EviLink: Multi-Path Schema Linking with Uncertainty-Guided Evidence Acquisition for Large-Scale Text-to-SQL

Anonymous ACL submission

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 90.15% field-level strict recall rate, uses 123.30K 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 and 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 and 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 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

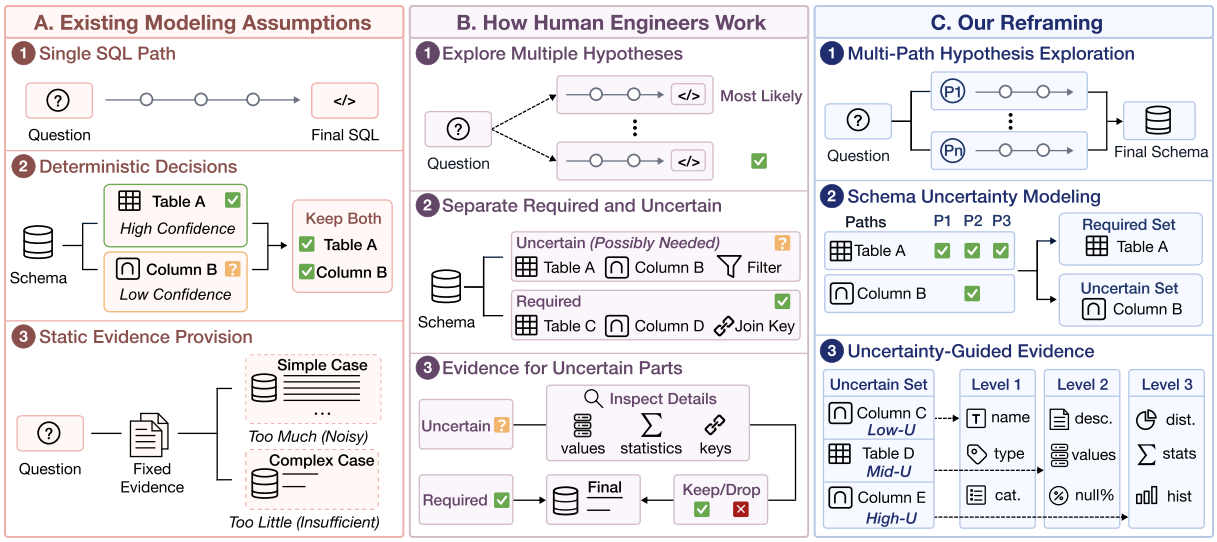

Figure 1: **From single-path schema linking to uncertainty-guided multi-path reasoning.** (A) Many existing formulations rely on a single SQL path, deterministic schema decisions, and static evidence provision. (B) Human engineers instead consider multiple plausible SQL paths, separate required from 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.

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. 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 propose **EviLink** as one implementation of 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 complex Text-to-SQL benchmarks and show that it achieves a better balance across schema completeness, schema relevance, token cost, and downstream SQL generation performance.

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 and 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 and 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. As a concurrent direction, ProSPy (Anonymous, 2026) studies enterprise Text-to-SQL as an end-to-end SQL–Python agentic framework, whereas EviLink focuses on schema linking. 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* (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.

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:

