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

AmirHossein Safdarian[1], Milad Mohammadi[1], Ehsan Jahanbakhsh[1], Mona Shahamat Naderi[2], Heshaam Faili[1]
[1]University of Tehran, Iran [2]Sharif University of Technology, Iran
`{a.safdarian, miladmohammadi, ehsan.jahanbakhsh, hfaili}@ut.ac.ir, mona.shahamat@sharif.edu`

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 a lightweight LLM to extract source and destination tables from the user query, followed by applying classical pathfinding 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. To handle real-world databases where foreign keys may be missing or inconsistent, we further propose an LLM-guided joinability discovery step that infers table connections before graph construction, ensuring robustness across diverse schemas. Despite being simple, cost-effective, and highly scalable, our method achieves state-of-the-art results on both the BIRD and Spider 2.0 benchmarks, outperforming previous specialized, fine-tuned, and complex multi-step LLM-based approaches.

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.

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 (Wang et al., 2025d), and prompt-based systems (Pourreza and Rafiei, 2023). 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., 2025; Wang et al., 2025b).

We address whether effective schema linking can be achieved without specialized fine-tuning or complex multi-stage prompting. Our answer is affirmative, based on a minimal design that uses one lightweight LLM decision plus deterministic graph search.

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; when foreign keys are missing or inconsistent, we first use an LLM, given the schema, to infer joinability

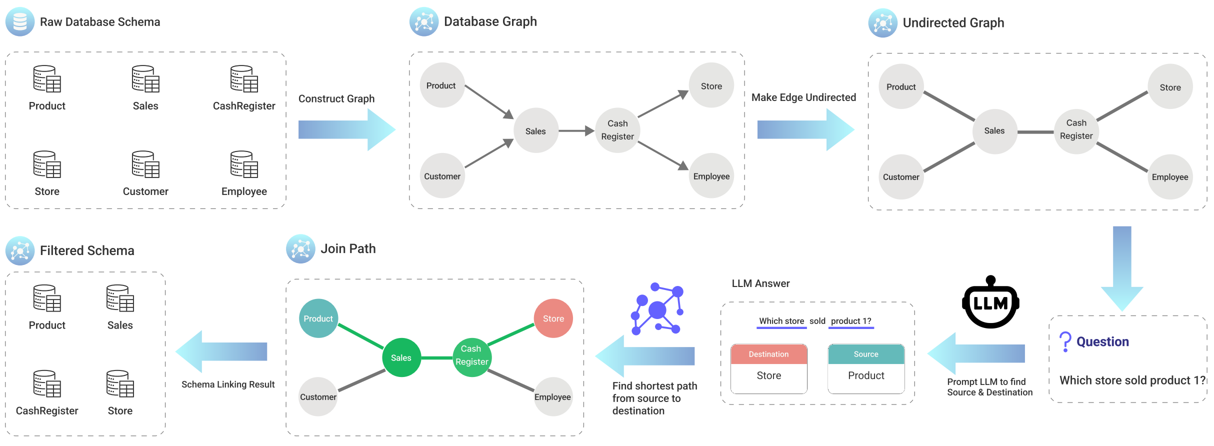

Figure 1: SchemaGraphSQL pipeline. Main flow (1–1): (i) build a table graph from foreign keys or, when FKs are missing, via LLM-based joinability discovery (Section 2); (ii) identify one source and one destination table with a single LLM call; (iii) find a shortest join path (Dijkstra-style) and pass only the filtered tables on that path to the SQL generator.

and construct the graph. 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 BIRD (Li et al., 2023c) and Spider 2.0 Lite (Lei et al., 2025) show that SchemaGraphSQL attains state-of-the-art recall-focused schema-linking scores and improves end-to-end execution accuracy across multiple model families. We also provide a systematic analysis demonstrating that this minimal design (one small LLM decision plus graph search) compares favorably to specialized neural or prompt-heavy pipelines in robustness and efficiency.

Main Contributions:
(1) We present a training-free, efficient schema linking framework that couples one lightweight LLM decision with deterministic graph search, delivering strong end-to-end accuracy with minimal computational overhead (on average only ~4.6K input and 14 output tokens per query).
(2) We propose a robust graph-construction procedure: directly from foreign keys when available, and via LLM-guided joinability discovery when keys are missing or noisy, enabling reliable operation across heterogeneous schemas.
(3) We conduct comprehensive evaluations and ablations on BIRD and Spider 2.0 Lite, showing that recall-oriented linking correlates best with execution accuracy and that our approach consistently improves downstream performance.

2 SchemaGraphSQL

SchemaGraphSQL is a training-free schema linker that selects a small, connected subset of a database schema sufficient to answer a user question. We model the database as a graph where *tables are nodes* and *foreign keys (FKs) are edges*. Given the question, we (i) identify a *source* table (where conditions apply) and a *destination* table (where requested outputs live), then (ii) find a shortest join path between them. The tables on this path form the filtered sub-schema that we pass to the downstream Text-to-SQL generator (Figure 1).

2.1 Step 1: Build the schema graph

In this step our goal is to construct a table-level graph that accurately reflects the joinability structure of the database, so shortest-path search corresponds to minimal-hop join chains.

