

CoreSemDB: Benchmarking Hybrid Semantic-Relational Query Processing over Text-Rich Databases

Yuchen Tian¹ Hao Zhang² Jing Ma^{1,†} Jianxiong Guo^{3,4†} Congli Gao⁵

¹Hong Kong Baptist University

²The Chinese University of Hong Kong

³Beijing Normal University

⁴Beijing Normal-Hong Kong Baptist University

⁵BeiHang University

Abstract

Real-world data workflows require hybrid reasoning that integrates high-precision structured retrieval with deep semantic understanding of unstructured text. Recently, several database vendors, including Google, Snowflake, and Databricks, have incorporated LLM-powered operators into their systems for querying unstructured data, yet benchmarks for evaluating their end-to-end performance under natural language inputs remain limited. We formalize this setting as NL2SemProg (Natural Language to Semantic Program Synthesis), which requires models to synthesize executable hybrid programs that interleave structured relational operators with LLM-powered semantic operators to resolve queries over text-rich databases. In this paper, we introduce CoreSemDB, a comprehensive benchmark built from 27 real-world text-rich databases. CoreSemDB is organized by an application-driven taxonomy spanning structural complexity across single and nested queries, along with core reasoning abilities including semantic inference and multi-table association. Our evaluation of 15 large language models on 330 queries shows that NL2SemProg remains challenging, with Gemini-3-Pro achieving only 23.64% accuracy. Further analysis reveals significant bottlenecks in nested compositional queries, trade-offs between precision and recall in multi-table contexts, and domain-specific strengths across different models. The code and data will be released at <https://github.com/yuchen814/CoreSemDB>.

1 Introduction

The advent of large language models (LLMs) has fundamentally transformed natural language interfaces for database management. Historically, two broad families of approaches have been developed for obtaining answers from databases. Database code-based methods, such as Text2SQL and SPARQL-based approaches, obtain answers by executing structured query programs over relational data with high precision, but are fundamentally confined to relational operations and cannot reason over unstructured text. QA-based methods, including RAG (Yang et al., 2024b) and TableQA (Wu et al., 2025), obtain answers by directly responding to natural language questions, but struggle in database scenarios where tables are both numerous and large, suffering from context-window constraints and an inability to enforce exact relational filters. As real-world databases increasingly combine structured records with massive volumes of unstructured text, neither family suffices. Both academia and industry are therefore rapidly converging on a new paradigm (Patel et al., 2024; Biswal et al., 2024): *hybrid semantic-relational query processing*, which deeply interleaves structured relational operators with LLM-powered semantic operators to handle both simultaneously.

[†] Corresponding authors. Correspondence to: Jing Ma <majing@hkbu.edu.hk>, Jianxiong Guo <jianxiongguo@bnu.edu.cn>

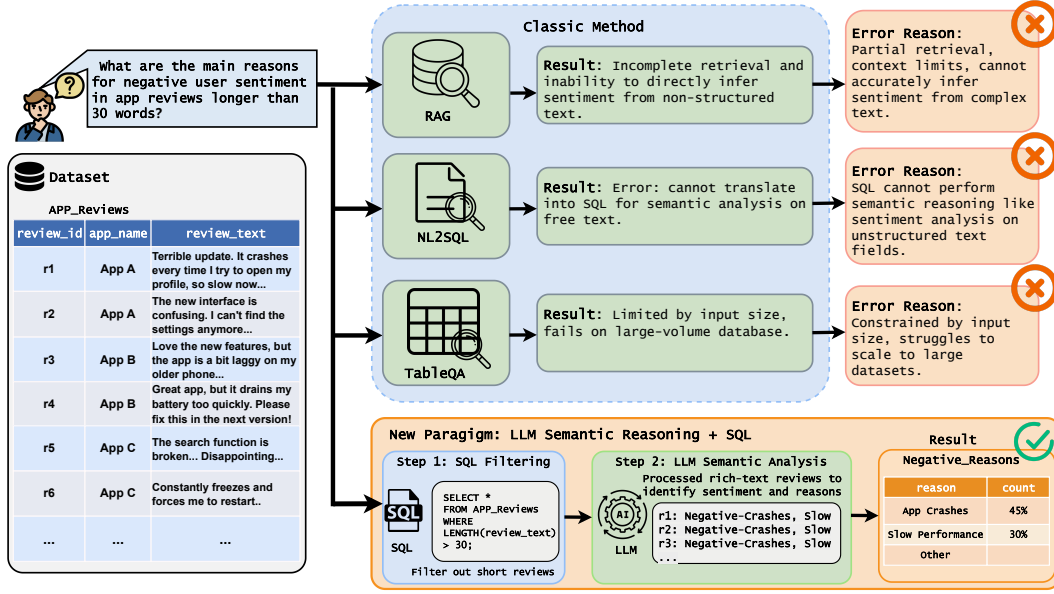


Figure 1: An example of the Hybrid Semantic-Relational Query Processing paradigm. To resolve complex analytical queries over text-rich databases, this new paradigm synthesizes an executable pipeline that deeply interleaves structured relational operators (e.g., SQL filtering for review length) with LLM-powered semantic operators (e.g., sentiment extraction on unstructured text), enabling precise and scalable reasoning over both structured fields and unstructured text.

As illustrated in Fig. 1, consider a practical analytical query over an application review database: “What are the main reasons for negative user sentiment in app reviews longer than 30 words?”. Answering this question requires deep semantic reasoning, such as inferring sentiment and aggregating reasons, tightly interleaved with exact relational constraints, such as filtering by string length. Such hybrid neuro-symbolic execution is beyond the reach of any single classic paradigm, yet LLMs demonstrate immense potential in synthesizing these complex data pipelines.

On the industry side, major database vendors have already begun integrating LLM-powered operators into their systems. Google’s BigQuery (Karmarkar & Wu, 2025) introduces an AI Query Engine to analyze unstructured data within SQL queries; Snowflake Cortex (Snowflake) leverages LLMs to answer free-form questions over unstructured data; and Databricks proposes AI Functions (Databricks) as built-in semantic operators. In academia, Patel et al. propose LOTUS (Patel et al., 2024), a unified framework that incorporates LLM-powered semantic operators, including semantic filtering, mapping, and aggregation, into relational data processing pipelines. As these LLM-enabled approaches proliferate, establishing systematic evaluation methodologies has become increasingly essential.

The community has made important initial strides toward this goal, and the evaluation landscape has evolved through two distinct stages. The first stage focused on **paradigm validation**. In this stage, TAGBench (Biswal et al., 2024) demonstrated the feasibility of the hybrid paradigm by showing that hand-crafted pipelines based on semantic operators (Patel et al., 2024) substantially outperformed Text2SQL and RAG baselines on 80 queries across 5 domains. This work shows the effectiveness of combining relational and semantic operators, showing that such integration leads to superior performance. The second stage focused on system-level benchmarking, with SemBench (Lao et al., 2025) evaluating four semantic query processing engines (i.e., LOTUS, Palimpsest, ThalamusDB, and BigQuery) on 55 multimodal queries in terms of cost–quality–latency tradeoffs. This work provided practical insights into how different system implementations affect processing overheads under the same LLM backbone.

However, as the paradigm evolves and system-level optimizations advance, the performance bottleneck increasingly lies in the *underlying LLM*, the core reasoning engine that powers every semantic operator. A critical question remains unanswered: **given a natural language query over a text-rich database, whether an LLM can synthesize executable hybrid programs and produce correct answers end-to-end?** Neither of the existing benchmarks is designed to evaluate this ability. Specifically, TAGBench employs a single LLM with hand-written pipelines, thereby bypassing the code synthesis challenge entirely and precluding cross-model comparison. SemBench fixes the LLM and varies the system, emphasizing processing efficiency rather than model-level reasoning capability. Both benchmarks contain fewer than 100 queries, limiting fine-grained diagnostic analysis. Moreover, neither benchmarks systematically evaluates *nested compositional queries* that require multi-stage reasoning over intermediate results, which remain a persistent challenge for current LLMs.

To bridge this gap, we present CoreSemDB (Complex Relational and Executable Semantics over DataBases), a comprehensive benchmark that advances the evaluation frontier to the third stage: **model-level diagnostic assessment**. Unlike prior work, CoreSemDB starts from natural language queries that combine relational and semantic operators, and produce grounded answers over real-world databases. To construct CoreSemDB, we leverage an agent-based framework for query generation, followed by rigorous two-phase human expert annotation and verification. The resulting benchmark comprises 330 high-quality samples spanning 27 real-world text-rich databases across diverse domains. To enable fine-grained capability diagnosis, we organize queries along an application-driven taxonomy that distinguishes structural complexity (*single* vs. *nested*) and isolates three core reasoning factors for single queries: *semantic reasoning*, *world knowledge*, and *multi-table linking*.