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

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

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

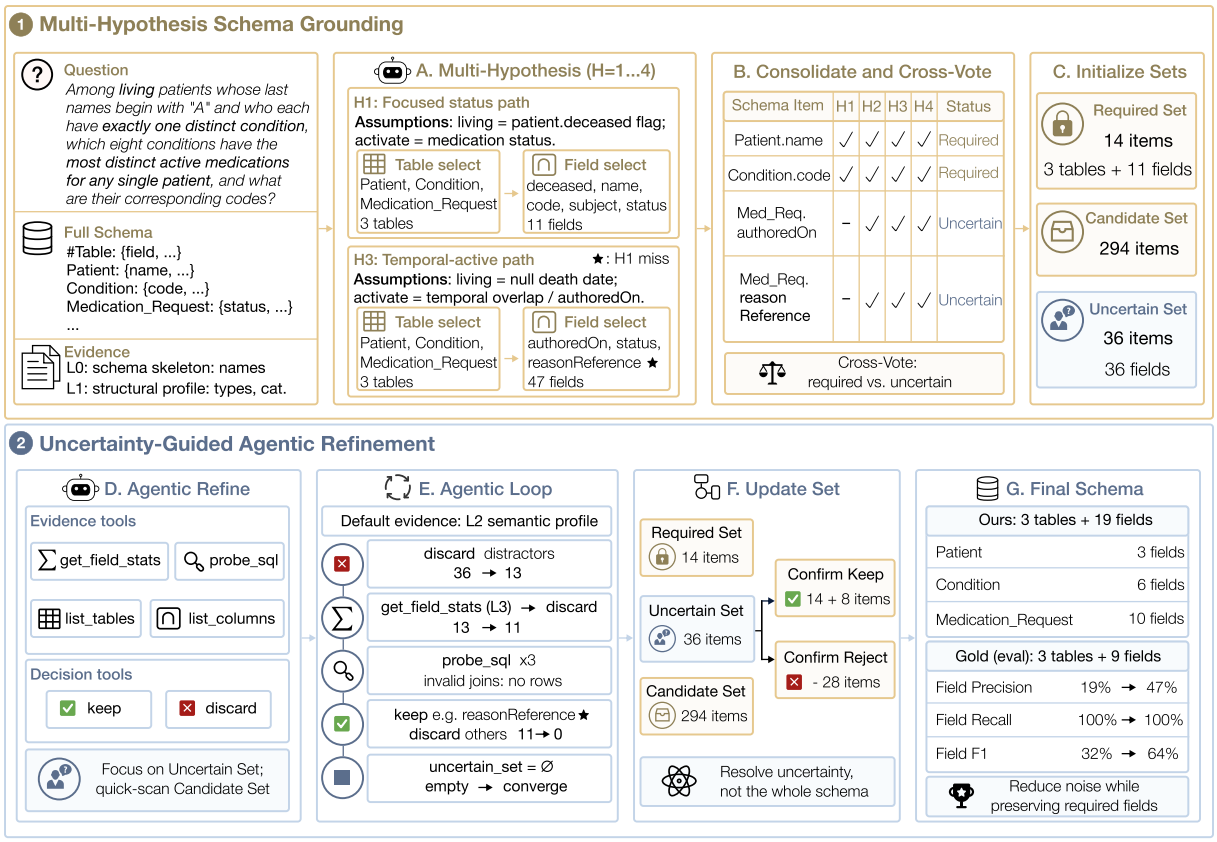

Figure 2: **Overview of EviLink.** The first stage performs multi-hypothesis schema grounding: (A) Multi-Hypothesis proposes multiple plausible SQL paths, (B) Consolidate and Cross-Vote aggregates their schema selections, and (C) Initialize Sets separates required, uncertain, and candidate items. The 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.

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

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

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:

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

and define its credibility score as

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

EviLink then initializes three sets. Items with $c_x \geq \tau_{\text{req}}$ are placed into the required set $\mathcal{S}_{\text{req}}$ and locked during refinement. Items with $0 < n_x$ but $c_x < \tau_{\text{req}}$ are placed into the uncertain set $\mathcal{S}_{\text{unc}}$. Items with zero support remain in the candidate set $\mathcal{S}_{\text{cand}}$, 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 $\tau_{\text{req}} = 0.85$; when four hypotheses are generated, an item supported by three hypotheses is still treated as uncertain rather than required.

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.

Uncertainty-aware bucketing. The refinement stage starts from the three sets initialized by cross-voting. Items in $\mathcal{S}_{\text{req}}$ are locked. They have strong cross-path support and cannot be removed by the agent. Items in $\mathcal{S}_{\text{unc}}$ form the decision set. Their necessity depends on which plausible SQL path is ultimately adopted, so the agent focuses on them for evidence acquisition and keep/drop decisions. Items in $\mathcal{S}_{\text{cand}}$ are not refined by default, but can still be recovered if the agent finds evidence that a missing table or field is needed. This bucketing narrows the agent's action space and avoids making binary decisions over the full database.

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. This policy avoids spending expensive evidence on already reliable items while still allowing the agent to inspect ambiguous fields in detail.

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.

The refinement state admits only three updates:

$$\mathcal{U} \xrightarrow{\texttt{keep}} \mathcal{K}, \qquad \mathcal{U} \xrightarrow{\texttt{discard}} \mathcal{R}, \qquad \mathcal{C} \xrightarrow{\texttt{keep}} \mathcal{K}. \quad (6)$$

Here, $\mathcal{K}$ and $\mathcal{R}$ denote confirmed-kept and confirmed-rejected items, respectively. This separation ensures that evidence acquisition remains read-only, while every schema update is tied to an explicit decision action.

