RASL: Retrieval Augmented Schema Linking for Massive Database Text-to-SQL

Jeffrey Eben
jeffeben@amazon.com
Amazon
Jersey City, NJ, USA

Aitzaz Ahmad
aitzaza@amazon.com
Amazon
Seattle, WA, USA

Stephen Lau
lausteph@amazon.com
Amazon
Seattle, WA, USA

ABSTRACT

Despite advances in large language model (LLM)-based natural language interfaces for databases, scaling to enterprise-level data catalogs remains an under-explored challenge. Prior works addressing this challenge rely on domain-specific fine-tuning—complicating deployment—and fail to leverage important semantic context contained within database metadata. To address these limitations, we introduce a component-based retrieval architecture that decomposes database schemas and metadata into discrete semantic units, each separately indexed for targeted retrieval. Our approach prioritizes effective table identification while leveraging column-level information, ensuring the total number of retrieved tables remains within a manageable context budget. Experiments demonstrate that our method maintains high recall and accuracy, with our system outperforming baselines over massive databases with varying structure and available metadata. Our solution enables practical text-to-SQL systems deployable across diverse enterprise settings without specialized fine-tuning, addressing a critical scalability gap in natural language database interfaces.

KEYWORDS

Retrieval Augmented Generation, Text-to-SQL, Schema Linking, Table Retrieval

1 INTRODUCTION

Text-to-SQL systems translate natural language questions into executable SQL queries, enabling non-technical users to extract insights from databases without SQL expertise. While these systems have evolved from rule-based approaches to powerful large language model (LLM) solutions [8], scaling them to industrial settings with massive database catalogs remains an underexplored challenge.

Current state-of-the-art methods primarily leverage LLMs using techniques like task decomposition and prompt optimization, avoiding the overhead of maintaining fine-tuned models [11, 13, 17, 22]. However, these approaches face critical limitations when applied to enterprise environments with thousands of tables and tens of thousands of columns. In such massive database settings, providing comprehensive schema context to LLMs becomes untenable due to token limitations, computational costs, and semantic overload [12, 21, 23]. For example, a typical enterprise data catalog with 10,000 tables averaging 50 columns each would require over 500,000 schema entities—far exceeding current LLM context windows and creating prohibitive costs for commercial API usage.

Existing solutions for scaling to massive catalogs either rely on hierarchical selection methods that require domain-specific training and well-defined database-table hierarchies [23], employ computationally intensive multi-agent frameworks that process entire schemas through many LLM calls [21], or use complex optimization techniques that don't scale to truly massive schemas [6]. These methods struggle in real-world deployments where database architectures often follow monolithic NoSQL paradigms, metadata about join relationships is incomplete, or database schema is often changing. The scalability challenge becomes exponentially worse as database size increases: while methods may work reasonably well on benchmarks with hundreds of tables, they fail to maintain acceptable performance and cost efficiency when scaled to enterprise catalogs with thousands of tables [21].

We present Retrieval Augmented Schema Linking (RASL), a novel approach designed specifically for text-to-SQL over massive database schemas without requiring fine-tuning or well defined database relations. RASL decomposes schemas into semantic entities, indexes them in a vector database, and employs a multi-stage retrieval process with relevance calibration to efficiently narrow the search space while maintaining compatibility with hosted LLM services.

Our contributions include:

- A zero-shot schema linking architecture that scales to massive databases with minimal preprocessing, no model training, and no requirement of known database hierarcy and join relations
- An entity-level decomposition strategy with keyword-based context retrieval and entity-type relevance calibration
- Empirical evidence of RASL's effectiveness on industrial-scale benchmarks

Our work bridges the gap between academic text-to-SQL research and industrial requirements, providing a practical solution for natural language interfaces to massive data environments.

2 RELATED WORKS

Schema linking—mapping natural language elements to database components—becomes exponentially more challenging as database size increases. While some works have found schema-linking to be unnecessary on standard benchmarks with the latest foundation model offerings [15], others have shown that text-to-SQL performance degrades as database size increases and state-of-the-art methods are unable to scale to full industry-scale data catalogs [21].

Several approaches have been proposed for massive database environments, each with limitations in industrial settings. DBCopilot [23] models schema linking as path generation through a hierarchical graph, first predicting the database, then tables, before generating SQL. While effective with clear database boundaries and known join relations, this approach struggles in monolithic data lake settings and requires extensive training on synthetic question-schema

Paper accepted to the KDD Workshop on Structured Knowledge for Large Language Models (SKnowLLM '25), Toronto, Canada.

pairs, making adaptation to schema changes difficult. CHESS [21] incorporates an Information Retriever and Schema Selector to retrieve and prune context, but faces severe scalability issues with industry-sized datasets. It processes the full schema via many LLM calls, only using retrieval to augment already-identified schema elements with additional indexed context.

CRUSH4SQL [12] embeds column names during build time and hallucinates a candidate schema from input questions to retrieve relevant columns. While conceptually similar to our approach, CRUSH is limited to column name schema context, showing degraded performance on complex benchmarks containing additional context such as descriptions and value formats. Additionally, its reliance on LLMs to hallucinate schemas for querying can lead to misalignment with ground truth database schemas where columns follow non-standard naming conventions.

RASL addresses these limitations through a zero-shot architecture that leverages both column-level and table-level context without requiring database hierarchy knowledge or comprehensive table relationship metadata. By separating build-time schema decomposition from inference-time retrieval, RASL provides an effective balance between accuracy and efficiency for industrial deployments where schemas frequently evolve across diverse storage paradigms.

3 PRELIMINARIES

We begin by establishing notation for the components of our approach.

Database Schema. We define a database schema S as a collection of tables $T = \{t_1, t_2, \ldots, t_n\}$. Each table t_i consists of a set of columns $C_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,m_i}\}$, where m_i is the number of columns in table t_i. The complete set of columns across all tables is denoted as $C = \cup_{i=1}^{n} C_i$.

Entity Types. We define $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_l\}$ as the set of all entity types, where each λ_j represents a specific type of schema information (e.g., table description). Entity types are partitioned into table-level types $\Lambda_T \subset \Lambda$ and column-level types $\Lambda_C \subset \Lambda$.

Schema Entities. For each table t_i and entity type $\lambda_j \in \Lambda_T$, we define entity $e(t_i, \lambda_j)$ as the representation of table t_i according to type λ_j (e.g., its name or description). Similarly, for each column $c_{i,k}$ and entity type $\lambda_j \in \Lambda_C$, we define entity $e(c_{i,k}, \lambda_j)$ as the representation of column $c_{i,k}$ according to type λ_j. The complete set of all entities is denoted as E.

Vector Representations. Each entity $e \in E$ is embedded in a d-dimensional space using an embedding function $\phi : E \rightarrow \mathbb{R}^d$. Similarly, a natural language question q is embedded as $\phi(q) \in \mathbb{R}^d$. The similarity between two embeddings is measured using cosine similarity.

4 METHODOLOGY

4.1 Overview

RASL addresses the challenge of scaling text-to-SQL to massive database catalogs through a two-phase approach: build-time knowledge base construction and inference-time retrieval augmented schema linking. Figure 1 illustrates our pipeline.

At build time, RASL decomposes database schema S into semantic entities E_{Λ_T} and E_{Λ_C}, which are embedded and indexed in a vector database with metadata tags incorporating full schema context for later reconstruction. At inference time, given a natural language question q, RASL extracts keywords K and performs parallel retrieval for each $k \in K \cup \{q\}$ across each entity type $\lambda_j \in \Lambda$. For each retrieval query, RASL applies entity-type-level calibration to account for variably discriminative entity types when training samples are available. RASL then filters entities to retain only those belonging to the top N tables, considering both table-level and column-level entities for table ranking. This filtered subset serves as input for LLM-based table prediction to identify the most relevant tables for the query. Finally, RASL loads the complete schema context for these predicted tables to support downstream SQL generation.