We further introduce task-specific evaluation metrics calibrated against human expert judgments. Specifically, we propose Quasi-Exact Match (QEM) to measure approximate structural alignment between ground-truth and predicted answers, and Semantic Alignment Match (SAM) to assess semantic consistency between expert-annotated and model-generated code. Both metrics are validated via Cohen’s Kappa analysis, ensuring reliable and reproducible evaluation. Based on these metrics, we conduct extensive experiments on 15 representative LLMs and uncover several key findings:

- Nested compositional queries remain a major challenge, with all models showing substantial performance drops compared to single queries, suggesting that multi-stage hybrid reasoning is a key bottleneck.
- Across both proprietary and open-source models, multi-table linking is consistently the weakest capability dimension, suggesting a shared difficulty in integrating cross-table information under semantic constraints.
- A clear scaling trend is observed: larger models consistently outperform smaller ones. However, even the best-performing model (Gemini-3-Pro) achieves only 23.64% accuracy, leaving substantial room for improvement.
- In Natural Language to Semantic Program Synthesis, open-source models now perform much closer to proprietary models. The top open-source model (DeepSeek-V3.2, 20.30%) now trails the best proprietary model by only ~ 3 percentage points.

2 Related Work

We first summarize methods for querying over data, then introduce LLM-powered systems for semantic data processing, and finally discuss existing benchmarks and position CoreSemDB within this landscape.

Natural Language Interfaces for Data Querying. Text2SQL methods (Zhong et al., 2017; Yu et al., 2018; Li et al., 2023; Pourreza et al., 2024) translate natural language queries into SQL code to retrieve data from databases. RAG-based methods (Lewis et al., 2020; Yang et al., 2024b; Chen et al., 2024) leverage similarity matching to find semantically relevant data from vector databases. TableQA methods (Zhu et al., 2021; Hegselmann et al., 2023; Wu et al., 2025) are generally effective for summarization-oriented queries by feeding tables

as input to LLMs. While each paradigm addresses a specific subset of user needs, none supports the hybrid semantic-relational reasoning required by NL2SemProg.

Data+AI Systems for Semantic Understanding. A growing ecosystem of systems bridges structured and unstructured data analysis using LLMs. Industrial platforms (Karmarkar & Wu, 2025; Snowflake; Databricks) and academic systems such as LOTUS (Patel et al., 2024) and Palimpzest (Liu et al., 2025b) provide the execution backbone for hybrid semantic-relational queries, but their evaluation has remained fragmented.

Existing Benchmarks. TAGBench (Biswal et al., 2024) validated the hybrid paradigm by evaluating hand-written LOTUS pipelines, but uses a single LLM with manually crafted programs over only 80 queries, bypassing code synthesis entirely. SemBench (Lao et al., 2025) benchmarks semantic query engines on cost–quality–latency tradeoffs, but fixes the LLM backbone and accepts pre-defined semantic SQL rather than natural language input. Both benchmarks thus circumvent *program synthesis*—the critical step where an LLM must interpret a natural language query and compose a fully executable hybrid program. CoreSemDB is, to our knowledge, the first benchmark specifically designed to evaluate this end-to-end synthesis capability across multiple LLMs over text-rich databases.

3 Task Definition

We formalize **NL2SemProg** (Natural Language to Semantic Program Synthesis) as follows. Let $D = \{R_1, \dots, R_n\}$ be a text-rich database whose attributes are typed over $\mathcal{T}_{\text{STR}} \cup \mathcal{T}_{\text{TXT}}$ (structured and unstructured, respectively). We define a hybrid operator vocabulary $\Omega = \Omega_{\text{rel}} \cup \Omega_{\text{sem}}$, where Ω_{rel} contains standard relational operators ($\sigma_\phi, \pi_A, \bowtie_\theta, \gamma_{g,f}$) and Ω_{sem} contains LLM-powered semantic operators (Patel et al., 2024) ($\text{sem_filter}_\psi, \text{sem_map}_\psi, \text{sem_agg}_\psi, \text{sem_join}_\psi$), each parameterized by a natural language predicate $\psi \in \Psi$. Given a query $q \in \mathcal{Q}$, the goal is to synthesize an executable program $\hat{\mathcal{P}}$ that composes operators from Ω and produces answer $\hat{A}$ matching the gold answer A^* :

$$\hat{\mathcal{P}} = f_\theta(q, D, \Omega), \quad \hat{A} = \llbracket \hat{\mathcal{P}} \rrbracket_D \approx A^* = \llbracket \mathcal{P}^* \rrbracket_D, \quad (1)$$

where f_θ is the synthesis model and $\llbracket \cdot \rrbracket_D$ denotes execution over D . NL2SemProg strictly generalizes NL2SQL (which restricts to Ω_{rel}) and RAG/TableQA (which collapse to a single LLM call), requiring instead a *compositionally structured* program over $\Omega_{\text{rel}} \cup \Omega_{\text{sem}}$.

4 CoreSemDB

To evaluate LLMs on the NL2SemProg task, we construct CoreSemDB with three design principles: (1) queries should require genuine hybrid reasoning that interleaves structured relational operators with LLM-powered semantic operators, (2) the benchmark should support fine-grained diagnosis of model capabilities, and (3) ground-truth annotations must be robust to the inherent stochasticity of LLM-powered execution. As depicted in Fig. 2, we organize CoreSemDB along an application-driven taxonomy and construct it through three stages: database collection, query generation, and query annotation.

Query Taxonomy. CoreSemDB organizes queries into two primary structural types: *single* and *nested*. Single queries can be resolved through a single-stage reasoning or retrieval process, while nested queries involve compositional structures that require sequential reasoning over intermediate results. For example: “What are the main reasons for negative user sentiment in app reviews longer than 30 words, and how do these reasons differ by app category?”. To enable fine-grained capability diagnosis, we further categorize single queries along three orthogonal factors: *semantic reasoning*, *world knowledge*, and *multi-table linking*. This design choice is deliberate: while nested queries are inherently more challenging, their structural complexity makes it difficult to disentangle the contribution of individual factors. By contrast, restricting factor-level analysis to single queries enables clearer attribution of performance variations. Here, *semantic reasoning* targets the core capability of applying LLM-powered operators to interpret unstructured text, such as inferring sentiment or extracting intent from free-form reviews. *World knowledge* captures queries requiring commonsense

understanding beyond the database content, such as knowing that “app category” refers to domain-specific groupings, while *multi-table linking* requires joining information across multiple tables within one database, highlighting multi-hop relational reasoning. Table 1 presents the detailed statistics of CoreSemDB.

4.1 Database Collection

To ground CoreSemDB in realistic data, we select databases from BIRD Li et al. (2023), a widely used NL2SQL benchmark featuring large-scale real-world databases across various domains. Since our focus is on hybrid semantic-relational reasoning, we specifically select databases with rich textual content, specifically those containing substantial unstructured text fields such as user reviews, plot descriptions, and free-form comments, that necessitate LLM-powered semantic operators beyond standard SQL. Applying this criterion, we curate 27 text-rich databases covering diverse domains; detailed statistics are provided in Table 2.

Category	Factor	Count		
		F1	Judge	Total
Single	Sem. Reasoning	143	26	169
	World Knowledge	33	2	35
	Multi-table	22	0	22
Nested	–	97	7	104
Total	–	295	35	330

Table 1: Query statistics of CoreSemDB.

4.2 Query Generation

Constructing a large-scale and diverse query set that requires both semantic understanding and structural filtering is inherently challenging and labor-intensive. To address this, we adopt an expert-LLM agent collaborative procedure (as illustrated in Fig. 2). First, human experts craft a set of *seed queries* based on the taxonomy described above and design tailored generation rules for prompting LLMs. The agent is equipped with tools that can randomly sample data instances from the databases. It then receives the combination of *rules*, *seed queries*, and *sampled data* as input and produces candidate queries for subsequent filtering and annotation.

Statistic	Value
# Databases	27
# Tables	178
# Rows	47,822,963
Avg. tables / DB	6.6
Avg. rows / table	268,668.3
Avg. text cols / DB	21.6

Table 2: DB statistics of CoreSemDB.

4.3 Query Annotation

Query annotation proceeds in two phases to ensure both correctness and reproducibility.

Phase 1: Code Writing and Preliminary Ground Truth. Experts first review and filter the generated candidate queries, retaining those that genuinely require hybrid semantic-relational reasoning. For each retained query, experts write executable code using LOTUS Patel et al. (2024), a framework that provides LLM-powered semantic operators, such as semantic filtering and mapping, composable with standard relational operations. The execution results of these expert-written programs serve as the preliminary ground truth.

Phase 2: Validation via LLM Voting and Re-execution. To further validate correctness, we employ a majority voting mechanism Yang et al. (2024a): three flagship LLMs (Gemini-3-Pro, Claude-4.5-Sonnet, and GPT-5.1) independently assess whether each query-code pair is logically sound and correctly answerable. Based on the voting results and textual critiques, a separate group of experts makes the final determination on each sample’s correctness.