Convergence and final schema. The refinement loop terminates when the uncertain set becomes empty, when the agent stops requesting tools, or when a fixed turn budget is reached. Let $\mathcal{S}_{\text{keep}}$ and $\mathcal{S}_{\text{drop}}$ denote the items confirmed as kept and discarded, respectively, and let $\mathcal{S}_{\text{pending}}$ 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

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

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 and has been adopted by many recent Text-to-SQL systems. Spider2-Snow is used as our primary benchmark because it targets enterprise-level database workflows and poses greater challenges in schema scale, schema ambiguity, realistic data access, and execution-based answer evaluation. This setting better reflects scenarios where schema linking becomes a

Method	BIRD-Dev ($N = 1534$)					Spider2-Snow ($N = 518$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
TA-SQL	73.73	89.84	**81.91**	**83.98**	**3.38K**	44.59	75.02	**69.53**	**68.78**	156.84K
RSL-SQL	89.90	97.10	44.32	57.89	6.13K	83.20	96.20	23.84	33.62	238.54K
ReFoRCE	20.01	54.61	71.87	58.64	7.29K	42.28	82.85	48.89	54.68	434.01K
LinkAlign	95.11	98.51	10.68	18.05	8.09K	64.67	81.29	9.48	15.52	**78.92K**
AutoLink	**98.89**	**99.76**	11.39	19.28	51.81K	73.36	83.01	7.62	12.61	79.84K
DSR-SQL	79.40	88.14	47.19	58.58	40.01K	74.71	81.42	21.49	30.07	283.37K
APEX-SQL	90.94	97.33	41.97	55.27	50.95K	81.85	95.07	46.02	57.20	574.53K
EviLink	82.20	94.06	69.79	77.01	22.00K	**90.15**	**97.01**	31.78	43.08	123.30K

Table 1: Field-level schema linking performance on BIRD-Dev and Spider2-Snow. Best results 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.

bottleneck. The model must identify a compact but sufficient schema context before SQL generation. We therefore report results on both benchmarks, while using Spider2-Snow as the main testbed for detailed analysis.

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 reproduce the baselines from their official implementations, making only minimal engineering adaptations required by our benchmark interface. All methods preserve their original linking logic and are evaluated under the same schema-labeling protocol.

Models. For a fair comparison, all baselines and EviLink are evaluated with the same backbone within each benchmark. We choose the benchmark-level backbone by balancing task performance and inference cost: Qwen3.6-35B-A3B for BIRD-Dev and GLM-5.1 for Spider2-Snow. GLM-5.1 is better suited to the more complex Spider2-Snow setting, while Qwen3.6-35B-A3B provides a more cost-efficient backbone for standard-scale evaluation. To examine whether EviLink is tied to a specific backbone, we further report a Qwen-family comparison on Spider2-Snow in Appendix E. For ablation studies, we keep the backbone fixed as Qwen3.6-35B-A3B so that performance differences can be attributed to the removed components rather than model changes. For downstream SQL generation, we use Qwen3.6-27B as a fixed SQL generator, with the schema context supplied by each linking method. Detailed parameter settings are provided in Appendix G, and results are from a single run under this fixed configuration.

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 gold SQL queries through execution verification, AST-based parsing, multi-agent cross-checking, and human verification; the full construction procedure is detailed in Appendix A. These metrics can be computed with either tables or fields as retrieval units, and the exact computation procedure is provided in Appendix B. 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 C 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. We additionally report **Avg. Tokens** to quantify the token cost of each method.

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, the results mainly reflect the recall–precision trade-off under a relatively standard benchmark where several baselines already reach near-saturated recall. LinkAlign achieves 95.11% SRR and 98.51% NSR, while AutoLink further reaches 98.89% SRR and 99.76% NSR. However, their substantially lower NSP and NSF scores suggest that these high recall scores come with considerable over-retrieval. EviLink instead achieves 82.20% SRR and 94.06% NSR while maintaining 69.79% NSP and 77.01% NSF. Compared with recall-oriented baselines such as RSL-SQL, LinkAlign, and AutoLink, EviLink provides a cleaner schema context while preserving competitive coverage. This result matches the intended role of EviLink. Rather than serving as a recall-only linker, EviLink is designed to balance schema coverage, relevance, and token cost, with its advantages becoming more pronounced as schema scale and ambiguity increase.

On Spider2-Snow, where databases are larger and closer to enterprise-style settings, EviLink shows its clearest advantage. It achieves the best recall among all methods, with 90.15% SRR and 97.01% NSR, while using only 123.30K tokens on average. Compared with APEX-SQL, EviLink improves SRR from 81.85% to 90.15% and NSR from 95.07% to 97.01%, while reducing token usage from 574.53K to 123.30K. This indicates that multi-hypothesis grounding improves coverage in complex databases, and uncertainty-guided evidence acquisition avoids the cost of uniformly expanding evidence for all schema elements.

