SQLStructEval: Structural Evaluation of LLM Text-to-SQL Generation

Yixi Zhou[1,2*] **Fan Zhang**[3*] **Zhiqiao Guo**[4*] **Yu Chen**[3†]
Haipeng Zhang[2†] **Preslav Nakov**[5] **Zhuohan Xie**[5]
[1]Hong Kong Baptist University [2]ShanghaiTech University
[3]The University of Tokyo [4]University of Michigan [5]MBZUAI
yxzhou@comp.hkbu.edu.hk, zhang-fan@g.ecc.u-tokyo.ac.jp
guozhq@umich.edu, chen@edu.k.u-tokyo.ac.jp, zhanghp@shanghaitech.edu.cn
{preslav.nakov,zhuohan.xie}@mbzuai.ac.ae

Abstract

Large language models (LLMs) can generate SQL queries that return correct answers while differing in program structure. We introduce SQLStructEval, a framework that analyzes this behavior through canonical abstract syntax tree (AST) representations. Experiments on Spider across seven models reveal structural variation within execution-correct generations and sensitivity to question paraphrases and schema presentation. Additional 100-example subsets of BIRD Mini-Dev, Spider-Syn, and Dr.Spider provide scope validation. As a mitigation case study, a pipeline that generates structured intermediate representations before deterministic compilation improves execution accuracy and structural agreement among correct outputs in the evaluated setting. Structural diversity is not inherently erroneous: our measures expose differences for further inspection rather than determine semantic correctness. These findings establish structural analysis as a complementary diagnostic alongside execution-based evaluation. Our code is available at https://anonymous.4open.science/r/StructEval-2435.

1 Introduction

Large language models (LLMs) support program generation from natural language, including code synthesis and Text-to-SQL translation (Chen et al., 2021; Austin et al., 2021; Rajkumar et al., 2022). Evaluation commonly focuses on functional correctness: unit tests check generated code, and execution accuracy checks whether SQL produces the reference result on a database (Yu et al., 2018; Zhong et al., 2020). These tests are essential for evaluating whether a program answers the intended question. They leave open a different question: how consistently does a model construct that program across repeated generations and equivalent presentations of the input?

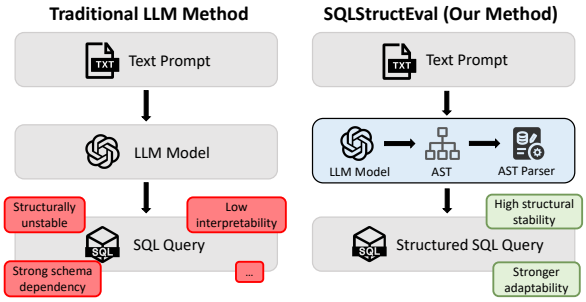


Figure 1: Comparison between traditional LLM-based Text-to-SQL generation and our proposed SQLStructEval framework. Traditional methods directly generate SQL queries from text prompts, which often leads to structurally inconsistent outputs across repeated generations. In contrast, SQLStructEval introduces an AST-based representation that enables explicit structural analysis and improves the stability and comparability of generated programs.

Programs that return the same result can use different joins, nested subqueries, aggregation arguments, or redundant operators. Some differences are harmless alternatives; others agree only on the evaluated database instance. Execution agreement alone does not distinguish these cases. Moreover, a model can change its preferred formulation when only the wording of a question or the order of schema elements changes. Studying such behavior requires comparing distributions of generated programs, rather than evaluating each output in isolation.

We introduce SQLStructEval to summarize canonical SQL structures across repeated generations. It measures concentration, diversity, reference alignment, and agreement under input perturbations. Figure 1 illustrates this structural perspective. We analyze execution-correct outputs separately and include a compile-style generation case study.

*Equal contribution.
†Corresponding authors.

We use *structural reliability* operationally to describe consistency of canonical program representations across repeated generations and semantically invariant input changes. This definition does not imply that the least diverse model is the best model. A system can consistently generate an incorrect query, and multiple valid formulations can remain equally useful. Structural analysis instead reveals where a fixed intent admits different generated constructions and where further semantic or downstream inspection may be informative.

Across seven LLMs on Spider, we observe structural disagreement even within execution-correct outputs. Perturbations can also change the dominant generated structure. Additional scope validation on three 100-example subsets shows that execution-correct disagreement appears beyond the original Spider development set. Finally, a compile-style pipeline improves execution accuracy and agreement among correct outputs in our setting. This is evidence about the complete pipeline; it does not isolate the effects of the intermediate representation, compiler, or model reasoning.

Our contributions are as follows:

- We show that execution-correct SQL generations can remain structurally diverse, exposing behavior that a single execution-accuracy score does not describe.
- We introduce SQLStructEval, a framework for measuring structural concentration, diversity, reference alignment, and perturbation sensitivity through canonical AST representations.
- We document this behavior across model families and additional dataset subsets, and evaluate compile-style generation as a full-pipeline mitigation case study with explicit limits on causal attribution.

2 Related Work

Variation in model generations. Self-consistency aggregates multiple sampled reasoning paths (Wang et al., 2023); related work studies the propagation of early errors (Zhang et al., 2024), limits of self-correction (Huang et al., 2024), and the cost of repeated sampling (Zhu et al., 2025). Instance-level randomization addresses variation in evaluation scores arising from random experimental factors (Li et al., 2025). These studies motivate distinguishing a model's behavior across samples from the outcome of a single generation.

Program structure has also received direct attention. Song et al. (2025) analyze code-generation stability through AST-based structural entropy, while Rajput et al. (2025) study algorithmic and runtime variation through opcode distributions. Our work focuses on SQL and connects repeated-generation structural statistics with execution-correct subsets and semantically invariant input perturbations. We do not claim that AST analysis alone measures runtime costs or establishes a unique correct program structure.

Execution and semantic evaluation. Text-to-SQL benchmarks and surveys document the role of execution accuracy and reference-based evaluation (Yu et al., 2018; Deng et al., 2022). HumanEval and MBPP similarly test generated code through functional behavior (Chen et al., 2021; Austin et al., 2021). Stronger tests can expose errors missed by an original test set (Liu et al., 2023). For SQL, distilled test suites probe semantic equivalence across database instances (Zhong et al., 2020), and FLEX studies false judgments in execution-based evaluation (Kim et al., 2025). These approaches improve correctness assessment. SQLStructEval instead characterizes how generated structures vary, including among outputs that pass the available execution check.

Robustness benchmarks. Spider-Syn examines sensitivity to synonym substitutions in natural language questions (Gan et al., 2021). Dr.Spider provides diagnostic perturbations affecting questions, databases, and SQL (Chang et al., 2023). BIRD broadens evaluation toward larger, database-grounded Text-to-SQL settings (Li et al., 2023). We use small subsets of these resources to validate the presence of execution-correct structural disagreement. This analysis complements their original evaluation goals and does not constitute a full benchmark comparison.

Structured program generation. Intermediate representations already play an important role in semantic parsing. IRNet generates a SemQL representation before constructing SQL (Guo et al., 2019). DIN-SQL decomposes prompting into several steps (Pourreza and Rafiei, 2023), while LEVER uses execution to verify language-to-code candidates (Ni et al., 2023). Synchromesh and PICARD constrain generation using program struc-

ture or incremental parsing (Poesia et al., 2022; Scholak et al., 2021). Grammar-constrained decoding further supports structured outputs and logical parsing (Geng et al., 2023; Raspanti et al., 2025); other methods incorporate context-sensitive or logical constraints (Albinhassan et al., 2025; Ma and Hu, 2025).

These methods differ in their representations, constraints, and training or inference procedures. Our compile-style experiment is a case study within a structural evaluation framework, rather than a claim to invent intermediate representations or outperform the latest Text-to-SQL systems. We evaluate the final compiled SQL through the same structural analysis used for directly generated SQL.