In databases with declared FKs (e.g., BIRD), we build an undirected graph whose nodes are tables and whose edges reflect FK relationships. This preserves the natural connectivity of the schema and allows shortest-path search over valid joins.

In some collections (e.g., SPIDER 2), explicit FK metadata may be missing or incomplete. In these cases, we use an LLM-guided *joinability discovery* step: given the schema (table and column names, types, and brief descriptions when available), the LLM proposes joinable table pairs and candidate key columns. We add an edge for each validated pair to obtain the table graph. Further details are provided in the Appendix. Overall, this makes graph construction reliable across heterogeneous, large-scale databases.

2.2 Step 2: Identify sources and destinations

We ask a lightweight model to return exactly *one* source table and *one* destination table (see the prompt 2 in Appendix A). By **source** we mean the table whose columns appear in the query's filters/conditions (e.g., predicates in `WHERE`/`HAVING`). By **destination** we mean the table that supplies the requested output columns (e.g., attributes shown in `SELECT`).

This choice mirrors common practice when writing SQL over large schemas: practitioners typically first locate where to filter and where to read the final outputs, then connect them with the necessary joins. A single, tightly scoped call gives precise anchors for the subsequent graph search without training or multi-stage prompting.

2.3 Step 3: Shortest-path backbone

Given the source and destination from Step 2, we run a Dijkstra-style shortest-path algorithm on the table graph to obtain a minimal-hop join chain between them. The tables on this path constitute the filtered sub-schema used by the SQL generator. This yields a compact, connected context that preserves the joins necessary to answer the question without overwhelming the model with irrelevant tables.

If multiple shortest paths exist between the chosen source and destination, we issue a second, tiny LLM call to select the most appropriate path (based on path semantics and column names). On the BIRD-Dev set this occurred in only *1.7%* of examples.

2.4 Force-Union configuration (primary, recall-oriented)

While we describe 1–1 above for clarity, our primary configuration in experiments is a recall-oriented force-union variant. Here, the LLM returns lists of sources and destinations instead of a single source and destination. We then take the union of shortest paths across all source–destination pairs, forming a connected join backbone that maximizes coverage (details in Section 4). This configuration consistently achieved the highest recall and the strongest end-to-end execution accuracy on complex, multi-join queries. In the ablation study (Section 6), we further analyze different configuration settings and examine how varying the number of sources, destinations, and path-selection strategies affects the precision–recall balance. Code, prompts, and outputs will be released to support reproducibility.

3 Experimental Setup

3.1 Datasets

We evaluate on two recent Text-to-SQL benchmarks that differ in schema annotation and structure, enabling a robust assessment across settings.

- **BIRD (Dev)** The BIRD (Li et al., 2023c) development split contains 1,534 natural-language questions over 11 heterogeneous relational databases. Gold SQL is available for each question. We use it to compute schema-level metrics by extracting referenced tables, and to compute execution accuracy with the official BIRD evaluation.
- **Spider 2.0 (Lite)** Spider 2.0 focuses on real-world enterprise workflows (Lei et al., 2025). We use the Lite test subset with 547 test examples across 158 test databases. Unlike BIRD, foreign keys may be missing or incomplete; this lets us assess robustness of our graph construction (Section 2) and schema linking under weaker supervision. Schema-level metrics are computed from gold SQL by extracting referenced tables; execution accuracy follows the canonical execute-and-compare protocol.

BIRD and Spider 2.0 are among the most up-to-date benchmarks and are complementary: BIRD provides clean FK annotations; Spider 2.0 Lite intentionally stresses more realistic, partially specified schemas. Using both ensures the proposed method is tested under different schema conditions.

3.2 Compared Methods

- **SchemaGraphSQL (ours)** We evaluate two configurations: **SchemaGraphSQL$_{\textbf{force-union}}$**, our primary and recall-oriented method, asks the

Table 1: Schema linking results on BIRD-Dev. Prompt-based methods, including ours, use Gemini 2.5 Flash for source/destination extraction; retrieval uses a multilingual-E5-large-instruct table encoder. Fine-tuned rows (e.g., ExSL, DTS-SQL) are reported from their papers and were not re-run. Metrics compare predicted tables to tables referenced in the gold SQL.