4.2 Knowledge Base Construction

Schema Entity Decomposition. We decompose database schema S into semantic entities at table and column levels as defined in our preliminaries. Specific E vary by dataset, with examples of E_{Λ_T} including table names, aliases, and descriptions and examples of E_{Λ_C} containing column names, aliases, descriptions, and value format descriptions. For example, consider a table `student_club.member` with columns `first_name`, `last_name`, and `zip`. RASL would create separate entities: $e($`student_club.member`$, \lambda_{\text{table name}})$ = "student_club.member", $e($`first_name`$, \lambda_{\text{column name}})$ = "first_name", $e($`last_name`$, \lambda_{\text{column name}})$ = "last_name", and $e($`zip`$, \lambda_{\text{column name}})$ = "zip". Each entity $e \in E$ is indexed with metadata tags linking it to the original schema structure, preserving hierarchical relationships for inference-time schema re-construction.

Vector Embedding and Indexing. Each semantic entity $e \in E$ is embedded using the embedding function ϕ to capture its semantic meaning in a d-dimensional vector space. These embeddings are indexed in a vector database optimized for similarity search, enabling efficient retrieval without task-specific fine-tuning.

Table description Synthesis. For tables with limited or missing descriptions, which is common across all datasets evaluated, we explore synthesizing descriptive text using an LLM that analyzes table structure, column names, and available metadata. This description is saved as $\lambda_{\text{table descr.}} \in \Lambda_T$, with specific details on synthesis prompts provided in C.4. For fair comparison with baselines, we primarily evaluate our system without the inclusion of $E_{\lambda_{\text{table descr.}}}$, with ablation studies exploring the effect of adding additional synthesized context on RASL's performance.

4.3 Retrieval-Augmented Schema Linking

4.3.1 Question Decomposition. Inspired by CHESS-SQL [21], we decompose each user question q into keywords $K = \{k_1, k_2, ..., k_m\}$ using a light-weight LLM to enhance retrieval effectiveness for complex questions referencing multiple schema elements, with prompt details provided in C.3. These extracted keywords serve as independent retrieval queries that help capture relevant schema elements even when the full question's semantic representation doesn't closely match corresponding schema elements.

For each $e \in E$ we concurrently perform retrieval across each $k \in K \cup \{q\}$, with an ablation study in 5.7 analyzing the impact

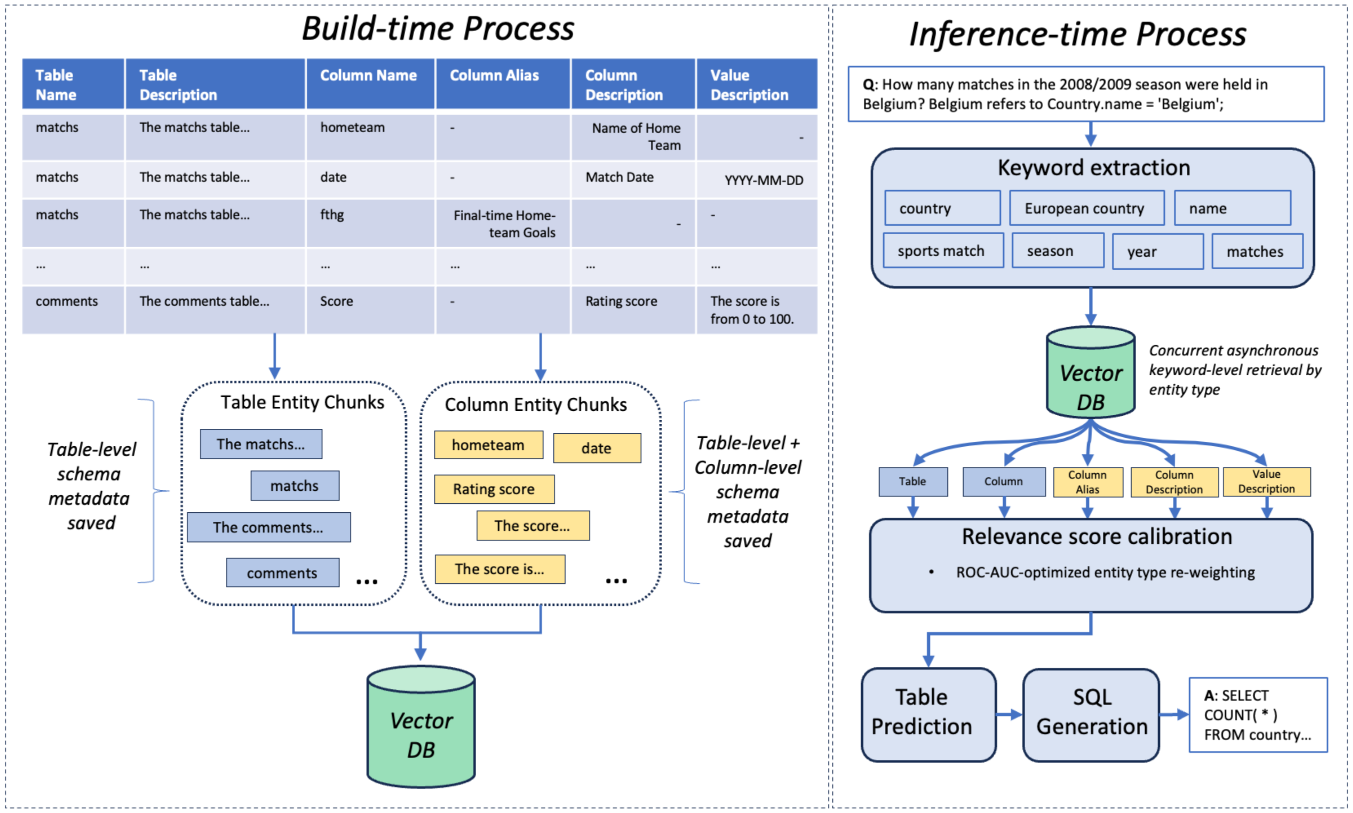

Figure 1: System overview. (left) Build-time process of constructing the schema metadata knowledge base. (right) Inference-time retrieval process for text-to-SQL applications.

of keyword-level and question-level retrieval. We also evaluate appending k directly to q as done in CHESS, but we find that this under-performs direct keyword-level retrieval.

4.3.2 Entity-Type Relevance Calibration. We hypothesize that each $\lambda \in \Lambda$ will have varying levels of importance within each dataset, which may not be captured by direct relevance scores. To account for this, we propose an entity-type relevance score calibration, where weights are calibrated over ground truth training samples when available. For each entity type $\lambda \in \Lambda$, we compute:

$$w_\lambda = \frac{|\Lambda| \cdot \text{AUC}(\lambda)^2}{\sum_{\lambda' \in \Lambda} \text{AUC}(\lambda')^2} \tag{1}$$

where AUC(λ) is the area under the table-level recall curve for entity type λ over training data. We square the AUC values to amplify differences between entity types, giving greater weight to those with stronger predictive power. These weights are then applied to scale relevance scores at inference time, ensuring that entity types with consistently stronger predictive power receive higher influence in the final ranking.

Inspired by CRUSH [12], we also explored keyword-level entropy-guided relevance calibration prior to entity-level calibration, which is designed to address the variable discriminative power of different keywords across schema entities. However, we found that this component did not improve system performance and have excluded it from our results; details on the component methodology and observed impact are discussed in B.

4.4 SQL Generation

Table Prediction. While the resulting schema entities can be used directly to construct a schema for SQL generation, we find it is beneficial to perform an intermediary table prediction prior to final query generation. This step constructs a candidate schema and applies an LLM to predict rank-ordered tables relevant to q, with specific prompts used detailed in C.1. Full schemas of tables identified as revelant are then loaded for final SQL generation. We find that this step is especially beneficial for covering unknown join relations which cannot be inferred from q, as well as better leveraging semantic context in E_{Λ_T}.

SQL Generation. Following retrieval, any SQL generation pipeline can be applied to the final schema, with our specific evaluations using zero-shot text-to-SQL with self-correction. Details on the specific prompt used in experiments is provided in C.2.

5 EXPERIMENTS

5.1 Dataset Details

We evaluate our method over three benchmarks with dataset statistics provided in Table 1. The Spider and BIRD benchmarks are designed for the single-database setting, which we adapt to the massive catalog setting by considering the full set of training and test schema for each test record.

