SIPC-SQL: Enhancing Zero-Shot Text-to-SQL via Structure-Aware Intent Parsing and Complexity-Adaptive Routing

Qinghan Wang[1,2] Chuantao Li[1,2] Jintao Li[1,2]
Yang Zhang[1,2] Jianglin Ma[1,2] Wenhui Yang[1,2] Zhigang Zhao[1,2⋆]
[1] Key Laboratory of Computing Power Network and Information Security, Ministry of Education,
Shandong Computer Science Center (National Supercomputer Center in Jinan),
Qilu University of Technology (Shandong Academy of Sciences), Jinan, China
[2] Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing,
Shandong Fundamental Research Center for Computer Science, Jinan, China
⋆Corresponding Author: Zhigang Zhao, Email: zhaozhg@sdas.org

***Abstract*—Zero-shot Text-to-SQL methods often struggle with structural hallucinations in complex schemas and incur unnecessary inference cost due to over-reasoning on simple queries. To address these challenges, we propose SIPC-SQL, a structured inference framework that improves both reliability and efficiency without relying on fine-tuning or in-context demonstrations. SIPC-SQL decouples intent understanding from SQL generation through structure-aware intent parsing, producing a schema-grounded intermediate representation that explicitly encodes target entities, logical operations, and join paths. To ensure structural validity, we further employ schema-strict constrained decoding to eliminate hallucinated schema elements. In addition, we introduce a complexity-adaptive routing mechanism that dynamically selects between a lightweight direct generation path and a structure-enhanced reasoning path based on query complexity. Experiments on the Spider and BIRD benchmarks demonstrate that SIPC-SQL achieves state-of-the-art performance among zero-shot methods, reaching 84.8% execution accuracy on the Spider test set, while reducing token consumption by approximately 40% with negligible performance loss. Cross-backbone evaluations further show that SIPC-SQL consistently improves performance across different model scales, enabling smaller models to outperform larger zero-shot baselines. These results highlight the effectiveness of structured intent grounding and adaptive inference control for building reliable and cost-efficient Text-to-SQL systems.**

***Index Terms*—Text-to-SQL, Large Language Models, Semantic Parsing, Adaptive Inference, Zero-Shot Learning**

I. Introduction

Text-to-SQL, the task of translating natural language questions into executable SQL queries [1]–[3], plays a fundamental role in enabling natural language interfaces to structured databases. It has broad applications in data analytics, business intelligence, and interactive database systems. Recent advances in large language models (LLMs) have substantially improved Text-to-SQL performance, particularly under supervised [5], [6] or in-context learning [7], [8] settings. However, ensuring reliable and efficient SQL generation in zero-shot scenarios [9], [10] remains a significant challenge, especially when facing complex database schemas and diverse user intents.

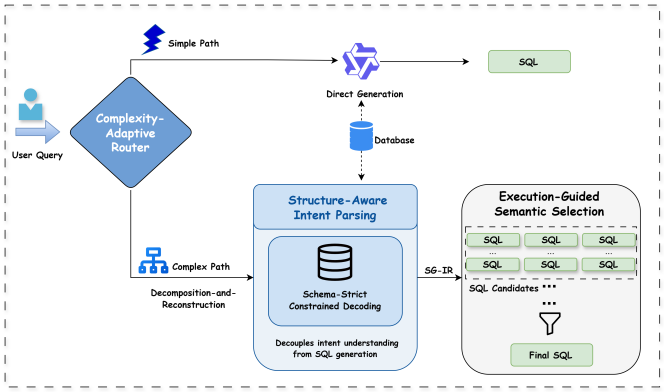

Fig. 1. SIPC-SQL overall architecture. The framework decouples intent understanding from SQL generation through a Schema-Grounded Intermediate Representation (SG-IR). Complexity-adaptive routing directs queries to appropriate generation paths, and execution-guided semantic selection ensures hallucination-free final output.

A key difficulty in zero-shot Text-to-SQL lies in the mismatch between unstructured natural language intent and rigid database schemas. Without explicit supervision or demonstrations, LLMs frequently generate SQL queries that reference non-existent tables or columns, select incorrect join paths, or exhibit logical inconsistencies. Such schema hallucinations become more severe as schema size and structural complexity increase. At the same time, many existing approaches attempt to mitigate these issues by relying on large models, extensive reasoning prompts, or execution-guided decoding, which often incur substantial computational cost even for structurally simple queries. These observations highlight two intertwined challenges for zero-shot Text-to-SQL systems: (1) improving structural reliability under complex schemas, and (2) reducing unnecessary inference cost while preserving accuracy.

To this end, we propose SIPC-SQL, a structured inference framework that explicitly targets both reliability and efficiency in zero-shot Text-to-SQL. We transform the noisy

user query into a deterministic Schema-Grounded Intermediate Representation (SG-IR), which explicitly identifies target entities, reasoning logic, and filtering constraints. To ensure the integrity of this representation, we employ Schema-Strict Constrained Decoding, utilizing a prefix tree to enforce hard constraints on schema identifiers during generation, thereby eliminating hallucinations of non-existent schema elements. To balance accuracy and efficiency, we implement a complexity-aware Adaptive Router. Simple queries follow a high-speed Direct Generation path, while complex queries are processed through a Decomposition-and-Reconstruction path. For complex cases, we further utilize Execution-Guided Semantic Selection, which executes multiple candidates and selects the optimal query based on execution feedback and semantic consistency.