3 SQLStructEval

3.1 Problem Setup

Given a natural language question x and database schema S, a Text-to-SQL model generates a query intended to answer x. We sample N candidate queries $\{y_i\}_{i=1}^{N}$ for the same input. Each candidate receives two separate assessments: whether its execution result matches the annotated reference, and which canonical program structure it represents. The latter permits comparisons within the full sample and within the execution-correct subset.

3.2 Structural Representation

An AST represents a query through operators such as `SELECT`, `JOIN`, `WHERE`, and `GROUP BY`, together with their arguments and syntactic relationships. We parse SQL using `sqlglot` in the SQLite dialect and apply alias and logical-operator normalization. We rename table aliases by first appearance, update column references, and flatten and sort conjunctions. We then render the AST into SQL and normalize whitespace, case, trailing semicolons, and simple column aliases. Appendix D describes this procedure.

The resulting canonical SQL string serves as the comparison representation. Equality means equality under these normalization rules, rather than full semantic equivalence. In particular, the representation retains alternative subqueries, aggregation arguments, and other query constructions. It is therefore more specific than an unlabeled tree shape and less sensitive to formatting than raw SQL text. All generation paradigms use the same canonicalization procedure, so the compile-style JSON object itself is never the object scored in the structural comparison.

3.3 Structural Evaluation Measures

For a question x, parsing maps generation i to a canonical structure $a_i \in \mathcal{A}$ or a failure $a_i = \perp$. Define the valid index set $\mathcal{I}_x = \{i : a_i \neq \perp\}$ and its size $M_x = |\mathcal{I}_x|$. Among these generations, let $s_1, \ldots, s_K$ be the distinct structures, with counts n_k and relative frequencies $p_k = n_k/M_x$.

Concentration and diversity. The majority-structure ratio measures concentration:

$$\mathrm{Cons}(x) = \max_{1 \le k \le K} p_k. \tag{1}$$

We report this as *Majority*. Diversity and dispersion are described by the number of distinct structures and their entropy:

$$\mathrm{Div}(x) = K, \qquad \mathrm{H}(x) = -\sum_{k=1}^{K} p_k \log_2 p_k. \tag{2}$$

Table 1 reports normalized entropy $\mathrm{H}_{\mathrm{norm}}(x) = \mathrm{H}(x)/\log_2 K$ when $K > 1$, and zero when $K \le 1$. The additional subset table reports unnormalized entropy. These quantities answer different questions. Two samples can have the same distinct count but distribute their probability mass differently. For example, counts of nine and one have a higher majority ratio and lower entropy than counts of five and five, although both contain two structures. Diversity is descriptive; a lower count or entropy is not universally preferable.

Annotated-reference alignment. Let a^* be the canonical representation of the annotated reference SQL. We compute

$$\mathrm{RefAlign}(x) = \frac{1}{M_x} \sum_{i \in \mathcal{I}_x} \mathbf{1}[a_i = a^*]. \tag{3}$$

This is a reference-alignment diagnostic, not structural correctness. The annotated query is one available formulation and does not exhaust all valid ways to answer the question. A low alignment rate may coexist with high execution accuracy.

Execution-correct structural agreement. Let E_x contain all execution-correct generation indices and let $C_x = E_x \cap \mathcal{I}_x$ contain those with a parsed

structure. For a nonempty selected subset $B_x \subseteq \mathcal{I}_x$, define

$$\mathrm{Sim}(B_x) = \frac{\max_{s \in \mathcal{A}} |\{i \in B_x : a_i = s\}|}{|B_x|}. \quad (4)$$

AST Sim (corr) applies this measure to C_x. It measures majority concentration, not pairwise tree-edit similarity. *Distinct (corr)* counts the different structures in the same subset. At the question level, this conditions structure on correct outputs. The reported dataset averages also include questions with no parsed correct output: the implementation records their agreement and distinct count as zero. The aggregate therefore reflects both availability of parsed correct outputs and concentration within available subsets.

Diagnostic mismatch indicators. Let $e(x)$ be the fraction of the N generations whose execution result matches the reference. We report two question-level indicators. *Exec-Corr Struct-Diff* flags questions with $|E_x| \geq 2$ and $\widetilde{\mathrm{Sim}}(C_x) < 0.7$, where $\widetilde{\mathrm{Sim}}$ equals Sim for nonempty subsets and zero otherwise. *High-Acc Low-Struct* flags questions with $e(x) \geq 0.8$ and more than one execution-correct canonical structure. Tables report the fraction of questions satisfying each condition. Because the implementation uses zero for missing structural agreement, Exec-Corr Struct-Diff can also flag an execution-correct question with no parsed correct structure. We interpret this indicator as a low-agreement diagnostic affected by structural coverage, rather than a pure count of observed AST disagreements. High-Acc Low-Struct explicitly requires multiple parsed correct structures.

The cutoffs 0.7 and 0.8 are heuristic diagnostic choices, not learned parameters or universal standards. With ten generations, the accuracy cutoff requires at least eight correct outputs; for nonempty subsets, the concentration cutoff requires that no structure account for 70% of the selected correct subset. The indicators are not interchangeable: high execution accuracy can coexist with two structures even when one is dominant. We interpret them alongside the continuous measures, without claiming invariance to alternative thresholds.

Robustness under perturbations. For an original input $x^{(0)}$ and T variants $x^{(1)}, \ldots, x^{(T)}$, let s_t^* denote the majority structure for variant t. Cross-variant agreement is

$$\mathrm{Cons}_{\mathrm{var}}(x) = \frac{2}{T(T+1)} \sum_{0 \leq u < v \leq T} \mathbf{1}[s_u^* = s_v^*]. \quad (5)$$

Perturbation sensitivity compares each variant with the original:

$$\mathrm{Sens}(x) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}[s_t^* \neq s_0^*]. \quad (6)$$

The sensitive fraction is the proportion of questions with a changed majority structure under at least one perturbation. *AST Sim* in the perturbation tables denotes cross-variant agreement, whereas *AST Sim (corr)* in the execution tables denotes within-subset concentration. Appendix A summarizes the table mappings and parse-failure conventions.

3.4 Compile-Style Generation

We also study a pipeline that asks the model to generate a structured JSON representation with fields for selection, tables, joins, predicates, grouping, ordering, and limits. A deterministic compiler converts this representation into executable SQL. Appendix G illustrates the sequence from question to JSON, internal AST, and SQL.

This pipeline makes program structure explicit during generation, while direct generation emits SQL text. The comparison evaluates their end-to-end behavior. It does not separately identify the effects of JSON constraints, deterministic compilation, prompting, or the model's internal reasoning. Improved agreement in the final SQL is a property of the complete pipeline, not evidence by itself that the model reasons more consistently.

4 Experiments and Evaluation

4.1 Experimental Setup

Datasets. Our main experiments use the 1,034-question Spider development set (Yu et al., 2018). Spider supplies annotated SQL, executable databases, and a controlled setting for measuring execution and structure on the same questions. Experiment 2 additionally includes 100-example subsets of BIRD Mini-Dev, Spider-Syn, and Dr.Spider (Li et al., 2023; Gan et al., 2021; Chang et al., 2023). These are scope checks, not full benchmark evaluations.

Models and sampling. We evaluate GPT-5-mini, GPT-4.1-mini, Claude-4.5-Sonnet, Claude-4.5-Opus, DeepSeek-V3.1, Gemini-2.5-Flash, and

Gemini-3-Pro. Each input includes the question and database schema, including tables, columns, and foreign keys. We sample ten outputs per input, using temperature 1.0 and default nucleus sampling unless the model API restricts these settings. Appendix E provides the prompting details.