Method	**Schema Linking Method**	**Precision (%)**	**Recall (%)**	**F1 (%)**	**F6 (%)**
LLM as Schema Linker	Prompt-based	91.79	89.90	90.83	89.95
Retrieval (Top1)	Retrieval-based	86.70	44.46	58.78	45.05
Retrieval (Top2)	Retrieval-based	66.59	67.80	67.19	67.77
Retrieval (Top3)	Retrieval-based	53.67	80.91	64.54	79.82
Retrieval (Top4)	Retrieval-based	45.79	87.64	60.15	85.52
Retrieval (Top5)	Retrieval-based	39.89	91.11	55.49	88.06
Retrieval (Top6)	Retrieval-based	35.43	93.31	51.36	89.37
DIN-SQL (Pourreza and Rafiei, 2023)	Prompt-based	79.90	55.70	65.64	56.16
PET-SQL (Li et al., 2024)	Prompt-based	81.60	64.90	72.30	65.26
MAC-SQL (Wang et al., 2025a)	Prompt-based	76.30	56.20	64.73	56.60
MCS-SQL (Lee et al., 2025)	Prompt-based	79.60	76.90	78.23	76.97
RSL-SQL (Cao et al., 2024)	Prompt-based	78.10	77.50	77.80	77.52
LinkAlign Agent (Wang et al., 2025d)	Prompt-based	77.10	79.40	78.23	79.34
DTS-SQL (Pourreza and Rafiei, 2024)	Fine-tuned	95.07	92.74	93.89	92.80
Gen (Glass et al., 2025)	Fine-tuned	90.40	95.50	92.88	95.35
ExSL$_c$ (Glass et al., 2025)	Fine-tuned	95.86	93.94	94.89	93.99
ExSL$_f$ (Glass et al., 2025)	Fine-tuned	**96.35**	93.85	**95.08**	93.92
SchemaGraphSQL$_{1-1}$	Prompt + Graph	94.89	84.02	89.12	84.28
SchemaGraphSQL$_{force-union}$	Prompt + Graph	86.21	**95.71**	90.71	**95.43**

LLM for lists of sources and destinations and returns the union of shortest paths over all source–destination pairs. **SchemaGraphSQL$_{1\text{-}1}$** is a compact variant that asks the LLM for exactly one source and one destination and uses a single shortest path between them (with a rare tie-break via a second small LLM call when multiple shortest paths exist).

- **LLM as Schema Linker (baseline)** A single LLM call is prompted to list all tables that should appear in the `FROM`/`JOIN` clause given the question, mirroring prior single-step linkers.

- **Dense Retriever (baseline)** A table retriever encodes table names (optionally concatenated with column names) and selects the top-k tables by cosine similarity; we report results for $k \in \{1, \ldots, 6\}$.

- **Published systems** For completeness, we include published BIRD development results from recent schema-linking systems (e.g., Extractive Schema Linking; LinkAlign). We do not re-run these systems, so they are excluded from execution-accuracy comparisons.

3.3 Models evaluated

We evaluate end-to-end performance with several LLM families: Gemini 2.5 Flash (Comanici et al., 2025), Gemma 3 (27B, 12B, and 4B instruction-tuned variants) (Team et al., 2025), and GPT-5 nano / GPT-5 mini (OpenAI, 2025). For each family, end-to-end accuracy is computed by using that same model for the entire pipeline: the model identifies sources/destinations (schema linking) and then generates SQL (no cross-model mixing).

3.4 Evaluation Metrics

Schema-level: We report *precision*, *recall*, and F-scores, calculated by comparing the predicted table set against the tables referenced in the gold SQL. We prioritize the recall-weighted F_6 score ($\beta = 6$), as prior work (Glass et al., 2025) has demonstrated that F_6 correlates most strongly with downstream SQL execution accuracy. We also report *Exact*

Match Rate (EMR), the percentage of examples where the predicted table set exactly matches the gold set:

$$\text{EMR} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[Predicted_i = Gold_i].$$

End-to-end: *Execution accuracy* measures whether the generated SQL, when executed against the database, returns the same result as the reference query.

4 Results

Table 1 shows that **SchemaGraphSQL$_{\textbf{force-union}}$** attains the highest **recall (95.71%)** and F_6 **(95.43%)** among all reported systems, including fine-tuned methods (e.g., ExSL$_f$ at $F_6 = 93.92\%$). This is notable because our method is training-free and uses only a single, lightweight LLM call plus deterministic graph search. As expected, precision is lower than ExSL variants, but in Text-to-SQL the downstream impact is typically dominated by coverage: missing any required table prevents forming a valid join chain, whereas extra tables are usually ignored by the generator. The **SchemaGraphSQL$_{\textbf{1–1}}$** variant exhibits very strong precision (94.89%) with reduced recall (84.02%), illustrating the compactness–coverage trade-off.

On Spider 2.0 Lite (Table 3),the single-step "LLM as linker" baseline shows high precision (89.93%) but *low recall* (40.18%). In contrast, **SchemaGraphSQL$_{\textbf{force-union}}$** increases both recall (47.37%) and F_6 (47.64%), indicating better coverage under FK sparsity. The **1–1** variant again achieves the highest precision (91.17%) but suffers in recall (16.45%), reinforcing that unionizing shortest paths is valuable when a database's join structure is only partially specified.

Across model families (Table 2), **SchemaGraphSQL$_{\textbf{force-union}}$** consistently achieves the strongest non-oracle accuracy. With Gemini 2.5 Flash, total accuracy reaches **62.91%**, closely trailing the oracle "Ideal Schema Linking" (64.41%). Improvements are concentrated in the *Moderate* and *Challenging* categories, where multi-table joins are common; for example, Gemini shows substantial gains on challenging cases. Similar trends hold for Gemma 3 (27B/12B/4B) and GPT-5 (Nano/Mini). In every family, the recall-oriented **force-union** surpasses the compact **1–1** variant on total accuracy. Overall, schema-level F_6 correlates best with end-to-end outcomes, whereas precision alone is less predictive.

