EVILINK: MULTI-PATH SCHEMA LINKING WITH UNCERTAINTY-GUIDED EVIDENCE ACQUISITION FOR LARGE-SCALE TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Schema linking is a difficult and important step in large-scale Text-to-SQL, where systems must identify a compact yet sufficient schema context from large and ambiguous databases. Existing methods often treat schema linking as deterministic selection around a single SQL path, but complex questions may admit multiple valid realizations with different schema needs. We reframe schema linking as uncertainty-aware schema-need inference over multiple plausible SQL paths, where the system distinguishes required schema items from path-dependent uncertain ones and acquires evidence only where needed. We instantiate this reframing with EviLink, which combines multi-hypothesis schema grounding with uncertainty-guided evidence acquisition. Experiments on BIRD-Dev and Spider2-Snow show that this perspective improves the balance among schema completeness, schema relevance, and token cost. On Spider2-Snow, EviLink achieves 93.04% field-level strict recall rate, uses 116.55K average tokens, and improves downstream SQL generation under a fixed generator.

1 INTRODUCTION

Text-to-SQL translates natural language questions into executable SQL queries, making database analysis accessible to non-expert users (Yu et al., 2018; Li et al., 2024a). Despite recent progress driven by large language models (Pourreza & Rafiei, 2023; Gao et al., 2024), complex database scenarios still face a fundamental bottleneck. Before generating SQL, the system must identify a compact yet sufficient schema context (Wang et al., 2025). This step, known as *schema linking*, determines which tables and fields are available to the SQL generator (Wang et al., 2020; Maamari et al., 2024). High-quality schema linking reduces irrelevant schema noise, alleviates context-window pressure, and provides the information basis for faithful SQL generation (Cao et al., 2024). Recent work further suggests that strong schema linking can help a relatively simple agentic backend achieve promising end-to-end Text-to-SQL performance (Wang et al., 2026).

Existing methods usually formulate schema linking as selecting or ranking schema items relevant to a question (Li et al., 2023a; Cao et al., 2024). Earlier methods model schema-question alignment with semantic matching, structured encoders, or learned rankers (Wang et al., 2020; Li et al., 2023a). Recent LLM-based and agentic methods further improve schema selection through SQL-draft-based pruning, contextual augmentation, database retrieval, or schema exploration (Cao et al., 2024; Wang et al., 2025; 2026; Cao et al., 2026). However, as illustrated in Figure 1, many existing formulations still reduce schema linking to selecting a final schema subset under assumptions that can become limiting in complex settings. They tend to reason around a single SQL path, treat schema items as deterministic keep-or-drop decisions, or allocate schema evidence in a largely fixed way.

We argue that these assumptions limit schema linking in complex Text-to-SQL, especially in realistic enterprise-scale benchmarks (Lei et al., 2025). A question may have multiple valid SQL implementations (Bhaskar et al., 2023; Saparina & Lapata, 2024). These implementations can use different join paths, bridge tables, filtering fields, aggregation choices, or intermediate query structures. Since different SQL paths may require different schema elements, linking schema around a single assumed path can prematurely discard tables or fields needed by other valid solutions. Once removed, these elements become unavailable to the downstream SQL generator, causing irreversible

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

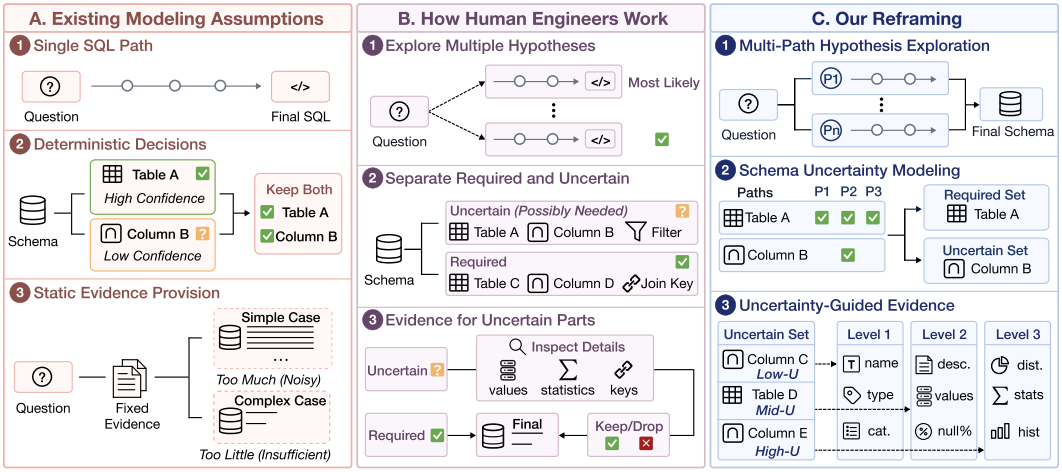


Figure 1: **From single-path schema linking to uncertainty-guided multi-path reasoning.** (A) Many existing formulations rely on a single SQL path, deterministic schema decisions, and static evidence provision. (B) Human engineers instead consider multiple plausible SQL paths, separate required from uncertain schema items, and inspect evidence selectively. (C) We reframe schema linking as uncertainty-guided multi-path reasoning, which preserves multiple solution paths, models schema uncertainty, and uses targeted evidence to resolve uncertain schema needs.

information loss. Schema linking should therefore infer schema needs across plausible SQL paths, rather than merely recover the schema appearing in one reference SQL.

This multi-path view also changes how schema uncertainty and evidence should be handled. Some schema elements are consistently required, some are clearly irrelevant, and others are conditionally needed depending on the SQL path. However, existing formulations often collapse this uncertainty into deterministic keep/drop decisions or fixed ranking thresholds. Static evidence makes this problem worse. Lightweight evidence may be insufficient for difficult cases, while detailed evidence for all schema elements adds redundant context and token cost. For interactive data agents, token efficiency is therefore a practical concern alongside schema completeness. As Figure 1 suggests, a more faithful schema linker should distinguish required and uncertain schema items, and acquire additional evidence only where uncertainty remains.

Motivated by this observation, we introduce **EviLink** to instantiate this uncertainty-aware view of schema linking. EviLink first constructs multiple plausible SQL hypotheses. It then estimates schema support across these hypotheses to separate required schema items from uncertain ones, and allocates stronger evidence only to uncertain items. By jointly modeling path diversity, schema uncertainty, and targeted evidence acquisition, EviLink preserves schema elements needed by alternative SQL paths while avoiding the noise and cost of uniformly exposing detailed evidence.

Our contributions are summarized as follows:

- **Problem reframing.** We reframe schema linking for complex Text-to-SQL as schema-need inference over multiple plausible SQL hypotheses, rather than deterministic selection around a single assumed SQL path. This reframing highlights three limiting assumptions in existing formulations: single-path reasoning, deterministic schema decisions, and static schema evidence.
- **Method instantiation.** Concretely, we instantiate this reframing as EviLink, which combines multi-hypothesis schema grounding, required/uncertain schema bucketing, and uncertainty-guided evidence acquisition to preserve alternative schema needs while controlling irrelevant context.
- **Systematic validation.** We evaluate EviLink on BIRD-Dev and Spider2-Snow, achieving high schema completeness and reducing schema-linking tokens by 49.6% to 79.9% relative to APEX-SQL, with slightly higher execution accuracy.

2 Related Work

Prior schema-linking methods formulate the task as selecting or ranking schema items relevant to a question. Early neural methods model question–schema interactions with structured encoders, semantic matching, rankers, or contextual schema representations (Wang et al., 2020; Cao et al., 2021; Li et al., 2023a). Recent LLM-based Text-to-SQL methods improve schema selection and schema-aware generation through prompting, in-context demonstrations, schema augmentation, or SQL-draft-based refinement (Pourreza & Rafiei, 2023; Dong et al., 2023; Gao et al., 2024; Lee et al., 2025; Qu et al., 2024; Liu et al., 2025; Yang et al., 2024; Cao et al., 2024; Li et al., 2024b; Talaei et al., 2024). More recent systems introduce interaction and exploration into schema linking. LinkAlign (Wang et al., 2025) separates database retrieval from schema item grounding, AutoLink (Wang et al., 2026) performs iterative retrieval, exploration, verification, and expansion, ReFoRCE (Deng et al., 2025) and DSR-SQL (Hao et al., 2025) incorporate field exploration or adaptive context, and APEX-SQL (Cao et al., 2026) uses logical planning and data-grounded verification.

These methods advance schema linking, but they generally use interaction, exploration, or evidence to improve one consolidated schema context. In contrast, EviLink treats schema needs as SQL-path dependent, since different plausible SQL realizations may use different join paths, filters, aggregations, or intermediate structures (Bhaskar et al., 2023; Saparina & Lapata, 2024). It keeps path-wise schema needs explicit before consolidation, estimates cross-path support, separates required from uncertain items, and acquires additional evidence only where uncertainty remains. Overall, EviLink reframes schema linking as uncertainty-aware schema-need inference rather than single-context schema selection.

3 Methodology

Task definition. Given a natural language question q and a database schema $\mathcal{S}$, schema linking aims to identify a subset of schema items $\hat{\mathcal{S}} \subseteq \mathcal{S}$ as the schema context for downstream SQL generation. In this work, schema items include both tables and fields. The objective is to preserve items that may be required for producing a valid SQL query, while filtering out clearly irrelevant ones. Unlike SQL generation, schema linking does not produce an executable query; instead, it determines which parts of the schema are made available to the SQL generator.

Overview. Figure 2 shows the workflow of EviLink. The method proceeds in two stages. First, *multi-hypothesis schema grounding* (Figure 2, A–C) explores multiple SQL paths for the question instead of committing to a single path early. Each path selects tables and fields under lightweight evidence, and the path-wise selections are consolidated into required, uncertain, and candidate sets. Second, *uncertainty-guided agentic refinement* (UGAR; Figure 2, D–G) focuses the agent on the uncertain set rather than the whole database. The agent acquires stronger profiling or execution evidence as needed, keeps supported items, discards unsupported distractors, and returns a compact schema context for downstream SQL generation.

3.1 Multi-Hypothesis Schema Grounding

The first phase constructs a recall-oriented schema subset for later refinement. Rather than making final keep/drop decisions upfront, EviLink collects tables and fields that may be required by different plausible SQL realizations, reducing the risk of prematurely pruning alternative valid paths.

Multi-hypothesis path construction. We ask the model to generate a set of structured hypotheses $\mathcal{H} = \{h_1, \ldots, h_M\}$, where $M \leq K$ is adaptively chosen and K is the maximum hypothesis budget. Each hypothesis describes one plausible SQL realization of the question, including relevant entities, filters, joins, aggregations, and ambiguous schema interpretations. The hypotheses should be structurally diverse, not superficial paraphrases. They should differ in at least one SQL-relevant dimension, such as join route, filter encoding, aggregation granularity, or temporal interpretation. For simple questions, the model may output a single hypothesis; for ambiguous questions, it may output multiple structurally distinct hypotheses.

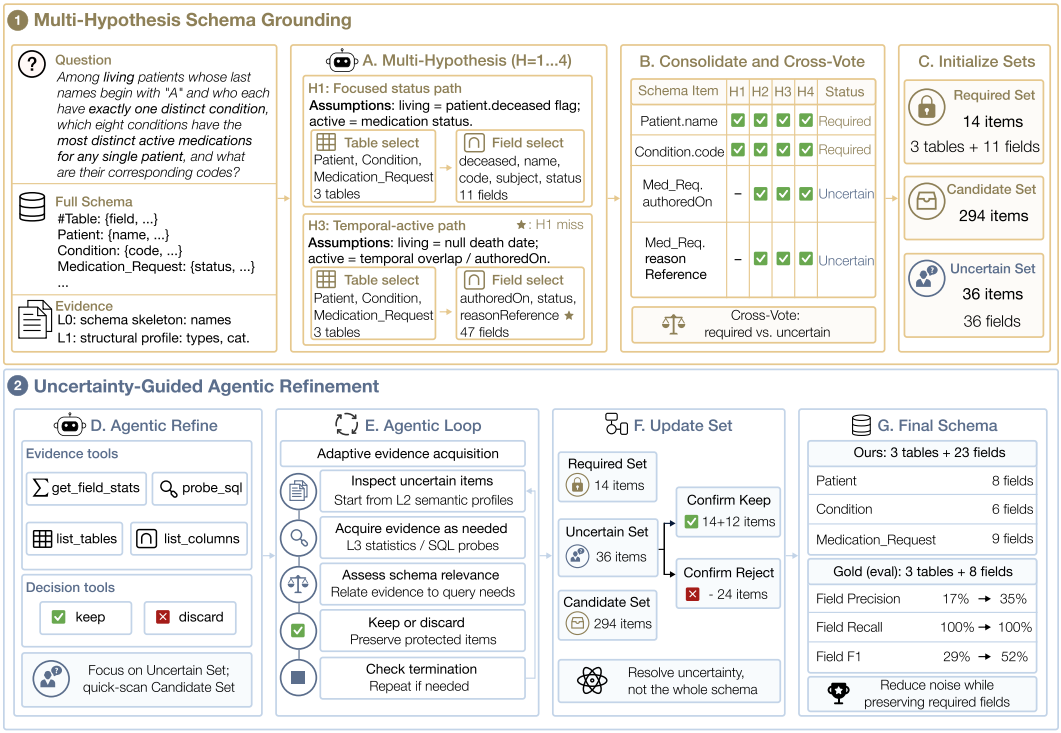


Figure 2: **Overview of EviLink.** The first stage performs multi-hypothesis schema grounding: (A) Multi-Hypothesis proposes multiple plausible SQL paths, (B) Consolidate and Cross-Vote aggregates their schema selections, and (C) Initialize Sets separates required, uncertain, and candidate items. The second stage performs uncertainty-guided agentic refinement: (D) Agentic Refine provides evidence and decision tools, (E) Agentic Loop resolves uncertain items with targeted evidence, (F) Update Set applies keep/discard decisions, and (G) Final Schema returns a compact schema while preserving required fields.