Execution and aggregation. We execute each candidate on its corresponding SQLite database and compare its result with the annotated reference result. Execution errors and result mismatches count as incorrect. Execution accuracy is the mean fraction of correct generations per question. Structural statistics summarize canonical representations per question and then average across all questions, recording zero for empty structural subsets. Appendix A records these conventions. Passing this execution check establishes agreement on the evaluated instance, not equivalence on every possible database.

Interpretation of sampled results. The tables summarize the sampled generations rather than repeated independent batches of ten outputs. They do not provide confidence intervals or an estimate of variation across complete reruns. We therefore use the cross-model results to document observed structural behavior and avoid interpreting small differences as statistically established rankings. For the additional 100-question subsets, we also show counts for the diagnostic indicators so that the size of their empirical support is explicit.

4.2 Experiment 1: Repeated-Generation Variance

We first analyze structural variation under an unchanged question and schema. For each model, ten generations on each of the 1,034 questions yield over 10,000 sampled queries. We canonicalize these queries and summarize their structural distributions in Table 1.

GPT-5-mini produces an average of 1.913 distinct structures per question, with a reported majority ratio of 0.650. Other models also exhibit structural variation. The measures describe different aspects of these samples: a distinct count records how many structures occur, whereas the majority ratio and normalized entropy characterize how concentrated the distribution is. Reference alignment adds a separate comparison with the annotated SQL rather than a preference for the majority structure.

The below-one distinct counts for several models require care. Under the count convention, inputs with no parsed structure contribute zero. Consequently, a low average distinct count can reflect missing valid structures as well as concentration among parsed queries. Neither a low count nor a zero-filled concentration average independently establishes stronger stability. We retain the count as a description of the evaluated samples and interpret it together with execution outcomes, rather than rank models by diversity alone.

Model	Distinct	Majority↑	Norm. entropy	Ref Align
GPT-5-mini	1.913	0.650	0.413	0.197
GPT-4.1-mini	1.620	0.793	0.301	0.253
Claude-4.5-Opus	0.779	0.687	0.049	0.391
Claude-4.5-Sonnet	1.241	0.775	0.195	0.356
DeepSeek-V3.1	1.023	0.735	0.136	0.321
Gemini-3-Pro	0.845	0.595	0.133	0.369
Gemini-2.5-Flash	0.623	0.556	0.044	0.332

Table 1: Structural statistics on the 1,034-question Spider development set with ten generations per question. Distinct counts canonical structures, Majority measures concentration, Norm. entropy describes normalized dispersion, and Ref Align measures annotated-reference alignment. Diversity has no universal preferred direction. All-parse-failure inputs contribute zero to Distinct; see Appendix A.

4.3 Experiment 2: Execution and Structure

Experiment 2 reuses the generations from Experiment 1, executes them, and analyzes the execution-correct subset. Table 2 reports execution accuracy and structural statistics. The all-generation distinct counts appear in Table 1; here, the focus is on correct outputs. The auxiliary execution column uses a conditional denominator.

Cond. Exec Acc divides the number of correct outputs by the number that execute without error for each question, using zero when none execute, and averages across questions. It is not an execution-success rate or a question-level success probability. For example, Claude-4.5-Sonnet has accuracy 0.757 over all generations and conditional accuracy 0.804. Excluding execution errors changes the denominator without making incorrect results correct.

GPT-5-mini reaches execution accuracy 0.741, while the correct generations contain 1.378 distinct structures per question on average and have reported AST agreement 0.552. Its Exec-Corr Struct-Diff proportion is 0.297: these questions have low or unavailable correct-subset agreement under the implemented convention. Its High-Acc Low-Struct proportion is 0.351, showing that frequent execution correctness can coexist with multiple correct

Model	Exec Acc↑	Cond. Exec Acc	Distinct (corr)	AST Sim (corr)↑	High-Acc Low-Struct	Exec-Corr Struct-Diff
GPT-5-mini	0.741	0.741	1.378	0.552	0.351	0.297
GPT-4.1-mini	0.754	0.757	1.129	0.656	0.250	0.163
Claude-4.5-Sonnet	0.757	0.804	0.714	0.624	0.054	0.201
Claude-4.5-Opus	0.816	0.816	0.650	0.600	0.038	0.224
DeepSeek-V3.1	0.771	0.771	0.774	0.612	0.101	0.191
Gemini-2.5-Flash	0.638	0.743	0.509	0.484	0.016	0.191
Gemini-3-Pro	**0.842**	0.843	0.721	0.537	0.139	0.337

Table 2: Execution and structure on Spider. Exec Acc is accuracy over all outputs; Cond. Exec Acc conditions on error-free execution; Distinct (corr) and AST Sim (corr) summarize execution-correct structures. AST Sim (corr) averages majority concentration with zeros for empty subsets. High-Acc Low-Struct requires accuracy ≥ 0.8 and multiple correct structures; Exec-Corr Struct-Diff requires at least two correct outputs and agreement < 0.7, including missing agreement recorded as zero. These cutoffs are heuristic. All-generation distinct counts appear in Table 1.

structures.

These quantities are compatible because they select different patterns. A question with eight correct outputs split seven-to-one across two structures passes High-Acc Low-Struct but not Exec-Corr Struct-Diff. Conversely, a question with only two correct outputs, each of a different structure, passes Exec-Corr Struct-Diff but not the high-accuracy criterion. The proportions should therefore not be added together or interpreted as a partition of failures.

The same qualitative distinction between execution and structure appears across model families. The reported Exec-Corr Struct-Diff proportions range from 0.163 to 0.337, including the structural-coverage effect described above. Together with the distinct counts and High-Acc Low-Struct indicator, these results document multiple structures among execution-correct outputs. The low-agreement indicator alone also reflects missing parsed structures. It does not establish that all these disagreements are semantic errors, that every model has the same kind of variation, or that the proportions measure downstream deployment failures.

Scope validation beyond Spider. We apply the same analysis to three additional 100-example subsets, generating ten SQL queries per question with GPT-5-mini, Claude-4.5-Sonnet, and DeepSeek-V3.1. Table 3 shows the results and the diagnostic counts alongside the proportions. The selected models provide a cross-family check of the phenomenon, rather than the complete seven-model comparison used on Spider.

The reported low-agreement diagnostic is nonzero on all three subsets. Exec-Corr Struct-Diff ranges from 6 to 17 questions on BIRD Mini-Dev, from 1 to 22 on Spider-Syn, and from 3 to 11 on Dr.Spider. The one-question result on Spider-Syn is particularly small and should be read as limited observed evidence. High-Acc Low-Struct also occurs in several settings, including 15 of 100 questions for GPT-5-mini on Spider-Syn.

The explicit multiple-structure counts broaden the setting in which disagreement has been observed, but their size limits what can be inferred about prevalence. Each question changes a reported proportion by 0.01. Dataset shift, perturbation design, and the available correct subset also affect the interpretation. We do not infer stable model rankings, precise population rates, or full benchmark generalization from these subsets.

4.4 Experiment 3: A Full-Pipeline Case Study

We compare Direct SQL, DIN-SQL, and compile-style generation on the Spider development set, with ten outputs per question using GPT-5-mini and the same decoding configuration. Direct SQL provides the direct-generation reference. DIN-SQL supplies a decomposed-prompting baseline (Pourreza and Rafiei, 2023). The compile-style setting emits structured JSON and uses deterministic compilation to obtain SQL.

Table 4 reports execution accuracy 0.785 for the compile-style pipeline, compared with 0.742 for Direct SQL and 0.736 for DIN-SQL. Structural agreement among execution-correct outputs rises from 0.552 to 0.632 relative to Direct SQL. These are end-to-end improvements in the evaluated setting, with no component-level attribution or statistical significance claim.

