SQuAR: Synthetic Queries as Join-Aware Retrieval Anchors for Text-to-SQL

Anonymous authors
Paper under double-blind review

Abstract

Schema retrieval is critical in text-to-SQL systems over large database collections, where a system must identify both the target database and the tables needed to answer a natural-language question. Existing approaches either rely on costly query-time LLM reasoning or struggle to preserve multi-table structural relationships as the schema pool scales. To address this limitation, we present SQuAR, an open-pool schema retrieval approach that uses synthetic questions as join-aware retrieval anchors. Offline, SQuAR samples multi-table sub-schemas along foreign-key paths and indexes schema-grounded synthetic questions as retrieval anchors. At query time, it aggregates database-level and table-level evidence from retrieved anchors without query-time LLM calls. We further show that table-level recall alone is insufficient, as missing even one required table or selecting an incorrect database makes the retrieved schema insufficient to construct the target SQL query. Thus, we introduce Execution-Ready Coverage (ERC), which requires the correct database and all required tables, and Execution-Ready Jaccard (ERJ), which additionally penalizes over-retrieval. We evaluate SQuAR on four benchmarks and their unified pool of 37 databases and 619 tables. Across the four benchmarks, SQuAR improves Table Recall and ERC by up to 17.5 and 26.1 percentage points over the strongest evaluated baselines, while also maintaining a median retrieval latency of approximately 0.1 seconds. In the unified open pool, SQuAR improves DB R@1, Table Recall, and ERC by 15.5, 8.7, and 20.1 percentage points, respectively.

1 Introduction

Relational databases are central to most modern information systems, but writing SQL queries requires specialized knowledge. Text-to-SQL bridges this gap by translating natural-language (NL) questions into SQL queries, allowing non-technical users to access structured data (Yu et al., 2018; Li et al., 2023). *Schema retrieval*, i.e., identifying the target database and relevant tables from a large pool of databases, is a fundamental bottleneck in scalable text-to-SQL systems. In a realistic enterprise setting, known as an *open-pool* setting, an NL question is not associated with a specific target database; the system must determine which database in the pool the question targets. Thus, answering an NL question requires the system to screen hundreds or thousands of tables distributed across many databases (Wang et al., 2025a; Chen et al., 2024a; Wang et al., 2025b). As a prerequisite for accurate SQL generation, the system must identify the target database and the subset of tables required to express the query (TU, 2026). This step is critical because missing a single table among those required to answer a query prevents the system from generating the correct SQL query.

Existing approaches address schema retrieval through LLM-based reasoning or retrieval-based methods. LLM-based approaches can be effective, but repeated LLM calls and full-schema prompting become costly and limit scalability as the number of databases and tables grows (Pourreza & Rafiei, 2023; Talaei et al., 2024; Xie et al., 2024; Pourreza et al., 2025; Cao et al., 2024; Chung et al., 2025; Maamari et al., 2024). Retrieval-based methods instead index schema representations offline and retrieve candidates according to their similarity to the user query (Balaka et al., 2025; Eben et al., 2025). Recent methods enrich these representations using synthetic questions, including questions over the schema (Liang et al., 2025; Chou et al., 2026; Toteja et al., 2025), while join-aware methods explicitly exploit relationships among tables during retrieval (Zhang et al., 2025; Boutaleb et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2025; Soliman et al., 2026; Chen et al., 2024b). For example, answering "Which customers have unpaid orders?" requires traversing the connected table path `Customers–Orders–Payments`. Although such structural relationships can be modeled, constructing the query-specific coherent table set may still require additional query-time search, expansion, filtering, or reasoning.

These observations point to three challenges. **Challenge 1: Complex Open-pool Setting.** Schema retrieval must scale to large database pools containing hundreds of tables while identifying the relevant database and tables; retrieval becomes difficult as schema complexity increases. **Challenge 2: Lack of Execution-Aware Measures.** For SQL generation, the retrieved schema must be *execution-ready*: it should identify the correct database, contain all tables required to express the gold-answer SQL, and exclude unnecessary schema elements, i.e., 100% recall with high precision. Missing a single required table makes the correct query impossible to generate. **Challenge 3: Question–schema Mismatch.** NL questions rarely align with database metadata, making surface-level matching difficult, especially as the candidate pool grows and contains more semantically similar tables.

We address this gap with a **query-centric, join-aware retrieval** method. Our key idea is to use NL questions that represent groups of related tables as retrieval anchors, and aggregate evidence from the retrieved anchors to identify the target database and tables. Thus, retrieval considers both the user's intent and the relationships among the required tables. Offline, we construct a schema graph, sample connected multi-table subgraphs, generate NL questions that can be answered using each subgraph, and embed these into a dense vector index. At query time, a user question retrieves similar synthetic questions, and the database and tables associated with the retrieved neighbors are aggregated through voting to produce a candidate schema for SQL generation. This requires only a single dense retrieval step per query, while encoding the multi-hop schema structure offline.

We evaluate our method on four benchmarks: Spider (Yu et al., 2018), BIRD (Li et al., 2024), and BEAVER-DW/BEAVER-NW (Chen et al., 2024a), where the task is to identify both the target database and the relevant tables from an open pool (Wang et al., 2025a;b). Existing multi-database evaluations report database-routing metrics, schema-retrieval metrics defined differently across systems, or final SQL execution accuracy. However, these metrics do not capture whether the retrieved candidate schema is execution-ready. Specifically, correct database identification alone does not guarantee complete table retrieval, and high table recall does not guarantee recovery of the complete set of tables needed to express the gold SQL. In both cases, the candidate schema may still be insufficient for generating the gold SQL even when aggregate routing or table-recall metrics appear strong, i.e., close to 100%. To evaluate this retrieval-specific failure mode, we introduce two additional metrics: *Execution-Ready Coverage* (ERC) and *Execution-Ready Jaccard* (ERJ). ERC requires the retrieved schema to identify the correct database and contain all tables needed by the ground-truth SQL, while ERJ penalizes over-retrieval among successful full-coverage cases.

Our contributions are: (1) we introduce SQuAR, a query-centric schema retrieval method designed for large open-pool settings; (2) we introduce ERC and ERJ, two retrieval metrics that measure complete, correctly routed retrieval and account for over-retrieval; and (3) we evaluate SQuAR across multiple benchmarks, demonstrating its effectiveness and efficiency and analyzing the relationship between ERC/ERJ and downstream SQL generation quality.

2 Related Work

Prompt-Based Text-to-SQL Pipelines. A line of work tackles the text-to-SQL problem by decomposing the task into LLM-guided stages, including schema selection, query decomposition, candidate generation, validation, and self-correction (Deng et al., 2025; Pourreza & Rafiei, 2023; Talaei et al., 2024; Xie et al., 2024; Pourreza et al., 2025; Cao et al., 2024). Several of these methods include a schema retrieval stage before SQL generation. However, as the number of candidate databases and their schemas grows, this stage may require more LLM calls over increasingly large schema contexts, increasing token usage, inference latency, and computational cost (Luoma & Kumar, 2026). SQuAR serves either as a standalone retriever that identifies the relevant database and tables, or as a complement to these pipelines by first retrieving a compact set of candidate databases and tables, allowing the downstream text-to-SQL method to operate over a smaller schema space. A related extension of these prompt-based pipelines reduces or eliminates explicit schema retrieval by providing the model with a large schema context in the prompt (Maamari et al., 2024; Chung et al., 2025). Long-context approaches provide large schema contexts directly to the model, but their to-

Table 1: Comparison of representative schema retrieval methods for text-to-SQL.

Method	Retrieval Unit	Synthetic Questions	Structural / Join Signal	Multi-Table Selection	Query-Time LLM Call
Pneuma	Tables	No	None	No	Yes
DBCopilot	DB + Tables	Yes	Schema graph	Yes	No
JAR	Tables	No	Join compatibility + MIP	Yes	Yes
Iterative-JAR	Tables	No	Join compatibility + iteration	Yes	Yes
CORE-T	Tables	No	Table compatibility	Yes	Yes
QGpT	Question-Augmented Tables	Yes	None	No	No
SQuAR	Multi-Table Question Anchors	Yes	FK-connected anchors	Yes	No

ken and query costs increase with schema size. Moreover, they often rely on frontier models with sufficient context capacity, raising additional privacy and data-sharing concerns.