4.3 Ablation Study

Table 2 reports ablation results on Spider2-Snow using Qwen3.6-35B-A3B as the backbone. We use an 80-case ablation subset sampled from the 518 gold-labeled Spider2-Snow instances before running ablations. The subset is obtained by ID-prefix stratified random sampling with a fixed seed of 42, so that different instance groups are proportionally represented. Removing multi-hypothesis grounding reduces SRR from 63.75% to 55.00% and NSR from 86.41% to 83.26%, while NSP and NSF remain close to Full. This suggests that multi-hypothesis grounding mainly improves schema coverage by preserving alternative SQL paths, rather than simply adding more schema candidates.

Method	SRR	NSR	NSP	NSF	Token
Full	63.75	86.41	59.06	66.47	112.84K
- w/o Multi	55.00	83.26	59.41	66.22	46.60K
- w/o Refine	72.50	89.00	50.54	58.94	93.02K
Static L0	56.25	84.43	59.04	65.87	84.38K
Static L1	66.25	88.82	54.51	62.67	280.65K
Static L2	50.00	83.71	59.53	64.96	738.57K
Static L3	67.50	88.65	55.28	61.95	2712.40K

Table 2: Ablation results on Spider2-Snow. Rows prefixed with "- w/o" remove the corresponding component from Full: Multi denotes multi-hypothesis grounding, and Refine denotes uncertainty-guided agentic refinement. Static L0–L3 replace adaptive evidence acquisition with fixed evidence levels.

Uncertainty-guided refinement shows a complementary effect. Removing refinement increases SRR from 63.75% to 72.50% and NSR from 86.41% to 89.00%, but sharply decreases NSP from 59.06% to 50.54% and NSF from 66.47% to 58.94%. This trade-off is expected because refinement prunes uncertain candidates after additional evidence inspection. We therefore do not interpret the higher SRR of - *w/o Refine* as better schema linking quality. An effective schema context should be both sufficiently complete and sufficiently clean, since excessive distractor fields can interfere with field grounding and join reasoning. The refinement stage sacrifices part of the recall-oriented candidate set, but substantially improves precision and yields the best overall NSF.

Adaptive evidence acquisition improves the accuracy-cost trade-off. Static L0 uses limited evidence and obtains lower SRR, indicating that lightweight profiling is often insufficient for resolving uncertain schema items. Static L1 and Static L3 achieve slightly higher SRR than Full, but both produce lower NSF and require much higher token cost; Static L3 consumes 2712.40K tokens, more than 24× Full, while its NSF is 4.52 percentage

Method	EX ↑	Token ↓
TA-SQL	33.75	36.09K
RSL-SQL	36.25	44.09K
ReFoRCE	36.25	50.44K
LinkAlign	32.50	67.86K
AutoLink	33.75	74.96K
DSR-SQL	31.25	54.57K
APEX-SQL	41.25	38.49K
Oracle Schema	42.50	**34.86K**
EviLink	**43.75**	38.19K

Table 3: Downstream SQL generation performance and average token cost on the 80-case Spider2-Snow subset.

points lower. Static L2 is less effective despite using 738.57K tokens. These results show that fixed evidence levels are either insufficient or inefficient, supporting uncertainty-guided evidence allocation.

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 the same 80-case Spider2-Snow subset used for ablation. We reuse its SQL-generation pipeline and prompts, where the generator samples multiple SQL candidates, revises them with execution feedback, and selects the final SQL by execution-based voting. For each method, we replace only the schema context with its linked schema, while keeping the question, prompt templates, database, and generator fixed. We use Qwen3.6-27B for all methods, with three SQL candidates and at most three revision rounds per candidate, keeping the comparison focused on the provided schema context.