We evaluate SIPC-SQL on the Spider and BIRD benchmarks under strictly zero-shot settings. Experimental results demonstrate that SIPC-SQL achieves competitive execution accuracy compared to strong zero-shot and few-shot baselines, while reducing inference cost by approximately 40% through the proposed adaptive routing mechanism. Ablation studies and diagnostic analyses further show that structured intent grounding, schema-aware constrained decoding, and adaptive inference jointly contribute to improved reliability and efficiency across different model backbones.

II. METHODOLOGY

We propose SIPC-SQL, a structured inference framework that addresses the stochastic nature of LLMs through three stages, as illustrated in Figure 1: Structure-Aware Intent Parsing, Complexity-Adaptive Routing, and Execution-Guided Semantic Selection.

A. Problem Formulation

The Text-to-SQL task aims to map a natural language question Q and a database schema S into an executable SQL query Y. The objective is to maximize the probability $P(Y|Q,S)$ such that the execution result $E(Y,D)$ on database content D is correct.

B. Structure-Aware Intent Parsing

Directly mapping complex questions to SQL often suffers from schema linking [11], [12] failures and logical hallucinations. To mitigate this, we propose a Structure-Aware Intent Parsing module that transforms the noisy query Q into a deterministic, Schema-Grounded Intermediate Representation (SG-IR).

1) Schema-Grounded Intermediate Representation: The Schema-Grounded Intermediate Representation (SG-IR) explicitly captures the structural intent of the user query while remaining independent of executable SQL syntax.

Formally, given a natural language query Q and database schema S, we prompt the language model to generate a structured Schema-Grounded Intermediate Representation (SG-IR) in a predefined JSON format. The SG-IR is defined as:

$$\hat{Q} = \{E_{target}, L_{logic}, A_{agg}, F_{filter}, J_{paths}\} \tag{1}$$

The prompt specifies the required fields and their semantic roles, while the database schema is provided as context to support schema grounding.

where E_{target} denotes the set of schema-grounded target entities (tables and columns), L_{logic} characterizes the high-level query structure (filtering, ordering, grouping, or nested queries), A_{agg} specifies aggregation operations bound to schema attributes, F_{filter} represents literal filtering constraints extracted from the query, and J_{paths} explicitly specifies join paths derived from foreign-key relations in S.

The SG-IR serves as a deterministic structural blueprint for SQL reconstruction. Schema-bound components (E_{target}, A_{agg}, and J_{paths}) provide explicit grounding to the database schema, while L_{logic} functions as a coarse-grained control signal guiding query structure. Literal values in F_{filter} preserve semantic flexibility without introducing schema identifiers. This design enables strict schema validation during intent parsing while maintaining robustness to linguistic variation.

2) Schema-Strict Constrained Decoding: To ensure that the generated SG-IR strictly adheres to the database schema while preserving the model's ability to extract open-ended constraint values, we introduce a Schema-Strict Constrained Decoding strategy during intent parsing.

Given the database schema S, we construct a schema-specific prefix tree that encodes all valid schema identifiers, including table names, column names, and foreign-key-based join paths. All identifiers are normalized to lowercase to avoid case-sensitivity issues. During decoding, the valid token set is dynamically determined by the current SG-IR field being generated.

For schema-bound components (E_{target}, A_{agg}, and J_{paths}), the decoding space is strictly constrained to tokens reachable in the prefix tree. The probability of generating the next token x_t is rectified as:

$$P(x_t|x_{<t}) = \begin{cases} \frac{1}{Z_t} \cdot P_{\text{LLM}}(x_t|x_{<t}) & \text{if } x_t \in \mathcal{V}_{\text{valid}}^{(t)} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\mathcal{V}_{\text{valid}}^{(t)}$ denotes the set of schema-valid tokens at step t, and Z_t is a normalization constant.

To support aliasing, once a schema alias is generated, it is dynamically inserted into the prefix tree and added to $\mathcal{V}_{\text{valid}}^{(t)}$, enabling subsequent references without violating schema constraints.

For value-related components in F_{filter}, the constraints are selectively relaxed to allow open-vocabulary generation [18], following the hybrid constrained decoding paradigm. This design ensures a rigid structural backbone while maintaining flexibility for literal value extraction. For the logic component L_{logic}, we do not apply schema-level constraints during decoding. This component characterizes the required query structure at a coarse granularity (whether the query involves filtering, ordering, grouping, or nested subqueries), rather than encoding executable logic or schema-specific operations. Since L_{logic} does not introduce schema identifiers, it is generated in an unconstrained manner and used solely as a high-level control

Algorithm 1: Schema-Strict Constrained Decoding for SG-IR Generation

Input : Pre-trained LLM $\mathcal{M}$, database schema S, prompt P
Output: Schema-Grounded Intermediate Representation $\hat{Q}$

