ADVANCING THE TEXT-TO-SQL FRONTIER THROUGH GRAPH-LANGUAGE VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Translating natural language queries into Structured Query Language (SQL) in real world databases requires reconciling ambiguous user intent with complex relational schemas. While modern LLMs can generate candidate SQL queries with ease, identifying whether a query is functionally correct remains a fundamental challenge. Existing selection heuristics (e.g. execution-based self-consistency and LLM-as-a-judge evaluators) fail to handle the inherent syntax-semantics asymmetry of SQL. In other words, SQL has many ways to express equivalent queries, but small (single-token) variations can dramatically change a query's result.

In this work, we formulate SQL candidate verification as a *cross-modal alignment* problem between unstructured textual context (user intent, schema definitions, and domain evidence) and structured relational computation graphs. We introduce WARBLER, a unified framework coupling execution-guided candidate generation with a learned graph-language verifier which algorithmically compiles SQL into directed acyclic graphs (DAGs), normalizes syntactic variance, and encodes relational inductive biases via a graph-language architecture. On the highly competitive BIRD benchmark, WARBLER establishes a **new state of the art** on the official single model track with **80.04% execution accuracy** on the held-out test set, outperforming prior leading method by 2.90 points.

1 INTRODUCTION

Relational databases store the vast majority of structured business and scientific data. Translating natural language questions into Structured Query Language (SQL) enables users to query these databases directly without specialized programming knowledge (Yu et al., 2018; Li et al., 2023). While recent large language models (LLMs) have substantially advanced code and SQL generation, applying them reliably to real-world databases remains difficult.

Text-to-SQL faces three primary hurdles. First, real-world database schemas are large and interconnected, often containing dozens or hundreds of tables with complex foreign key constraints. Second, user questions are frequently underspecified, omitting implicit domain assumptions, required aggregations, or multi-table join paths. Third, code generation suffers from a fundamental verification asymmetry: generating candidate SQL queries is fast and inexpensive, but determining whether a generated query is semantically correct before running it is hard, e.g., a query can execute without errors and still return the wrong data due to an incorrect join condition or a reversed filter.

To improve reliability, modern Text-to-SQL pipelines generate multiple candidate queries and apply a selection strategy. However, prevailing selection methods have clear limitations. Execution-based self-consistency (Wang et al., 2023; Pourreza & Kubik, 2025) clusters candidates by their execution output and selects the majority result. While intuitive, this heuristic fails when ambiguous questions lead the model into shared failure modes, causing majority voting to reinforce an incorrect query. Alternatively, prompting an LLM to evaluate candidate SQL strings directly (Zheng et al., 2023) treats queries as flat text sequences. This makes the model overly sensitive to harmless syntactic variations (e.g. table aliases, predicate ordering, and join sequencing) while potentially overlooking critical single-token mistakes (e.g. misplaced column names or comparison operators).

In this work, we formulate SQL candidate verification as a *cross-modal alignment* problem. Natural language inputs (the user question, schema definitions, and domain evidence) describe the analytical

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

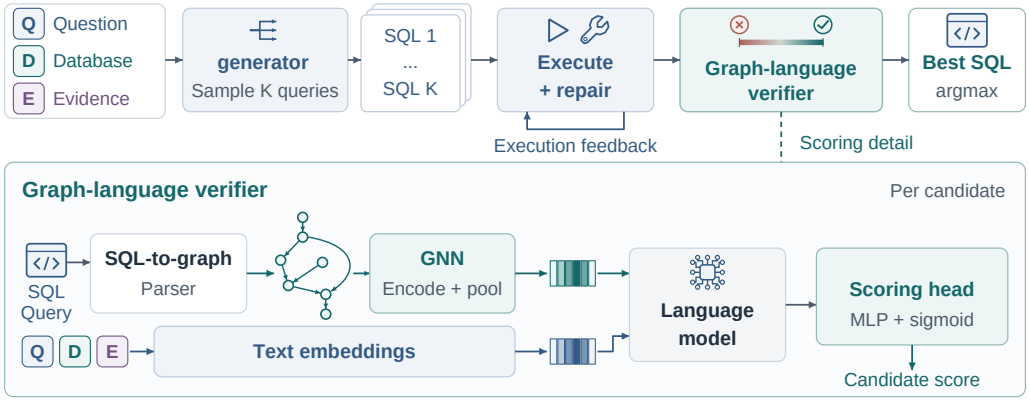


Figure 1: Overview of WARBLER. Top (Generation): Diverse candidate queries are sampled conditioned on question Q, database schema $\mathcal{D}$, and evidence E, followed by execution-guided repair against database engine feedback. Bottom (Verification): Candidate queries are compiled into typed relational DAGs and encoded via a GNN. Structural representations are fused with language model contextual embeddings to predict a calibrated semantic correctness score $S \in [0, 1]$ for optimal query selection.

intent, while candidate SQL queries represent structured relational computation graphs. Rather than forcing a language model to infer structural equivalence from sequential text tokens, we introduce a graph-language architecture that explicitly represents the relational query structure.