A key challenge is the inherent stochasticity of LLM-powered execution: the same semantic operator may produce slightly different outputs across invocations, such as borderline sentiment judgments on ambiguous reviews. To account for this, experts in Phase 2 re-execute each query code and compare the results against the preliminary ground truth from

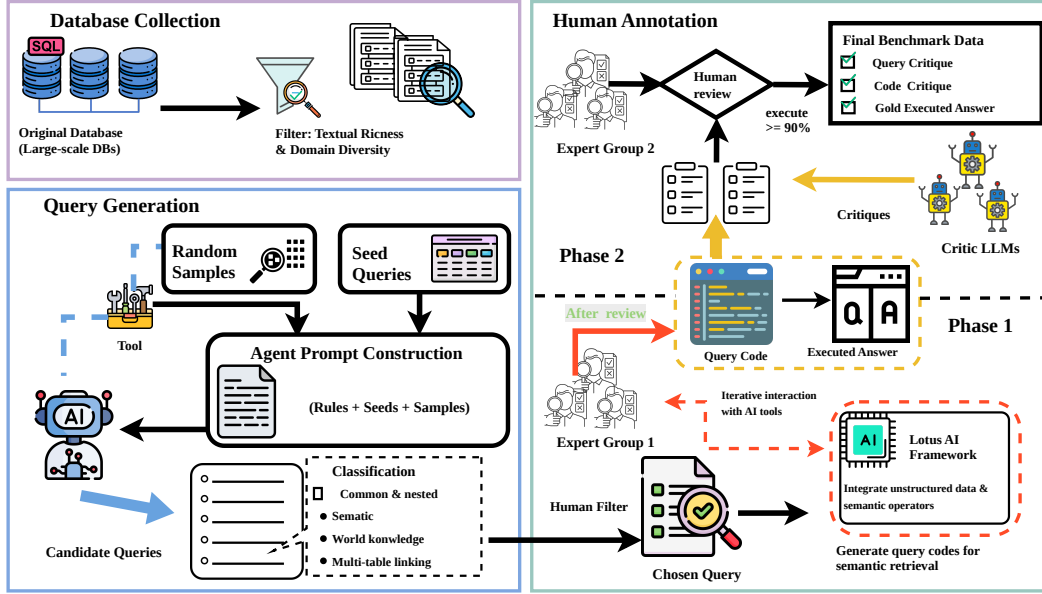


Figure 2: Overview of the CoreSemDB construction pipeline. The pipeline consists of three main stages: database collection, query generation, and two-phase query annotation.

Phase 1. Only samples achieving an F1-score ≥ 0.9 between the two executions are retained, ensuring that the ground truth is robust to LLM output variability.

5 Evaluation and Analysis

5.1 Experimental Setup

Metrics. We report standard evaluation metrics, including Precision (P), Recall (R), and F1-score (F_1). The calculation details are in Appendix A.1. To better assess model performance on CoreSemDB, we further introduce three task-specific metrics tailored to different query answer types.

We categorize query answers into two types: Scalar and Textual. Scalar answers return a finite set of items such as key reasons or entities. Let Q_s denote the set of such queries. We define *Quasi-Exact Match* (QEM) as:

$$QEM = \sum_{i \in Q_s} \mathbb{I}(F1_i \geq 0.9), \quad (2)$$

where $\mathbb{I}(\cdot)$ is the indicator function. A prediction is considered correct if the F1-score between the generated and annotated query code exceeds a threshold of 0.9, indicating high-fidelity structural alignment. We adopt a relaxed threshold rather than exact match because semantic operators invoke LLMs at execution time, and the inherent stochasticity of LLM outputs, amplified over the large number of invocations in a single query—means that even an identical, fully correct program may produce slightly different results across runs. The 0.9 threshold thus accounts for this execution-level noise rather than tolerating logical errors, and its optimality is validated through human calibration (Section 5.3).

Textual answers require higher-level semantic analysis grounded in data. Let Q_t denote this query set. We evaluate the generated query code using an LLM-based judge (DeepSeek-V3.2), which assesses its semantic consistency with expert-annotated code and its logical effectiveness in accurately resolving the query. Details of the evaluation prompt are provided in Appendix 7. We define *Semantic Alignment Match* (SAM) as:

$$SAM = \sum_{j \in Q_t} \mathbb{I}(\text{Score}_j \geq 0.8). \quad (3)$$

Here, Score_j is the assigned score for query j by the judge model.

Model	Single						Nested						Total					
	P	R	F1	QEM	SAM	ACC	P	R	F1	QEM	SAM	ACC	P	R	F1	QEM	SAM	ACC
Proprietary Models																		
Gemini-3-Pro	47.15	47.25	44.28	46	17	27.88	35.28	34.79	31.51	12	3	14.42	43.25	43.15	40.08	58	20	23.64
Claude-4.5-Sonnet	47.63	47.68	43.66	43	18	26.99	33.67	32.66	30.13	13	4	16.35	43.04	42.74	39.21	56	22	23.64
Grok-4	45.32	41.28	38.98	39	14	23.45	39.91	38.34	35.92	17	3	19.23	43.54	40.31	37.97	56	17	22.12
GPT-5.1	37.98	35.44	31.96	23	26	20.80	49.22	46.90	43.06	20	5	24.04	41.68	39.21	35.61	43	29	21.82
Claude-4.5-Haiku	27.36	51.73	29.15	21	17	16.81	25.68	42.23	26.05	10	3	12.50	26.81	48.61	28.13	31	20	15.45
Gemini-3-Flash	33.83	31.83	30.18	30	12	18.58	14.11	19.75	13.40	5	3	7.69	27.35	27.86	24.66	35	15	15.15
Grok-4.1-Fast	19.74	19.77	18.04	17	5	9.73	16.45	12.86	12.29	6	0	5.77	18.66	17.50	16.15	23	5	8.48
Open-source Models																		
DeepSeek-V3.2	43.70	41.13	39.06	33	17	22.12	37.31	35.00	30.87	13	4	16.35	41.60	39.11	36.37	46	21	20.30
Kimi-K2	31.54	44.73	31.50	24	18	18.58	27.10	31.87	24.18	11	5	15.38	30.08	40.50	29.09	35	23	17.58
GLM-4.7	40.36	40.60	37.41	30	15	19.82	26.52	24.90	23.02	10	2	11.54	35.82	35.46	32.69	40	17	17.22
Qwen3-235B-A22B	33.55	30.78	28.69	24	8	14.16	28.45	26.92	21.80	7	2	8.65	31.87	29.51	26.42	31	10	12.42
Qwen3-32B	29.20	29.73	27.11	19	5	10.62	29.68	30.07	26.96	11	2	12.50	29.36	29.84	27.06	30	7	11.21
Qwen3-30B-A3B	34.03	27.61	26.12	19	5	10.62	18.54	21.68	15.24	7	2	8.65	28.94	25.66	22.54	26	7	10.00
Qwen3-14B	28.38	26.61	24.22	21	7	12.33	16.14	17.14	13.93	5	0	4.81	24.37	23.51	20.85	26	7	9.97
Qwen3-8B	16.64	16.32	14.84	12	3	6.64	14.34	17.07	12.95	6	0	5.77	15.88	16.57	14.22	18	3	6.36

Table 3: Multi-dimensional performance comparison across different complexity levels.

Finally, since $Q = Q_t \cup Q_s$, we define the overall accuracy (ACC) as:

$$ACC = (QEM + SAM) / |Q|. \quad (4)$$

Target LLMs. To provide a comprehensive evaluation on CoreSemDB, we select 15 representative LLMs as shown in Table 3. Specifically, we adopt proprietary models: GPT-5.1 [OpenAI \(2025\)](#), Grok-4 series [xAI \(2025\)](#), Claude-4.5 series [Anthropic \(2025\)](#) and Gemini-3 series [Google \(2025\)](#); mainstream open-source models: GLM-4.7 [Z.ai \(2025\)](#), Kimi-K2-Thinking [Team et al. \(2025\)](#), DeepSeek-V3.2 [Liu et al. \(2025a\)](#) and Qwen3 series [Yang et al. \(2025\)](#) including the versions of 235B-A22B-Instruct, 30B MoE and 8B/14B/32B.

5.2 Main Results

Table 3 presents the complete results across CoreSemDB. The data reveals that proprietary models still maintain a leading position across all performance tiers and complexity levels. Gemini-3-Pro and Claude-4.5-Sonnet achieve SOTA performance with a Total ACC of 23.64%, while Gemini-3-Pro leads with the highest overall F1-score of 40.08. Notably, GPT-5.1 exhibits an atypical competency profile, achieving a significantly higher accuracy in *nested* structures (24.04%) than in *single* structures (20.80%). This suggests a specialized architectural advantage in handling hierarchical dependencies that deviates from the performance trends observed in other models.

