Text-to-SQL Benchmarks Toward Enterprise Realities: Under Massive Scopes, Complex Schemas and Scattered Knowledge

Yuchen Ji[1♡], Chengxi Yu[1♡], Jie Shi[2♠], Weixin Zhou[2♡], Xi Cao[1♡], Jiaqing Liang[1♠*]
Yu Mao[3], Hai Chen[3], Deqing Yang[1♠*], Yanghua Xiao[2♠]
[1]School of Data Science, Fudan University
[2]College of Computer Science and Artificial Intelligence, Fudan University
[3]Ant Group
[♡]{ycji24,yucx24,wxzhou25,caox22}@m.fudan.edu.cn,
[♠]{jieshi,liangjiaqing,yangdeqing,shawyh}@fudan.edu.cn
[3]{songhao.my,chenhai.ch}@antgroup.com

Abstract

Existing Text-to-SQL benchmarks remain idealized compared with enterprise scenarios, which require massive query scopes, complex schemas, and scattered external knowledge. We present BIRD-Ent and Spider-Ent, two enterprise-oriented benchmarks constructed via a refinement framework, together with Dual-Retrieval-Augmented-Generation (DRAG) Text-to-SQL, which formalizes table-schema and knowledge retrieval before SQL generation. The benchmarks feature over 4,000 columns per query and, for BIRD-Ent, documents totaling 1.5M tokens. Evaluation on state-of-the-art models shows sharp drops, with only 39.1% EX on BIRD-Ent and 60.5% EX on Spider-Ent. These results establish the Ent-series as a discriminative testbed for structurally robust and retrieval-aware Text-to-SQL.

1 Introduction

Text-to-SQL maps natural language questions to SQL, enabling non-experts to query relational databases and helping professionals write SQL more efficiently. It has therefore attracted sustained attention from both the NLP and database communities (Li et al., 2023a; Pourreza and Rafiei, 2023; Gao et al., 2024; Shkapenyuk et al., 2025).

Benchmarks define the task setting and provide a basis for measuring progress. Although large language models (LLMs) now reach 82.0% execution accuracy (EX) on BIRD (Li et al., 2023b) and 91.2% on Spider (Yu et al., 2018), such performance does not necessarily transfer to enterprise scenarios. As shown in Figure 1, academic benchmarks typically involve compact query scopes, simplified schemas, and tightly coupled or inlined knowledge. Recent enterprise-oriented benchmarks such as Spider 2.0 (Lei et al., 2025) and BEAVER (Chen et al., 2025a) make important progress, but their query scopes remain limited and external knowledge is often provided as concise, query-specific snippets rather than large, loosely organized document collections.

Our observations of real enterprise systems show recurring structural challenges also reflected in public data-warehouse practices and prior studies (Wang et al., 2025; Eben et al., 2025; Cai et al., 2025; Chen et al., 2025b). Enterprise databases are usually organized as domain-specific data assets containing hundreds or thousands of tables from diverse sources. Their schemas often use hierarchical, highly abbreviated, and redundant naming conventions, while required knowledge such as column descriptions or domain rules is scattered across heterogeneous documents. Enterprise Text-to-SQL therefore requires retrieving relevant tables and knowledge before SQL generation, rather than operating over a predefined compact schema.

Directly collecting and releasing real enterprise data, especially database values and internal documents, is often infeasible due to privacy, confidentiality, and cost constraints. To address these challenges, we propose a **cost-effective benchmark refinement framework** that builds upon existing academic benchmarks and systematically incorporates recurring structural and workflow properties observed from non-sensitive enterprise data, rather than applying random perturbations. To explicitly manage synthetic construction risks, we combine controlled construction with targeted quality verification and practitioner-oriented realism assessment. The refinement proceeds along three dimensions. At the **domain level**, we expand original databases into large, heterogeneous data-asset domains. At the **schema level**, we introduce enterprise-style structural complexity into table and column naming, and further add redundant tables and columns to better reflect realistic enterprise schemas. At the **knowledge level**, we transform query-specific knowledge snippets into large-scale external document collections, yielding a more realistic knowl-

*Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 42028–42060
October 24-29, 2026 ©2026 Association for Computational Linguistics

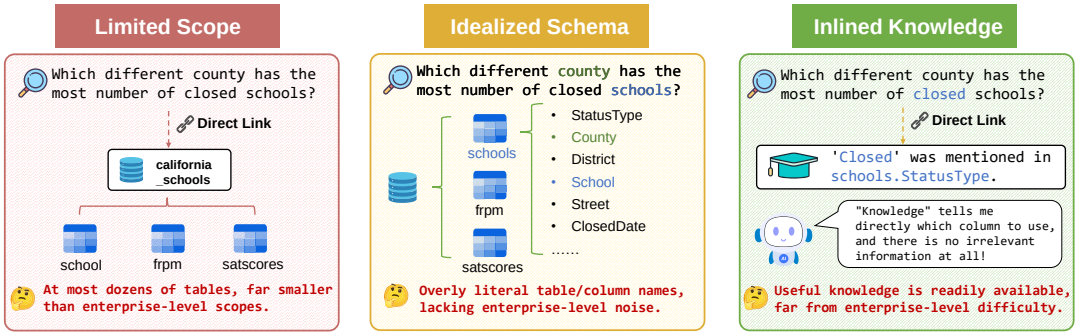

Figure 1: Limitations of existing benchmarks in reflecting real-world enterprise challenges.

edge storage environment. To make the resulting benchmarks resistant to synthetic shortcuts, the construction process adopts controlled variation strategies, detailed in Appendix B, so that superficial generation cues are unreliable for solving the task. Throughout this process, the original questions and SQL structures are preserved, and refinement is largely automated using LLMs, requiring minimal human intervention.

To further align with enterprise workflows, we formalize Dual-Retrieval-Augmented-Generation (DRAG) Text-to-SQL as an evaluation setting that makes joint table-schema and external-knowledge retrieval explicit and separately measurable from SQL generation. Applying our refinement framework to two widely used academic benchmarks, BIRD and Spider, we construct two enterprise benchmarks, **BIRD-Ent** and **Spider-Ent**. BIRD-Ent expands the query scope to an average of **4,150.3** columns and pairs each query with up to **1.5M** tokens of enterprise-style documents, while Spider-Ent expands the schema to **4,053.3** columns. Experimental results on state-of-the-art models reveal a substantial performance drop on these benchmarks, highlighting the gap between current academic performance and the structural and retrieval-driven requirements of enterprise Text-to-SQL scenarios. We therefore view the Ent-series benchmarks as a phase-1 enterprise benchmark focusing on retrieval-grounded structural challenges, a prerequisite layer for broader enterprise Text-to-SQL deployment.

Our contributions are summarized as follows:

- We formalize DRAG Text-to-SQL as a task formulation and evaluation setting that models retrieval-driven workflows commonly encountered in enterprise environments.
- We propose a low-cost benchmark refinement framework that transforms academic benchmarks into enterprise-oriented benchmarks, and apply it to construct two high-quality datasets: **BIRD-Ent** and **Spider-Ent**.
- Our benchmarks expose the substantial challenges of enterprise Text-to-SQL, with state-of-the-art models achieving only **39.1%** EX on BIRD-Ent and **60.5%** EX on Spider-Ent.

2 Task Formulation

Unlike prior settings where each question is paired with a compact database and sometimes inline knowledge, DRAG Text-to-SQL requires dual retrieval before SQL generation.

Given a question Q, the system retrieves top-k table schemas $\mathcal{C}_k$ from a large data-asset domain $\mathcal{D}$:

$$\mathcal{C}_k = \text{Retrieve}_{\text{schema}}(Q, \mathcal{D}), \tag{1}$$

and the top-n external knowledge documents $\mathcal{K}_n$ from heterogeneous document collections $\mathcal{K}$:

$$\mathcal{K}_n = \text{Retrieve}_{\text{knowledge}}(Q, \mathcal{K}). \tag{2}$$

Finally, the target SQL query Y is generated conditioned on the question and the retrieved context:

$$Y = f(Q, \mathcal{C}_k, \mathcal{K}_n \mid \theta). \tag{3}$$

where $f(\cdot \mid \theta)$ is the SQL generation model. This formulation captures the retrieval-first workflow of enterprise Text-to-SQL.

3 Benchmark Refinement

As shown in Figure 2, our benchmark refinement follows a controlled construction design to reduce exploitable synthetic shortcuts and simulates enterprise complexity through three steps: (i) Domain-level refinement expands query scopes to reflect

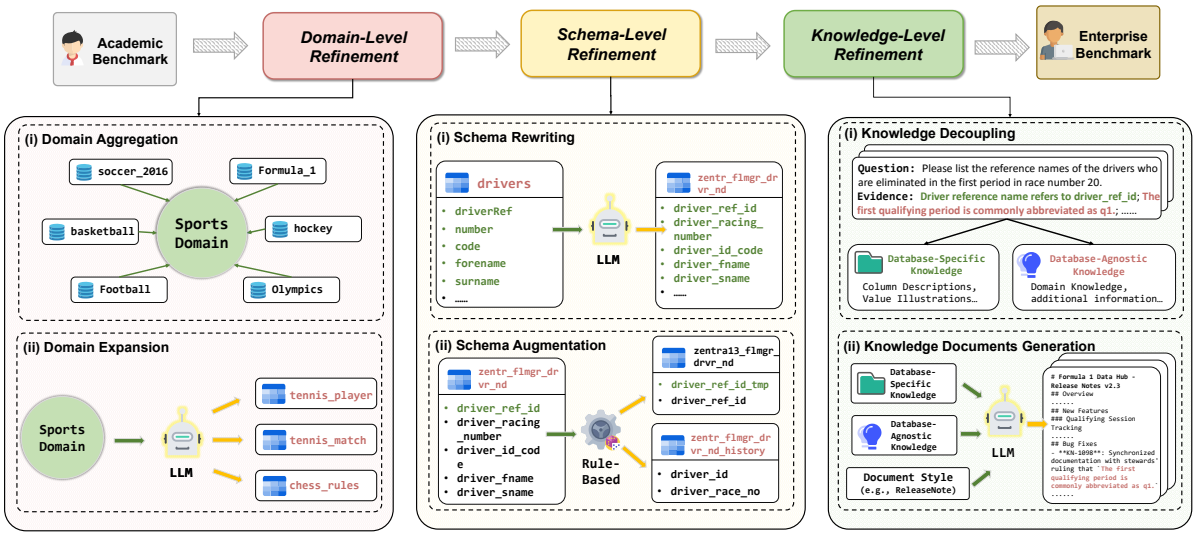

Figure 2: An overview of our benchmark refinement process.

real-world data-asset scale; (ii) Schema-level refinement introduces realistic naming and structural noise; and (iii) Knowledge-level refinement decouples query-specific knowledge and embeds it into LLM-generated enterprise documents. Our refinement targets structural and retrieval-driven enterprise challenges rather than company-specific business logic, and synthetic schemas and documents are designed to preserve enterprise-style patterns. Details of this section are provided in Appendix B.

Domain-level refinement Enterprise queries often operate over data scopes far larger than existing benchmarks (Wang et al., 2025; Eben et al., 2025). We group original databases into topical domains and expand each domain with LLM-generated tables guided by real enterprise schema exemplars, while preserving the validity of the original question–SQL pairs.

Schema-level refinement Enterprise schemas often contain hierarchical, abbreviated, and redundant names (Cai et al., 2025; Chen et al., 2025b). We rewrite table names into hierarchical conventions (<*project*>_<*area*>_<*content*>), enrich and abbreviate column names, and inject redundant tables and columns with varied prefixes, suffixes, and structural variants. These additions mimic legacy artifacts and create realistic interference for schema linking.

Knowledge-level refinement Enterprise knowledge is usually retrieved from large document collections rather than provided inline. For BIRD, we decouple query-specific evidence, keep only database-agnostic knowledge when needed, and rely on BIRD description files for database-specific content. We then expand these materials into heterogeneous enterprise-style documents, embedding required evidence in natural contexts instead of regular key–value templates. This yields 1,412 documents with about 1.5M tokens; Appendix G shows that refinement preserves semantic faithfulness without abnormal retrieval noise.

Quality control We apply targeted quality checks to ensure benchmark reliability. Specifically, we verify (i) *answer uniqueness*, ensuring each question admits a single semantically valid SQL; (ii) *semantic alignment*, confirming that questions remain consistent with their SQL after schema rewriting; and (iii) *document correctness*, ensuring generated documents preserve required knowledge without semantic drift. We manually inspect at least 15% of samples with double annotation and enterprise expert arbitration. For Spider, due to widespread underspecification in the original questions, we observe frequent loss of answer uniqueness after refinement. We therefore conduct a full manual review and retain 602 unambiguous samples. Overall, BIRD-Ent achieves 93.8% answer uniqueness, 98.4% semantic alignment, and 97.2% document correctness, while Spider-Ent reaches 94.2% answer uniqueness and 98.3% semantic alignment, indicating that our refinement preserves high benchmark quality. Detailed annotation protocols are provided in Appendix C.

Benchmark	# Example	Avg. Table Name Len.	Avg. Column Name Len.	Avg. token Zipf freq.	# Table / QS	# Col /QS	# Knowl. Tok. /Question	Hierarchical Naming	Table Retrieval	Knowledge Retrieval
Spider	2147	1.4	1.8	4.9	4.0	24.6	-	✗	✗	✗
BIRD	1789	1.2	1.9	4.7	6.8	72.6	25.0	✗	✗	✗
BEAVER	203	2.6	2.1	4.6	77.5	708.4	-	✗	✓	✗
Spider 2.0-lite	547	8.3	3.8	4.7	49.0	803.6	343.8	✓	✗	✗
Spider 2.0-snow	547	8.4	3.8	4.6	51.7	812.1	344.0	✓	✗	✗
Spider 2.0-DBT	68	3.3	2.3	4.6	21.4	337.7	1.3M	✗	-	-
Real Enterprise	324	10.2	3.5	3.9	625.8	19145.7	1.8M	✓	✓	✓
BIRD-Ent	1534	5.2	2.1	4.3	392.1	4150.3	1.5M	✓	✓	✓
Spider-Ent	602	5.2	2.1	4.3	413.0	4053.3	-	✓	✓	✗

Table 1: Data statistics comparison between different benchmarks and real enterprise data. QS denotes the query scope. Avg. table and column name lengths are measured in tokens. Avg. token Zipf frequency is computed by tokenizing table and column names and averaging their Zipf frequencies obtained from the *wordfreq* (Speer, 2022), where lower values indicate heavier use of abbreviations and domain-specific terms.