We present WARBLER, a Text-to-SQL framework built around two coordinated stages. In the first stage, a generator produces a diverse set of candidate queries conditioned on the database schema and question, then uses database execution feedback to automatically repair runtime errors. In the second stage, a learned graph-language verifier scores the surviving candidates. The verifier algorithmically parses each candidate query into a typed directed acyclic graph (DAG), normalizing surface syntactic differences. A graph neural network is adopted to encode this relational structure, and its representations are fused with the language model's contextual embeddings of the question, schema, and evidence to predict a semantic correctness score.

We evaluate WARBLER on the BIRD benchmark (Li et al., 2023). On the official held-out test set, WARBLER achieves **80.04% execution accuracy**, ranking first on the Single Model Track and outperforming the prior leading method by 2.90 percentage points. Ablation studies on development set show that execution-guided repair improves accuracy by 1.3%, while graph-language verification outperforms standard execution self-consistency by 1.1% and text-only LLM scoring by 0.8%.

The main contributions of this work are:

- We formulate Text-to-SQL candidate verification as a cross-modal alignment task between unstructured linguistic intent and structured relational query graphs.
- We propose a graph-language verifier that normalizes SQL syntax variations into typed computation graphs and aligns graph representations with language model embeddings.
- We achieve a **new state of the art** of 80.04% execution accuracy on the BIRD benchmark. *This evaluation was conducted by the BIRD team themselves.* In addition, we present ablations validating the efficacy of each design component.

2 PROBLEM FORMULATION

We formalize Text-to-SQL translation as a two-stage process: *candidate generation* followed by *cross-modal candidate verification*. Let the input context be denoted as a tuple $\mathcal{C} = (Q, \mathcal{D}, E)$, where Q is the user's natural language question, $\mathcal{D}$ encapsulates the target database schema, integrity constraints, and sampled column values, and E denotes optional external domain evidence such as

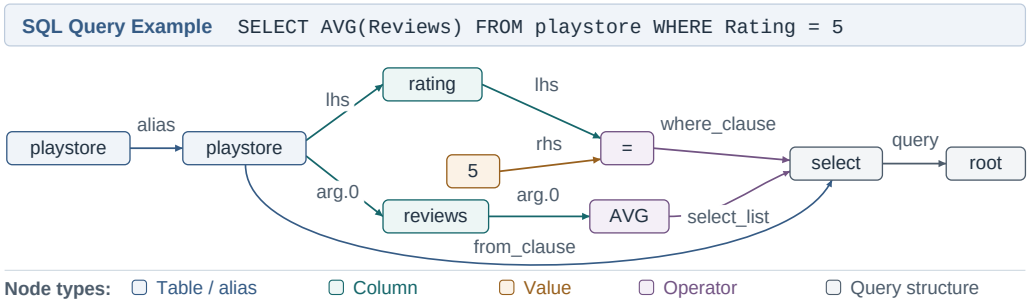


Figure 2: Algorithmic compilation of a SQL query into a typed relational DAG. Abstract Syntax Tree (AST) nodes are mapped into typed entity, relational operator, and literal vertices, preserving data flow and operator hierarchy while eliminating surface syntax formatting.

business rules and calculation formulas. The objective is to produce an executable SQL query Y^* that correctly answers Q when executed on $\mathcal{D}$.

Stage 1: Candidate Generation. Given context $\mathcal{C}$, the generator samples a candidate set $\mathcal{Y} = \{Y_1, \ldots, Y_K\}$ of K diverse SQL queries:

$$Y_k \sim P_{\text{gen}}(Y \mid \mathcal{C}), \quad k \in \{1, \ldots, K\}, \tag{1}$$

where P_{gen} is a generative distribution parameterized by an LLM. Generating multiple candidates addresses the underspecification of natural language questions by exploring alternative join paths, filtering criteria, and aggregation strategies.

Stage 2: Verification as Cross-Modal Alignment. Selecting the functionally correct query from $\mathcal{Y}$ is challenging because surface SQL syntax does not align monotonically with execution semantics: semantically identical queries can differ markedly in syntax (e.g., predicate ordering, join styles, and alias naming), while minute textual edits can drastically alter execution results. Furthermore, single-model verification faces an inherent self-confirmation barrier: an LLM struggles to detect errors in queries it just generated under the same textual representation, making an alternative structural modality essential to provide an unbiased verification perspective.

To overcome the syntax-semantics gap and this self-confirmation barrier, we formulate candidate selection as a cross-modal alignment problem between: (1) *textual context modality* $\mathcal{C} = (Q, \mathcal{D}, E)$, and (2) *relational graph modality* $G_Y = (\mathcal{V}, \mathcal{E})$, where each candidate query Y is algorithmically compiled into a canonical, typed directed acyclic graph (DAG) representing relational operations and data flow, with $\mathcal{V}$ denoting the set of nodes and $\mathcal{E}$ denoting the set of directed edges.