- **Spider** [24]: A widely-used cross domain text-to-SQL benchmark with $\Lambda_C = \{column\ name, column\ alias\}$, $\Lambda_T = \{table\ name, table\ alias\}$. Database schemas are well-named, with schema elements that are well-aligned with questions.
- **BIRD** [14]: A challenging text-to-SQL benchmark emphasizing database content understanding. BIRD contains richer schema context designed to test text-to-SQL systems' abilities to incorporate domain knowledge, with $\Lambda_C = \{column\ name, column\ alias, column\ descr., value\ descr.\}$, $\Lambda_T = \{table\ name\}$. Questions may involve references to associated schema context, as opposed to direct column and table names as in Spider.
- **Fiben** [19]: An enterprise-focused benchmark developed by IBM across financial schemas. Fiben uses minimal metadata with $\Lambda_C = \{column\ name\}$, $\Lambda_T = \{table\ name\}$. Questions often lack context, making schema identification challenging.

For all datasets, we additionally synthesize $E_{\lambda_{\text{table descr.}}}$ from table schemas following the process in C.4 and analyze the impact of adding it to Λ_T on token consumption and performance in ablations. Primary evaluations are reported with $\lambda_{\text{table descr.}}$ excluded for fair comparison.

Dataset	Size		Schema			
	Train	Test	N DBs	N Tables	N Cols	Avg Cols per Table
Spider	7000	1034[a]	166	876	4503	5.1
BIRD	9427	1534[a]	80	597	4337	7.26
Fiben	0	279	1	152	374	2.46

Table 1: Statistics of datasets. [a] We use the original development sets for testing.

5.2 Baseline Methods

We compare RASL to various retrieval baselines, including both general methods adapted to text-to-SQL and methods proposed specifically for text-to-SQL schema retrieval.

- **BM25** [5]: BM25 is a lexical retrieval method widely used for document retrieval and ranking. We adapt this to table retrieval by treating each table's complete schema as one document.
- **Sentence transformer (SXFMR)** [18]: The sentence transformer is a semantic retrieval model designed to embed text to the same latent representation. Similar to BM25, table schemas are treated as documents for embedding.
- **CRUSH** [12]: CRUSH uses an LLM to hallucinate a schema from q, followed by retrieving target schema columns by lexical or semantic similarity to hallucinated schema columns. For table-level retrieval evaluations, we take the distinct tables corresponding to top ranking columns.
- **DTR** [10]: DTR uses contrastive learning on (q, t) pairs to train a table retriever. This method requires synthesizing extensive training data over the target database, and may not be best suited for enterprise settings with continuously evolving databases.
- **DBCopilot** [23]: DBCopilot synthesizes text-to-SQL pairs over a hierarchical graph of known database and table relations and uses these to train a constrained decoder for table prediction. Similar to DTR, this requires synthesis of training data over the target database schema.

5.3 Evaluation Details

Consistent with previous works [10, 12, 23], we primarily evaluate our method using macro-average Recall@N with respect to ground truth tables used in each SQL query, which measures the fraction of relevant instances in the top-N predicted tables. For primary evaluations on Recall@N, we directly adopt the metrics reported in [23], where specific method configurations are provided in F and RASL is applied over identical testing sets. For ablation studies, we focus on comparison to retrieval-based baselines which do not involve model fine-tuning, as these methods are most relevant to industry settings with evolving catalog schemas. For all ablations, we closely follow the implementation details reported in [23], with the exception of using Anthropic Claude 3.5 Sonnet-v2 [2] instead of OpenAI GPT-3.5-turbo [16] for CRUSH schema hallucination due to model access constraints.

Model	Spider		BIRD		Fiben	
	R@5	R@15	R@5	R@15	R@5	R@15
BM25	86.5	93.9	68.3	82.8	33.3	38.6
SXFMR	80.4	92.3	67.6	83.1	28.2	46.5
CRUSH$_{\text{BM25}}$	87.2	95.0	68.4	87.8	34.9	54.0
CRUSH$_{\text{SXFMR}}$	82.2	93.9	70.6	85.1	34.1	50.8
DTR	76.3	93.2	76.2	92.0	37.7	48.9
DBCopilot	91.6	97.6	85.8	94.6	41.1	56.9
RASL$_{\text{retriever}}$	72.1	94.1	70.5	92.6	34.6	64.5
RASL$_{\text{full}}$	**97.0**	**98.0**	**97.5**	**97.8**	**69.1**	**69.2**

Table 2: Primary performance comparison measuring Recall@N, where N is the maximum number of tables. RASL is applied without $\lambda_{\text{table descr.}}$ in this setting.

5.4 Experiment Settings

For all experiments, RASL uses Anthropic Claude 3.5 Haiku [1] for keyword extraction, Cohere Embed-v3-English [7] in OpenSearch Serverless through Amazon Bedrock Knowledge Bases [3] for vector database embedding, and Anthropic Claude 3.5 Sonnet-v2 [2] for table prediction and SQL generation; temperature is set to 0.0 throughout for reproducibility.

For all datasets except Fiben, which does not contain a training set, we perform entity-type weight calibration using 200 randomly sampled training instances. During retrieval, the top 100 entities within each keyword and entity type are retrieved prior to relevance calibration and filtering, as this is the maximum allowed by Bedrock Knowledge Bases.

5.5 Table Retrieval Results

In Table 2 we evaluate RASL against baseline systems for table retrieval. We evaluate RASL in two settings: (1) *retriever-only*, which ranks tables by calibrated relevance scores across all entities E, and

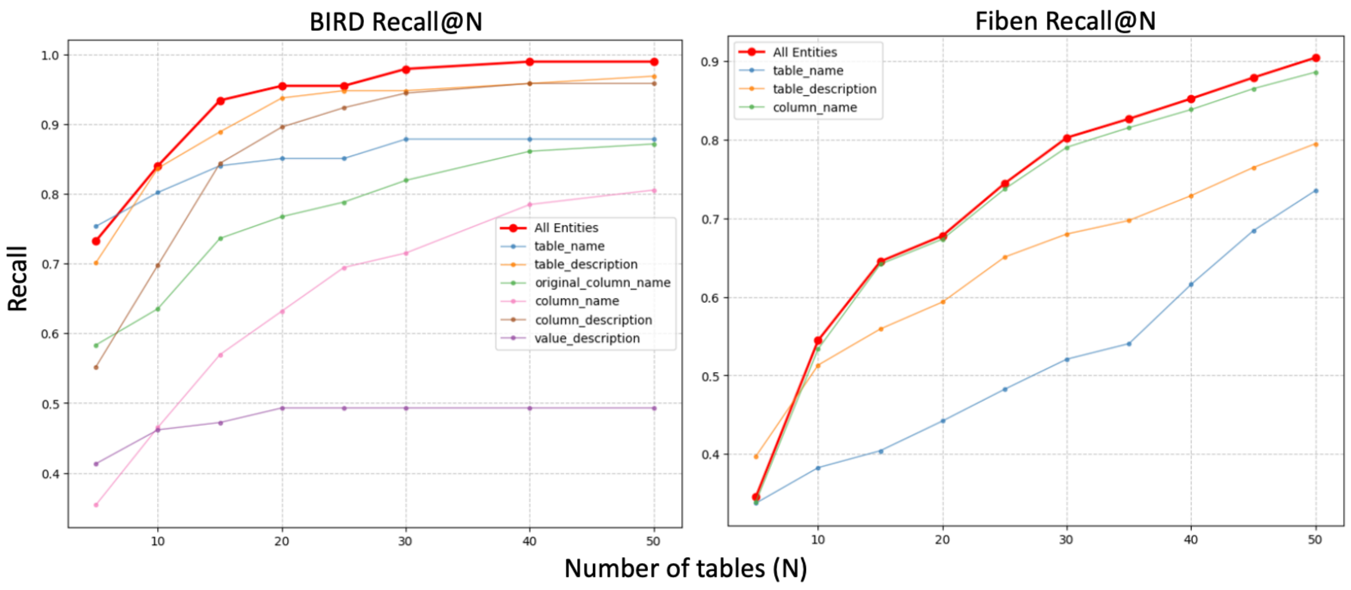

Figure 2: Table *Recall@N* over BIRD (left) and Fiben (right). BIRD benefits from both Λ_C and Λ_T, achieving notable recall improvement over individual λ at higher N, while Fiben primarily leverages E_C.