4 Empirical Study on Enterprise Realism

This section presents an empirical study to examine whether our characterization of structural and retrieval-driven enterprise challenges is accurate and whether the proposed Ent-series benchmarks reflect these targeted aspects of enterprise scenarios beyond surface-level resemblance. We conduct the study from two complementary perspectives: (i) quantitative data statistics that compare properties of benchmarks against real enterprise data, and (ii) a blind expert evaluation that assesses enterprise realism from the perspective of practitioners.

4.1 Data Statistics

We first analyze the statistical properties of existing Text-to-SQL benchmarks, our Ent-series benchmarks, and real enterprise data. The enterprise statistics are derived from a reference set collected from a data intelligence platform department of a large technology company, covering 324 enterprise data-asset domains and over 200,000 tables across diverse business lines. Table 1 summarizes the comparison results, our findings are as follows.

Query scope Real enterprise queries typically operate over hundreds of tables and tens of thousands of columns due to long-term data accumulation and cross-system integration. As shown in Table 1, real enterprise data involve over 600 tables and more than 19,000 columns per query on average. BIRD-Ent and Spider-Ent substantially narrow the gap, operating approximately 400 tables and 4,000 columns per query, whereas other benchmarks remain markedly smaller.

Schema style Beyond scale, enterprise schemas are characterized by hierarchical naming conventions, information-rich naming, and extensive use of abbreviations. Hierarchical naming conventions are directly observable in real enterprise schemas, while the latter two properties are reflected through longer name lengths and lower average token Zipf frequencies in real enterprise data. Our Ent-series benchmarks closely follow these patterns, whereas most existing benchmarks exhibit much simpler naming structures. This observation also aligns with widely adopted industry best practices for enterprise data warehouse design[1].

Knowledge organization Enterprise systems commonly rely on large, loosely structured external knowledge repositories rather than compact, query-attached evidence. BIRD-Ent models this setting by associating each query with approximately 1.5M tokens of candidate knowledge, several orders of magnitude larger than prior benchmarks.

Task paradigms In real enterprise workflows, query generation is typically preceded by both schema and knowledge retrieval. This dual-retrieval paradigm is explicitly reflected in the Ent-series benchmarks, enabling evaluation beyond SQL generation alone.

4.2 Blind Expert Evaluation

We further conduct a blind expert evaluation to assess enterprise realism from a practitioner-oriented perspective.

Evaluation setup We invite five enterprise experts with more than five years of experience in production-level data systems and Text-to-SQL applications. Their roles include data platform R&D engineers and database-intelligence product

[1] https://docs.oracle.com/cd/E38317_01/doc.11117/e37988/namingconv.htm; https://www.alibabacloud.com/help/en/dataworks/user-guide/data-warehouse-layering

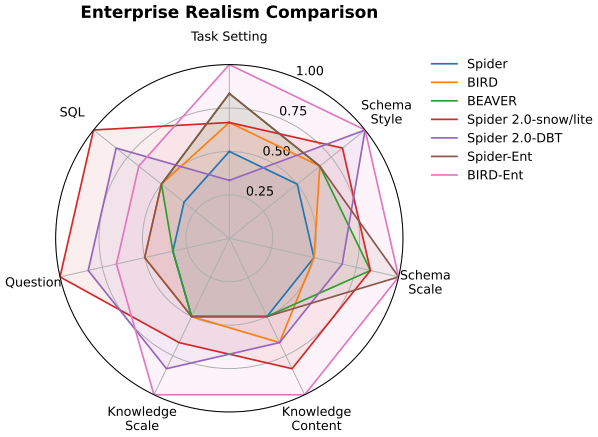

Figure 3: Enterprise realism scores from blind evaluation across seven dimensions, averaged across experts.

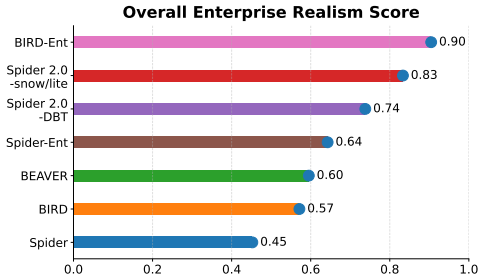

Figure 4: Overall enterprise realism score, further averaged across seven dimensions.

R&D engineers, with long-term experience in enterprise data warehouses and Text-to-SQL application development, covering data platform construction, data-warehouse engineering, schema governance, analytics workflows, and production Text-to-SQL deployment. To reduce bias, benchmarks are anonymized and presented in randomized order. Experts are asked to comparatively evaluate benchmarks along seven enterprise-related dimensions, including task setting, schema scale and style, knowledge scale and content, natural language questions, and SQL queries.

For each dimension, experts rank benchmarks by their closeness to real enterprise scenarios. Rankings are then aggregated across experts and dimensions to produce normalized rank-based scores $s = \frac{N-r}{N-1}$, where r denotes the rank of a benchmark under a given dimension and N is the total number of benchmarks. To assess inter-expert consistency, we compute Kendall's coefficient of concordance over the collected rankings and obtain $W = 0.68$ ($p = 0.013$), indicating reasonably strong agreement among experts. More details are provided in Appendix D.

Findings Figure 3 presents the per-dimension enterprise realism scores under blind evaluation. The Ent-series benchmarks consistently receive higher scores across most dimensions, particularly in task setting, schema-related and knowledge-related aspects. Figure 4 reports the overall enterprise realism score averaged across all seven dimensions. BIRD-Ent is ranked closest to real enterprise scenarios, followed by the Spider 2.0, while traditional academic benchmarks such as Spider and BIRD are consistently ranked lower.

Together with the observed inter-expert agreement, these results provide independent validation that the structural properties identified in our data statistics are also recognized by enterprise practitioners, and that the Ent-series benchmarks more accurately capture the combined characteristics of real-world enterprise Text-to-SQL within our intended scope.

5 Experiments

5.1 Implementation Details

During benchmark refinement, we use Deepseek-R1-0528 (DeepSeek-AI, 2025) to synthesize new tables, rewrite schemas, and generate documents, leveraging its strong reasoning and instruction-following capabilities.

In the main experiments, we follow a single-pass DRAG pipeline: the retrieved schemas, retrieved knowledge documents, and input question are concatenated and fed into an LLM for zero-shot SQL generation. We additionally evaluate an iterative ReAct-style pipeline in the controlled comparison reported in Table 7.

5.2 Experimental Setup

Evaluation metrics For retrieval, we report precision, recall, F1, and Perfect Recall (PR), following BEAVER (Chen et al., 2025a). PR measures whether all gold items appear in the top-k retrieved candidates, which is critical because missing any relevant table or document can block SQL generation. For SQL generation, we report Execution Accuracy (EX) (Li et al., 2023b; Lei et al., 2025).

Baselines We evaluate BM25, semantic retrievers, and a hybrid retriever (Appendix E). Semantic retrievers include Qwen3-Embedding-0.6B (Zhang et al., 2025), bge-m3 (Chen et al., 2024), and multilingual-e5-large-instruct (Wang et al., 2024); Qwen3-Embedding-0.6B is used by default. For SQL generation, we evaluate GPT-4o variants

Model	EX			
	BIRD		Spider	
	dev	**Ent**	dev†	**Ent**
GPT-4o	57.6	34.4	84.4	56.8
GPT-4o-mini	53.6	26.4	83.4	53.8
GPT-o4-mini	58.3	39.0	83.9	**60.5**
Qwen3-32B-no-thinking	53.6	25.5	82.4	47.2
Qwen3-32B	59.2	33.8	**84.7**	57.8
Qwen3-235B-A22B-no-thinking	56.3	27.8	82.8	54.0
Qwen3-235B-A22B	**59.6**	36.9	82.4	58.5
Deepseek-R1-0528	58.5	**39.1**	81.3	56.8

Table 2: EX performance of different single-pass generation baselines on the Ent-series benchmarks and their original counterparts. The Spider dev set marked with † denotes the subset retained after quality control (Section 3). Models are provided with the top-10 tables and top-5 knowledge documents retrieved to obtain optimal results.

(OpenAI et al., 2024), Qwen3 variants (Yang et al., 2025), and DeepSeek-R1-0528. Published agentic Text-to-SQL systems often rely on different tool interfaces, retrieval implementations, full-schema access, or task-specific execution environments, making direct black-box comparison difficult under the Ent-series setting. We therefore avoid cross-system numerical comparisons under incompatible assumptions and instead implement a controlled ReAct-style baseline using the same backbone and evaluation subset.

5.3 Computational Cost

Under the default single-pass DRAG setting, retrieval adds modest overhead: with Qwen3-Embedding-0.6B and preloaded FAISS indexes, online retrieval takes about 55–59 ms per query, while FAISS search itself is sub-millisecond on average. The estimated SQL-only API cost is also modest: even with GPT-4o, the average cost is only about 2 cents per BIRD-Ent query and below 0.4 cents per Spider-Ent query. Details are provided in Appendix F.

5.4 Main Results

We present our main evaluation results in Table 2, highlighting three core findings that offer deeper insights into model behavior under enterprise-level conditions.

Existing LLMs exhibit unsatisfactory performance on enterprise Text-to-SQL tasks Moving from academic to enterprise settings, EX drops sharply on both benchmarks, with BIRD decreasing by up to 28.5 pp and Spider by up to 35.2 pp. Even the EX performance of strongest models on dev sets fall to 36.9% on BIRD-Ent and to 57.8% on Spider-Ent, indicating that today's high EX scores do not translate to enterprise conditions.

Ent-series benchmarks faithfully capture the reasoning-intensive nature of enterprise scenarios Reasoning-enhanced variants such as Qwen3-32B and Qwen3-235B-A22B reduce the Ent-side drop by up to 5.8 pp on BIRD and 8.3 pp on Spider, while GPT-o4-mini shows the smallest declines overall (-19.3 pp on BIRD and -23.4 pp on Spider). In contrast, their non-reasoning counterparts degrade much more severely, underscoring that success on our benchmarks requires explicit reasoning to carefully handle massive query scopes, complex schemas, and scattered knowledge.

Ent-series benchmarks increase separation among models On BIRD, the spread between the best and worst systems grows from 6.0% on dev (59.6% vs 53.6%) to 13.6% on BIRD-Ent (39.1% vs 25.5%). On Spider, the spread expands from 3.4% on dev (84.7% vs 81.3%) to 13.3% on Spider-Ent (60.5% vs 47.2%). This larger margin reveals capability differences that are obscured by traditional settings and makes the Ent-series benchmarks suitable for tracking real progress.

5.5 More Analysis

Each of the three refinement levels independently poses substantial enterprise-level obstacles for current LLMs Table 3 shows the ablation studies to compare the effects of different refinement stages. All three refinements lead to significant performance drops, confirming that each introduces non-trivial complexity. Among them, knowledge-level refinement (Ent_K) imposes the greatest challenge on BIRD, while schema-level refinement (Ent_S) has the largest impact on Spider. Consistent with our main results, models with explicit reasoning abilities demonstrate stronger robustness under refinement, and enterprise benchmarks amplify the performance gap between models, indicating a higher discriminative power in evaluating real-world readiness.

Model	EX						
	BIRD				Spider		
	dev	**Ent**$_D$	**Ent**$_S$	**Ent**$_K$	dev†	**Ent**$_D$	**Ent**$_S$
Qwen3-32B-no-thinking	53.6	44.2 (-9.4)	41.7 (-11.9)	42.4 (-11.2)	82.4	72.8 (-9.6)	64.3 (-18.1)
Qwen3-32B	59.2	49.0 (-10.2)	49.2 (-10.0)	46.5 (-12.7)	84.7	79.3 (-5.4)	70.1 (-14.6)
Qwen3-235B-A22B-no-thinking	56.3	46.7 (-9.6)	45.2 (-11.1)	42.6 (-13.7)	82.8	75.6 (-7.2)	64.3 (-18.5)
Qwen3-235B-A22B	59.6	51.5 (-8.1)	52.5 (-7.1)	49.7 (-9.9)	82.4	80.0 (-2.4)	64.6 (-17.8)
Deepseek-R1-0528	58.5	51.0 (-7.5)	56.5 (-2.0)	49.9 (-8.6)	81.3	78.6 (-2.7)	65.6 (-15.7)

Table 3: Ablation results on EX performance across different refinement stages. Ent$_D$, Ent$_S$, and Ent$_K$ represent benchmarks after domain-level, schema-level, and knowledge-level refinement, respectively. Top-10 tables and top-5 knowledge documents retrieved are provided to the model to obtain optimal results.

Model / Method	Top-5				Top-10			
	P	R	F1	PR	P	R	F1	PR
BIRD-Ent								
Qwen3-Emb-0.6B	28.7	76.8	40.8	62.5	16.4	86.0	27.0	76.5
E5-Large-Ins	26.0	70.1	37.0	52.8	15.1	79.8	24.9	66.4
bge-m3	20.4	55.1	29.0	38.2	12.1	65.2	20.0	49.9
BM25	15.7	42.7	22.4	27.2	10.5	56.4	17.4	41.7
Hybrid	28.7	76.8	40.7	62.7	16.1	84.9	26.7	75.0
Spider-Ent								
Qwen3-Emb-0.6B	26.3	93.1	40.0	88.9	13.7	95.9	23.6	92.9
E5-Large-Ins	23.9	85.9	36.4	79.8	12.9	91.2	22.2	86.7
bge-m3	18.3	65.8	28.0	57.1	10.4	74.3	18.0	66.2
BM25	14.0	48.4	21.2	39.7	8.2	55.6	14.1	47.1
Hybrid	24.6	83.1	36.8	76.1	13.3	89.0	22.7	84.6

Table 4: Table retrieval performance across the BIRD-Ent and Spider-Ent benchmarks under different retrieval models and methods.