Compiling queries into graphs introduces structural inductive biases: logically equivalent queries $Y_a \equiv Y_b$ map to canonical graph structures $G_{Y_a} \simeq G_{Y_b}$[1], neutralizing superficial syntactic variations. The verifier learns a scoring function $S(G_Y, \mathcal{C}) \in [0, 1]$ measuring the semantic compatibility between the structured execution graph G_Y and intent $\mathcal{C}$. The optimal query Y^* is then selected via:

$$Y^* = \arg\max_{Y \in \mathcal{Y}} S(G_Y, \mathcal{C}). \tag{2}$$

3 Methodology

The architecture of Warbler couples an execution-guided candidate generator with a cross-modal graph-language verifier, as illustrated in Figure 1. Instead of relying on single-pass decoding, the framework first samples a diverse pool of executable candidate queries through execution feedback, and then evaluates their correctness using a graph-augmented LLM.

3.1 Candidate Generator

The candidate generator synthesizes a candidate pool $\mathcal{Y} = \{Y_1, \ldots, Y_K\}$ exploring alternative join strategies, filtering constraints, and aggregations according to user intent and database context.

[1] $Y_a \equiv Y_b$ denotes functional execution equivalence across any valid database instance, while $G_{Y_a} \simeq G_{Y_b}$ denotes typed directed graph isomorphism preserving operator semantics and data flow.



Figure 3: Illustration of relational invariance versus functional divergence under SQL rewrites. Blue: Semantically equivalent variations (e.g., predicate commutativity, symmetric comparisons, and join syntax) compile into isomorphic computation graphs. Amber: Semantically divergent perturbations (e.g., altered literals or projection ordering) yield topologically distinct graphs.

Context Encoding and Sampling. Effective candidate generation depends on grounding the model in the given database schema. Prior to generation, we conduct database profiling by extracting Data Definition Language (DDL) declarations directly from the database engine to capture table schemas, column types, and foreign key relationships. Then, we sample cell values and compute summary statistics (e.g., uniqueness, ranges, and null ratios) for individual columns. This empirical grounding enables the model to resolve domain-specific entities and literal values without requiring brittle pre-linking heuristics. Conditioned on the question Q, profiled schema $\mathcal{D}$, and evidence E, we sample K candidate queries using non-zero temperature and deterministic column-order shuffling within the prompt to encourage structural diversity and mitigate schema positional biases.

Execution-Guided Triage and Repair. In order to verify basic correctness of the generated SQL queries, they are immediately executed against the database engine. The candidates are probed against three failure modes: runtime compilation errors, anomalous execution results (such as unexpected empty tables or excessive null outputs indicating misconfigured join conditions), and output type deviations from the question's expected schema. Faulty candidates are routed to a deterministic (zero temperature) repair step. This targeted repair prompt supplies the model with the original context, the flawed query, and the exact runtime execution feedback, allowing the model to repair syntax and logical misalignments before advancing candidates to the verification stage.

3.2 Cross-Modal Graph-Language Verifier

The verifier scores each candidate query $Y \in \mathcal{Y}$ against the natural language context $\mathcal{C} = (Q, \mathcal{D}, E)$ by projecting SQL logic into a structural graph modality.

Algorithmic SQL-to-DAG Parsing. To achieve syntax-invariant representation, each candidate query Y is parsed into an Abstract Syntax Tree (AST) using `sqlglot` and compiled into a canonical, typed Directed Acyclic Graph (DAG) $G_Y = (\mathcal{V}, \mathcal{E})$. As illustrated in Figure 2, nodes $u \in \mathcal{V}$ represent database entities (tables, columns), relational operators (e.g., `JOIN`, `WHERE`, `GROUP BY`), aggregations, and literal constants across distinct semantic types. Labeled directed edges $e = (u, v) \in \mathcal{E}$ capture data flow and functional dependencies, each associated with an edge relation type $\tau(u, v)$. During graph compilation, the parser applies three normalization rules:

1. *Hierarchical Edge Resolution:* Column nodes are explicitly bound to their source table nodes via schema cross-referencing, resolving implicit column projections and arbitrary join sequence permutations.
2. *Alias and Identifier Standardization:* Arbitrary table aliases are normalized and linked to canonical relation nodes, rendering graph topology invariant to aliases.
3. *Node Uniqueness:* Repeated invocations of the same relation or attribute within the same scope are collapsed into unique nodes.