Retrieval-Based Table Discovery and Candidate Refinement. A complementary line of work decouples schema retrieval from SQL generation by retrieving databases, tables, or schema elements before constructing the downstream SQL-generation prompt.

Table 1 summarizes representative schema retrieval methods along five design dimensions: the retrieval unit, usage of synthetic questions, incorporation of structural or join information, joint selection of multiple tables, and whether an LLM is invoked at query time. Prior retrieval methods follow two directions. Some improve semantic matching between the user question and schema representations (Balaka et al., 2025; Eben et al., 2025; Kothyari et al., 2023; Wang et al., 2025a), while others incorporate relational structure or iterative refinement to recover coherent sets of tables (Soliman et al., 2026; Chen et al., 2024b; Boutaleb et al., 2025; Zhang et al., 2025). Within the first direction, several methods reduce the semantic gap with the help of synthetic NL questions generated from tables. QGpT generates questions from table segments (Liang et al., 2025), CGPT uses synthetic questions as supervision for contrastive retriever training (Chou et al., 2026), and ICRL-RAG (Toteja et al., 2025) uses in-context reinforcement learning to improve multi-table question generation, followed by an LLM-based pooling stage over retrieved schema candidates.

These methods differ in how they distribute LLM computation between offline preprocessing and query-time retrieval. Some retrieval approaches rely primarily on non-LLM similarity or learned retrieval signals, such as lexical matching, dense embeddings, schema metadata, or table-table relevance, and therefore avoid query-time LLM calls during retrieval. Other methods invoke LLMs at query time during retrieval or refinement, such as CRUSH4SQL, CORE-T, and LinkAlign (Kothyari et al., 2023; Soliman et al., 2026; Wang et al., 2025b), increasing per-query inference cost. Several methods also shift substantial LLM computation offline by precomputing table descriptions, embeddings, or synthetic questions, as in Pneuma, QGpT, CGPT, and ICRL-RAG.

Our work is closest to the family of offline synthetic-question-based retrieval approaches. SQuAR generates schema-grounded synthetic questions during offline index construction and uses them as retrieval anchors. At query time, SQuAR aggregates database- and table-level evidence across the retrieved anchors to construct the final candidate schema. This design captures semantic and structural information during offline indexing while keeping the query-time retrieval stage lightweight and free of generative LLM calls.

3 Schema Retrieval Formulation and Evaluation

3.1 Problem Formulation

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_M\}$ be a pool of M databases. Each database $d \in \mathcal{D}$ contains a set of tables $\mathcal{T}_d = \{t_{d,1}, t_{d,2}, \ldots, t_{d,n_d}\}$, $|\mathcal{T}_d| = n_d$.

Definition 1 (Schema Retrieval)**.** Given an NL query q and a database pool D, schema retrieval aims to recover the execution-ready schema, consisting of (a) the target database $d_q^* \in D$ over which the SQL query s_q corresponding to q is expressed, and (b) the table set $T_q^* \subseteq T_{d_q^*}$ referenced by s_q.

A retrieval method outputs a predicted database $\hat{d}_q \in D$ and a ranked list of tables; we write $\hat{T}_q^{(K)}$ for the set of its top-K tables.

Definition 2 (Full Coverage)**.** A retrieval output achieves *full coverage at cutoff* K if it selects the correct database and its top-K tables include every gold table:

$$\hat{d}_q = d_q^* \quad \text{and} \quad T_q^* \subseteq \hat{T}_q^{(K)}. \tag{1}$$

Following existing benchmarks (Yu et al., 2018; Li et al., 2024; Chen et al., 2024a), each query in our evaluation is answerable within a single database. Note that full coverage is necessary but not sufficient for schema retrieval, since returning all tables in the schema trivially satisfies it. Over-retrieval increases context and token usage and introduces noise that degrades downstream SQL generation; the retrieval objective must therefore balance complete coverage with minimizing unnecessary tables. This is addressed by our novel evaluation metrics that are introduced in Section 3.2.

3.2 Schema Retrieval Evaluation

For a given natural language question, we need to address two competing issues: tables missing from the output cannot be recovered by the downstream SQL generator, while superfluous tables inflate its context and invite spurious joins. We therefore introduce two new metrics to better evaluate retrieval: (i) *Execution-Ready Coverage (ERC)*, which measures whether the correct database and all required tables are retrieved, and (ii) *Execution-Ready Jaccard (ERJ)*, which penalizes unnecessary retrieved tables. These complement traditional recall metrics used in schema retrieval evaluation.

Definition 3 (Execution-Ready Coverage (ERC))**.**

$$\mathrm{ERC@}K = \frac{1}{|Q|} \sum_{q \in Q} c_q^{(K)}, \quad c_q^{(K)} = \mathbf{1}\Big(\hat{d}_q = d_q^* \wedge T_q^* \subseteq \hat{T}_q^{(K)}\Big). \tag{2}$$

where $c_q^{(K)}$ is the complete-retrieval indicator.

ERC represents the fraction of queries in a set Q for which the retrieved schema provides full coverage of all the relevant tables. The retrieved schema for a query $q \in Q$ is execution-ready if and only if its complete-retrieval indicator $c_q^{(K)} = 1$, i.e., it identifies the correct database and retrieves all referenced tables. Otherwise, the retrieved schema is insufficient to express the gold SQL because at least one required table is missing. Appendix C.3 further illustrates why this distinction matters since Table Recall can increase with the retrieval budget even as database accuracy and ERC decrease.

Definition 4 (Execution-Ready Jaccard (ERJ))**.** ERJ additionally penalizes over-retrieval:

$$\mathrm{ERJ@}K = \frac{1}{|Q|} \sum_{q \in Q} c_q^{(K)} \cdot \frac{\left|T_q^* \cap \hat{T}_q^{(K)}\right|}{\left|T_q^* \cup \hat{T}_q^{(K)}\right|} \tag{3}$$

Because ERC does not penalize retrieving additional tables that are not needed, it equally rewards solutions that retrieve exactly the needed tables and those that retrieve many extra, unnecessary tables and create significant noisy context for downstream SQL generators. In contrast, ERJ penalizes unnecessary tables among full-coverage queries. If the correct database or any required table is missing, the score is 0; otherwise, it is the Jaccard overlap between the retrieved and required tables, reaching 1 only for an exact match, represented by the ratio $\frac{\left|T_q^* \cap \hat{T}_q^{(K)}\right|}{\left|T_q^* \cup \hat{T}_q^{(K)}\right|}$.

4 SQuAR: Open-pool Schema Retrieval

This section introduces SQuAR (Figure 1): an open-pool schema retrieval technique for text-to-SQL systems. Given a pool of databases and an NL query, the goal is to recover an execution-ready schema consisting of the correct database and the tables required to express the target SQL. SQuAR has two stages: (1) an offline stage, in which we build a vector index of synthetic questions grounded in schema structure as retrieval anchors, and (2) an online stage, in which a query is matched against this index to retrieve candidate databases and tables.

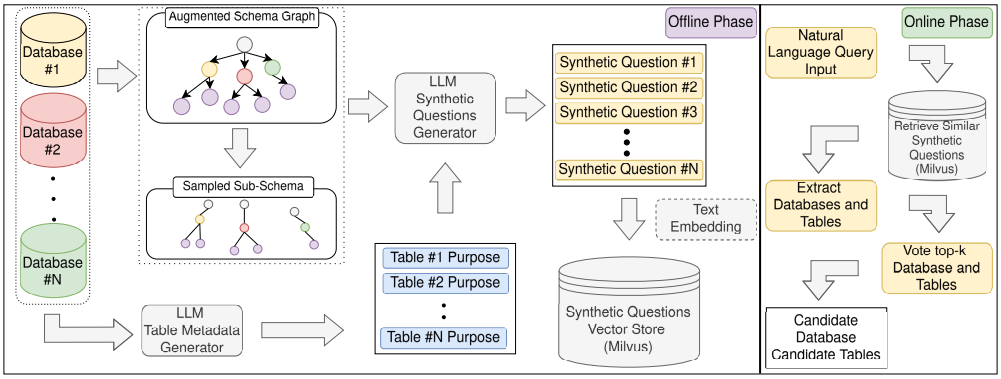


Figure 1: Overview of SQuAR.

4.1 SQuAR Offline Stage

The offline stage constructs schema-grounded synthetic questions that serve as retrieval anchors to match incoming natural language user questions with relevant subsets of the schema. The offline stage consists of the following steps: *table description generation* attaches a NL gloss for each table; *augmented schema graph construction* links related tables; *synthetic question generation* samples connected sub-schemas and generates questions answerable over them; and *question indexing* embeds all questions into a dense vector index. These steps are outlined in the rest of this section.