5 Analysis

5.1 Cost and Efficiency

We compare the token cost of SchemaGraphSQL against a standard "No Schema Linking" baseline (where the full schema is provided in a single prompt). Using Gemini-2.5-Flash on BIRD-Dev:

- **No Schema Linking:** 1 LLM call per query. Avg Input: 1,892 tokens. Avg Output: 70.2 tokens.
- **SchemaGraphSQL:** 2 LLM calls (Linking + Generation). Avg Input: 6,666 tokens. Avg Output: 84.2 tokens.

While our method increases input tokens (due to the separate linking step), the output tokens (which typically drive API latency and cost) remain comparable (an increase of only $\sim$14 tokens). This input-heavy but output-light pre-processing yields an absolute accuracy gain of +11.4% (51.50% to 62.91%), making it a highly cost-effective trade-off compared to iterative or agentic pipelines.

5.2 Error Analysis

We analyzed the 23.4% of cases where SchemaGraphSQL failed to achieve an Exact Match (EMR) on BIRD-Dev.

Superset vs. Subset: The vast majority of EMR errors (15.7% of total queries) are supersets, where the predicted tables include all gold tables plus extra ones. Only 4.8% are subsets (missing gold tables). This confirms that our force-union strategy effectively maximizes recall, with the primary error mode being slightly lower precision, which powerful LLMs handle well.

Graph Connectivity: Of the missing tables, only 0.27% were unreachable in the graph. However, 4.0% were reachable but lay on *longer* join paths than the shortest one found. This indicates that while the shortest-path heuristic is robust, future work could explore semantic path weighting to capture these edge cases.

6 Ablation Study

We analyze how configuration choices affect schema-level metrics and, by extension, downstream performance. We vary: (i) how many *sources* and *destinations* the LLM may return, and (ii) how we aggregate shortest paths.

Table 2: SQL execution accuracy on BIRD-Dev. Each model family is evaluated end-to-end using the same model for both schema linking (source/destination identification) and SQL generation. “Ideal Schema Linking” supplies the gold table set; “No Schema Linking (LLM-only)” is a single-step LLM listing all join tables; “Retriever-Based Schema Linking” selects top-k tables by embedding similarity. Scores are reported overall and by difficulty buckets.

LLM	Method	Simple (%)	Moderate (%)	Challenging (%)	Total (%)
Gemma-3-4B	Ideal Schema Linking	42.49	21.94	16.67	33.83
	No Schema Linking (LLM-only)	30.05	13.76	7.64	23.01
	Retriever-Based Schema Linking	33.51	17.20	13.19	26.66
	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
	No Schema Linking (LLM-only)	42.59	22.15	16.67	33.96
	Dense Retrieval	46.38	30.97	27.08	39.90
	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
	No Schema Linking (LLM-only)	49.41	31.40	25.69	41.72
	Retriever-Based Schema Linking	52.22	41.51	33.33	47.20
	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
	No Schema Linking (LLM-only)	59.35	41.08	34.72	51.50
	Retriever-Based Schema Linking	64.11	50.97	45.83	58.41
	SchemaGraphSQL$_{1-1}$	66.81	51.61	43.06	59.97
	SchemaGraphSQL$_{force-union}$	68.32	56.13	50.00	**62.91**
GPT-5-nano	Ideal Schema Linking	50.16	32.47	25.00	42.44
	No Schema Linking (LLM-only)	43.78	23.87	24.31	35.92
	Retriever-Based Schema Linking	44.00	30.11	18.75	37.42
	SchemaGraphSQL$_{1-1}$	45.84	27.10	21.53	37.87
	SchemaGraphSQL$_{force-union}$	47.35	27.96	20.83	**38.98**
GPT-5-mini	Ideal Schema Linking	59.35	41.29	31.94	51.30
	No Schema Linking (LLM-only)	52.22	31.40	27.08	43.55
	Retriever-Based Schema Linking	52.32	35.27	33.33	45.37
	SchemaGraphSQL$_{1-1}$	54.49	37.85	26.39	46.81
	SchemaGraphSQL$_{force-union}$	57.62	39.78	33.33	**49.93**

We experiment with four source–destination cardinalities: **1–1**, **1–n**, **n–1**, and **n–n**, which control whether the LLM returns single or multiple source and destination tables. For each identified pair, multiple shortest join paths may exist in the schema graph. In the **no-union** setting, the LLM chooses the most appropriate path among them. In the standard configuration for all four cardinalities, we additionally construct a union of all shortest paths and let the LLM choose between individual paths and this union. The **force-longest** variant removes the LLM decision and automatically selects the longest of the shortest paths, and the **force-union** variant always uses the union path directly. These configurations allow us to isolate the effect of path aggregation decisions on schema linking

Table 3: Schema linking results on SPIDER 2.0 *Lite*. Prompt-based methods, including ours, use Gemini 2.5 Flash for source/destination extraction; retrieval uses a multilingual-E5-large-instruct table encoder. Metrics compare predicted tables to tables referenced in the gold SQL.