Hypothesis-conditioned schema grounding. For each hypothesis h_i, we run an independent schema grounding path. The path first selects tables from the full catalog and then selects fields from the retained tables:

$$\mathcal{T}_i = \mathrm{SelectTable}(q, h_i, \mathcal{S}), \tag{1}$$

$$\mathcal{F}_i = \mathrm{SelectField}(q, h_i, \mathcal{T}_i). \tag{2}$$

Both selectors are recall-oriented: a table or field is retained if it may be required by any plausible SQL query under the current hypothesis. When uncertain, the selector is instructed to keep the item rather than discard it, which helps preserve bridge tables, join keys, temporal fields, and alternative encodings of the same semantic condition.

To keep grounding affordable on large schemas, table selection is performed over a compact schema view, and field selection is applied only to the retained tables. The hypothesis guides selection but does not act as a whitelist: schema items not explicitly mentioned in the hypothesis can still be selected when compact profiling signals indicate potential relevance.

Cross-voting and set initialization. After grounding all hypotheses, EviLink consolidates the path-wise outputs through cross-voting. Let $\mathcal{S}_i = \mathcal{T}_i \cup \mathcal{F}_i$ denote the schema items selected under hypothesis h_i. For each schema item $x \in \mathcal{S}$, we define its path support as

$$n_x = \left|\{i \in \{1, \ldots, M\} \mid x \in \mathcal{S}_i\}\right|. \tag{3}$$

Items supported by most hypotheses are likely to be required, while items supported by only part of the hypotheses reflect path-dependent uncertainty.

Because the number of generated hypotheses is small, we smooth this support signal with a uniform Beta prior. We model the latent selection tendency of item x as p_x:

$$p_x \mid n_x, M \sim \mathrm{Beta}(n_x + 1, M - n_x + 1), \tag{4}$$

and define its credibility score as

$$c_x = \Pr(p_x > 0.5 \mid n_x, M). \tag{5}$$

We use c_x as a smoothed measure of support within the generated hypothesis set, rather than a calibrated probability that x is necessary for every valid SQL realization. EviLink then initializes three sets. For $M \geq 2$, items with $c_x \geq \tau_{\mathrm{req}}$ are placed into the required set $\mathcal{S}_{\mathrm{req}}$ and locked during refinement, while items with $0 < n_x$ but $c_x < \tau_{\mathrm{req}}$ are placed into the uncertain set $\mathcal{S}_{\mathrm{unc}}$. For $M = 1$, all selected items are instead locked in $\mathcal{S}_{\mathrm{req}}$ to preserve the single grounding path. Items with zero support remain in the candidate set $\mathcal{S}_{\mathrm{cand}}$, which is not refined by default but can still be inspected for recall-oriented recovery. In our implementation, the maximum hypothesis budget is $K = 4$ and $\tau_{\mathrm{req}} = 0.85$; when four hypotheses are generated, an item supported by three hypotheses is still treated as uncertain rather than required. Appendix I analyzes sensitivity to the hypothesis budget K.

3.2 Uncertainty-Guided Agentic Refinement

The grounding stage is intentionally conservative: it preserves schema items that may be needed by at least one plausible SQL path. Refinement therefore focuses on resolving path-dependent uncertainty rather than pruning the full database again.

Evidence hierarchy. EviLink allocates profiling evidence according to schema uncertainty. We organize evidence into four levels:

- **Schema skeleton** (L0) contains table and field names;
- **Structural profile** (L1) augments L0 with field types and coarse semantic categories;
- **Semantic profile** (L2) provides descriptions, representative values, and compact value-level summaries;
- **Statistical profile** (L3) stores additional field statistics when available, such as ranges, summaries, histograms, and distributional signals.

The grounding component uses L0 for broad table selection and L1 for field selection. During refinement, uncertain items are shown with L2 evidence by default, and L3 evidence is acquired only through `get_field_stats` when compact evidence is insufficient. Incomplete or noisy profiles may affect refinement decisions; unresolved uncertain items are retained by default to limit unsupported pruning.

Tool-based refinement. The refinement agent uses two types of tools. Evidence tools are read-only: `list_tables` and `list_columns` retrieve schema context for uncertain items or candidate recovery, `get_field_stats` upgrades selected fields to L3 and returns available evidence, and `probe_sql` tests whether a hypothesized schema usage is feasible. Decision tools update the refinement state, where `keep` confirms an uncertain or recovered candidate item and `discard` removes an uncertain item with an explicit reason.

Schema items can move between sets through three updates:

$$S_{\mathrm{unc}} \xrightarrow{\texttt{keep}} S_{\mathrm{keep}}, \qquad S_{\mathrm{unc}} \xrightarrow{\texttt{discard}} S_{\mathrm{drop}}, \qquad S_{\mathrm{cand}} \xrightarrow{\texttt{keep}} S_{\mathrm{keep}}. \tag{6}$$

Here, S_{keep} and S_{drop} contain items confirmed as kept and discarded, respectively. Evidence acquisition remains read-only, while schema updates follow agent decisions and predefined protection and recovery rules.

Convergence and final schema. The uncertain-item loop terminates when the uncertain set becomes empty, when the agent stops requesting tools, or when its turn budget is reached; candidate recovery is handled separately. Let $\mathcal{S}_{\mathrm{keep}}$ and $\mathcal{S}_{\mathrm{drop}}$ denote the items confirmed as kept and discarded, respectively, and let $\mathcal{S}_{\mathrm{pending}}$ denote uncertain items that remain unresolved at termination.

To preserve the recall-first property of the grounding stage, unresolved uncertain items are kept by default. The linked schema returned to the downstream SQL generator is

$$\hat{\mathcal{S}} = \mathcal{S}_{\text{req}} \cup \mathcal{S}_{\text{keep}} \cup \mathcal{S}_{\text{pending}}. \tag{7}$$

Items in $\mathcal{S}_{\text{drop}}$ are excluded from the final schema. This fallback does not override the agent's decisions; it only specifies how unresolved uncertainty is handled under the turn budget. As a result, refinement improves precision by rejecting unsupported uncertain items while avoiding aggressive pruning of schema items that may still be required by a valid SQL path.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate our method on two widely used Text-to-SQL benchmarks, BIRD-Dev (Li et al., 2023b) and Spider2-Snow (Lei et al., 2025). BIRD-Dev provides a broad testbed for database question answering, while Spider2-Snow targets enterprise-level workflows with greater challenges in schema scale, schema ambiguity, and realistic data access. Together, these benchmarks enable evaluation across different database scales and task complexities. We conduct detailed experiments on both benchmarks to assess schema completeness, schema relevance, and token cost, as well as the impact on downstream SQL generation.

Baselines. We compare against seven recent Text-to-SQL and schema-linking baselines: TA-SQL (Qu et al., 2024), RSL-SQL (Cao et al., 2024), ReFoRCE (Deng et al., 2025), LinkAlign (Wang et al., 2025), AutoLink (Wang et al., 2026), DSR-SQL (Hao et al., 2025), and APEX-SQL (Cao et al., 2026). These baselines span methods centered on schema linking and broader Text-to-SQL systems that explicitly model schema usage through linking, pruning, or exploration modules. For a fair and reproducible comparison, we adapt the official implementations only for input formatting, a unified model interface, and output normalization for evaluation. We preserve their method-defining prompts, decision procedures, retrieval and selection logic, and thresholds. Further details on the baselines and comparison scope are provided in Appendix A.

Models. All methods use the same backbone within each benchmark for the main schema-linking comparison: Qwen3.6-35B-A3B on BIRD-Dev and GLM-5.1 on Spider2-Snow. Schema-linking ablations use Qwen3.8-27B on both benchmarks. Downstream SQL generation uses Qwen3.8-27B on BIRD-Dev and Qwen3.6-27B on Spider2-Snow, with generators fixed across methods within each benchmark and schema contexts taken from the corresponding main linking runs. Appendix G reports an additional backbone comparison. Detailed parameter settings are provided in Appendix D.

Evaluation metrics. Following RSL-SQL (Cao et al., 2024) and APEX-SQL (Cao et al., 2026), we evaluate schema linking as a retrieval problem using four complementary metrics: **Strict Recall Rate (SRR)**, **Non-Strict Recall (NSR)**, **Non-Strict Precision (NSP)**, and **Non-Strict F1-score (NSF)**. To ensure reliable evaluation, we derive gold tables and fields from reference SQL through abstract syntax tree parsing and multi-agent cross-checking on both benchmarks, with additional reference SQL construction, execution verification, and targeted human SQL review for Spider2-Snow; details are provided in Appendix B. These metrics can be computed with either tables or fields as retrieval units, and the exact computation procedure is provided in Appendix C. All four metrics are reported as percentages. In the main experiments, we report field-level results by default because field-level evaluation is more fine-grained and stricter; table-level results are included in Appendix E for reference. **SRR** measures strict schema completeness: an example is counted as correct only when all gold units at the chosen granularity are retained. Following prior work, we treat SRR as the primary recall-oriented metric, since missing any required field can make the correct SQL unreachable. **NSR** measures average schema coverage, i.e., the average fraction of gold units recovered per example. **NSP** measures average predicted-schema relevance, i.e., the average fraction of predicted units that are relevant per example. **NSF** measures the average per-example balance between coverage and relevance. Following prior work, we report **Avg. Tokens** to quantify each method's token cost. We sum API-reported input and output tokens over all LLM calls, separately for schema linking and SQL generation. Prompts, evidence, tool outputs, revisions, and execution

Table 1: Field-level schema linking performance on BIRD-Dev and Spider2-Snow. Token denotes the average per-example token consumption during schema linking. Best results within each dataset are shown in bold. Results on BIRD-Dev are obtained with Qwen3.6-35B-A3B, while results on Spider2-Snow are obtained with GLM-5.1.

Method	**BIRD-Dev** ($N = 1534$)					**Spider2-Snow** ($N = 517$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
TA-SQL	73.73	89.84	**81.91**	**83.98**	**3.38K**	42.94	73.12	**70.02**	**68.13**	157.41K
RSL-SQL	89.90	97.10	44.32	57.89	6.13K	79.50	95.22	24.71	34.48	240.34K
ReFoRCE	20.01	54.61	71.87	58.64	7.29K	40.43	81.91	50.01	55.48	422.47K
LinkAlign	95.11	98.51	10.68	18.05	8.09K	63.44	80.70	10.04	16.00	79.37K
AutoLink	**98.89**	**99.76**	11.39	19.28	51.81K	73.11	82.37	8.00	12.97	**77.93K**
DSR-SQL	79.40	88.14	47.19	58.58	40.01K	72.92	80.88	21.95	30.47	281.08K
APEX-SQL	90.94	97.33	41.97	55.27	50.95K	80.85	95.26	47.50	58.64	580.80K
EviLink	93.22	98.05	51.58	63.60	25.68K	**93.04**	**97.86**	26.04	36.65	116.55K

feedback are counted whenever included in an LLM call. Other cost metrics, such as latency, tool-call counts, and monetary costs, are less directly comparable due to differences in hardware, tool design, and provider pricing.

4.2 Main Results

Table 1 reports the field-level schema linking results on BIRD-Dev and Spider2-Snow. The two benchmarks allow us to examine EviLink under different database scales and complexity levels.

On BIRD-Dev, EviLink achieves a field-level SRR of 93.22%, surpassed only by LinkAlign and AutoLink, while maintaining substantially higher precision than both. LinkAlign and AutoLink achieve SRR scores of 95.11% and 98.89%, respectively, but their NSP scores are only 10.68% and 11.39%, indicating substantial over-retrieval. In comparison, EviLink achieves 51.58% NSP and 63.60% NSF while retaining 98.05% NSR, providing high schema coverage with considerably less irrelevant context. EviLink also outperforms RSL-SQL and APEX-SQL across all four schema-linking metrics. Its average token consumption is 25.68K, compared with 51.81K for AutoLink and 50.95K for APEX-SQL. These results demonstrate a favorable balance among schema coverage, relevance, and token cost on BIRD-Dev.