4.1.1 Table Description Generation

To reduce the semantic gap between schema identifiers and queries expressed in natural language, we generate a short NL description for each table in the database pool. Specifically, we prompt the LLM with the database name, table name, and up to η column names, where η depends on the LLM context length, to infer and summarize the real-world purpose of the table. We do not use sample rows or cell values.

4.1.2 Augmented Schema Graph Construction

This phase constructs an augmented schema graph $\mathcal{G}$, following Wang et al. (2025a), that represents the databases, tables, and their structural relationships in the input schema pool. Graph $\mathcal{G}$ is a utility that helps the synthetic question generator to sample related groups of tables along foreign-key paths, rather than arbitrary table combinations, thus reducing the likelihood of generating questions from structurally unrelated table combinations.

We construct a single *augmented schema graph* $\mathcal{G}$ over the entire pool of databases, rather than one graph per database. This unified representation provides a common sampling structure over the entire database pool, allowing each traversal to first select a database and then sample related tables within that database. For example, Figure 2 shows a `Sales DB` containing `Customers`, `Orders`, and `Payments`. The highlighted walk selects `Sales DB` and traverses from `Customers` to `Orders`, yielding a related table group for synthetic question generation. As shown in the left panel of Figure 2, graph $\mathcal{G}$ contains a virtual source node connected to one database node for each input database in $\mathcal{D}$, and one table node for each table across all databases. Each database node connects to all tables in that database. We derive table-to-table edges from the primary-key–foreign-key relationships provided in the database schema metadata.

Each database thus forms its own table-level subgraph: its database node connects to all of its table nodes, which are further linked by undirected foreign-key edges when such relationships exist. Tables from different databases remain disconnected from one another and are only jointly reachable through the shared source node. Since all edges are undirected, random walks can traverse join relationships in either direction.

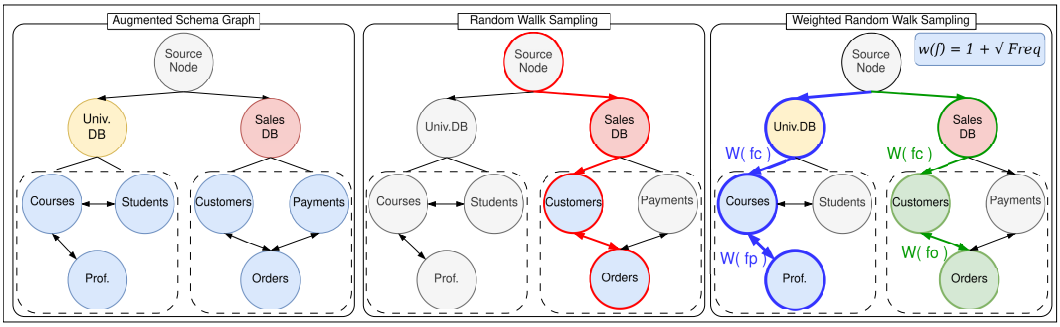


Figure 2: Augmented schema graph and random-walk sampling in SQUAR.

4.1.3 SYNTHETIC QUESTION GENERATION

The synthetic question generator uses the augmented schema graph $\mathcal{G}$ together with the cached table descriptions to construct schema-grounded synthetic questions. When no historical workload is available, the generator first samples a sub-schema from $\mathcal{G}$ using a random walk. The walk starts from the shared source node, selects a database, and then traverses foreign-key relationships among tables within that database. The resulting sub-schema, together with the cached descriptions of its tables and a randomly selected set of its columns, is used to construct the question generation prompt to an underlying LLM. When a historical SQL workload is available, SQUAR augments the schema-driven sampler with a workload-weighted sampler. As illustrated in the right panel of Figure 2, it uses the same schema graph and random-walk procedure, but biases database and table transitions according to square-root-smoothed training frequencies. The fraction of the generation budget assigned to this sampler is determined adaptively from the recurrence of (database, table-set) structures in the training workload using the lower bound of a 95% Wilson confidence interval; the remaining budget uses the original sampler. This conservative estimate limits workload bias when historical data is sparse. Full details and experiments are provided in Appendix D.3.

The LLM generates a single natural language query that could be answered using the sampled schema. The generation prompt instructs the model to produce exactly one question to limit redundancy and generation cost, use only the supplied schema, avoid SQL-specific phrasing, avoid merely copying obvious table and column names, and remain concise. Each generated question is stored together with its sampled schema, including its source database and selected tables. The synthetic question is encoded using a dense text encoder, and its embedding is stored in a vector index alongside the associated schema metadata. The synthetic questions are used only as retrieval anchors with cosine similarity, not for training or evaluation.

The number of synthetic questions is an implementation choice that controls both schema coverage and indexing cost. Increasing the number of questions can improve coverage of possible sub-schema configurations, but also increases offline generation cost and index size. Our ablation shows that retrieval performance on Spider and BIRD saturates at relatively small stores, while BEAVER-DW benefits from a larger store (Appendix D.4). We also find that the required store size does not follow the input database schema size. In practice, we recommend evaluating a small set of candidate store sizes and selecting the smallest size that achieves near-saturated retrieval performance. In our experiments, 10K synthetic questions provide a strong shared default across the evaluated benchmarks.

4.2 SQUAR ONLINE STAGE

The offline stage builds a vector index of synthetic questions, each associated with its source database and a set of tables. The online stage uses this index to process an incoming NL query. We embed each incoming NL user query with the same dense encoder used to index the synthetic questions. We then retrieve the top-N most similar synthetic questions from the vector store; each retrieved synthetic question is associated with a source database and a set of tables. Based on the retrieved N sets, the database prediction and table ranking are produced by two *separate* frequency-based voting procedures. At the database level, each retrieved synthetic question casts one vote for its source database, and $\hat{d}_q$ is selected as the database with the highest vote count. At the table level,

each table appearing in a retrieved synthetic schema receives one vote; tables are ranked by vote count, and the top-K tables are returned as $\hat{T}_q^{(K)}$. Table voting is performed independently of the database vote. We retain this open-pool table ranking for retrieval evaluation so that database routing and table identification can be measured separately. When constructing the final schema, tables that do not belong to the selected database $\hat{d}_q$ are filtered out.

5 Experimental Evaluation

5.1 Experimental Setup

We evaluate SQuAR from several perspectives: **Q1:** How effectively does SQuAR recover the correct database and required tables compared with existing methods? **Q2:** How does the retrieval budget affect SQuAR's performance? **Q3:** How well does SQuAR scale to a unified open pool containing all four benchmarks? **Q4:** How does SQuAR's online retrieval latency compare with existing methods? Finally, **Q5:** How do SQuAR's retrieval improvements affect SQL generation?

Datasets. We evaluate on four text-to-SQL benchmarks: Spider (Yu et al., 2018), BIRD (Li et al., 2024), BEAVER-DW, and BEAVER-NW (Chen et al., 2024a). For each benchmark, we pool tables across databases after removing target-database identifiers. The benchmarks differ substantially in query complexity: BEAVER-DW and BEAVER-NW require 3.93 and 5.15 tables per query on average, compared with 1.51 and 1.93 for Spider and BIRD. This makes complete schema recovery more difficult on BEAVER and can require a larger retrieval budget. We also evaluate on a unified pool over all four benchmarks. Full statistics are provided in Appendix A.

Baselines. We compare against four schema/table retrieval methods: **DBCopilot** (Wang et al., 2025a), **IterJAR** (Boutaleb et al., 2025), **CORE-T** (Soliman et al., 2026), and **QGpT** (Liang et al., 2025). We adapt each method to our common evaluation setting and rerun the released pipelines where available. Reproduction details are provided in Appendix B.

Implementation details. SQuAR uses `Qwen2.5-14B-Instruct` for offline synthetic-question generation and `Qwen3-Embedding-4B` for dense retrieval using cosine similarity. At query time, we retrieve the top 50 anchors and return the top-K tables for $K \in \{5, 10, 15\}$. We construct 10,000 anchors for each benchmark-specific index and 40,000 anchors for the unified index (see Appendix B for more details).