Model / Method	Top-5				Top-10			
	P	R	F1	PR	P	R	F1	PR
Qwen3-Emb-0.6B	29.6	46.2	34.4	14.9	20.8	62.9	30.2	31.8
E5-Large-Ins	28.7	43.9	33.1	12.8	21.1	62.2	30.4	31.8
bge-m3	21.1	31.9	24.2	7.1	14.7	43.7	21.3	14.9
BM25	30.0	47.6	35.0	14.3	19.3	59.3	28.1	25.5
Hybrid	30.3	47.4	35.2	15.8	21.2	64.2	30.8	32.7

Table 5: Knowledge retrieval performance on BIRD-Ent under different retrieval models and methods.

Existing retrievers remain insufficient for enterprise-level Text-to-SQL grounding As shown in Tables 4 and 5, consistent limitations are observed across all retrievers for both tables and knowledge documents. Overall, all retrievers achieve relatively high recall but suffer from low precision. Keyword retriever performs substantially worse, while hybrid retriever provides slight improvements over semantic retriever only in knowledge retrieval and remains limited overall, largely because schema-level refinement weakens direct lexical alignment between queries and schema elements, and knowledge-level refinement introduces additional contextual noise that interferes with keyword matching. In general, knowledge retrieval is uniformly more challenging than table retrieval; meanwhile, low PR scores in table retrieval indicate that many downstream SQL generation instances are grounded on incomplete or incorrect schemas. To separate retrieval errors from downstream reasoning difficulty, we further evaluate an oracle retrieval setting that removes missing-context errors entirely.

Model	EX	
	BIRD-Ent	Spider-Ent
Qwen3-32B-no-thinking	41.8	68.6
Qwen3-32B	46.9	70.0
Qwen3-235B-A22B-no-thinking	41.9	64.8
Qwen3-235B-A22B	50.7	64.6
Deepseek-R1-0528	51.8	65.0

Table 6: EX performance under the oracle retrieval setting, where gold-only table schemas and knowledge documents are provided without retrieval noise or distractors.

Oracle retrieval shows that the difficulty is not reducible to retrieval failure We evaluate EX performance under an oracle setting, where only gold table schemas and gold knowledge documents are provided without distractors (Table 6). This setting removes errors caused by weak retrievers or missing retrieved context, but performance remains far from solved: the best EX reaches only 51.8% on BIRD-Ent and 70.0% on Spider-Ent. Compared with the best retrieved-context results in Table 2, oracle retrieval improves EX by 12.7 pp on BIRD-Ent and 9.5 pp on Spider-Ent, confirming that retrieval quality matters. However, the remaining gap indicates that models must still reason over enterprise-style naming, redundant schemas, scattered knowledge, and SQL structure even when the

Method	Retrieval	Avg. Steps	EX
Single-pass DRAG	Static dense top-k	1.0	37.3
ReAct agent	Adaptive tool-based	5.2	20.7

Table 7: Controlled comparison on the same 150-question BIRD-Ent subset using Qwen3-235B-A22B.

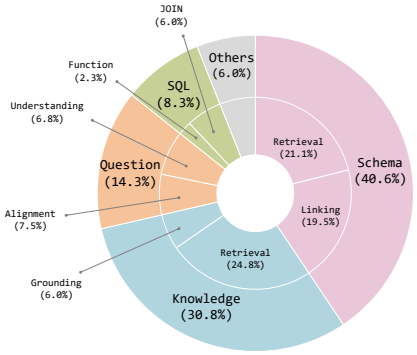

Figure 5: Error type distribution; case studies of each error type are provided in the Appendix H.

correct context is guaranteed.

A ReAct-style agent does not improve performance under the controlled setting We compare the single-pass DRAG pipeline with a ReAct-style agent on the same 150-question BIRD-Ent subset using Qwen3-235B-A22B. The single-pass pipeline uses static dense top-k retrieval, whereas the ReAct agent adaptively searches schemas and knowledge documents and iteratively refines its SQL through tool use. As shown in Table 7, ReAct obtains 20.7% EX with 5.2 steps on average, compared with 37.3% EX for single-pass DRAG. This result suggests that additional agentic interactions do not automatically alleviate the benchmark difficulty in this setting, as errors in tool selection and retrieval can accumulate over multiple steps.

6 Error Analysis

We randomly sample a total of 200 erroneous cases from BIRD-Ent and Spider-Ent for detailed error analysis, and categorize the common error types into 5 groups, the distribution of error types are shown in Figure 5.

Schema errors (40.6%) (i) Schema-retrieval errors (21.1%). Existing retrieval systems frequently fail to retrieve the tables required to answer a question due to the massive query scopes and complex schemas in our benchmarks. (ii) Schema-linking errors (19.5%). Even if all of the gold tables are recalled, the model may still make mistakes in choosing the right tables and columns.

Knowledge errors (30.8%) (i) Knowledge-retrieval errors (24.8%). The knowledge documents required to answer the question may be absent from the retrieved ones. This is the most prevalent error type, indicating that the methodology used to retrieve knowledge documents requires improvement. (ii) Knowledge-grounding errors (6.0%). Even when the gold documents are retrieved, existing models may ignore or misinterpret the knowledge in the documents. This can be attributed to the fact that useful knowledge in our benchmark is often scattered across lengthy documents.

Question errors (14.3%) (i) Question-understanding errors (6.8%). The models may misinterpret the user's question and generate an irrelevant SQL. (ii) Question-alignment errors (7.5%). The SQL generated by the models fails to align with the user's requirements. For example, the user's question asks for *ID* and *name*, while the models only return the *ID*.

SQL errors (8.3%) (i) SQL-function errors (2.3%). The models occasionally make mistakes in the use of functions. (ii) SQL-JOIN errors (6.0%). In some cases, the model may join unnecessary tables or perform joins without following foreign key constraints.

Other errors (6.0%) Apart from the above cases, we recognize some other errors such as commonsense errors and value mismatch errors. For example, the models may use *date* = '2007/01/02' to query the date, whereas the database stores the value of *date* in the format '2007-01-02'.

7 Related Work

Text-to-SQL benchmarks constructed from scratch Early benchmarks such as WikiSQL (Zhong et al., 2018) and Spider (Yu et al., 2018) establish large-scale cross-domain Text-to-SQL evaluation, with Spider introducing complex multi-table reasoning. Subsequent work improves realism by incorporating documentation or external knowledge, exemplified by KaggleDBQA (Lee et al., 2021) and BIRD (Li et al., 2023b). More recent efforts shift toward enterprise-oriented settings. BEAVER (Chen et al., 2025a) introduces data-warehouse-style databases and a RAG-based formulation, while Spider 2.0 (Lei et al., 2025) explores complex schemas and agentic task designs motivated by enterprise use cases.

Text-to-SQL benchmarks constructed by refinement To reduce annotation cost, many studies refine existing benchmarks through controlled perturbations. Representative examples include Spider-DK (Gan et al., 2021) for knowledge augmentation, Spider-Realistic (Deng et al., 2021) for lexical noise, and Dr. Spider (Chang et al., 2023) for comprehensive robustness evaluation. Other refinement directions explore new generalization axes, such as multi-turn interaction in SParC (Yu et al., 2019) and compositional generalization in Spider-SS&CG (Gan et al., 2022).

8 Conclusion

We present BIRD-Ent and Spider-Ent, two enterprise Text-to-SQL benchmarks with massive query scopes, complex schemas, and scattered knowledge, together with the DRAG Text-to-SQL paradigm. Experiments reveal large performance gaps in current models, showing that the Ent-series provides a rigorous and discriminative testbed for evaluating retrieval, grounding, and structural robustness toward enterprise Text-to-SQL deployment.

9 Limitations

Our work is constrained by practical limitations inherent to enterprise data and environments. Due to privacy and confidentiality requirements, real enterprise databases and documents cannot be released publicly. Consequently, our benchmarks rely on refined and synthesized data environments that preserve enterprise-style structural characteristics, rather than exposing raw enterprise content. We acknowledge that more open and diverse enterprise data would enable the construction of benchmarks that are even closer to real-world deployments, and we hope future community efforts can help move in this direction.

In addition, enterprise data systems vary widely across organizations and industries, and no single benchmark can capture the full spectrum of enterprise practices. Our benchmark therefore abstracts only a subset of common and recurring enterprise challenges, rather than modeling the idiosyncrasies of specific enterprises. Expanding the coverage of enterprise challenges and scenarios within a unified benchmark framework remains an important direction for future work.

Scope and intended use The benchmarks are designed to stress-test retrieval quality, schema/knowledge grounding, and structural robustness, while leaving broader deployment concerns such as business-rule reasoning, dirty data, access control, SQL dialects, and production integration to future extensions.

Acknowledgments

This work was supported by the Ant Group Research Fund.

References

Ting Cai, Stephen Sheen, and AnHai Doan. 2025. Columbo: Expanding abbreviated column names for tabular data using large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 24774–24792, Suzhou, China. Association for Computational Linguistics.

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 *The Eleventh International Conference on Learning Representations*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Mike Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025a. BEAVER: An enterprise benchmark for text-to-SQL. In *The 4th Table Representation Learning Workshop at ACL 2025*.

Yongnan Chen, Zhuo Chang, Shijia Gu, Yuanhang Zong, Mei Zhang, Shiyu Wang, Zixiang He, HongZhi Chen, Wei Jin, and Bin Cui. 2025b. ADEPT-SQL: A high-performance text-to-SQL application for real-world enterprise-level databases. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 275–283, Vienna, Austria. Association for Computational Linguistics.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Jeffrey Eben, Aitzaz Ahmad, and Stephen Lau. 2025. Rasl: Retrieval augmented schema linking for massive database text-to-sql. *Preprint*, arXiv:2507.23104.

Yujian Gan, Xinyun Chen, Qiuping Huang, and Matthew Purver. 2022. Measuring and improving compositional generalization in text-to-SQL via component alignment. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 831–843, Seattle, United States. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. 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)*, pages 2261–2273, Online. Association for Computational Linguistics.

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

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press.

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

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

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

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

Robyn Speer. 2022. rspeer/wordfreq: v3.0.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*.

Yihan Wang, Peiyu Liu, and Xin Yang. 2025. LinkAlign: Scalable schema linking for real-world large-scale multi-database text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 977–991, Suzhou, China. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren

Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Seq2SQL: Generating structured queries from natural language using reinforcement learning.

A Examples

A.1 Real-World Data Asset

Figure 6 shows an example of data asset schema from real enterprise scenarios. Sensitive information was anonymized during the review process. In real scenarios, the column and table descriptions provided in this example are often missing; we include them here solely for ease of understanding.

A.2 Ent-Series Benchmarks

A.2.1 Data Asset

Figure 7 shows an example of the data assets in our BIRD-Ent benchmark. The data assets constructed in our benchmarks simulate real enterprise assets in both naming conventions and scale. It is worth noting that real enterprise data assets' tables (see in Figure 6) often include partitioning information (e.g., *dd*) and data warehouse layering information (e.g., *dwd*). In our schema rewriting step, however, such details are omitted, as the refined benchmarks are built on SQLite databases and we are concerned that forcibly introducing them would cause confusion for benchmark users.

A.2.2 Knowledge Document

Figure 9 shows an example of an external knowledge document in our BIRD-Ent benchmark. The document is presented in an FAQ style and contains column descriptions and value illustrations for a subset of the tables. Some content is omitted in the example due to the document's length.

B Details on Benchmark Refinement

B.1 Domain-Level Refinement

To simulate the way data tables are typically stored in enterprises, we introduce a domain-level refinement. In real-world enterprise settings, data tables are usually organized by domain to facilitate topic-specific analysis by data analysts. To mimic this practice, we first leverage LLMs to classify and aggregate the original databases in BIRD and Spider by domain, thereby constructing domain-specific data assets. The resulting classification of BIRD is shown in Table 9. Since our refinement is built upon the development sets of BIRD and Spider, we omit domains that are not covered in their development sets.

After aggregation by domain, the query scope is expanded, but it still falls far short of the enterprise scale, where a single data asset may comprise hundreds of tables. To bridge this gap, we perform domain expansion, prompting the LLM to generate new databases within a given domain. The tables from these newly generated databases are then added to the corresponding domain assets. We generate entire databases before extracting their tables, rather than generating tables directly, in order to ensure inherent logical connections among the tables, reflecting enterprise reality, where some tables within a data-asset domain may originate from the same project or database. Moreover, in designing the prompts, we provide the model with the databases already present in the data-asset domain to prevent semantic collisions, which could otherwise lead to non-unique answers in the original datasets and degrade the quality of the final benchmarks. We also include examples of real enterprise table schemas in the prompt to guide the model in mimicking authentic schema styles. These schemas are randomly sampled from over 200,000 tables in real-world enterprise environments, ensuring diversity. The reference schemas are used only as structural exemplars for scale, naming style, data types, and constraints; table contents, values, and business-specific logic are not copied into the released benchmarks. During expansion, exemplars are combined with the topical databases already present in each domain, so the generated assets follow domain-specific semantics rather than a single reusable enterprise template. An example prompt for the domain expansion step is shown in Figure 16.

It is worth noting that we do not generate any sample values for the synthetic tables, nor do we verify their logical consistency or construct corresponding databases. This is because the original question-SQL pairs do not reference these tables, and their execution results are unaffected by them. The schemas of the synthetic tables serve as pseudo-schemas, they are designed solely to simulate enterprise scenarios by expanding the domain and introducing challenge to the schema linking process.