Build a schema-specific prefix tree T from S;
Initialize $\hat{Q} \leftarrow \emptyset$, $node \leftarrow T$.root;
while *decoding is not finished* **do**
  Identify the current SG-IR field $f \leftarrow \textsc{CurrentField}(\hat{Q})$;
  if $f \in \{E_{target}, A_{agg}, J_{paths}\}$ **then**
    $\mathcal{V}_{\text{valid}} \leftarrow \textsc{ValidTokens}(node)$; `// Schema-strict decoding`
  else if $f = F_{filter}$ **then**
    $\mathcal{V}_{\text{valid}} \leftarrow \textsc{HybridTokens}(node)$; `// Relax value constraints`
  else if $f = L_{logic}$ **then**
    $\mathcal{V}_{\text{valid}} \leftarrow \mathcal{V}$; `// Unconstrained logic generation`
  Obtain logits $\mathbf{z} \leftarrow \mathcal{M}(\hat{Q}, P)$;
  Mask $\mathbf{z}$ using $\mathcal{V}_{\text{valid}}$;
  Sample the next token x_t from the masked distribution;
  Append x_t to $\hat{Q}$;
  if $f \in \{E_{target}, J_{paths}\}$ **then**
    $node \leftarrow node.\textsc{Next}(x_t)$;
    if x_t *defines a schema alias* **then**
      Insert alias x_t into T;
return $\hat{Q}$

signal to guide the subsequent SQL reconstruction stage. Algorithm 1 summarizes the constrained decoding procedure.

C. Complexity-Adaptive Routing

To optimize the trade-off between inference cost and reasoning accuracy, we introduce a routing mechanism [17] that dynamically allocates computational resources based on query complexity.

1) Complexity Criteria: We employ a lightweight prompt to analyze linguistic and logical features. The model explicitly identifies Complex queries as those meeting the following criteria:

- **High-order Multi-table Joins:** Queries necessitating joins across three or more distinct tables, or those exhibiting cyclic schema dependencies that complicate the join path.
- **Deeply Nested Logic:** Queries characterized by recursive structures where sub-queries are embedded within conditional clauses, requiring multi-step reasoning.
- **Advanced Set Operations:** Queries utilizing composite operators such as intersection, difference, and union, as well as those containing aggregations with ambiguous scoping.

Conversely, queries involving standard single-table retrieval or basic joins are labeled as Simple.

2) Uncertainty-Aware Decision: While the prompt provides the classification criteria, relying solely on the generated text label ignores the model's intrinsic uncertainty. To address this, we frame the routing as a probabilistic classification task. Let $\mathcal{V} = \{\text{Simple}, \text{Complex}\}$ be the set of target labels. We compute the calibrated confidence score S_{conf} for the simple class by normalizing the logits over the target set:

$$S_{\text{conf}} = \frac{\exp(\mathbf{z}_{\text{Simple}})}{\exp(\mathbf{z}_{\text{Simple}}) + \exp(\mathbf{z}_{\text{Complex}})} \tag{3}$$

where $\mathbf{z}$ represents the output logits of the LLM. The routing decision is then governed by a calibration threshold τ:

$$\text{Path}(Q) = \begin{cases} \text{Direct Generation} & \text{if } S_{\text{conf}} > \tau \\ \text{Decomposition (Complex)} & \text{otherwise} \end{cases} \tag{4}$$

A query is routed to the Simple Path only if the model identifies it as Simple with high confidence. Any query that induces model hesitation, indicating low confidence, is defaulted to the robust complex path even when the argmax label is Simple.

3) Direct Generation (Simple Path): For queries classified as simple by the adaptive router, we employ a direct generation strategy designed to minimize inference latency and token consumption. In this pathway, the system bypasses the complex decomposition and selection modules, relying on a robust zero-shot prompting mechanism to map the user intent to SQL. The model generates the SQL query Y directly from the raw natural language query Q and the database schema S:

$$Y = M_{gen}(Q, S) \tag{5}$$

Specifically, this path follows a Standard Zero-Shot configuration. We follow the Basic Prompt organization defined in DAIL-SQL [8], the input concatenates the CREATE TABLE statements with the user question, prefixed by a system instruction to output valid SQLite queries.

4) Decomposition-and-Reconstruction (Complex Path): For complex queries, we trigger a structure-aware reconstruction process. Unlike standard generation which relies solely on the noisy raw query, we introduce the parsed SG-IR as an explicit reasoning plan to guide the generation.

- **Decomposition:** The query is first processed by the Intent Parsing module to yield the Schema-Grounded Intermediate Representation $\hat{Q}$. This step acts as a hard filter, identifying the precise subgraph of the schema relevant to the user's intent.
- **Reconstruction:** Given the inherent ambiguity in complex user intents, a single decoding path is often insufficient, we employ a Diversity-Promoting Sampling strategy. Instead of relying on a single deterministic output, we generate a set of k diverse candidate SQL queries by sampling with a specific temperature T. The generation model is conditioned on both the original query Q and the intermediate representation $\hat{Q}$ to generate a set of candidate SQLs. Formally:

$$\mathcal{Y} = \{Y_1, Y_2, ..., Y_k\} \sim M_{gen}(Q, \hat{Q}) \tag{6}$$

Here, the original query Q provides the necessary natural language context that might be abstract in the JSON representation.

D. Execution-Guided Semantic Selection

To identify the correct SQL query from the candidate set $\mathcal{Y}$, it is essential to distinguish valid reasoning paths from stochastic noise. We propose an execution-guided semantic selection strategy to filter candidates and determine the final output Y_{final}. The selection procedure integrates execution consistency, efficiency, and semantic alignment in a three-stage pipeline.

