SteinerSQL: Graph-Guided Mathematical Reasoning for Text-to-SQL Generation

Anonymous ACL submission

Abstract

Large Language Models (LLMs) struggle with complex Text-to-SQL queries that demand both sophisticated mathematical reasoning and intricate schema navigation. Existing methods often tackle these challenges in isolation, creating a fractured reasoning process that compromises logical and structural correctness. To resolve this, we introduce SteinerSQL, a framework that unifies these dual challenges into a single, graph-centric optimization problem. SteinerSQL operates in three stages: mathematical decomposition to identify required tables (terminals), optimal reasoning scaffold construction via a Steiner tree problem, and multi-level validation to ensure correctness. On the challenging LogicCat and Spider2.0-Lite benchmarks, SteinerSQL establishes a new state-of-the-art with 36.10% and 40.04% execution accuracy, respectively, using Gemini-2.5-Pro. Beyond accuracy, SteinerSQL presents a new, unified paradigm for Text-to-SQL, paving the way for more robust and principled solutions to complex reasoning tasks.

1 Introduction

Text-to-SQL generation converts natural-language questions into executable SQL, promising frictionless access to structured data (Yu et al., 2018; Katsogiannis-Meimarakis and Koutrika, 2023; Hong et al., 2024). While large language models (LLMs) achieve impressive performance on standard benchmarks (Gao et al., 2024a; Pourreza et al., 2025), they face significant challenges when confronted with complex mathematical reasoning that requires multi-step analytical computations across interconnected database relationships (Lei et al., 2025; Deng et al., 2025). Although recent advances suggest that modern LLMs can handle schema navigation more effectively when schema fit within context windows (Sun et al., 2023; Maamari et al., 2024; Chung et al., 2025), mathematical reasoning introduces different complexities that require structured approaches to preserve computational dependencies across schema relationships (Zheng et al., 2024; He et al., 2025; Stoisser et al., 2025).

The core challenge lies in mathematical reasoning complexity within SQL generation (Zheng et al., 2024; He et al., 2025). Questions involving statistical analysis, multi-level aggregations, conditional mathematics, and cross-table numerical dependencies create reasoning spaces that overwhelm current LLM approaches (Guo et al., 2024; Mitsopoulou and Koutrika, 2025; Yun and Lee, 2025). Specifically, current LLM-based approaches struggle with such mathematical reasoning for two reasons: (1) **Mathematical decomposition complexity**: LLMs lack systematic frameworks for breaking down complex mathematical operations into coherent SQL analytical workflows (Tai et al., 2023; Guo et al., 2024; He et al., 2025), often generating queries that compute incorrect intermediate results or miss essential mathematical relationships (Gao et al., 2024a; Mitsopoulou and Koutrika, 2025; Zhai et al., 2025). (2) **Schema navigation under mathematical constraints**: When mathematical operations require data from multiple tables, LLMs struggle to identify the optimal sufficient schema connections needed to support the computational logic (He et al., 2025; Shen et al., 2025), leading to either incomplete reasoning paths that break mathematical dependencies or excessive joins that introduce computational noise.

These challenges naturally map to a graph optimization problem. Prior work has shown that mathematical reasoning questions in Text-to-SQL correspond to finding optimal sufficient connections that preserve computational dependencies in the database schema graph (Baik et al., 2019; Nascimento et al., 2025). This correspondence emerges because mathematical operations inherently require specific data relationships to be maintained—aggregations need their constituent elements (Fent and Neumann, 2021) and calculations

demand complete computational chains from input to output (Weng et al., 2025; Shen et al., 2025).

This paper makes the following key contributions:

- We propose a novel approach that integrates mathematical decomposition, graph-based schema navigation, and multi-level validation into a cohesive pipeline. This method bridges the gap between logical reasoning and structural traversal by formulating the challenge as a Steiner tree optimization problem.
- Our framework establishes a new state-of-the-art on two challenging benchmarks, achieving 36.10% execution accuracy on LogicCat and 40.04% on Spider2.0-Lite.

2 Related Work

Structured Reasoning Gaps in Text-to-SQL. We state that LLM-based Text-to-SQL models face two critical challenges: mathematical decomposition complexity and schema navigation under mathematical constraints. Recent approaches address each challenge separately. For mathematical reasoning, STaR-SQL achieves 31.6% improvement through self-taught reasoning (He et al., 2025), while ExCoT improves execution accuracy from 57.37% to 68.51% on BIRD via execution-guided optimization (Zhai et al., 2025). For schema navigation, graph-based methods like TEMPLAR formulate join-path inference as Steiner tree search (Baik et al., 2019), while Nascimento et al. (Nascimento et al., 2025) retrieves minimal Steiner subgraphs to constrain generation. These approaches solve challenges in isolation—mathematical reasoning improvements ignore schema structure, while graph-based navigation methods delegate arithmetic operations entirely to LLMs, making the system unable to solve complex problems that require both an understanding of data structure and precise mathematical calculations.

3 The SteinerSQL Framework

Our framework, **SteinerSQL**, builds a principled bridge between the abstract mathematical logic of a question and the concrete structure of a database. We achieve this by casting the discovery of a valid reasoning path as a Steiner tree problem (Baik et al., 2019; Nascimento et al., 2025) on the schema graph. The goal is to find a *reasoning scaffold*—the lowest-cost subgraph that links all mathematically required tables (terminals) while preserving the full computational flow. This integrated approach is implemented as a three-stage pipeline: (i) **Mathematical Decomposition**, (ii) **Schema Navigation**, and (iii) **Multi-Level Validation**. We demonstrated two case examples how this pipeline is executed in Appendix D.

3.1 Overall Pipeline

In the SteinerSQL framework, the LLM acts as the core reasoning agent, guided by a structured system prompt organized into five components: Role-Play, Critical Requirements, Build Relation, Optimal Query Plan, and Behavioral Guidelines shown in Figure 1. This design creates a synergy between the LLM's language capabilities and the framework's deterministic graph algorithms. Initially, the LLM deconstructs the user's query to identify the required terminal tables. The framework's algorithmic core then takes these terminals, constructs a schema graph, and solves the Steiner tree problem to produce an optimal reasoning scaffold. This scaffold is subsequently passed back to the LLM, which translates the structured path into the final SQL query and performs a multi-level validation check to ensure correctness. The psuedocode of the LLM agent pipeline for this framework is shown in Algorithm 2 in Appendix A.

3.2 Stage 1: Mathematical Decomposition

This initial stage deconstructs the natural language question to identify all mathematical entities and their computational dependencies. This process establishes the formal requirements that will guide the construction of the reasoning scaffold.

Mathematical Entity Extraction. We systematically extract mathematical entities from natural language questions through multi-dimensional analysis: (1) Computational Keywords: Identify mathematical operations (SUM, COUNT, AVG, etc.) and their target attributes; (2) Quantitative Expressions: Extract numerical constraints, ranges, and comparison operators; (3) Aggregation Requirements: Detect grouping needs and hierarchical computations; (4) Temporal Dependencies: Identify time-based calculations and sequential operations.

Computational Dependency Analysis. After extracting mathematical entities, we analyze their computational dependencies to understand the logical flow through three key steps shown in Algorithm 1.

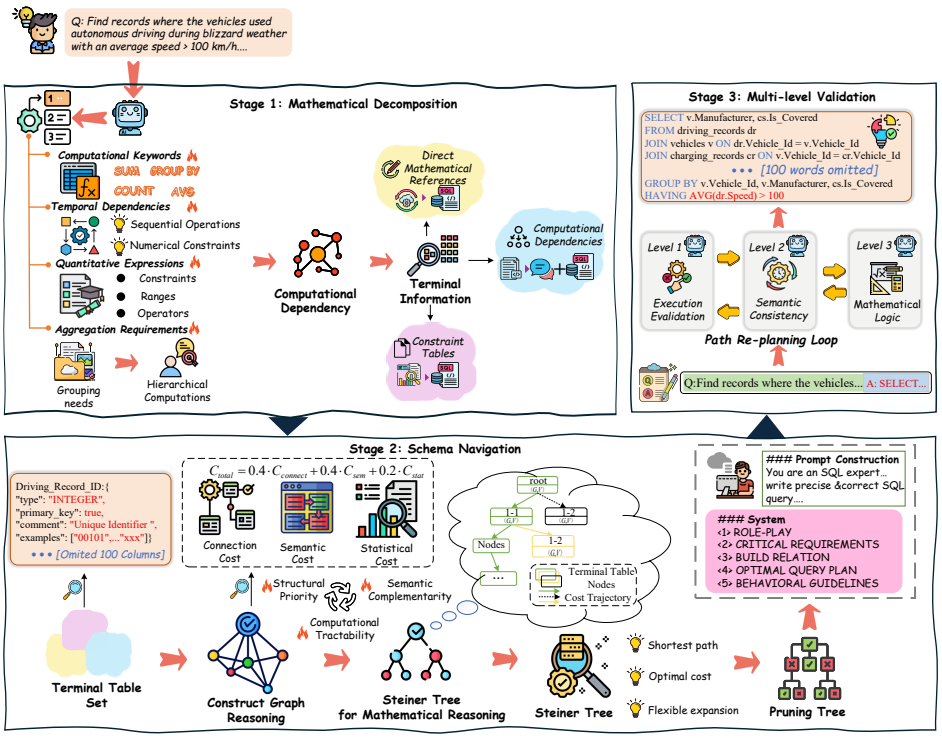

Figure 1: An overview of the SteinerSQL framework. The workflow consists of three main stages: (1) Mathematical Decomposition, which deconstructs the user's question into formal computational requirements and identifies terminal tables; (2) Schema Navigation, which models the problem as a Steiner tree on the schema graph to find an optimal reasoning scaffold; and (3) Multi-level Validation, which verifies the generated SQL for correctness and triggers a re-planning loop if errors are detected.