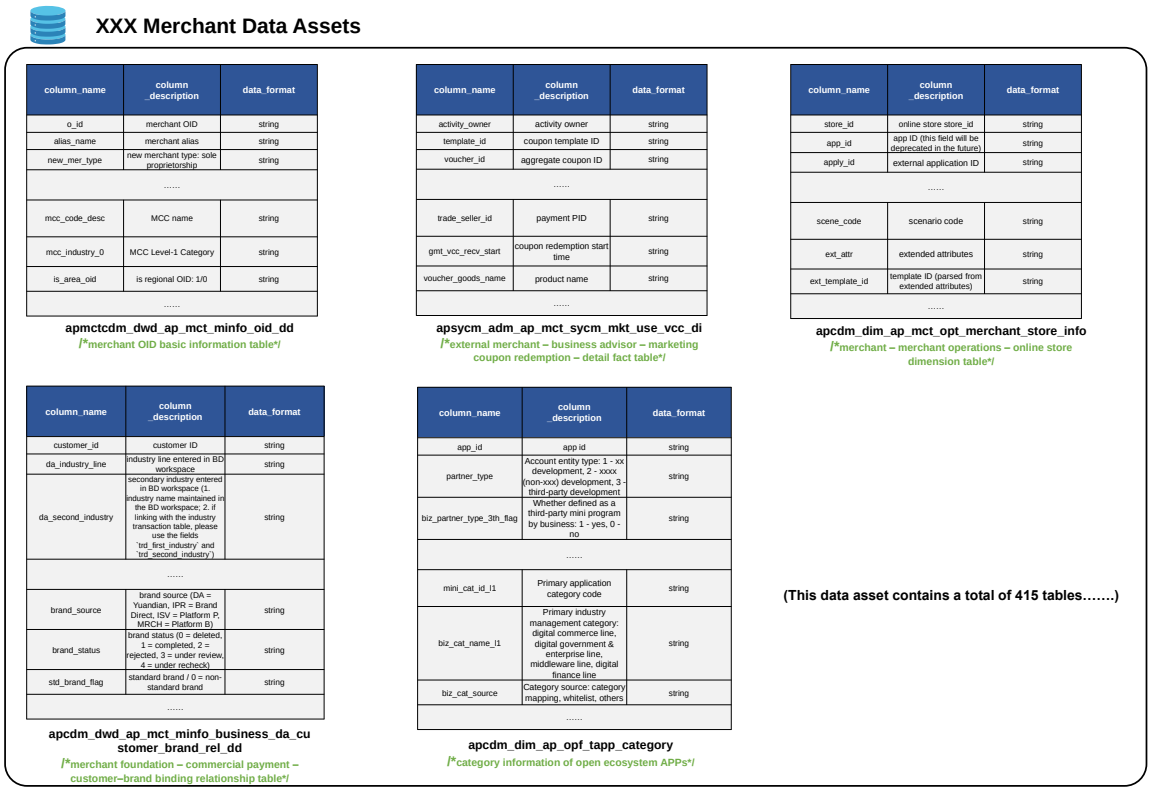

Figure 6: An example of enterprise data assets.

	Table	Column
Suffix or Prefix	bak, hist, drop, tmp, mid, snapshot, history	backup, tmp, migrated,legacy, deprecated

Table 8: Common Suffixes and Prefixes Summary.

B.2 Schema-Level Refinement

At the schema level, the complexity of academic benchmarks is generally not enough compared to real-world enterprise databases. The enterprise-level database fields are often more intricate and widely abbreviated. In addition, in enterprise systems, the processes of maintenance and iterative updates frequently generate historical, temporary, or backup tables and columns. Although they are no longer actively used, they remain within the system and often exhibit strong similarities to certain existing tables. We collectively call them redundant tables and columns.

To better simulate these real-world challenges, we propose schema rewriting and schema augmentation strategies for academic benchmarks at the schema level. The first step is schema rewriting. For table names, enterprise scenarios often have naming conventions. Motivated by common enterprise practices, we propose a hierarchical naming convention <*project*>_<*area*>_<*content*> to simulate the enterprise environment. In this convention, <*project*> refers to the id of the project in enterprise scenarios; <*area*> is a generalization of the business area to which a table belongs; <*content*> is a summary of the content of the table. To transform the original table names into the desired format, we first provide LLM with available database information to generate the project and area name. Then, the model produces an abbreviated representation of the original table name and concatenate it with the previously generated project and domain names to construct a fully structured table name. The prompt of table name rewriting is shown in Figure 17 and Figure 18. For column names, considering the high degree of abbreviation observed in enterprise schema, we propose a CoT process to rewrite column names by using LLM. First, we generate a concise, domain-aware contextual summary for each database using available schema information and database descriptions (when present). Its prompt can be seen in Figure 19. Second, by us-

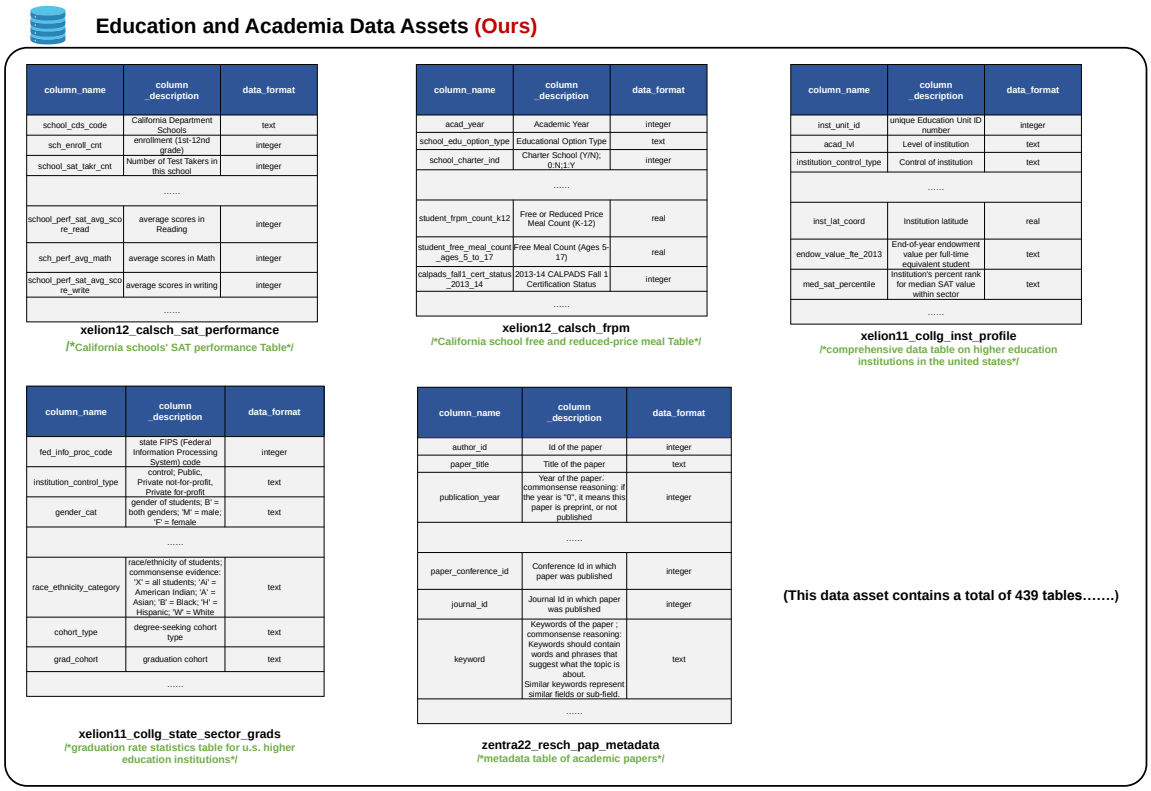

Figure 7: An example of BIRD-Ent's data assets.

ing the domain context together with the columns' descriptions within its table schema, we expand the column names. Its prompt can be seen in Figure 20. Last, to emulate enterprise terseness, we will abbreviate the column names with LLM. Details prompt is illustrated in Figure 21. For example, in the *card game* database, the table name *card* is rewritten as *zentra11_mcard_crd_catalog*, where *zentra11* is the project id of the enterprise, *mcard* represents the table belongs the card game area, and *crd_catalog* indicates that the table contains card catalog information, and the column name *power* (representing the power value of a card) is rewritten as *card_pwr_val*, a longer, more specific, and more heavily abbreviated field name.

The injection of redundant tables and columns (i.e., schema augmentation) is achieved in the following way: We randomly select a subset of tables from each database and, following the table-naming convention, generate redundant table names by appending common noise suffixes or prefixes. For each selected table, we clone its schema to create the corresponding noise table. Then we randomly add some columns related to the table and remove a few columns that were not primary keys or foreign keys from this table to simulate the subtle differences between the noise table and the existing table caused by database evolution. Finally, we append the noise columns, which are created by copying the existing column and adding common noise suffixes or prefixes to the original table. We control the ratio of redundant tables to original tables to be 1:4 to simulate the real environment of the enterprise. Common prefixes and suffixes of redundant tables and columns are shown in Table 8.

B.3 Knowledge-Level Refinement

BIRD acknowledges the necessity of external knowledge in Text-to-SQL tasks, since user queries are often concise and may naturally omit information that is crucial for answering the question (e.g., domain-specific knowledge or descriptions of database contents). However, in BIRD, such knowledge is directly appended as part of the query, which is unrealistic in enterprise scenarios where relevant information must instead be retrieved from large-scale external document collections. Moreover, much of the external knowledge in BIRD is overly simplistic, which rarely appears in enterprise environments. To address these issues, our knowledge-level refinement consists of two steps: knowledge cleaning and decoupling, and document generation.

Benchmark	Domain	Databases
BIRD	retail_and_e-commerce	car_retails, retails, retail_world, regional_sales, retail_complains, sales, sales_in_weather, superstore, debit_card_specializing
	finance_and_economy	coinmarketcap, financial
	education_and_academia	authors, books, book_publishing_company, citeseer, college_completion, university, computer_student, cs_semester, language_corpus, student_loan, california_schools, student_club
	data_science_and_technology	codebase_comments, image_and_language, talkingdata, codebase_community
	healthcare_and_bioinformatics	synthea, genes, mental_health_survey, thrombosis_prediction, toxicology
	sports_and_athletes	hockey, ice_hockey_draft, european_football_1, soccer_2016, olympics, professional_basketball, european_football_2, formula_1
	entertainment_and_media	disney, movie, movies_4, movie_3, movielens, movie_platform, simpson_episodes, law_episode, shakespeare, video_games, superhero, card_games

Table 9: Domain aggreation results of the BIRD databases.

B.3.1 Knowledge Cleaning and Decoupling

In the cleaning step, we manually examined the external knowledge provided in BIRD and categorized its types (as illustrated in Figure 10). This included redundant knowledge such as commonsense facts, basic arithmetic, fundamental SQL syntax, trivial synonyms, and simple reasoning, all of which were identified and removed.

column_name	Column _description	data_format	Value _description
Id	Id of the paper	integer	
Title	Title of the paper	text	
Year	Year of the paper	integer	commonsense reasoning: if the year is "0", it means this paper is preprint, or not published
ConferenceId	Conference Id in which paper was published	integer	
JournalId	Journal Id in which paper was published	integer	commonsense reasoning: If a paper contain "0" in both ConferenceID and JournalId, it means this paper is preprint

Figure 8: An example of a BIRD database description file (for the *Paper* table in the *authors* database)

For the remaining non-redundant knowledge, our analysis revealed two main categories: database-specific and database-agnostic. Database-specific knowledge typically consists of column descriptions and value illustrations, most of which originate from the database description documents released alongside BIRD datasets. As exemplified in Figure 8, these documents cover nearly all columns across BIRD's databases, including their column descriptions, types, and value illustrations. We merged this category of knowledge with the corresponding database descriptions, treating them as source material for generating external knowledge documents, which more closely reflects enterprise settings where such information is typically scattered across heterogeneous documents.

Database-agnostic knowledge can be further divided into query-dependent and query-independent. Query-dependent knowledge supplements a specific question but is invalid in isolation. For example, as shown in Figure 10, the same noun (*Artifact cards*) may refer to different columns (*types* =' Artifact' or *originalType* = 'Artifact') across queries; such knowledge only makes sense when tied to the query context. Therefore, we attach it directly to the original query rather than treating it as external knowledge. In contrast, query-independent knowledge remains valid outside the original query context. Owing to its generality, this category is retained as additional source material for generating external knowledge documents.

B.3.2 Document Generation

The pipeline for generating knowledge documents is illustrated in Figure 11. In the first step, we segment the database description documents into chunks, each containing several columns along with their names, descriptions, types, value illustrations, and sampled values. These chunks are then

combined with the independent external knowledge associated with the corresponding columns to form the raw external knowledge.

In the second step, we randomly select one of the predefined enterprise document genres (see in Table 16) and prompt DeepSeek-R1-0528 to generate a document in the chosen genre, where the raw external knowledge is naturally and contextually embedded. To facilitate subsequent inspection and correction, we further instruct the LLM to annotate the generated text with special tags for each piece of knowledge content: $< ocn_i >$ for column names, $< cd_i >$ for column descriptions, $< vd_i >$ for value illustrations, and $< ek_i >$ for independent external knowledge. These annotations align the knowledge with standardized tags, enabling us to locate and verify the correctness of the embedded content. The prompt for this step is shown in Figure 22.

In the third step, we perform rule-based checks on the generated documents to ensure: (i) all required tags are present, (ii) tags follow the correct format, and (iii) no unintended tags are included. If violations are detected, an $< error_mark >$ tag is appended to the corresponding content, preparing the documents for LLM-based self-correction.

In the fourth step, we provide the annotated knowledge documents, the raw external knowledge, and the document generation rules as inputs to the LLM, which is tasked with correcting the documents. The prompt used for this step is illustrated in Figure 23. In the quality control step, we also leverage these pre-generated tags when inspecting the knowledge documents.

Through this elaborate pipeline, we accurately transform BIRD's external knowledge into a large collection of enterprise-style knowledge documents.

C Human Verification Protocol for Benchmark Quality

This section describes the human verification protocol adopted to ensure the quality of the refined benchmarks. We employ a double-blind annotation with an expert-in-the-loop strategy to validate three quality indicators, namely answer uniqueness, semantic alignment, and document correctness. The design of this protocol aims to minimize individual annotator bias, ensure consistency in ambiguous cases, and provide a transparent and reproducible quality assurance process for benchmark construction and future updates.