Method	Ref / Schema Linking Method	Precision (%)	Recall (%)	F1 (%)	F6 (%)
LLM as Schema Linker	Prompt-based	89.93	40.18	**55.55**	40.79
Retrieval (Top1)	Retrieval-based	67.09	9.08	16.00	9.30
Retrieval (Top2)	Retrieval-based	58.00	14.30	22.94	14.59
Retrieval (Top3)	Retrieval-based	51.87	18.56	27.34	18.89
Retrieval (Top4)	Retrieval-based	46.26	21.03	28.91	21.34
Retrieval (Top5)	Retrieval-based	43.22	23.61	30.54	23.91
Retrieval (Top6)	Retrieval-based	40.20	25.70	31.35	25.95
SchemaGraphSQL$_{1-1}$	Prompt + Graph	**91.17**	16.45	27.88	16.83
SchemaGraphSQL$_{force-union}$	Prompt + Graph	59.88	**47.37**	52.89	**47.64**

performance.

6.1 Findings on BIRD-Dev

Table 4 reveals three clear lessons:

- **SchemaGraphSQL$_{\textbf{force-union}}$** achieves the best recall (95.71%) and best F_6 (95.43%). Removing union (*no-union*) reduces both F_1 and F_6, indicating that coverage matters more than compactness for downstream success.

- *force-longest* consistently underperforms other settings, suggesting that adding intermediate tables along equally short routes introduces noise without benefits.

- Among non-union settings, **n–n** attains the best F_1 (92.93%) and the highest EMR (78.29%), making it a good choice when compactness is required but we still want strong overall match to gold table sets.

6.2 Findings on Spider 2.0 Lite

Table 5 shows similar, but starker, patterns in FK-sparse environments:

- **force-union** yields the best recall (47.37%) and best F_6 (47.64%), which is crucial when join structure is only partially specified.

- Among non-union settings, **n–1** delivers the highest EMR (35.28%) and competitive F_1 (56.07%), reflecting that multiple candidate sources can help when sources are uncertain.

- **1–1** has excellent precision (91.17%) but low recall (16.45%), confirming that a single anchor on each side is risky without reliable FK metadata.

Table 4: Ablation of SCHEMAGRAPHSQL configurations on BIRD-Dev using Gemini 2.5 Flash.

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

Table 5: Ablation of SCHEMAGRAPHSQL configurations on SPIDER 2.0 *Lite* using Gemini 2.5 Flash.

Method	EMR (%)	Prec. (%)	Rec. (%)	F6 (%)
SchemaGraphSQL$_{1-1}$	20.48	91.17	16.45	16.83
SchemaGraphSQL$_{1-n}$	32.91	77.60	31.84	32.36
SchemaGraphSQL$_{n-1}$	35.28	78.53	43.60	44.13
SchemaGraphSQL$_{n-n}$	31.44	70.09	46.47	46.89
SchemaGraphSQL$_{force\text{-}longest}$	23.22	75.05	18.92	19.31
SchemaGraphSQL$_{no\text{-}union}$	24.31	81.89	19.49	19.90
SchemaGraphSQL$_{force\text{-}union}$	33.27	59.88	47.37	47.64

6.3 Practical guidance

When downstream execution accuracy is the priority (especially for complex, multi-join queries or FK-sparse schemas) use **SchemaGraphSQL$_{\textbf{force-union}}$**. If prompts must remain minimal and you can tolerate recall risk, **SchemaGraphSQL$_{\textbf{1–1}}$** offers the most compact context. For a balanced compromise without union, **n–n** provides the strongest F_1 and EMR on BIRD-Dev.

7 Related Work

7.1 Schema Linking in Text-to-SQL

Schema linking is the process of identifying and aligning the tables and columns in a database schema that are relevant to a natural language query. It is a crucial component of Text-to-SQL systems, as it reduces irrelevant context, improves query generation accuracy, and enables scalability to large schemas (Lei et al., 2020; Li et al., 2023c). Early approaches relied primarily on string matching and type-based heuristics (Yu et al., 2018), which often failed to capture semantic correspondences, paraphrases, or structural relationships across complex schemas. As database size and schema complexity have increased, more sophisticated methods have emerged to bridge the semantic gap between natural language and relational structures.

7.2 Neural and Prompt-Based Methods

Recent work has leveraged pretrained language models and neural encoders to improve schema linking performance (Gan et al., 2023; Glass et al., 2025). Neural linkers treat schema linking as a supervised classification or ranking task and are often integrated as modules within larger Text-to-SQL systems (Pourreza and Rafiei, 2024; Li et al., 2023a). Prompt-based approaches instead query large language models directly to identify relevant schema elements, sometimes incorporating retrieval-augmented generation or multi-step prompting (Wang et al., 2025d). Systems such as RSL-SQL (Cao et al., 2024) and Solid-SQL (Liu et al., 2025) aim to balance recall and precision by pruning irrelevant schema items while preserving necessary context. These methods demonstrate strong performance but can involve complex multi-stage pipelines or require supervised training data.