Terminal Table Identification. Based on the mathematical entity and dependency analysis, we identify the essential tables, or terminals, that are required to fulfill the query's mathematical logic: (1) Direct Mathematical References: Tables containing attributes mentioned in mathematical expressions; (2) Computational Dependencies: Tables required for intermediate calculations; (3) Constraint Tables: Tables needed to satisfy mathematical constraints and filters.

Thus, this stage produces a formal set of terminal tables, denoted as $\mathcal{T}_{\text{req}}$, which represent the indispensable schema components for the query's mathematical logic. This set $\mathcal{T}_{\text{req}}$ serves as the direct input—the set of terminals—for the Steiner tree problem we will formally define and solve in the subsequent Schema Navigation stage.

3.3 Stage 2: Schema Navigation

This stage takes the set of terminal tables $\mathcal{T}_{\text{req}}$ from Stage 1 and navigates the database schema to construct an optimal reasoning scaffold. To achieve this in a principled manner, we first formally define the search space as a schema graph and then formulate our navigation task as a Steiner tree problem on this graph.

3.3.1 Definitions

Definition 1 (Schema Graph). Given a database schema $\mathcal{S} = (\mathcal{T}, \mathcal{R})$, we define the schema graph as a weighted undirected graph $G_{\mathcal{S}} = (V, E, C)$, where: (1) The set of vertices V corresponds to the set of tables $\mathcal{T}$. (2) An edge $(t_i, t_j) \in E$ exists if a set of join predicates is met: a foreign key relationship exists between t_i and t_j, or their highest column name/type similarity exceeds a reproducible threshold τ. (3) C is a cost function assigning a positive weight $C(e)$ to each edge $e \in E$, representing join dis-utility. Since $G_{\mathcal{S}}$ is finite and $w(e) \geq 0$, such a minimizer exists.

Definition 2 (Steiner Tree on a Schema Graph). Given $G_{\mathcal{S}} = (V, E, C)$ and terminals $\mathcal{T}_{\text{req}} \subseteq V$, a Steiner tree is a connected, acyclic subgraph $T \subseteq G_{\mathcal{S}}$ that spans $\mathcal{T}_{\text{req}}$ and minimizes

$$C_{\text{total}}(T) = \sum_{e \in E(T)} C(e),$$

where $C(e) \geq 0$ and C_{total} is additive.

Algorithm 1 Mathematical Dependency Analysis

Input: Question q, Mathematical entities $\mathcal{E}_{\text{math}}$
Output: Dependency graph G_{dep}, Terminal set $\mathcal{T}_{\text{req}}$
Initialize: $G_{\text{dep}} \leftarrow$ empty graph, $\mathcal{T}_{\text{req}} \leftarrow \emptyset$
// Step 1: Data Flow Analysis
for each entity $e \in \mathcal{E}_{\text{math}}$ **do**
  $attr_{\text{req}} \leftarrow$ EXTRACTREQUIREDATTRIBUTES(e)
  $tables_{\text{direct}} \leftarrow$ FINDCONTAININGTABLES($attr_{\text{req}}$)
  $\mathcal{T}_{\text{req}} \leftarrow \mathcal{T}_{\text{req}} \cup tables_{\text{direct}}$
  Add dependency edges for attribute flow to G_{dep}
end for
// Step 2: Join Requirements Analysis
for each pair $(t_i, t_j) \in \mathcal{T}_{\text{req}}$ **do**
  if REQUIRESJOIN(t_i, t_j) based on mathematical operations **then**
    $join_{\text{path}} \leftarrow$ FINDJOINPATH(t_i, t_j)
    Add $join_{\text{path}}$ tables to $\mathcal{T}_{\text{req}}$
    Add join dependencies to G_{dep}
  end if
end for
// Step 3: Constraint Propagation
for each constraint c in mathematical entities **do**
  $affected_{\text{tables}} \leftarrow$ PROPAGATECONSTRAINT($c, \mathcal{T}_{\text{req}}$)
  Update $\mathcal{T}_{\text{req}}$ with constraint-required tables
  Add constraint edges to G_{dep}
end for
return $G_{\text{dep}}, \mathcal{T}_{\text{req}}$

3.3.2 Modeling Reasoning Scaffolds

Assumption. We assume (i) schema fidelity, (ii) terminal soundness, (iii) additive and monotone nonnegative edge costs, and (iv) operation locality. See Appendix A.1 (A1–A4) for precise statements.
.

Theorem. *Under Assumption, the minimum Steiner tree on $G_{\mathcal{S}}$ with terminals $\mathcal{T}_{req}$ corresponds to the optimal (most efficient and relevant) reasoning scaffold required for the mathematical query.*

Proof Sketch. The mapping is direct: (1) **Terminals** $\mathcal{T}_{\text{req}}$ are the tables essential for the query's mathematical logic. (2) **Connectivity** in the Steiner tree ensures a valid join path exists for all required tables, a prerequisite for a valid SQL query. (3) **Cost Minimization** finds the closest set of intermediate tables and joins, minimizing complexity.

3.3.3 Edge Cost Formulation

We construct the schema graph $G_{\mathcal{S}}$ using a holistic cost function where lower values indicate better joins. To ensure consistency, all similarity and correlation metrics used in the cost components are normalized to the range $[0, 1]$. The total cost is an aggregation of structural, semantic, and statistical dissimilarity[1]:

$$C_{\text{total}} = \alpha \cdot C_{\text{connect}} + \beta \cdot C_{\text{sem}} + \gamma \cdot C_{\text{stat}}. \quad (1)$$

The cost components are:

- **Connection Cost (C_{connect}):** Measures structural distance based on foreign keys (Guo et al., 2019; Li et al., 2021), name dissimilarity (Gan et al., 2023; Zhang et al., 2023), and type incompatibility (Bernstein et al., 2011). The table-pair similarities sim_{name} and sim_{type} are calculated by taking the maximum similarity score over column pairs between the tables.

$$\begin{aligned} C_{\text{connect}} = w_1 \cdot \mathbb{I}_{\neg\text{FK}}(t_i, t_j) \\ + w_2 \cdot (1 - \text{sim}_{\text{name}}(t_i, t_j)) \\ + w_3 \cdot (1 - \text{sim}_{\text{type}}(t_i, t_j)). \end{aligned} \quad (2)$$

  where $\mathbb{I}_{\neg\text{FK}}$ is an indicator function that is 0 if a direct FK exists and 1 otherwise. We set the internal weights equally.

- **Semantic Cost (C_{sem}):** Captures semantic distance using SentenceTransformer embeddings (all-MiniLM-L6-v2) (Reimers and Gurevych, 2019). A table's embedding vector $\mathbf{e}_t$ is derived by averaging the embeddings of its table name and all of its column names.

$$C_{\text{sem}} = 1 - \cos(\mathbf{e}_{t_i}, \mathbf{e}_{t_j}). \quad (3)$$

- **Statistical Cost (C_{stat}):** Measures statistical implausibility from join selectivity (Wu et al., 2023) and correlation (Zhu et al., 2023; Halford et al., 2020).

$$\begin{aligned} C_{\text{stat}} = w_4 \cdot (1 - \text{join_selectivity}(t_i, t_j)) \\ + w_5 \cdot (1 - \text{corr_strength}(t_i, t_j)). \end{aligned} \quad (4)$$

  The internal weights are set equally with $w_4 = 0.5$ and $w_5 = 0.5$.