Simultaneously, open-source models have made significant progress, and the performance gap between the two ecosystems is rapidly narrowing. DeepSeek-V3.2, the top-performing open-source model with a Total ACC of 20.30%, now closely rivals the leading proprietary flagships, with the absolute accuracy margin narrowed to approximately 3 percentage points. However, a near-universal performance decay is still observed when transitioning from *Single* to *Nested* tasks, as evidenced by the sharp decline in F1-score and the widening gap between Precision and Recall. For instance, most models maintain relatively high Recall but suffer from plummeting Precision in nested scenarios, indicating that while models can identify potential data points, they struggle to correctly filter them through multi-layer logical constraints. Mastering the semantic mapping between nested headers and their corresponding logical operations remains the core technical bottleneck for models.

Scaling law. The results further validate the impact of the scaling law as a universal driver for performance across various architectures and model families. In the proprietary sector, there is a clear performance hierarchy correlated with model scale and computational budget. Flagship versions consistently outperform their efficiency-oriented counterparts by a wide

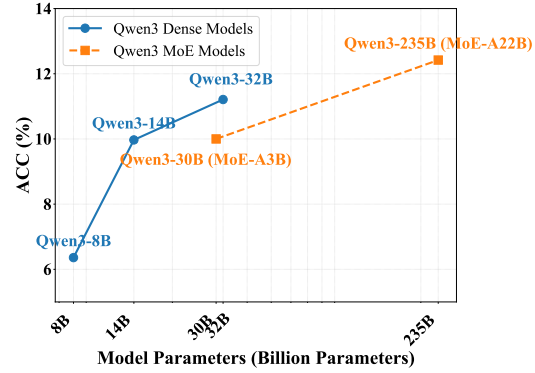


Figure 3: Scaling law on ACC across Qwen3 series.

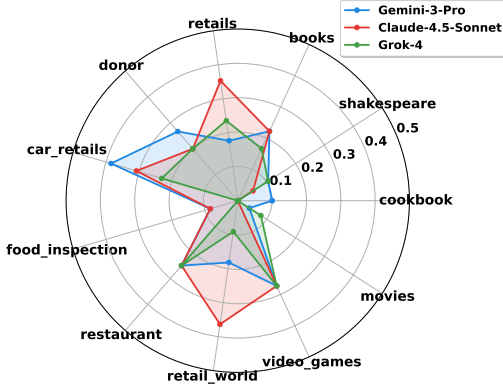


Figure 4: Comparative Analysis of ACC Across Database Domains for Top 3 Proprietary Models.

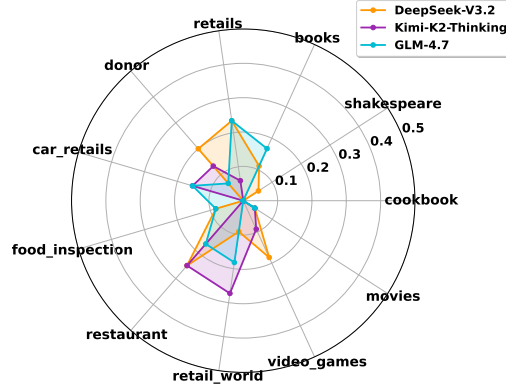


Figure 5: Comparative Analysis of ACC Across Database Domains for Top 3 Open-source Models.

margin. For instance, Gemini-3-Pro (23.64% ACC) significantly exceeds Gemini-3-Flash (15.15% ACC), and Grok-4 maintains a substantial lead over its faster variant, Grok-4.1-Fast. These findings confirm that larger parameter counts and increased compute are essential for mastering the high-level reasoning required for hybrid semantic-relational query processing.

The scaling effect is equally evident in the open-source cohort across both Dense and Mixture-of-Experts (MoE) architectures. Figure 3 illustrates the detailed relationship between parameter scale and accuracy for the Qwen3 series. Within the dense Qwen3 series, performance improves steadily as parameters scale from 8B (6.36% ACC) to 14B (9.97% ACC) and up to 32B (11.21% ACC). A parallel trajectory is observed in the MoE variants, where scaling from the smaller Qwen3-30B-A3B (10.00% ACC) to the massive Qwen3-235B-A22B (12.42% ACC) results in a significant performance jump. While the dense Qwen3-32B remains highly competitive, the superior overall accuracy of the 235B-A22B MoE model underscores that massive scale is instrumental in resolving semantic ambiguities and achieving exact matches. These results suggest that increasing scale remains the most reliable path toward improving performance in reasoning.

5.3 Further Analysis

Fine-grained Performance Analysis across the factor of Single Category. The experimental results for single-query factors are presented in Table 4. Proprietary models Gemini-3-Pro and Claude-4.5-Sonnet demonstrate significant superiority in Semantic Reasoning and World Knowledge, achieving high logical rigor as evidenced by their ACC scores. However, performance across all evaluated models suffers from varying degrees of degradation in the more challenging Multi-table Linking task, indicating that cross-table information integration remains a primary bottleneck for current LLMs. Notably, the open-source model Kimi-K2 emerges as a strong performer in this dimension, achieving the highest Recall of 60.05% and matching Gemini-3-Pro in overall Accuracy.

A deeper analysis suggests that Kimi-K2 adopts a more conservative code generation strategy in complex relational scenarios: by generating broader and more inclusive query logic, the model captures a wider range of potential candidate answers. While this occasionally leads to a trade-off in Precision, it significantly bolsters retrieval robustness in multi-table joins, which is especially important because multi-table tasks are highly susceptible to failure from a single omitted join condition. In contrast, proprietary models maintain a more balanced equilibrium between Precision and Recall in World Knowledge, suggesting more refined synergy between internal knowledge retrieval and structural logical constraints.

Performance difference on database domain. As shown in Figure 4, proprietary models exhibit a higher performance ceiling in highly specialized vertical industries such as ‘car_retails’ and ‘retails’, where Gemini-3-Pro and Claude-4.5-Sonnet achieve accuracy scores within the range of 0.3 to 0.5. This phenomenon indicates that when handling Python code

Model	Semantic Reasoning						World Knowledge						Multi-table Linking					
	P	R	F1	QEM	SAM	ACC	P	R	F1	QEM	SAM	ACC	P	R	F1	QEM	SAM	ACC
Proprietary Models																		
Gemini-3-Pro	48.76	50.80	47.11	34	16	29.59	49.70	43.41	43.96	8	1	25.71	32.81	29.96	26.34	4	-	18.18
Claude-4.5-Sonnet	47.05	46.50	43.93	35	16	30.18	47.90	54.88	44.70	6	2	22.96	51.00	44.55	40.37	2	-	9.09
Grok-4	45.26	41.09	39.07	30	13	25.44	43.61	37.44	35.92	6	1	20.0	48.29	48.29	42.96	3	-	13.64
Open-source Models																		
DeepSeek-V3.2	45.42	42.82	41.75	25	16	24.26	40.32	40.15	32.81	5	1	17.14	37.61	31.61	30.91	3	-	13.64
Kimi-K2	28.55	40.88	28.73	14	17	18.34	36.86	50.75	36.16	6	1	20.0	42.93	60.05	42.54	4	-	18.18
GLM-4.7	40.08	39.16	37.59	23	14	21.89	44.64	50.80	41.50	4	1	14.29	35.83	34.69	30.07	3	-	13.64

Table 4: Performance comparison of proprietary and open-source models across Semantic Reasoning, World Knowledge, and Multi-table Linking.

generation and semantic operator mapping tasks characterized by high logical complexity and dense industry-specific knowledge, top-tier proprietary models possess superior logical deconstruction capabilities and instruction-following precision. In contrast, Figure 5 demonstrates that the peak accuracy of open-source models in similar domains typically fails to surpass 0.3, revealing an objective performance gradient between the open-source community and flagship proprietary models in deep business logic analysis.

In domains with high structural regularity and relatively standardized business logic such as ‘retail world’, open-source models demonstrate significant competitiveness. Notably, Kimi-K2 achieves an accuracy near 0.3 in this field, reaching a level comparable to that of proprietary models. However, when tasks shift to domains with highly complex semantic contexts and a lack of rigorous logical paradigms such as ‘shakespeare’, ‘movies’, or ‘cookbook’, the performance of both model categories suffers a sharp decline, with accuracy generally remaining at a low level of approximately 0.1. This reveals a common bottleneck in current semantic analysis technologies: the generalization capability of models to translate natural language into deterministic procedural operators remains constrained when processing unstructured and highly subjective contexts, regardless of the model architecture.