7.3 Graph-Based Approaches

A parallel line of research models database schemas as graphs, with nodes representing tables or columns and edges representing foreign-key or semantic relations. Graph neural networks (GNNs) and relation-aware transformers have been widely used to encode such structures. RAT-SQL (Wang et al., 2020) introduced relation-aware attention over a joint question–schema graph, inspiring successors such as LGESQL (Cao et al., 2021) and ShadowGNN (Chen et al., 2021). Further work integrates schema graph reasoning directly into pretrained models, as in Graphix-T5 (Li et al., 2023b) and GRL-SQL (Gong and Sun, 2024), while SQL-former (Bazaga et al., 2024) incorporates schema structure as inductive bias for SQL generation.

Beyond learned representations, several studies explore classical graph algorithms for schema reasoning. DBCopilot (Wang et al., 2025b) constructs a directed schema graph and performs depth-first traversal to linearize sub-schemas for routing, and Interactive-T2S (Xiong et al., 2024) employs breadth-first search for join path discovery during multi-turn dialogue. Wang et al. (Wang et al., 2025c) propose a human-guided framework for multi-table question answering that integrates a schema graph to guide LLM reasoning.

7.4 Positioning Our Work

While prior studies have explored schema linking through supervised neural models, prompt-based linkers, and graph-enhanced architectures, less attention has been given to approaches that rely solely on classical graph algorithms without additional training. Existing graph-based methods often treat the schema graph as an auxiliary aid or depend on human-curated structure, leaving the potential of deterministic graph reasoning for schema selection relatively underexplored. Our work investigates this direction by modeling schema linking itself as a graph search problem, leveraging automatically constructed schema graphs to identify relevant tables for Text-to-SQL generation.

8 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 relies on heavy prompting strategies or supervised fine-tuning, our method achieves state-of-the-art schema linking performance with minimal computational overhead. In addition, SCHEMAGRAPHSQL is robust to incomplete or noisy database schemas (where foreign keys may be missing or inconsistent) through an LLM-guided joinability discovery step that reconstructs reliable graph connectivity before path search. Beyond accuracy gains, SCHEMAGRAPHSQL provides a transparent and interpretable mechanism for schema filtering, making it well-suited for real-world deployment in production 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 multi-step subquery reasoning. Second, on dense schema graphs with excessive or noisy foreign key links, shortest-path enumeration may yield overly broad candidate sets, reducing precision. Third, we currently treat all join paths equally and do not apply weights or heuristics based on foreign key importance or estimated join cost, which could further refine path selection and improve SQL execution quality.

In terms of experimental scope, our evaluation relied on hosted API access rather than on-premise hardware, which limited our ability to test a wider range of open-source models or to fine-tune models locally. Nevertheless, we included models from multiple families and sizes (Gemma, Gemini, GPT-5) to provide broad coverage. Future work could extend this by creating a small instruction-tuned model specifically trained for the source–destination identification step, enabling a lightweight, locally deployable variant of SCHEMAGRAPHSQL.

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.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3290 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

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.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353, Abu Dhabi, UAE. Association for Computational Linguistics.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*.

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, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin 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. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, Inc.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *Preprint*, arXiv:2403.09732.

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.

OpenAI. 2025. Gpt-5 system card. Technical report, OpenAI. Technical report. Describes GPT-5 model variants including `gpt-5-mini` and `gpt-5-nano`.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

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

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

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.

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

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

Xixi Wang, Miguel Costa, Jordanka Kovaceva, Shuai Wang, and Francisco C Pereira. 2025c. Plugging schema graph into multi-table qa: A human-guided framework for reducing llm reliance. *arXiv preprint arXiv:2506.04427*.

Yihan Wang, Peiyu Liu, and Xin Yang. 2025d. 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 System Prompts

This section provides the system prompts used throughout the SchemaGraphSQL pipeline. Each prompt corresponds to a specific stage in the schema linking and query generation process. All prompts are issued via Gemini 2.5 Flash with low (0.2) temperature settings to ensure stable and deterministic outputs.

A.1 Source and Destination Table Identification

The first step in the SchemaGraphSQL pipeline is to identify which tables are relevant for answering a question. Specifically, we classify tables as *source* tables (those containing columns used in filters or conditions) and *destination* tables (those providing the final result columns). This step narrows the schema to a focused subset before any join path search is performed. The LLM instructions used for this step are provided below.

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`

While the above prompt is designed for the general case where multiple source and destination tables may be involved, we also consider a simplified **1–1** setting. This configuration is useful for analyzing schema linking behavior in the most constrained scenario, where the LLM must identify exactly one source and one destination table. This reduces ambiguity in evaluation and isolates the core reasoning ability of the model.

Prompt 2: System prompt for source and destination extraction (1–1)

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

INSTRUCTIONS

1. Internally inspect every table to determine
   - which table participates in filtering, and
   - which table supplies 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=TableName, dst=TableName`

A.2 Join Path Selection

Once the relevant tables are identified, the next step is to select the most appropriate join path among candidate paths in the schema graph. This ensures that the resulting query connects the necessary tables correctly and follows valid join logic.

Prompt 3: 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>`

A.3 SQL Generation After Schema Linking

With the join path selected, the system can now generate the final SQL query. At this stage, the LLM uses the reduced schema and the chosen join path to synthesize a syntactically valid and semantically correct SQLite query.

Prompt 4: 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.