Stage 1: Execution Clustering

All candidates are executed in a sandboxed environment and grouped according to execution result equivalence:

$$\mathcal{C} = \{G_j \mid G_j = \{Y_i \mid \mathrm{E}(Y_i, D) = Res_j\}\}. \tag{7}$$

This clustering step captures execution-level semantic equivalence among sampled candidates. Clusters with the largest cardinality are retained, as they represent the most consistent reasoning outcomes across multiple generations.

Stage 2: Representative Selection

To eliminate redundant variants within each retained cluster, we select a representative query based on execution efficiency. Specifically, from each top-sized cluster G_j, we select the query with the minimum execution latency:

$$\mathcal{C}' = \left\{ \arg\min_{Y_i \in G_j} \mathrm{Latency}(Y_i) \;\middle|\; |G_j| = \max_m |G_m| \right\}. \tag{8}$$

In the special case where a single largest cluster G^* exists, this step directly yields the final candidate:

$$Y_{\text{final}} = \arg\min_{Y_i \in G^*} \mathrm{Latency}(Y_i). \tag{9}$$

Stage 3: Semantic Alignment

For each remaining candidate $Y_i \in \mathcal{C}'$, we generate a natural language description $Des_i = \mathrm{NL}(Y_i)$ and compute its semantic similarity to the original question Q:

$$\mathrm{Sim}(Des_i, Q) = \cos(\mathbf{e}(Des_i), \mathbf{e}(Q)). \tag{10}$$

The final output is selected as:

$$Y_{\text{final}} = \arg\max_{Y_i \in \mathcal{C}'} \mathrm{Sim}(Des_i, Q). \tag{11}$$

III. Experimental Setup

A. Datasets and Metrics

Spider Spider [4] is a standard cross-domain Text-to-SQL benchmark consisting of 10,181 questions over 200 databases with diverse schemas. In this work, Spider serves as the primary benchmark under strictly zero-shot settings to compare against strong baselines, evaluate generalization across model backbones, and validate the proposed Complexity-Adaptive Router. It is also used as the primary dataset for conducting ablation studies and diagnostic analyses of individual components. We report Execution Accuracy (EX) on both the development and test sets.

BIRD To complement the evaluation on Spider, we conduct experiments on the BIRD benchmark [13], which reflects real-world Text-to-SQL scenarios with large-scale databases and noisy values. In this work, BIRD provides a complementary robustness evaluation of SIPC-SQL under challenging and realistic conditions. Following the official evaluation protocol, we report Execution Accuracy (EX) and Valid Efficiency Score (VES).

B. Baselines

To comprehensively evaluate the effectiveness of SIPC-SQL, we compare our method against representative state-of-the-art baselines categorized into three distinct paradigms.

1. Fine-tuning Methods These methods involve updating model parameters on domain-specific datasets.

- RESDSQL-3B [12]: A framework that decouples schema linking and skeleton parsing, fine-tuned on T5-3B.
- DTS-SQL [5]: A recent two-stage fine-tuning approach based on the DeepSeek-7B model.

2. Zero-shot Prompting Methods These methods, like ours, do not require example retrieval.

- ChatGPT-SQL [10]: Standard zero-shot prompting using the ChatGPT API.
- C3 [9]: A zero-shot prompting method that focuses on Clear Context and Calibration, serving as a strong baseline for prompt engineering without demonstrations.

3. Few-shot Prompting Methods (In-Context Learning) These methods retrieve similar examples to guide generation and typically employ the powerful GPT-4 model. They serve as a high-performance reference point.

- DIN-SQL [7]: A decomposed in-context learning approach that breaks down SQL generation into sub-tasks.
- DAIL-SQL [8]: A framework that optimizes example selection and organization for few-shot learning.

C. Implementation Details

Model Setup For our SIPC-SQL, we utilize the open-weights Qwen-2.5-72B-Instruct [14] as the backbone model under a zero-shot setting. This setup allows for a fair comparison of how our structured reasoning framework enhances performance compared to both closed-source giants and specialized fine-tuned models. Furthermore, we employ the ALBERT [15] model as the embedding encoder to calculate semantic similarity.

Routing Setup The complexity-adaptive router operates with a confidence calibration threshold τ. We performed a grid search on the Spider development set to optimize τ for maximum recall of complex queries. We determined $\tau = 0.8$ as the optimal setting, which prioritizes safety by routing uncertain queries to the robust decomposition path.