Threshold calibration with human analysis. To determine the optimal decision thresholds for our evaluation metrics, we conducted an expert-led calibration analysis on 200 representative samples (detailed information about sampling is provided in Appendix A.2). During the analysis, we specifically focused on boundary cases with scores ranging from 0.7 to 1.0. We identified the optimal threshold τ by maximizing the Cohen’s Kappa coefficient κ Landis & Koch (1977) against human-annotated ground truth. Mathematical definition for Cohen’s Kappa is provided in Appendix A.3.

As shown in Table 5, the optimal thresholds are finalized at 0.9 for QEM and 0.8 for SAM. These configurations yield Kappa scores of 0.82 and 0.78 respectively, indicating substantial agreement with human experts. Our results indicate that a strict threshold of 1.0 for QEM leads to an underestimation of performance. This is primarily because natural language diversity within the same operator can introduce semantically equivalent but linguistically varied expressions, leading to discrepancies in the returned sets and preventing the F1-score from reaching a perfect 1.0. Similarly, while the SAM judge model demonstrates strong discriminative power, a threshold of 1.0 proves overly conservative. It often penalizes variants such as minor adjustments in execution order or alternative valid logical paths that do not compromise functional correctness. By adopting these calibrated thresholds, our evaluation framework more accurately identifies functional correctness and effectively filters out noise from syntax and natural language expressions.

Error analysis and Case study. We provide a very detailed error analysis of Gemini-3-Pro in Appendix A.4. To facilitate in-depth analysis, we provide case studies for qualitative analysis in Appendix A.7 across all benchmark categories: Semantic Reasoning, World Knowledge, Multi-table Linking, and Nested. These examples encompass raw model generations as well as their subsequent execution outcomes within our framework.

QEM (F1-score)		SAM (LLM judge score)	
Threshold τ	Kappa κ	Threshold τ	Kappa κ
0.70	0.54	0.70	0.62
0.80	0.68	0.80	0.78
0.90	0.82	0.90	0.69
1.00	0.61	1.00	0.54

Table 5: Threshold calibration results. Bold values indicate the optimal Kappa scores for each metric.

6 Conclusion

In this paper, we introduce CoreSemDB, a holistic and reliable benchmark for evaluating the end-to-end performance of LLMs on the NL2SemProg task. Extensive experiments on 15 representative LLMs demonstrate the significant challenges of hybrid reasoning over text-rich databases. Further analysis reveals valuable insights, including the sharp performance decay in nested queries, varying domain-specific strengths between proprietary and open-source models, and the critical bottleneck presented by multi-table information integration. Our CoreSemDB advances the evaluation of semantic-relational program synthesis for LLMs, enabling precise and executable data interactions.

Reproducibility Statement

We are strongly committed to the reproducibility of our work. CoreSemDB is constructed from publicly available databases sourced from BIRD (Li et al., 2023), and all evaluation metrics, annotation procedures, and threshold calibration details are fully described in the main paper and Appendix. The complete benchmark, along with the evaluation framework and model inference code, will be publicly released upon institutional approval.

Ethics Statement

This research involved human employees to finish the query annotation during the construction of our benchmark. The following considerations were adhered to ensure the protection and ethical treatment of participants: 1) Voluntary Participation: All participants were informed about the nature of the research and their role in it. Participation was entirely voluntary, with participants having the right to withdraw at any time without any consequences. 2) Informed Consent: Written informed consent was obtained from all participants. This consent form detailed the purpose of the research, the procedures involved, potential risks, and measures taken to safeguard participant data. 3) Data Anonymity and Confidentiality: All data collected during the study were anonymized. Personal identifiers were removed to maintain confidentiality and data were stored securely to prevent unauthorized access. 4) Minimal Risk: The study involved minimal risk to participants. The tasks performed were similar to everyday activities, and no sensitive personal information was requested or recorded.

Acknowledgment

This work is partially supported by industrial donation grant and the 2025 First Processing Cycle of the Research Matching Grant Scheme (RMGS). We would like to extend our sincere thanks to Jinghao Luo, Jingze Wang, and Man Tik SO for their meticulous work on data annotation and their substantial contributions to the experimental procedures. Their efforts were essential to the successful completion of this study.

References

- Anthropic. Introducing claude sonnet 4.5. <https://www.anthropic.com/news/claude-sonnet-4-5>, 2025. URL <https://www.anthropic.com/news/claude-sonnet-4-5>. Accessed: 2025-12-XX.
- Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E Gonzalez, Carlos Guestrin, and Matei Zaharia. Text2sql is not enough: Unifying ai and databases with tag. *arXiv preprint arXiv:2408.14717*, 2024.
- Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17754–17762, 2024.

- Databricks. Apply ai on data using databricks ai functions. <https://docs.databricks.com/aws/en/large-language-models/ai-functions>. Accessed: 2025-11-23.
- Google. A new era of intelligence with gemini 3. <https://blog.google/products/gemini/gemini-3/>, 2025. URL <https://blog.google/products/gemini/gemini-3/>. Accessed: 2025-12-XX.
- Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models. In *International conference on artificial intelligence and statistics*, pp. 5549–5581. PMLR, 2023.
- Sandeep Karmarkar and Jiaxun Wu. Introducing bigquery dataframes 2.0 for the era of multimodal data science. *Google Cloud Blog*, 2025. URL <https://cloud.google.com/blog/products/data-analytics/a-closer-look-at-bigquery-dataframes-2-0>. Accessed: 2025-11-23.
- J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pp. 159–174, 1977.
- Jiale Lao, Andreas Zimmerer, Olga Ovcharenko, Tianji Cong, Matthew Russo, Gerardo Vitagliano, Michael Cochez, Fatma Özcan, Gautam Gupta, Thibaud Hottelier, et al. Sembench: A benchmark for semantic query processing engines. *arXiv preprint arXiv:2511.01716*, 2025.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.
- Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.
- Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. Palimpsest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*, pp. 2, 2025b.
- OpenAI. Gpt-5.1: A smarter, more conversational chatgpt. <https://openai.com/index/gpt-5-1/>, November 2025. URL <https://openai.com/index/gpt-5-1/>. Accessed: 2025-12-XX.
- Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. Semantic operators: A declarative model for rich, ai-based data processing. *arXiv preprint arXiv:2407.11418*, 2024.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*, 2024.
- Snowflake. Snowflake ai and ml. <https://docs.snowflake.com/en/guides-overview-ai-features>. Accessed: 2025-11-23.
- Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

- Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. Tablebench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25497–25506, 2025.
- xAI. Grok 4. <https://x.ai/news/grok-4>, 2025. URL <https://x.ai/news/grok-4>. Accessed: 2025-12-XX.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- Joshua C Yang, Damian Dailisan, Marcin Korecki, Carina I Hausladen, and Dirk Helbing. Llm voting: Human choices and ai collective decision-making. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, volume 7, pp. 1696–1708, 2024a.
- Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze D Gui, Ziran W Jiang, Ziyu Jiang, et al. Crag-comprehensive rag benchmark. *Advances in Neural Information Processing Systems*, 37:10470–10490, 2024b.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.
- Z.ai. Glm-4.7: Advancing the coding capability. <https://z.ai/blog/glm-4.7>, 2025. URL <https://z.ai/blog/glm-4.7>. Accessed: 2025-12-XX.
- Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.
- Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*, 2021.

A Appendix

A.1 Evaluation Metrics

To evaluate the performance of our model in predicting the target elements, we employ three standard metrics: Precision (P), Recall (R), and the F1-score (F_1). These metrics are calculated based on the overlap between the predicted set and the ground truth set.

Let $\mathcal{S}_{pred}$ denote the set of elements predicted by the model, and $\mathcal{S}_{gt}$ denote the set of ground truth elements. The evaluation metrics are defined as follows:

- **Precision (P):** Measures the proportion of correctly predicted elements among all elements in the predicted set:

$$P = \frac{|\mathcal{S}_{pred} \cap \mathcal{S}_{gt}|}{|\mathcal{S}_{pred}|} \quad (5)$$

- **Recall (R):** Measures the proportion of ground truth elements that are successfully captured by the prediction:

$$R = \frac{|\mathcal{S}_{pred} \cap \mathcal{S}_{gt}|}{|\mathcal{S}_{gt}|} \quad (6)$$

- **F1-score (F_1):** The harmonic mean of Precision and Recall, providing a balanced assessment of the model’s performance:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \quad (7)$$

Where $|\cdot|$ denotes the cardinality of the set, and $\mathcal{S}_{pred} \cap \mathcal{S}_{gt}$ represents the intersection of the predicted and ground truth sets. All metrics range from 0 to 1, with values closer to 1 indicating higher accuracy and completeness.

A.2 Threshold Calibration Details

During the preliminary design phase, we conducted a systematic sampling of model outputs to validate our automated evaluation criteria. This process ensures that the selected thresholds accurately distinguish functionally correct code from erroneous outputs based on human expert judgment.