[1] We reproducibly add similarity edges using a frozen column-name encoder (`all-MiniLM-L6-v2`) and a rule-based type match. For tables (t_i, t_j), compute $s = \max_{c_i \in t_i, c_j \in t_j} \left[\alpha \cos(e(c_i), e(c_j)) + (1-\alpha)\, \mathbf{1}\{\text{type}(c_i) = \text{type}(c_j)\}\right]$ with $\alpha = 0.85$, and add (t_i, t_j) iff $s \geq \tau$ with $\tau = 0.75$.

The weight selection for our multi-dimensional cost function is guided by the following theoretical principles: (1) Connection costs is important ($\alpha = \beta > \gamma$) because database integrity constraints provide the fundamental feasibility boundaries for any valid query (Codd, 1970; Aho et al., 1979); (2) Semantic costs should also exceed statistical costs ($\beta > \gamma$) as mathematical intent understanding is more critical than statistical optimization for correctness (Wang et al., 2022; Shen et al., 2025).

Based on these principles and empirical validation in Section 4.3, we derive our choice: $\alpha = 0.4, \quad \beta = 0.4, \quad \gamma = 0.2$. This allocation reflects the hierarchical importance: structural feasibility, semantic alignment to statistical optimization.

3.3.4 Steiner Tree Solution

We use the Kou–Markowsky–Berman (KMB) 2-approximation algorithm (Kou et al., 1981) to solve the Steiner Tree problem, which requires the graph distances to satisfy the triangle inequality. The solution process is as follows:

1. **Metric Closure:** First, we compute the metric closure of $G_{\mathcal{S}}$ by running an all-pairs shortest path algorithm (Floyd-Warshall). This creates a new complete graph $G'_{\mathcal{S}}$ where the edge weight between any two nodes is the shortest path distance in $G_{\mathcal{S}}$.

2. **MST on Terminals:** We construct a minimal spanning tree (MST) on the subgraph of $G'_{\mathcal{S}}$ induced by the terminal set $\mathcal{T}_{\text{req}}$.

3. **Scaffold Construction:** The edges of the MST are mapped back to their corresponding shortest paths in the original graph $G_{\mathcal{S}}$. The union of these paths forms a subgraph that connects all terminals.

4. **Pruning:** A final traversal removes any cycles to yield the final Steiner tree.

This principled approach guarantees the resulting scaffold's cost is at most twice that of the optimal solution (Kou et al., 1981). The time complexity of this approach is dominated by the Floyd-Warshall algorithm, resulting in $O(|V|^3)$, where $|V|$ is the number of tables in the schema. In cases where multiple paths or edges have identical costs during the shortest path or MST construction, we employ a deterministic tie-breaking rule based on the lexicographical order of the table and column names involved, ensuring reproducibility.

3.4 Stage 3: Multi-level Validation

To ensure final correctness, we employ a three-level validation process with precise checks designed to catch errors common in complex SQL queries. If a Level 2 or 3 error is detected, which indicates a flawed reasoning scaffold, our Path Re-planning Loop is triggered to generate a new scaffold with updated constraints.

Level 1: Execution Validation. This foundational check verifies that the generated SQL is syntactically correct and executable against the database schema. It primarily catches surface-level errors such as invalid SQL syntax, or incorrect table and column names that would cause immediate database execution failures.

Level 2: Semantic Consistency Validation. This level scrutinizes the query's logical alignment with the user's intent. Specific checks include verifying that all terminal tables identified in Stage 1 are present in the query's `FROM` or `JOIN` clauses, ensuring that join conditions use semantically appropriate columns and that the scaffold does not contain joins irrelevant to the query, and confirming that attributes in the `SELECT`, `WHERE`, and `HAVING` clauses correctly map to the entities mentioned in the natural language question.

Level 3: Mathematical Logic Validation. This is the deepest level, auditing the computational structure of the query. The checks are highly specific: (1) enforcing the rule that any non-aggregated column in the `SELECT` list also appears in the `GROUP BY` clause; (2) validating that the aggregation functions used (e.g., `AVG`, `SUM`, `COUNT`) and the columns they apply to directly match the operations requested in the prompt; (3) Checking that numerical constraints are correctly translated into `WHERE` or `HAVING` clauses with proper operators and values.

Path Re-planning Loop. When a semantic (Level 2) or mathematical (Level 3) error is detected, the framework treats it as a signal that the underlying reasoning scaffold is flawed. The validation error is translated into a new constraint for the graph search in Stage 2.

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets

We evaluate our work on four challenging Text-to-SQL benchmarks which are all publically available

[2]: (1) **LogicCat (Liu et al., 2025):** a challenging text-to-SQL benchmark comprising 4,038 English questions focusing on complex multi-domain reasoning including physical knowledge, mathematical logic, commonsense reasoning, and hypothetical scenarios in real-world setting. We use the official public subset includes 2,369 questions for evaluation. (2) **Spider (Yu et al., 2018)**: A foundational, large-scale, cross-domain Text-to-SQL benchmark. It consists of 10,181 questions and 200 databases, focusing on complex query structures and testing a model's ability to generalize to new database schemas. We use the 1,034 questions from the test set for evaluation. (3) **Spider2.0-Lite (Lei et al., 2025):** Spider 2.0 is a challenging benchmark that evaluates language models on 632 real-world enterprise text-to-SQL workflows. We use the derivative of Spider2.0-Lite which includes 547 questions and host on diverse databases such as Snowflake, BigQuery and SQLite. Our pipeline is adopted into successfully execution for this benchmarking evaluation. (4) **BIRD (Li et al., 2023):** BIRD is the classic benchmark for Text-to-SQL evaluation, which contains over 12,751 questions and 95 databases. We use the dev subset for evaluation which includes 1,534 questions.

4.1.2 Backbone LLMs

To evaluate the generalization of our framework, we integrate SteinerSQL with four diverse, state-of-the-art LLMs. Our selection is consistent with contemporary literature (Pourreza and Rafiei, 2023; Pourreza et al., 2025; Liu et al., 2025) in complex reasoning tasks: DeepSeek-R1-0528 (DeepSeek-AI et al., 2025), GPT-4o-2024-11-20 (OpenAI, 2024), GPT-4.1-2025-04-14 (OpenAI, 2025), and Gemini-2.5-Pro-preview-05-06 (Google Cloud, 2025).

4.1.3 Model Comparison

Our method is compared against nine state-of-the-art Text-to-SQL methods across LogicCat, Spider2.0-Lite, and BIRD-dev datasets. These **published baselines** include DIN-SQL (Pourreza and Rafiei, 2023), DAIL-SQL (Gao et al., 2024a), CHESS (Talaei et al., 2024), XiYanSQL-QwenCoder-32B (Gao et al., 2024b), Ask-Data (Shkapenyuk et al., 2025), CHASE-SQL (Pourreza et al., 2025), LinkAlign (Wang and Liu, 2025), OmniSQL-32B (Li et al., 2025), and ReFoRCE (Deng et al., 2025), representing diverse methodological approaches including decomposed prompting, in-context learning, hierarchical encoding, and reinforcement learning.

We also compare our SteinerSQL approach with *standard prompting* baselines applied to our four backbone LLMs (DeepSeek-R1, GPT-4o, GPT-4.1, and Gemini-2.5-Pro) across all datasets[3]. This standard prompting includes the same template of role-playing, critical requirements, and behavioral guidelines with chain-of-thought annotation (LogicCat), evidence and schema (BIRD and Spider) or external knowledge (Spider2.0-Lite) that the datasets provided, but without our specialized three-stage framework components.

4.1.4 Evaluation Metric

We adopt to use the widely-used metric: Execution Accuracy (EX) (Yu et al., 2018) as our primary metric. This evaluates where a query is correct only if its execution result matches that of the ground-truth query. For each example, we use the evaluation script to produce a score of either 0 or 1.

4.1.5 Implementation Details

Experiments were conducted on a cloud-based server with two NVIDIA A100 GPUs. LLM inference was managed via the Hugging Face repository and official APIs. We set all LLMs with a temperature of 0, which is in greedy decoding settings. Hyperparameters of all baselines are default associated with their original implementation and all other hyperparameters of LLMs are default. All databases are hosted and managed using SQLite except Spider-Lite and we follow the official documentations for functions and usage.

4.2 Main Results

4.2.1 Performance Comparison

As shown in Table 1, our method, SteinerSQL, significantly outperforms existing state-of-the-art (SOTA) frameworks and standard prompting baselines across all datasets and backbone LLMs.

On the complex reasoning benchmark LogicCat, SteinerSQL (Gemini-2.5-Pro) establishes a new SOTA with 36.10% EX. On Spider2.0-Lite, it also

[2]For the LogicCat benchmark, our experiments utilize the complete dataset, which is an extension of the publicly available version and is accessible upon request from the original authors.