Compile-style generation also has a larger all-generation distinct count, 2.527 compared with 1.908 for Direct SQL. The two trends need not conflict: diversity across all generations and concentration within the correct subset refer to different distributions. Higher correct-subset agreement

Subset	Model	Exec Acc	AST Sim (corr)	Exec-Corr Struct-Diff	High-Acc Low-Struct
BIRD Mini-Dev	GPT-5-mini	0.145	0.091	0.090 (9/100)	0.110 (11/100)
	Claude-4.5-Sonnet	0.214	0.210	0.060 (6/100)	0.040 (4/100)
	DeepSeek-V3.1	0.246	0.212	0.170 (17/100)	0.090 (9/100)
Spider-Syn	GPT-5-mini	0.229	0.186	0.070 (7/100)	0.150 (15/100)
	Claude-4.5-Sonnet	0.332	0.410	0.010 (1/100)	0.000 (0/100)
	DeepSeek-V3.1	0.355	0.300	0.220 (22/100)	0.100 (10/100)
Dr.Spider	GPT-5-mini	0.122	0.121	0.030 (3/100)	0.070 (7/100)
	Claude-4.5-Sonnet	0.244	0.276	0.030 (3/100)	0.010 (1/100)
	DeepSeek-V3.1	0.266	0.287	0.110 (11/100)	0.060 (6/100)

Table 3: Scope validation on 100 questions per subset and ten generations per question. Parentheses give counts, not confidence intervals. The indicators follow the heuristic definitions in Section 3.3. These subsets document observed disagreement rather than full-benchmark performance or model rankings. Appendix B retains the additional structural statistics.

Measure	Direct SQL	DIN-SQL	Compile-style
Execution Accuracy	0.742	0.736	**0.785**
AST Similarity (correct)	0.552	0.579	**0.632**
Distinct Structures (all)	1.908	1.553	2.527
JSON Valid Rate	–	–	0.998
Compilable Rate	–	–	0.971
SQL Parse Rate	–	–	0.902
End-to-End Success	–	–	0.959

Table 4: Full-pipeline comparison on Spider using GPT-5-mini. Structural agreement concerns execution-correct outputs; diversity concerns all generations. Intermediate validity statistics describe the reported pipeline stages. The comparison does not attribute gains separately to the JSON representation, compilation, or model reasoning.

Model	AST Sim (para)↑	Distinct	Sensitivity↓	Sensitive Frac↓
GPT-5-mini	0.328	13.220	0.622	0.900
GPT-4.1-mini	0.493	6.640	0.462	0.770
DeepSeek-V3.1	0.655	3.330	0.280	0.515
Claude-4.5-Sonnet	0.755	2.190	0.179	0.365
Claude-4.5-Opus	0.755	2.190	0.179	0.365
Gemini-2.5-Flash	0.787	2.870	0.158	0.395
Gemini-3-Pro	**0.894**	1.530	**0.079**	**0.195**

Table 5: Structural robustness of SQL generation under paraphrase perturbations on the Spider development set. We measure cross-paraphrase structural consistency (AST Sim), diversity (Distinct), and sensitivity to input variations (Sensitivity, Sensitive Frac).

can coexist with a broader collection of structures overall. Neither quantity alone shows that every generated query is simpler, more efficient, or more robust to database changes.

The validity statistics report different stages of the structured pipeline and should not be substituted for execution correctness. A JSON object can be well formed while describing an incorrect query; a compiled query can execute while returning the wrong answer. The supplementary examples illustrate both corrected queries and failures involving omitted joins or predicates. The purpose of this experiment is to assess a practical intervention under a shared setting, not to establish superiority over current benchmark systems. Broader comparisons and targeted ablations remain future work.

4.5 Experiment 4: Input Perturbations

We next examine sensitivity to semantically invariant input changes. The experiment uses 200 Spider development questions, with question paraphrases and schema-presentation variants. Candidate paraphrases were generated with an LLM prompt and manually checked to preserve requested attributes, aggregation, filters, comparisons, ordering, limits, and schema meaning. Appendix E.3 gives the prompt and exclusion criteria. Schema variants reorder tables or columns while preserving the database entities and foreign keys.

For each variant, repeated generation produces a structural distribution and its majority structure. The analysis then compares these majorities across variants. This distinguishes within-input sampling variation from observed differences across input presentations, although it does not eliminate the sampling uncertainty in the estimated majorities.

Question paraphrases. Table 5 reports cross-paraphrase agreement, pooled structural diversity, perturbation sensitivity, and the fraction of questions with a changed majority structure. GPT-5-mini has agreement 0.328 and sensitive fraction 0.900. Gemini-3-Pro has agreement 0.894 and sensitive fraction 0.195. Thus, even the highest observed agreement in this table does not imply invariance on every evaluated question.

The distinct count pools structures across variants, so it can exceed the ten generations collected for a single input presentation. This explains why

Model	AST Sim (schema)↑	Distinct	Sensitivity↓	Sensitive Frac↓
GPT-5-mini	0.490	7.330	0.500	0.640
GPT-4.1-mini	0.727	3.600	0.263	0.360
DeepSeek-V3.1	0.808	1.870	0.170	0.260
Gemini-2.5-Flash	0.895	1.790	0.098	0.145
Claude-4.5-Sonnet	0.955	1.130	0.043	0.065
Claude-4.5-Opus	0.955	1.130	0.043	0.065
Gemini-3-Pro	**0.957**	1.060	**0.043**	**0.065**

Table 6: Structural robustness of SQL generation under schema presentation perturbations on the Spider development set. We evaluate cross-variant structural consistency (AST Sim), diversity (Distinct), and sensitivity to schema changes (Sensitivity, Sensitive Frac).

the GPT-5-mini value is 13.220, rather than a violation of the per-input sampling budget. Sensitivity and sensitive fraction also have different denominators: the former averages changed variants, whereas the latter records whether a question has any observed majority change. A question with one changed variant can therefore contribute to the sensitive fraction without having high sensitivity.

Schema presentation. We hold the question fixed and vary only the presentation order of the schema. Table 6 reports higher cross-variant agreement than the paraphrase table for every evaluated model. GPT-5-mini has agreement 0.490 and sensitive fraction 0.640, while Gemini-3-Pro has agreement 0.957 and sensitive fraction 0.065. These comparisons describe the evaluated perturbation sets rather than a universal ordering of perturbation difficulty.

Both perturbation types can coincide with changes in the dominant generated structure. The observation supports analyzing the mapping from a fixed intended question to its generated program under different presentations. A changed majority is a structural sensitivity signal, not a direct measurement of internal reasoning or proof that the resulting query is harmful. The available experiments do not quantify how often these changes alter answers on new database instances.

5 Discussion

5.1 When Structural Differences Matter

The distinction between structure and semantics is central to interpreting SQLStructEval. Canonicalization removes selected superficial differences, but it does not prove that the remaining differences are consequential. Alternative formulations may be interchangeable for a particular application. Accordingly, distinct counts and entropy have no universal preferred direction, and a high majority ratio does not imply that the majority query is correct.

The minimum-horsepower example in Appendix F illustrates an instance-dependent distinction. The reference uses `ORDER BY Horsepower ASC LIMIT 1`; the generated query instead compares horsepower with a `MIN` subquery and selects distinct model names. The example reports matching results on the evaluated database. If multiple distinct car models later share the non-null minimum horsepower, the ordered query returns one row while the predicate-based query can return multiple model names. This is a consequence of the displayed query semantics, not an additional experiment or an observed failure rate.

Conversely, the appendix compares `COUNT(*)` with `COUNT(Singer_ID)`. When the identifier is non-null for every row in the relevant group, both expressions count the same rows. Their structural difference can therefore be benign under that constraint. These examples motivate inspection of both query semantics and database assumptions before assigning a reliability interpretation to a structural disagreement.