On Spider2-Snow, where databases are larger and closer to enterprise-style settings, EviLink shows its clearest recall advantage. It achieves the best recall among all methods, with 93.04% SRR and 97.86% NSR, while using 116.55K tokens on average. Compared with APEX-SQL, EviLink improves SRR from 80.85% to 93.04% and NSR from 95.26% to 97.86%, while reducing token usage from 580.80K to 116.55K. However, its NSP and NSF are lower than those of APEX-SQL, at 26.04% and 36.65% versus 47.50% and 58.64%, respectively. These results show that EviLink's main advantage in this setting lies in schema completeness and linking efficiency. Appendix F further shows that EviLink achieves the highest field-level SRR on medium and large schemas, exceeding the strongest baseline in each group by 5.06 and 14.29 percentage points, respectively. On the 95-example subset with unchanged officially released SQL, EviLink also achieves the highest field-level SRR (91.58%) and NSR (98.17%) with the lowest linking token consumption (55.27K). These results further support EviLink's recall and token-efficiency advantages (Appendix B.4).

4.3 Ablation Study

Table 2 reports ablation results on BIRD-Dev and Spider2-Snow using Qwen3.8-27B for all variants. We evaluate on all 1,534 BIRD-Dev examples and the 517 gold-labeled Spider2-Snow instances. On Spider2-Snow, removing multi-hypothesis grounding reduces SRR from 74.66% to 63.25% and NSR from 91.54% to 86.78%, while NSP and NSF increase from 54.58% to 60.56% and from 63.32% to 66.77%, respectively. On BIRD-Dev, removing this component similarly reduces SRR from 92.57% to 79.92% and NSR from 97.82% to 93.61%, while increasing NSP and NSF. This sug-

Table 2: Field-level ablation results on BIRD-Dev and Spider2-Snow. Full denotes the EviLink model. Multi and UGAR denote multi-hypothesis grounding and uncertainty-guided agentic refinement, respectively. Static L0–L3 replace adaptive evidence acquisition with fixed evidence levels. SRR, NSR, NSP, and NSF are reported as percentages. Token denotes the average per-example token consumption during schema linking. Higher values are better for metrics except Token.

Method	**BIRD-Dev** ($N = 1534$)					**Spider2-Snow** ($N = 517$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
Full	92.57	97.82	56.13	67.77	28.16K	74.66	91.54	54.58	63.32	91.36K
w/o Multi	79.92	93.61	77.89	82.59	9.03K	63.25	86.78	60.56	66.77	44.15K
w/o UGAR	92.37	97.73	55.12	66.73	25.29K	74.47	91.26	54.03	62.47	79.43K
Static L0	89.18	96.91	66.17	76.10	17.56K	62.86	88.57	64.74	71.33	76.62K
Static L1	92.63	98.04	56.97	68.78	37.18K	75.24	91.33	53.19	62.16	179.63K
Static L2	91.79	97.80	61.12	72.05	52.39K	78.34	92.47	54.59	63.03	521.07K
Static L3	91.66	97.66	62.55	73.22	87.45K	74.08	92.01	55.99	63.62	1991.14K

gests that multi-hypothesis grounding mainly improves schema coverage by preserving alternative SQL paths, at the cost of lower precision and higher token consumption.

Uncertainty-guided refinement shows a complementary effect, providing modest but consistent improvements in schema coverage and relevance. Removing refinement decreases all four field-level metrics on both benchmarks. On Spider2-Snow, SRR drops from 74.66% to 74.47% and NSR from 91.54% to 91.26%, while on BIRD-Dev, NSP decreases from 56.13% to 55.12%. Relative to *w/o UGAR*, Full improves NSF by 1.04 percentage points on BIRD-Dev and 0.85 on Spider2-Snow, with additional token consumption of 2.87K and 11.93K, respectively. Appendix J further shows that refinement's field F1 gains are more pronounced on Spider2-Snow examples whose schemas change, with cases illustrating both selective pruning and recovery.

Adaptive evidence acquisition provides a practical recall–cost trade-off. Static L0–L3 each use one evidence level throughout. Static L0 uses limited evidence and obtains lower SRR on both benchmarks, although its higher NSP and NSF reflect a more selective schema context. On Spider2-Snow, Static L1 and Static L2 achieve higher SRR than Full, but require 179.63K and 521.07K tokens, respectively, compared with 91.36K for Full. Static L2 reaches 78.34% SRR, while its NSF of 63.03% remains slightly below Full's 63.32%. Static L3 consumes 1991.14K tokens, more than $21\times$ Full, while its NSF is only 0.30 percentage points higher and its SRR is lower (74.08% vs. 74.66%). On BIRD-Dev, Full achieves 92.57% SRR with 28.16K tokens, while Static L1 reaches 92.63% with 37.18K tokens. Static L1–L3 obtain higher NSF at greater token cost. These results support uncertainty-guided evidence allocation as a cost-efficient way to retain high schema coverage, although it does not maximize every retrieval metric.

4.4 Downstream SQL Generation

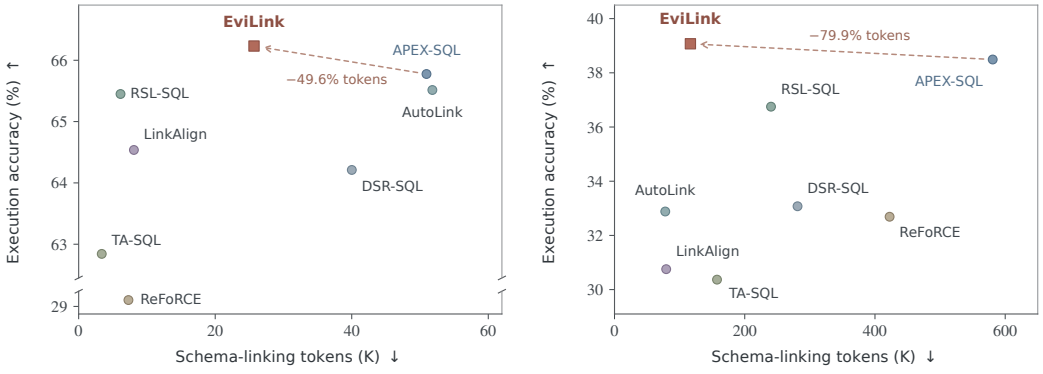
To examine whether schema-linking quality translates into better end-to-end SQL generation, we follow the downstream evaluation protocol of AutoLink (Wang et al., 2026) on BIRD-Dev ($N = 1534$) and Spider2-Snow ($N = 517$), using schema inputs from the corresponding main linking runs. We reuse its pipeline and prompts, sampling three SQL candidates, allowing up to three execution-guided revision rounds per candidate, and selecting the final SQL by execution-based voting. Within each benchmark, only the linked schema varies across methods; the question, database, prompt templates, and SQL generator remain fixed.

As shown in Table 3, EviLink achieves the highest execution accuracy (EX), measured by execution-result agreement with gold answers, on both benchmarks. With the SQL generator fixed within each benchmark, it obtains 66.23% EX on BIRD-Dev and 39.07% on Spider2-Snow, exceeding APEX-SQL by modest margins of 0.46 and 0.58 percentage points, respectively. Appendix H further illustrates how different evaluator-accepted SQL paths can induce different schema requirements.

Figure 3 shows these accuracy gains alongside linking-token reductions of 49.6% on BIRD-Dev and 79.9% on Spider2-Snow relative to APEX-SQL. Generation tokens are slightly lower on BIRD-Dev

Table 3: Downstream SQL generation performance on BIRD-Dev and Spider2-Snow. Correct denotes the number of correctly answered queries, and EX denotes execution accuracy. Linking and Generation denote the average per-example token consumption during schema linking and SQL generation, respectively. All methods use the same SQL generator, and the best results within each dataset are shown in bold.

Method	**BIRD-Dev** ($N = 1534$)				**Spider2-Snow** ($N = 517$)			
	Correct ↑	EX ↑	Token ↓		Correct ↑	EX ↑	Token ↓	
			Linking	Generation			Linking	Generation
TA-SQL	964	62.84	**3.38K**	**2.27K**	157	30.37	157.41K	**40.33K**
RSL-SQL	1004	65.45	6.13K	2.80K	190	36.75	240.34K	52.41K
ReFoRCE	450	29.34	7.29K	4.18K	169	32.69	422.47K	52.64K
LinkAlign	990	64.54	8.09K	9.07K	159	30.75	79.37K	72.40K
AutoLink	1005	65.51	51.81K	7.50K	170	32.88	**77.93K**	65.82K
DSR-SQL	985	64.21	40.01K	2.99K	171	33.08	281.08K	56.08K
APEX-SQL	1009	65.78	50.95K	2.90K	199	38.49	580.80K	45.53K
EviLink	**1016**	**66.23**	25.68K	2.83K	**202**	**39.07**	116.55K	50.17K



(a) BIRD-Dev (b) Spider2-Snow

Figure 3: Downstream SQL generation performance on BIRD-Dev and Spider2-Snow.

(2.83K vs. 2.90K) but higher on Spider2-Snow (50.17K vs. 45.53K). Overall, total tokens decrease by approximately 47.1% and 73.4%, respectively.

5 Conclusion

This paper reframes schema linking for large-scale Text-to-SQL along three dimensions. It moves from single-path matching to schema-need inference over multiple plausible SQL paths, from deterministic keep/drop selection to explicit modeling of required and uncertain schema items, and from static evidence provision to uncertainty-guided evidence acquisition. EviLink instantiates this reframing and shows that schema linking should provide not only a compact schema context, but a context that preserves the evidence needed for robust SQL reasoning. Experiments on BIRD-Dev and Spider2-Snow demonstrate that this perspective improves the balance among schema completeness, schema relevance, and token cost, with further gains in downstream SQL generation. These results suggest that robust schema linking can benefit from uncertainty-aware schema-need inference across multiple SQL paths, supported by evidence acquired where needed.

AI USE STATEMENT

During the preparation of this paper, we used large language models (LLMs) in a limited auxiliary capacity, mainly for polishing author-written text, identifying potentially relevant prior work, and assisting with coding. We also used LLMs to assist with reference SQL construction and schema annotation, with the construction and validation procedures detailed in Appendix B. All references and code suggestions involving LLM assistance were manually examined and validated by the authors. All substantive claims, methodological design, experimental implementation, data analysis, and result interpretation were conducted and verified by the authors. The authors take full responsibility for the final content of the manuscript.

ETHICS STATEMENT

This work studies schema linking for Text-to-SQL using public benchmark resources, including BIRD-Dev and Spider2-Snow. We do not collect new user data, use private user databases, or conduct human-subject studies. We follow the licenses and terms of use of the benchmark resources and baseline implementations. We additionally construct reference SQL for Spider2-Snow and schema annotations for both benchmarks. Appendix B documents their construction, automated validation, targeted human review, and limitations. EviLink acquires database evidence through read-only tools without modifying the underlying data. Nevertheless, deployment on sensitive databases may expose information through schema metadata, representative values, or query results, while incorrect schema linking or SQL generation may produce misleading analyses. Applications should therefore enforce least-privilege database access, protect sensitive information shared with model services, validate queries and their outputs, and maintain audit logs.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 3 describes EviLink's multi-hypothesis schema grounding, uncertainty estimation, and agentic refinement, including the evidence hierarchy and tool-based update rules. Appendix K details the agent's tool interfaces, including their inputs, outputs, and usage constraints, while Appendix L provides system prompt excerpts for multi-hypothesis grounding and uncertainty-guided agentic refinement. Section 4.1 and Appendix A describe the benchmarks, model choices, baselines, and scope of comparison. Appendix B documents reference SQL construction, schema-label extraction and validation, evaluation-subset selection, and the exclusion of reference SQL, answer files, and schema labels from inference inputs. Appendix D summarizes the key inference settings and algorithmic hyperparameters, while Appendix C provides the definitions and computation procedures for the schema-linking metrics. Section 4.4 describes the downstream SQL-generation and execution-evaluation protocol, with the generator and prompt templates held fixed across methods within each benchmark. We also provide runnable code and the gold table and field annotations used in our evaluation in the **supplementary material**.

REFERENCES

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. Benchmarking and Improving Text-to-SQL Generation under Ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7053–7074, 2023. doi: 10.18653/v1/2023.emnlp-main.436. URL https://aclanthology.org/2023.emnlp-main.436/.

Bowen Cao, Weibin Liao, Yushi Sun, Dong Fang, Haitao Li, and Wai Lam. APEX-SQL: Talking to the data via Agentic Exploration for Text-to-SQL, 2026. URL https://arxiv.org/abs/2602.16720.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 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)*, pp. 2541–2555, 2021. doi: 10.18653/v1/2021.acl-long.198. URL https://aclanthology.org/2021.acl-long.198/.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation, 2024. URL https://arxiv.org/abs/2411.00073.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Consensus Enforcement, and Column Exploration, 2025. URL https://arxiv.org/abs/2502.00675.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024. doi: 10.14778/3641204.3641221. URL https://doi.org/10.14778/3641204.3641221.