As shown in Table 3, EviLink achieves the best execution accuracy (EX), where EX measures the percentage of generated SQL queries whose execution results match the gold answers. It outperforms the strongest baseline APEX-SQL by 2.50 percentage points (43.75% vs. 41.25%) with nearly identical SQL-generation token consumption (38.19K vs. 38.49K). Interestingly, EviLink also slightly outperforms the reference-SQL Oracle Schema in this setting (43.75% vs. 42.50%). This result suggests that a schema derived from one reference SQL is not necessarily the most useful context for an LLM generator. It is consistent with the observation of Katsogiannis-Meimarakis et al. (2026) that a minimal oracle schema may remove useful contextual signals. Our result provides a complementary explanation. A reference SQL captures one valid realization of the question, whereas an LLM generator may follow another plausible path that benefits from additional contextual schema items. EviLink can therefore help downstream generation by preserving schema items needed by alternative plausible SQL paths while still pruning unsupported uncertainty. We provide a qualitative analysis of such cases in Appendix D, showing that different accepted SQL paths can induce different schema requirements.

The SQL-generation token results further support this view. Methods with noisier schema contexts, such as LinkAlign and AutoLink, consume substantially more generation tokens while obtaining lower EX. In contrast, EviLink keeps the context compact and achieves the lowest generation-token usage among the top-performing methods. This pattern is consistent with the role of schema precision. Cleaner linked schemas reduce the reasoning burden on the generator, while multi-path preservation prevents the context from becoming overly narrow. Thus, effective schema linking should not simply approximate the smallest oracle schema, but provide a clean yet sufficiently expressive context for SQL generation.

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 requires moving from one-shot schema selection to uncertainty-aware schema-need inference across multiple SQL paths, supported by evidence acquired where needed.

Limitations

EviLink is mainly designed for large-scale and ambiguous Text-to-SQL settings, where questions may involve multiple plausible SQL paths and detailed schema evidence is costly to expose uniformly. For simpler cases with small databases and nearly deterministic SQL structures, its multi-hypothesis grounding, required/uncertain schema separation, and on-demand evidence use may be less necessary than in complex enterprise-style settings. In addition, our uncertainty estimation and evidence hierarchy are practical design choices rather than optimal solutions, and the quality of available evidence itself may vary across databases. When schema descriptions, representative values, or field statistics are incomplete or noisy, the refinement process may be less reliable.

Ethical Considerations

This work studies schema linking for Text-to-SQL using existing public benchmark resources, including BIRD-Dev and Spider2-Snow. We do not collect new user data, involve human participants, or use private user databases. The proposed method operates on benchmark-provided schemas and database resources for research evaluation. We follow the licenses and terms of use of all benchmark resources and official implementations used in this work. We do not introduce or release any additional user-level data, personally identifying information, or offensive content. The ethical considerations are therefore limited to general Text-to-SQL deployment settings, where database access control, query auditing, and application-level validation should be properly enforced.

References

Anonymous. 2026. ProSPy: A Profiling-Driven SQL-Python Agentic Framework for Enterprise Text-to-SQL. Under review.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and Improving Text-to-SQL Generation under Ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074.

Bowen Cao, Weibin Liao, Yushi Sun, Dong Fang, Haitao Li, and Wai Lam. 2026. APEX-SQL: Talking to the data via Agentic Exploration for Text-to-SQL. *Preprint*, arXiv:2602.16720.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2541–2555.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *Preprint*, arXiv:2411.00073.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Consensus Enforcement, and Column Exploration. *Preprint*, arXiv:2502.00675.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. *Preprint*, arXiv:2307.07306.

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

Zhifeng Hao, Qibin Song, Ruichu Cai, and Boyan Xu. 2025. Text-to-SQL as Dual-State Reasoning: Integrating Adaptive Context and Progressive Generation. *Preprint*, arXiv:2511.21402.

George Katsogiannis-Meimarakis, Katsiaryna Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. 2026. In-depth Analysis of LLM-based Schema Linking. In *International Conference on Extending Database Technology*, pages 117–130.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353.

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

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

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *Proceedings*

of the AAAI Conference on Artificial Intelligence, 37(11):13067–13075.

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

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. *Preprint*, arXiv:2305.03111.

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

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. *Preprint*, arXiv:2408.07702.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, pages 36339–36348.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. 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*, pages 5456–5471.

Irina Saparina and Mirella Lapata. 2024. AMBROSIA: A Benchmark for Parsing Ambiguous Questions into Database Queries. In *Advances in Neural Information Processing Systems*, pages 90600–90628.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. *Preprint*, arXiv:2405.16755.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Yihan Wang, Peiyu Liu, and Xin Yang. 2025. 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*, pages 977–991.

Ziyang Wang, Yuanlei Zheng, Zhenbiao Cao, Xiaojin Zhang, Zhongyu Wei, Pei Fu, Zhenbo Luo, Wei Chen, and Xiang Bai. 2026. 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.