The calibration dataset includes 200 samples in total where 100 samples are allocated respectively to the QEM and SAM evaluation methods. We specifically prioritize the critical interval between 0.7 and 1.0 to rigorously test the discriminative performance of the metrics on near-correct solutions. Table 6 summarizes the detailed distribution of these samples across different score ranges.

Score Range	Samples(QEM)	Samples(SAM)	Total
0.0 to 0.4	10	10	20
0.4 to 0.7	20	20	40
0.7 to 0.8	20	20	40
0.8 to 0.9	30	30	60
0.9 to 1.0	20	20	40
Total	100	100	200

Table 6: Combined distribution of calibration samples across evaluation methods.

A.3 Mathematical Definitions of Cohen’s Kappa

To evaluate the consistency between automated metrics and human labels, we utilize the following statistical measures. Cohen’s Kappa coefficient denoted as κ is employed to provide a robust measure of inter-rater reliability by accounting for the possibility of agreement occurring by chance. The coefficient is calculated using the following formula:

$$\kappa = \frac{P_o - P_e}{1 - P_e}. \quad (8)$$

In this equation, P_o represents the observed proportionate agreement while P_e denotes the hypothetical probability of chance agreement. We define P_o as the sum of TP and TN divided by the total number of samples N . The expected agreement P_e is derived from the marginal totals of the contingency table as follows:

$$P_e = ((TP + FP)(TP + FN) + (TN + FN)(TN + FP)) / (N^2). \quad (9)$$

The variables TP , TN , FP , and FN represent true positives, true negatives, false positives, and false negatives respectively.

Following the standard academic interpretation proposed by Landis and Koch, the strength of agreement is categorized based on the value of κ . As shown in Table 7, a κ value exceeding 0.81 is considered to indicate almost perfect agreement between the automated metric and human ground truth.

A.4 Error Analysis

To investigate the underlying limitations of Gemini-3-Pro in generating Python programs integrated with semantic operators, we perform a comprehensive manual error analysis on 252 failure instances. The error distribution is categorized into eight distinct failure modes as illustrated in Figure 6.

Kappa Value Range	Strength of Agreement
Below 0.00	Poor
0.00 to 0.20	Slight
0.21 to 0.40	Fair
0.41 to 0.60	Moderate
0.61 to 0.80	Substantial
0.81 to 1.00	Almost Perfect

Table 7: Interpretation of Cohen’s Kappa coefficient according to Landis and Koch standards.

The primary bottlenecks stem from Fine-grained Discrepancies (19.8%) and Structural Incoherence (19.0%), where the model either fails to adhere to precise operator interface protocols or exhibits gaps in the data-flow chaining between semantic steps. Significant proportions of failures are also attributed to Task Misalignment (15.9%) and Convergence Failure (14.7%), reflecting challenges in intent representation and stable sequence generation, respectively. Notably, Scope Planning Errors (12.7%) represent a critical failure mode where the model lacks sufficient boundary constraints for semantic predicates, leading to excessive execution complexity and timeouts. Furthermore, the analysis identifies Edge-case Logic Residuals (8.3%), Semantic Alignment Divergence (8.3%), and Environmental Exceptions (1.2%). These modes characterize residual logical flaws in state transitions, subtle conceptual shifts in semantic predicates, and infrastructure-level anomalies that hinder the final verification of otherwise functional code.

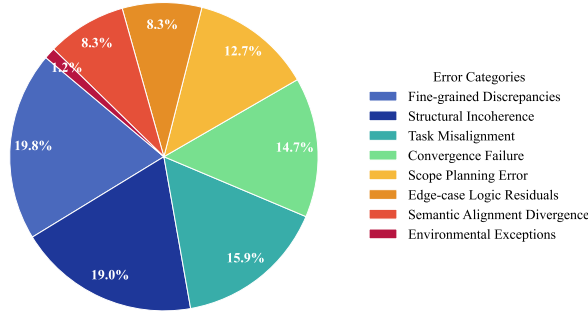


Figure 6: The Error Distribution of Gemini-3-Pro.

A.5 Human Annotator Information

During the query annotation, the two discrete human expert groups totally include 8 people who has a strong background of computer science field.

A.6 Prompt

Code Comparison Prompt

Role. You are an expert code reviewer. Your task is to compare two Lotus pipeline code implementations and determine their quality.

Query: {query}

Ground Truth Code: {ground_truth_code}

Generated Code: {generated_code}

Instructions.

- First, analyze whether the generated code implements the same logical steps as the ground truth code. Consider:
 - Do they use the same data sources and tables?
 - Do they perform the same filtering, joining, and transformation operations?
 - Do they use the same semantic operators (sem_filter, sem_join, sem_agg, sem_topk, sem_map) with equivalent logic?
 - Do they produce the same type of output (list, set, single value, etc.)?
 - Are the key computational steps and data flow equivalent?
- IMPORTANT:** If the code logic is not completely identical, but based on your analysis of the code logic, the generated code can correctly solve the query (i.e., it will produce the same or equivalent results as the ground truth code), you should still give a **HIGH score** (≥ 0.75). Different approaches that achieve the same goal are acceptable.
- Scoring Guidelines:**
 - 1.0: logically equivalent, same approach and steps
 - 0.9–0.95: different approach but based on code analysis, it will correctly solve the query
 - 0.75–0.85: mostly equivalent logic with minor differences, or different but correct approach based on code analysis
 - 0.5–0.7: partially similar, some steps match but key differences exist, and it may not solve the query correctly
 - 0.0–0.4: completely different logic, different approach, and based on code analysis, it cannot solve the query correctly
- Note:** Minor differences in variable names, code style, or implementation details are acceptable as long as the core logic is the same **OR** based on your code analysis, the code can correctly solve the query.

Output Format.

Similarity Score: <score>[0.0–1.0]</score>
 Reasoning: [Your detailed explanation of the logical comparison and whether the generated code can solve the query correctly based on your code analysis]

Example.

Similarity Score: <score>0.9</score>
 Reasoning: The generated code uses a different approach (uses sem_join instead of merge), but based on the code logic analysis, it will correctly solve the query and produce the same result. Both codes load the same tables and filter for free Tools apps, but use different joining methods that achieve the same goal.

Figure 7: Prompt used for Lotus pipeline code comparison and evaluation.

Code Generation Prompt for Table Reasoning

Task. Generate executable code to answer the query using the provided table data and the operators listed below.

Available Operators.

Standard Pandas operators: `df.groupby()`, `df.merge()`, `df.sort_values()`, `df.filter()`, etc.

Natural-language-based operators (column names must be wrapped in `{{}}`):

- **sem_filter:** Filter rows based on a natural language condition *Example:* `df.sem_filter('{{location}} is a county in the Bay Area')`
- **sem_join:** Join tables based on a natural language condition *Example:* `df1.sem_join(df2, 'match rows where {{name1}} and {{name2}} are similar')`
- **sem_agg:** Aggregate data based on a natural language description *Example:* `df.sem_agg('What qualities are represented by the badge {{Name}}s')._output[0]`
- **sem_topk:** Get the top-*k* rows based on a natural language ranking *Example:* `df.sem_topk('What {{School}} sounds most futuristic?', 1)`
- **sem_map:** Transform data based on a natural language description (creates a new column named `_map`) *Example:* `df.sem_map('{{Body}} is relevant to machine learning. Answer with YES if it is and NO if it is not.')`

Table Data. `{table_data}`

Query. `{query}`

Additional Instructions. `{format_instruction}` Please generate the code in “lotus” format.

Important Notes.

- Natural-language-based operators invoke LLM calls and are more time-consuming and costly than standard Pandas operations.
- When appropriate, use standard Pandas operations (e.g., `df[df['col'] > val]`) to narrow down the dataset before applying semantic operators.
- Always explicitly use `return` to return the result.
- Do **not** return a variable name alone.
- Do **not** define nested functions; write the code logic directly.

Code Template.

```
“lotus
import lotus
import pandas as pd
from lotus.models import LM
# [Step-by-step logic here]
# ...
return result
“
```

Figure 8: Prompt template for LLM-based code generation with table reasoning.

A.7 Case Analysis

Case Study 1: Semantic Reasoning Type

Query: What patterns can be found in the award categories and results (Winner vs. Nominee) for *The Simpsons* over the years?

Database: simpson_episodes

Code Generated Model: google/gemini-3-pro-preview

1. Ground Truth

The Simpsons has received numerous nominations and awards across various categories.

Most Frequent Wins:

- **OFTA Television Award:** Best Animated Series; Best Voice-Over Performance (once each).
- **Primetime Emmy:** Outstanding Voice-Over Performance (one win).
- **WGA Award (TV):** Animation (one win).
- **Jupiter Award:** Best International TV Series (one win).
- **People's Choice Award:** Favorite Animated Comedy (one win).