Zhifeng Hao, Qibin Song, Ruichu Cai, and Boyan Xu. Text-to-SQL as Dual-State Reasoning: Integrating Adaptive Context and Progressive Generation, 2025. URL https://arxiv.org/abs/2511.21402.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 337–353, 2025. URL https://aclanthology.org/2025.coling-main.24/.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *International Conference on Learning Representations*, pp. 28691–28735, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/46c10f6c8ea5aa6f267bcdabcb123f97-Paper-Conference.pdf.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. The Dawn of Natural Language to SQL: Are We Fully Ready? . *Proc. VLDB Endow.*, 17(11):3318–3331, 2024a.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(11):13067–13075, 2023a. doi: 10.1609/aaai.v37i11.26535. URL https://ojs.aaai.org/index.php/AAAI/article/view/26535.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data*, 2(3), 2024b. doi: 10.1145/3654930. URL https://doi.org/10.1145/3654930.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems*, pp. 42330–42357, 2023b. doi: 10.52202/075280-1835. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf.

Geling Liu, Yunzhi Tan, Ruichao Zhong, Yuanzhen Xie, Lingchen Zhao, Qian Wang, Bo Hu, and Zang Li. Solid-SQL: Enhanced Schema-linking based In-context Learning for Robust Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 9793–9803, 2025. URL https://aclanthology.org/2025.coling-main.654/.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models, 2024. URL https://arxiv.org/abs/2408.07702.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, pp. 36339–36348, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 5456–5471, 2024. doi: 10.18653/v1/2024.findings-acl.324. URL https://aclanthology.org/2024.findings-acl.324/.

Irina Saparina and Mirella Lapata. AMBROSIA: A Benchmark for Parsing Ambiguous Questions into Database Queries. In *Advances in Neural Information Processing Systems*, pp. 90600–90628, 2024. doi: 10.52202/079017-2876. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/a4c942a8405cc910f0a833d28d2573cc-Paper-Datasets_and_Benchmarks_Track.pdf.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual Harnessing for Efficient SQL Synthesis, 2024. URL https://arxiv.org/abs/2405.16755.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7567–7578, 2020. doi: 10.18653/v1/2020.acl-main.677. URL https://aclanthology.org/2020.acl-main.677/.

Yihan Wang, Peiyu Liu, and Xin Yang. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 977–991, 2025. doi: 10.18653/v1/2025.emnlp-main.51. URL https://aclanthology.org/2025.emnlp-main.51/.

Ziyang Wang, Yuanlei Zheng, Zhenbiao Cao, Xiaojin Zhang, Zhongyu Wei, Pei Fu, Zhenbo Luo, Wei Chen, and Xiang Bai. AutoLink: Autonomous Schema Exploration and Expansion for Scalable Schema Linking in Text-to-SQL at Scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(40):33809–33817, 2026. doi: 10.1609/aaai.v40i40.40672. URL https://ojs.aaai.org/index.php/AAAI/article/view/40672.

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. SQL-to-Schema Enhances Schema Linking in Text-to-SQL, 2024. URL https://arxiv.org/abs/2405.09593.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425/.

A Benchmarks and Baseline Methods

This appendix describes the two evaluation benchmarks and the published designs of the seven baselines. We emphasize how each method identifies or explores relevant schema elements, while also describing its original SQL-generation pipeline where necessary. Our experiments evaluate the resulting schema contexts under a shared evaluation protocol, as clarified at the end of this appendix.

A.1 BENCHMARKS

Following established Text-to-SQL evaluation practice, we select BIRD-Dev and Spider2-Snow as complementary benchmarks. BIRD-Dev evaluates schema linking under realistic database contents and external knowledge requirements, while Spider2-Snow emphasizes enterprise-scale schemas and complex SQL workflows. Together, they allow us to assess schema completeness, relevance, and token cost across different database scales and task complexities.

BIRD-Dev. BIRD (Li et al., 2023b) evaluates Text-to-SQL grounded in realistic database contents. The original benchmark contains 12751 question–SQL pairs over 95 databases spanning 37 domains, with a total database size of 33.4 GB. Its central challenges include noisy database values, external knowledge required to connect questions to stored data, and efficient query execution. We use its development split, which contains 1534 examples over 11 databases. The released resources include SQLite databases, database descriptions, question-specific evidence, and reference SQL. The benchmark evaluates execution accuracy and also introduces Valid Efficiency Score, which accounts for query efficiency conditional on correctness. In this work, we use BIRD-Dev to examine schema selection under realistic value and knowledge requirements. Its released reference SQL supports the construction of question-specific table and field labels using the procedure in Appendix B; these labels serve as evaluation targets rather than inference inputs.

Spider2-Snow. Spider2-Snow is the Snowflake-based Text-to-SQL setting of Spider 2.0 (Lei et al., 2025), with 547 tasks accompanied by database metadata and documentation. It is distinct from Spider2-Lite, which spans multiple database systems, and from the broader project-level workflow setting. The original paper reports 152 databases and an average of 812.1 columns per database for Spider2-Snow; these statistics characterize the original benchmark, rather than our retained evaluation subset. Large schemas, nested data structures, domain-specific documentation, and dialect-dependent SQL operations make it a useful setting for evaluating schema linking at enterprise scale. The benchmark uses execution-based evaluation with task-specific answer-comparison criteria. In the snapshot used in our study, reference SQL is released for 120 of the 547 tasks. Following the construction and validation procedure in Appendix B, we obtain reference SQL and schema labels for 517 tasks and evaluate all methods on this same subset. These results therefore characterize the retained subset and are not directly interchangeable with scores on the full official benchmark.

A.2 BASELINE METHODS

TA-SQL. TA-SQL (Qu et al., 2024) uses task alignment to reduce hallucinations in schema linking and logical synthesis. Its Task-Aligned Schema Linking module first generates a *dummy SQL query* from the question and database schema, then extracts the referenced schema entities. The dummy query supplies a concrete SQL-based context for selecting tables and columns; it need not already be a correct executable answer. The original framework subsequently applies Task-Aligned Logical Synthesis, representing the required computation through pandas-like symbolic operations whose functions resemble SQL keywords, before producing SQL. The original implementation uses zero-shot prompting for schema linking and few-shot prompting for logical synthesis. The linking component is relevant to our comparison because it grounds schema selection in an intermediate SQL draft, allowing us to examine the coverage and relevance of schema contexts obtained in this way.

RSL-SQL. RSL-SQL (Cao et al., 2024) combines forward and backward schema linking to reduce the risk of omitting necessary elements. Forward linking directly selects tables and columns from the question and full schema, with value samples and optional additional context. Backward linking recovers schema elements referenced by a preliminary SQL query. The two selections are united to construct a reduced schema. The paper's chosen backward-linking implementation uses exact column-name matching, favoring recall even when this includes additional same-named columns; it discusses SQLGlot-based extraction as an alternative with a different recall–noise trade-off. Beyond linking, the original system augments the reduced context with descriptions and predicted SQL components, selects between queries generated from full and reduced schemas using their execution results, and applies iterative correction. Thus, its schema-linking contribution is the complementary combination of direct selection and SQL-derived retrieval, while its native query-selection machinery belongs to the broader end-to-end system.

ReFoRCE. ReFoRCE (Deng et al., 2025) addresses enterprise Text-to-SQL through database information compression, SQL self-refinement, consensus enforcement, and column exploration. Compression groups tables with similar naming patterns, retaining detailed metadata for a representative table and names for the remaining tables. When the compressed context remains large, an LLM-based table-selection step further reduces the input. The system generates multiple SQL candidates, refines them with execution feedback, and selects high-confidence answers by agreement among execution results. Cases without a unique leading result are deferred to column exploration, where targeted SQL queries investigate relevant values, nested structures, and dialect-dependent operations before answer generation continues. Exploration queries can themselves be corrected using execution feedback. ReFoRCE is therefore relevant as a system that combines schema reduction with selective, data-grounded exploration.

LinkAlign. LinkAlign (Wang et al., 2025) separates database retrieval from fine-grained schema grounding in large, potentially multi-database environments. It first retrieves candidate schema information and uses LLM-guided query rewriting to recover potentially missing elements over multiple rounds. It then filters irrelevant candidates and, when multiple databases are present, localizes the target database. Finally, schema parsing identifies the tables and columns needed for SQL generation. The paper provides both a pipeline strategy using a single LLM call for each step and a multi-agent collaboration strategy. In the latter, schema auditing guides query rewriting, while agents with complementary roles assess candidate databases and verify schema selections. For single-database settings, the paper additionally describes random preservation with exponential decay and post-retrieval to mitigate excessive pruning. LinkAlign is relevant to our study as a retrieval-and-reasoning approach to schema grounding.

AutoLink. AutoLink (Wang et al., 2026) formulates schema linking as iterative exploration and expansion by an LLM agent. The agent interacts with a live database for SQL-based inspection and a vector index for semantic column retrieval. Its initial context contains the question, a complete table-name list, and an initial retrieved schema subset, rather than the full field-level schema. During interaction, the agent can explore schema metadata or data, retrieve additional columns, test schema sufficiency through SQL execution, explicitly add discovered elements, and terminate. The linked schema grows through set union with the added elements, emphasizing recovery of missing information. The original downstream pipeline samples SQL candidates, corrects them using execution feedback, and votes over their execution results. AutoLink provides a direct comparison with agent-driven schema acquisition, particularly the recall–relevance trade-off of progressively expanding a retrieved context.

DSR-SQL. DSR-SQL (Hao et al., 2025) organizes Text-to-SQL around a context state and a generation state. Context construction first consolidates structurally similar tables, removes uninformative columns, and summarizes external knowledge. It then selects relevant schema information adaptively: global exploration considers potential SQL structures and tables appearing across high-confidence candidates when the schema fits the context window, whereas partitioned exploration assesses tables individually for larger schemas. A subsequent alignment stage executes lightweight probing queries and summarizes their results into semantic cues such as enumerations and units. The original generation stage progressively extends or revises partial SQL, explores the database when feedback is insufficient, and finalizes the query when its subgoals are resolved. Multiple SQL candidates thus contribute to table selection, after which schema-aware alignment and generation operate on a consolidated schema context. EviLink additionally uses explicit path-wise table and field selections to estimate cross-path support and derive schema-item uncertainty for evidence acquisition.

APEX-SQL. APEX-SQL (Cao et al., 2026) grounds schema linking in logical planning and database exploration. It first generates multiple schema-agnostic logical plans and synthesizes them into a unified master plan P^* before schema pruning and grounding. This consolidated plan guides subsequent pruning and semantic-role assignment. Dual-pathway pruning processes schema batches with complementary deletion and preservation passes: a column survives if it is not deleted or is explicitly selected for preservation. Next, semantic linking assigns tentative roles to schema elements, and table-specific agents issue profiling queries to verify these roles against actual data. Global synthesis combines the retained schema and observations into the final schema subgraph. The original

Table 4: Spider2-Snow annotation statistics. SQL sources are counted before semantic repair. Audit categories overlap. Table and field totals count label instances across examples.

Statistic	Count
Accepted / excluded examples, out of 547	517 / 30
Released / constructed SQL selected at the merge	110 / 407
Recorded construction attempts	1079
Examples flagged in the initial SQL audit	49
Examples with conflicting initial judgments	39
Examples with documented human SQL review	44
Examples entering targeted regeneration	59
Examples with changed table / field labels after refinement	4 / 30
Final table / field instances	3770 / 14324
Examples approved by all three final label reviews	517
Human adjudications in the final label review	0

SQL-generation stage additionally retrieves exploration guidance and iterates through profiling, consolidation, SQL synthesis, and confirmation. Its multiple initial plans therefore inform exploration through a shared planning context, and the profiling agents are organized by table. EviLink grounds schema needs separately for each path before using cross-path support to construct the required and uncertain sets that guide refinement.

A.3 Scope of Our Comparison

Our comparison targets schema-context quality under a controlled downstream setting. As described in the main experimental setup, all methods use the same backbone within each benchmark and are evaluated against the same table and field labels. For downstream evaluation, each method supplies its linked schema to the same SQL generator, with the question, prompt templates, and database held fixed. The resulting execution scores measure the utility of the supplied schema contexts under that generator; they do not reproduce the complete native SQL-generation and answer-selection pipelines of all seven systems. Likewise, differences from the original papers' reported scores may reflect the backbone, evaluation subset, schema labels, and generation protocol used here.

Schema-label agreement and downstream execution evaluate complementary properties. The labels in Appendix B describe the tables and fields used by a selected reference SQL query, while another correct query may use a different schema subset. We therefore interpret schema recall and precision relative to these reference labels, and assess answer correctness separately through execution. This distinction is particularly relevant to methods that retain additional schema elements to support alternative SQL realizations.

B Construction of Gold Tables and Fields

Spider2-Snow provides gold SQL for only 120 of the 547 examples in our snapshot, limiting the coverage of schema-linking evaluation. Neither Spider2-Snow nor BIRD-Dev directly provides question-specific gold tables and fields.