C.1 Annotation Workflow

The verification process is carried out by two groups of annotators. The student group consists of two master's students with strong SQL expertise and two undergraduate students, all of whom are responsible for independent double-blind annotation. In addition, an enterprise expert with extensive experience in real-world data systems and Text-to-SQL applications serves as the expert group, acting as the final arbiter in cases of disagreement. Prior to annotation, student annotators are provided with unified guidelines and calibration examples to align their understanding of the task and decision criteria.

For each sampled instance, a complete verification context is constructed and provided to annotators. This context includes the natural language question, the schemas of the top-20 tables ranked by semantic similarity (ensuring coverage of the gold tables), the external knowledge required to answer the question, and the gold SQL query together with its execution result when execution is available. Each instance is randomly assigned to two annotators from the student group, who work independently and do not have access to each other's judgments. If the two annotations are consistent, the decision is directly accepted. If they differ, the instance is escalated to the enterprise expert, who reviews both annotations and makes the final determination after discussion.

C.2 Quality Indicators

The first quality indicator, answer uniqueness, evaluates whether a question admits exactly one reasonable SQL answer under the provided context. Annotators are instructed to enumerate all SQL queries that they consider semantically valid for answering the question, rather than stopping at the first plausible solution. If both annotators independently identify exactly one SQL query and its execution result matches that of the gold query, the instance is marked as having a unique answer. All other cases are labeled as non-unique, including situations where annotators propose different numbers of valid SQL queries, obtain different execution results, or identify multiple semantically valid interpretations of the question. Disagreements on answer uniqueness are resolved by the expert, who determines whether alternative SQLs reflect genuine ambiguity in the question or stem from incor-

Quality Indicator	Agreement	Gwet's AC1
Answer uniqueness	0.884	0.866
Semantic alignment	0.957	0.955
Document correctness	0.926	0.919

Table 10: Inter-annotator agreement for the three benchmark quality indicators. Document correctness is evaluated on BIRD-Ent.

rect reasoning.

The second indicator, semantic alignment, verifies whether the SQL query correctly captures the intent of the natural language question. This validation is primarily conducted through execution-based checking when possible, complemented by semantic reasoning when execution is unavailable. Annotators examine whether the SQL selects the correct entities, applies appropriate filters and aggregation operations, and follows the intended join logic implied by the question. An instance is labeled as semantically aligned if the SQL accurately reflects the question's intent and yields the correct result; otherwise, it is marked as misaligned. For annotators proficient in SQL, this verification is generally straightforward, as semantic errors often manifest clearly in either execution results or mismatches between query structure and question intent.

The third indicator, document correctness, assesses the reliability of the refined external knowledge documents. Annotators check whether the generated documents faithfully preserve the original external knowledge required for answering the question and whether they introduce any contradictory statements. This includes verifying that key facts, rules, or descriptions from the source knowledge are correctly retained, while ensuring that no conflicting or misleading information is introduced during document expansion. An instance is marked as correct if the refined document contains the necessary knowledge without contradiction; otherwise, it is labeled as incorrect. This verification is manageable for annotators with formal English education and is aided by translation tools when specialized terminology is involved.

All cases involving annotator disagreement are logged together with the expert's final decision and a brief rationale. These records are used to refine annotation guidelines and to identify common sources of ambiguity, such as underspecified questions or overlapping schemas.

C.3 Inter-Annotator Agreement and Rejected Examples

To quantify annotation reliability, we compute pairwise raw agreement and Gwet's AC1 on the independently annotated samples before expert adjudication. As shown in Table 10, all three quality indicators exhibit consistently high agreement.

We further provide representative examples rejected during quality control.

Answer uniqueness The Spider question "Count the number of documents" becomes ambiguous after databases are pooled into a document-management data asset. Multiple tables represent document-like entities, such as `zentor1_docex_doc_metadata` and `proj_x7_docmgmt_doc_objects`, allowing multiple semantically plausible SQL queries with different results.

Semantic alignment In one rejected case, a column representing a singer's name is rewritten as `venue_name`, although its sampled value is "Joe Sharp." The rewritten schema therefore suggests a venue rather than a person and no longer preserves the original column semantics.

Document correctness In another rejected case, a generated document states that the eligible-free rate is already materialized in the column *Percent (%) Eligible Free (K–12)* and can be queried directly. However, the original evidence defines the value as the ratio of *Free Meal Count* to *Enrollment*. The generated statement therefore conflicts with the source knowledge.

C.4 Annotation Cost Analysis

Although our benchmark construction involves human verification, the associated annotation cost is substantially lower than that of traditional Text-to-SQL benchmarks. This is primarily because the most labor-intensive stages in benchmark construction, including database collection, natural language question writing, and gold SQL annotation, incur almost zero manual cost in our pipeline. Instead, human effort is concentrated solely on targeted quality verification of a small sampled subset.

To make this comparison concrete, we quantify the actual human annotation cost and contrast it with Spider 1.0, which is one of the few benchmarks that explicitly reports annotation effort. Other benchmarks largely follow similar fully

Benchmark	Task	#Items	Cost / Item (person-hour)	Total Cost (person-hour)
BIRD-Ent	Answer uniqueness	256	0.17	43.5
	Semantic alignment	256	0.07	17.9
	Document correctness	282	0.10	28.2
	Total	–	–	**89.6**
Spider-Ent	Answer uniqueness	1034 + 120	0.10	115.4
	Semantic alignment	120	0.05	6.0
	Document correctness	–	–	–
	Total	–	–	**121.4**
Spider 1.0	**Total**	–	–	**1100.0**

Table 11: Comparison of human annotation cost (in person-hours) between our benchmarks and Spider 1.0.

manual annotation and validation paradigms as Spider 1.0, making it a representative reference point.

Table 11 summarizes the annotation cost measured in person-hours. For our benchmarks, costs are broken down by quality indicator. The reported cost per item reflects the average time spent by annotators, including double-blind annotation and expert arbitration when needed.

The results show that our benchmarks require an order of magnitude less human effort than Spider 1.0. This reduction stems from our design choice to avoid fully manual data creation and instead rely on automatic construction followed by targeted human verification. As a result, we achieve strong quality guarantees while keeping annotation cost low, making the benchmark construction process scalable and practical for future extensions.

D Blind Expert Evaluation of Enterprise Realism

This section describes the blind expert evaluation conducted to assess how closely different Text-to-SQL benchmarks resemble real-world enterprise scenarios. The evaluation provides an external, practitioner-oriented validation of enterprise realism.

D.1 Expert Recruitment and Evaluation Setting

During the preliminary investigation stage, prior to formal benchmark construction, we conducted in-depth discussions with multiple enterprise experts and collected a large number of real-world Text-to-SQL use cases. These discussions informed our characterization of enterprise challenges and were validated against concrete examples encountered in practice.

After the benchmarks were constructed, we further invited five enterprise experts to participate in a blind evaluation. All experts have hands-on experience with enterprise data systems and Text-to-SQL-related tasks in production environments. To reduce potential bias, the experts were not informed of the benchmark identities or their construction details during evaluation; instead, benchmarks were anonymized and presented in a randomized order.

D.2 Evaluation Dimensions and Questionnaire Design

The evaluation assesses benchmark realism along seven dimensions that are repeatedly emphasized in enterprise practice:

- **Task setting**: whether the overall task formulation aligns with real enterprise workflows.
- **Schema scale**: whether the number of tables and columns reflects enterprise-scale data assets.
- **Schema style**: whether naming conventions, abbreviations, and structural patterns resemble enterprise schemas.
- **Knowledge scale**: whether the amount of external knowledge matches enterprise environments.
- **Knowledge content**: whether the form and organization of knowledge are realistic.
- **Natural language questions**: whether user questions resemble those encountered in enterprise usage.
- **SQL queries**: whether the generated SQL reflects enterprise-style query logic.

The questionnaire consists primarily of Likert-style comparative questions, asking experts to judge each benchmark relative to others along each

dimension rather than assigning absolute scores. Open-ended comment fields are also provided to allow experts to explain their judgments or note specific strengths and weaknesses.

D.3 Ranking and Scoring Method

For each dimension, experts rank the benchmarks according to their closeness to real enterprise scenarios, with lower ranks indicating higher realism. To obtain comparable numerical scores, each rank r is converted into a normalized rank-based score using

$$s = \frac{N - r}{N - 1},$$

where N denotes the total number of benchmarks. This linear reverse normalization maps the best-ranked benchmark to a score of 1 and the worst-ranked benchmark to 0, emphasizing relative enterprise realism rather than absolute differences.

The normalized scores are then averaged across all experts and evaluation dimensions to produce a single overall enterprise realism score for each benchmark. These aggregated scores are used for the empirical analyses and visualizations reported in the main paper.

E Hybrid Retriever Implementation

To systematically study the performance of different retrieval paradigms, we implement a hybrid retriever that combines sparse lexical retrieval and dense semantic retrieval in a two-stage pipeline. The retriever first uses BM25 as a high-recall candidate generator, followed by embedding-based re-ranking within the BM25 candidate set.

In the first stage, we retrieve the top-250 tables and top-500 knowledge documents using BM25, achieving recall rates of 96.3% and 92.9%, respectively. These cutoffs are determined empirically: increasing the BM25 top-k beyond these values yields diminishing recall improvements while incurring higher computational costs. We therefore select BM25 candidate sizes that maintain consistently high recall with minimal redundancy.

In the second stage, we perform dense retrieval using Qwen3-Embedding-0.6B. The query is encoded into a dense vector, and similarity search is conducted only over the embeddings of the BM25-retrieved candidates. This design preserves the strong lexical coverage of BM25 while allowing the dense retriever to focus on semantic discrimination within a restricted candidate pool, which is critical under enterprise-scale corpora.

Overall, this hybrid retrieval pipeline provides a recall-oriented first stage and a precision-oriented second stage, offering a computationally efficient and modular baseline for comparing keyword, semantic, and hybrid retrieval strategies in enterprise-level Text-to-SQL settings.

F Computational Cost Analysis

This section analyzes the computational cost of the DRAG evaluation pipeline. We focus on three practical factors: retrieved-context size, online retrieval latency, and estimated LLM inference cost. Unless otherwise specified, the analysis follows the default evaluation setting used in the main experiments: top-10 schema retrieval for both BIRD-Ent and Spider-Ent, and top-5 knowledge-document retrieval for BIRD-Ent.

Retrieved-context size Table 12 reports the token statistics of the retrieved context. The schema context is computed from the formatted top-10 retrieved table schemas, including the same topic and schema headers used in our evaluation prompt. The document context is computed from the formatted top-5 retrieved documents on BIRD-Ent. Results show that the retrieved context remains bounded by the top-k design. On average, BIRD-Ent uses about 2.0k schema tokens and 5.7k document tokens, while Spider-Ent uses about 1.3k schema tokens.

Online retrieval latency We measure per-query online retrieval latency using Qwen3-Embedding-0.6B and FAISS indexes. The model and indexes are preloaded before measurement, so the reported latency excludes model loading and index loading. Table 13 separates query embedding time from FAISS search time. The results show that FAISS search itself is consistently sub-millisecond on average. The main online cost comes from query embedding, and the full encode-plus-search latency is around 55–59 ms per query.

Estimated LLM inference cost Table 14 estimates the average per-query LLM inference cost for three representative models. Input tokens are computed from the reconstructed evaluation prompts. Output tokens are estimated using the reference SQL wrapped in the required `<answer>` format, since the evaluation prompt instructs models to output only SQL. We do not apply prompt-cache discounts. For reasoning models, this should be interpreted as a SQL-only lower-bound estimate,

Benchmark	Retrieved Context	N	Mean	P50	P75	P90	P95	Max
BIRD-Ent	Schema top-10	1534	2012.6	1835.5	2341.8	3587.0	3715.0	5747.0
BIRD-Ent	Docs top-5	1534	5725.4	5712.5	6148.5	6549.9	6806.0	10382.0
Spider-Ent	Schema top-10	602	1276.8	1244.0	1397.8	1594.1	1781.0	2177.0

Table 12: Token statistics of retrieved contexts under the default DRAG setting. Tokens are counted with the Qwen3-Embedding-0.6B tokenizer.

Benchmark	Retrieval	N	Encode Mean (ms)	Search Mean (ms)	Total Mean (ms)	Total P95 (ms)	Total Max (ms)
BIRD-Ent	Schema top-10	1534	55.43	0.18	55.61	65.86	94.31
BIRD-Ent	Docs top-5	1534	57.94	0.57	58.52	65.84	88.16
Spider-Ent	Schema top-10	602	55.28	0.20	55.48	65.36	71.92

Table 13: Per-query online retrieval latency with Qwen3-Embedding-0.6B and preloaded FAISS indexes. "Encode" denotes query embedding time, "Search" denotes FAISS top-k search time, and "Total" denotes their sum.

because additional chain-of-thought or reasoning tokens would increase output cost.

Overall, these results indicate that DRAG introduces measurable but controlled online overhead. Retrieval cost is dominated by query embedding rather than FAISS search, and the LLM-side cost is mainly determined by the bounded top-k retrieved context. In practice, these costs can be further mitigated through prebuilt indexes, resident embedding models, parallel schema and knowledge retrieval, caching, and context compression.

G Embedding Similarity Analysis of Knowledge Refinement

To analyze how knowledge-level refinement alters the semantic properties of external knowledge, we conduct an semantic similarity analysis between the original BIRD knowledge and their refined counterparts. In this analysis, the original BIRD knowledge is treated as the initial document set, while the refined documents are obtained through our knowledge refinement pipeline. All documents are encoded using Qwen3-Embedding-0.6B, and cosine similarity is used throughout.

G.1 Similarity Between Original Knowledge and Their Refined Counterparts

We first compute the cosine similarity between each original document and its corresponding refined version. This analysis directly measures the degree to which refinement preserves the core semantics of the source document. The average similarity is 0.65 with a standard deviation of 0.08, indicating a moderate but consistent semantic shift.

These results suggest that the refined documents largely inherit the core semantic content of the original knowledge, while introducing additional enterprise-style narrative and contextual elaboration. The similarity distribution shows no evidence of semantic drift or meaning replacement, supporting the semantic faithfulness of the refinement process.