Evaluation metrics. We report four metrics. DB R@1 measures correct database identification, while Table Recall@K measures required-table recall. ERC@K measures correct database identification and complete recovery of the required table set, while ERJ@K additionally penalizes unnecessary retrieved tables (Section 3.2).

5.2 Retrieval Effectiveness

Table 2 shows how SQuAR fares against its competitors across all four benchmarks at $K \in \{5, 10, 15\}$. SQuAR achieves the highest Table Recall and ERC at every retrieval budget, demonstrating its usefulness to both individual table recovery and complete schema recovery. Figure 3 further shows how ERC changes as the retrieval budget increases.

On Spider and BIRD, SQuAR achieves near-complete table coverage and high ERC. The BEAVER benchmarks remain substantially more challenging, with absolute ERC remaining considerably lower. BEAVER-NW particularly illustrates the gap between database identification and complete table recovery. SQuAR maintains DB R@1 near 0.98, while its ERC increases from 0.1705 at $K = 5$ to 0.4318 at $K = 15$, indicating that the main difficulty lies in recovering the complete table set rather than identifying the database. This behavior is consistent with the structure of BEAVER-NW, which contains only five databases but 366 tables.

The larger gold schemas in BEAVER also make retrieval more sensitive to the output budget. At $K = 5$, the gold table set exceeds the retrieval budget for 9.1% of BEAVER-DW queries and 30.7% of BEAVER-NW queries. As Figure 3 shows, increasing K substantially improves ERC on both benchmarks. However, at $K = 15$, every gold table set fits within the retrieval budget, while SQuAR reaches ERC values of only 0.6612 on BEAVER-DW and 0.4318 on BEAVER-NW.

Table 2: Retrieval effectiveness across the four benchmarks at $K \in \{5, 10, 15\}$.

		BIRD				Spider				BEAVER-DW				BEAVER-NW			
Method	K	**DB R1**	**TR**	**ERC**	**ERJ**	**DB R1**	**TR**	**ERC**	**ERJ**	**DB R1**	**TR**	**ERC**	**ERJ**	**DB R1**	**TR**	**ERC**	**ERJ**
DBCopilot	5	79.5	79.9	68.4	25.4	79.4	90.6	76.2	23.8	–	39.3	8.3	4.6	77.3	41.1	12.5	7.3
	10	79.5	87.9	69.9	14.3	79.4	93.4	76.2	14.5	–	56.5	19.8	6.2	77.3	50.1	13.6	3.9
	15	79.5	89.7	69.9	12.4	79.4	94.6	76.2	13.6	–	61.5	28.9	8.0	77.3	52.8	13.6	3.0
IterJAR	5	96.2	93.1	85.3	31.6	87.7	97.5	85.7	26.3	–	53.2	20.7	12.6	90.9	42.3	6.8	3.9
	10	90.2	97.2	87.6	16.8	70.4	97.7	69.2	10.5	–	66.4	28.9	9.1	89.8	47.7	9.1	2.7
	15	72.7	98.1	70.9	9.3	60.9	98.5	60.4	6.2	–	73.6	41.3	9.2	83.0	52.5	6.8	1.2
CORE-T	5	97.8	89.7	79.3	**54.0**	92.9	96.2	87.5	**52.0**	–	41.1	10.7	8.8	**98.9**	37.9	9.1	6.5
	10	97.3	93.1	84.8	**54.5**	92.1	97.3	88.1	**49.2**	–	53.3	16.5	11.7	**98.9**	53.3	20.5	**12.1**
	15	97.3	93.8	85.7	**54.2**	90.9	97.9	87.7	**48.6**	–	56.1	16.5	8.8	95.5	56.3	17.0	9.0
QGpT	5	95.2	92.3	82.0	29.9	85.8	98.5	85.0	26.4	–	44.7	15.7	9.8	79.5	30.6	2.3	1.4
	10	92.3	96.9	87.0	18.9	75.7	99.6	75.7	13.6	–	62.1	27.3	8.7	71.6	43.8	5.7	1.9
	15	92.2	96.9	86.9	18.8	75.6	99.7	75.6	13.6	–	70.8	34.7	7.7	69.3	54.8	8.0	1.4
SQuAR	5	**98.9**	**96.4**	**91.4**	39.7	**96.3**	**99.4**	**96.1**	40.7	–	**59.8**	**22.3**	**13.9**	97.7	**49.1**	**17.1**	**10.2**
	10	**99.0**	**99.3**	**97.5**	34.8	**96.8**	**99.9**	**96.8**	38.6	–	**79.9**	**49.6**	**17.4**	97.7	**63.7**	**28.4**	10.3
	15	**98.8**	**99.5**	**97.5**	34.4	**96.8**	**99.9**	**96.8**	38.5	–	**87.8**	**66.1**	**17.1**	**97.7**	**73.8**	**43.2**	**13.2**

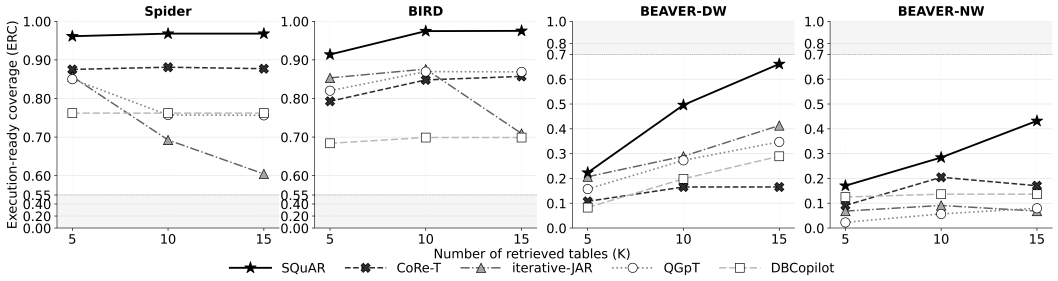


Figure 3: Execution-Ready Coverage (ERC) for different K values on different benchmarks.

Thus, the remaining gap reflects retrieval difficulty rather than output capacity alone. ERJ reveals a complementary trade-off. CORE-T achieves higher ERJ on Spider and BIRD, indicating more selective retrieval, whereas SQuAR achieves higher ERC and therefore recovers the complete required schema for more queries. On BEAVER, SQuAR achieves the strongest ERC–ERJ combination.

5.3 Scaling to a Unified Open Pool

We combine all four benchmarks into a single unified pool containing 37 databases and 619 tables. This setting increases the search space and directly evaluates the open-pool retrieval scenario motivating SQuAR. IterJAR and CORE-T were not included because both require corpus-wide compatibility preprocessing over tables and columns, as discussed in Appendix C.4.1.

Table 3 reports performance at $K \in \{5, 10, 15\}$. SQuAR achieves the highest performance across all four metrics. At $K = 5$, it reaches 0.9561 DB R@1, 0.9349 Table Recall, 0.8531 ERC, and 0.3659 ERJ. Compared with QGpT, SQuAR improves ERC by 20.1 percentage points and Table Recall by 8.7 percentage points, while increasing DB R@1 from 0.8012 to 0.9561. This shows that SQuAR maintains strong retrieval performance in the unified search space, recovering a larger fraction of relevant tables while preserving database identification and complete-schema recovery.

5.4 Online Cost

We evaluate the online efficiency of each retrieval method using per-query latency, with full results across retrieval budgets reported in Appendix C.2. Table 4 reports median and P95 latency at $K = 5$, excluding warm-up queries. SQuAR maintains 107.6–109.6 ms median latency across the four benchmarks, with P95 below 116 ms except on BEAVER-NW (184.1 ms). It is 3.5–11.2× faster than DBCopilot, 4.3–33.2× faster than IterJAR, and 66.5–92.5× faster than CORE-T. QGpT remains the fastest method (41.4–41.8 ms median) but achieves lower ERC. Overall, SQuAR combines strong

Table 3: Retrieval results in the unified pool

Method	K	DB R@1	TR	ERC	ERJ
DBCopilot	5	67.2	73.4	54.4	18.9
	10	67.2	82.6	55.7	11.5
	15	67.2	85.4	55.7	10.3
QGpT	5	80.1	84.8	65.2	22.5
	10	65.7	90.1	58.0	10.5
	15	49.2	92.2	43.2	5.6
SQuAR	5	**95.6**	**93.5**	**85.3**	**36.6**
	10	**96.0**	**97.1**	**90.8**	**32.6**
	15	**96.0**	**98.0**	**92.0**	**32.1**

Table 4: Online retrieval latency at $K = 5$ (ms).