For BIRD-Dev, we use table and field labels derived from the official SQL for all 1,534 development examples. Tables are identified as `database.table`, and fields are associated with their parent tables. We verify SQL and database consistency across the official files and check example alignment, including questions and external evidence. We release the reference SQL and evaluation labels.

For Spider2-Snow, we additionally construct and validate reference SQL, yielding 517 annotated examples (94.5% coverage). Both benchmarks share the schema-label extraction and checking procedure described below; we report detailed Spider2-Snow records to illustrate this procedure. Table 4 summarizes the construction and audit records.

Table 5: Schema annotation rules. Field labels follow explicit top-level column references after scope and lineage resolution.

Construct	Rule
CTEs and aliases	Trace references to physical sources; exclude derived names.
Correlated columns	Resolve the source in the outer scope defining its alias.
Nested fields and `FLATTEN`	Label the top-level physical column; distinguish pseudo-columns from real columns with the same names.
`COUNT(*)`	Label the source table without adding every column.
Intermediate `SELECT *`	Trace columns explicitly referenced downstream.
Direct `SELECT *`	Do not expand the wildcard to all catalog columns.
Table shards	Label each referenced physical shard separately.

B.1 Constructing and Auditing Reference SQL

We use Codex to attempt SQL construction for all 547 examples, including those with released SQL. Given the question, schema metadata, task knowledge, and official answer CSV files, the agent develops Snowflake SQL using database inspection and execution feedback. We use `GPT-5.6 terra high` for initial construction and `GPT-5.6 sol high` for retries and semantic repair, without a fixed iteration cap. Instructions prohibit answer enumeration, hardcoding data-derived values or identifiers, and replacing required lookups with answer-derived or database-derived constants. Question-specified constants, unit conversions, output labels, and ordinary null defaults are permitted.

Candidates are executed on Snowflake and checked with the official evaluation functions, respecting designated columns, row order, a floating-point tolerance of 0.01, and alternative answer files. Before semantic repair, the selected references comprise 110 released and 407 constructed queries, with released SQL preferred when both pass. The 30 excluded examples comprise 18 timeout or availability-related failures and 12 execution errors or result mismatches.

We use Claude Code, Codex, and CodeBuddy to review SQL against the question and task knowledge, checking for hardcoding, omitted lookups, and unnecessary dependencies. Review covers joins, filters, existence tests, and row multiplicity, including tables that contribute no output columns. Human review covers 44 examples; these and 15 additional cases identified through other audit signals undergo targeted regeneration. Revisions must pass execution comparison before acceptance; if no accepted revision is produced, the existing reference is retained. Final SQL-audit records contain no outstanding flags for the 517 retained queries.

Of the 110 released queries selected before semantic repair, 15 are subsequently revised, leaving 95 unchanged official references in the final set. Appendix B.4 reports results on this subset.

B.2 Extracting and Checking Schema Labels

For both benchmarks, table labels identify fully qualified physical tables, and field labels identify physical-table/column pairs. We use SQLGlot with the corresponding SQL dialect to resolve aliases and column lineage across nested query structures. The Spider2-Snow records reported here use SQLGlot 29.0.1 in the Snowflake dialect. Extraction includes references in joins, filters, grouping, ordering, `HAVING`, and `QUALIFY`. Catalog metadata supports name resolution without excluding valid cross-database references. Table 5 specifies the annotation conventions.

Field labels follow an explicit-reference convention: direct `SELECT *` does not by itself add all output columns, although these remain execution dependencies. Field labels therefore do not represent complete execution lineage. All retained examples have nonempty field labels. A separate validator checks label structure, catalog membership, and missing catalog-known table references, while targeted checks correct extraction errors.

One dynamic-SQL example uses a documented static expansion for label extraction while retaining the original SQL as its execution reference. Catalog validation is limited for three SQL-referenced fields absent from the supplied metadata and one annotated table with an empty catalog entry.

Table 6: Schema-linking performance on the Spider2-Snow subset with unchanged officially released SQL ($N = 95$), using GLM-5.1. Token denotes the average per-example token consumption during schema linking; the same cost is reported at both evaluation levels. Higher values are better for all metrics except Token.

Method	**Field level** ($N = 95$)					**Table level** ($N = 95$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
TA-SQL	40.00	76.55	**75.36**	**73.15**	70.41K	77.89	89.09	**89.23**	**87.03**	70.41K
RSL-SQL	75.79	95.83	24.06	35.30	224.75K	**100.00**	**100.00**	28.66	39.47	224.75K
ReFoRCE	35.79	83.16	54.20	59.87	165.30K	93.68	98.07	59.67	67.86	165.30K
LinkAlign	68.42	83.65	8.73	15.12	56.94K	94.74	95.79	23.41	34.67	56.94K
AutoLink	85.26	93.24	8.52	14.87	72.78K	91.58	95.61	23.63	35.16	72.78K
DSR-SQL	89.47	93.70	27.14	37.40	169.33K	89.47	93.80	78.25	82.94	169.33K
APEX-SQL	82.11	96.14	47.88	59.12	362.43K	97.89	99.39	50.60	60.95	362.43K
EviLink	**91.58**	**98.17**	29.20	40.17	**55.27K**	98.95	99.91	56.57	66.42	**55.27K**

Claude Code, Codex, and CodeBuddy check the final labels for missing or extra tables and fields on both benchmarks. In the Spider2-Snow records reported here, all 517 examples receive approving judgments from all three agents. These checks complement extractor corrections and, for Spider2-Snow, the earlier human SQL review; agreement does not provide an independent estimate of annotation accuracy. Records retain final SQL, labels, review comments, and label changes.

B.3 Evaluation Set and Protocol

The 517 annotated Spider2-Snow examples form our schema-linking evaluation set. We correct two database identifiers to match the databases referenced by their SQL, leaving questions, example membership, and schema labels unchanged. All methods use the same examples, labels, and corrected metadata, with reference SQL, answer files, and schema labels excluded from inference inputs.

Schema metrics measure agreement with one selected reference SQL path per question, while downstream SQL is evaluated separately against official answers. Correct alternatives may use different tables or fields, and execution agreement on the available database does not establish equivalence across all database instances. Results apply to the retained 517 examples, with potential selection bias toward tasks whose reference SQL can be constructed in the available environment.

B.4 Evaluation on the Official-SQL Subset

We evaluate a subset with unchanged officially released SQL to examine whether the main findings also hold without constructed or revised reference queries. Of the 120 examples with released SQL in our snapshot, 113 belong to the final 517-example evaluation set. Among these, 18 use final reference SQL that differs from the official text, leaving 95 examples whose final references are byte-identical to the released SQL. The SQL used for schema-label extraction also matches the official text for all 95 examples. Subset membership is determined by reference provenance and exact SQL agreement, not method performance.

We reuse the frozen schema-linking and downstream SQL predictions from the main experiments, restricting evaluation and token aggregation to the same 95 examples for every method. Schema linking uses GLM-5.1, and downstream SQL generation uses Qwen3.6-27B with the same generation protocol across methods. Schema labels follow the extraction and validation procedures above, and downstream EX is evaluated against the official answers.

Table 6 shows that EviLink achieves the highest field-level SRR and NSR, at 91.58% and 98.17%, respectively, with the lowest linking token consumption of 55.27K per example. Compared with APEX-SQL, it uses 84.8% fewer linking tokens while obtaining higher field-level recall, although its NSP and NSF remain lower. At the table level, EviLink obtains 98.95% SRR and 99.91% NSR.

Table 7: Downstream SQL generation performance on the same official-SQL subset ($N = 95$). All methods use GLM-5.1 for schema linking and Qwen3.6-27B for SQL generation. Correct denotes the number of correctly answered queries, and EX denotes execution accuracy. Linking and Generation denote the average per-example token consumption in the corresponding stages.

Method	Correct ↑	EX (%) ↑	**Token ↓**	
			Linking	Generation
TA-SQL	28	29.47	70.41K	40.49K
RSL-SQL	35	36.84	224.75K	49.21K
ReFoRCE	36	37.89	165.30K	45.66K
LinkAlign	32	33.68	56.94K	71.40K
AutoLink	31	32.63	72.78K	69.70K
DSR-SQL	34	35.79	169.33K	55.55K
APEX-SQL	40	42.11	362.43K	**39.03K**
EviLink	**41**	**43.16**	**55.27K**	47.28K

RSL-SQL reaches 100% on both recall metrics, but EviLink retains higher table precision, with 56.57% NSP versus 28.66%.

Table 7 reports downstream performance on the same subset. EviLink achieves the highest EX at 43.16% (41/95), compared with 42.11% (40/95) for APEX-SQL. The downstream gain is one query, while generation token consumption is higher than APEX-SQL's (47.28K vs. 39.03K). These results support the main findings on field-level schema completeness and linking efficiency using unchanged official reference SQL.

C Metric Computation

This appendix provides the computation details for the schema-linking metrics introduced in Section 4.1. For each evaluable question i, let G_i denote the gold schema item set and $\hat{G}_i$ denote the predicted schema item set. The item unit can be either a table or a field, depending on the evaluation granularity.

For each question, we compute:

$$\mathrm{TP}_i = |G_i \cap \hat{G}_i|. \tag{8}$$

$$n_i^G = |G_i|, \quad n_i^P = |\hat{G}_i|. \tag{9}$$

$$R_i = \frac{\mathrm{TP}_i}{n_i^G}, \quad P_i = \frac{\mathrm{TP}_i}{n_i^P}. \tag{10}$$

$$F_{1,i} = \frac{2P_iR_i}{P_i + R_i}. \tag{11}$$

If a denominator is zero, the corresponding score is set to zero. Empty predictions are included in the evaluation set.

We define $s_i = 1$ if $R_i = 1$, and $s_i = 0$ otherwise. Given N evaluable questions, the reported metrics are:

$$\mathrm{SRR} = \frac{1}{N}\sum_{i=1}^{N} s_i, \quad \mathrm{NSR} = \frac{1}{N}\sum_{i=1}^{N} R_i. \tag{12}$$

$$\mathrm{NSP} = \frac{1}{N}\sum_{i=1}^{N} P_i, \quad \mathrm{NSF} = \frac{1}{N}\sum_{i=1}^{N} F_{1,i}. \tag{13}$$

The same computation is applied to table-level and field-level evaluation by changing the schema item unit. In the main experiments, we report field-level results by default.

Table 8: Experiment-specific inference configurations.

Dataset	Model	Temp.	Top-p	Token limit	Reasoning
Main schema linking					
BIRD-Dev	Qwen3.6-35B-A3B	1.0	0.95	16384	Off
Spider2-Snow	GLM-5.1	1.0	0.95	16384	Off
Schema-linking ablation					
Both	Qwen3.8-27B	1.0	0.95	16384	Off
Hypothesis-budget analysis					
Spider2-Snow	Qwen3.8-27B	1.0	0.95	16384	Off
Main SQL generation					
BIRD-Dev	Qwen3.8-27B	0.7	0.8	65536	Off
Spider2-Snow	Qwen3.6-27B	0.7	0.8	65536	Off
Backbone comparison					
Spider2-Snow	GLM-5.2	1.0	0.95	65536	Off
Spider2-Snow	DeepSeek-4 Pro	1.0	0.95	65536	Off
Spider2-Snow	GLM-5.3	1.0	0.95	65536	Low
Spider2-Snow	Qwen3.8-27B	1.0	0.95	65536	Off

D Parameter Settings

Model assignment and inference configurations. Table 8 summarizes the inference configurations for each experiment. For the main schema-linking comparisons, all methods use Qwen3.6-35B-A3B on BIRD-Dev and GLM-5.1 on Spider2-Snow, while retaining their method-specific procedures. The schema-linking ablations and hypothesis-budget analysis use Qwen3.8-27B. Across these experiments, EviLink uses temperature 1.0, top-$p = 0.95$, and a maximum generation length of 16384 tokens per call, with reasoning requested to be disabled. All token limits in this section refer to configured maximum generation lengths per call, not context-window sizes or measured token consumption.

Additional backbone evaluation. We additionally evaluate EviLink on Spider2-Snow with GLM-5.2, DeepSeek-4 Pro, GLM-5.3, and Qwen3.8-27B. All four configurations use temperature 1.0, top-$p = 0.95$, and a maximum generation length of 65536 tokens per call. We request low-effort reasoning for GLM-5.3 and disabled reasoning for the other three backbones. This evaluation therefore compares the listed backbone configurations, including their respective reasoning settings.

Downstream SQL generation. We use Qwen3.8-27B on BIRD-Dev and Qwen3.6-27B on Spider2-Snow. Within each dataset, all compared methods share the same SQL generator and candidate-generation, revision, and selection procedure, using their respective schema-linking outputs. Both protocols use temperature 0.7, top-$p = 0.8$, top-$k = 20$, minimum sampling probability 0, presence penalty 1.5, and repetition penalty 1.0, with reasoning requested to be disabled. They generate three SQL candidates and allow up to three revision rounds, using a base seed of 42. The maximum generation length is set to 65536 tokens per call on both datasets. The protocols account for the database dialect and evaluation procedure of each benchmark. Reference schema labels and gold execution results are not provided to schema linking or SQL candidate selection during inference.