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024. SQL-to-Schema Enhances Schema Linking in Text-to-SQL. *Preprint*, arXiv: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. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

A Construction of Gold Tables and Fields

Our schema-linking evaluation is conducted on two benchmarks: BIRD-Dev and Spider2-Snow. BIRD-Dev provides official gold SQL queries for all examples, which can be directly used to derive reference schema labels. Spider2-Snow, however, only publicly releases gold SQL queries for 120 examples. To extend schema-level supervision for Spider2-Snow beyond the released SQL set, we construct gold SQL queries for examples without released SQL. After execution-based and manual verification, we retain 398 additional queries.

Completing gold SQL for Spider2-Snow. Spider2-Snow contains 547 examples, among which 120 have publicly released gold SQL queries. For each remaining example, we use an agentic coding system, Claude Code, to synthesize candidate SQL queries based on the natural language question, the database schema, and the official gold execution result. The agent is instructed to infer an executable SQL query that reproduces the official result, rather than simply generating a plausible query from the question alone. We then execute each candidate SQL query against the corresponding database and retain it only if its execution result matches the official gold result. After execution-based filtering, we further conduct manual inspection to verify that the retained SQL queries are semantically reasonable and do not merely exploit accidental result-matching artifacts. This process yields 398 additional verified gold SQL queries. Since we could not obtain execution-matching and semantically reasonable SQL queries for the remaining 29 examples, our Spider2-Snow gold-schema evaluation uses 518 verified examples.

Deriving gold tables and fields. Given this verified gold SQL set, we extract gold tables and fields from each query using SQLGlot, an abstract-syntax-tree-based SQL parser. SQLGlot parses each SQL query into an AST and performs scope analysis and alias resolution, allowing us to map field references back to their corresponding physical tables. For unqualified field references in multi-table queries, we further use schema profiling metadata to resolve ambiguities, such as cases where the same field name may appear in multiple joined tables. This produces an initial set of gold tables and fields for each example.

Cross-checking and human verification. To ensure the reliability of the extracted schema labels, we conduct an additional cross-checking step with three independent agentic coding systems: Claude Code, Codex, and CodeBuddy. Each agent independently reads the gold SQL query and judges whether the extracted tables and fields are complete and correct. When the three agents produce inconsistent judgments, we manually inspect the corresponding case and revise the extracted labels when necessary. All constructed SQLs used for schema-label extraction are execution-verified against the official gold execution results and manually inspected for semantic reasonableness. These constructed SQLs are used only to derive reference schema labels for evaluation and are not provided to any schema-linking method during inference. This combination of AST-based extraction, execution-verified SQL construction, multi-agent cross-checking, and human verification provides a controlled basis for the gold tables and fields used in our evaluation.

B 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. Schema identifiers are canonicalized in a case-insensitive manner before comparison.

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:

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

Method	BIRD-Dev ($N = 1534$)					Spider2-Snow ($N = 518$)				
	SRR	NSR	NSP	NSF	Token	SRR	NSR	NSP	NSF	Token
TA-SQL	90.03	95.41	92.13	92.66	3.38K	73.94	83.90	82.18	79.96	156.84K
RSL-SQL	98.83	99.53	45.95	59.11	6.13K	97.88	99.16	30.52	40.24	238.54K
ReFoRCE	39.63	64.42	79.90	67.83	7.29K	90.15	96.12	54.04	62.07	434.01K
LinkAlign	100.00	100.00	33.06	47.10	8.09K	88.80	94.87	23.10	32.92	78.92K
AutoLink	99.93	99.97	33.93	48.01	51.81K	79.92	85.52	22.90	30.74	79.84K
DSR-SQL	94.72	97.71	81.88	86.73	40.01K	74.90	81.75	68.18	71.00	283.37K
APEX-SQL	99.02	99.63	41.33	54.65	50.95K	94.98	97.94	53.67	63.26	574.53K
EviLink	93.68	97.04	85.59	89.29	22.00K	95.75	98.04	64.12	71.90	123.30K

Table 4: Table-level schema linking performance on BIRD-Dev and Spider2-Snow.

$$\text{NSP} = \frac{1}{N}\sum_{i=1}^{N} P_i, \quad \text{NSF} = \frac{1}{N}\sum_{i=1}^{N} F_{1,i}. \quad (13)$$

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.