Relational Graph Encoding. For each node $u \in \mathcal{V}$, an initial feature vector $x_u \in \mathbb{R}^{d_{\text{in}}}$ encoding categorical node type and lexical identity is mapped to an initial representation $h_u^{(0)} = \text{Linear}(x_u) \in \mathbb{R}^{d_g}$ via a trainable linear layer. To capture relational computation dependencies across the DAG,

node representations are updated through an L-layer Graph Isomorphism Network (GIN) (Xu et al., 2019), where $L = 2$. For each layer $l \in \{1, \ldots, L\}$, the representation of node u is updated via:

$$h_u^{(l)} = \text{MLP}^{(l)}\left((1 + \epsilon^{(l)})h_u^{(l-1)} + \sum_{v \in \mathcal{N}(u)} \text{ReLU}\left(h_v^{(l-1)} + b_{\tau(v,u)}\right)\right), \tag{3}$$

where $\mathcal{N}(u) = \{v \in \mathcal{V} \mid (v, u) \in \mathcal{E}\}$ denotes the incoming neighborhood of node u, $b_{\tau(v,u)} \in \mathbb{R}^{d_g}$ is a learnable embedding vector corresponding to edge type $\tau(v, u)$, $\epsilon^{(l)} \in \mathbb{R}$ is a learnable scalar parameter, and $\text{MLP}^{(l)}$ denotes a multi-layer perceptron. We then apply sum-pooling to get the final graph representation $h_G \in \mathbb{R}^{d_g}$:

$$h_G = \sum_{u \in \mathcal{V}} h_u^{(L)}. \tag{4}$$

We illustrate the intuition for how this model is applied to SQL queries in Figure 3. Two SQL queries can be semantically equivalent while exhibit surface-level differences in text, for example, by reordering `WHERE` filters, swapping the two sides of a comparison, or using different join syntax. Standard LLMs process text sequentially, making them sensitive to these surface changes. Our graph representation is invariant to such perturbations. During graph construction, the parser standardizes equivalent syntax into matching graph structures. In addition, the sum aggregations in the GNN layers (Equation 3) and graph pooling (Equation 4) naturally ignore the order of independent operations. As a result, queries with identical execution logic produce isomorphic graphs ($G_{Y_a} \simeq G_{Y_b}$) and identical representations ($h_G(Y_a) = h_G(Y_b)$), allowing the verifier to score queries based on their actual logic rather than how they are written.

Cross-Modal Fusion and Candidate Scoring. Concurrently, the unstructured context $\mathcal{C} = (Q, \mathcal{D}, E)$ is encoded by the LLM backbone. Mean-pooling the output hidden states of the context tokens from the final layer yields a contextual text representation $h_T \in \mathbb{R}^{d_t}$. Inspired by Perozzi et al. (2024), which demonstrated that directly encoding graph topology into continuous representations for a frozen LLM avoids the serialization overhead and ordering biases of converting graphs to text, we treat the pooled graph embedding h_G as a continuous structural representation and project it directly into the language model's latent space rather than serializing execution graphs into text.

Specifically, we project h_G and h_T into a shared d_m-dimensional space via linear projections $W_g \in \mathbb{R}^{d_m \times d_g}$ and $W_t \in \mathbb{R}^{d_m \times d_t}$, fusing them via concatenation and layer normalization:

$$h_{\text{fused}} = \text{LayerNorm}\left([W_g h_G \parallel W_t h_T]\right) \in \mathbb{R}^{2d_m}. \tag{5}$$

The fused vector h_{fused} is passed to a classification head (a two-layer MLP with a sigmoid activation σ) to compute the alignment score:

$$S(G_Y, \mathcal{C}) = \sigma\left(\text{MLP}(h_{\text{fused}})\right). \tag{6}$$

During training, the base LLM remains frozen to preserve its general semantic knowledge and minimize training cost; only the initial linear projection, GIN layers, projection matrices, and classification head are trained with the binary cross-entropy loss against execution correctness labels:

$$\mathcal{L} = -\left[y \log S(G_Y, \mathcal{C}) + (1 - y) \log(1 - S(G_Y, \mathcal{C}))\right], \tag{7}$$

where $y \in \{0, 1\}$ indicates whether candidate Y executed on $\mathcal{D}$ exactly matches the ground-truth execution result. At inference time, the candidate maximizing $S(G_Y, \mathcal{C})$ is chosen as Y^*.

4 Experiments

We evaluate Warbler empirically to answer three primary questions: (1) How does the proposed model perform on Text-to-SQL compared to existing models? (2) How much do the individual components of the proposed model contribute to overall performance? (3) Does the joint embedding space reliably capture relational semantic equivalence while isolating logical failure modes?