Default EviLink method settings. Table 9 summarizes the default method parameters. Multi-hypothesis grounding generates at most $K = 4$ SQL hypotheses and may return fewer when the model considers them sufficient. Table selection starts with L0 evidence, field selection uses L1 evidence, and refinement uses L2 evidence by default, acquiring L3 evidence on demand. The required-set threshold is $\tau_{\text{req}} = 0.85$. An auxiliary threshold of 0.50 partitions uncertain items during refinement; it does not determine the final keep/discard decision. Refinement has separate budgets of up to eight turns for uncertain-item processing and up to three turns for candidate recovery. Both budgets are upper bounds, and the corresponding procedures may terminate earlier.

Table 9: Default EviLink method parameters and downstream SQL-generation budgets.

Parameter	Value
Multi-hypothesis grounding	
Maximum hypotheses, K	4
Hypothesis count	Adaptive (1–4)
Table evidence	L0
Field evidence	L1
Agentic refinement	
Base evidence	L2
On-demand evidence	L3
Uncertain-item turn budget	8
Candidate-recovery turn budget	3
Uncertainty thresholds	
Required threshold, τ_{req}	0.85
Uncertain-item partition threshold	0.50
Main SQL generation	
SQL candidates	3
Maximum revision rounds	3
Base seed	42

Ablation and sensitivity settings. Within each dataset, all schema-linking ablation variants use the same backbone, inference configuration, and evaluation set. The single-hypothesis variant sets $K = 1$, while the variant without uncertainty-guided agentic refinement (UGAR) directly uses the grounding output without refinement. Static-L0 through Static-L3 each use a fixed evidence level throughout table selection, field selection, and refinement. The hypothesis-budget analysis varies $K \in \{1, \ldots, 6\}$ on a fixed subset of 200 instances randomly sampled from our Spider2-Snow evaluation set ($N = 517$) using seed 42. All other method settings remain unchanged in this analysis. Dataset-specific prompts and configuration files accompany our implementation.

E TABLE-LEVEL RESULTS

Main results. Table 10 shows that EviLink achieves high table recall while retaining higher precision than strongly recall-oriented baselines. On BIRD-Dev, EviLink obtains 96.94% SRR and 98.79% NSR, together with 75.67% NSP and 83.12% NSF, using 25.68K tokens per example. Although RSL-SQL, LinkAlign, AutoLink, and APEX-SQL achieve higher recall, their NSP ranges from 33.06% to 45.95%, indicating a greater proportion of irrelevant tables in their predictions. On Spider2-Snow, EviLink achieves the highest SRR and NSR at 96.91% and 98.64%, respectively. It also exceeds RSL-SQL and APEX-SQL in NSP and NSF, while using 116.55K tokens compared with 240.34K and 580.80K, respectively. TA-SQL achieves higher NSP and NSF on both benchmarks, but lower recall, particularly on Spider2-Snow, where its SRR is 68.86%. These results highlight EviLink's ability to retain necessary tables while limiting irrelevant table inclusion relative to other high-recall methods.

Ablation study. Table 11 examines the contributions of multi-hypothesis grounding, agentic refinement, and adaptive evidence acquisition using Qwen3.8-27B on both benchmarks. Removing multi-hypothesis grounding reduces SRR from 96.48% to 92.50% on BIRD-Dev and from 88.59% to 82.59% on Spider2-Snow, while increasing precision and reducing token consumption. This trade-off supports its role in recovering necessary tables that a single hypothesis may miss. Refinement provides smaller recall gains: relative to w/o UGAR, Full increases SRR by 0.20 and 0.97 percentage points on the two benchmarks, respectively, with little change in NSP or NSF. Compared with Static L1–L3, adaptive evidence acquisition uses fewer tokens on both benchmarks. On Spider2-Snow, Full matches Static L1's SRR with 91.36K rather than 179.63K tokens, whereas Static L2 reaches a higher SRR of 90.91% at 521.07K tokens. Static L0 is cheaper and yields higher NSP and NSF, but lower SRR on both benchmarks. These results indicate that multi-hypothesis grounding

Table 10: Table-level schema linking performance on BIRD-Dev and Spider2-Snow. Token denotes the average per-example token consumption during schema linking. Higher values are better for all metrics except Token. Best results within each dataset are shown in bold.

Method	**BIRD-Dev** ($N = 1534$)					**Spider2-Snow** ($N = 517$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
TA-SQL	90.03	95.41	**92.13**	**92.66**	**3.38K**	68.86	81.50	**82.92**	**79.13**	157.41K
RSL-SQL	98.83	99.53	45.95	59.11	6.13K	96.32	98.52	31.97	41.59	240.34K
ReFoRCE	39.63	64.42	79.90	67.83	7.29K	89.75	95.86	55.86	63.34	422.47K
LinkAlign	**100.00**	**100.00**	33.06	47.10	8.09K	88.20	94.18	24.48	34.35	79.37K
AutoLink	99.93	99.97	33.93	48.01	51.81K	78.92	84.87	24.27	31.81	**77.93K**
DSR-SQL	94.72	97.71	81.88	86.73	40.01K	73.11	81.28	70.67	72.35	281.08K
APEX-SQL	99.02	99.63	41.33	54.65	50.95K	94.00	97.93	55.45	64.76	580.80K
EviLink	96.94	98.79	75.67	83.12	25.68K	**96.91**	**98.64**	58.85	67.05	116.55K

Table 11: Table-level ablation results on BIRD-Dev and Spider2-Snow. Full denotes the complete EviLink model. Multi and UGAR denote multi-hypothesis grounding and uncertainty-guided agentic refinement, respectively. Static L0–L3 replace adaptive evidence acquisition with fixed evidence levels. SRR, NSR, NSP, and NSF are reported as percentages. Token denotes the average per-example token consumption during schema linking. Higher values are better for all metrics except Token.

Method	**BIRD-Dev** ($N = 1534$)					**Spider2-Snow** ($N = 517$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
Full	96.48	98.55	77.78	84.69	28.16K	88.59	94.97	78.88	82.56	91.36K
w/o Multi	92.50	96.71	90.86	92.53	9.03K	82.59	91.90	81.86	83.42	44.15K
w/o UGAR	96.28	98.47	77.85	84.71	25.29K	87.62	94.61	79.08	82.46	79.43K
Static L0	96.09	98.38	80.96	86.76	17.56K	85.11	94.28	82.82	85.76	76.62K
Static L1	96.81	98.77	78.95	85.48	37.18K	88.59	94.62	77.11	80.83	179.63K
Static L2	96.35	98.56	81.26	86.94	52.39K	90.91	95.81	77.51	81.29	521.07K
Static L3	96.81	98.74	82.39	87.83	87.45K	88.97	95.18	76.31	79.64	1991.14K

improves table coverage, while adaptive evidence acquisition offers a practical trade-off between coverage and token consumption.

F Performance across Schema Sizes

We analyze the frozen GLM-5.1 main-experiment predictions for all 517 Spider2-Snow examples spanning 135 databases. We group examples using fixed thresholds on the number of top-level fields in the full database schema: small (≤ 91), medium (92–505), and large (≥ 506), containing 171, 178, and 168 examples, respectively. Table 12 compares field-level SRR across all eight methods.

On small schemas, EviLink achieves 98.25% SRR while retaining a more precise schema context than LinkAlign, with field-level NSP of 32.16% versus 14.14%. This demonstrates strong schema completeness together with more selective field retention.

EviLink achieves the highest field-level SRR on both medium and large schemas. Its advantage over the strongest baseline in each group increases from 5.06 percentage points over DSR-SQL on medium schemas to 14.29 percentage points over RSL-SQL on large schemas. In the large-schema group, EviLink retains all reference fields on 148 of 168 examples, compared with 124 for RSL-SQL. It also exceeds APEX-SQL across all three groups, with the largest margin of 18.45 percentage points on large schemas. These results show that EviLink's recall advantage is particularly pronounced in the large-schema setting targeted by our method.

Table 12: Field-level SRR across schema sizes on Spider2-Snow ($N = 517$), using GLM-5.1. Groups are defined by the number of top-level fields in the full database schema. SRR is reported as a percentage, and the best result in each group is shown in bold. Gains discussed in the text are computed from unrounded scores.

Method	**Small** ($N = 171$) ≤ 91 fields	**Medium** ($N = 178$) 92–505 fields	**Large** ($N = 168$) ≥ 506 fields
TA-SQL	49.71	43.82	35.12
RSL-SQL	83.63	80.90	73.81
ReFoRCE	47.95	39.89	33.33
LinkAlign	**100.00**	74.16	14.88
AutoLink	99.42	87.08	31.55
DSR-SQL	91.81	87.64	38.10
APEX-SQL	87.72	84.83	69.64
EviLink	98.25	**92.70**	**88.10**

Table 13: Schema linking performance and token consumption of EviLink with four LLM backbones on Spider2-Snow ($N = 517$). SRR, NSR, NSP, and NSF are reported as percentages. Token denotes the average schema linking token consumption per example, in thousands; the same cost is reported at both evaluation levels. Bold indicates the highest schema linking scores and the lowest token consumption within each evaluation level.

Backbone	**Spider2-Snow** ($N = 517$)				
	SRR	NSR	NSP	NSF	Token
Field level					
GLM-5.2	94.97	98.51	22.29	32.53	163.73K
DeepSeek-4 Pro	92.07	97.52	26.42	36.89	170.82K
Qwen3.8-27B	73.31	91.45	**53.60**	**62.72**	**87.18K**
GLM-5.3 Low	**96.52**	**98.70**	25.46	37.18	130.16K
Table level					
GLM-5.2	97.49	**98.94**	51.84	60.98	163.73K
DeepSeek-4 Pro	95.36	98.11	61.68	69.37	170.82K
Qwen3.8-27B	86.85	94.29	**77.51**	**81.45**	**87.18K**
GLM-5.3 Low	**97.68**	98.86	54.77	64.58	130.16K

EviLink uses an average of 21.56K, 39.21K, and 295.19K linking tokens in the small, medium, and large groups, respectively. On large schemas, its token consumption is 80.85% lower than APEX-SQL's, combining stronger schema completeness with substantially lower linking cost.

G Additional Backbone Evaluation

We further evaluate EviLink with four additional LLM backbones (GLM-5.2, GLM-5.3 Low, DeepSeek-4 Pro, and Qwen3.8) on the same 517 Spider2-Snow examples to examine its applicability across models. As shown in Table 13, EviLink maintains high schema recall across all four configurations, with field-level NSR ranging from 91.45% to 98.70% and table-level NSR from 94.29% to 98.94%. Among these configurations, GLM-5.3 Low achieves the highest field- and table-level SRR of 96.52% and 97.68%, respectively, indicating strong complete-schema coverage. Qwen3.8 achieves the highest NSF at both the field level (62.72%) and the table level (81.45%), together with the lowest token consumption of 87.18K per example. These results support the applicability of EviLink across multiple model families, with high schema recall maintained across the evaluated backbones.

Table 14: Representative schema-dependency differences in downstream generation and gold SQL construction. In the upper examples, EviLink's linked schemas have complete table recall but incomplete field recall, while the generated SQL is accepted by the evaluator. The lower examples compare released and constructed SQL implementations; implementation differences alone do not establish semantic equivalence.

Source	Case	Alternative SQL paths	Schema implication
Downstream SQL generation: evaluator-accepted SQL despite imperfect field recall			
SQLGen	`sf_bq309`	The reference projects `title`; generated SQL omits it and returns question identifiers, body lengths, and user statistics.	A reference projection can contain fields outside the evaluated output requirements.
SQLGen	`sf_bq023`	The reference uses `entity_tp` to filter donations; generated SQL omits this filter and computes tract-level averages matching an accepted reference output.	Execution-based acceptance can coexist with omission of a reference filtering field.
SQLGen	`sf_bq346`	The reference traverses image-reference sequences; generated SQL directly joins segmentation records and reproduces the accepted category counts.	Accepted aggregate outputs may be obtained without traversing reference-specific auxiliary fields.
Gold SQL construction: implementation-path differences between released and constructed SQL			
Gold SQL	`sf_local003`	Released SQL computes spending from payment records; constructed SQL aggregates item prices plus freight charges before customer-level RFM analysis.	Different monetary-value sources introduce different tables, fields, and aggregation paths.
Gold SQL	`sf_bq233`	Released SQL reads R files directly from the contents table; constructed SQL uses a unified file–content join for both Python and R.	Language-specific versus unified retrieval paths use different sources for file-path fields.
Gold SQL	`sf_local075`	Released SQL attributes purchases using event order; constructed SQL identifies purchase-containing visits through an event-dictionary join.	Sequence-based versus visit-based attribution changes both join and field dependencies.