5.2 Using the Measures Together

A practical reading starts with execution correctness, then examines the structures represented among correct outputs, and finally considers whether the dominant structure changes under input perturbations. This order avoids treating a consistent wrong answer as reliable or treating every valid alternative as a defect. The reference-alignment measure supplies additional context, but should not become a requirement that all accepted outputs reproduce the annotation.

The framework can identify candidates for further auditing: for example, one can inspect a question with frequent correct outputs but several distinct structures, or compare the dominant queries before and after a paraphrase. Whether these candidates differ in query cost, maintainability, policy compliance, or behavior after database changes requires the corresponding downstream checks. AST agreement is not a substitute for those measurements.

The compile-style results likewise favor joint reporting: higher agreement among correct outputs coexists with greater overall diversity. This describes the evaluated pipeline without attributing the improvement to intrinsic reasoning stability or predicting other downstream properties.

5.3 Coverage and Sampling in Structural Evaluation

A structural summary depends on which generations have usable representations. Parsing and execution are distinct operations: a query may execute in SQLite while failing the structural analysis pipeline, and a parsed query may fail execution or return an incorrect answer. The zero convention in our aggregation makes these distinctions important. In particular, an average AST Sim (corr) below one combines questions whose correct outputs disagree with questions for which the correct structural subset is empty. It should not be interpreted as agreement conditioned on every question having at least one parsed correct output.

This issue is especially relevant to the additional dataset subsets, whose parse rates are reported in Appendix B. The same numerical concentration can arise from different combinations of coverage and within-question agreement. A conditional analysis restricted to nonempty correct subsets would answer a different question and require a separately reported aggregation. That alternative quantity should not be conflated with the reported average. Counts that explicitly require more than one parsed correct structure provide a more direct indication of observed diversity.

Finite sampling imposes another boundary. Ten generations provide an empirical view of the model's output distribution, not an exhaustive account of all structures it can generate. A structure absent from a sample is not necessarily impossible, and a dominant structure can change between independent samples. Likewise, the majority selected for a paraphrase is itself estimated from sampled outputs. The perturbation results establish observed structural changes under the specified protocol, but do not isolate all variation due to input changes from the uncertainty of estimating a majority.

For these reasons, the measures are most informative as a profile tied to a dataset, sampling budget, representation, and aggregation convention. Comparing such profiles requires matching those conditions. Reporting the observed counts alongside the scope-validation proportions makes one aspect of this dependence visible; independent repeats and explicit coverage-conditioned estimates remain useful extensions. None of these qualifications changes the distinction between an execution match and the structure of the query that produced it.

The relationship with semantic evaluation is complementary. Test-suite evaluation compares queries across database instances (Zhong et al., 2020); structural analysis compares query constructions. A structural disagreement can suggest a semantic test, such as introducing tied minima in the example above, without establishing its frequency or severity. Equal canonical structures likewise do not guarantee a correct interpretation of the question.

Useful diversity remains possible: a downstream selector may benefit from several valid formulations. Reducing distinct structures is then not necessarily the objective. A consistency measure describes concentration, while an application-specific assessment determines which alternatives satisfy its requirements. The distinction separates an observation about generation from a decision about deployment.

6 Conclusion and Future Work

We introduced SQLStructEval to characterize the structural behavior of LLM-generated SQL alongside execution accuracy. Across repeated generations on Spider, we observed structural disagreement within execution-correct outputs; additional subset evaluations showed that this observation was not confined to the original Spider development set. Perturbation experiments further documented changes in dominant structures under question paraphrases and schema presentation changes.

Our compile-style case study improved execution accuracy and structural agreement among correct outputs in the evaluated setting. These results concerned the complete generation-and-compilation pipeline and did not isolate improved model reasoning. More broadly, our analysis showed why structural diversity, concentration, reference alignment, and execution correctness need separate interpretations. The framework exposes behavior for inspection without treating every structural difference as an error.

Future work should connect these diagnostics with measured outcomes under changed databases and execution environments, estimate uncertainty through independent sampling batches, and isolate the effects of prompting, structured representations, and compilation. Broader datasets and generation methods will also help establish where the observed patterns persist and when structural diagnostics are useful for deployment decisions.

Limitations

Our full-scale experiments use Spider, a controlled Text-to-SQL benchmark. The additional BIRD Mini-Dev, Spider-Syn, and Dr.Spider results use 100-example subsets and three models. They validate the presence of observed structural disagreement in additional settings, but do not establish precise population rates or comprehensive cross-benchmark generalization. Public benchmark exposure during model pretraining may also influence both execution and structural behavior; the present study does not isolate that effect.

The experiments report one sampled collection per input, rather than repeated independent collections with confidence intervals. Small differences between models should not be interpreted as statistically established rankings. The diagnostic thresholds are heuristic, and no threshold-sensitivity analysis is reported. Manual paraphrase checking is described through its acceptance criteria; we do not report an agreement statistic or a checker count for that procedure.

Canonical representations capture selected syntactic distinctions and retain query arguments. They do not establish full semantic equivalence or predict execution cost, query-plan stability, maintainability, or failures after database changes. Parse failures affect structural coverage, zero-filled averages, and the low-agreement diagnostic; that diagnostic is not a pure count of structurally different correct queries. Matching one annotated execution result can also overlook semantic errors that would appear on another database instance. The reference query serves as a diagnostic comparison, not a unique correct structure.

Finally, the compile-style case study jointly changes the output representation and introduces deterministic compilation. It does not separate their contributions from prompting or model reasoning, and its comparison is limited to Spider and the included baselines. Targeted ablations and downstream validation remain necessary before making stronger causal or deployment claims.

Ethical Considerations

We study publicly available Text-to-SQL benchmarks in an offline setting. The work does not deploy generated queries against operational databases or involve interaction with research participants. Incorrect queries can nevertheless cause misleading decisions if used in practice. Structural diagnostics should accompany appropriate execution and semantic checks rather than serve as a guarantee of correctness or safety.

Data and artifact use. We use Spider, BIRD Mini-Dev, Spider-Syn, and Dr.Spider for research evaluation under their respective access and licensing terms. Derived question variants and generated queries remain associated with those source resources. The code link provides the evaluation implementation; use and redistribution of datasets and model outputs must respect the original resource and provider terms. Hosted model updates can affect reproducibility even when the prompt remains fixed.

AI assistance. AI assistants supported language polishing and camera-ready editorial checks. No new model-generation experiments were added during this revision.

References

Mohammad Albinhassan, Pranava Madhyastha, Mark Law, and Alessandra Russo. 2025. Learning and enforcing context-sensitive control for LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, ACL-SRW '25, pages 834–842, Vienna, Austria. Association for Computational Linguistics.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv preprint*, arXiv:2108.07732.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.Spider: A diagnostic evaluation benchmark towards text-to-SQL robustness. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, Kigali, Rwanda. OpenReview.net.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *arXiv preprint*, arXiv:2107.03374.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, COLING '22, pages 2166–2187, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, ACL-IJCNLP '21, pages 2505–2515, Online. Association for Computational Linguistics.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, EMNLP '23, pages 10932–10952, Singapore. Association for Computational Linguistics.

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*, ACL '19, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *Proceedings of the Twelfth International Conference on Learning Representations*, ICLR '24, Vienna, Austria. OpenReview.net.

Heegyu Kim, Taeyang Jeon, Seunghwan Choi, Seungtaek Choi, and Hyunsouk Cho. 2025. FLEX: Expert-level false-less EXecution metric for reliable text-to-SQL benchmark. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, NAACL '25, pages 4448–4475, Albuquerque, New Mexico. Association for Computational Linguistics.

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 *Advances in Neural Information Processing Systems*, volume 36 of *NeurIPS '23*, pages 42330–42357, New Orleans, LA, USA. Curran Associates, Inc.