Candidate Sampling To determine the optimal number of candidates k, we conducted a grid search on the development set to balance reasoning accuracy and computational overhead. Empirical results indicated a performance saturation point at $k = 12$, with negligible gains observed for larger

TABLE I
EXECUTION ACCURACY COMPARISON ACROSS FINE-TUNED, ZERO-SHOT, AND FEW-SHOT METHODS ON THE SPIDER DEVELOPMENT AND TEST SETS

Method Classification	Method Name	Setting	Backbone Model	Spider (Dev)	Spider (Test)
Fine-tuning Methods	RESDSQL-3B	Fine-tune	T5-3B	84.1	79.9
	DTS-SQL	Fine-tune	DeepSeek-7B	85.5	84.4
Prompting Methods	ChatGPT-SQL	Zero-shot	ChatGPT	72.3	-
	C3	Zero-shot	ChatGPT	82.3	81.8
	DIN-SQL	Few-shot	GPT-4	82.7	85.3
	DAIL-SQL	Few-shot	GPT-4	84.4	86.6
Our Method (Ours)	SIPC-SQL	Zero-shot	Qwen-2.5-72B	84.2	84.8

TABLE II
ABLATION STUDY ON THE SPIDER AND BIRD DEVELOPMENT SETS. ALL VARIANTS ARE COMPARED AGAINST THE FULL SIPC-SQL FRAMEWORK IN TERMS OF EXECUTION ACCURACY AND EFFICIENCY

#	Method	Spider Dev (EX)	BIRD Dev (EX)	BIRD Dev (VES)
1	SIPC-SQL (Full)	84.2	47.3	51.3
2	w/o SG-IR	79.6(↓ 4.6)	39.2(↓ 8.1)	42.0(↓ 9.3)
3	w/o Schema-Strict Constrained Decoding	82.5(↓ 1.7)	43.8(↓ 3.5)	47.5(↓ 3.8)
4	w/o Adaptive Router (All-Complex)	84.5(↑ 0.3)	47.9(↑ 0.6)	53.8(↑ 2.5)

values. Accordingly, we set $k = 12$ for all subsequent experiments. During the reconstruction stage, we employ a Diversity-Promoting Sampling strategy with the temperature set to $T = 0.7$. This elevated temperature encourages the model to explore diverse reasoning paths and valid schema traversals, mitigating stochastic errors inherent in standard LLM generation.

IV. RESULTS AND ANALYSIS

A. Main Results

We evaluated the execution accuracy of SIPC-SQL on the Spider development and test sets. Table I summarizes the performance comparison across fine-tuning methods, zero-shot prompting methods, and few-shot in-context learning baselines.

SIPC-SQL achieves an execution accuracy of 84.2% on the development set and 84.8% on the test set. Compared with the previous best zero-shot method, C3, SIPC-SQL improves execution accuracy by 3.0%. Under a strictly zero-shot setting, SIPC-SQL also achieves comparable performance to strong few-shot baselines such as DIN-SQL and DAIL-SQL, with a performance gap within 1.8%. In addition, SIPC-SQL attains higher execution accuracy than representative fine-tuned methods, including RESDSQL-3B and DTS-SQL.

Overall, these results indicate that SIPC-SQL achieves competitive execution accuracy under zero-shot settings while avoiding fine-tuning and in-context demonstrations.

B. Ablation Study

To evaluate the contribution of each component in the SIPC-SQL framework, we conduct an ablation study on the Spider and BIRD development sets. The results are summarized in Table II.

Removing the Structure-Aware Intent Parsing module leads to the most substantial performance degradation. When SG-IR generation is bypassed and queries are processed directly from the raw natural language input, execution accuracy drops by 4.6% on Spider and 8.1% on BIRD. The larger degradation on BIRD is consistent with its more complex schemas and noisier database values, indicating that structured intent grounding plays a critical role in aligning ambiguous queries with large-scale schemas.

Excluding Schema-Strict Constrained Decoding also results in a noticeable performance decline, with execution accuracy decreasing by 3.5% on BIRD. This suggests that enforcing schema-level constraints during intent parsing is particularly important in large databases, where unconstrained generation is more prone to producing invalid schema references.

In contrast, disabling the Adaptive Router and forcing all queries to follow the complex path yields only marginal accuracy gains. This observation indicates that the routing strategy primarily contributes to efficiency improvements while preserving execution accuracy.

Finally, the full SIPC-SQL framework achieves a Valid Efficiency Score (VES) of 51.3% on BIRD, whereas the ablated variants consistently exhibit lower VES values. This result indicates that jointly applying structured intent parsing, constrained decoding, and adaptive routing improves not only execution correctness but also query efficiency.

C. Efficiency-Accuracy Analysis

A key contribution of this work is the Complexity-Adaptive Routing. To validate its efficiency, we compare the average token consumption per query against the accuracy on the Spider Dev set. As shown in Table III , we compare three settings:

All-Simple Forces all queries through the Direct Generation path. While extremely efficient, it suffers from a significant accuracy drop.

TABLE III
EFFICIENCY–ACCURACY TRADE-OFF ON THE SPIDER DEVELOPMENT SET, COMPARING AVERAGE TOKEN CONSUMPTION AND EXECUTION ACCURACY UNDER ALL-SIMPLE, ALL-COMPLEX, AND ADAPTIVE ROUTING STRATEGIES

Routing Strategy	EX (%)	Avg. Cost (Tokens)
All-Complex	84.5	1,468
All-Simple	79.1	412
Adaptive (Ours)	84.2	887

TABLE IV
ABLATION OF SELECTION STRATEGIES ON THE SPIDER DEVELOPMENT SET, ISOLATING THE EFFECT OF EXECUTION-GUIDED AND SEMANTIC-AWARE RE-RANKING UNDER A FIXED UPSTREAM GENERATION PIPELINE

Selection Strategy	EX (%)	△ vs. Baseline
Greedy Decoding	82.4	-
Standard Self-Consistency	83.6	+1.2%
SIPC-Selection (Ours)	84.2	+1.8%

All-Complex Forces all queries through the Decomposition-and-Reconstruction path. This yields the peak accuracy but incurs the highest computational cost.