[3]For LogicCat evaluation, we adopt the execution accuracy of the standard prompting with four backbones reported in published papers because we use the same prompting strategy. For Spider2.0-Lite specifically, we evaluate using the Spider-agent with our four backbones for standard prompting comparison.

Methods	LogicCat	Spider	Spider2.0	BIRD
Baselines				
DIN-SQL	19.28†[a]	85.30†[b]	1.46†[b]	50.72†[a]
DAIL-SQL	23.71†[a]	86.17†[b]	5.68†[b]	54.76†[a]
CHESS	33.20†‡[a]	87.23	3.84†[c]	68.31†[a]
XiYanSQL	20.11†	89.65†	–	69.03†
MiniSeek	–	**91.20**†‡	–	–
AskData	–	–	–	**75.36**†‡
CHASE	29.81†[a]	87.62†[c]	–	74.90†[d]
LinkAlign	–	–	33.09†‡	–
OmniSQL	22.41†	88.30	–	72.05†
ReFoRCE	–	–	30.35†	–
Standard				
DS-R1	28.17†	86.00	24.50	66.69
GPT-4o	28.03†	85.49	14.99	60.10
GPT-4.1	30.06†	85.30	13.35	64.80
Gemini-2.5	29.26†	86.27	26.87	70.08
SteinerSQL				
DS-R1	33.98	88.59	34.92	70.01
GPT-4o	32.00	87.43	25.05	64.93
GPT-4.1	35.04	87.23	26.33	67.08
Gemini-2.5	**36.10**	88.30	**40.04**	73.92

Table 1: Main results on Execution Accuracy (EX) in percentages (%) across datasets. Our method is presented with four different backbone LLMs. † indicates results from published papers/leaderboards. ‡ indicates current state-of-the-art on that dataset. – indicates results not available.

Backbone LLMs for baselines: [a] GPT-4o [b] GPT-4 [c] Gemini-1.5-Pro [d] Gemini-2.5-Pro

achieves a new SOTA of 40.04% EX, surpassing the previous best by over 3.6%. Furthermore, our method demonstrates highly competitive performance on BIRD-dev and Spider-test with 73.92% EX and 88.59% EX, closely approaching the current SOTA.

Figure 2 details our performance on LogicCat, where the advantage of SteinerSQL is most pronounced on the Hard difficulty subset. On these challenging queries, our methods achieve accuracies up to 26.0%, substantially outperforming standard prompting baselines. This highlights our workflow's superior ability to handle complex queries. When analyzed by reasoning category, SteinerSQL shows significant gains in tasks requiring mathematical and hypothesis-based reasoning, improving performance by up to 6 absolute points over standard methods.

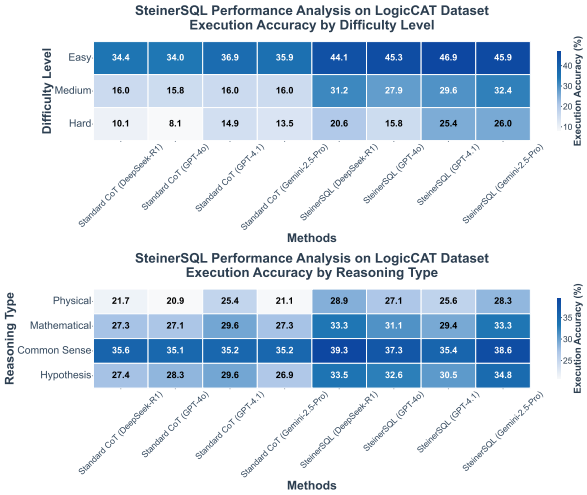

Figure 2: Performance heatmap of different methods on the LogicCat dataset across three difficulty levels (Easy, Medium, Hard) and different reasoning types. The heatmap reveals that SteinerSQL methods consistently outperform all standard prompting baselines across difficulty levels and reasoning types.

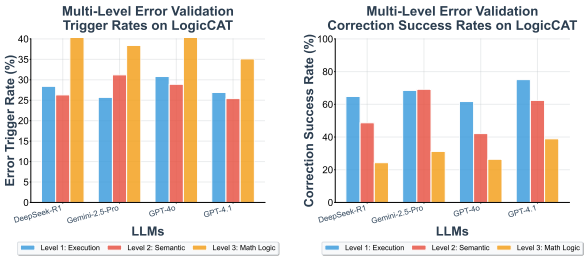

Figure 3: Analysis of the multi-level validation and correction mechanism on LogicCat. (Left) Trigger rates for each validation level, showing that semantic and mathematical logic errors are common. (Right) Correction success rates, demonstrating the effectiveness of our targeted correction strategies for each error type.

4.2.2 Error Validation Analysis

Our error validation analysis confirms that the primary challenges in complex Text-to-SQL are logical, not syntactic. As shown in Figure 3, the high rate of semantic (Level 2) and math logic (Level 3) errors validates our central premise that failures stem from flawed mathematical decomposition and schema navigation, rendering simple execution-only validation insufficient. Our results demonstrate the practical value of multi-level validation, particularly highlighting our principled path re-planning loop as essential for resolving these deep reasoning failures where simpler feedback mechanisms would fall short. This analysis confirms that a sophisticated, multi-level approach to error validation is not merely beneficial but indispens-

Configuration	DeepSeek-R1		GPT-4o		GPT-4.1		Gemini-2.5-Pro	
	EX	Drop	EX	Drop	EX	Drop	EX	Drop
Full Model (SteinerSQL)	**33.98**	-	**32.00**	-	**35.04**	-	**36.10**	-
Core Stage Isolation								
- Stage 2+3 Only (w/o Decomposition)	27.48	6.50	27.48	4.52	30.41	5.37	29.42	5.68
- Stage 1+3 Only (w/o Navigation)	28.13	5.85	25.46	6.54	29.17	5.87	30.01	6.09
- Stage 1+2 Only (w/o Validation)	28.24	5.74	26.83	5.17	30.29	4.75	31.44	4.66
Stage 2 Ablation								
- w/o Connection Cost	31.53	2.45	29.68	2.32	33.19	1.85	33.83	2.27
- w/o Semantic Cost	32.08	1.90	30.22	1.78	32.63	2.41	34.64	1.46
- w/o Statistical Cost	32.40	1.58	30.86	1.14	35.	0.89	34.28	1.82
- Remove Graph Module	29.17	4.81	26.29	5.71	29.83	5.21	30.87	5.23
- w/o Structured Examples	29.69	4.29	26.94	5.06	29.50	5.54	31.19	4.91

Table 2: Comprehensive ablation study of SteinerSQL components across four backbone models on the LogicCat dataset. We report Execution Accuracy (EX) and the absolute performance drop in percentages (%).

able for tackling the true complexity of advanced Text-to-SQL tasks.

4.3 Ablation and Component Analysis

Component Contribution. Our ablation study, summarized in Table 2,allows us to analyze the distinct role and contribution of each component within our framework. The results confirm that while the integrated system achieves optimal performance, each stage offers significant, stand-alone value. The Schema Navigation stage is the primary driver of performance, with its removal causing the largest drop (6.09%). This highlights its central role in structuring the reasoning path. The Math Decomposition stage provides the essential logical foundation for this navigation, and its removal also leads to a major performance decrease (5.52%). This suggests that our decomposition approach could potentially benefit other graph-based Text-to-SQL systems by improving their input formulation. Finally, the Multi-level Validation stage acts as a powerful refinement module, responsible for a 5.08% gain, demonstrating its utility as a post-processing mechanism. Further analysis shows that the Graph Module (5.24% drop) and the use of Structured Examples (4.95% drop) both serve as powerful guidance mechanisms. Within the graph, the different cost functions provide finer-grained signals for path selection.

Design Choice Justification. Figure 4 validates our core design choices by comparing the full model against variants using alternative algorithms. Our model consistently outperforms simpler graph traversal algorithms, including Shortest Path Combination and MST on Terminals, by a margin of 1-3 percentage points across all tested LLMs.

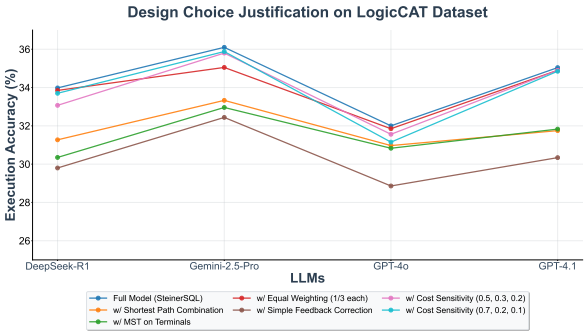

Figure 4: Design choice justification on the LogicCat dataset. The full SteinerSQL model consistently outperforms variants using alternative path-finding algorithms (Shortest Path Combination, MST on Terminals), cost sensitivity, and correction strategies (Simple Feedback).