H Understanding Multiple SQL Paths and Gold Schema Variability

Why multiple SQL paths matter. A central premise of EviLink is that a natural-language question in large-scale Text-to-SQL does not always determine a unique SQL realization. In practice, valid solutions may differ in source tables, projection granularity, aggregation units, output organization, or the auxiliary fields used to instantiate a particular query strategy. Since gold tables and fields are derived from SQL dependencies, different SQL realizations can induce different reference schema annotations. We examine this phenomenon from two complementary sources: downstream SQL generation and gold SQL construction.

Evidence from downstream SQL generation. In the downstream SQL-generation experiment, we find cases where EviLink has perfect table recall but imperfect field recall, while the final generated SQL is accepted by the execution evaluator. In these cases, the missing gold-labeled fields are

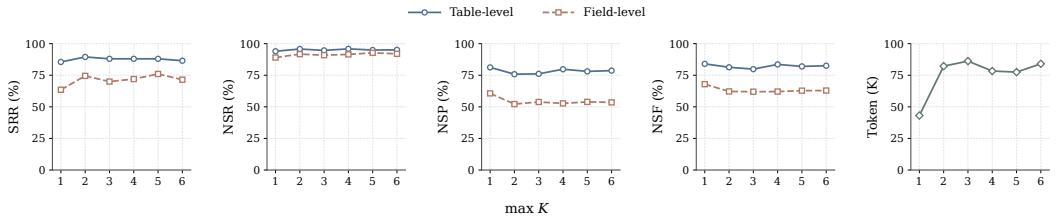


Figure 4: Sensitivity to the maximum number of SQL hypotheses K on 200 examples randomly sampled from our Spider2-Snow evaluation set ($N = 517$), using sampling seed 42. SRR, NSR, NSP, and NSF are reported at both table and field levels. Token denotes the average per-example token consumption during schema linking, measured in thousands. $K = 1$ uses a single hypothesis, and $K = 4$ is our default setting.

not referenced by the selected SQL. For example, Table 14 shows that a Stack Overflow query can return question identifiers and user statistics without projecting the reference title field. A census-tract donation query and a medical-image segmentation query also produce accepted outputs without using the reference entity-type field and image-reference sequence fields, respectively. These cases show that field-level recall is a useful diagnostic, but not a perfect proxy for execution-based acceptance: reference-specific fields need not appear in every accepted SQL realization. Acceptance on the evaluated database, however, does not by itself establish that every omitted field is semantically unnecessary.

Evidence from gold SQL construction. We observe a related phenomenon when comparing the released Spider2-Snow SQL with our final constructed reference SQL. Representative examples are shown in Table 14. An RFM query obtains monetary values from payment records versus item-level prices and freight charges; a source-code analysis query uses language-specific retrieval branches versus a unified file–content join; and a shopping-cart query uses event-sequence attribution versus purchase-visit membership with an event-dictionary join. These differences are not merely surface-level variations: they change the source tables, join structures, or fields consumed by the queries, and can therefore induce different reference schema sets. Some pairs also differ in filtering or aggregation semantics, so implementation differences alone should not be interpreted as proof of semantic equivalence. Nevertheless, they illustrate why schema annotations must be understood in relation to the particular SQL realization from which they are derived.

Takeaway. Together, these observations motivate our formulation of schema linking as schema-need inference under multiple plausible SQL hypotheses. A single reference SQL provides a useful evaluation target, but its dependency set should not automatically be equated with the unique set of question-essential schema items. This motivates EviLink to preserve schema evidence for multiple plausible reasoning paths, distinguish required from uncertain schema items, and acquire additional evidence only when uncertainty remains.

I Sensitivity to the Number of SQL Hypotheses

Multi-hypothesis grounding improves schema coverage but introduces a recall–precision trade-off. Figure 4 compares hypothesis budgets $K \in \{1, \ldots, 6\}$ using Qwen3.8-27B on the same 200 examples randomly sampled from our 517-example Spider2-Snow evaluation set using seed 42. On this sample, all multi-hypothesis settings achieve higher SRR and NSR than $K = 1$ at both table and field levels, supporting the benefit of considering alternative SQL realizations during grounding. However, these recall gains are accompanied by lower precision and F1, particularly at the field level. Most of the recall improvement is already obtained at $K = 2$; further increases yield limited changes in NSR and non-monotonic changes in SRR.

Token consumption also varies non-monotonically with K, so a larger hypothesis budget does not necessarily incur a higher total cost. At $K = 4$, table-level precision and F1 improve over $K = 2$ and $K = 3$, while token consumption is lower than at $K \in \{2, 3, 6\}$. On this sample, $K = 5$ achieves the highest field-level SRR and NSR and the lowest token cost among the multi-hypothesis

Table 15: Schema changes before and after uncertainty-guided agentic refinement (UGAR) on Spider2-Snow ($N = 517$). Arrows indicate before $\rightarrow$ after refinement. SRR and NSF are field-level percentages; all counts refer to examples. Complete recall means retaining all reference fields. Refinement Token denotes average refinement tokens per example, in thousands.

Statistic	**GLM-5.1** Main experiment	**Qwen3.8-27B** Ablation experiment
Field SRR	$94.00 \rightarrow 93.04$	$74.47 \rightarrow 74.66$
Field NSF	$35.16 \rightarrow 36.65$	$62.47 \rightarrow 63.32$
Unchanged schema	340	384
Examples with fields removed	175	95
Examples with fields added	4	39
Field F1 improved / decreased	175 / 2	104 / 27
Complete recall before refinement	486	385
Complete recall gained / lost	1 / 6	8 / 7
UGAR Token	16.27K	11.93K

settings, while $K = 4$ achieves higher table-level precision and F1 than $K = 5$. We retain the default $K = 4$, fixed before this analysis; these results illustrate the recall–precision–cost trade-off rather than establish a universally optimal hypothesis budget.

J Analysis of Uncertainty-Guided Agentic Refinement

We analyze how uncertainty-guided agentic refinement (UGAR) complements multi-hypothesis schema grounding through selective pruning and schema recovery on all 517 Spider2-Snow examples. Table 15 reports the GLM-5.1 main experiment and the Qwen3.8-27B ablation separately. In the ablation, Full and *w/o UGAR* share the same grounding output, isolating the effect of refinement on a fixed initial schema.

Concentrated gains from selective refinement. UGAR's benefits are concentrated in a subset of examples, explaining why aggregate improvements appear modest. The selected tables and fields remain unchanged for 65.76% of GLM-5.1 examples and 74.27% of Qwen3.8-27B examples. Among examples whose schemas change, however, mean field F1 increases by approximately 4.35 and 3.28 percentage points, respectively. Full-dataset averages spread these improvements over many examples with unchanged schemas. Grouping examples by their observed refinement outcomes therefore reveals the gains associated with UGAR's schema adjustments more clearly.

Recovery beyond strict recall. Strong initial grounding leaves limited room for further gains in strict recall: GLM-5.1 already retains all reference fields on 486 of 517 examples before refinement. UGAR also provides useful recovery that is more visible in cases with incomplete initial schemas. On Qwen3.8-27B, among the 132 examples initially missing reference fields, UGAR raises mean field recall from 65.79% to 68.20%, improves recall on 15 examples, and restores complete recall on eight. Since SRR records only whether all reference fields are retained, partial recovery contributes to schema coverage without necessarily changing SRR. These results highlight the value of examining both aggregate completeness and the schema dependencies recovered on individual examples.

Illustrative pruning and recovery cases. Individual cases show how UGAR improves the schema context for specific query requirements. For `sf_local163` in the GLM-5.1 run, UGAR removes five faculty attributes unrelated to the requested salary comparison, reducing the selection from 11 to six fields while retaining all four reference fields. This produces a more focused schema context without losing reference coverage. For `sf_bq273` in the Qwen3.8-27B run, UGAR adds fields for sales price, inventory cost, and the inventory identifier needed for the profit calculation, raising field recall from 75% to 100%. Together, these cases illustrate UGAR's complementary roles in removing distracting attributes and recovering missing schema dependencies, showing how targeted refinement can provide useful improvements beyond those visible in aggregate scores.

Tool Design: Uncertainty-Guided Agentic Refinement

[TOOL INTERFACE] Six Tools for Uncertainty-Guided Evidence Resolution

The agent uses six tools to acquire evidence, inspect missed schema items, and commit keep / discard decisions. Evidence tools expose information; decision tools change the final schema.

[1] get_field_stats(field_ids)
Purpose: request available L3-level statistics beyond inline L2 evidence for uncertain working-set fields.
Params: field_ids, a list of working-set field identifiers.
Returns: available L3 profiling increments, such as ranges, summary statistics, histograms, and distributional signals when available.
Constraint: use only when description, samples, and null_ratio are insufficient and L3 evidence may affect a keep/discard decision.

[2] probe_sql(sql)
Purpose: run a lightweight SQL probe to verify whether a join works or whether a value range overlaps the question.
Params: sql, a diagnostic query written by the agent.
Returns: up to 5 rows, or the raw database error if execution fails.
Constraint: runs with automatic LIMIT <= 5 and runtime timeout 30s; use for verification, not open-ended database exploration.

[3] list_tables()
Purpose: inspect currently unselected tables, mainly during the recall-recovery phase.
Params: none.
Returns: a compact list of unselected tables; isomorphic tables may be grouped to expose partitions or duplicated schemas.
Constraint: this tool only exposes candidate tables. The agent must call keep(...) to add any table to the final schema.

[4] list_columns(table_id)
Purpose: expand one candidate table when the agent suspects Stage 1 may have missed useful schema items.
Params: table_id, the identifier of one candidate table.
Returns: columns of the table with L2 evidence, including name, type, description, samples, and null_ratio when available.
Constraint: this tool only exposes columns. Any useful column must be explicitly passed to keep(...).

[5] keep(item_ids)
Purpose: retain schema items in the final linked schema.
Params: item_ids, a list of schema item identifiers.
Returns: the specified items are added to the final keep set.
Constraint: accepts working-set or candidate-pool items. Field-level keep may propagate across isomorphic siblings; table-level keep does not.

[6] discard(item_ids, reason)
Purpose: remove clearly unnecessary working-set items from the final linked schema.
Params: item_ids, a list of working-set item identifiers; reason, a one-line semantic justification.
Returns: the specified items are removed from the final schema.
Constraint: reason is required and must be grounded in the item's own semantics, not missing literal mention, redundancy, or weak guesswork.

Figure 5: Tool design for uncertainty-guided agentic refinement. The six tools support targeted evidence acquisition, recall recovery, and keep/discard decisions under a recall-first policy.

K TOOL DESIGN

Figure 5 summarizes the tool interface used in uncertainty-guided agentic refinement. The tools are designed to separate evidence acquisition from schema decisions. Specifically, `get_field_stats` and `probe_sql` allow the agent to request stronger field-level evidence or lightweight SQL-level verification when the inline profile is insufficient. `list_tables` and `list_columns` support recall recovery by exposing schema items that were not selected in the initial grounding stage. Finally, `keep` and `discard` commit the refinement decisions. We make discard deliberately stricter than keep: a discard action requires an explicit semantic reason, while unresolved uncertain items are kept by default to preserve recall. The refinement agent provides a concrete implementation of our uncertainty-aware schema-linking formulation. Our focus is on using cross-path schema support to guide refinement and selective evidence acquisition. Future work can explore alternative agent architectures and tool configurations to further improve refinement quality and efficiency.

L SYSTEM PROMPTS

Figures 6–9 show the main system prompts used by EviLink: multi-hypothesis schema-need elicitation, table selection, field selection, and uncertainty-guided agentic refinement. For readability, we present faithful excerpts that preserve the operational instructions, constraints, and output requirements, while abbreviating long illustrative examples and verbose JSON templates.

System Prompt: Multi-Hypothesis Schema-Need Elicitation

You are a query analyst. Enumerate every structurally DISTINCT way the question could be answered over an unknown schema. You are NOT picking one "best" solution; you are listing candidates so that downstream schema selection does not get locked onto a single path.

DO NOT reference any concrete table or column name. Describe concepts semantically.

[INPUT]
You will receive a natural-language question, optional external knowledge, and optional primary-key / foreign-key hints. Use them to reason about structure, but output only concept-level language.

[CORE DEFINITION] Schema Mapping
A schema mapping is a choice of: (a) which tables hold each entity, (b) which join graph connects them, and (c) which columns encode each filter / measure / grouping / ordering. Two mappings are DISTINCT iff they differ in (a), (b), or (c) beyond SQL syntax.

[COVERAGE PRINCIPLE] Avoid Single-Path Lock-in
Generate complementary hypotheses within the configured budget.

[HARD CONSTRAINTS] Bind Literals First
Before enumerating hypotheses, bind every literal constraint stated by the question: version identifiers, calendar periods, geographic scope, dataset/source identifiers, named entities, and numeric thresholds. Every hypothesis MUST be consistent with all such constraints.