4.1 Experimental Setup

Benchmark. We conduct our primary evaluation on the BIRD benchmark (Li et al., 2023), the leading public benchmark for cross-domain, database-grounded Text-to-SQL. BIRD comprises 12,751

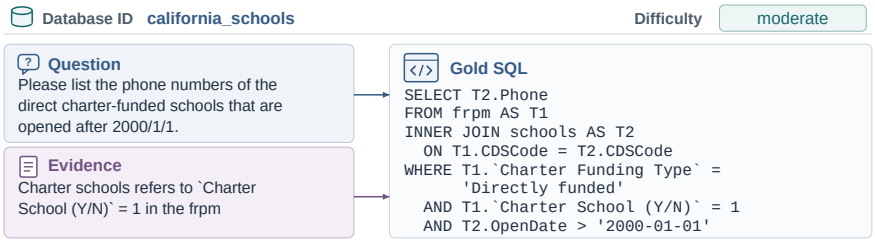


Figure 4: A representative example from the BIRD benchmark, illustrating the relational complexity of real-world Text-to-SQL. Correct query synthesis requires joint reasoning across underspecified user questions, multi-table schemas with foreign key constraints, and domain-specific calculation rules.

natural language question-SQL pairs spanning 95 real-world databases across 37 domains (e.g., healthcare, financial registries, education), totaling over 33.4 GB of structured data. As illustrated in Figure 4, queries in BIRD reflect enterprise conditions: they feature massive multi-table schemas with implicit foreign keys, underspecified user questions, and dirty cell values requiring domain-specific reasoning and external evidence.

The benchmark is partitioned into a training split (9,428 examples), a development split (1,534 examples), and a held-out test split evaluated strictly by the benchmark organizers.

Evaluation Metric. Performance is evaluated using Execution Accuracy (EX), defined as the proportion of test (resp. development) examples where executing the predicted query $\hat{Y}$ on the target database instance $\mathcal{D}$ yields the exact same result set as the ground-truth query Y^*:

$$\text{EX} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\hat{Y}_{i\mathcal{D}_i} = Y_i^{*}{}_{\mathcal{D}_i}\right). \tag{8}$$

Because EX validates runtime execution outputs rather than surface string equality, it accommodates valid alternative SQL implementations.

Implementation Details. For candidate generation, we employ Gemini 3.1 Pro as the base LLM. Schema profiling extracts column types, primary/foreign keys, and five representative values per column. We set the generation temperature to 0.7 and apply deterministic column-order shuffling to sample a diverse pool of $K = 7$ candidates per question. Execution triage executes candidates on an SQLite engine; candidates encountering runtime exceptions or anomalous empty returns are repaired using a single-pass prompt at temperature 0.

For the verifier, candidate queries are compiled via `sqlglot` into typed DAGs. The GNN consists of $L = 2$ Graph Isomorphism Network layers with hidden dimension $d_g = 128$. Node types and edge relations are embedded with dimension 64. The textual context $(Q, \mathcal{D}, E)$ is encoded by Gemini 3.1 Pro with frozen parameters. Linear adapters map graph and text embeddings into a shared dimension $d_m = 256$, and the classification head is trained with AdamW (learning rate 10^{-4}, batch size 32) using binary cross-entropy on candidate queries generated from the BIRD training set.

4.2 Main Results

Table 1 reports execution accuracy on the BIRD development and held-out test sets across the official Single Model Track. WARBLER achieves **80.04% execution accuracy** on the held-out test set, securing 1st place on the official BIRD Single Model Track. This marks the first single-model system to cross the 80% milestone on BIRD. On the development set, WARBLER attains **74.12% execution accuracy**, similarly leading all competing single-model architectures.

Progress on the BIRD benchmark has historically been characterized by tight clustering among top models. As observed in Table 1, ranks 2 through 9—encompassing both proprietary foundation models and specialized open-source models fine-tuned for Text-to-SQL such as Q-SQL, SiriusAI-SQL, and Spektr-SQL—are tightly compressed within a narrow 3.69% band (from 73.45% to 77.14%), with performance differences between adjacent ranks routinely below 0.5%. In such a highly saturated benchmark where even a 1.0% gain represents a substantial leap, WARBLER outperforms the previous state-of-the-art (Gemini-SQL at 77.14%) by **+2.90 percentage points**, and exceeds prominent base reasoning models such as Claude Opus 4.6 (70.15%) and Qwen3-Coder-480B (68.14%) by more than 9 to 11 points.

The performance advantage of WARBLER is most pronounced on queries categorized by BIRD as *Moderate* and *Challenging*. These instances involve multi-way table joins, complex nested subqueries, and non-trivial aggregation logic. In such complex queries, generative LLMs produce wide syntactic variability across candidate queries (e.g., varying join trees, different subquery nesting depths, and commutative predicate orderings). While text-only verifiers and voting heuristics struggle to discern true semantic alignment amid this surface noise, our graph-language verifier normalizes these structural variation, reliably identifying the functionally correct query.

4.3 ABLATION STUDIES

To understand the contribution of each component in WARBLER, we conduct systematic ablation experiments on the BIRD development set, with the results summarized in Table 2.