This validates that Steiner trees provide a more accurate model in delivering sophisticated SQL queries. Also, the use of a principled, weighted cost function describe in Section 3.3.3 surpasses other weighting. Finally, our multi-level validation mechanism consistently outperforms a Simple Feedback loop, proving that a structured approach of validation is more effective.

5 Conclusion

In this work, we introduced SteinerSQL, a framework that bridges the critical gap between mathematical reasoning and schema navigation in complex Text-to-SQL generation. This method establishes new state-of-the-art on challenging benchmarks. Collectively, our work introduces a new, principled paradigm for complex Text-to-SQL that demonstrates the transformative potential of integrated optimization approaches in building more powerful reasoning systems for SQL generation.

6 Limitation

While SteinerSQL demonstrates strong performance, we identify two areas for future exploration. First, the framework's initial mathematical decomposition relies on the reasoning capabilities of the backbone LLM. Although our multi-level validation mitigates errors, advancements in more structured or hybrid decomposition techniques could further improve the initial identification of terminal tables. Second, our edge cost function, while principled, uses fixed weights. Future work could explore adaptive weighting schemes that dynamically adjust to the specific query or database schema, potentially offering more precise path-finding.

References

Alfred V Aho, Catriel Beeri, and Jeffrey D Ullman. 1979. The theory of joins in relational databases. *ACM Transactions on Database Systems (TODS)*, 4(3):297–314.

Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with sql query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 374–385. IEEE.

Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701.

Yeounoh Chung, Gaurav T Kakkar, Yu Gan, Brenton Milne, and Fatma Ozcan. 2025. Is long context all you need? leveraging llm's extended context for nl2sql. *arXiv preprint arXiv:2501.12372*.

Edgar F Codd. 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:arXiv:2501.12948.

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, format restriction, and column exploration. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.

Philipp Fent and Thomas Neumann. 2021. A practical approach to groupjoin and nested aggregates. *Proceedings of the VLDB Endowment*, 14(11):2383–2396.

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

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024a. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024b. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql. *Preprint*, arXiv:2411.08599.

Google Cloud. 2025. Gemini 2.5 pro.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Ziming Guo, Chao Ma, Yinggang Sun, Tiancheng Zhao, Guangyao Wang, and Hai Huang. 2024. Evaluating and enhancing llms for multi-turn text-to-sql with multiple question types. *arXiv preprint arXiv:2412.17867*.

Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2020. Selectivity estimation with attribute value dependencies using linked bayesian networks. *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLVI*, pages 154–188.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Star-sql: Self-taught reasoner for text-to-sql. *arXiv preprint arXiv:2502.13550*.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *CoRR*, abs/2406.08426.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Lawrence Kou, George Markowsky, and Leonard Berman. 1981. A fast algorithm for steiner trees. *Acta informatica*, 15:141–145.

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

Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting things into context: Rich explanations for query answers using join graphs. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1051–1063.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Preprint*, arXiv:2503.02240.

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. 2023. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Tao Liu, Hongying Zan, Yifan Li, Dixuan Zhang, Lulu Kong, Haixin Liu, Jiaming Hou, Aoze Zheng, Rui Li, Yiming Qiao, Zewei Luo, Qi Wang, Zhiqiang Zhang, Jiaxi Li, Supeng Liu, Kunli Zhang, and Min Peng. 2025. Logiccat: A chain-of-thought text-to-sql benchmark for multi-domain reasoning challenges. *Preprint*, arXiv:2505.18744.

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. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of text-to-sql benchmarks: limitations, challenges and opportunities. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025*, pages 199–212. OpenProceedings.org.

Eduardo R Nascimento, Caio Viktor S Avila, Yenier T Izquierdo, Grettel M García, Lucas Feijó L Andrade, Michelle SP Facina, Melissa Lemos, and Marco A Casanova. 2025. Text-to-sql based on large language models and database keyword search. *arXiv preprint arXiv:2501.13594*.

OpenAI. 2024. Gpt-4o.

OpenAI. 2025. Gpt-4.1.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Jiawei Shen, Chengcheng Wan, Ruoyi Qiao, Jiazhen Zou, Hang Xu, Yuchen Shao, Yueling Zhang, Weikai Miao, and Geguang Pu. 2025. A study of in-context-learning-based text-to-sql errors. *arXiv preprint arXiv:2501.09310*.

Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. 2025. Automatic metadata extraction for text-to-sql. *arXiv preprint arXiv:2505.19988*.

Josefa Lia Stoisser, Marc Boubnovski Martell, and Julien Fauqueur. 2025. Sparks of tabular reasoning via text2sql reinforcement learning. *arXiv preprint arXiv:2505.00016*.

Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and 1 others. 2023. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *arXiv preprint arXiv:2306.00739*.

Chang-Yu Tai, Ziru Chen, TIANSHU ZHANG, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-SQL. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *Preprint*, arXiv:2405.16755.

Jun Wang, Patrick Ng, Alexander Hanbo Li, Jiarong Jiang, Zhiguo Wang, Bing Xiang, Ramesh Nallapati, and Sudipta Sengupta. 2022. Improving text-to-SQL semantic parsing with fine-grained query understanding. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 306–312, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Han Weng, Puzhen Wu, Cui Longjie, Yi Zhan, Boyi Liu, Yuanfeng Song, Dun Zeng, Yingxiang Yang, Qianru Zhang, Dong Huang, and 1 others. 2025. Graph-reward-sql: Execution-free reinforcement learning for text-to-sql via graph matching and stepwise reward. *arXiv preprint arXiv:2505.12380*.

Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. Factor-join: a new cardinality estimation framework for join queries. *Proceedings of the ACM on Management of Data*, 1(1):1–27.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Janghyeon Yun and Sang-goo Lee. 2025. Seed: Enhancing text-to-sql performance and practical usability through automatic evidence generation. *arXiv preprint arXiv:2506.07423*.

Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. 2025. Excot: Optimizing reasoning for text-to-sql with execution feedback. *arXiv preprint arXiv:2503.19988*.

Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. 2023. NameGuess: Column name expansion for tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13276–13290, Singapore. Association for Computational Linguistics.

Danna Zheng, Mirella Lapata, and Jeff Pan. 2024. Archer: A human-labeled text-to-SQL dataset with arithmetic, commonsense and hypothetical reasoning. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 94–111, St. Julian's, Malta. Association for Computational Linguistics.

Zichen Zhu, Xiao Hu, and Manos Athanassoulis. 2023. Nocap: Near-optimal correlation-aware partitioning joins. *Proceedings of the ACM on Management of Data*, 1(4):1–27.

A Methodology Appendix

We demonstrate the LLM agent pipeline for the SteinerSQL framework in Algorithm 2 and shown the proof of the theorem proposed in Section 3.3.2.

A.1 Proof

We formalize the correspondence between minimum Steiner trees on the schema graph and optimal reasoning scaffolds for answering a mathematical query. Let $G_{\mathcal{S}} = (V, E, C)$ be the schema graph from the main text. For notational convenience, we set $w \equiv C$ so that $w(e) = C(e)$ for all $e \in E$, and we assume $w(e) \geq 0$. We also write

$$C_{\text{total}}(H) \;:=\; \sum_{e \in E(H)} w(e)$$

for any subgraph $H \subseteq G_{\mathcal{S}}$.

Edges E comprise (i) declared foreign-key/primary-key (or other well-typed equi-join) links and (ii) reproducible similarity-based links whose maximum column-name/type similarity is at least τ, as in Definition 1 of the main text.

All statements below are conditioned on Assumption 1 (A1–A4) from the main text: schema fidelity, terminal soundness, additive and monotone surrogate edge costs, and operation locality.

Existence. Since $|V| < \infty$ and $w(e) \geq 0$, a minimum-cost connected subgraph spanning $\mathcal{T}_{\text{req}}$ (hence a minimum Steiner tree) exists.

Assumption. We rely on the following standard conditions.

1. **Schema fidelity.** If a set of tables induces a connected subgraph of $G_{\mathcal{S}}$, then there exists a well-typed equi-join plan over those tables; conversely, every well-typed join plan over base tables uses only edges present in $G_{\mathcal{S}}$.

2. **Terminal soundness.** Every correct answer to the query depends on attributes drawn from all tables in $\mathcal{T}_{\text{req}}$ (no proper subset suffices).