[DIVERGENCE SIGNALS] Each Genuine Signal Can Create a New Mapping
S1. MULTI-SOURCE ENTITY — the same entity could live in sources with different row semantics.
S2. MULTI-PATH JOIN — entities could be connected directly, through a bridge, or through alternative relationship paths; bridge concepts MUST be listed.
S3. MULTI-ENCODING FILTER / MEASURE — a condition or measure could be encoded by different column configurations.
S4. MULTI-GRANULARITY MEASURE — the measure is available at raw and pre-aggregated levels.
S5. TERM AMBIGUITY — a term plausibly refers to different entities or attributes.

[FORBIDDEN]
✗ Do NOT split hypotheses that share the same table set, join graph, AND column configuration.
✗ Do NOT merge signal-distinct mappings for brevity.
✗ Do NOT use differentiation unless it is grounded in table set, join graph, or column configuration.
✗ Do NOT use concrete table or column identifiers.

[PER-HYPOTHESIS FIELDS]
Describe, in concept-level language: target, entities, filters, joins, aggregation, grouping, ordering, post_aggregation_filters, window_analysis, temporal_constraints, rationale, other_considerations, and differentiation. For joins, list every entity-to-entity link, including bridge / junction concepts and join-key concepts on BOTH sides.

[COVERAGE PITFALLS]
P1. Bridge path omitted. P2. Multi-source entity collapsed. P3. Multi-encoding filter collapsed.
P4. Implicit connector / long-format table missed. [Detailed wrong-vs-correct examples omitted for space.]

[SELF-CHECK]
Verify that rejected divergence signals are structurally justified; bridge / junction concepts are explicitly listed; join keys are described on both sides; each hypothesis pair differs in table set, join graph, or column configuration; and num_hypotheses matches the length of hypotheses.

[OUTPUT FORMAT]
Return ONLY a pure JSON object with: complexity_assessment, num_hypotheses, and hypotheses[...]. Each hypothesis contains hypothesis_id, target, entities, filters, joins, aggregation, grouping, ordering, post_aggregation_filters, window_analysis, temporal_constraints, rationale, other_considerations, and differentiation.

[OUTPUT RULES]
All required keys MUST be present. entities / filters / joins are arrays of plain strings. complexity_assessment MUST name the signals that drove the count or explain uniqueness. differentiation MUST identify a table-set, join-graph, or column-configuration difference.

Figure 6: System prompt excerpt: multi-hypothesis schema-need elicitation.

System Prompt: Table Selection

```
You are a schema linker. Select every table that COULD be needed to answer the question.

[INPUT]
 You will receive a natural-language question, an optional structured hypothesis describing the question's intent, and a candidate table list rendered
as a name-level L0 view, with table names and field names.

[HARD RULE]  Recall Is the Only Objective
 RECALL IS THE ONLY OBJECTIVE. Include a table whenever you cannot rule it out with certainty. Precision is handled by a downstream stage — do NOT
 optimize for it here.

 If you catch yourself thinking “this table is probably not needed,” that is NOT a valid exclusion reason. Include it.

[DECISION PROCEDURE]
 For each candidate table, ask:

   “Is there ANY plausible SQL that answers the question and uses this table?”

 If yes OR unsure, select it. Only exclude when you can state a concrete reason that no such SQL exists.

[SELECTIONS YOU MUST NOT MISS]
 1. Any table whose name or field names align with an entity, concept, attribute, filter value, or synonym in the question.

 2. Bridge / junction / mapping / relationship tables that could connect two entities mentioned in the question — even when a direct foreign key
   also exists.

 3. When several tables share the same structure and represent partitions of one logical dataset by date, shard suffix, region, version, or duplicated
   schema, and the question's scope may touch more than one partition, select ALL such partitions.

 4. When the same entity is recorded in multiple tables at different granularities or from different sources, such as detail vs. summary, primary vs.
   history / audit, or two independent providers of the same measure, select ALL such candidates.

 5. Cross-database and cross-schema tables whose names or semantics match a concept in the question. Case and schema boundaries do not exempt a table.

 6. When the question carries any temporal scope, such as a date range, a year, “recent,” or “latest,” select every time-series or time-keyed table
   whose coverage could overlap that scope.

 7. Tables whose name or visible field names signal a structural connector role: co-occurrence, mapping, link, transition, or event tables between two or more
   concepts already named by the question. Such tables often expose multiple identifier-shaped columns and may carry timestamp or status fields.

   Default to keeping connector tables when both endpoint concepts appear in the question, even when a direct foreign key exists. The connector may
   encode the only join path that preserves row semantics, such as history vs. current or many-to-many vs. one-to-many.

[RECALL PITFALLS]
 P1. Date / shard partitions.
    Do NOT select only one partition when the question scope may cover many same-schema partitions.

 P2. Bridge table hidden behind a direct foreign key.
    Do NOT assume the direct foreign key is sufficient when a relationship table could encode an alternative join path.

 P3. Same concept in two sources.
    Do NOT choose only one provider when multiple independent tables record the same measurement entity.

 [Detailed wrong-vs-correct examples omitted for space.]

[SELF-CHECK]
 Before emitting JSON, verify that:
   1. every excluded table has a concrete reason why no correct SQL could use it;
   2. any table without such a reason has been moved back to the selected set;
   3. every selected table name matches an input table exactly, case-sensitive.

[OUTPUT FORMAT]
 Return ONLY a single-line minified JSON object with:

   {"selected_tables":["<DB>.<SCHEMA>.<TABLE>", ...]}

 Use the EXACT full path string from the input. Do NOT prepend literal words such as db, schema, or database.

[OUTPUT RULES]
 1. Use exact full table names from the input, case-sensitive.
 2. selected_tables must be non-empty and contain no duplicates.
```

Figure 7: System prompt excerpt: table selection.

System Prompt: Field Selection

You are a schema linker. Select every field that COULD be needed to answer the question, within the already-filtered tables.

[INPUT]
You will receive a natural-language question, an optional structured hypothesis, and retained tables with L1 field profiles: field name, original type, and coarse semantic category. Use the profile as evidence — but absence of an obvious match is NOT a reason to drop a field.

[HARD RULE] Recall Is the Only Objective
RECALL IS THE ONLY OBJECTIVE. Include a field whenever you cannot rule it out with certainty. Precision is handled downstream — do NOT optimize for it here.

Omitting a field requires a concrete reason it cannot appear in any correct SQL. If you think "this field is probably not needed," include it.

[DECISION PROCEDURE]
For each field, ask:

"Is there ANY plausible SQL that answers the question and references this field?"

This includes SELECT, WHERE, JOIN, GROUP BY, ORDER BY, HAVING, window analysis, subquery, or CASE. If yes OR unsure, select it.

[SELECTIONS YOU MUST NOT MISS]
1. Any field whose name, type, category, or table context aligns with an entity, attribute, filter value, synonym, or display target in the question.

2. Every JOIN key must be selected on BOTH sides of the join. If you select an id / code / key on one table, select its counterpart on every other table it could join to.

3. When the same concept, such as entity id, timestamp, location, status, or name, can be encoded by columns in several retained tables, select it in EVERY such table. This also applies to same-schema partition / shard tables. Do NOT deduplicate across tables.

4. Select time, date, and ordering columns whenever the question mentions any temporal or ranking notion and the column could plausibly carry that semantics.

5. When a filter or measure admits multiple encodings, such as flag vs. NULL-ness, name vs. code, or dedicated column vs. descriptive pattern match, select ALL candidate columns.

6. When the question implies a hierarchy, such as country / state / county / zip, year / month / day, or category / subcategory, select identifiers at EVERY level that could be touched.

7. Within a retained table, default-keep retrieval-unit fields: primary identifiers, foreign-key-shaped columns, categorical / status / type / flag columns, human-readable name / title / label columns, and timestamp / date / time columns.

8. Hypothesis text is a HINT, not a whitelist. A field not mentioned in the hypothesis may still be required.

9. Surface-form alignment between the question and a column is a positive signal. If question tokens and column-name segments share a non-trivial substring, default-keep the column unless a concrete exclusion reason exists.

[EXCLUDE ONLY IF CERTAIN]
A field may be omitted ONLY if it is an engineering / ETL artifact with no business semantics AND the question cannot plausibly reference it, such as load_time, etl_timestamp, _batch_id, _file_hash, partition_key, shard_id, *_backup, *_bak, or *_deprecated.

Fields that could carry real data — status flags, soft-delete markers, authorship, versions, timestamps, geographies, or codes — are NOT in this category.

[RECALL PITFALLS]
P1. JOIN key on only one side.
P2. Cross-table same concept deduplicated.
P3. Latent filter column missed, e.g., "US" requiring country_code.
P4. Hypothesis silence treated as a veto.
P5. Same-schema partitions collapsed to one.
P6. Surface-form alignment ignored.

[Detailed wrong-vs-correct examples omitted for space.]

[SELF-CHECK]
Before emitting JSON, verify that:
1. every excluded field has a concrete reason why no correct SQL could use it;
2. every selected JOIN key has counterparts selected on all tables it could join to;
3. question tokens have been re-scanned against retained-table column names;
4. every selected field name matches an input field exactly, case-sensitive.

[OUTPUT FORMAT]
Return ONLY a single-line minified JSON object with:

{"selected_fields":["<DB>.<SCHEMA>.<TABLE>.<FIELD>", ...]}

Use the EXACT full path string from the input. Do NOT prepend literal words such as db, schema, or database.

[OUTPUT RULES]
1. Use exact full field names from the input, case-sensitive.
2. selected_fields must contain no duplicates.

Figure 8: System prompt excerpt: field selection.

System Prompt: Uncertainty-Guided Agentic Refinement

You are a schema verifier. Two hard rules govern every action you take:

[RULE 1] Recall First
A correct SQL may exist along multiple plausible reasoning paths. Any schema item that could be needed by any path must be retained. Missing a needed item is fatal; keeping a slightly redundant one is a minor cost. When in doubt, KEEP.

[RULE 2] Precision Second
Once recall is safe, prune only what you can clearly prove unneeded. Quality of discards matters more than quantity.

[STAGE-1 OUTPUT]
Stage 1 produced M hypotheses, voted by cross-hypothesis support:

LOCKED
Strong cross-hypothesis evidence. Final and cannot be modified.

UNCERTAIN
Selected by some hypotheses but not strong enough to lock. THIS IS YOUR JOB. Each item is shown with L2 evidence: name, type, description, samples, and null_ratio. Pull L3 statistics only when needed.

CANDIDATE
Zero hypotheses selected, but the database has it. After resolving Uncertain items, scan candidates and keep anything the pipeline missed.

[HOW TO ACT ON UNCERTAIN ITEMS]
1. Read the L2 view and decide whether it is enough for keep or discard.
2. If more evidence is needed, use L3 statistics or SQL probes.
3. Or leave the item undecided. Undecided items default to KEEP at convergence.

[DISCARD REQUIRES A JUSTIFICATION]
Every discard call must include a one-line reason explaining why no plausible SQL path could need this field.

Valid reasons cite the field's own semantics: ETL bookkeeping, internal system metadata, or L3 evidence showing irrelevant / constant values.

Invalid reasons include:
- "the question does not literally mention this column";
- "another column already covers the same concept";
- "this column is probably not needed."

[DISCARD PITFALLS]
D1. JOIN-key counterpart
Keep FK-shaped counterparts on joinable tables. Missing counterparts can make joins unresolvable.

D2. No literal wording match
Absence of a question-token match is evidence-NULL, not evidence-against. Filters may be encoded by flags, NULL-ness, codes, or names.

D3. Same hierarchy, different level
Do not collapse geography, time, taxonomy, or organization hierarchies to one level.

D4. Surface-form alignment
If a meaningful question token aligns with a column-name segment, this is positive evidence.

D5. Retrieval-unit role
Identifiers, FK-shaped columns, names, codes, labels, and timestamps on kept tables are frequent SELECT, JOIN, GROUP BY, or filter targets.

D6. Isomorphic-group propagation
If a discard propagates across sibling tables, the reason must hold for the whole group.

[EVIDENCE BEFORE DECISION]
If L2 leaves you unsure, use get_field_stats(...) or probe_sql(...). Do NOT discard on a thin reason.

[PIPELINE ROLE HINTS]
Pipeline role hints guide retention. Protected items cannot be discarded; other items are assessed using the question and available evidence. The absence of a role hint is not evidence for discarding an item.

[RECALL RECOVERY]
After Uncertain items are resolved, scan the CANDIDATE pool. Use list_tables and list_columns when needed. Keep anything that could matter to any plausible solution path, including join-key counterparts, latent encodings, hierarchy levels, surface-aligned fields, and retrieval-unit fields.

[TOOLS AND TERMINATION]
Tools: get_field_stats, probe_sql, list_tables, list_columns, keep, discard. After resolving uncertain items, perform targeted candidate recovery when needed. Stop when no further action is needed or the applicable turn budget is reached. Remaining uncertain items are kept by default.

Figure 9: System prompt excerpt: uncertainty-guided agentic refinement.