)$ shortest-path search completes in under 15 ms on commodity hardware. Thus SchemaGraphSQL is compatible with real-time database interfaces and low-resource deployments.

6 Conclusion

We have presented SCHEMAGRAPHSQL, a lightweight, zero-shot schema linking framework that integrates classical path-finding algorithms into modern LLM-based Text-to-SQL systems. Unlike prior work that often relies on heavy prompting techniques or supervised fine-tuning, our method outperforms prior work in schema linking with minimal computational overhead. Beyond accuracy gains, SCHEMAGRAPHSQL offers a transparent and interpretable mechanism for schema filtering, making it well-suited for practical deployment in real-world text-to-SQL systems.

Limitation

While SCHEMAGRAPHSQL delivers strong performance on large-scale databases with well-structured foreign key relations, it has several limitations. First, our approach is not optimized for deeply nested or compositional queries that require complex subquery reasoning. Second, on dense schema graphs with excessive or noisy foreign key links, the shortest-path enumeration may yield overly broad candidate sets, affecting precision. Lastly, we treat all join paths equally and do not incorporate heuristics or weights for foreign key importance or estimated join costs, which could further improve path selection and SQL execution quality.

References

Adrián Bazaga, Pietro Liò, and Gos Micklem. 2024. Sqlformer: Deep auto-regressive query graph generation for text-to-sql translation. *Preprint*, arXiv:2310.18376.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555, Online. Association for Computational Linguistics.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: Robust schema linking in text-to-sql generation. *Preprint*, arXiv:2411.00073.

Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai Yu. 2021. ShadowGNN: Graph projection neural network for text-to-SQL parser. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5567–5577, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2023. Re-appraising the schema linking for text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 835–852, Toronto, Canada. Association for Computational Linguistics.

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

Zheng Gong and Ying Sun. 2024. Graph reasoning enhanced language models for text-to-sql. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2447–2451, New York, NY, USA. Association for Computing Machinery.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, Online. Association for Computational Linguistics.

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

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *Preprint*, arXiv:2301.07507.

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. 2023c. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Preprint*, arXiv:2305.03111.

Aiwei Liu, Xuming Hu, Li Lin, and Lijie Wen. 2022. Semantic enhanced text-to-sql parsing via iteratively learning schema linking graph. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 1021–1030, New York, NY, USA. Association for Computing Machinery.

Geling Liu, Yunzhi Tan, Ruichao Zhong, Yuanzhen Xie, Lingchen Zhao, Qian Wang, Bo Hu, and Zang Li. 2024. Solid-sql: Enhanced schema-linking based in-context learning for robust text-to-sql. *Preprint*, arXiv:2412.12522.

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

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *Preprint*, arXiv:2402.01117.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. Dbcopilot: Natural language querying over massive databases via schema routing. *Preprint*, arXiv:2312.03463.

Yihan Wang and Peiyu Liu. 2025. Linkalign: Scalable schema linking for real-world large-scale multi-database text-to-sql. *Preprint*, arXiv:2503.18596.

Guanming Xiong, Junwei Bao, Hongfei Jiang, Yang Song, and Wen Zhao. 2024. Interactive-t2s: Multi-turn interactions for text-to-sql with large language models. *Preprint*, arXiv:2408.11062.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594, New Orleans, Louisiana. Association for Computational Linguistics.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *Preprint*, arXiv:2403.02951.

Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024. Large language model enhanced text-to-sql generation: A survey. *Preprint*, arXiv:2410.06011.

A Prompts

This section includes all system prompts used throughout the SchemaGraphSQL pipeline. These prompts are designed to be modular and reusable across different configurations and model sizes.

- **Prompt 2**: Selection of the most appropriate join path among candidate schema paths.
- **Prompt 3**: SQL query generation using the filtered schema and join path.
- **Prompt 4**: Baseline SQL generation prompt using the full schema without schema linking.

These prompts are issued via Gemini 2.5 Flash with low temperature settings to ensure stability and determinism during inference.

Prompt 2: System prompt for join path selection

ROLE & OBJECTIVE
You are a database expert tasked with selecting the optimal *join path* to answer user questions using a provided SQL schema.

TASK
Choose the *single most appropriate* join path from a list of candidates that correctly connects the relevant tables.

INSTRUCTIONS

1. Internally inspect each path to determine:
   - whether it connects all necessary tables,
   - whether joins are complete and valid,
   - and whether it satisfies the intent of the question.

   Briefly justify your decision *internally* but *do not* include any reasoning in the final output.
2. Output one line in the following format: `Final Answer: path_id: <ID>`

Prompt 3: System prompt for SQLite query generation after schema linking

ROLE & OBJECTIVE
You are an expert in SQLite query generation. Your task is to generate a valid query to answer a user question based on the given schema and join path.

INPUTS

- **Schema:** `{schema}`
- **Join Path:** `{join_path_string}`
- **Question Context:** `{evidence_string}`

INSTRUCTIONS

1. Use the provided schema and join path to construct a valid `SQLite` query.
2. Ensure the query correctly answers the user's question.
3. Format the query clearly and confirm it adheres to SQLite syntax.