C Main Results on Tables

Table 4 reports table-level schema linking results. Since table-level evaluation is less strict than field-level evaluation, most methods obtain higher recall on both benchmarks. On BIRD-Dev, EviLink maintains competitive table recall while producing a cleaner schema context, achieving 85.59% NSP and 89.29% NSF with 22.00K tokens. On Spider2-Snow, EviLink reaches 95.75% SRR and 98.04% NSR, while retaining a stronger precision–recall balance than highly recall-oriented methods such as RSL-SQL and APEX-SQL. These results are consistent with the field-level findings, showing that EviLink improves schema linking quality not by indiscriminately expanding the schema context, but by preserving necessary tables while reducing irrelevant context.

D 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, filtering predicates, output organization, or the auxiliary fields used to instantiate a particular query strategy. Since gold tables and fields are derived from SQL dependencies, different accepted SQL paths can induce different gold schema annotations. We observe 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 still execution-correct. In these cases, the missing gold-labeled field is not referenced by the selected SQL and is not semantically required by the successful query plan. For example, Table 5 shows that a COVID-19 query can aggregate U.S. records by date without explicitly using an aggregation-level field, and an accident-level NHTSA query can aggregate vehicle-derived features without retaining a vehicle-level identifier. These cases show that field-level recall is a useful diagnostic, but not a perfect proxy for downstream SQL solvability: some gold-labeled fields are strategy-dependent rather than question-essential.

Evidence from gold SQL construction. We observe a related phenomenon during the construction of Spider2-Snow gold SQL. Among the instances where both the official SQL and our constructed SQL are executable and accepted by the evaluator, we find more than 20 clear cases with different but plausible SQL realizations. Importantly, these differences are not merely surface-level variations: because gold tables and fields are extracted from SQL dependencies, different accepted SQL paths can lead to different gold schema sets. Representative examples are shown in Table 5. This observation

Source	Case	Alternative SQL paths	Schema implication
Downstream SQL generation: correct SQL despite imperfect field recall			
SQLGen	`sf_bq018`	Gold schema includes `aggregation_level`; generated SQL filters U.S. records and aggregates by date.	Auxiliary gold fields may be unnecessary under another correct query plan.
SQLGen	`sf_bq043`	Gold schema includes `Mutation_Status`; generated SQL uses gene-symbol rows in the MC3 mutation table.	Different evidence tables can induce different required fields.
SQLGen	`sf_bq067`	Gold schema includes `vehicle_number`; generated SQL aggregates vehicle features by accident identifiers.	Row-level identifiers may be unnecessary for aggregate-level schema needs.
Gold SQL construction: different accepted SQLs induce different gold schemas			
Gold SQL	`sf_bq166`	Accepted SQLs organize subtype statistics in wide versus long formats.	Different output organizations lead to different projected fields and schema annotations.
Gold SQL	`sf_local209`	Accepted SQLs report the same delivery ratio as a fraction versus a percentage.	Different operationalizations can require different computation fields.
Gold SQL	`sf_local354`	Accepted SQLs return driver-year-constructor details versus distinct drivers.	Different projection granularities yield different gold field sets.

Table 5: Representative examples showing that multiple accepted SQL paths can induce different schema requirements.

suggests that the official SQL should be treated as one valid realization of the question, rather than the unique source of all necessary schema evidence.

Takeaway. Together, these observations support our formulation of schema linking as schema-need inference under multiple plausible SQL hypotheses. A single reference SQL provides a useful evaluation target, but it should not be mistaken for the only schema evidence required by the question. 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.

E Additional Backbone Evaluation

We further evaluate our method on Spider2-Snow with two Qwen3.6 backbones to assess its sensitivity to the underlying LLM. Since the main comparison against baselines is already conducted under the fixed evaluation setup in Section 4.2, this appendix focuses on how our method behaves when the backbone changes rather than providing another full method-to-method comparison. As shown in Table 6, Qwen3.6-27B achieves stronger recall-oriented performance than Qwen3.6-35B-

Model	SRR	NSR	NSP	NSF	Token
Field level					
35B-A3B	59.46	86.72	62.37	67.82	101.44K
27B	71.24	90.67	56.67	64.57	142.26K
Table level					
35B-A3B	83.59	93.38	79.15	82.44	101.44K
27B	88.80	94.82	78.67	82.63	142.26K

Table 6: Schema linking performance on Spider2-Snow ($N = 518$) with Qwen3.6 backbones.