Yiyang Li, Yonghuang Wu, Ying Luo, Liangtai Sun, Zishu Qin, Lin Qiu, Xuezhi Cao, and Xunliang Cai. 2025. Instance-level randomization: Toward more stable LLM evaluations. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Findings-EMNLP '25, pages 3411–3425, Suzhou, China. Association for Computational Linguistics.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36 of *NeurIPS '23*, pages 21558–21572, New Orleans, LA, USA. Curran Associates, Inc.

Franklin Ma and Alan J. Hu. 2025. Logically constrained decoding. In *Proceedings of the 3rd Workshop on Mathematical Natural Language Processing*, MathNLP '25, pages 150–167, Suzhou, China. Association for Computational Linguistics.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida Wang, and Xi Victoria Lin. 2023. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128, Honolulu, HI, USA. PMLR.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. In *Proceedings of the Tenth International Conference on Learning Representations*, ICLR '22, Online. OpenReview.net.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36 of *NeurIPS '23*, pages 36339–36348, New Orleans, LA, USA. Curran Associates, Inc.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the Text-to-SQL capabilities of large language models. *arXiv preprint*, arXiv:2204.00498.

Prateek Rajput, Abdoul Aziz Bonkoungou, Yewei Song, Abdoul Kader Kaboré, Iyiola E. Olatunji, Jacques Klein, and Tegawendé F. Bissyandé. 2025. Dynamic stability of LLM-Generated code. *arXiv preprint*, arXiv:2511.07463.

Federico Raspanti, Tanir Ozcelebi, and Mike Holenderski. 2025. Grammar-constrained decoding makes large language models better logical parsers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, ACL-Industry '25, pages 485–499, Vienna, Austria. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yewei Song, Tiezhu Sun, Xunzhu Tang, Prateek Kumar Rajput, Tegawendé F. Bissyandé, and Jacques Klein. 2025. Measuring LLM code generation stability via structural entropy. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering*, ASE '25, pages 3922–3926, Seoul, Republic of Korea. IEEE.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, Kigali, Rwanda. OpenReview.net.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, EMNLP '18, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. 2024. How language model hallucinations can snowball. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 59670–59684, Vienna, Austria. PMLR.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 396–411, Online. Association for Computational Linguistics.

Jiace Zhu, Yuanzhe Huang, Yingtao Shen, Jie Zhao, and An Zou. 2025. Path-Consistency with prefix enhancement for efficient inference in LLMs. *arXiv preprint*, arXiv:2409.01281.

A Structural Measure Details

Parse-failure conventions. Our evaluation implementation assigns zero to counts, majority ratios, and entropies when the selected structural subset is empty. Dataset summaries average these values over all questions. This differs from conditioning an average on nonempty subsets. The denominator therefore includes questions with no usable structure. An average below one can consequently reflect structural coverage as well as concentration among available structures.

Table-column mapping. Distinct and Distinct (all) count canonical structures over all parsed generations; Distinct (corr) restricts this count to parsed execution-correct generations. Majority denotes the majority-structure ratio, Norm. entropy in Table 1 denotes entropy divided by $\log_2 K$ when $K > 1$, with zero otherwise; Entropy (corr) in Table 7 is unnormalized entropy, and Ref Align denotes annotated-reference alignment. AST Sim (corr) denotes majority concentration within the correct subset; AST Sim (para) and AST Sim (schema) denote agreement across input variants' majority structures. Sensitivity is the fraction of changed variants relative to the original input; Sensitive Frac is the fraction of questions with a majority change.

Thresholded diagnostics. Exec-Corr Struct-Diff uses at least two execution-correct outputs and majority concentration below 0.7, recording missing correct-subset agreement as zero. It can therefore include cases without a parsed correct structure and must be interpreted together with structural coverage. High-Acc Low-Struct uses sample execution accuracy at least 0.8 and multiple execution-correct structures. The two indicators can overlap, but neither is a substitute for execution accuracy or a count of proven semantic failures. The thresholds are heuristic and no sensitivity sweep is reported.

B Additional Scope-Validation Statistics

Table 7 provides further statistics for the 100-question subset evaluations. The main-text scope table shows execution accuracy and diagnostic counts. Majority (corr) and AST Sim (corr) denote the same concentration measure and are displayed once here.

C Additional Error Analysis

To better understand the behavioral differences between traditional direct SQL generation and the proposed compile-style generation paradigm, we conduct an additional qualitative error analysis on representative examples from the Spider development set.

C.1 Error Taxonomy

Based on manual inspection of generated SQL queries, we categorize errors into six major types:

- **A. Schema Linking Errors.** Incorrect or missing references to tables, columns, or foreign-key relationships between tables.
- **B. Logical Form Errors.** Errors in high-level query logic such as aggregation, grouping, nested queries, or set operations.
- **C. Join Path Errors.** Incorrect or incomplete join paths between tables, including missing joins or incorrect join conditions.
- **D. Predicate / Constraint Errors.** Incorrect or missing filtering conditions in `WHERE` or `HAVING` clauses, such as incorrect comparison operators or incorrect attribute references.
- **E. Structural Variance but Executable.** Queries that produce correct execution results but differ substantially in structure from the gold query or other correct solutions.
- **F. Intermediate Representation / Compilation Errors** Errors introduced by the structured intermediate representation or the compilation process in the compile-style generation pipeline.

C.2 Error Distribution

Table 8 shows the distribution of error types observed in baseline and compile-style generation. Each ratio represents the fraction of inspected examples assigned to the corresponding primary category. We inspected 100 examples for each setting, sampled from cases with execution errors or structural disagreement. One author annotated the examples and a second author checked the labels. We assigned mutually exclusive primary categories, choosing the earliest issue that determined failure or structural divergence when multiple issues occurred.

Subset	Model	Parse Rate	Distinct (all)	Distinct (corr)	Majority (corr)	Entropy (corr)
BIRD Mini-Dev	GPT-5-mini	0.657	3.380	0.560	0.091	0.200
	Claude-4.5-Sonnet	0.496	1.060	0.290	0.210	0.041
	DeepSeek-V3.1	0.399	1.600	0.500	0.212	0.152
Spider-Syn	GPT-5-mini	0.713	2.630	0.540	0.186	0.177
	Claude-4.5-Sonnet	0.686	1.200	0.410	0.410	0.000
	DeepSeek-V3.1	0.453	1.260	0.530	0.300	0.126
Dr.Spider	GPT-5-mini	0.668	2.610	0.300	0.121	0.092
	Claude-4.5-Sonnet	0.720	1.460	0.290	0.276	0.010
	DeepSeek-V3.1	0.470	1.490	0.420	0.287	0.072

Table 7: Additional scope-validation statistics. Each row summarizes 100 questions with ten sampled generations per question; these are descriptive subset results.

Error Type	Baseline	Compile-style	Δ
A. Schema linking	0.20	0.13	-0.07
B. Logical form	0.18	0.15	-0.03
C. Join path	0.14	0.09	-0.05
D. Predicate / constraint	0.16	0.10	-0.06
E. Structural variance (exec OK)	0.28	0.40	+0.12
F. Intermediate representation / compile failure	0.04	0.13	+0.09

Table 8: Error type statistics for baseline and compile-style generation on the Spider development set. Each value represents the fraction of the 100 inspected examples assigned to the corresponding primary category. This selected sample is not an estimate of error prevalence over the full benchmark. **Baseline** refers to direct SQL generation, while **Compile-style** refers to generation using structured intermediate representations followed by deterministic compilation into SQL. Δ denotes the difference between the two settings (compile-style minus baseline). Positive values indicate a larger proportion within the selected compile-style sample.

As shown in Table 8, the selected compile-style sample has smaller proportions of several semantic-error categories, including schema linking, logical form mistakes, incorrect join paths, and predicate errors. However, two categories increase: structurally diverse but executable queries (Type E) and intermediate representation or compilation-related failures (Type F). This pattern is consistent with the quantitative observations reported in Experiment 3.