(2) *full-system*, which applies table prediction after filtering E to entities from the top N ranked tables. For *full-system*, we limit the retrieved candidate entities to $N = 50$ tables, as we find that this consistently keeps schema entities below 3% per type (A) while maintaining high table recall, ensuring manageable prompt lengths for prediction.

RASL$_{\text{full}}$ out-performs all baselines, with similar $R@5$ and $R@15$, demonstrating the effectiveness of our dual-stage retrieve-then-predict apporach for precise table identification using rich granular context. While RASL$_{\text{retriever}}$ shows lower recall at $N = 5$, analyses in E attributes this to highly overlapping columns/tables across databases, with performance improving at higher N and consistently ranking among the top three retrieval-based methods at $N = 15$. This stronger recall at higher N proves particularly valuable when combined with entity-level schema construction, as we can significantly reduce context while preserving the most informative schema entities for table prediction and SQL generation. Figure 2 further demonstrates the advantages of our combined entity retrieval approach, which outperforms entity-specific methods on both context-rich (BIRD) and context-sparse (Fiben) datasets.

Without RASL Table Descriptions						
Model	LLM-Filtered Prediction			Initial Retrieval Pool		
	Spider (R@5)	BIRD (R@5)	Fiben (R@5)	Spider (R@N)	BIRD (R@N)	Fiben (R@N)
BM25	87.0	71.8	41.5	87.7	72.3	43.4
SXFMR	88.0	66.9	40.1	89.0	70.1	41.8
CRUSH$_{\text{BM25}}$	91.7	83.5	57.1	96.4	86.7	61.1
CRUSH$_{\text{SXFMR}}$	94.2	93.4	58.2	97.6	97.7	73.7
RASL$_{\text{full}}$	**97.0**	**97.5**	**69.1**	99.3	98.1	90.6
With RASL Table Descriptions						
Model	LLM-Filtered Prediction			Initial Retrieval Pool		
	Spider (R@5)	BIRD (R@5)	Fiben (R@5)	Spider (R@N)	BIRD (R@N)	Fiben (R@N)
BM25	95.6	84.8	68.9	96.7	86.5	99.9
SXFMR	96.2	83.2	70.0	99.1	87.8	99.9
CRUSH$_{\text{BM25}}$	95.4	87.8	70.7	100.0	95.0	99.1
CRUSH$_{\text{SXFMR}}$	94.5	93.1	67.1	100.0	99.5	99.9
RASL$_{\text{full}}$	**98.2**	**97.5**	**77.6**	99.1	98.9	90.4

Table 3: Performance comparison across datasets with and without RASL table descriptions. LLM-Filtered Prediction shows the recall@5 after filtering, while Initial Retrieval Pool shows the table recall@N of the initial retriever output prior to filtering, where N varies by model and dataset.

5.6 Impact of Retrieval Mechanism on Table Prediction

We believe that RASL's primary strength lies in efficiently loading relevant context for precise table identification within manageable context budgets, rather than standalone context retrieval. To validate this, we compare RASL$_{\text{retriever}}$ against BM25, SXFMR, and CRUSH for context retrieval prior to LLM-based table prediction. However, since these methods operate at different granularities–RASL employs multi-entity retrieval, BM25 and SXFMR work at the table level, and CRUSH operates at the column level–we implement a standardized protocol to ensure fair comparison across these diverse approaches:

- For RASL: Filter entities to those from top N=50 tables by relevance
- For baselines: Add schema elements until reaching RASL's context budget
  - BM25/SXFMR: Add full table schemas

- CRUSH: Add column names, where the first column includes the table name
- All baselines: Include all benchmark-provided E_Λ in constructed schemas

Given $\lambda_{\text{table descr.}}$'s substantial length compared to other entity types, we evaluate performance both with and without it, adjusting baseline context budgets accordingly. While this approach doesn't guarantee identical contexts, it ensures RASL's context size never exceeds baselines.

Results in Table 3 show RASL significantly outperforming baselines in both settings, demonstrating strong synergy between multi-entity retrieval and table prediction. Without table descriptions, RASL achieves higher initial retrieval recall than all baselines under equal context budgets. With table descriptions included, baselines often achieve higher initial recall by fitting more distinct schemas within the budget. However, RASL still achieves superior table prediction performance, suggesting it retrieves more relevant context. This is further evidenced by RASL's larger improvements in prediction recall versus initial pool recall when including table descriptions, indicating these descriptions provide unique value beyond other retrieved entities.

5.6.1 Impact of Table Descriptions on Context Usage. While including $\lambda_{\text{table descr.}}$ improves RASL's table prediction performance (Table 3), it significantly increases token consumption. Analysis of token usage statistics in Table 4 reveals that consumption patterns strongly depend on dataset structure, and given RASL's fixed N=50 table retrieval limit, $\lambda_{\text{table descr.}}$ token usage inversely correlates with average columns per table. For BIRD, which has 7.26 columns/table, table descriptions consume only 3% of the original schema tokens. In contrast, for Fiben, with just 2.46 columns/table and significantly smaller total database size, table descriptions exceed the entire original schema size.

While experiments validate the utility of comprehensive table descriptions, their token consumption generally outweighs performance benefits across tested datasets. Though this approach may prove valuable for settings with wide tables or high-accuracy requirements, we believe future work on more concise table context synthesis could better serve RASL under tight context budgets.

Component	Spider		BIRD		Fiben	
	Toks. (Total)	Avg. Toks. (N=50)	Toks. (Total)	Avg Toks. (N=50)	Toks. (Total)	Avg Toks. (N=50)
Table name	3,120	63	1,594	39	1,180	210
Table alias	3,178	44	-	-	-	-
Column name	15,143	65	11,257	50	1,887	339
Column alias	15,657	93	10,105	38	-	-
Column descr.	-	-	34,903	202	-	-
Value descr.	-	-	28,379	55	-	-
Table descr.	259,885	3,670	146,442	2,357	37,780	6,441
Total without table descr.	37,098	266	86,238	384	3,067	549
Total with table descr.	296,983	3,936	232,680	2,741	40,847	6,990

Table 4: Token usage breakdown by schema components with and without including $\lambda_{\text{table descr.}}$. We compare the total database schema tokens to the average tokens used by $\text{RASL}_{\text{retriever}}$ at $N = 50$. Each token is approximately 3.5 characters.

5.7 RASL Component Ablation

Table 5 presents our analysis of keyword-level retrieval and entity-type relevance score calibration. Maintaining our experimental setup of $\text{RASL}_{\text{full}}$ with N=50 table filtering, we evaluate both final table prediction and initial retrieval pool recall. Results show keyword-based retrieval (K) significantly outperforms question-based retrieval (q) across all datasets, demonstrating the importance of granular search queries for E retrieval. While combining both approaches ($K \cup \{q\}$) yields slight improvements in most settings, keyword-based retrieval remains the primary driver of performance.

The impact of entity-type weight calibration (W_Λ) varies by dataset, providing substantial gains for Spider but only modest improvements for BIRD; results are excluded for Fiben due to no training samples being available. We observe that while W_Λ can be beneficial, it is not a necessary component of RASL, which demonstrates strong utility even in settings where no labeled data exists.