Adaptive (Ours) Uses the Complexity-Adaptive Routing. As shown in Table III, the Adaptive Router directs approximately 55% of queries to the simple path, reducing the average token consumption from 1,468 to 887 tokens, corresponding to a 39.6% cost reduction. At the same time, SIPC-SQL retains 99.6% of the peak accuracy achieved by the All-Complex setting, with only a 0.3% absolute accuracy difference. It reflects a deliberate strategic trade-off where we accept a negligible risk of misclassifying edge cases in exchange for massive reductions in latency. For real-world deployments, a cost saving of nearly 40% significantly outweighs the utility of a less than 0.5% improvement in theoretical accuracy.

While the standard EX score indicates a small gap, this metric does not fully capture the semantic correctness of the simple path. To further substantiate that our routing strategy does not compromise real-world reliability, we provide a more granular execution-level verification in Section IV-F.

D. Effectiveness of Execution-Guided Selection

To quantify the contribution of the execution-guided selection module, we conduct a controlled ablation study on the Spider development set. The results are reported in Table IV. This experiment aims to distinguish performance gains introduced by the selection strategy itself from those arising solely from candidate sampling.

To isolate the effect of selection, we fix the upstream generation pipeline across all compared methods. All settings in this section employ the same Structure-Aware Intent Parsing and Complexity-Adaptive Routing. The only difference lies in the decoding and selection strategy:

Greedy Decoding generates a single SQL query using greedy search [16].

Standard Self-Consistency samples multiple candidates and selects the final output via majority voting on execution results [19], without considering efficiency or semantic alignment.

SIPC-Selection (Ours) samples multiple candidates and applies the proposed execution-guided selection strategy.

As shown in Table IV, standard self-consistency improves execution accuracy by 1.2% over greedy decoding, indicating that sampling multiple candidates helps mitigate stochastic generation errors. Under the same sampling setting, SIPC-Selection further improves accuracy to 84.2%, yielding an additional gain of 0.6% over standard majority voting.

These results suggest that the observed improvement is not solely due to candidate sampling, but also benefits from the execution-guided and semantic-aware selection mechanism. By introducing semantic alignment as a re-ranking signal, the selection module filters semantically deviated candidates and prioritizes concise SQL among execution-equivalent variants. This multi-dimensional verification ensures the final output is not only correct but also structurally optimized.

TABLE V
EXECUTION ACCURACY GAINS OF SIPC-SQL OVER STANDARD ZERO-SHOT PROMPTING ACROSS DIFFERENT LLM BACKBONES ON THE SPIDER DEVELOPMENT SET

Backbone Model	Method	EX (%)	Improvement
Qwen-2.5-72B	Standard Zero-Shot	79.1	-
	SIPC-SQL (Ours)	84.2	+5.1%
Llama-3-70B	Standard Zero-Shot	77.5	-
	SIPC-SQL (Ours)	83.2	+5.7%
DeepSeek-Coder-33B	Standard Zero-Shot	74.8	-
	SIPC-SQL (Ours)	81.5	+6.7%

E. Generalizability Across Backbones

To assess whether the effectiveness of SIPC-SQL depends on a specific backbone model or generalizes across architectures, we evaluate the framework on two additional large language models with different parameter scales and specializations: Llama-3-70B-Instruct, a strong general-purpose model, and DeepSeek-Coder-33B, a code-specialized model.

For each backbone, we consider two settings. **Standard Zero-Shot** evaluates the vanilla performance of the backbone without any framework-level enhancements, using the same prompting format as the Simple Path (Section II-C3) to directly map the user query and schema to SQL. **SIPC-SQL (Ours)** integrates the backbone into the full framework, where SQL generation is routed through either the direct path or the SG-IR–guided reconstruction path via Complexity-Adaptive Routing.

As shown in Table V, SIPC-SQL consistently improves execution accuracy across all evaluated backbones, with gains ranging from 5.1% to 6.7%. These results indicate that the performance improvements are not tied to a specific model

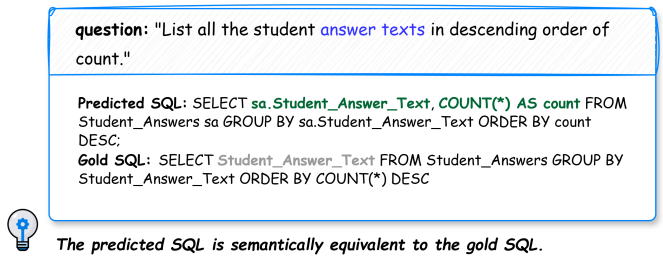

Fig. 2. Example of intent-matching SQL predictions that are semantically equivalent to the gold SQL but differ in non-critical output fields, leading to execution mismatches.

architecture and extend to both general-purpose and code-specialized LLMs.

In addition, the comparison reveals a clear decoupling between model scale and execution accuracy. When equipped with SIPC-SQL, the smaller DeepSeek-Coder-33B achieves an execution accuracy of 81.5%, exceeding the Standard Zero-Shot performance of the larger Llama-3-70B. This observation suggests that structured intent grounding via SG-IR can substantially improve the effectiveness of compact models, enabling competitive or superior performance without relying solely on increased model size.