Method	Stat.	Spider	BIRD	BEAVER-DW	BEAVER-NW
DBCopilot	Median	381.6	429.8	1163.3	1219.6
	P95	677.4	654.5	1573.4	2660.6
IterJAR	Median	751.8	469.6	3633.5	524.6
	P95	1250.0	742.7	5631.0	948.9
CORE-T	Median	7153.2	7664.7	10138.3	10018.3
	P95	9927.1	10675.1	13808.7	14072.0
QGpT	Median	**41.7**	**41.8**	**41.4**	**41.7**
	P95	**45.5**	**43.9**	**50.4**	**57.4**
SQuAR	Median	107.6	108.0	109.6	108.7
	P95	109.9	109.9	115.6	184.1

complete-schema retrieval with approximately 0.1-second median latency, providing a favorable effectiveness–efficiency trade-off.

5.5 Downstream SQL Generation

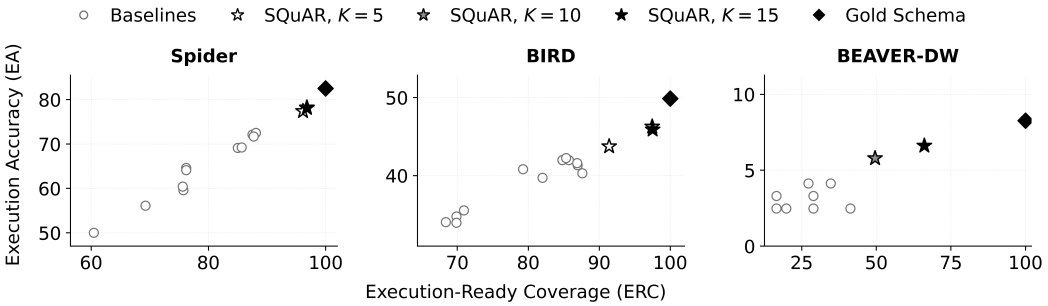


Figure 4: ERC versus downstream execution accuracy on Spider, BIRD, and BEAVER-DW.

We next evaluate how improvements in schema retrieval translate into downstream text-to-SQL generation. For each retrieval method, we use its retrieved schema as input to the same SQL generation pipeline, with GPT-5.5 as a fixed one-shot generator. We evaluate the generated SQL using execution accuracy (EA) and compare it with the ERC of the corresponding retrieved schema.

Figure 4 plots ERC against EA. The Gold Schema condition, shown at 100% ERC, provides the ground-truth schema reference for the downstream generator. Across the evaluated benchmarks, higher complete-schema retrieval is generally associated with higher execution accuracy, showing that ERC captures retrieval quality that is directly relevant to downstream SQL generation. BEAVER-NW is excluded because GPT-5.5 achieves zero correct executions on its 75 non-empty queries even when provided with the Gold Schema. We therefore report the ERC–execution-accuracy relationship on Spider, BIRD, and BEAVER-DW.

6 Conclusion

We introduced SQuAR, a join-aware schema retrieval method that uses schema-grounded synthetic questions as retrieval anchors, together with Execution-Ready Coverage (ERC) and Execution-Ready Jaccard (ERJ) for evaluating open-pool schema retrieval. By shifting semantic and structural reasoning offline, SQuAR requires no generative LLM calls at query time and maintains approximately 0.1-second median retrieval latency while achieving strong complete-schema retrieval across four benchmarks and a unified open pool. Our downstream evaluation further shows that improving complete schema recovery translates into better SQL execution accuracy. These results also highlight a limitation of standard table-level recall in open-pool settings: high recall does not guarantee correct database routing or recovery of the complete required schema. ERC directly captures this requirement, while ERJ complements it by penalizing unnecessary retrieved tables.

AI USE STATEMENT

Generative AI tools were used to assist with related-work review and paper discovery, code development and debugging, and manuscript drafting, revision, and polishing. Generative AI was also used as part of the experimental pipeline for synthetic-question generation and downstream SQL generation, as described in the paper. All AI-assisted outputs, code, experimental procedures, analyses, and claims were reviewed and verified by the authors.

REFERENCES

Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. *Proceedings of the ACM on Management of Data*, 3(3):1–28, 2025.

Allaa Boutaleb, Bernd Amann, Rafael Angarita, and Hubert Naacke. Exploring multi-table retrieval through iterative search. *arXiv preprint arXiv:2511.13418*, 2025.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, and Wei Chen. Rsl-sql: Robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*, 2024.

Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. Beaver: an enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038*, 2024a.

Peter Baile Chen, Yi Zhang, and Dan Roth. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 2687–2699, 2024b.

Tsung-Hsiang Chou, Chen-Jui Yu, Shui-Hsiang Hsu, and Yao-Chung Fan. Cgpt: Cluster-guided partial tables with llm-generated supervision for table retrieval. In *Proceedings of the ACM Web Conference 2026*, pp. 8585–8588, 2026.

Yeounoh Chung, Gaurav T Kakkar, Yu Gan, Brenton Milne, and Fatma Özcan. Is long context all you need? leveraging llm's extended context for nl2sql. *Proceedings of the VLDB Endowment*, 18(8):2735–2747, 2025.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. Reforce: A text-to-sql agent with self-refinement, consensus enforcement, and column exploration. *arXiv preprint arXiv:2502.00675*, 2025.

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

Mayank Kothyari, Dhruva Dhingra, Sunita Sarawagi, and Soumen Chakrabarti. Crush4sql: Collective retrieval using schema hallucination for text2sql. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14054–14066, 2023.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 13067–13075, 2023.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.

Hsing-Ping Liang, Che-Wei Chang, and Yao-Chung Fan. Improving table retrieval with question generation from partial tables. In Shuaichen Chang, Madelon Hulsebos, Qian Liu, Wenhu Chen, and Huan Sun (eds.), *Proceedings of the 4th Table Representation Learning Workshop*, pp. 217–228, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-268-8. doi: 10.18653/v1/2025.trl-1.19. URL https://aclanthology.org/2025.trl-1.19/.

Kyle Luoma and Arun Kumar. A comparative evaluation of schema subsetting for llm-based nl-to-sql over large-schema databases. *Proceedings of the VLDB Endowment*, 19:2019–2031, 08 2026. doi: 10.14778/3819518.3819531.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*, 2024.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, 36:36339–36348, 2023.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CvGqMD5OtX.

Hassan Soliman, Vivek Gupta, Dan Roth, and Iryna Gurevych. Core-t: Coherent retrieval of tables for text-to-sql. *arXiv preprint arXiv:2601.13111*, 2026.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

Rishit Toteja, Arindam Sarkar, and Prakash Mandayam Comar. In-context reinforcement learning with retrieval-augmented generation for text-to-SQL. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 10390–10397, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.692/.

TU. What we learned building llm-powered text-to-sql — gst workshop spring 2026. https://youtu.be/ORJlWKud9xM, 2026.

Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. Dbcopilot: Natural language querying over massive databases via schema routing. In Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (eds.), *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pp. 707–721. OpenProceedings.org, 2025a. doi: 10.48786/EDBT.2025.57. URL https://doi.org/10.48786/edbt.2025.57.

Yihan Wang, Peiyu Liu, and Xin Yang. 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*, pp. 977–991, 2025b.

Wenxuan Xie, Gaochen Wu, and Bowen Zhou. Mag-sql: Multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. *arXiv preprint arXiv:2408.07930*, 2024.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. MURRE: Multi-hop table retrieval with removal for open-domain text-to-SQL. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 5789–5806, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.386/.

Table 5: Dataset statistics for the pooled open-pool evaluation.

Dataset	Questions	Databases	Tables	Avg. gold tables
Spider dev	1,034	20	81	1.51
BIRD dev	1,534	11	75	1.93
BEAVER-DW	121	1	97	3.93
BEAVER-NW	88	5	366	5.15
Combined	**2,777**	**37**	**619**	**1.96**

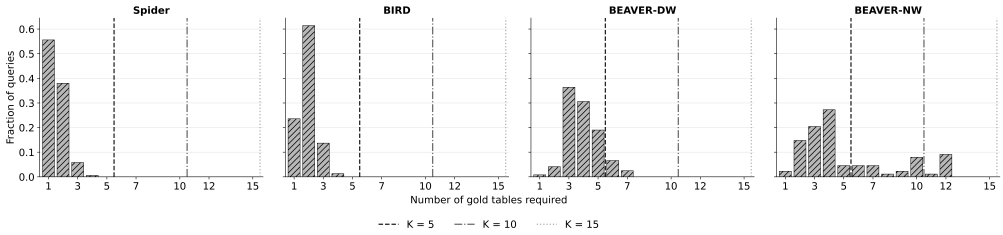


Figure 5: Distribution of the number of gold tables required per query across the four benchmarks.