Dissecting the Candidate Generator. We first evaluate the impact of the generative backbone and triage mechanisms:

- *Foundation Model Capacity:* Replacing Gemini 3.1 Pro with Gemini 2.5 Pro reduces accuracy on the development set by 2.4% (from 74.12% to 71.72%), while using smaller backbones such as Gemma-4-it-31b or Gemini 3.5 Flash results in drops of 1.4% and 1.9%, respectively. This highlights that a high-capacity base generator is crucial for generating high-quality candidate pools.
- *Execution-Guided Repair:* Disabling the execution repair triage degrades performance by 1.3%. Eliminating runtime compilation errors and empty table states before scoring ensures that the verifier evaluates viable candidates.
- *Column Meaning Context:* Omitting column descriptions from the generator's prompt context leads to a negligible decline of only 0.1% (from 74.12% to 74.02%). This indicates that concrete database profiling—specifically exact DDL declarations, foreign key constraints, and sampled column values—provides sufficient empirical grounding, making generation robust even without extensive annotations of column meanings.

Validating the Graph-Language Verifier. We next analyze the specific design choices of our verifier architecture:

- *Comparison with Execution Self-Consistency:* Replacing our learned verifier with standard execution self-consistency (majority voting over execution output sets) causes a 1.1% drop in EX. When database queries are complex or schemas contain ambiguous foreign key paths, LLMs frequently exhibit correlated failure modes, producing identical, logically flawed queries that evaluate to matching empty or degenerated tables. Majority voting mistakenly selects these erroneous consensus outputs, whereas our verifier evaluates the query's structural graph against question intent $\mathcal{C}$ to identify the correct query.
- *Comparison with Text-Only LLM Scoring:* Removing the SQL-to-DAG parser and GNN, instead prompting the language model to score raw SQL text strings directly, results in a 0.8% drop. This degradation demonstrates that text-only judges remain susceptible to superficial syntactic differences (e.g., alias names and predicate order) that our graph-language verifier neutralizes.
- *Cross-Modal Fusion:* Restricting the GNN hidden dimension from 128 to 32 leads to a 0.5% drop, showing that encoding complex relational dependencies requires sufficient parameter capacity. We also investigate the impact of different fusion mechanisms between graph and language representations. In our default *late-fusion* architecture (Equation 5), the graph embedding h_G and pooled textual embedding h_T are projected and concatenated externally before the MLP scoring head, keeping the base LLM entirely frozen. In contrast, an *early-fusion* approach injects intermediate GNN node representations directly into the language model's transformer layers via cross-modal attention adapters. Evaluating on a compact backbone (Gemma3-4b), early fusion improves accuracy by 0.3% over late fusion (reducing the drop from -0.5% to -0.2%), demonstrating the benefit of fine-grained token-level cross-attention. However, late fusion with Gemini 3.1 Pro provides superior overall performance while reducing the training overhead and architectural modification of early-fusion adapters.

4.4 REPRESENTATION QUALITY AND ERROR ANALYSIS

Latent Space Separability. To evaluate whether the cross-modal verifier learns genuine execution semantics rather than surface text regularities, we extract the fused representations h_{fused} for all

Table 1: Execution accuracy (EX, %) on the development (dev) and held-out test sets from the BIRD benchmark across the official Single Model Track. Best results are in **bold**; second-best results are underlined. Δ_{Ours} denotes the test score difference relative to WARBLER. "–" indicates undisclosed model parameter size.

Model	Size	Execution Accuracy (EX, %)		
		Dev	Test‡	Δ_{Ours}
WARBLER (ours)	–	**74.12**	**80.04**	–
Gemini-SQL (Pourreza & Kubik, 2025)	–	73.27	77.14	−2.90
Q-SQL (Shankar et al., 2025b)	30B-3B-MoE	72.99	76.47	−3.57
SiriusAI-SQL (Luo et al., 2026)	27B	73.70	75.85	−4.19
Spektr-SQL (Shankar et al., 2025a)	30B-3B-MoE	72.10	74.85	−5.19
Sophon-Text2SQL-32B	32B	72.43	74.79	−5.25
Arctic-Text2SQL-R1-32B (Yao et al., 2025)	32B	72.20	73.84	−6.20
Databricks RLVR 32B	32B	70.80	73.56	−6.48
Jiayin-Pangu-Text2SQL-14B	14B	71.10	73.45	−6.59
Claude Opus 4.6† (Anthropic, 2026)	–	68.77	70.15	−9.89
Qwen3-Coder-480B-A35B† (Alibaba, 2024)	480B	66.17	68.14	−11.90
Claude 4.5 Sonnet† (Anthropic, 2025)	–	67.34	66.85	−13.19
GLM-4.7† (Z.AI, 2025)	–	63.82	62.94	−17.10
DeepSeek-R1† (DeepSeek-AI et al., 2025)	–	61.67	60.93	−19.11

†Submissions evaluated on BIRD Dev 20251106.
‡The test set is strictly held-out and evaluated by the BIRD benchmark organizers.