A3B, improving field-level SRR from 59.46% to 71.24% and NSR from 86.72% to 90.67%. The improvement is milder at the table level, where SRR increases from 83.59% to 88.80% and NSR from 93.38% to 94.82%, suggesting that the lower-capacity backbone can still identify most required tables but is more likely to miss fine-grained fields that require precise field-level reasoning. In contrast, Qwen3.6-35B-A3B uses fewer tokens and obtains higher field-level NSP, indicating a more selective, precision-oriented but less complete schema selection behavior. Together with the main results in Tables 1 and 4, these results suggest that EviLink is not tied to a single model instance. The stronger Spider2-Snow results with GLM-5.1 further indicate that more capable backbones can better exploit multi-hypothesis grounding and evidence-guided refinement, especially for fine-grained field selection.

F Sensitivity to the Number of SQL Hypotheses

We study the effect of the maximum number of SQL hypotheses K in Figure 3. The single-hypothesis setting ($K = 1$) generally underperforms the multi-hypothesis settings on recall-oriented metrics, suggesting that committing to one assumed SQL path can miss schema items needed by alternative plausible realizations. Moving from $K = 1$ to moderate multi-hypothesis settings improves schema coverage, and the results remain relatively stable for $K \in \{2, 3, 4\}$. Larger values do not yield consistent additional gains and can substantially increase token cost, especially when more hypotheses introduce redundant or weakly useful schema evidence. We therefore set $K = 4$ by default, as it captures the benefit of multi-path grounding while maintaining a stable trade-off among recall, precision, F1, and inference cost.

G Parameter Settings

Table 7 summarizes the key implementation settings of our method. We keep the inference configuration fixed across all runs, including temperature, top-p, top-k, and the maximum generation length. For the first stage, the model can generate up to four hypotheses, while using fewer paths for simpler questions. For evidence use, table selection starts from L0 evidence, field selection uses L1 evidence, and refinement uses L2 evidence by default, with L3 evidence requested only when needed. The refinement loop is capped at eight turns. This design avoids giving the model uniformly expensive profiling information, while still allowing uncertain schema items to be checked with more detailed evidence when needed.

Component	**Configuration**
Models	
Temperature	1.0
Top-p	0.95
Top-k	40
Max tokens	16384
Multi-hypothesis grounding	
Maximum hypotheses	$K = 4$
Hypothesis	adaptive, 1–4
Table evidence	L0
Field evidence	L1
Agentic refinement	
Base evidence	L2
On-demand evidence	L3
Agent max turn	8
Uncertainty thresholds	
Required threshold	$\tau_{\text{req}} = 0.85$

Table 7: Summary of implementation settings.

H The Use of Large Language Models

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. 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.

I Tool Design

Figure 4 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.

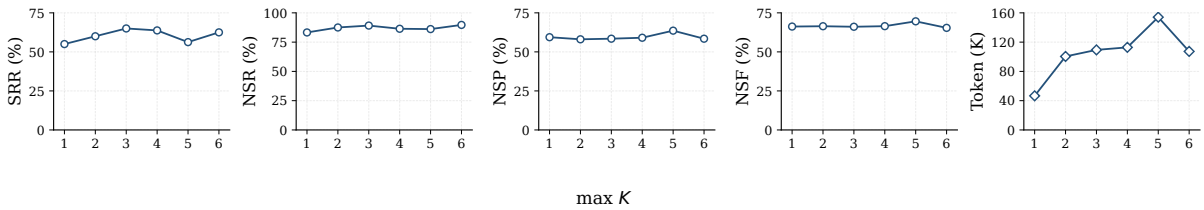

Figure 3: **Sensitivity to the hypothesis budget K.** Results are reported on the Spider2-Snow ablation subset. $K = 1$ corresponds to single-hypothesis grounding, while moderate multi-hypothesis settings provide a more stable balance between schema-linking quality and token cost. We use $K = 4$ as the default setting.

`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.

J System Prompts

Figures 5–8 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.

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 4: 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.

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
Decide the number of hypotheses yourself. Two error modes are equally bad: (a) emitting too few hypotheses, which silently locks downstream onto one mapping; (b) emitting redundant hypotheses that share the same table set and join graph. Let structural divergence signals — not perceived difficulty — drive the count. K=1 is correct only when no divergence signal applies.

[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 AND join graph.
✗ 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 5: 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 6: 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 7: 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 K 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]
retrieval_unit, connector, and surface_aligned are positive keep signals, not absolute rules. No tag is not evidence against keeping.
If discarding a tagged field, explicitly explain why the tag is misleading for this question.

[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. The run ends when the Uncertain Set is empty, recall recovery finds no further keeps, the agent stops calling tools, or the turn budget is reached. Remaining uncertain items are kept by default.

Figure 8: System prompt excerpt: uncertainty-guided agentic refinement.