A Dataset Statistics

This section summarizes the scale and gold-schema characteristics of the benchmarks used in our open-pool evaluation. Table 5 reports the benchmark sizes, while Section A.1 examines the number of tables required per query and its implications for the retrieval budget.

A.1 Gold Schema Size Distribution

Figure 5 shows the distribution of gold schema sizes across the four benchmarks. Spider and BIRD require 1.51 and 1.93 tables per query on average, respectively, compared with 3.93 for BEAVER-DW and 5.15 for BEAVER-NW.

Gold-schema size directly constrains complete retrieval under a fixed output budget: whenever $|T_q^*| > K$, complete table recovery is impossible. This constraint is particularly relevant for the BEAVER benchmarks and motivates our evaluation at $K \in \{5, 10, 15\}$.

B Implementation and Baseline Reproduction Details

This section provides implementation details for SQuAR and the reproduction and adaptation of the evaluated baselines. All methods use the same benchmark representation and evaluation workload. Unless otherwise stated, the main SQuAR experiments use workload-agnostic random-walk sampling.

B.1 SQuAR

We construct a deterministic schema graph and generate reproducible node2vec-style random walks with $p = q = 1$ and seed 42. For each benchmark, we form empirical distributions of the number of tables per example and columns per annotated table. Target cardinalities are sampled from these distributions; the walk determines table identities, while columns are sampled uniformly without replacement. Counts are clipped to schema availability. Sampled schemas are kept unique whenever possible, with duplicates allowed after 25 refill attempts to reach the requested store size.

Workload-aware sampling. When historical SQL queries are available, we additionally use a workload-weighted sampler. A training-query structure is defined by its database and unique set of referenced tables, and is recurrent if it appears at least twice. Let $\hat{p}$ be the fraction of valid training queries belonging to recurrent structures. We set the workload mixture weight λ to the lower bound of the 95% Wilson confidence interval of $\hat{p}$. For N anchors, $\mathrm{round}(N\lambda)$ use workload-aware sampling and the remainder use the standard sampler.

Workload frequencies count each database once per query and each referenced table and column at most once. The same schema graph is used, with transition weights

$$w(f) = 1 + \sqrt{f},$$

where f is the corresponding training frequency. Source-to-database edges use database frequency, database-to-table edges use table frequency, and foreign-key transitions use the destination-table frequency. Column sampling uses the same weighting, while unseen schema elements retain weight one. Table and column counts still follow the standard empirical distributions, so workload information changes element identities rather than schema size. Duplicate schemas are allowed in the workload-aware portion to preserve recurring patterns. Both portions are combined and deterministically shuffled before question generation. The subsequent question-generation, indexing, and retrieval pipeline is shared by the standard and workload-aware samplers.

For each sampled schema, we use `Qwen/Qwen2.5-14B-Instruct` to generate one realistic natural-language query answerable from the schema. The prompt also includes cached short purpose descriptions for each table; missing descriptions are generated with the same model. Generation uses batch size 128, maximum input length 8,192, maximum output length 192, temperature 0.7, and top-p 0.9. Python, NumPy, PyTorch, and Transformers are seeded for reproducibility.

Generated queries are embedded with `Qwen/Qwen3-Embedding-4B`, L2-normalized, and stored with their schemas in a local Milvus-Lite collection using cosine similarity. Embedding and insertion batch sizes are 128 and 2,048, respectively.

At inference, we embed the input query and retrieve the 50 nearest anchors. Database prediction uses one vote per anchor for its source database, while table prediction counts occurrences across all retrieved anchors and ranks tables globally by vote count. Fully qualified database–table identifiers distinguish identically named tables. For budget K, the first K ranked tables form the predicted set; the final candidate schema filters out tables outside the selected database. We report $K \in \{5, 10, 15\}$.

Online latency includes query embedding, vector search, and construction of the global table ranking, and the database vote.

B.2 Baseline Reproduction

For each baseline, we distinguish components regenerated locally from released pretrained models, caches, and auxiliary artifacts that are reused.

B.2.1 DBCopilot

We generate new synthetic routing examples for each benchmark and train the schema router locally using the released procedure with seed 123, 20 epochs, and batch size 32. The `t5-base` router uses 10 beams/groups, 10 returned sequences, and diversity penalty 2. The pretrained upstream schema-questioning/question-generation component is reused, while evaluation uses our locally trained router checkpoints.

We extend the implementation to Beaver-DW, Beaver-NW, and the unified benchmark and apply compatibility fixes for the Ray training interface, checkpoint handling, and constrained decoding. Latency uses batch size 1 with explicit per-query timing. Our reproduction covers routing-data generation, router training, and evaluation.

B.2.2 IterJAR

Dense retrieval uses `facebook/contriever-msmarco` to retrieve an initial top-20 candidate set, followed by greedy selection with coverage, join, and coarse weights of 2/1/4. We rerun Contriever scoring and greedy selection for Spider, BIRD, Beaver-DW, and Beaver-NW at each reported K.

The greedy stage reuses released compatibility artifacts for uniqueness, semantic and exact column similarity, Jaccard similarity, and table metadata. The released evaluation excludes queries whose gold schema contains one table; we remove this filtering to evaluate all methods on the same workload.

B.2.3 QGpT

We use the official QGpT implementation with our benchmark adaptation. Question generation uses the original prompt with `meta-llama/Llama-3.1-8B-Instruct`, while retrieval uses `BAAI/bge-m3` with 1,024-dimensional embeddings. Milvus-Lite uses inner-product similarity and an `IVF_FLAT` index with `nlist=256`.

For each table, we construct the representation from its schema and sampled SQLite rows and generate synthetic questions using the original prompting strategy. We regenerate representations, generated questions, embeddings, indexes, retrieval outputs, and evaluation results for Spider, BIRD, Beaver-DW, Beaver-NW, and the unified 619-table collection.

B.2.4 CORE-T

We use the official CORE-T implementation. Dense retrieval uses `WhereIsAI/UAE-Large-V1`, while table selection and augmentation use `Qwen/Qwen2.5-7B-Instruct`. We use a locally hosted Hugging Face version of the embedding model instead of the released remote interface.

For Beaver-DW and Beaver-NW, we regenerate table preprocessing, metadata, and pairwise compatibility artifacts. For Spider and BIRD, we reuse the released offline caches. Starting from these representations, we rerun dense retrieval, LLM-based table selection, and table augmentation. We additionally extend the implementation to Beaver-DW and Beaver-NW.

B.2.5 Common Evaluation Protocol

Table-level evaluation uses fully qualified database–table identifiers. For methods without a native database prediction, we infer one by majority voting over the databases of their retrieved tables, using the same rule across methods. SQuAR instead uses the anchor-level database vote described in Section 4.2. This enables consistent computation of DB R@1 and ERC while preserving native database predictions when available.

C Extended Retrieval Results

This section extends the main-paper evaluation in four directions. We first report full benchmark-wise results and latencies across retrieval budgets, then examine the divergence between table-level recall and complete, correctly routed retrieval, and finally evaluate all benchmarks jointly in a unified retrieval pool.

C.1 Full Benchmark Results

Tables 6 and 7 report the complete benchmark-wise results for $K \in \{5, 10, 15\}$. Beyond the main-paper metrics, we additionally report table precision (P) and Complete Recall (CR), which measures whether all required tables are retrieved irrespective of database prediction. Best values within each retrieval budget are shown in bold. DB R@1 is omitted for BEAVER-DW because it contains a single database.

Across retrieval budgets, SQuAR consistently achieves the strongest coverage-oriented performance, leading Table Recall and ERC on all four benchmarks. CORE-T exhibits complementary selectivity, particularly on Spider and BIRD, where it achieves substantially higher precision and ERJ. Increasing K also affects the methods differently: SQuAR rapidly saturates on Spider and BIRD while continuing to improve on the larger BEAVER schemas, whereas some baselines gain Table Recall without corresponding improvements in ERC.