3. **Additive, monotone surrogate cost.** The complexity/efficiency proxy of a reasoning scaffold decomposes additively over used joins:

$$\text{cost}(\text{scaffold}) \;=\; \sum_{e \in E(\text{scaffold})} w(e),$$

   and adding edges/joins does not decrease the cost (i.e., the surrogate is monotone in edge inclusion).

4. **Operation locality.** Selections, projections, grouping, and aggregation required by the query can be implemented on top of any join plan that correctly connects $\mathcal{T}_{\text{req}}$; they do not require additional base tables beyond those appearing in the scaffold.

Definition 1 (Reasoning scaffold)**.** *A reasoning scaffold for the query is any connected subgraph $H = (V_H, E_H)$ of $G_{\mathcal{S}}$ such that $\mathcal{T}_{req} \subseteq V_H$ and there exists a well-typed SQL plan whose FROM/JOIN component uses exactly E_H and whose SELECT/WHERE/GROUP BY/HAVING components implement the query's non-join operators. An* optimal *scaffold is one minimizing* $\text{cost}(H) = \sum_{e \in E_H} w(e)$.

Algorithm 2 SteinerSQL LLM Agent Framework Pipeline

Input: Natural language question q, Database schema $\mathcal{S} = (\mathcal{T}, \mathcal{R})$
Output: Valid SQL query Q_{sql} or error message
// Stage 1: Mathematical Decomposition
$\mathcal{E}_{\text{math}} \leftarrow$ LLM-EXTRACTMATHENTITIES(q)
$\mathcal{T}_{\text{req}} \leftarrow$ MATHEMATICALDEPENDENCYANALYSIS($q, \mathcal{E}_{\text{math}}$)
for $iteration = 1$ to 3 **do**
  // Stage 2: Schema Navigation
  $G_{\mathcal{S}} \leftarrow$ CONSTRUCTSCHEMAGRAPH($\mathcal{S}$) {Create vertices $V = \mathcal{T}$}
  Add edges (t_i, t_j) if FK exists or sim$(t_i, t_j) \geq 0.75$
  Compute edge costs: $C = 0.4 \cdot C_{\text{connect}} + 0.4 \cdot C_{\text{sem}} + 0.2 \cdot C_{\text{stat}}$
  // Steiner Tree Solution
  $G'_{\mathcal{S}} \leftarrow$ FLOYDWARSHALL($G_{\mathcal{S}}$) {Metric closure}
  $MST \leftarrow$ MINSPANNINGTREE($G'_{\mathcal{S}}[\mathcal{T}_{\text{req}}]$)
  $T_{steiner} \leftarrow$ MAPTOORIGINALPATHS($MST, G_{\mathcal{S}}$) and prune cycles
  // LLM Query Generation with Structured Prompt
  $prompt \leftarrow$ BUILDSYSTEMPROMPT($T_{steiner}$) {5 components: role, requirements, relations, plan, guidelines}
  $Q_{sql} \leftarrow$ LLM-GENERATESQL($prompt, q$)
  // Stage 3: Multi-level Validation
  $level1 \leftarrow$ CHECKSYNTAXANDEXECUTABILITY($Q_{sql}, \mathcal{S}$)
  if NOT $level1$ **then**
    return "Syntax Error"
  end if
  $level2 \leftarrow$ CHECKSEMANTICCONSISTENCY($Q_{sql}, q, \mathcal{T}_{\text{req}}$)
  $level3 \leftarrow$ CHECKMATHLOGIC($Q_{sql}, q, \mathcal{E}_{\text{math}}$)
  if $level2$ AND $level3$ **then**
    return Q_{sql}
  else
    $\mathcal{T}_{\text{req}} \leftarrow$ UPDATETERMINALS($\mathcal{T}_{\text{req}}, level2, level3$)
  end if
end for
return "Max iterations reached"

Definition 2 (Steiner tree)**.** *A (vertex-)Steiner tree for $(G_{\mathcal{S}}, \mathcal{T}_{req})$ is a connected, cycle-free subgraph $T = (V_T, E_T)$ of $G_{\mathcal{S}}$ that spans $\mathcal{T}_{req}$ and is of minimum total weight $\sum_{e \in E_T} w(e)$ among all connected subgraphs that contain $\mathcal{T}_{req}$.*

We prove the theorem via (i) soundness of Steiner trees as valid scaffolds, (ii) completeness of scaffolds as terminal-connecting subgraphs, and (iii) optimality via cost alignment.

A.1.1 Soundness: Steiner trees yield valid scaffolds

Lemma 1 (Soundness)**.** *Let T be any Steiner tree for $(G_{\mathcal{S}}, \mathcal{T}_{req})$. Then T is a valid reasoning scaffold.*

Proof. By definition, T is connected and $\mathcal{T}_{\text{req}} \subseteq V_T$. By (A1) Schema fidelity, any connected subgraph of $G_{\mathcal{S}}$ induces a well-typed equi-join plan over its vertices using exactly E_T. By (A4) Operation locality, the non-join operators of the mathematical query (selections, projections, grouping, aggregation) can be layered on top of this join plan without introducing new base tables. Hence there exists a complete SQL plan whose JOIN component uses E_T and that computes the desired quantity, making T a valid scaffold. □

A.1.2 Completeness: Every valid scaffold connects the terminals

Lemma 2 (Completeness)**.** *Let H be any valid reasoning scaffold. Then H is a connected subgraph of $G_{\mathcal{S}}$ that spans $\mathcal{T}_{req}$.*

Proof. By definition of scaffold and (A1), H is a

subgraph of $G_{\mathcal{S}}$ over which a well-typed join plan exists; such a plan requires connectivity among the used tables, hence H is connected. By (A2) Terminal soundness, any correct answer depends on all tables in $\mathcal{T}_{\text{req}}$, so these must appear in the FROM/JOIN portion; therefore $\mathcal{T}_{\text{req}} \subseteq V_H$. □

A.1.3 Cost preservation and cycle pruning

Lemma 3 (Cycle pruning)**.** *Proof.* If H contains a cycle C, remove any edge $e \in C$. In an undirected graph, deleting one edge on a cycle preserves connectivity; thus $H \setminus \{e\}$ remains connected and still spans $\mathcal{T}_{\text{req}}$. Because $w(e) \geq 0$, the cost does not increase and strictly decreases if $w(e) > 0$. Iterate until no cycles remain to obtain H'. □

Proof. If H contains a cycle C, remove any edge $e \in C$ that does not disconnect $\mathcal{T}_{\text{req}}$ (such an edge exists because cycles have redundant connectivity). Since $w(e) \geq 0$ by (A3), removing e does not increase cost and strictly decreases it if $w(e) > 0$. Repeat until no cycles remain. Connectivity and coverage of $\mathcal{T}_{\text{req}}$ are preserved at each step, yielding H' with $\text{cost}(H') \leq \text{cost}(H)$. □

Lemma 4 (Cost preservation from plans to graphs)**.** *Under (A3), the surrogate cost of any valid reasoning scaffold equals the sum of weights over its join edges, and adding superfluous joins (edges) cannot reduce this cost.*

Proof. Immediate from additivity and monotonicity in (A3). □

A.1.4 Optimality and equivalence

Proposition 1 (From Steiner optimality to scaffold optimality)**.** *Let T be a minimum Steiner tree for $(G_{\mathcal{S}}, \mathcal{T}_{req})$. Then T achieves the minimum surrogate cost among all valid reasoning scaffolds.*

Proof. By Lemma 1, T is a valid scaffold. Consider any valid scaffold H. By Lemma 2, H is a connected subgraph spanning $\mathcal{T}_{\text{req}}$. By Lemma 3, there exists a cycle-free $H' \subseteq H$ that still spans $\mathcal{T}_{\text{req}}$ with $\text{cost}(H') \leq \text{cost}(H)$. Since T is the minimum-cost connected, cycle-free subgraph spanning $\mathcal{T}_{\text{req}}$, we have $\text{cost}(T) \leq \text{cost}(H') \leq \text{cost}(H)$. By Lemma 4, these costs coincide with surrogate scaffold costs. Hence no valid scaffold beats T. □

Proposition 2 (From scaffold optimality to Steiner optimality)**.** *If $H^\star$ is an optimal reasoning scaffold, then there exists a minimum Steiner tree T with* $\text{cost}(T) = \text{cost}(H^\star)$.