C.3 Representative Case Studies

We further illustrate typical failure modes using representative examples from the Spider development set.

Case Group A: Baseline Execution Failures. The inspected examples include schema-linking and join-path errors in direct generation. An explicit intermediate representation can express the required table relations, although it does not guarantee that the model will select them correctly.

For example, for the question:

> "Which model of the car has the minimum horsepower?"

the gold query joins `CAR_NAMES` and `CARS_DATA` to retrieve the corresponding model. An inspected baseline output omits the join; such queries directly access the `CARS_DATA` table, resulting in incorrect results. In contrast, compile-style generation constructs the join structure explicitly in the intermediate representation, producing SQL queries that are structurally equivalent to the gold query.

Case Group B: Execution-Correct but Structurally Divergent Queries. We also observe cases where multiple generated queries produce identical execution results but differ substantially in their syntactic structure. For example, for the question:

> "Show all countries and the number of singers in each country."

the inspected baseline outputs include structurally distinct SQL queries, including variants with redundant subqueries or unnecessary `DISTINCT` operators. Although these queries produce identical results, they correspond to different AST structures and therefore increase structural entropy. The compile-style example uses a simple `GROUP BY` aggregation.

Case Group C: Paraphrase Sensitivity. Semantically equivalent questions can trigger different query structures in baseline generation. For example, the two questions

> "Show all countries and the number of singers in each country"
> "How many singers are from each country?"

share the same semantic intent and annotated reference SQL query. However, the inspected baseline generations use different structural patterns for

the two inputs, including variations involving subqueries or derived tables. This example illustrates why agreement across paraphrases is evaluated separately from agreement across repeated generations of one input.

Case Group D: Compile-style Failure Cases. Despite the overall improvements, compile-style generation introduces new failure modes related to the intermediate representation and compilation process. For instance, when answering the question

> "How many car models are produced in the USA?"

the correct query requires joining `MODEL_LIST`, `CAR_MAKERS`, and `COUNTRIES`. In some compile-style generations, the intermediate representation omits the final join to `COUNTRIES` and instead applies the filtering condition directly to the `CAR_MAKERS` table. This leads to execution errors despite the structured generation pipeline.

These cases illustrate both successful query construction and failures of the intermediate representation. They motivate further inspection of representation design and compilation, but do not isolate a change in semantic reasoning or estimate error prevalence.

D Implementation Details

This section describes the implementation details of SQLStructEval, including SQL parsing and canonicalization, canonical AST representation, execution evaluation, and generation settings.

D.1 SQL Parsing and Canonicalization

We parse all generated SQL queries using the `sqlglot` parser, which provides robust SQL parsing and AST construction across multiple SQL dialects. In our experiments we use the SQLite dialect to ensure compatibility with the Spider benchmark databases.

To remove surface-level differences that are irrelevant to program structure, we apply both text-level normalization and AST-level canonicalization.

Text-level normalization. After rendering the parsed AST back into SQL text, we perform lightweight normalization including: (1) removing trailing semicolons and leading/trailing whitespace, (2) replacing line breaks with spaces and collapsing redundant whitespace, (3) removing simple column aliases (e.g., `SELECT col AS x`), and (4) converting the SQL string to lowercase.

AST-level canonicalization. To ensure that structurally equivalent queries share the same representation, we further canonicalize the AST structure:

- **Alias normalization.** Table aliases are renamed according to their first appearance order (e.g., `t1, t2, ...`). All column references are updated accordingly so that structurally identical joins remain equivalent under different alias choices.
- **Logical operator normalization.** Nested `AND` predicates are flattened into condition lists, sorted by their SQL string representation, and reconstructed into canonical AST nodes. This ensures that logically equivalent expressions such as `A AND B` and `B AND A` share the same canonical form.

These normalization steps are shared across all experiments (Exp1–Exp4) and applied to both baseline and compile-style generated queries.

D.2 Canonical AST Representation

For each SQL query we obtain a canonical representation through the following pipeline:

1. Parse the SQL query into an AST using `sqlglot`;
2. Apply AST-level canonicalization (alias normalization and logical operator normalization);
3. Render the canonical AST back into SQL;
4. Apply text-level normalization.

The resulting canonical SQL string serves as the canonical comparison representation and is used as the basis for all structural measures in our analysis.

For each question with k generated SQL queries, we compute the frequency distribution over canonical structures. Structural statistics are then derived from this distribution:

- **Distinct structures:** the number of unique canonical SQL representations.
- **Majority structure ratio:** the relative frequency of the most common structure among the k generations.
- **Structural entropy:** the Shannon entropy of the structure distribution, measuring structural diversity.

To compare generated structures with annotated reference queries, the annotated reference SQL is canonicalized using the same pipeline, and structural matches are determined by canonical representation equality.

D.3 Execution Environment

The main experiments use the Spider SQLite database files; the additional scope checks use the corresponding benchmark databases. We compare query execution results against the annotated reference result on the same database instance.

For each generated SQL query, we execute the query against the corresponding SQLite database and obtain the result set. A generated query is considered execution-correct if its execution result exactly matches the result produced by the annotated reference SQL query. Queries that produce execution errors or mismatched results are considered incorrect.

D.4 Generation Settings

For each question we generate multiple candidate SQL queries from each model to analyze structural variation.

Sampling configuration. We sample $k = 10$ SQL queries for every $(question, schema)$ pair. For APIs that cannot return multiple samples in a single request, we perform multiple independent calls and aggregate the generated outputs.

Decoding parameters. Unless restricted by the model API, we use stochastic decoding with temperature $= 1.0$ and default nucleus sampling parameters. Beam search or greedy decoding is not used, as our goal is to analyze structural variation under realistic sampling conditions.

Compile-style generation. In Experiment 3, the compile-style setting requires models to generate a structured intermediate representation instead of directly producing SQL text. The structured representation is expressed as JSON and contains fields such as `select`, `from`, `joins`, `where`, `group_by`, `having`, `order_by`, and `limit`. A deterministic compiler converts the JSON representation into executable SQL. The compiled SQL is then evaluated using the same canonicalization and execution evaluation pipeline as the baseline generation.

E Prompt and Generation Settings

This section provides the prompt templates used in our experiments.

E.1 Prompt for Direct SQL Generation

In the baseline setting, models are prompted to directly generate a SQL query given a natural language question and the corresponding database schema.

```
You are an expert SQL generator.
Use SQLite dialect.
Only output ONE SQL query, no explanation.

Database ID: <db_id>

Database schema (JSON):
<schema_json>

Question:
<natural_language_question>

SQL:
```

E.2 Prompt for Compile-style JSON Generation

In the compile-style setting, the model is instructed to generate a structured JSON representation of the SQL query rather than directly producing SQL text. The prompt used for this setting is shown below.

```
You are an expert Text-to-SQL system for the
    Spider benchmark.
Your task is to write a structured JSON
    representation of a SQL query
for the given question and database schema.

Requirements:

- Use ONLY tables and columns that exist in the
    provided schema.
- Assume the database uses the SQLite dialect.
- You MUST output a single JSON object, and
    nothing else (no explanations).
- The JSON must describe the logical structure
    of the SQL query with the following fields:
  - type: "query"
  - query: {
    select: [ ... ],
    from: { ... },
    joins: [ ... ],
    where: [ ... ],
    group_by: [ ... ],
    having: [ ... ],
    order_by: [ ... ],
    limit: ...,
    distinct: ...
  }
- Do NOT include any natural language text in
    the JSON.

Database ID: {db_id}

Database schema (JSON):
{schema_json}

Question:
{question}

Now output ONLY the JSON object for the query
    structure:
```

E.3 Paraphrase Generation and Checking

Candidate paraphrases were generated using an LLM prompt and manually checked before use. The submitted prompt is:

```
Rewrite the following Text-to-SQL
    question into semantically
    equivalent paraphrases. Preserve the
    requested attributes, aggregation,
    filters, comparison operators,
    ordering, limits, and all schema-
    specific meanings.
Do not add or remove constraints. Do not
    make the question more ambiguous.

Original question:
<question>

Output <n> paraphrases, one per line.
```

We excluded paraphrases that changed selected columns, aggregation targets, predicates, comparison relations, ordering, limits, or intended schema entities. We also excluded variants that added conditions, omitted necessary constraints, or introduced ambiguity. These criteria describe the intended semantic-preservation check; no inter-annotator agreement statistic is reported for this procedure.