$1,534 \times 7$ candidate queries on the development split and project them into two dimensions using t-SNE, as shown in Figure 5. The visualization exhibits clear topological clustering governed by functional correctness. Candidate queries belonging to the same execution equivalence class (i.e., queries that yield identical, correct result tables) map into dense, cohesive clusters. This holds true even when candidates exhibit divergent syntactic structures, such as explicit ANSI `JOIN` clauses versus implicit `WHERE` joins, or permuted boolean conjunctions. This confirms that the GNN aggregation and normalization rules effectively enforce permutation and structural invariance. Conversely, candidate queries with subtle semantic errors—such as referencing an incorrect foreign key column, selecting an inappropriate aggregation function, or flipping an inequality operator—are projected into distant latent neighborhoods, despite sharing nearly identical surface tokens with valid queries. This indicates that the verifier does not merely evaluate lexical similarity against the prompt, but actively captures relational computation semantics.

Diagnostic Failure Mode Resolution. We further conduct an empirical error analysis on development instances where WARBLER successfully selected the correct query while competitive baselines failed. On multi-table queries requiring three or more joins, candidate generators often succumb to shared schema pitfalls—such as selecting an intuitive but incorrect join path—causing 4 out of 7 sampled candidates to produce an identical empty table. Because majority voting is blind to query logic, execution self-consistency uniformly selects this empty result. In contrast, WARBLER evaluates the candidate computation DAGs against question intent $\mathcal{C}$, identifies that the majority candidates violate foreign key logic, and assigns the highest score to a minority candidate that correctly traversed the schema, returning valid data. Moreover, for instances where two candidate queries are semantically equivalent but one serializes predicates in the exact sequential order mentioned in the natural language question, text-only LLM judges display a pronounced prompt-order bias, systematically favoring the mirrored query. Because our graph parser canonicalizes commutative operations and GIN aggregation is permutation-invariant, WARBLER produces identical alignment scores ($S(G_{Y_a}, \mathcal{C}) = S(G_{Y_b}, \mathcal{C})$), proving immune to sequential presentation order.

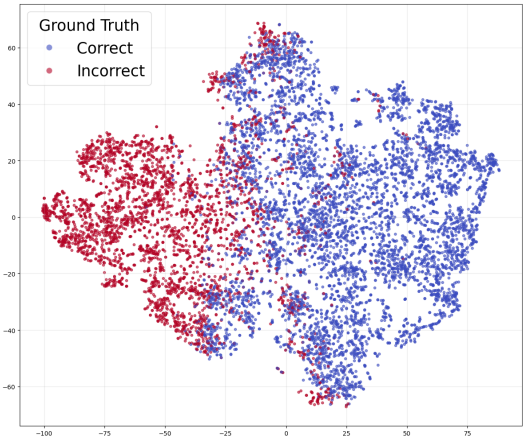
5 RELATED WORKS

Text-to-SQL with Large Language Models. The application of Large Language Models to Text-to-SQL has largely replaced specialized semantic parsing architectures with in-context learning and instruction tuning (Wang et al., 2022; Pourreza & Rafiei, 2023; Gao et al., 2024). Early work on benchmarks such as Spider (Yu et al., 2018) focused primarily on prompt design and few-shot example selection. However, the introduction of the BIRD benchmark (Li et al., 2023) highlighted the

Table 2: Ablation analysis on the BIRD development set evaluating individual design choices across candidate generation (LLM backbone, execution repair, and generator context) and cross-modal verification (scoring strategies, graph representations, and cross-modal fusion).

Changes from default model	Δ EX
Generator	
Backbone: Gemini 2.5 Pro	−2.4
Backbone: Gemma-4-it-31b	−1.9
Backbone: Gemini 3.1 Flash	−1.5
Backbone: Gemini 3.5 Flash	−1.4
No execution-based self-correction	−1.3
No `column_meaning` evidence	−0.1
Verifier	
Self-consistency selection	−1.1
Text only (no graph parser & GNN)	−0.8
Backbone: Gemma3-270b	−0.6
GNN hidden dimension: 32	−0.5
Backbone: Gemma3-4b (late fusion)	−0.5
Backbone: Gemma3-4b (early fusion)	−0.2

Figure 5: t-SNE visualization of fused candidate representations h_{fused} on the BIRD development set. Semantically valid queries cluster tightly regardless of surface syntactic rewrites, while candidates with execution-altering logical errors are cleanly separated in latent space.



challenges of enterprise databases, where massive schemas, noisy data values, and underspecified queries cause standard single-pass generation to fail. This has motivated pipelines that combine candidate sampling with targeted post-generation selection.