C.2 Latency Across Retrieval Budgets

Figures 6 and 7 summarize the online latency and coverage–latency trade-off at $K = 10$, while Table 8 reports median and P95 latency across $K \in \{5, 10, 15\}$. Across all four benchmarks, QGpT has the lowest median latency and SQuAR the second lowest. At $K = 10$, SQuAR requires only approximately 105–109 ms per query while achieving the strongest Table Recall and ERC across the four benchmarks.

Table 6: Full baseline results on BIRD and Spider across retrieval budgets.

		BIRD						Spider					
Method	K	**DB R@1**	**P**	**R**	**CR**	**ERC**	**ERJ**	**DB R@1**	**P**	**R**	**CR**	**ERC**	**ERJ**
SQuAR	5	**0.9889**	0.4245	**0.9640**	**0.9185**	**0.9140**	0.3972	**0.9632**	0.4174	**0.9942**	**0.9942**	**0.9613**	0.4068
	10	**0.9896**	0.3537	**0.9934**	**0.9831**	**0.9746**	0.3479	**0.9681**	0.3940	**0.9990**	**0.9990**	**0.9681**	0.3858
	15	**0.9876**	0.3496	**0.9948**	**0.9863**	**0.9752**	0.3444	**0.9681**	0.3928	**0.9990**	**0.9990**	**0.9681**	0.3848
CORE-T	5	0.9785	**0.6462**	0.8968	0.7953	0.7934	**0.5409**	0.9294	**0.5728**	0.9624	0.9294	0.8752	**0.5197**
	10	0.9713	**0.6159**	0.9299	0.8553	0.8435	**0.5418**	0.9207	**0.5399**	0.9726	0.9468	0.8810	**0.4921**
	15	0.9720	**0.6089**	0.9374	0.8716	0.8566	**0.5418**	0.9091	**0.5355**	0.9792	0.9574	0.8772	**0.4861**
IterJAR	5	0.9615	0.3536	0.9306	0.8664	0.8533	0.3164	0.8772	0.2960	0.9752	0.9594	0.8569	0.2627
	10	0.9022	0.1866	0.9725	0.9478	0.8761	0.1677	0.7041	0.1482	0.9771	0.9642	0.6925	0.1051
	15	0.7269	0.1258	0.9814	0.9654	0.7093	0.0926	0.6093	0.1000	0.9847	0.9749	0.6044	0.0621
QGpT	5	0.9524	0.3485	0.9226	0.8383	0.8201	0.2991	0.8578	0.3003	0.9853	0.9758	0.8501	0.2636
	10	0.9231	0.2120	0.9685	0.9309	0.8696	0.1887	0.7573	0.1749	0.9956	0.9952	0.7573	0.1358
	15	0.9224	0.2117	0.9692	0.9316	0.8690	0.1884	0.7563	0.1747	0.9966	0.9961	0.7563	0.1356
DBCopilot	5	0.7947	0.3091	0.7994	0.7197	0.6838	0.2541	0.7940	0.3077	0.9062	0.8830	0.7621	0.2378
	10	0.7947	0.1867	0.8785	0.8070	0.6988	0.1433	0.7940	0.2040	0.9336	0.9139	0.7621	0.1454
	15	0.7947	0.1627	0.8974	0.8253	0.6988	0.1242	0.7940	0.1923	0.9465	0.9284	0.7621	0.1357

Table 7: Full baseline results on BEAVER-DW and BEAVER-NW across retrieval budgets.

		BEAVER-DW						BEAVER-NW					
Method	K	**DB R@1**	**P**	**R**	**CR**	**ERC**	**ERJ**	**DB R@1**	**P**	**R**	**CR**	**ERC**	**ERJ**
SQuAR	5	–	**0.4512**	**0.5975**	**0.2231**	**0.2231**	**0.1388**	0.9773	0.3937	**0.4912**	**0.1932**	**0.1705**	**0.1017**
	10	–	0.3118	**0.7994**	**0.4959**	**0.4959**	**0.1744**	0.9773	0.2953	**0.6372**	**0.2955**	**0.2841**	0.1028
	15	–	0.2399	**0.8780**	**0.6612**	**0.6612**	**0.1710**	**0.9773**	0.2619	**0.7380**	**0.4432**	**0.4318**	**0.1322**
CORE-T	5	–	0.4510	0.4105	0.1074	0.1074	0.0883	**0.9886**	**0.5477**	0.3793	0.0909	0.0909	0.0653
	10	–	**0.4045**	0.5328	0.1653	0.1653	0.1172	**0.9886**	**0.5263**	0.5333	0.2045	0.2045	**0.1211**
	15	–	**0.3276**	0.5615	0.1653	0.1653	0.0877	0.9545	**0.4613**	0.5628	0.1818	0.1705	0.0897
IterJAR	5	–	0.3967	0.5320	0.2066	0.2066	0.1256	0.9091	0.3674	0.4229	0.0682	0.0682	0.0386
	10	–	0.2488	0.6644	0.2893	0.2893	0.0909	0.8977	0.2304	0.4769	0.0909	0.0909	0.0273
	15	–	0.1851	0.7363	0.4132	0.4132	0.0920	0.8295	0.1711	0.5254	0.1250	0.0682	0.0121
QGpT	5	–	0.3256	0.4468	0.1570	0.1570	0.0975	0.7955	0.2822	0.3061	0.0227	0.0227	0.0136
	10	–	0.2298	0.6210	0.2727	0.2727	0.0868	0.7159	0.2078	0.4383	0.0795	0.0568	0.0193
	15	–	0.1785	0.7084	0.3471	0.3471	0.0766	0.6932	0.1717	0.5477	0.1705	0.0795	0.0144
DBCopilot	5	–	0.2975	0.3930	0.0826	0.0826	0.0463	0.7727	0.3489	0.4111	0.1364	0.1250	0.0727
	10	–	0.2283	0.5654	0.1983	0.1983	0.0617	0.7727	0.2250	0.5014	0.1705	0.1364	0.0389
	15	–	0.2160	0.6153	0.2893	0.2893	0.0803	0.7727	0.1729	0.5283	0.1932	0.1364	0.0301

SQuAR's latency remains nearly constant as the retrieval budget increases, with median latency between approximately 105 and 110 ms for all datasets and values of K. This stability follows from its fixed online computation: the timed stage performs Milvus search over 50 anchors and constructs the global database and table rankings. Consequently, changing K does not alter the timed retrieval and ranking workload, and the small observed differences reflect run-to-run timing variation.

QGpT and DBCopilot are also invariant to K by design. QGpT performs one shared retrieval to $K = 15$ and evaluates shorter ranked prefixes for $K = 5$ and $K = 10$. DBCopilot similarly performs one fixed routing generation with 10 returned sequences and evaluates prefixes of that output rather than rerunning generation for each K. IterJAR exhibits only small latency variation: its expensive query-decomposition and Contriever stages operate over a fixed 20-table candidate pool, and the compatibility matrix is computed once over those candidates. Increasing K therefore adds only a small number of greedy-selection iterations.

CORE-T exhibits a substantially different scaling pattern. From $K = 5$ to $K = 15$, its median latency increases from 7.15 to 10.27 s on Spider, 7.66 to 9.89 s on BIRD, 10.14 to 16.68 s on BEAVER-DW, and 10.02 to 15.39 s on BEAVER-NW. The dominant cost is its LLM-based table-selection stage. CORE-T performs one selection call per query, but increasing K expands the prompt with additional table headers, purposes, and pairwise compatibility information, increasing the cost of that call. The P95 measurements show the same overall scaling behavior.

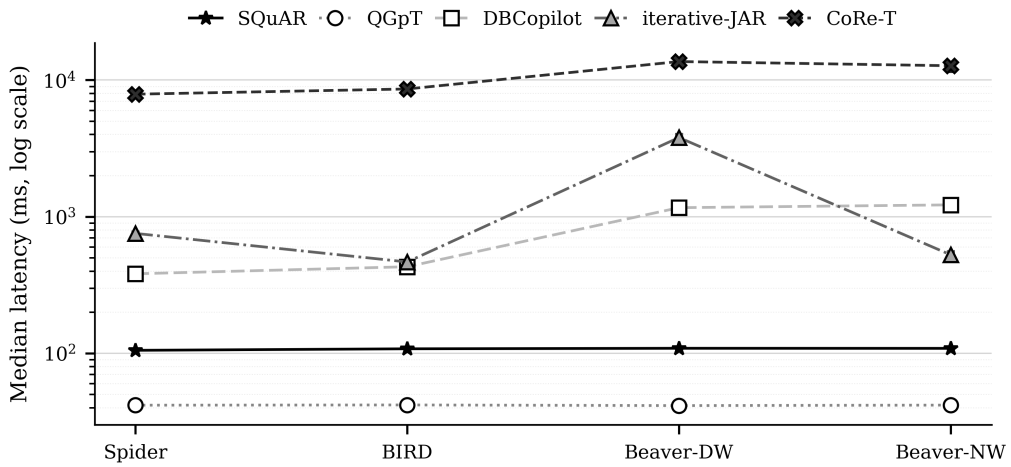


Figure 6: Median online latency across benchmarks at $K = 10$.