		$\text{RASL}_{\text{full}}$		$\text{RASL}_{\text{retriever}}$
Dataset	Configuration	R@5	R@15	R@50
Spider	$K \cup \{q\}$	**97.0**	98.0	**99.3**
	K only	96.6	**98.2**	99.1
	q only	92.7	93.2	97.8
	$K \cup \{q\}$ w/o W_Λ	94.3	95.8	96.8
BIRD	$K \cup \{q\}$	**97.5**	**97.8**	98.1
	K only	96.8	97.1	**98.3**
	q only	90.1	90.1	95.8
	$K \cup \{q\}$ w/o W_Λ	97.2	97.4	98.0
Fiben	$K \cup \{q\}$	**69.1**	**69.2**	**90.6**
	K only	**69.1**	**69.2**	89.7
	q only	67.4	67.5	79.4
	$K \cup \{q\}$ w/o W_Λ	-	-	-

Table 5: Ablation study on the impact of retrieval query type (K vs. q) and entity-level weight calibration (W_Λ). $\text{RASL}_{\text{full}}$ filters E to the top $N = 50$ tables prior to table prediction, with recall reported over final table prediction and initial candidate tables.

5.8 Error Analysis

Next we cover some common error cases observed across baseline methods. In Figure 3 we show how assumptions in CRUSH schema hallucination can impact retrieval performance. In this case, we see that an incorrect *person* table name causes all segments to over-index on people-related columns and associated tables. In contrast, RASL uses granular and isolated keywords directly extracted from the question. When paired with entity-level retrieval, this allows for highly relevant specific tables and columns to be loaded from any keywords extracted from the question, without assumptions on how the schema is structured.

In Figure 4 and 5 we show the most common causes of error in table retrieval baselines, which is insufficient granular retrieval context. We see that important keywords, such as *circuits* in Figure 4 or *cards* in Figure 5 are not sufficiently captured in table-level similarities, resulting in necessary tables being missed.

Incorrect CRUSH hallucinated schema

Question: Where is Amy Firth's hometown? Hometown refers to city, county, state

Ground Truth Tables: ['student_club.member', 'student_club.zip_code']

Ground Truth Columns: ['member.first_name', 'member.last_name', 'member.zip', 'zip_code.city', 'zip_code.state', 'zip_code.county', 'zip_code.zip_code']

CRUSH$_{\text{SXFMR}}$

Hallucinated Schema: ['person.first_name', 'person.last_name', 'person.hometown_city', 'person.hometown_county', 'person.hometown_state']

Top 5 Tables: ['human_resources.employee', 'works_cycles.person', 'movie_3.actor', 'address.state', 'address.country']
Index of Correct Table: {'student_club.member': 31, 'student_club.zip_code': 30}

RASL

Keywords: ['person' 'town', 'city', 'county', 'state', 'name', 'first name', 'last name', 'location']

Top 5 Tables: ['law_episode.person', 'regional_sales.‘store locations‘', **'student_club.zip_code'**, **'student_club.member'**, 'retail_complains.district']

Figure 3: Example of schema hallucination leading to poor matching by CRUSH.

Failure of Semantic Table Retrieval

Question: How many formula_1 races took place on the circuits in Italy?

Ground Truth Tables: ['formula_1.circuits', 'formula_1.races']

Ground Truth Columns: ['formula_1.circuits.circuitid', 'formula_1.circuits.country', 'formula_1.races.circuitid']

SXFRMR

Top 3 Tables: ['formula_1.qualifying', 'formula_1.results', 'formula_1.constructorstandings']

Index of Correct Table: {'formula_1.races': 4, 'formula_1.circuits': 24}

RASL

Keywords: ['formula_1', 'races', 'country', 'circuits', 'race circuits']

Top 3 Tables: ['formula_1.circuits', 'formula_1.races', 'formula_1.qualifying']

Figure 4: Failure of semantic table retriever due to lack of granular context.

Failure of Lexical Retrieval

Question: List down the name of artists for cards in Chinese Simplified. Chinese Simplified' is the language;

Ground Truth Tables: ['card_games.cards', 'card_games.foreign_data']

Ground Truth Columns: ['cards.artist', 'cards.uuid', 'foreign_data.language', 'foreign_data.uuid']

BM25

Top 3 Tables: ['language_corpus.langs', 'mondial_geo.language', 'mondial_geo.religion']

Index of Correct Table: {'card_games.cards': 61, 'card_games.foreign_data': 15}

RASL

Keywords: ['artists', 'cards', 'language', 'name']

Top 3 Tables: ['card_games.cards', 'card_games.foreign_data', 'card_games.set_translations']

Figure 5: Failure of lexical table retriever due to lack of granular context.

5.9 End-to-End SQL Generation

In Table 6 we compare the performance of RASL to retrieval-based baselines on end-to-end SQL generation. This evaluation applies the full RASL pipeline with the same setting as previously, where all E belonging to the top $N = 50$ tables by relevance are used for table prediction. For SQL generation we load the full original schema for each predicted table, retaining only Λ provided by the benchmarks and excluding $\lambda_{\text{table descr.}}$ from RASL due to increased token consumption outweighing benefits for practical enterprise applications. Due to relatively low recall at $N = 5$ for all baseline methods, we evaluate performance over $N = 15$ and $N = 30$, as well as compare to the standard single database text-to-SQL setting where all tables corresponding to the target database are loaded. We report text-to-SQL execution accuracy and table recall with respect to ground truth SQL queries as our primary metrics. We also list the average number of tokens used to construct schemas used in prompting. Since RASL contains two prompting steps (table prediction and SQL generation), we sum the total schema tokens over both steps for RASL.

We see that RASL consistently ranks best in both SQL table recall and execution accuracy compared with all baselines. We note that for both BIRD and Spider, RASL never predicts over 15 tables, accounting for identical metrics for both $N = 15$ and $N = 30$. We also observe that RASL constructs token-efficient schemas, being the most efficient method at $N = 30$ and ranking second at $N = 15$. For both Spider and BIRD, we see that on average the table prediction schema accounts for approximately 2/3 of the total schema tokens, with the final SQL generation over full schemas from selected tables accounting for the other 1/3. A detailed cost analysis based on commercial API pricing is provided in Appendix G.

	BIRD						Spider					
	N=15			N=30			N=15			N=30		
Model	Acc.	Recall	Tokens	Acc.	Recall	Tokens	Acc.	Recall	Tokens	Acc.	Recall	Tokens
BM25	47.4	84.6	**4,057**	49.5	89.3	8,577	58.3	87.0	1,304	59.3	89.2	2,662
SXFMR	43.6	82.6	4,989	48.0	89.1	8,993	58.0	85.7	**1,182**	59.0	87.9	2,364
CRUSH$_{\text{BM25}}$	43.5	80.92	5,133	50.1	87.5	9,978	60.1	89.3	1,395	62.5	90.3	2,764
CRUSH$_{\text{SXFMR}}$	49.4	91.6	5,589	52.9	93.0	10,385	55.9	87.0	1,374	58.0	91.6	2,744
RASL$_{\text{full}}$	**53.5**	**94.6**	4,696	**53.5**	**94.6**	**4,696**	**64.5**	**92.5**	1,266	**64.5**	**92.5**	**1,266**
Gold	54.0	96.7	2,489	54.0	96.7	2,489	69.0	99.5	389	69.0	99.5	389

Table 6: Execution accuracy and table recall of RASL versus baselines on end-to-end SQL generation. Token counts represent total schema tokens consumed for each method, including both table prediction and SQL generation steps for RASL. Bold indicates best method.

6 DISCUSSION AND NEXT STEPS

RASL successfully addresses the challenge of scaling text-to-SQL systems to enterprise-level databases through an effective component-based retrieval architecture. Our experiments demonstrate significant performance improvements over baseline retrieval-based methods while maintaining practical context budgets. We outline key insights, limitations, and future research directions that emerge from our work.

6.1 Key Insights

Entity-level decomposition proves highly effective for context retrieval under token constraints. By decomposing both questions and schemas into granular semantic units, RASL preserves critical information while scaling to massive schemas without requiring domain-specific training. This approach shows particular strength in context-rich environments like BIRD, where entity-level retrieval achieves greater recall improvements over table-level methods, which often fail at covering important details within user questions.

Two-stage retrieval-prediction creates a powerful synergy that consistently outperforms alternative methods. Our approach first narrows the search space through multi-entity retrieval, then applies LLM reasoning to identify the most relevant tables. While RASL$_{\text{retriever}}$ does not excel directly at stand-alone high-precision table identification, we find that it retrieves more valuable context than baseline retrieval methods under the same token budget, with improved performance on table prediction. Furthermore, we show that our end-to-end system consistently enables higher SQL generation accuracy than baseline methods while consuming fewer overall input tokens.