A.4 Baseline SQL Generation (Without Schema Linking)

For comparison, we also provide a baseline prompt that generates SQL queries directly from the full schema without schema linking. This serves as the baseline used in our experiments.

Prompt 5: 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

C Extended Results on BIRD MiniDev

This section presents extended evaluation results that complement those in the main text. We report schema linking scores and end-to-end execution accuracy on the MiniDev split of the BIRD dataset, providing a more complete view of SchemaGraphSQL's performance and robustness.

C.1 Schema Linking Performance

We first compare SchemaGraphSQL against baseline approaches for schema linking, including a single-step LLM linker and dense retrieval baselines. As shown in Table 6, SchemaGraphSQL substantially improves recall and F_6 while maintaining high precision, indicating that its deterministic graph-based approach enables more complete and accurate table selection.

Table 6: Schema linking results on the MiniDev split of BIRD. SchemaGraphSQL achieves higher recall and F_6 than both LLM-based and retrieval-based baselines.

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

C.2 Ablation on Graph Configurations

Next, we study how different graph configurations affect schema linking quality. We vary both the source–destination cardinality (**1–1**, **1–n**, **n–1**, **n–n**) and the path aggregation strategy (**no-union**, **force-union**, **force-longest**). Table 7 shows that configurations allowing multiple source and destination tables (**n–1** and **n–n**) yield the best results overall. Additionally, the **force-union** variant achieves the highest recall, confirming that including all shortest-path tables can improve coverage, albeit with a trade-off in precision.

Table 7: Schema linking results across different graph configurations on MiniDev.

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

C.3 End-to-End SQL Execution Accuracy

Finally, we evaluate the impact of schema linking choices on downstream SQL generation. Table 8 reports execution accuracy across question difficulty levels for multiple LLM backends. SchemaGraphSQL consistently outperforms baseline and retrieval-based approaches across all model sizes, and the **force-union** configuration achieves the best overall execution accuracy. This highlights the downstream benefits of improved schema linking quality.

D Robustness on Stronger Baselines

To verify that SchemaGraphSQL provides benefits beyond smaller models, we evaluated performance on the BIRD MiniDev split using significantly stronger baselines.

D.1 Scaling with Larger LLMs

We replaced Gemini-2.5-Flash with **Gemini-2.5-Pro** for both schema linking and SQL generation.

Table 9: Performance on BIRD MiniDev using **Gemini-2.5-Pro**.

Method	EMR	Recall	F6	Exec Acc.
No Schema Linking	72.09	82.88	83.13	55.22
SchemaGraphSQL	**81.33**	**95.62**	**95.46**	**64.46**

Even with a stronger model, SchemaGraphSQL provides a +9.24% gain in end-to-end execution accuracy, proving it serves as a valuable inductive bias regardless of model scale.

D.2 Stronger Dense Retrieval

We compared our method against a retriever using **OpenAI text-embedding-3-large** (Top-K retrieval).

Table 10: Comparison with OpenAI text-embedding-3-large on BIRD MiniDev.

Method	EMR	Recall	F6
Retrieval (Top-1)	15.66	43.29	43.91
Retrieval (Top-6)	0.80	93.09	89.49
SchemaGraphSQL	**81.33**	**95.62**	**95.46**

SchemaGraphSQL outperforms even state-of-the-art dense retrieval embeddings in EMR and F_6 without requiring vector indexing or threshold tuning.

E Graph Construction on Spider 2.0 Lite

Spider 2.0 Lite does not provide declared foreign key metadata, which are essential for building a table-level schema graph. Since SchemaGraphSQL relies on such a graph to reason about table joinability and to enable shortest-path search during schema linking, we infer the missing relationships automatically from the database schema definitions.

Graph construction method: We infer likely foreign key relationships directly from raw SQL DDL statements using a single LLM call. The model examines table and column names and proposes probable references based on naming patterns (e.g., `user_id` likely referencing `users.id`) even when no explicit foreign key is declared. We then apply lightweight plausibility filters to discard clearly invalid edges: candidate edges must satisfy basic type compatibility, follow common key-naming conventions (e.g., `*_id`), and exhibit lexical similarity between referenced table names. The filtered edges are used to build an undirected schema graph whose nodes are tables and whose edges represent inferred joinability. This graph is the sole structural input to SchemaGraphSQL's path-finding stage, which searches for shortest join paths among relevant tables during schema linking.

Prompt for foreign key inference: We use the following system prompt to guide the LLM when inferring foreign key relationships from raw DDL statements:

Prompt 6: System prompt for foreign key inference

You are a database schema analyzer. Your job is to infer likely foreign key relationships from SQL DDL statements that do NOT explicitly define foreign keys. Infer relationships purely from table and column names (e.g., 'user_id' likely references 'users.id').

Instructions:

- Respond with one inferred foreign key per line in the exact format: `source_table.source_column -> target_table.target_column`
- Use lowercase and uppercase table and column names letters exactly as they appear in the DDL.
- Do NOT add explanations, comments, or extra text.
- If no foreign keys can be reasonably inferred, respond with exactly: `NONE`

Validation of graph quality. Although Spider 2.0 Lite lacks gold foreign keys, we vali-

Table 8: SQL execution accuracy on the MiniDev split across different LLMs and schema linking configurations.