Proof. By Lemma 2, $H^\star$ is a connected subgraph spanning $\mathcal{T}_{\text{req}}$. Apply Lemma 3 to obtain a cycle-free $H' \subseteq H^\star$ with $\text{cost}(H') \leq \text{cost}(H^\star)$. Optimality of $H^\star$ forces equality, so H' is also optimal among connected subgraphs spanning $\mathcal{T}_{\text{req}}$. Hence H' is a minimum Steiner tree T. □

Proof of Theorem. By Proposition 1, any minimum Steiner tree T is an optimal reasoning scaffold under the surrogate cost. Conversely, by Proposition 2, any optimal scaffold collapses (by cycle pruning) to a minimum Steiner tree of equal cost. Together with (A4) ensuring that non-join operators do not require additional base tables beyond those in the scaffold, this proves the equivalence and establishes that the minimum Steiner tree corresponds exactly to the most efficient and relevant scaffold. □

Remarks and edge cases. (1) *Non-uniqueness.* If multiple minimum Steiner trees exist, each is an optimal scaffold under the surrogate; downstream tie-breaking (e.g., favoring lower estimated cardinality) can be applied without affecting Theorem. (2) *Bridge tables as Steiner nodes.* Many-to-many link tables naturally appear as Steiner vertices; their inclusion is compelled by connectivity and captured by the cost. (3) *Self-joins and aliases.* If a base table must be joined to itself, represent each alias as a distinct vertex with an alias edge to preserve correctness of the mapping; the above arguments remain unchanged. (4) *Selections and filters.* Predicate pushdown may *reduce* runtime but does not change the necessity of connecting $\mathcal{T}_{\text{req}}$; our result concerns the structural scaffold.

Scope and limitations. Theorem concerns the JOIN skeleton only: selections/projections/grouping/aggregation are implemented on top of the scaffold without introducing extra base tables (operation locality). Cases requiring external sources or materialized auxiliaries (e.g., complex windowed analytics) are outside our guarantee. Moreover, if Stage 1 under- or over-identifies $\mathcal{T}_{\text{req}}$, the selected scaffold may deviate; our validation and re-planing loop updates $\mathcal{T}_{\text{req}}$ when inconsistencies are detected.

B Experiment Details

Hardware & Operation. All experiments ran on a single node with 2× NVIDIA A100 GPUs, 64 CPU cores, and 512 GB RAM with Ubuntu 22.04.3 LTS, CUDA12.1.

Software Stack. Python 3.10.13; PyTorch 2.3.1+cu121; transformers 4.43; accelerate 0.31; datasets 2.19; sentence-transformers 2.7; networkx 3.3; sqlglot 23.x; sqlite3 3.45.

Decoding and Runtime Configuration. Unless specified otherwise, we use *greedy decoding*:

- temperature 0.0, top_p/top_k disabled; repetition/presence/frequency penalties 1.0/0/0;
- max input context 32,768 tokens (subject to model limits), max new tokens 512;
- stop sequences: `{";\n-END-", "<eot_id>"}`.

Batched inference is used only where supported by the provider; otherwise, calls are sequential. Concurrency is capped to respect provider rate limits; failed calls are retried up to three times with exponential backoff.

C Ablation Results

To demonstrate that the contributions of Steiner-SQL's components are robust and not specific to a single dataset, we replicated our main ablation study across all three other benchmarks:BIRD-dev, Spider1.0 and Spider2.0-Lite benchmarks. The results, presented in Tables 3, 4 and 5, show generally consistent trends of ablation.

D Cases

Table 6 and 7 are execution pipeline example in LogicCat, and the question is "If the status of a data collector shows 'shutdown' and its installation altitude is 3000 meters, assuming the last data collection record before shutdown indicates a temperature of -10°C, please calculate the atmospheric pressure value at that location and analyze possible reasons for the shutdown." Table 8 and 9 are execution pipeline example in Spider2.0-Lite, and the question is "Between April 1 and July 31 of 2017, using the hits product revenue data along with the totals transactions to classify sessions as purchase (transactions ≥ 1 and productRevenue not null) or non-purchase (transactions null and productRevenue null), compare the average pageviews per visitor for each group by month."

Table 3: Ablation study on BIRD-dev dataset across four LLM backbones. We report Execution Accuracy (EX) in percentages (%) and the absolute performance drop compared to the full model.

	DeepSeek-R1		GPT-4o		GPT-4.1		Gemini-2.5-Pro	
Configuration	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**
Full Model (SteinerSQL)	**70.01**	-	**64.93**	-	**67.08**	-	**73.92**	-
Core Stage Isolation								
- Stage 2+3 Only (w/o Decomposition)	61.80	8.21	57.30	7.63	60.17	6.91	65.19	8.73
- Stage 1+3 Only (w/o Navigation)	65.45	4.56	61.08	3.85	63.89	3.19	70.14	3.78
- Stage 1+2 Only (w/o Validation)	63.10	6.91	58.80	6.13	59.71	7.37	67.01	6.91
Stage 2 Ablation								
- w/o Connection Cost	68.78	1.23	65.15	-0.98	65.71	1.37	72.82	1.10
- w/o Semantic Cost	69.10	0.91	63.23	1.70	68.12	-1.04	72.56	1.36
- w/o Statistical Cost	70.14	-0.13	64.99	-0.06	65.91	1.17	73.99	-0.07
- Remove Graph Module	64.86	5.15	58.93	6.00	61.02	6.06	68.19	5.73
- w/o Structured Examples	65.32	4.69	59.26	5.67	62.13	4.95	69.17	4.75

Table 4: Ablation study on Spider2.0-Lite dataset across four LLM backbones. We report Execution Accuracy (EX) in percentages (%) and the absolute performance drop compared to the full model.

	DeepSeek-R1		GPT-4o		GPT-4.1		Gemini-2.5-Pro	
Configuration	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**
Full Model (SteinerSQL)	**34.92**	-	**25.05**	-	**26.33**	-	**40.04**	-
Core Stage Isolation								
- Stage 2+3 Only (w/o Decomposition)	27.61	7.31	19.93	5.12	19.93	6.40	33.09	6.95
- Stage 1+3 Only (w/o Navigation)	29.80	5.12	21.21	3.84	22.67	3.66	35.28	4.76
- Stage 1+2 Only (w/o Validation)	28.15	6.77	19.01	6.04	19.56	6.77	33.46	6.58
Stage 2 Ablation								
- w/o Connection Cost	34.92	0.00	23.95	1.10	26.51	-0.18	39.12	0.92
- w/o Semantic Cost	33.46	1.46	23.40	1.65	27.80	-1.47	39.55	1.49
- w/o Statistical Cost	34.82	-1.10	24.13	0.92	25.78	0.55	38.94	1.10
- Remove Graph Module	29.62	5.30	18.83	6.22	20.66	5.67	34.19	5.85
- w/o Structured Examples	30.16	4.76	19.38	5.67	20.84	5.49	34.92	5.12

Table 5: Ablation study on Spider-test dataset across four LLM backbones. We report Execution Accuracy (EX) in percentages (%) and the absolute performance drop compared to the full model.

	DeepSeek-R1		GPT-4o		GPT-4.1		Gemini-2.5-Pro	
Configuration	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**	**EX**	**Drop**
Full Model (SteinerSQL)	**88.59**	-	**87.43**	-	**87.23**	-	**88.30**	-
Core Stage Isolation								
- Stage 2+3 Only (w/o Decomposition)	80.56	8.03	80.46	6.97	80.27	6.96	79.30	9.00
- Stage 1+3 Only (w/o Navigation)	84.62	3.97	83.46	3.97	84.24	2.99	83.27	5.03
- Stage 1+2 Only (w/o Validation)	82.59	6.00	81.43	6.00	80.27	6.96	82.30	6.00
Stage 2 Ablation								
- w/o Connection Cost	89.07	-0.48	86.46	0.97	86.07	1.16	87.33	0.97
- w/o Semantic Cost	87.62	0.97	85.98	1.45	86.27	0.96	87.14	1.16
- w/o Statistical Cost	87.62	0.97	88.39	-0.96	86.75	0.48	89.46	-1.16
- Remove Graph Module	83.56	5.03	81.43	6.00	81.24	5.99	83.27	5.03
- w/o Structured Examples	84.62	3.97	82.40	5.03	82.21	5.02	84.33	3.97

Table 6: **Case of SteinerSQL Pipeline Execution in LogicCat (Part 1): Query and Decomposition**

SteinerSQL Pipeline Execution for Collector bq004	
Component	**Details and Values**
Question	If the status of a data collector shows 'shutdown' and its installation altitude is 3000 meters, assuming the last data collection record before shutdown indicates a temperature of -10°C, please calculate the atmospheric pressure value at that location and analyze possible reasons for the shutdown.