6.2 Limitations

While we validate that synthesizing additional semantic context such as table descriptions can further improve performance, their inclusion creates significant token consumption trade-offs, particularly for databases with few columns per table. We believe that although promising, the proposed approach requires more refinement before additional synthesized context can provide low-cost performance benefits.

For all evaluations, we apply RASL to retrieve context relating to the top $N = 50$ tables by relevance. While absolute token usage at this setting is significantly lower than LLM context window budgets, we believe that a deeper analysis of performance over different N values would help identify optimal trade-offs between context size and retrieval effectiveness.

Lastly, while RASL proves effective using independent entity retrieval, we believe that further information may be contained in relative relevance across entity types. For instance, it may be beneficial to increase column-level entity scores when multiple distinct entities are retrieved from the same table.

6.3 Future Directions

Several promising research directions emerge from our work. Synthesizing more concise and granular table context entities may improve retrieval quality while keeping token consumption manageable. Exploring dynamic entity-level token allocation based on database characteristics could further enhance performance. Additional opportunities include extending evaluation of RASL over cross-database queries, integrating with recent advances in SQL generation techniques (e.g., multi-prompting and self-verification), and using agentic retrieval approaches to iteratively retrieve context using guided keyword searches based on past retrieval observations.

RASL represents a significant step toward practical natural language interfaces for massive database environments. By addressing the critical bottleneck of schema linking at scale without requiring specialized fine-tuning, it enables more accessible deployment of text-to-SQL systems across diverse enterprise settings. Future work building on this foundation has the potential to further bridge the gap between natural language understanding and database access.

7 CONCLUSION

In this work, we present RASL, a zero-shot framework for scaling natural language querying to massive databases. RASL decomposes database schemas into granular semantic entities, retrieves relevant context via calibration-enhanced similarity to important question components, and reasons over the resulting reduced schema for predicting relevant tables and generating SQL queries. We demonstrate that RASL out-performs baselines across multiple datasets varying in database size, relational information, and available semantic context. While prior works addressing this challenge often rely on domain-specific fine-tuning, which complicates deployment, RASL is designed to be robust to database schema and context changes–only requiring syncing the vector database with no model training required–and can be easily deployed in serverless computing environments.

REFERENCES

[1] Anthropic. 2024. Claude 3.5 Haiku. https://www.anthropic.com/claude/haiku

[2] Anthropic. 2024. Claude 3.5 Sonnet-v2. https://www.anthropic.com/news/claude-3-5-sonnet

[3] Amazon Web Services (AWS). [n. d.]. Bedrock Knowledge Base. https://docs.aws.amazon.com/bedrock/latest/userguide/knowledge-base-build.html

[4] Amazon Web Services (AWS). [n. d.]. Bedrock Pricing. https://aws.amazon.com/bedrock/pricing/

[5] Dorian Stuart Brown. [n. d.]. Okapi BM25 Algorithm. https://pypi.org/project/rank-bm25/

[6] Peter Baile Chen, Yi Zhang, and Dan Roth. 2025. Is Table Retrieval a Solved Problem? Exploring Join-Aware Multi-Table Retrieval. arXiv:2404.09889 [cs.IR] https://arxiv.org/abs/2404.09889

[7] Cohere. 2022. Cohere Embed v3. https://cohere.com/blog/introducing-embed-v3

[8] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. arXiv:2308.15363 [cs.DB] https://arxiv.org/abs/2308.15363

[9] 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. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599 [cs.AI] https://arxiv.org/abs/2411.08599

[10] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. arXiv:2103.12011 [cs.CL] https://arxiv.org/abs/2103.12011

[11] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426 [cs.CL] https://arxiv.org/abs/2406.08426

[12] Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. 2023. CRUSH4SQL: Collective Retrieval Using Schema Hallucination For Text2SQL. arXiv:2311.01173 [cs.CL] https://arxiv.org/abs/2311.01173

[13] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. arXiv:2405.07467 [cs.CL] https://arxiv.org/abs/2405.07467

[14] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. arXiv:2305.03111 [cs.CL] https://arxiv.org/abs/2305.03111

[15] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. arXiv:2408.07702 [cs.CL] https://arxiv.org/abs/2408.07702

[16] OpenAI. [n. d.]. gpt-3.5-turbo-0125. https://platform.openai.com/docs/models

[17] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2024. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. arXiv:2410.01943 [cs.LG] https://arxiv.org/abs/2410.01943

[18] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL] https://arxiv.org/abs/1908.10084

[19] Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Ozcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.* 13, 11 (2020), 2747–2759.

[20] sentence transformers. [n. d.]. all-mpnet-base-v2. https://huggingface.co/sentence-transformers/all-mpnet-base-v2

[21] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755

[22] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL] https://arxiv.org/abs/2312.11242

[23] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. DBCopilot: Natural Language Querying over Massive Databases via Schema Routing. arXiv:2312.03463 [cs.CL] https://arxiv.org/abs/2312.03463

[24] 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. 2019. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. arXiv:1809.08887 [cs.CL] https://arxiv.org/abs/1809.08887

A SCHEMA ENTITY USAGE BY $N = 50$ TOP RELEVANT TABLES

Below we show the percent of total database schema entities used with entities filtered by top $N = 50$ distinct tables over Spider and BIRD benchmarks. We see that total schema is consistently reduced to less then 3% of the total schema entities for Λ_T, and less than 1% of total schema entities for Λ_C.

	Dataset	
Entity Type	Spider	BIRD
Table Name	1.78%	1.99%
Table Alias	1.27%	-
Table Descr.	1.32%	2.18%
Column Name	0.50%	0.42%
Column Alias	0.66%	0.37%
Column Descr.	-	0.49%
Value Descr.	-	0.29%

Table 7: Entity usage across different datasets

B EXPLORATION OF ENTROPY-GUIDED KEYWORD-LEVEL WEIGHT CALIBRATION

Information retrieval systems often struggle with keywords that have varying levels of specificity–some terms may be highly specific to certain schema elements, while others may be generic and match well with many elements. To address this challenge, we explored applying the entropy-guided similarity approach introduced by CRUSH [12], adapting it to handle our keyword-level retrieval approach.

Building on CRUSH's core insight that keyword specificity should influence matching weights, we developed a calibration system that automatically identifies and adjusts for differences in keyword discriminative power. For example, a keyword like "latitude" that strongly matches only geographic coordinates should have more influence than "value" which may match well with many different schema elements.

We quantify this specificity using an information theory approach. For each keyword k and entity type λ, we first compute a probability distribution over matching entities:

$$p(e|k,\lambda) = \frac{\exp(\alpha \cdot r_0(e,k))}{\sum_{e' \in C_{k,\lambda}} \exp(\alpha \cdot r_0(e',k))} \tag{2}$$

where $r_0(e,k)$ is the initial relevance score and α is a scaling factor that helps differentiate between similar scores. We then measure how focused or diffuse these matches are using entropy:

$$H(k,\lambda) = -\sum_{e \in C_{k,\lambda}} p(e|k,\lambda) \log p(e|k,\lambda) \tag{3}$$

A low entropy indicates the keyword strongly prefers certain schema elements (high specificity), while high entropy suggests the keyword matches broadly across many elements (low specificity).

We adjust the final relevance scores:

$$r(e,k) = r_0(e,k) \cdot \sigma(\alpha(\bar{H}_\lambda - H(k,\lambda))) \tag{4}$$

where σ is the sigmoid function and $\bar{H}_\lambda$ is the mean entropy for entity type λ. This calibration is designed to automatically reduce the influence of generic keywords while amplifying the impact of specific ones. The scaling parameter α controls the sharpness of the probability distribution.