Prompt 4: Baseline prompt for SQLite query generation

ROLE & OBJECTIVE
You are an expert in SQLite query generation. Your task is to produce a valid query that answers a user's question using the provided schema.

INPUTS

- **Schema:** `{schema}`
- **Question Context:** `{evidence_string}`

INSTRUCTIONS

1. Generate a correct `SQLite` query that answers the user question.
2. Ensure the query is syntactically valid and aligns with the schema.
3. Format the query clearly and cleanly.

B Additional Results

This section presents extended evaluation results that complement those in the main text. We report schema linking scores and execution accuracy on the MINIDEV split of the BIRD dataset to validate robustness and generalization.

- **Table 4**: Comparison of schema linking methods on MiniDev, including LLM baselines, dense retrievers, and SchemaGraphSQL.
- **Table 5**: SchemaGraphSQL ablation results across different graph configurations on MiniDev.
- **Table 6**: End-to-end SQL execution accuracy for all models and schema linking variants on MiniDev, broken down by question difficulty.

These extended results reinforce the strong recall and execution performance of SchemaGraphSQL, especially on complex and multi-table SQL queries.

Table 4: Schema Linking Results in MiniDev Dataset

Method	**Exact Match Rate (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**F6 (%)**
LLM as Schema Linker	75.70	92.82	90.56	91.68	90.62
Retrieval (Top1)	14.40	86.40	41.24	55.83	41.83
Retrieval (Top2)	28.00	68.30	64.67	66.43	64.76
Retrieval (Top3)	4.80	55.00	77.73	64.42	76.88
Retrieval (Top4)	1.00	47.29	85.00	60.77	83.20
Retrieval (Top5)	0.80	41.52	89.64	56.75	86.92
Retrieval (Top6)	0.80	37.06	92.26	52.87	88.69
SchemaGraphSQL (Ours)	82.33	94.80	**93.97**	**94.38**	**93.99**

Table 5: Schema Linking Results Across Different Graph Settings (Minidev)

Method	**Exact Match Rate (%)**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**F6 (%)**
SchemaGraphSQL$_{1-1}$	64.86	**96.47**	79.67	87.27	80.05
SchemaGraphSQL$_{1-n}$	74.10	95.93	87.16	91.34	87.38
SchemaGraphSQL$_{n-1}$	82.13	95.81	93.39	**94.58**	93.45
SchemaGraphSQL$_{n-n}$	**82.33**	94.80	93.97	94.38	93.99
SchemaGraphSQL$_{force-longest}$	72.29	92.97	86.19	89.45	86.36
SchemaGraphSQL$_{no-union}$	74.90	95.16	87.94	91.41	88.12
SchemaGraphSQL$_{force-union}$	80.72	89.36	**94.75**	91.97	**94.59**

Table 6: SQL Execution Accuracy Results - MiniDev

LLM	**Method**	**Simple (%)**	**Moderate (%)**	**Challenging (%)**	**Total (%)**
Gemma-3-4B	Ideal Schema Linking	47.97	21.37	18.63	28.71
	Baseline	32.43	10.08	6.86	16.06
	Retrieval	36.49	18.55	13.73	22.89
	SchemaGraphSQL$_{n-n}$	42.57	18.15	12.75	24.3
	SchemaGraphSQL$_{1-1}$	31.08	10.08	6.86	15.66
	SchemaGraphSQL$_{force-union}$	41.89	18.95	15.69	**25.1**
Gemma-3-12B	Ideal Schema Linking	63.51	45.56	34.31	48.59
	Baseline	38.51	18.95	16.67	24.3
	Retrieval	50.68	35.08	28.43	38.35
	SchemaGraphSQL$_{n-n}$	57.43	37.5	30.39	41.97
	SchemaGraphSQL$_{1-1}$	54.73	29.03	23.53	35.54
	SchemaGraphSQL$_{force-union}$	60.14	42.34	33.33	**45.78**
Gemma-3-27B	Ideal Schema Linking	72.97	53.63	43.14	57.23
	Baseline	50.0	27.82	21.57	33.13
	Retrieval	60.81	44.76	36.27	47.79
	SchemaGraphSQL$_{n-n}$	66.22	50.81	35.29	52.21
	SchemaGraphSQL$_{1-1}$	61.49	38.71	27.45	43.17
	SchemaGraphSQL$_{force-union}$	68.92	52.02	44.12	**55.42**
Gemini-2.5-Flash	Ideal Schema Linking	83.78	66.13	56.86	69.48
	Baseline	58.78	43.95	36.27	46.79
	Retrieval	75.0	53.63	53.92	60.04
	SchemaGraphSQL$_{n-n}$	77.03	58.87	50.98	62.65
	SchemaGraphSQL$_{1-1}$	76.35	56.85	41.18	59.44
	SchemaGraphSQL$_{force-union}$	77.7	62.5	50.98	**64.66**