Final Procedure	`/* Fictional Procedure for Calculation and Analysis */`
	`1. SELECT installation_altitude_m, last_temp_c FROM collectors WHERE status = 'shutdown' AND collector_id = 'bq004';`
	`2. INPUT altitude (h) and temperature (T) INTO Barometric_Formula(h, T);`
	`3. RETURN calculated_pressure;`
	`4. ANALYZE environmental_factors (T, h, calculated_pressure) FOR shutdown_causation;`

Stage 1: Mathematical Decomposition	
1a. Mathematical	- **Calculation:** Atmospheric pressure using the Barometric Formula: $P = P_0 \cdot e^{-\frac{gMh}{RT}}$ - **Constants:** - Sea level pressure (P_0): 101325 Pa - Gravitational acceleration (g): 9.80665 m/s² - Molar mass of Earth's air (M): 0.0289644 kg/mol - Universal gas constant (R): 8.31446 J/(mol·K) - **Variables:** - Altitude (h): 3000 m - Temperature (T): -10°C, which must be converted to Kelvin (263.15 K). - **Analysis:** Qualitative assessment of failure modes based on environmental data.
1b. Computational	- The calculation requires `installation_altitude` and `last_temperature`. - The analysis requires the `status`, `last_temperature`, and the calculated pressure.
1c. Terminal Table	- A conceptual schema is assumed for this problem. - `collectors` (contains `collector_id`, `status`, `installation_altitude_m`) - `readings` (contains `collector_id`, `timestamp`, `temperature_c`) **Resulting Terminals $\mathcal{T}_{\text{req}}$:** $\{$`collectors`, `readings`$\}$

Table 7: **Case of SteinerSQL Pipeline Execution in LogicCat (Part 2): Navigation and Validation**

Stage 2: Schema Navigation	
2a. Schema Graph	- **Vertices** V**:** $\{$`collectors`, `readings`$\}$ - **Edges** E **and Costs** $C(e)$**:** - A direct foreign key relationship exists between the tables. - $C(\text{collectors, readings}) = 0.05$ (Direct FK link, cost is very low).
2b. Steiner Tree	- **Problem:** Find the path to link collector properties with their latest readings. - **Terminals:** $\{$`collectors`, `readings`$\}$ - **Reasoning Scaffold:** The resulting tree is a direct join: **`collectors – readings`**. This links the static altitude data to the last recorded temperature.
Stage 3: Multi-level Validation & Execution	
3a. Level 1:	- **Check:** Is the procedural logic sound? - **Result:** Pass. The procedure correctly identifies the necessary inputs, applies the correct physical formula, and outlines an analysis stage.
3b. Level 2:	- **Check:** Does the procedure align with the user's intent? - **Result:** Pass. It directly addresses both parts of the question: calculating the pressure and analyzing the shutdown.
3c. Level 3:	- **Check:** Is the mathematical logic and final analysis correct? - **Result:** Pass. - **Calculation:** - $T = -10 + 273.15 = 263.15$ K - Exponent: $-\frac{(9.80665 \cdot 0.0289644 \cdot 3000)}{(8.31446 \cdot 263.15)} = -\frac{851.65}{2187.95} \approx -0.3892$ - $P = 101325 \cdot e^{-0.3892} = 101325 \cdot 0.6776 \approx 68669.5$ Pa - Converting to hectopascals: 68669.5 Pa $=$ **686.7 hPa**. (Note: Using a standard atmosphere model gives $\sim$700 hPa; this direct calculation is more precise for the given temperature). - **Shutdown Analysis:** - **Low Temperature Failure:** At -10°C, battery performance drops significantly and can lead to a low-voltage shutdown. This is the most probable cause. - **Icing:** Moisture could freeze on sensors, anemometers, or solar panels (if present), causing mechanical failure or power loss. - **Component Stress:** Extreme cold can make materials like plastics and wiring insulation brittle, leading to physical damage.

Table 8: **Case of SteinerSQL Pipeline Execution in Spider2.0-Lite (Part 1): Query and Decomposition**

Component	Details and Values
Question	Between April 1 and July 31 of 2017, using the hits product revenue data along with the totals transactions to classify sessions as purchase (transactions ≥ 1 and productRevenue not null) or non-purchase (transactions null and productRevenue null), compare the average pageviews per visitor for each group by month.
Final Query	`SELECT FORMAT_DATE('%Y%m', PARSE_DATE('%Y%m%d', date)) AS month, CASE WHEN totals.transactions >= 1 AND hits.productRevenue IS NOT NULL THEN 'Purchase' WHEN totals.transactions IS NULL AND hits.productRevenue IS NULL THEN 'Non-Purchase' END AS session_type, SUM(totals.pageviews) / COUNT(DISTINCT fullVisitorId) AS avg_pageviews_per_visitor FROM bigquery-public-data.google_analytics_sample.ga_sessions_*; UNNEST(hits) AS hits WHERE _TABLE_SUFFIX BETWEEN '20170401' AND '20170731' AND ((totals.transactions >= 1 AND hits.productRevenue IS NOT NULL) OR (totals.transactions IS NULL AND hits.productRevenue IS NULL)) GROUP BY month, session_type ORDER BY month, session_type;`
Stage 1: Mathematical Decomposition	
1a. Mathematical	- **Aggregation:** Average pageviews per visitor (SUM(pageviews)/COUNT(DISTINCT fullVisitorId)). - **Grouping:** By month and by a new classification group. - **Temporal Filter:** Date between April 1, 2017 and July 31, 2017. - **Classification Logic (CASE WHEN):** - Group 1 ('Purchase'): `transactions >= 1` AND `productRevenue IS NOT NULL`. - Group 2 ('Non-Purchase'): `transactions IS NULL` AND `productRevenue IS NULL`.
1b. Computational	- The final average metric requires `pageviews`, and `fullVisitorId`. - The classification requires `transactions` and `productRevenue`. - The grouping requires the `date` (to extract the month) and the classification result. - The filtering requires the `date`.
1c. Terminal Table	Based on the required attributes, the essential conceptual tables (terminals) are identified. In the context of the GA schema, these are fields within the main `ga_sessions` table. - `ga_sessions` (contains `date`, `fullVisitorId`) - `totals` (contains `transactions`, `pageviews`) - `hits` (contains `productRevenue`) **Resulting Terminals $\mathcal{T}_{\text{req}}$:** {`ga_sessions`, `totals`, `hits`}

Table 9: **Case of SteinerSQL Pipeline Execution in Spider2.0-Lite (Part 2): Navigation and Validation**

Component	Details and Values
Stage 2: Schema Navigation	
2a. Schema Graph	- **Vertices V:** {ga_sessions, totals, hits} - **Edges E and Costs $C(e)$:** - Edges exist for foreign key (FK) relationships or high similarity. In the BigQuery schema, totals and hits are nested structures within ga_sessions, implying a direct, low-cost link. - C(ga_sessions, totals) $= 0.08$ (Direct FK-like link, cost is very low). - C(ga_sessions, hits) $= 0.09$ (Direct FK-like link, cost is very low). - C(totals, hits) $= 0.58$ (No direct link, higher cost based on dissimilarity).
2b. Steiner Tree	- **Problem:** Find the minimum cost subgraph connecting all terminals in $\mathcal{T}_{\text{req}}$. - **Terminals:** {ga_sessions, totals, hits} (all nodes in the graph). - **Solution:** The problem becomes finding the Minimum Spanning Tree (MST) of the graph. The MST algorithm selects the edges with the lowest costs that connect all vertices without forming a cycle. - Select edge (ga_sessions, totals) with cost 0.08. - Select edge (ga_sessions, hits) with cost 0.09. - **Reasoning Scaffold:** The resulting tree is **totals – ga_sessions – hits**. This scaffold indicates that ga_sessions is the central entity required to link information from totals and hits.
Stage 3: Multi-level Validation	
3a. Level 1:	- **Check:** Is the SQL syntactically correct and executable? - **Result:** Pass. The query uses valid BigQuery SQL syntax, including FORMAT_DATE, PARSE_DATE, UNNEST, and filtering on the _TABLE_SUFFIX pseudo-column.
3b. Level 2:	- **Check:** Does the query align with the user's semantic intent? - **Result:** Pass. - All terminal concepts (ga_sessions, totals, hits) are correctly utilized in the query. - The join logic (via UNNEST) correctly reflects the ga_sessions – hits path from the scaffold. - Attributes in SELECT, WHERE, and GROUP BY correctly map to the question's entities (month, classification, pageviews, etc.).
3c. Level 3:	- **Check:** Is the query's mathematical logic correct? - **Result:** Pass. - The AVG is correctly implemented as SUM(...) / COUNT(DISTINCT ...). - The CASE WHEN conditions for 'Purchase' and 'Non-Purchase' sessions exactly match the logic specified in the prompt. - The date range filter in the WHERE clause correctly implements "Between April 1 and July 31 of 2017". - The GROUP BY clause correctly includes both the month and the derived session type, as required to compare the groups.