F. Reliability Analysis of Simple Path under Adaptive Routing

To assess whether routing queries to the Simple Path compromises execution reliability, we analyze cases where the execution result of the predicted SQL differs from that of the gold SQL. Figure 3 presents a hierarchical breakdown of these mismatches.

At the top level, execution discrepancies are first categorized according to the correctness of the gold SQL. As shown in the inner ring of Figure 3, 50% of the mismatches are attributed to incorrect gold SQLs, 45% correspond to cases where the gold SQL is correct, and the remaining 5% are caused by missing or inconsistent database entries.

For cases with incorrect gold SQLs (50%), the majority of predictions are in fact correct: 40% of all mismatches fall into this category, indicating annotation defects such as syntactic errors, logical incompleteness, or semantic misalignment in the gold SQL, as documented in Figure 4. Only 10% of the mismatches involve errors in both the gold and predicted SQLs.

For cases where the gold SQL is correct (45%), mismatches are further divided into genuinely incorrect predictions (33%) and intent-matching predictions (12%), as illustrated in Figure 2. The latter refers to cases where the predicted SQL is semantically equivalent to the gold SQL but differs in non-critical aspects of the output, and therefore should not be considered true model failures. Among the genuinely incorrect predictions, errors are mainly caused by incorrect join paths or aggregation logic (21%) and cell value mismatches between natural language expressions and database entries (12%).

Based on this hierarchical analysis, actual model failures are defined as cases where the gold SQL is correct and the predicted SQL is incorrect, or where both are incorrect. Under this definition, the execution accuracy of the Simple Path reaches 98.2%, indicating that direct generation achieves high reliability while substantially reducing computational cost.

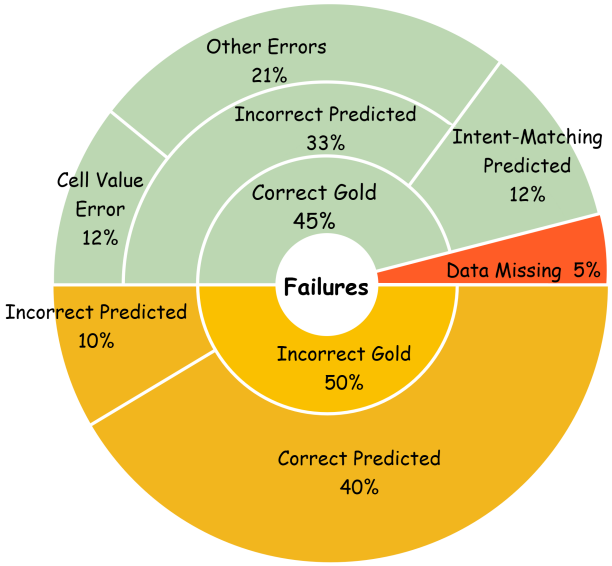

Fig. 3. Hierarchical breakdown of execution discrepancy sources for simple queries, categorized by gold SQL correctness and prediction outcomes.

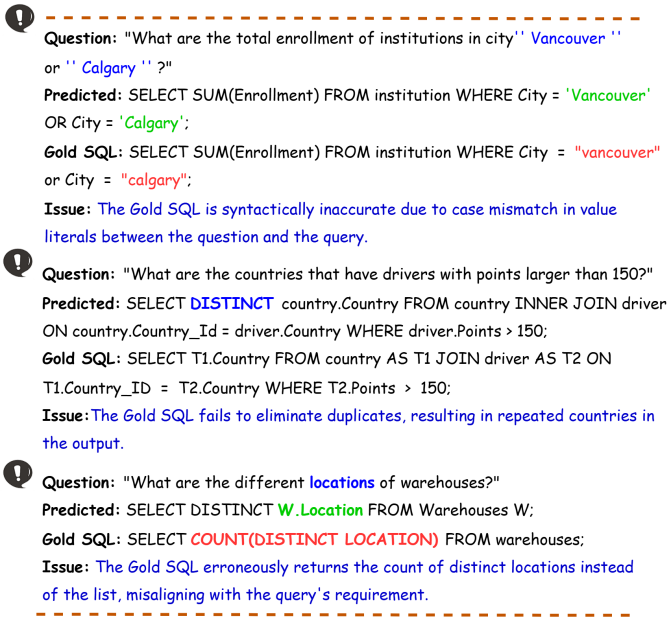

Fig. 4. Breakdown of 40% mismatched cases caused by defective gold SQLs, categorized by error type.

G. Diagnostic Analysis of Adaptive Routing

To open the black box of the Complexity-Adaptive Routing and quantify its decision integrity, we perform a diagnostic evaluation on the 1,034 queries of the Spider development set. Based on manual annotation, the dataset contains 67.1%

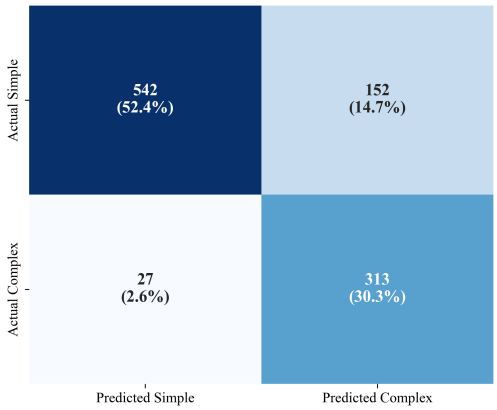

Fig. 5. Confusion matrix of the Complexity-Adaptive Router on the Spider development set, illustrating routing decisions for Simple and Complex queries.