Recurring Nominations:

- **Primetime Emmy:**
 - Outstanding Animated Program (For Programming Less Than One Hour): 20 nominations.
 - Outstanding Voice-Over Performance: 2 nominations.
 - Outstanding Music Composition for a Series (Original Dramatic Score): 1 nomination.
- **WGA Award (TV):**
 - Comedy Series: 25 nominations.
 - Animation: 6 nominations.
- **Annie Award:**
 - Writing in a Television Production: 3 nominations.
 - Best Animated Television Production: 1 nomination.
 - Best Directing in an Animated Television Production or Short Form: 1 nomination.
- **Blimp Award:** Favorite Cartoon (3 nominations).
- **Image Award:** Outstanding Writing in a Comedy Series (2 nominations).
- **OFTA Television Award:**
 - Best Voice-Over Performance: 1 nomination.
 - Best Music in a Series: 1 nomination.

Category-Specific Success Rates:

- **Primetime Emmy:** Wins concentrated in Outstanding Voice-Over Performance.
- **WGA Award (TV):** Single win in Animation despite numerous nominations.
- **OFTA Television Award:** One win each in Best Animated Series and Best Voice-Over Performance.

2. Code Executed Result

The Simpsons has received numerous awards and nominations across a wide range of categories, demonstrating its consistent critical acclaim and cultural impact over several decades.

Award Categories: The awards span both **creative** and **technical** aspects of animation and television production.

- **Creative Categories:** These are dominant and include:
 - **Outstanding Animated Program:** A recurring and highly successful category for *The Simpsons*, with multiple wins and nominations at the Primetime Emmy Awards.
 - **Voice Acting:** Awards for “Outstanding Voice-Over Performance” (Emmy) and “Voice Acting in an Animated Television Production” (Annie) highlight the show’s strong vocal talent.
 - **Writing:** “Outstanding Writing for a Comedy Series” (Emmy) and “Writing in an Animated Television Production” (Annie) recognize the show’s comedic and narrative strengths.

- **Directing:** “Outstanding Directing for a Comedy Series” (Emmy) and “Directing in an Animated Television Production” (Annie) acknowledge the visual storytelling.
 - **Music:** “Outstanding Music and Lyrics” (Emmy) and “Outstanding Original Music Composition for a Series” (Emmy) point to the show’s memorable musical contributions.
 - **Character Animation / Design:** “Outstanding Individual Achievement in Character Animation” (Emmy) and “Character Design in an Animated Television Production” (Annie) recognize the visual artistry.
 - **Storyboarding:** “Storyboarding in an Animated Television Production” (Annie) represents another technical-creative category.
 - **Overall Best Animated Feature / Program:** Awards such as the “Annie Award for Best Animated Television Production” and the “Producers Guild of America Award for Outstanding Producer of Animated Television” recognize the show’s overall excellence.
 - **Technical Categories:** While less frequent than creative awards, technical aspects are also recognized:
 - **Sound Mixing / Editing:** “Outstanding Sound Mixing for a Comedy or Drama Series” (Emmy) and “Outstanding Sound Editing for a Series” (Emmy) acknowledge the audio production quality.
 - **Picture Editing:** “Outstanding Picture Editing for a Comedy Series” (Emmy) highlights post-production work.
- Organizations Giving Awards:** The awards come from prestigious organizations in television and animation.
- **Primetime Emmy Awards:** The most prominent and frequently awarding body, with numerous wins and nominations across a wide range of categories.

Figure 9: Query Case 1: semantic reasoning type.

Code comparison between human and model (Case 1)**Code written by human expert**

```
import lotus
import pandas as pd
from lotus.models import LM

award_df = pd.read_csv('csv_tables/Award.csv')

result = award_df.sem_agg(
    "Analyze_patterns_in_the_{award_category}'_and_{result}'_"
    "(Winner_vs_Nominee)_for_The_Simpsons_over_the_years."
)

result_set = set(result['_output'].dropna())

return result_set
```

Code generated by gemini-3-pro-preview

```
import lotus
import pandas as pd
from lotus.models import LM

# Load the Award table
df_awards = pd.read_csv('simpson_episodes/csv_tables/Award.csv')

df_analysis = df_awards[['year', 'organization', 'award_category', 'result',
    'award']]
df_analysis = df_analysis.sort_values('year')

df_analysis['description'] = (
    df_analysis['year'].astype(str) + ":_ " +
    df_analysis['organization'] + "_- " +
    df_analysis['award_category'] + "_(" +
    df_analysis['result'] + ")"
)

analysis_prompt = (
    "Analyze_the_following_list_of_awards_for_The_Simpsons._"
    "Identify_patterns_in_the_award_categories,_organizations,_ "
    "and_the_results_(Winner_vs_Nominee)_over_the_years."
)

result = df_analysis.sem_agg(analysis_prompt)._output[0]

return result
```

Figure 10: Code comparison between human-written and model-generated solutions for Case 1.

Case Study 2: Multi-table Linking Type

Query: Which street names that have the most restaurants are recognized as iconic or central commercial streets in their cities?

Database: restaurant

Code Generated Model: google/gemini-3-pro-preview

Comparison of Ground Truth and Model Output:	
Ground Truth:	Code Executed Result:
• s. Main St. (Highway One) • Petaluma Blvd N • South Main St • Mission Blvd • Mission St • E 14th St • N Main St • S El Camino Real • San Anselmo Ave. • Robson St. • Sonoma Hwy • N Santa Cruz Ave • W El Camino Real • Bank St • Burlingame Ave • Grand Ave • Center St • Soquel Dr • Broadway • Freedom Blvd • Main St • San Ramon Valley Blvd • Bridgeway • North State St • El Camino Real • Sonoma Blvd • Hartz Ave • San Pablo Ave • Mount Diablo Blvd • Castro Valley Blvd • Stevens Creek Blvd • Fremont Blvd • Shattuck Ave • Healdsburg Avenue • Scotts Valley Dr	• Shattuck Ave • Geary Blvd • Telegraph Ave • Telegraph Ave (duplicate) • W El Camino Real

Figure 11: Query Case 2: multi-table linking type.

Code comparison between human and model (Case 2)
Code written by human expert <pre> import lotus import pandas as pd from lotus.models import LM # Load required tables location_df = pd.read_csv('restaurant/csv_tables/location.csv') generalinfo_df = pd.read_csv('restaurant/csv_tables/generalinfo.csv') # Merge to get complete location info (city comes from location table) merged_location = pd.merge(location_df[['id_restaurant', 'street_name', 'city']], generalinfo_df[['id_restaurant']], on='id_restaurant', how='inner') # Count restaurants per street and city </pre>

```

street_counts = (
    merged_location.groupby(['city', 'street_name'])
        .size()
        .reset_index(name='restaurant_count')
)
# Find streets with max restaurant count per city
max_streets = street_counts.loc[
    street_counts.groupby('city')['restaurant_count'].idxmax()
]
# Identify iconic streets using NL-based filter
iconic_streets = max_streets.sem_filter(
    '{street_name}_is_an_iconic_or_central_commercial_street_in_{city}'
)
# Extract unique street names and convert to set
result_set = set(iconic_streets['street_name'])
return result_set

```

Code generated by gemini-3-pro-preview

```

import lotus
import pandas as pd
from lotus.models import LM

# Load the location table to get street names and restaurant counts
df_location = pd.read_csv('restaurant/csv_tables/location.csv')
# Group by street name and city to count the number of restaurants
street_counts = (
    df_location.groupby(['street_name', 'city'])
        .size()
        .reset_index(name='restaurant_count')
)
# Sort by restaurant count in descending order to find the streets with the most
# restaurants
# We'll take a reasonable top K (e.g., top 20) to filter semantically,
# as checking every street is inefficient
top_streets = (
    street_counts.sort_values('restaurant_count', ascending=False)
        .head(20)
)
# Use sem_filter to identify which of these top streets are iconic
# or central commercial streets
top_streets = top_streets.sem_filter(
    '{street_name}_in_{city}_is_recognized_as_an_iconic_or_central_commercial_
    street'
)
# Extract the street names into a list
result = top_streets['street_name'].tolist()
return result

```

Figure 12: Code comparison between human-written and model-generated solutions for Case 2.

Case Study 3: World Knowledge Type	
Query: Which platform names contain abbreviations?	
Database: video_games	
Code Generated Model: anthropic/claude-haiku-4.5	
Comparison of Ground Truth and Model Output:	
Ground Truth:	Code Executed Result:
• 3DS • N64 • PC • WS • X360 • PS2 • PSP • PS3 • GEN • GBA • GG • PCFX • SNES • GB • XOne • PS4 • PSV • PS • SAT • NES	• Wii • NES • GB • DS • X360 • PS3 • PS2 • SNES • GBA • 3DS • PS4 • N64 • PS • XB • PC • 2600 • PSP • XOne • GC • WiiU • DC • PSV • SAT • SCD • WS • TG16 • 3DO • GG • PCFX

Figure 13: Query Case 3: world knowledge type.