G.2 Pairwise Similarity Distribution Within Document Sets

To examine whether refinement introduces abnormal redundancy or reduces semantic diversity, we further analyze the pairwise semantic similarity distributions within each document set. Specifically, we compute cosine similarity for all document pairs among the original documents and among the refined documents, respectively. The results are reported in Table 15

The results reveal several notable patterns. First, the average pairwise similarity decreases from 0.46 to 0.32 after refinement, indicating that refinement increases semantic dispersion rather than introducing excessive similarity. Second, the probability of observing highly similar document pairs ($\text{sim} > 0.8$) in the refined set is nearly zero, suggesting that refinement does not create large clusters of near-duplicate documents. Finally, the standard deviation remains comparable across the two sets, implying that refinement does not collapse semantic diversity.

Overall, these findings demonstrate that knowledge-level refinement is not a form of noise injection. Instead, it performs controlled textual restructuring that expands document length and contextual breadth while preserving core semantics and maintaining a healthy diversity of knowledge representations.

Model	Benchmark	Avg. Input Tok.	Avg. Output Tok.	Avg. Cost (USD)	Avg. Cost (cents)
GPT-4o	BIRD-Ent	7681.4	55.5	0.019759	1.9759
GPT-4o	Spider-Ent	1416.2	34.9	0.003890	0.3890
Qwen3-235B-A22B	BIRD-Ent	8041.2	57.0	0.003762	0.3762
Qwen3-235B-A22B	Spider-Ent	1460.9	35.2	0.000729	0.0729
DeepSeek-R1-0528	BIRD-Ent	8006.5	59.5	0.004534	0.4534
DeepSeek-R1-0528	Spider-Ent	1554.0	36.5	0.000935	0.0935

Table 14: Estimated average LLM inference cost per SQL generation query. Prices are queried on May 26, 2026: GPT-4o uses OpenAI pricing, Qwen3-235B-A22B uses OpenRouter pricing, and DeepSeek-R1-0528 uses DeepSeek `deepseek-reasoner` cache-miss pricing.

Document Set	Average	Std	$\mathbf{P}(\text{sim} > \mathbf{0.8})$
Original knowledge	0.46	0.07	0.0030
Refined documents	0.32	0.09	0.0001

Table 15: Pairwise cosine similarity statistics within original and refined knowledge document sets.

H Case Study

Figure 12 illustrates a typical schema-retrieval error. The failure of the predicted SQL stems from the fact that not all target tables were included in the retrieved schema. When the model is not provided with the complete set of relevant tables, it becomes exceedingly difficult to generate a correct SQL query.

Figure 13 reflects another type of schema error. In this case, although the target table is all retrieved in the previous stage, the model does not select all the right tables and columns. This situation may be caused by the semantic similarity among tables and columns within the same domain and the interference of redundant tables. In the error analysis, such cases are not uncommon, suggesting that existing models may still exhibit limitations in their sensitivity to distinguishing among different tables and columns.

Errors can also occur in the knowledge part. In some cases, if you want the model to generate a correct SQL in the specific areas, you need to supply the model with some specific knowledge. In Figure 14, the knowledge is the information that the model needs to gain from the retrieved documents. However, the model selected the wrong document library. Therefore, it did not obtain the correct knowledge, leading to subsequent SQL execution errors. This observation indicates that, beyond table retrieval, knowledge retrieval also poses a considerable challenge for the model.

In Figure 15, although the predicted SQL aligns with the gold SQL in terms of overall structure, it overlooks the critical rules regarding group references. This indicates that the model either neglects the relevant knowledge or exhibits deficiencies in its understanding of such knowledge.

Geographical Organization Data Management FAQ

```
# 🌍 Geographical Organization Data Management FAQ

> **Introduction**
This document addresses common questions about managing geographical organization data in our global reference
system. It covers data definitions, usage scenarios, and troubleshooting for tables tracking international
organizations and their memberships.
🔷 *New team members should review this before interacting with the `zentra25_georef` dataset.*

---

## 🏷️ Organization Metadata

### **Q1: How do we uniquely identify organizations in our system?**

We use two complementary identifiers:
- `organization_abbrev`: Stores the organization's abbreviation (e.g., **ASEAN** for the Association of
Southeast Asian Nations).
- `city_name`: Records the **full name** of the organization.

✅ Always cross-reference both fields when verifying organizational identity.

---

### **Q2: Where is headquarters location data stored?**

Headquarters details are captured through three fields:
- `City`: The city where the headquarters are located.
- `province`: The province/state/region of the headquarters.
- `country_code`: The country code (ISO 3166) where the headquarters are located.

⚠️ **Important**: Always validate province–country combinations to avoid mismatches like *"California,
Canada"*.


……(Some content of the document is omitted here)……


🛠️ Steps to resolve:
1. Verify spelling of `Organization` against `organization_abbrev`
2. Check `ORG_HISTORICAL_ALIAS` for legacy codes
3. If unresolved, run script: `OrgIntegrityCheck.py`

---

### **Scenario 2: Discrepancy between `city_name` and physical headquarters location**

🧠 Remember:
- `city_name` = Full **organization name**, NOT a geographic location.
- Physical HQs are only in the `City` field.

🚩 Action:
- Flag any entry where `city_name` includes place names (e.g., `"Nairobi Agreement Council"`) for correction.

---

### **Scenario 3: Missing Type classifications in membership records**

❗ Since `Type` lacks formal definition:
- Do **not** infer meaning from blank or populated values
- Tag affected records with `TYPE_UNDEFINED` in audit logs
- Escalate per **Section 4.3 of the Data Governance Policy**

---

## 📚 Appendix: Change History

| Date       | Change Description                                    | Ticket/Ref       |
|------------|-------------------------------------------------------|------------------|
| 2023-11    | Deprecated `Region` field                             | GLB-223          |
| 2022-08    | Rejected `Membership_Fee` field proposal              | RFC-771          |
| 2020-05    | Province naming standardization                       | DATA-441         |
| 2018-02    | Country code standardization (FIPS → ISO 3166)        | COMPLIANCE-98    |

---

> ℹ️ This document is maintained by the Global Reference Data Team.
> Last updated: Based on changes up to **2023-11**.
```

Figure 9: An example of BIRD-Ent's external knowledge document.

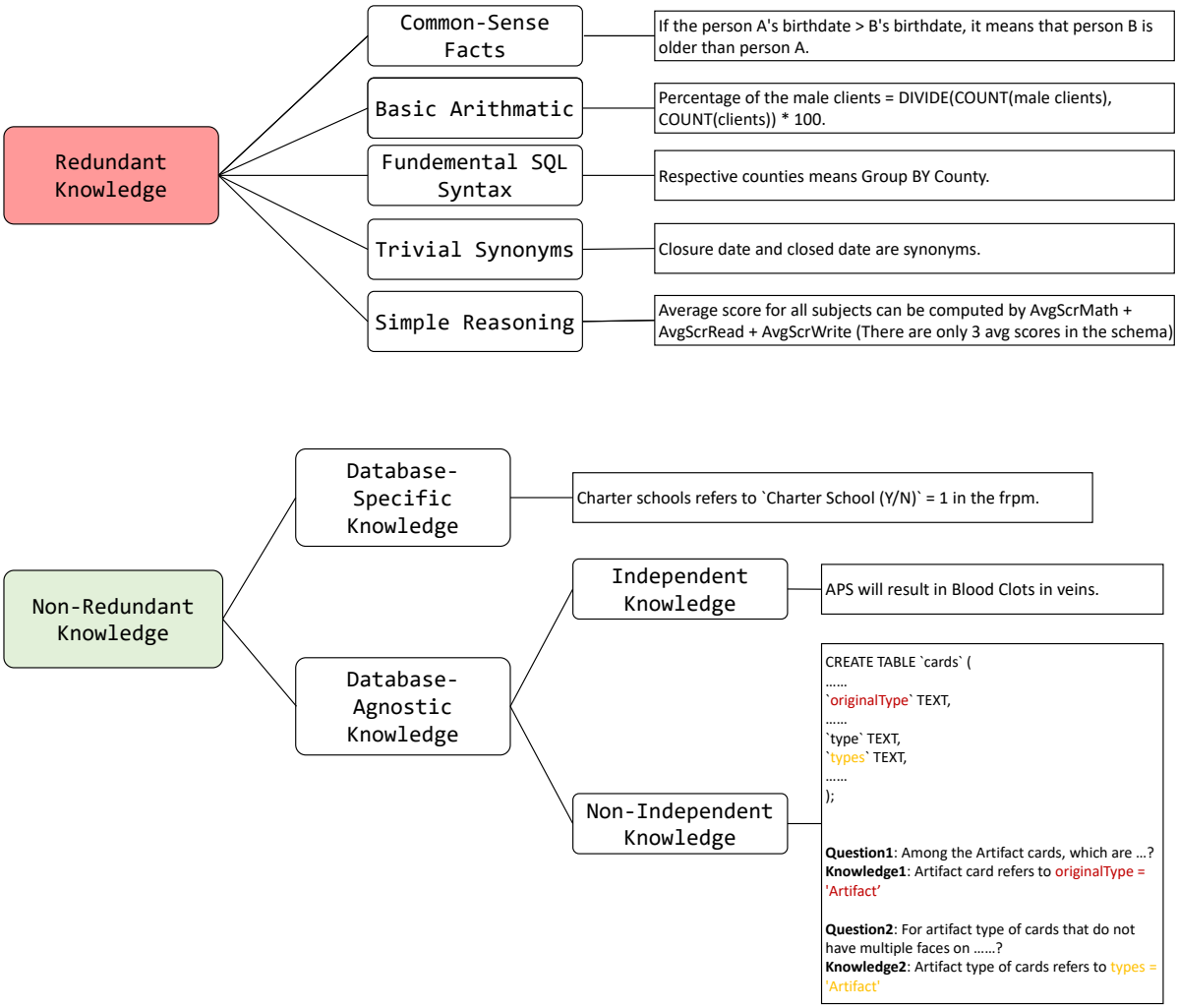

Figure 10: Classification and examples of BIRD external knowledge

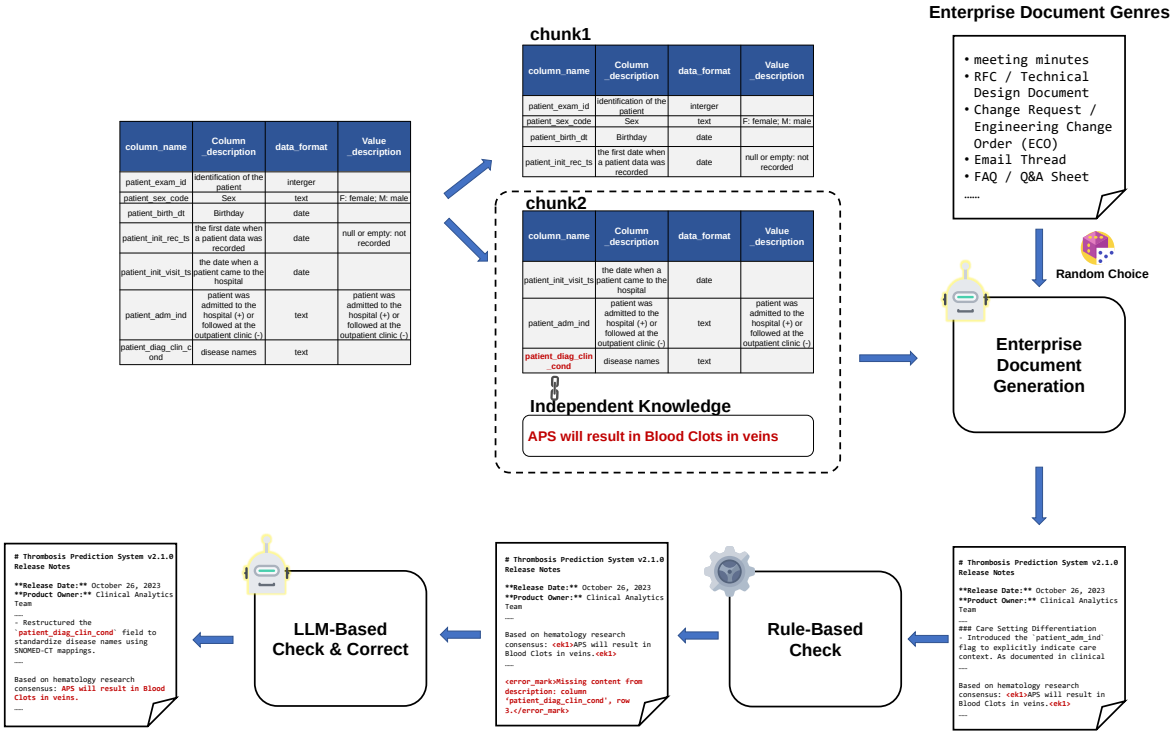

Figure 11: The pipeline for generating enterprise documents from raw external knowledge

Genre	Description
meeting minutes	Meeting minutes are written records of a meeting, typically including the time, location, attendees, topics discussed, decisions made, and action items for follow-up. It serves as an efficient communication tool to ensure that meeting participants and those who missed the meeting can stay informed and track the execution of decisions.
RFC / Technical Design Document	A Request for Comments (RFC) or Technical Design Document is used to propose and discuss technical decisions, design solutions, and system specifications. It provides a detailed explanation of the problem, proposed solutions, alternatives considered, risks, and potential impacts. This document serves as a reference to guide teams through the decision-making process, ensuring alignment and consistency in implementation.
Change Request / Engineering Change Order (ECO)	A Change Request (CR) or Engineering Change Order (ECO) is a formal document used to propose and approve changes to an existing system, product, or process. It details the nature of the proposed change, the reasons for the change, the potential impacts, and any associated risks. The document is typically reviewed and approved by relevant stakeholders before any changes are made.
Email Thread	An Email Thread is a chronologically ordered series of messages exchanged over the corporate mail system. Each message carries full header metadata (From, To, Cc, Date, Subject, Message-ID) and the complete body—often including quoted context from earlier replies. Because every revision and clarification is preserved verbatim, an email thread serves as a faithful, time-stamped record of decisions, technical justifications, and action items.
IM Thread	An Instant Messaging (IM) Thread captures a real-time chat conversation—typically from Slack, Microsoft Teams, or a similar platform—including timestamps, participants, and threaded replies. It preserves quick clarifications, decisions, and action items exchanged during day-to-day work.
FAQ / Q&A Sheet	An FAQ (Frequently Asked Questions) or Q&A Sheet provides answers to common questions, issues, or scenarios that are regularly encountered by teams or users. It organizes important clarifications, solutions, and best practices in a question-and-answer format, making it easy to refer to and address repeated inquiries.
API Reference / Interface Spec	An API Reference or Interface Specification document provides detailed information about the APIs (Application Programming Interfaces) or service endpoints that enable different software systems to communicate. This document includes the structure of API requests and responses, available methods, parameters, error handling, and any authentication or authorization requirements.
Release Notes / Changelog	Release Notes or Changelogs document the changes made in each version of a software product, including new features, enhancements, bug fixes, and any breaking changes. They help stakeholders, developers, and end-users track the progress and updates made to a system over time.
Test Plan / QA Checklist	A Test Plan or QA Checklist is a document that outlines the testing strategy for a new feature, system, or release. It defines the scope of testing, test cases, expected results, and the overall process to ensure that the software meets quality standards and is free from defects before it is released to production.