Simple queries and 32.9% Complex queries. Figure 5 presents the confusion matrix of our router's predictions.

The router achieves a 92.1% Recall for Complex queries, successfully capturing 313 out of 340 complex instances. Only 2.6% of total queries are complex but misrouted to the simple path. While 67.1% of queries are actually simple, the router only assigns 55% to the simple path. The 12.1% gap represents over-guarded simple queries that are routed to the complex path for extra verification. This conservative bias ensures a high Precision of 95.3% for the Simple Path, guaranteeing that queries undergoing direct generation are almost exclusively low-risk.

V. Conclusion

In this paper, we proposed SIPC-SQL, a structured inference framework for zero-shot Text-to-SQL that addresses both structural reliability and inference efficiency. By decoupling intent understanding from SQL generation through schema-aware intent parsing and enforcing schema-strict constrained decoding, SIPC-SQL effectively mitigates schema hallucinations in complex databases. Furthermore, the proposed complexity-adaptive routing dynamically balances accuracy and computational cost, achieving substantial efficiency gains with minimal performance degradation.

Extensive experiments on the Spider and BIRD benchmarks demonstrate that SIPC-SQL establishes a new state-of-the-art among zero-shot methods while generalizing consistently across different backbone models. These results suggest that structured intent grounding and adaptive inference control provide a practical and scalable alternative to model scaling and demonstration-based prompting for reliable Text-to-SQL systems.

VI. Acknowledgments

This research is supported by the Key R&D Program of Shandong Province No. 2024CXGC010111, and in part supported by Jinan Science and Technology Bureau, China No. 202333043.

References

[1] R. Cai, B. Xu, Z. Zhang, X. Yang, Z. Li, and Z. Liang, "An Encoder-Decoder Framework Translating Natural Language to Database Queries," in Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 3977–3983. doi: 10.24963/ijcai.2018/553.

[2] X. Xu, C. Liu, and D. Song, "SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning," Nov. 13, 2017, arXiv: arXiv:1711.04436. doi: 10.48550/arXiv.1711.04436.

[3] J. M. Zelle and R. J. Mooney, "1996-Learning to Parse Database Queries Using Inductive Logic Programming".

[4] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," Feb. 02, 2019, arXiv: arXiv:1809.08887. doi: 10.48550/arXiv.1809.08887.

[5] M. Pourreza and D. Rafiei, "DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models," Feb. 02, 2024, arXiv: arXiv:2402.01117. doi: 10.48550/arXiv.2402.01117.

[6] X.-B. Nguyen, X.-H. Phan, and M. Piccardi, "Fine-tuning text-to-SQL models with reinforcement-learning training objectives," Natural Language Processing Journal, vol. 10, p. 100135, Mar. 2025, doi: 10.1016/j.nlp.2025.100135.

[7] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction," Nov. 02, 2023, arXiv: arXiv:2304.11015. doi: 10.48550/arXiv.2304.11015.

[8] D. Gao et al., "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation," Nov. 20, 2023, arXiv: arXiv:2308.15363. doi: 10.48550/arXiv.2308.15363.

[9] X. Dong et al., "C3: Zero-shot Text-to-SQL with ChatGPT," Jul. 14, 2023, arXiv: arXiv:2307.07306. doi: 10.48550/arXiv.2307.07306.

[10] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability," Mar. 12, 2023, arXiv: arXiv:2303.13547. doi: 10.48550/arXiv.2303.13547.

[11] W. Lei et al., "Re-examining the Role of Schema Linking in Text-to-SQL," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online: Association for Computational Linguistics, 2020, pp. 6943–6954. doi: 10.18653/v1/2020.emnlp-main.564.

[12] H. Li, J. Zhang, C. Li, and H. Chen, "RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL," AAAI, vol. 37, no. 11, pp. 13067–13075, Jun. 2023, doi: 10.1609/aaai.v37i11.26535.

[13] J. Li et al., "Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs," Nov. 15, 2023, arXiv: arXiv:2305.03111. doi: 10.48550/arXiv.2305.03111.

[14] A. Yang et al., "Qwen2 Technical Report," Sep. 10, 2024, arXiv: arXiv:2407.10671. doi: 10.48550/arXiv.2407.10671.

[15] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," Feb. 11, 2020, arXiv: arXiv:1909.11942. doi: 10.48550/arXiv.1909.11942.

[16] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," Feb. 14, 2020, arXiv: arXiv:1904.09751. doi: 10.48550/arXiv.1904.09751.

[17] T. Schuster et al., "Confident Adaptive Language Modeling," in Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., Curran Associates, Inc., 2022, pp. 17456–17472.

[18] G. Poesia et al., "Synchromesh: Reliable Code Generation from Pre-trained Language Models," 2022.

[19] X. Wang et al., "Self-Consistency Improves Chain of Thought Reasoning in Language Models," 2023.