While beneficial in CRUSH, we found that the inclusion of keyword-level entropy calibration had minimal impact on performance, with marginal impact at low α (1.0) and negative impact at higher α (2.0, 3.0, 5.0, 10.0). One possible explanation for this is that sharp distributions with few informative e for a given (k,λ) will result in all e from that (k,λ) being up-weighted, leading to more non-informative e being included. Another possible explanation is that flat distributions with many high relevance e may not necessarily indicate they are uninformative. For instance, we see in E that there can be commonly named tables and columns across many databases. However, it may be more beneficial to include these at higher retrieval budgets, rather than down-weighting them due to lower specificity. For these reasons, we have excluded this component from our results, but we believe additional refinement of this method in future works may benefit RASL's performance.

C LLM PROMPTS

C.1 Table Prediction Prompt

```
You are a database expert assistant that helps identify which tables are relevant to answering SQL questions.

TASK:
Analyze the provided database schema and the user's question, then identify which specific tables (and their databases) are most relevant for answering the question. You must rank tables in strict order of relevance.

### Schema:
{SCHEMA}

### User Question:
{QUESTION}

### Instructions:
1. Examine the question carefully to understand what data would be needed to answer it
2. Analyze the database schema to determine which tables contain relevant information
3. Rank tables by relevance - tables listed first should be most central to answering the question
4. Consider both direct mentions and implied data needs
5. Select only tables that would contribute to a SQL query answering the question
6. Consider join paths needed to connect relevant information
7. IMPORTANT: The table schemas are incomplete and only contain possibly relevant columns. There are many columns not shown within each table.
```

```
Ranking Criteria:
- Primary tables: Directly contain data explicitly asked for in the question
- Secondary tables: Needed for joins or containing supplementary information
- Tertiary tables: Might be useful for contextual information but not essential

First, think through your reasoning step by step. Carefully consider how to rank the relevance of each table.

Then provide your answer in the following XML format:

<thinking>
Your detailed analysis explaining why specific tables are relevant to the question and how you determined their ranking order.
</thinking>

<relevant_tables>
<database name="database_name">
<!– Tables in strict order of relevance, most relevant first –>
<table rank="1">most_relevant_table</table>
<table rank="2">second_most_relevant_table</table>
<table rank="3">third_most_relevant_table</table>
</database>
<database name="another_database_name">
<table rank="1">most_relevant_table_in_this_db</table>
<!– Additional tables in decreasing relevance –>
</database>
</relevant_tables>

IMPORTANT:
- List tables in strict order of decreasing relevance within each database
- The "rank" attribute should reflect overall relevance across all databases (1 = most relevant overall)
- Only include tables that are genuinely relevant to answering the question
- If no tables from a particular database are relevant, do not include that database
- Only a subset of columns are shown for each table. Leverage these when relevant, but DO NOT assume the table is missing any columns
```

C.2 End-to-End SQL Generation Prompt

Below we show the prompt for end-to-end SQL generation. Since all datasets evaluated contain single-database target SQLs, we apply a prompt while involves first predicting the correct database, and then predicting the correct SQL over that database. For self-correction performed in experiments, we apply the same system prompt, with follow-up messages on the specific error encountered, or that the output table is empty if no results are returned, until the number of self-correction iterations is reached or a populated table is returned.

For extensions to cross-database querying using compatible SQL engines, the database prediction tag can be removed and instructions added on how to properly format tables with database name prefixes in the generated SQL query. There is no single-database constraint on context retrieval or schema construction.

```
You are a data science expert.
Below, you are presented with a database schema and a question.
Your task is to generate a SQL query to answer the question.
{DIALECT_INSTRUCTION}

### Database Schema
{DATABASE_SCHEMA}

### Question
{QUESTION}

First, think through your reasoning step by step. If there are multiple databases, determine which one you should use, and then carefully consider how to rank the relevance of each table.
Then provide your answer in the following XML format:

<thinking>
Your detailed analysis explaining why specific tables are relevant to the question and how you determined their ranking order.
</thinking>

<database>
The database you are executing the SQL query on
</database>

<sql_query>
Your executable SQL query
</sql_query>

IMPORTANT:
- Pay close attention to the specific columns used for selections and filtering, ensuring they are the correct ones.
- Pay close attention to any value formats provided in the question, as well as specific values. For value conditions, if the user question specifies a specific value, follow this closely.
- Think step by step to find the correct SQL query.
```

C.3 Keyword Extraction

The keyword extraction prompt applied is adapted from CHESS [21], where various few-shot examples are provided to guide the language model. We adopt their core structure, while removing instructions on value extraction, which CHESS uses to find relevant schema context based on specific values referenced in user questions. We do not explore additional indexing of database values, as this can lead to extreme cost and complexity for massive enterprise datasets with terabytes of data and constantly changing values.

You will be provided with a user question that can be answered by querying some database system. Your objective is to analyze the question to identify and extract keywords and keyphrases which might help indicate what parts of the database schema to use. These elements are crucial for understanding the core components of the question provided. This process involves recognizing and isolating significant terms and phrases that could be instrumental in formulating searches or queries related to the posed question. You should focus on entities such as column or table names that may be referenced, as well as descriptions of what these are. Do not focus on specific column values that may be referenced.

Instructions

- Read the Question Carefully: Understand the primary focus and specific details of the question. Look for any named entities types (such as organization, location, etc.), technical terms, and other phrases that encapsulate important aspects of the inquiry.

- Keywords: Single words that capture essential aspects of the question or hint.
- Keyphrases: Short phrases or named entity types that represent specific concepts, locations, organizations, or other significant details.

Example 1:
Question: "What is the annual revenue of Acme Corp in the United States for 2022? Focus on financial reports and U.S. market performance for the fiscal year 2022."

["annual revenue", "corporations", "country", "year", "financial reports", "U.S. market performance", "fiscal year", "corporate revenues"]

Example 2:
Question: "In the Winter and Summer Olympics of 1988, which game has the most number of competitors? Find the difference of the number of competitors between the two games. the most number of competitors refer to MAX(COUNT(person_id)); SUBTRACT(COUNT(person_id where games_name = '1988 Summer'), COUNT(person_id where games_name = '1988 Winter'));"

["olympic games", "competitors", "number of competitors", "person_id", "games", "games_name", "competitors competing in olympic games"]

Example 3:
Question: "How many Men's 200 Metres Freestyle events did Ian James Thorpe compete in? Men's 200 Metres Freestyle events refer to event_name = 'Swimming Men"s 200 metres Freestyle'; events compete in refers to event_id;"

["events", "event_id", "event_name", "compete in", "competitors", "competitive games", "competitors competing in events"]

Example 4:
Question: "List the infant mortality of country with the least Amerindian."

["mortality rate", "infant mortality rate", "country", "ethnicity", "population", "infant mortality"]

Example 5:
Question: "What are the first names of the students who live in Haiti permanently or have the cell phone number 09700166582?"

['students', 'first names', 'country', 'permanent address', 'cell phone number', 'contact information', 'student location', 'student contact details']

Task

Given the following question, identify and list all relevant keywords and keyphrases which may indicate which parts of a database schema might be necessary to answer the user question.

Question: {QUESTION}

Please provide your findings as a json list, capturing the essence of the question through the identified terms and phrases.
Only output the json list with no explanations.

C.4 Table Description Synthesis Prompt

Table descriptions are synthesized using the below LLM prompt, where table schemas are generated following D.

You are a database expert creating semantic table descriptions for a text-to-SQL retrieval system.

TASK
Generate a concise, high-level description of the provided database table that captures its semantic purpose and usage patterns. This description will be embedded in a structured XML format and used for retrieving relevant tables when processing natural language questions.