Table 16: Common Enterprise Document Genres and Their Descriptions

Question:

For the patient who has the highest Ig A within the normal range, what is his or her diagnosis?

Correct SQL:

```
SELECT patientData.patient_diag_clin_cond
FROM xelora1_thromb_patient AS patientData
INNER JOIN xelora1_thromb_lab_monitoring AS labData
  ON patientData.patient_exam_id = labData.patient_exam_id
WHERE labData.lab_iga_lvl BETWEEN 80
 AND 500 ORDER BY labData. lab_iga_lvl DESC LIMIT 1
```

Predicted SQL:

```
SELECT patient_diag_clin_cond
FROM xelora1_thromb_clin_exam
WHERE exam_anti_cardiolipin_iga_conc <= 10
ORDER BY exam_anti_cardiolipin_iga_conc DESC
LIMIT 1
```

Retrieved tables

```
CREATE TABLE `xelora1_thromb_clin_exam` (
`patient_exam_id` INTEGER,
`exam_ts` DATE,
…
CREATE TABLE `apghi56_rdr_dg_workflow` (
`diagnostic_id` INTEGER,
`patient_id` INTEGER,
…
CREATE TABLE `apghi60_nibr_clin_diag` (
`diagnosis_id` TEXT,
`subject_id` TEXT,
…
```

Missed tables

```
CREATE TABLE `xelora1_thromb_patient ` (
`patient_diag_clin_cond ` TEXT,
`patient_exam_id ` TEXT,
…
CREATE TABLE ` xelora1_thromb_lab_monitoring` (
`patient_exam_id ` TEXT,
`lab_iga_lvl ` TEXT,
…
```

Documents

Omitted…

Figure 12: An example for schema-retrieval error

Question:

What is the percentage of cards whose language is French among the Story Spotlight cards?

Correct SQL:

```
SELECT CAST(SUM(CASE WHEN T2.
card_trans_lang = 'French'
THEN 1 ELSE 0 END) AS REAL) * 100 /
COUNT(T1.card_id) FROM
zentra11_mcard_crd_catalog AS T1
INNER JOIN zentra11_mcard_foreign_data AS T2
      ON T1.card_id_uuid = T2. card_id_ uuid
WHERE T1. card_attr_story_spotlight_ind = 1
```

Predict SQL:

```
SELECT ROUND(100.0 * SUM(CASE WHEN
card_trans_lang = 'French' THEN 1 ELSE 0 END) /
COUNT(*), 2)
AS french_percentage
FROM zentra11_mcard_crd_catalog_snapshot
WHERE card_attr_story_spotlight_ind = 1;
```

Retrieved tables

```
CREATE TABLE `zentra11_mcard_foreign_data` (
`card_id` INTEGER /*Examples: 1*/,
`card_id_uuid` TEXT,
`card_trans_lang` TEXT ,
…
CREATE TABLE `zentra11_mcard_crd_catalog_snapshot` (
`card_id` INTEGER,
`card_id_uuid` TEXT,
`card_attr_story_spotlight_ind` INTEGER,
…
CREATE TABLE `zentra11_mcard_crd_catalog` (
`card_id` INTEGER,
`card_id_uuid` TEXT,
`card_attr_story_spotlight_ind` INTEGER,
…
```

Documents

Omitted...

Figure 13: An example for schema-linking error

Question:
How many drivers managed to finish the race in the 2008 Australian Grand Prix?

Correct SQL:

```
SELECT COUNT(T2. driver_id)
FROM zentra13_f1mgr_races AS T1
INNER JOIN zentra13_f1mgr_drvr_race_results AS T2
ON T2. const_standings_race_id = T1. const_standings_race_id
WHERE T1. circuit_full_name = 'Australian Grand Prix'
AND T1. race_season_yr = 2008
AND T2. race_finish_tm IS NOT NULL
```

Predicted SQL:

```
FROM zentra13_f1mgr_drvr_race_results rr
JOIN zentra13_f1mgr_races r
ON rr.constructor_race_id = r.const_standings_race_id
WHERE r.race_season_yr = 2008
AND r.circuit_full_name = 'Australian Grand Prix'
AND rr.race_result_fin_pos IS NOT NULL
```

Retrieved tables

Omitted…

Documents

Document1: Formula 1 Driver Information Service API v1 Reference
…
Field Specifications
`racing_number`: Integer representing the number permanently assigned to the driver during races. Maps to database field driver_racing_num.
…
Document2: F1 Database Schema Clarifications: Driver Laptimes & Standings Tables
From: Rajiv Desai
To: Data Engineering Team
…
Document3: Formula 1 Race Results Data: Frequently Asked Questions
…
Document4: Formula 1 Racing Data Management: Pit Stops & Results FAQ
…
Document5: Urgent: Clarifications Needed for F1 Reporting Schema

Missed
Target Document: RFC-2024-001:
The race_finish_tm field (finish time) uses different formats based on driver position: commonsense evidence: ;
1. if the value exists, it means the driver finished the race.

Figure 14: An example for knowledge-retrieval error

Question:
Which group does superhero A-Bomb belong to?

Correct SQL:

```
SELECT T2. superhero_race_classif
FROM zentos1_hrodom_char_metadata AS T1
INNER JOIN zentos1_hrodom_race AS T2
  ON T1. sup_race_id = T2. align_id
WHERE T1.superhero_name = 'A-Bomb'
```

Predict SQL:

```
SELECT ca.superhero_align_ethic_stance
FROM zentos1_hrodom_char_metadata AS cm
JOIN zentos1_hrodom_char_alignment AS ca
  ON cm.superhero_align_id = ca.align_id
WHERE cm.superhero_id_name = 'A-Bomb'
```

Retrieved tables

Omitted…

Documents

Document1: Superhero Management System v2.0.0 Release Notes
…
Overview
This major release introduces foundational changes to our superhero attribute
…
Document2: Superhero Registry API v1
Requests require HMAC-SHA256 signatures using client credentials issued by the Comic Industry Federation (CIF). Include headers:
…
Document3: Design Rationale
…
Document4: RFC-001: Design for Superhero Attribute Management System
…
Document5: RFC-001: Proposed Data Model Standardization for Superhero Attribute Management System
…
"commonsense evidence: In the context of superheroes, superhero_race_classif would refer to the particular group of people that the superhero belongs to base on these physical characteristics"
…

Figure 15: An example for knowledge-grounding error

Prompt for Domain Expansion

```
# Task Description
Create an enterprise-grade **SQLite** database schema related to the topic **{TOPIC_SLOT}**.

# Requirements
1. The schema must contain **at least {MIN_TABLES_SLOT} tables** and at **least {MIN_COLUMNS_SLOT} columns** in total.
2. The schema must be substantially different from existing databases in both intended functionality and naming, while
still falling within the scope of the given topic. Do not duplicate or closely imitate any existing database. **Table
names must be unique: generating any table with the table name that duplicates one from the existing databases is
strictly prohibited.**
3. Provide only:
    - the database name;
    - for each table: its name, column names with data types, primary-key definition(s) and foreign-key constraint(s).
    - Do not include sample data.
4. Design the schema to reflect a realistic enterprise use case.
5. The schema you generate needs to be relevant to the topic; But it (including the names of database, tables and
columns) does not necessarily have to include keywords from the topic itself. For example, databases under the
"sports_and_athletes" topic can be related to any sport, such as tennis, baseball, chess, and so on. We encourage
generating **a diverse range of** databases that align with the topic.
6. The schema you generate must comply with the **SQLite database specifications**.
7. Your output should be entirely in English.

# Existing Databases (for exclusion check)
{EXSISTING_DATABASES_SLOT}

# Enterprise Schema Example
Here are some examples of enterprise schemas to help you understand the characteristics of enterprise database schemas,
Do not reuse its content or imitate its format:
```plain
'anefi_ods_yeb_asset_increase_order_delta_hh'(asset_order_id ''|gmt_create ''|gmt_modified ''|original_order_id ''|biz_request_id ''|user_id
''|biz_type ''|quotient ''|real_amount ''|status ''|biz_no ''|out_biz_no ''|bill_detail_id ''|biz_dt ''|trans_dt ''|pmt_dt ''|biz_context ''|memo
''|ext_info ''|contract_id ''|asset_account_type ''|asset_account_no ''|inst_id ''|sub_biz_type ''|cnl_pd_code ''|cnl_ev_code ''|cnl_no
''|biz_pd_code ''|biz_ev_code ''|pd_code ''|ev_code ''|payment_id ''|gmt_commit ''|business_type ''|fund_code 'Fund Code'|fund_inst 'Fund
Institution'|clear_dt 'Clearing Date'|dt ''|hour '')

'anods.ods_cfm_fund_order_delta_hh'(prefix 'Rowkey Prefix'|id 'Fund Document Number'|parent_id 'Parent Fund Document Number'|biz_trans_id
'Business Transaction ID'|biz_trans_code 'Business Transaction Code'|request_user_id 'Merchant Account ID'|original_from 'Business
Source'|access_channel 'Access Channel'|out_biz_no 'External Business Number'|order_no ' Order Number'|payment_no 'Payment Serial
Number'|sub_biz_type 'Sub-business Type'|payer_party_id 'Payer Participant ID'|payer_card_id 'Payer Card Info ID'|payer_card_no 'Payer Card
Number'|payee_party_id 'Payee Participant ID'|payee_card_id 'Payee Card Info ID'|payee_card_no 'Payee Card Number'|currency 'Currency
Code'|amount 'Transaction Amount'|real_amount 'Actual Amount'|remark 'Remark'|status 'Status'|sub_status 'Sub-status'|error_code 'Error
Code'|fail_reason 'Failure Reason'|memo 'Memo'|gmt_execute 'Execution Time'|gmt_create 'Creation Time'|gmt_modified 'Modification
Time'|gmt_expired 'Expiration Time'|gmt_confirm 'Confirmation Time'|gmt_pay 'Payment Time'|order_fee 'Order Pre-charge Amount (frozen
charge amount)'|charge_consult_no 'Pre-charge Document Number (charge currently 28 digits)'|cnl_pd_code 'Channel Product Code'|cnl_ev_code
'Channel Event Code'|cnl_no 'Channel Serial Number'|biz_pd_code 'Upstream Business Product Code'|biz_ev_code 'Upstream Business Event
Code'|pd_code 'Business Product Code'|ev_code 'Business Event Code'|ext_info 'Extended Field'|parent_biz_trans_id 'Parent Business Transaction
ID'|dt ''|hour '')
```

Output Format
Return a single JSON object in your response and enclose it within <answer> and </answer>:

<answer>
{
    "db_name": "The name of the database you generated",
    "schema": {
        "the name of table 1": {
            "schema":{
                "the name of column 1": "data type of column 1",
                "the name of column 2": "data type of column 2",
                "the name of column 3": "data type of column 3",
                // other columns in table 1
            },
            "constraints":{
                "primary key": "PRIMARY KEY (`column x`)",
                "foreign keys": [
                    "FOREIGN KEY (`column x`) REFERENCES `table i` (`column y`)",
                    // other foreign key constraints
        ]
            }
        },
        "the name of table 2": {
            "schema":{
                // columns in table 2
            },
            "constraints":{
                "primary key": "primary key",
                "foreign keys" [
                    // foreign keys if exisit
                ]
            }
        },
        // other tables in this database
    }
}
</answer>

Output
```

Figure 16: The prompt for domain expansion.

Prompt for Project and Area Name Generation

```
You are a senior enterprise database architect. You are helping a company restructure its legacy database into an
enterprise-grade system.


Database Name: {database_name}

You are given:
- Database name
- Database schema (DDL)
- database_description

Please:
1. Generate a project name (4-6 characters) for this database.The project name is randomly generated, has nothing to
do with the database, and does not repeat with the domain.
2. Generate a high degree domain abbreviation (3-8 characters) that captures the business domain


Output format:
Project Name: [your_project_name]
Domain Abbreviation: [your_domain_abbr]



[Database Schema]
{schema}
{desc_section}
```

Figure 17: The prompt for table project and area name generation.

Prompt for Enterprise Table Name Generation

```
You are an enterprise database architect helping rename a table to follow enterprise naming conventions.

Please rename the table {table_name} using the following pattern:


Naming Pattern:
<project>_<domain>_<content>

Where:
- <project>: {project} (already determined)
- <domain>: {domain} (already determined)
- <content>: Semantic summary of the table's core entity or process


Example:
Original table: user_behavior
Expected: {project}_{domain}_usr_behavior

Now rename:

Table name: {table_name}



Guidelines:
1. Use precise, business-aware terms for <content>
2. Be concise but descriptive
3. Use common abbreviations (usr for user, txn for transaction, etc.)
4. Select the most appropriate partition suffix from the provided options

Only output the new table name as a single line. Do not include any explanations or formatting.
```

Figure 18: The prompt for table name generation.