F Additional Structural Examples

To complement the quantitative structural measures reported in the main experiments, we present several representative examples illustrating common forms of structural variation in LLM-generated SQL queries.

Example 1: Execution-correct but structurally different queries **Question.** Which model of the car has the minimum horsepower? (`db_id = car_1`)

Gold SQL.

```
SELECT T1.Model
FROM CAR_NAMES AS T1
JOIN CARS_DATA AS T2 ON T1.MakeId = T2.Id
ORDER BY T2.Horsepower ASC
LIMIT 1;
```

Generated SQL (baseline).

```
SELECT DISTINCT cn.Model
FROM cars_data cd
JOIN car_names cn ON cd.Id = cn.MakeId
WHERE cd.Horsepower = (
  SELECT MIN(Horsepower) FROM cars_data
);
```

Although both queries return identical execution results, their canonical AST structures differ substantially, as illustrated in Figure 2. The gold query computes the minimum value using an `ORDER BY + LIMIT` structure, while the generated query uses an aggregate subquery with `MIN(Horsepower)` and an equality predicate. This introduces a nested subquery node in the AST, resulting in a different structural representation.

Example 2: Structural sensitivity to paraphrase Consider the following pair of paraphrased questions from the Spider dataset (`db_id = concert_singer`):

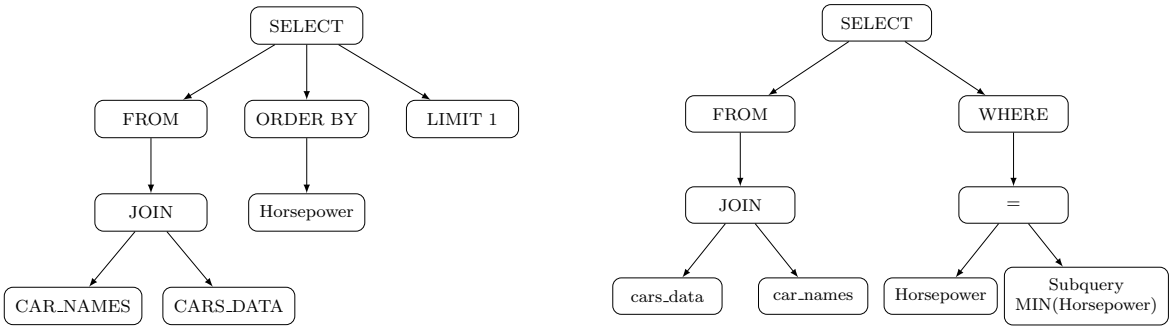
Question A. Show all countries and the number of singers in each country.

Question B. How many singers are from each country?

Both questions correspond to the same annotated reference SQL query:

```
SELECT country, COUNT(*)
FROM singer
GROUP BY country;
```

However, the baseline model produces slightly different structural variants:



(a) AST of annotated reference SQL query

(b) AST of generated SQL query

Figure 2: AST structures of two execution-equivalent SQL queries in Example 1 of Appendix F.

```
-- Variant A
SELECT Country, COUNT(Singer_ID)
FROM singer
GROUP BY Country;

-- Variant B
SELECT Country, COUNT(*)
FROM singer
GROUP BY Country;
```

Although both queries are execution-equivalent, their AST structures differ because the aggregation argument differs (`COUNT(*)` vs. `COUNT(Singer_ID)`). Across multiple generations, additional variations such as derived tables or `DISTINCT` operators may also appear. This example illustrates how small linguistic changes in the question can lead to different structural query plans.

Example 3: Compile-style correcting structural errors **Question.** What are the names and ids of all countries with at least one car maker? (`db_id = car_1`)

Gold SQL.

```
SELECT T1.CountryName, T1.CountryId
FROM COUNTRIES AS T1
JOIN CAR_MAKERS AS T2 ON T1.CountryId = T2.
    Country
GROUP BY T1.CountryId
HAVING COUNT(*) >= 1;
```

Baseline SQL (typical failure).

```
SELECT CountryName, CountryId
FROM COUNTRIES
JOIN CAR_MAKERS
ON COUNTRIES.CountryId = CAR_MAKERS.Country;
```

The illustrated baseline output omits the `GROUP BY` and `HAVING` clauses, producing incorrect query results.

Compile-style SQL.

```
SELECT COUNTRIES.CountryName, COUNTRIES.
    CountryId
FROM COUNTRIES
JOIN CAR_MAKERS
ON COUNTRIES.CountryId = CAR_MAKERS.Country
GROUP BY COUNTRIES.CountryId
HAVING COUNT(*) >= 1;
```

In the compile-style generation setting, the structured intermediate representation explicitly models aggregation components such as `group_by` and `having`. As a result, the compiled SQL consistently contains the correct aggregation structure and closely matches the gold AST.

G Example of the Compile-style Generation Pipeline

To illustrate the compile-style generation process used in Experiment 3, we present a concrete example showing the full pipeline from natural language to structured intermediate representation, internal AST, and final SQL query.

G.1 Input Natural Language Query

Question.

> Show the name and capacity of the stadium with the largest average attendance.

Database schema (simplified).

```
{
  "tables": [
    {"table_name": "stadium",
     "columns": ["Stadium_ID", "Name", "
    Location", "Capacity", "Average"]},
    {"table_name": "concert",
     "columns": ["concert_ID", "Stadium_ID", "
    Year"]}
  ],
  "foreign_keys": [
    {"source_table": "concert", "source_column":
     "Stadium_ID",
    "target_table": "stadium", "target_column":
     "Stadium_ID"}
  ]
}
```

G.2 LLM Output: Structured JSON Intermediate Representation

In compile-style generation, the model does not directly produce SQL text. Instead, it generates a structured JSON representation that encodes the program structure.

```
{
  "type": "query",
  "query": {
    "select": [
      {"expr": {"col": ["stadium", "Name"]}, "
    alias": null},
      {"expr": {"col": ["stadium", "Capacity"]},
     "alias": null}
    ],
    "from": {"table": "stadium", "alias": null},

    "joins": [],
    "where": [],
    "group_by": [],
    "having": [],
    "order_by": [
      {"expr": {"col": ["stadium", "Average"]},
     "direction": "desc"}
    ],
    "limit": 1,
    "distinct": false
  }
}
```

This JSON structure explicitly represents the logical components of the query, including `select`, `from`, `order_by`, and `limit`.

G.3 Intermediate AST Representation

The compiler then converts the JSON structure into an internal AST:

```
Select(
  columns = [
    Column(table="stadium", name="Name"),
    Column(table="stadium", name="Capacity")
  ],
  from = Table(name="stadium"),
  joins = [],
  where = [],
  group_by = [],
  having = [],
  order_by = [
    OrderBy(
      expr = Column(table="stadium", name="
    Average"),
      direction = DESC
    )
  ],
  limit = 1
)
```

This AST provides a canonical structural representation of the query, which can be analyzed and compared across different generations.

G.4 Compiled SQL Query

Finally, the AST is deterministically compiled into an executable SQL query:

```
SELECT stadium.Name, stadium.Capacity
FROM stadium
ORDER BY stadium.Average DESC
LIMIT 1;
```

This compiled SQL query is then used for execution evaluation and structural analysis in our experiments.