Candidate Selection and Code Verification. Sampling multiple candidates and selecting the most reliable output is an established technique across code synthesis (Chen et al., 2023; Ridnik et al., 2024). In Text-to-SQL, the dominant selection heuristic has been execution-based self-consistency (Wang et al., 2023; Pourreza & Kubik, 2025), which selects the candidate belonging to the largest execution equivalence set. While effective for simple queries, execution self-consistency suffers from correlated errors when multiple candidates hit identical schema pitfalls. An alternative approach prompts an LLM as a direct discriminative judge (Zheng et al., 2023). However, because standard LLMs evaluate queries as sequential token strings, they remain sensitive to surface-level syntactic variations. WARBLER addresses both limitations by evaluating candidate queries through an invariant structural representation rather than relying solely on execution frequency or raw text.

Graph Representations and Graph-Language Alignment. Graph Neural Networks (Chami et al., 2022) have previously been applied in Text-to-SQL primarily for schema linking, encoding database schema graphs to capture foreign key topologies (Bogin et al., 2019; Hui et al., 2021). However, using GNNs to represent the generated SQL queries themselves has remained largely unexplored. Our approach draws inspiration from Perozzi et al. (2024), which showed that projecting GNN embeddings directly into a language model's latent space allows frozen LLMs to reason over structured data without serialization loss. We extend this principle to relational code verification: by translating SQL into canonical computation DAGs, encoding them with a Graph Isomorphism Network (Xu et al., 2019), and aligning the structural vectors with language representations, we enable invariant semantic verification.

6 CONCLUSION

We presented WARBLER, a Text-to-SQL framework that formulates candidate verification as a cross-modal alignment problem between linguistic intent and relational computation graphs. By combining execution-guided candidate repair with a graph-language verifier, WARBLER addresses the syntax-semantics asymmetry inherent in flat SQL text representations. The verifier compiles candidate queries into canonical DAGs, uses a Graph Isomorphism Network to eliminate sensitivity to surface syntactic variations, and aligns structural representations with a foundation language model backbone. On the BIRD benchmark, WARBLER achieves a state-of-the-art 80.04% execution accuracy on the held-out test set, outperforming all existing single-model systems. Our ablations confirm that grounding language models with explicit graph inductive biases provides a principled, effective foundation for reliable database interaction.

REFERENCES

Alibaba. Qwen3-coder-480b-a35b-instruct. Hugging Face Model Card, 2024. URL https://huggingface.co/Qwen/Qwen3-Coder-480B-A35B-Instruct.

Anthropic. Introducing claude sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5, September 2025.

Anthropic. Introducing claude opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026.

Ben Bogin, Jonathan Berant, and Matt Gardner. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *International Conference on Learning Representations (ICLR)*, 2023.

DeepSeek-AI et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(11):3332–3345, 2024.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, et al. Dynamic hybrid relation exploration network for cross-domain context-dependent semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Jinyang Li, Binyuan Hui, GE QU, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Leo Luo, Haining Xie, Siqi Shen, Zhipeng Ma, Rui Ling, Hang Xu, Hefeng Jiang, Dingwei Chen, Yang Li, Peng Chen, et al. SIRIUS-SQL: Anchoring multi-candidate Text-to-SQL in execution feedback. *arXiv preprint arXiv:2606.01246*, 2026. URL https://arxiv.org/abs/2606.01246.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for LLMs. *arXiv preprint arXiv:2402.05862*, 2024.

Mohammadreza Pourreza and Tom Kubik. A new top score: Advancing Text-to-SQL on the BIRD benchmark. Google Cloud Blog, November 2025. URL https://cloud.google.com/blog/products/databases/how-to-get-gemini-to-deeply-understand-your-database.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 36332–36344, 2023.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with alphacodium: From prompt engineering to flow engineering. *arXiv preprint arXiv:2401.08500*, 2024.

Ravi Shankar, Kishore Kumar, Scott Roberts, and Desik Rengarajan. BIRD-submission SQL (SpektrBot Ads). Amazon Ads, October 2025a. URL https://app.notion.com/p/BIRD-Submission-SQL-SpektrBot-Ads-291e8bf3051480afa7b8cfac3aa26b10. Updated October 30, 2025.

Ravi Shankar, Leah Riley, and Adi Kalyanpur. Q-SQL: AWS, memory with RL for structured data. Technical report, AWS Quick Team, December 2025b. URL https://peridot-flood-234.notion.site/Q-SQL-AWS-2bfe8bf30514801ba112f92ff20784e3. BIRD Benchmark Submission.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Zhiruo Wang, Shuyan Zhou, Daniel Fried, and Graham Neubig. Execution-based evaluation for open-domain code generation. *arXiv preprint arXiv:2212.10481*, 2022.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *CoRR*, abs/2505.20315, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3911–3921, 2018.

Z.AI. Glm-4.7 released: Foundation model with improved coding, reasoning, and agentic capabilities. https://docs.z.ai/release-notes/new-released, December 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.