Prompt for Area Explanation

```
You are a senior enterprise database architect. You are helping a company restructure its legacy database into an
enterprise-grade system.

You are provided with the following:
- A relational database schema (DDL format);
- A column-level description file (if available), which includes: original column names, semantic names, textual
descriptions, and possible value explanations.

Your task is to perform a domain-level, enterprise-grade analysis of the database and return a structured summary for
downstream usage.

## Your output should include the following sections:

### 1. Business Domain Classification
- Clearly identify the high-level business domain the database belongs to.
- Provide one or more enterprise-level subdomains involved
- Use concise and enterprise-recognizable terminology.

### 2. Table Themes and Functional Definitions
- For each table in the schema, explain its business purpose.
- Describe the type of entity or process the table models

### 3. Field Classifications and Semantic Grouping
- Organize fields into functional groups, such as:
- Identification fields - Address/location fields
-  Operational attributes
-  Classification codes
- For each group, summarize its business purpose and usage context.

### 4. Semantic Repair and Enrichment
- Identify any columns whose descriptions are missing, vague, or purely abbreviations.
- For such fields, infer and supplement accurate definitions based on domain knowledge.
- Emphasize clarity and completeness over brevity. Avoid unexplained abbreviations unless they are domain standards
and include their expansion.

### Formatting Constraints:
- Be concise but informative.
- Avoid repeating the schema line-by-line; instead, provide a business-level summary.
- Use bullet points or short paragraphs for clarity.
- Do not assume a single fixed domain.

Expected Output Format:

Business Domain Classification:
This database primarily serves the following domain(s):
- [High-level domain, e.g., Public Education, Financial Services, Healthcare...]

Enterprise Subdomains:
- [Subdomain A]: [Short description, e.g., "Manages the lifecycle and attributes of registered educational
institutions."]
- [Subdomain B]: [Short description, e.g., "Tracks charter classification and operational status of schools."]
...

Table Classification and Definitions:
- table_name_1: [Concise description of what entity or process the table represents and its business purpose.]
- table_name_2: [Concise description...]
...

Field Classification and Groupings:
- Identification Fields:
  - field_a: [Business meaning]
  - field_b: ...
- Geolocation / Address Fields:
  - ...
- Status / Operational Attributes:
  - ...
- Classification / Typing Codes:
  - ...
...

Semantic Repair and Enrichment:
- [field_name_1]: [Inferred or repaired explanation. Expand abbreviation if unclear, and describe usage.]
- [field_name_2]: ...
...

**Database Schema:**
{schema_ddl}
{desc_section}
```

Figure 19: The prompt for column name area explanation.

Prompt for Column Names Expansion

```
You are an enterprise database architect specializing in field expansion and semantic enrichment. Your task is to
transform academic or legacy column names into enterprise-grade naming patterns.

Enterprise Naming Recommendations:
1. Avoid using casual prefixes like is, has, or was. Use neutral, abstract, and semantically-rich terms instead.
2. Ensure column names reflect business meaning, embedding semantics and context without redundancy.
3. Supplement the column name with contextual information about the table and related fields. The contextual
information needs to conform to the column name logic and structured naming.



Domain Adaptation Rule:
- The database you are working with may belong to a specific business domain.
- If a certain column concept or terminology is typically expressed using domain-specific naming conventions, please
adapt to that convention naturally.
### Domain Context:
{domain_analysis}
Instructions:
1: Suggest a full, enterprise-style column name that clearly expresses its business meaning. You may rewrite or
reorder components, but do not omit or change key context words already in the original name.
2: If a more commonly used business domain synonym exists for a component, perform the replacement. Only replace terms
that do not affect the core semantic scope.
3: Ensure that the column name covers both the column description and value meanings in a meaningful but non-redundant
way.

### Column Information:
- Table: {column_info.table_name}
- Original Name: {column_info.original_name}
- Description: {description_display}
- Data Type: {column_info.data_format}
- Value Semantics: {value_desc_display}
{sample_text}


Final Output Format:
### Final Column Name: ...


Your response must include the exact header '### Final Column Name:'
```

Figure 20: The prompt for column name expansion.

Prompt for Column Name Abbreviation

```
You are an enterprise database engineer. Your task is to abbreviate and normalized the following enterprise-style
column name.
Unify similar rules:
for any synonymous or overlapping terms in the input,
choose one canonical wording and apply it consistently across all outputs.
Prefer clearly abbreviated words over ad-hoc name.
For each category of indicators, choose one consistent naming convention within the same table. For example, for
level-type indicators, always use either level or lvl consistently. For countable indicators, unify choices among cnt,
count, or total and apply the same form to all related columns. Apply this principle to other types of indicators as
well.

Abbreviate Rules:
1. Use snake_case format only.
The abbreviation rule for snake case is to determine whether each component between underscores can be abbreviated.
Example:
description_product_number: desc_prod_num
2. Retain the integrity of the meaning of the core terms in the Original Name and ensure that experts in the relevant
field can understand its meaning after abbreviation.
Do NOT use single-letter initials.
3.If a token appears multiple times within the same output batch, always abbreviate it consistently.
4. Abbreviations should be as close to the original meaning as possible—easy to understand and semantically clear. If
you think a word is already concise, you don't need to abbreviate it.

### Column Information:
- Full Enterprise Name in the table: {full_column_names}

### Output Format:

Final Column Names in the table: [Final names]

Your response must include the exact headers shown above.
```

Figure 21: The prompt for column name abbreviation.

Prompt for Document Generation

```
# Background
To build a benchmark that truly reflects **enterprise-grade Text-to-SQL** scenarios, we must replicate how knowledge is distributed inside
real companies:
- Business / domain knowledge is scattered across meeting minutes, change requests, ops runbooks, emails, and other heterogeneous documents.
- A querying system must **retrieve** information from a vast collection of external documents **before** generating SQL.

Your task is to **sparsify** the external knowledge you receive—turning it into a single Markdown document that resembles a real-world
knowledge artifact.

# Task Description
You are given two sources of external knowledge:
1. **Database description** – full table/column names, value explanations, and embedded business / domain knowledge.
2. **Extra knowledge** – additional domain knowledge not present in the description file.

**Your goal is to transform this input into one Markdown document of at least {MIN_WORDS_SLOT} words, written in the specified genre as
detailed below**:
{DOC_GENRE_SLOT}

# Requirements
1. **No knowledge may be omitted.**
   - Every piece of original external knowledge **must appear somewhere in the document, except for `sample_value` and `data_format`**.
     *This explicitly includes:*
     * All rows from the database description table (Including 3 columns: `original_column_name`, `column_description`,
`value_description`),
     * every item contained in the extra-knowledge input.
   - If any information cannot be woven naturally into the narrative, list it verbatim in a "🕮 Loose Notes" block at the end of the
document—*otherwise do not include that block*.

2. **Tag original external knowledge for manual completeness checks. Use the tagging scheme below:**
   - For each row *i* (starting at 1) in the database-description table, tag each non-empty field as:
     - `original_column_name` → <ocn{i}> … </ocn{i}>
     - `column_description`   → <cd{i}> … </cd{i}>
     - `value_description`    → <vd{i}> … </vd{i}>
     - The `sample_value` and `data_format` columns are provided for your reference to help you understand the external knowledge. You
**may include relevant content** in the final document if appropriate, but **must not apply any tags** to it.
   - If a field in that row is blank, **omit the corresponding tag; never invent content**.
   - For each extra-knowledge item *k* (in provided order), tag it as <ek{k}> … </ek{k}>.
   - The text inside every tag must come directly from the original external knowledge (verbatim or with only trivial rephrasing that
leaves the meaning unchanged).
   - Distribute tagged elements throughout the document and do not cluster them in a single section if possible. Avoid mechanically listing
tags and the knowledge within tags (e.g., directly providing a list).

3. **Sparsification & contextualization**
   - **Do not** present a tidy data-dictionary list such as "table-field-meaning".
   - Knowledge should be woven naturally into the chosen genre and dispersed across the document.
   - You may appropriately expand upon the original external knowledge to make the document more natural and realistic. However, the final
document **must not** contain any content that contradicts the original external knowledge.
   - If new columns that are not present in the description file are mentioned (e.g., due to simulated discussions or change proposals),
they must be framed negatively—for example, discussing their removal or disagreement over adding them.

4. **Directory path**
   - Provide a relative path for the document (e.g., `design/RFC-0421.md` or `ops/playbook/alert_CPU.md`) to show hierarchy.

5. **Independent theme**
   - The document's subject matter **must not** directly reference the task itself (e.g., Text-to-SQL benchmark, enterprise Text-to-SQL,
External Knowledge). Choose varied, realistic business or technical topics instead.

# Output Format
Return a single JSON object in your response and enclose it within <answer> and </answer>:

<answer>
{
  "path": "your/path/here",
  "genre": "{DOC_GENRE_NAME_SLOT}", // Only the genre name needs to be given
  "title": "the title of the document",
  "content_md": "the content of the document"
}
</answer>

# Original External Knowledge
## Database description
{DB_DESCRIPTION_SLOT}

## Extra knowledge
{EXTRA_KNOWLEDGE_SLOT}

# Output
```

Figure 22: The prompt for enterprise knowledge document generation.

Prompt for Doucument Check & Correct

```
# Background
In the prior task, your colleague has taken concise, densely packed external knowledge and—following the
sparsification & tagging rules—expanded it into **one Markdown document** written in the specified genre. This
artifact will later feed a retrieval-based Text-to-SQL benchmark, so its accuracy is critical.

## Prior task description and materials
### Prior task description
You are given two sources of external knowledge:
1. **Database description** – full table/column names, value explanations, and embedded business / domain
knowledge.
2. **Extra knowledge** – additional domain knowledge not present in the description file.

Your task is to **sparsify** the external knowledge you receive—turning it into a single Markdown document
that resembles a real-world knowledge artifact.

### Database description
{DB_DESCRIPTION_SLOT}

### Extra knowledge
{EXTRA_KNOWLEDGE_SLOT}

### Target genre
{DOC_GENRE_SLOT}

### **Sparsification & tagging rules**
   - **No knowledge may be omitted.**
      - Every piece of original external knowledge **must appear somewhere in the document, except for
`sample_value` and `data_format`**.
……

(Omitted here, you can refer to the same part of the pormpt for document generation)

……
## Candidate document
To assist you in identifying errors, the candidate document has already been pre-checked according to specific
rules, and any detected issues have been marked accordingly. (Note: The rule-based check is limited in
capability and may miss some errors — you must carefully identify any remaining issues during your review.)
```md
{CANDIDATE_DOC_SLOT}
```

Your Task
Audit the candidate document against the source materials and the rules, then correct it where necessary.

Checklist
1. **Completeness**
   - Every element of the original external knowledge except for `sample_value` and `data_format` must appear
in the document—either in the main text or, if it cannot be woven in naturally, in a “🛈 Loose Notes” block.

2. **Tag validation & correction**
   - **Structural correctness** Each tag must have a valid index, a recognized tag type (ocn, cd, df, vd, ek),
and must appear in proper pairs (<tag> and </tag>). Fix any malformed or unmatched tags.
   - **Existence** If the corresponding field in the database description is blank, delete the tag **and its
enclosed content**.
   - **Accuracy** The text inside every tag must match the source verbatim, or with only trivial re-phrasing
that preserves meaning. Replace mismatches with the correct source text.

3. Check whether any other part of the document violates the sparsification and tagging rules. If such
violations are found, revise the document accordingly to ensure full compliance.

Requirements
1. Modify **only** the content that violates the checklist or rules; everything else must remain unchanged.
2. You only need to revise the document content — no need to provide the title, genre, or file path.
3. When checking for completeness, go through the database description row by row, verifying each column field
individually. Then do the same for each item in the extra knowledge section. Present the results as a
structured checklist.
4. When validating tags, iterate over each tag that appears in the document, and check the following
dimensions: Structural correctness, Existence, Accuracy. Present the tag validation as a checklist as well.

Output format
Provide **one final, corrected Markdown document** that meets all requirements, and wrap it exactly as shown:
<answer> ...your corrected Markdown... </answer>

Output
```

Figure 23: The prompt for checking and correcting enterprise knowledge document.

Prompt for Evaluation on BIRD-Ent

```
# Task Description
You own a data asset (similar to a data warehouse) that has collected a series of tables from different sources around
a specific topic. You are now given the schemas and data sources for all the tables within this data asset. To help
you better answer the question, your colleague has pre-selected several tables that may be relevant to the question.
Based on this information, please write an SQLite query to answer the following question.

# Requirements:
1. You only need to provide one complete, correct, and executable SQL statement.
2. Do not include any additional information outside of the SQL query in your response.

# Output Format
Your SQL should be placed between <answer> and </answer>, as follows:
<answer>your SQL query</answer>

# Data Table Naming Standards
Below are the naming conventions for the data tables, which will help you understand the following Data Asset
Information:
Table naming conventions: <project>_<area>_<content>.
- <project>: The project/product or data domain identifier to distinguish teams, business lines, or application
contexts.
- <area>: Business domain and subject area identifier which the table belongs.
- <content>: The summary of the table's content.

# Data Asset Information
{SCHEMA_SLOT}

# External Knowledge
The following document may contain useful information needed to answer the question and is provided for your reference:
{KNOWLEDGE_SLOT}

# Output
Question: {QUESTION_SLOT}
Your Answer:
```

Figure 24: The prompt for evaluation on the BIRD-Ent Benchmark.

Prompt for Evaluation on Spider-Ent

```
# Task Description
You own a data asset (similar to a data warehouse) that has collected a series of
tables from different sources around a specific topic. You are now given the schemas
and data sources for all the tables within this data asset. To help you better answer
the question, your colleague has pre-selected several tables that may be relevant to
the question. Based on this information, please write an SQLite query to answer the
following question.

# Requirements:
1. You only need to provide one complete, correct, and executable SQL statement.
2. Do not include any additional information outside of the SQL query in your
response.

# Output Format
Your SQL should be placed between <answer> and </answer>, as follows:
<answer>your SQL query</answer>

# Data Asset Information
{SCHEMA_SLOT}

# Output
Question: {QUESTION_SLOT}
Your Answer:
```

Figure 25: The prompt for evaluation on the Spider-Ent Benchmark.