LLM	Method	Simple (%)	Moderate (%)	Challenging (%)	Total (%)
Gemma-3-4B	Ideal Schema Linking	47.97	21.37	18.63	28.71
	No Schema Linking (LLM-only)	32.43	10.08	6.86	16.06
	Retriever-Based Schema Linking	36.49	18.55	13.73	22.89
	SchemaGraphSQL$_{n-n}$	42.57	18.15	12.75	24.30
	SchemaGraphSQL$_{1-1}$	31.08	10.08	6.86	15.66
	SchemaGraphSQL$_{force-union}$	41.89	18.95	15.69	**25.10**
Gemma-3-12B	Ideal Schema Linking	63.51	45.56	34.31	48.59
	No Schema Linking (LLM-only)	38.51	18.95	16.67	24.30
	Retriever-Based Schema Linking	50.68	35.08	28.43	38.35
	SchemaGraphSQL$_{n-n}$	57.43	37.50	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
	No Schema Linking (LLM-only)	50.00	27.82	21.57	33.13
	Retriever-Based Schema Linking	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
	No Schema Linking (LLM-only)	58.78	43.95	36.27	46.79
	Retriever-Based Schema Linking	75.00	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$_{force-union}$	77.70	62.50	50.98	**64.66**

date the inferred graphs indirectly through their downstream effect and basic connectivity checks. SchemaGraphSQL's schema linker relies entirely on this inferred graph; if the graph were incomplete or noisy, linking performance would degrade. Instead, SchemaGraphSQL achieves substantially higher recall and F_6 scores than both a single-step LLM baseline and a dense retrieval baseline (Table 3). For example, the force-union variant achieves **47.37%** recall and **47.64%** F_6, compared to **40.18%** and **40.79%** for the baseline. These improvements indicate that the inferred graph captures joinability structure that is useful for accurate table selection.

Additionally, we observe that in **533 out of 547** cases (**97.4%**), a valid path exists between the predicted source and destination tables within the inferred graph. This high connectivity suggests that the inferred structure is sufficient for the path-finding stage to operate reliably in nearly all cases.

F Force-Union Configuration: Extended Pipeline Illustration

Figure 2 illustrates the SchemaGraphSQL pipeline under the *force-union* configuration, which achieved the best F_6 score and overall performance in our experiments.

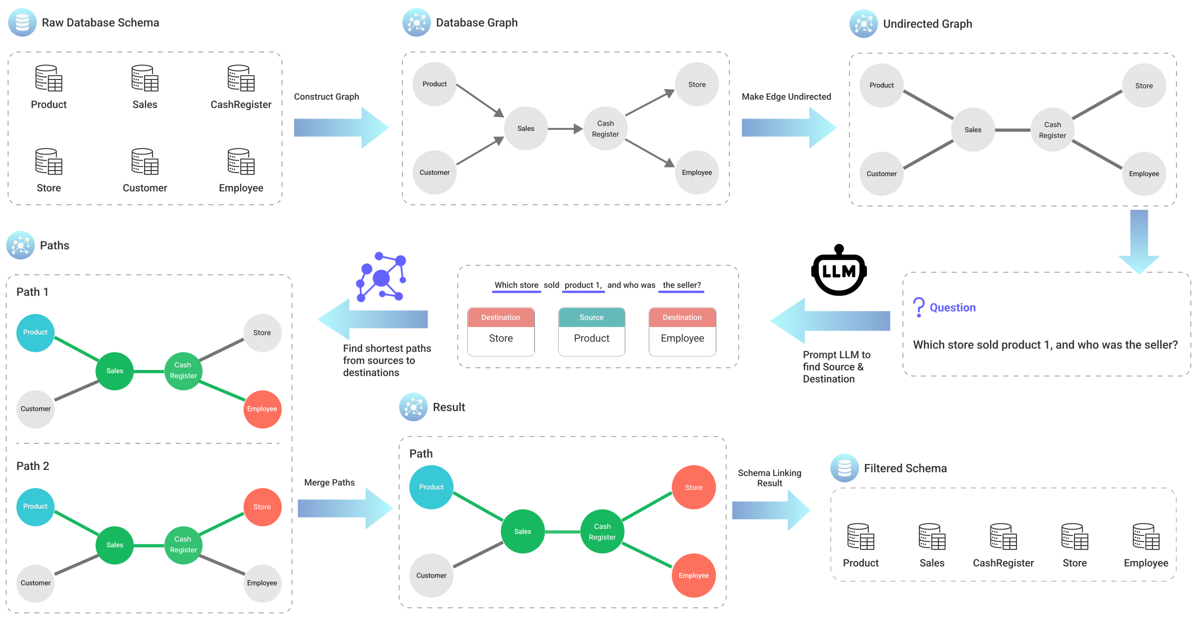

Figure 2: SchemaGraphSQL pipeline under the *force-union* configuration. (i) Build a table graph from foreign keys or, when FKs are missing, via LLM-based joinability discovery (Section 2); (ii) identify multiple source and destination tables with a single LLM call; (iii) compute all shortest join paths and merge them to obtain the filtered sub-schema passed to the SQL generator.