DATABASE SCHEMA
{TABLE_SCHEMA}

```
First, think through what makes this table important, what business concepts it represents, and how users might refer to it in natural language questions. Consider its likely role in the database without listing all columns.

Then, create a concise table description that covers:
1. The table's main purpose and real-world concept it represents
2. Its context within the broader database domain
3. Typical query patterns or business questions it helps answer
4. Key relationships with other tables (if any)
5. Alternative terms users might use when referring to this table

Keep your description under 150 words, focusing on semantic meaning rather than technical details. For tables with many columns, focus on the overall table purpose and categories of data rather than describing individual columns.

Format your response as follows:
<thinking>
Your analysis of the table and reasoning about its purpose, usage, and significance.
</thinking>

<description>
Paragraph 1: Purpose and domain context of the table in 2-3 sentences.
Paragraph 2: Usage patterns and typical questions this table answers in 2-3 sentences.
Paragraph 3: Key relationships with other tables in 1-2 sentences (if applicable).
Alternative terms: comma-separated list of 3-6 alternative phrases users might use.
</description>
```

D SCHEMA FORMAT

Below is the schema format used for all LLM operations (table prediction and SQL generation). We adopt a format similar to M-Schema [9], with XML tags substituted for better alignment with Anthropic Claude models. For any table-level entities we automatically add *table_name* from the metadata, and for all column-level entities we add in *column_name* and *data_type*. We have a separate XML section for *table_description*, as this is the only table-level contextual entity used, although other sections can be added if additional information types exist. For column-level context entities, we map from a short description of the context type to the value, with the specific example below shown for the BIRD dataset. Foreign keys are added at the database-level when available.

```
<db>$db_id

<table>$table_name
<desc>
$table_description
</desc>

<schema>
($column_name:$data_type, $column_alias, Column description: $column_description, Value description: $value_description),
($column_name:$data_type, $column_alias, Column description: $column_description, Value description: $value_description),
...
</schema>
</table>

<table>$table_name
<desc>
$table_description
</desc>

<schema>
($column_name:$data_type, ...),
($column_name:$data_type, ...),
...
</schema>
</table>

...

<foreign_keys>
$table_name.$column_name=$table_name.$column_name
...
</foreign_keys>

</db>

<db>$db_id

...

</db>
```

E ANALYSIS OF CONFLICTING INFORMATION OVERLAP

Below we highlight some common causes for low table-level recall at low N values due to highly overlapping table/column information across full datasets. We observe that while performance is affected at low N, this effect is eliminated at higher N values, while still reducing overall database schema to a small fraction (e.g. $< 3\%$ of E_{Λ_T} and $< 1\%$ of E_{Λ_C} at $N = 50$) of the original schema.

E.0.1 Example 1 (Spider). **Question**: Find the districts in which there are both shops selling less than 3000 products and shops selling more than 10000 products.
Ground truth tables: {'employee_hire_evaluation.shop'}
Ground truth columns: {'employee_hire_evaluation.shop.district', 'employee_hire_evaluation.
shop.number_products'}

RASL-Extracted Keywords: [districts, shops, products, product quantity, shop inventory, retail locations]

Finding: 19 tables contain the word 'products', 6 contain 'shop', and 8 contain 'store'. Similarly, 51 columns contain 'product', 14 contain 'shop', and 11 contain 'district'. While entropy-guided relevance calibration down-weights keywords such as column-level 'product', the effect is less pronounced on high overlap across only 5-15 entities, requiring larger top-N values to load all relevant context.

E.0.2 Example 2 (BIRD). **Question**: Where is Amy Firth's hometown? hometown refers to city, county, state
Ground truth tables: {'student_club.member', 'student_club.zip_code'}
Ground truth columns: {'student_club.member.first_name', 'student_club.member.last_name', 'student_club.member.zip', 'student_club.zip_code.city', 'student_club.zip_code.county', 'student_club.zip_code.state', 'student_club.zip_code.zip_code'}

RASL-Extracted Keywords: ['hometown', 'city', 'county', 'state', 'location', 'residence', 'person details']

Findings: 7 tables contain the word 'city', 7 contain 'location', and 5 contain 'state. Similarly, 19 columns contain 'city', 16 contain 'state', and 13 contain 'location'. Due to this highly overlapping information of similar data across different databases, it can be difficult to retrieve top-N context entities at low N values, whereas at higher N values (e.g. 20-50), all sufficient context can be included for schema reasoning.

F BASELINE METHOD IMPLEMENTATION DETAILS

For *Recall*@5 and *Recall*@15 metrics, we directly adopt the metrics reported by DBCopilot [23] on external benchmarks, which serializes table schemas into documents containing table names, column names, and column context for use in BM25, SXFMR, and DTR. BM25 uses Okapi BM25 with the two adjustable parameters optimized over training, and SXFMR uses *all-mpnet-base-v2* [20]. CRUSH uses *gpt-3.5-turbo-0125* [16] for schema hallucination and adopts the same BM25 and SXFMR settings. DTR and DBCopilot are fine-tuned using 1×10^5 question-SQL pairs, which are synthesized over the target databases using the process detailed in [23].

For ablation study reproductions, we leverage the same settings for BM25 and SXFMR, where we find BM25 works best on training samples using shingles of $K = 4$ for Spider and $K = 5$ for BIRD and Fiben, with word-level indexing performing poorly across all datasets. For CRUSH, the same settings are used for BM25 and SXFMR, but we leverage Anthropic Calude 3.5 Sonnet-v2 [2] for schema hallucination due to access constraints for OpenAI models.

G COST ANALYSIS

A detailed input token cost analysis of RASL versus baseline methods for end-to-end SQL generation is provided in Table 8. We adopt standard on-demand commercial API pricing provided by Amazon Bedrock [4], which charges \$0.0008 per 1,000 input tokens for Anthropic Claude 3.5 Haiku, \$0.003 per 1,000 input tokens for Anthropic Claude 3.5 Sonnet v2, and \$0.0001 per 1,000 input tokens for Cohere Embed 3 English. Cost metrics are broken down by retrieval, LLM prompt, and schema components, with all values calculated per 100 questions to improve readability. LLM prompt includes both the table prediction and SQL generation prompt steps for RASL and only SQL generation for baselines, with the schema prompt component removed to isolate variable costs.

The results demonstrate RASL's superior cost scaling characteristics despite retrieval overhead. While RASL incurs additional costs for retrieval (\$0.03 per 100 queries) and more complex prompting (\$0.39-0.41 vs \$0.10-0.11 for baselines), its key advantage lies in maintaining constant costs as the number of tables increases. Most notably, RASL costs remain identical at N=15 and N=30, while baseline costs scale linearly with the number of included tables. For Spider, RASL costs \$0.80 per 100 queries at both N=15 and N=30, compared to baselines ranging from \$0.45-0.52 at N=15 but increasing to \$0.81-0.93 at N=30. For BIRD, RASL costs \$1.85 per 100 queries versus \$1.32-1.78 for baselines at N=15, but baselines increase substantially to \$2.68-3.22 at N=30. This constant cost scaling, combined with superior accuracy performance, makes RASL particularly attractive for enterprise deployments where database catalogs continue to grow over time.

We believe further optimizations could provide additional cost benefits. Prompt reduction techniques could minimize the fixed prompt overhead (\$0.39-0.41 per 100 queries), while evaluations of lighter-weight models for table prediction (e.g., using Claude Haiku instead of Sonnet) represent promising avenues for cost reduction that warrant investigation. These optimizations represent important directions for future work to enhance RASL's cost-effectiveness in resource-constrained environments.

Method	Spider (per 100 queries)				BIRD (per 100 queries)			
	Schema (N=15/30)	Retrieval	Prompt	Total (N=15/30)	Schema (N=15/30)	Retrieval	Prompt	Total (N=15/30)
BM25	$0.39/0.80	$0.00	$0.10	$0.49/0.90	$1.22/2.57	$0.00	$0.11	$1.32/2.68
SXFMR	$0.35/0.71	$0.00	$0.10	$0.45/0.81	$1.50/2.70	$0.00	$0.11	$1.60/2.80
CRUSH_BM25	$0.42/0.83	$0.00	$0.10	$0.52/0.93	$1.54/2.99	$0.00	$0.11	$1.65/3.10
CRUSH_SXFMR	$0.41/0.82	$0.00	$0.10	$0.51/0.92	$1.68/3.12	$0.00	$0.11	$1.78/3.22
RASL_full	$0.38/0.38	$0.03	$0.39	$0.80/0.80	$1.41/1.41	$0.03	$0.41	$1.85/1.85

Table 8: Cost breakdown per 100 queries (USD) showing schema, retrieval, and prompt costs.