Code comparison between human and model (Case 3)
Code written by human expert <pre> import lotus import pandas as pd from lotus.models import LM platform_df = pd.read_csv('csv_tables/platform.csv') platform_df = platform_df[platform_df['platform_name'].notna() & platform_df['platform_name'].str.strip().ne('')] abbreviated_platforms = platform_df.sem_filter("Filter_platform_names_{platform_name}_that_contain_abbreviations_or_" "shortened_forms,_such_as_'PS'_for_PlayStation,_'XOne'_for_Xbox_One,_" "'NES'_for_Nintendo_Entertainment_System,_'PC'_for_Personal_Computer,_etc." </pre>

```
)

result_set = set(
    abbreviated_platforms['platform_name']
    .dropna()
    .unique()
)

return result_set
```

Code generated by claude-haiku-4.5

```
import lotus
import pandas as pd
from lotus.models import LM

# Load the platform data
platform_df = pd.read_csv('video_games/csv_tables/platform.csv')

# Use semantic filter to identify platform names containing abbreviations
# Abbreviations are typically acronyms or alphanumeric codes
filtered_df = platform_df.sem_filter(
    '{platform_name}_contains_an_abbreviation_such_as_alphanumeric_codes,_
    'acronyms,_or_short_names_like_3DS,_N64,_PC,_PS2,_PS3,_PS4,_PS5,_
    'Xbox,_Wii,_GB,_DS,_or_similar'
)

# Extract platform names as a list
result_list = filtered_df['platform_name'].tolist()

# Return the result
return result_list
```

Figure 14: Code comparison between human-written and model-generated solutions for Case 3.

Case Study 4: Nested Type

Query: Which of the top 10 publishers by sales have successfully ported their flagship games across multiple console generations?

Database: new_video

Code Generated Model: deepseek/deepseek-v3.2

Comparison of Ground Truth and Model Output:

Ground Truth:	Code Executed Result:
• Activision • Namco Bandai Games • Konami Digital Entertainment • Sony Computer Entertainment • Nintendo • Electronic Arts • Sega	• Activision • Electronic Arts • Konami Digital Entertainment • Namco Bandai Games • Nintendo • Sega • Sony Computer Entertainment • Take-Two Interactive • THQ • Ubisoft

Figure 15: Query Case 4: nested type.

Code comparison between human and model (Case 4): human code part one**Code written by human expert**

```

import pandas as pd
import lotus
from lotus.models import LM

game_df = pd.read_csv('csv_tables/game.csv')
game_publisher_df = pd.read_csv('csv_tables/game_publisher.csv')
publisher_df = pd.read_csv('csv_tables/publisher.csv')
game_platform_df = pd.read_csv('csv_tables/game_platform.csv')
platform_df = pd.read_csv('csv_tables/platform.csv')
region_sales_df = pd.read_csv('csv_tables/region_sales.csv')

merged_df = pd.merge(
    game_publisher_df,
    publisher_df,
    left_on='publisher_id',
    right_on='id',
    how='inner',
    suffixes=('_game_pub', '_publisher'))

merged_df = pd.merge(
    merged_df,
    game_df,
    left_on='game_id',
    right_on='id',
    how='inner',
    suffixes=('', '_game'))

merged_df = pd.merge(
    merged_df,
    game_platform_df,
    left_on='id_game_pub',
    right_on='game_publisher_id',
    how='inner',
    suffixes=('', '_game_platform'))

merged_df = pd.merge(
    merged_df,
    platform_df,
    left_on='platform_id',
    right_on='id',
    how='inner',
    suffixes=('', '_platform'))

merged_df = pd.merge(
    merged_df,
    region_sales_df,
    left_on='id_game_platform',
    right_on='game_platform_id',
    how='inner')

publisher_sales = (
    merged_df.groupby(['publisher_id', 'publisher_name'])
    .agg({'num_sales': 'sum'})
    .reset_index()
    .rename(columns={'num_sales': 'total_sales'})
    .sort_values(by='total_sales', ascending=False))

```

Figure 16: Code comparison between human-written and model-generated solutions for Case 4, and the part one of human code.

Code comparison between human and model (Case 4): human code part two**Code written by human expert**

```

#continue from part one
top_10_publishers = publisher_sales.head(10).copy()

if top_10_publishers.empty:
    return set()

top_10_publisher_ids = set(top_10_publishers['publisher_id'])
df_top_10 =
    merged_df[merged_df['publisher_id'].isin(top_10_publisher_ids)].copy()

publisher_game_info = (
    df_top_10.groupby(['publisher_id', 'publisher_name', 'game_id', 'game_name'])
    .agg({
        'platform_name': lambda x: ','.join(sorted(set(x))),
        'num_sales': 'sum',
        'release_year': lambda x: ','.join(
            sorted(set(str(y) for y in x if pd.notna(y)))
        )
    })
    .reset_index()
    .rename(columns={
        'platform_name': 'platforms',
        'num_sales': 'game_total_sales',
        'release_year': 'release_years'
    }))

publisher_portfolio = (
    publisher_game_info.groupby(['publisher_id', 'publisher_name'])
    .apply(
        lambda group: pd.Series({
            'games_info': ','.join([
                f"{row['game_name']}_{Sales:{row['game_total_sales']:.2f}M,_"
                f"Platforms:{row['platforms']},_Years:{row['release_years']}"
                for _, row in group.iterrows()
            ]),
            'total_games': len(group),
            'total_publisher_sales': group['game_total_sales'].sum()
        })
    )
    .reset_index())

publisher_portfolio['Description'] = publisher_portfolio.apply(
    lambda row: (
        f"Publisher:{row['publisher_name']}._"
        f"Total_Sales:{row['total_publisher_sales']:.2f}M._"
        f"Games_Portfolio:{row['games_info']}"
    ),
    axis=1)

successful_publishers = publisher_portfolio.sem_filter(
    "Filter_for_publishers_where_{Description}'_indicates_they_have_"
    "successfully_ported_their_flagship_games_across_multiple_console_"
    "generations_(e.g.,_PS2_to_PS3_to_PS4,_Xbox_360_to_Xbox_One,_"
    "Nintendo_DS_to_3DS_to_Switch).")

if successful_publishers.empty:

```

```

    return set()

result_set = set(successful_publishers['publisher_name'].dropna())

return result_set

```

Figure 17: Code comparison between human-written and model-generated solutions for Case 4, and the part two of human code.

Code comparison between human and model (Case 4): model code part

Code generated by deepseek-v3.2

```

import lotus
import pandas as pd
from lotus.models import LM
# Load all tables
game = pd.read_csv('video_games/csv_tables/game.csv')
game_platform = pd.read_csv('video_games/csv_tables/game_platform.csv')
game_publisher = pd.read_csv('video_games/csv_tables/game_publisher.csv')
genre = pd.read_csv('video_games/csv_tables/genre.csv')
platform = pd.read_csv('video_games/csv_tables/platform.csv')
publisher = pd.read_csv('video_games/csv_tables/publisher.csv')
region = pd.read_csv('video_games/csv_tables/region.csv')
region_sales = pd.read_csv('video_games/csv_tables/region_sales.csv')

# Step 1: Calculate total sales per publisher
sales_chain = region_sales.merge(
    game_platform, left_on='game_platform_id', right_on='id')
sales_chain = sales_chain.merge(
    game_publisher, left_on='game_publisher_id', right_on='id')
sales_chain = sales_chain.merge(
    publisher, left_on='publisher_id', right_on='id')

publisher_sales = (
    sales_chain.groupby(['publisher_id', 'publisher_name'])['num_sales']
    .sum()
    .reset_index())

top10_publishers = (
    publisher_sales.nlargest(10, 'num_sales')['publisher_id']
    .tolist())

# Step 2: Check multi-generation ports
publisher_games = game_publisher[
    game_publisher['publisher_id'].isin(top10_publishers)
]
publisher_game_platforms = publisher_games.merge(
    game_platform,
    left_on='id',
    right_on='game_publisher_id')

platform_counts = (
    publisher_game_platforms
    .groupby(['publisher_id', 'game_id'])['platform_id']
    .nunique()
    .reset_index(name='platform_count'))

ported_games = platform_counts[platform_counts['platform_count'] >= 2]

```

```
# Step 3: Identify publishers with at least one ported game
publishers_with_ported_games = (
    ported_games['publisher_id'].unique().tolist())

# Step 4: Get publisher names
result_df = publisher[publisher['id'].isin(publishers_with_ported_games)]
result_list = result_df['publisher_name'].tolist()

return result_list
```

Figure 18: Code comparison between human-written and model-generated solutions for Case 4, and the part of model code.