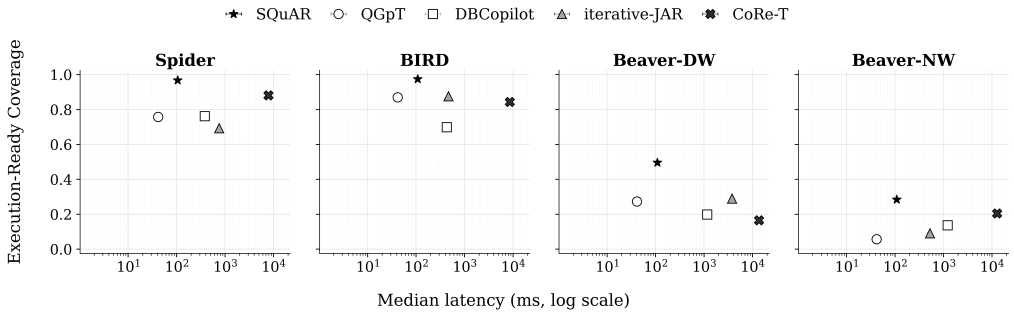


Figure 7: ERC versus median online latency at $K = 10$.

Overall, SQuAR combines the strongest coverage-oriented retrieval results with the second-lowest median latency across all four benchmarks. Its online latency also remains stable as the retrieval budget increases, unlike CORE-T, whose LLM-based selection cost grows substantially with K.

C.3 Divergence Between Table Recall and ERC

Figure 8 illustrates how increasing the retrieval budget can improve table coverage while reducing complete, correctly routed retrieval. IterJAR provides a clear example of this behavior, so we plot its Table Recall, DB R@1, and ERC across the three multi-database benchmarks.

On Spider, Table Recall increases slightly as K grows, but DB R@1 and ERC decrease substantially. This indicates that additional retrieved candidates improve gold-table coverage while also increasing cross-database interference under the common database-inference protocol. BIRD shows a related pattern: Table Recall increases monotonically, DB R@1 decreases with larger K, and ERC improves only up to $K = 10$ before dropping at $K = 15$. On BEAVER-NW, Table Recall again improves with K, whereas ERC remains low and does not improve monotonically. Together, these trends show that Table Recall alone does not fully characterize retrieval quality in the open-pool setting: complete retrieval also depends on preserving the correct database inference.

C.4 Unified Retrieval Pool

We further evaluate retrieval when all four benchmarks are combined into a single heterogeneous pool of 37 databases and 619 tables. Before presenting the results, we describe a scalability limitation that prevents two baselines from being evaluated in this setting.

Table 8: Online latency across retrieval budgets. Each entry reports median / P95 latency in milliseconds; warm-up queries are excluded.

Method	K	Spider	BIRD	BEAVER-DW	BEAVER-NW
SQuAR	5	107.6 / 109.9	108.0 / 109.9	109.6 / 115.6	108.7 / 184.1
	10	105.0 / 109.0	107.8 / 110.5	108.7 / 114.2	108.6 / 167.7
	15	105.9 / 110.6	108.6 / 110.8	108.9 / 115.9	109.1 / 172.9
QGpT	5	41.7 / 45.5	41.8 / 43.9	41.4 / 50.4	41.7 / 57.4
	10	41.7 / 45.5	41.8 / 43.9	41.4 / 50.4	41.7 / 57.4
	15	41.7 / 45.5	41.8 / 43.9	41.4 / 50.4	41.7 / 57.4
DBCopilot	5	381.6 / 677.4	429.8 / 654.5	1163.3 / 1573.4	1219.6 / 2660.6
	10	381.6 / 677.4	429.8 / 654.5	1163.3 / 1573.4	1219.6 / 2660.6
	15	381.6 / 677.4	429.8 / 654.5	1163.3 / 1573.4	1219.6 / 2660.6
IterJAR	5	751.8 / 1250.0	469.6 / 742.7	3633.5 / 5631.0	524.6 / 948.9
	10	754.0 / 1243.3	465.8 / 737.8	3793.4 / 5667.8	525.0 / 953.3
	15	755.0 / 1244.7	467.5 / 743.9	3763.6 / 5623.7	525.6 / 923.5
CORE-T	5	7153.2 / 9927.1	7664.7 / 10675.1	10138.3 / 13808.7	10018.3 / 14072.0
	10	7877.9 / 12363.0	8600.5 / 12074.9	13656.5 / 17599.7	12725.2 / 21767.8
	15	10266.0 / 15407.0	9893.8 / 13633.7	16682.2 / 22006.9	15387.6 / 27396.8

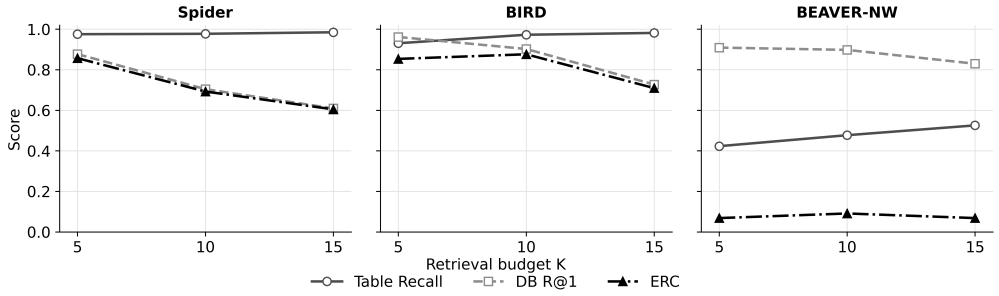


Figure 8: Table Recall, DB R@1, and ERC for IterJAR across retrieval budgets $K \in \{5, 10, 15\}$.

C.4.1 Compatibility Preprocessing at Unified Scale

CORE-T and IterJAR rely on offline compatibility computations across the schema collection. In the unified pool of 619 tables and 5,477 columns, these computations span 191,271 table pairs and approximately 15 million cross-table column pairs. At this scale, constructing the complete compatibility artifacts required by the available implementations was not computationally scalable in our experimental setting. We therefore report the unified-pool comparison for DBCopilot, QGpT, and SQuAR.

C.4.2 Full Results Across Retrieval Budgets

Table 9 reports the full unified-pool results for $K \in \{5, 10, 15\}$ over 2,777 queries, 37 databases, and 619 tables. P and R denote macro-averaged table precision and recall, respectively, and CR denotes Complete Recall.

SQuAR remains stable in database identification as the retrieval budget increases, while ERC improves from 0.8531 at $K = 5$ to 0.9204 at $K = 15$ and CR from 0.8815 to 0.9564. In contrast, QGpT's Table Recall increases from 0.8482 to 0.9223, while DB R@1 decreases from 0.8012 to 0.4923 and ERC from 0.6521 to 0.4321. This provides another example of why higher table-level recall alone does not imply complete, correctly routed retrieval.

D Additional Analyses

We next move from comparative retrieval results to analyses of SQuAR's design and behavior. We study the contribution of individual pipeline components, the importance of semantic question–schema alignment, the use of historical workload information, sensitivity to retrieval-store size, and the relationship between retrieval quality and downstream SQL generation.

Table 9: Full retrieval results on the unified pool ($N = 2{,}777$) across retrieval budgets.

Method	K	DB R@1	P	R	CR	ERC	ERJ
SQuAR	5	**0.9561**	**0.4131**	**0.9349**	**0.8815**	**0.8531**	**0.3659**
	10	**0.9597**	**0.3470**	**0.9712**	**0.9399**	**0.9078**	**0.3255**
	15	**0.9597**	**0.3366**	**0.9804**	**0.9564**	**0.9204**	**0.3209**
QGpT	5	0.8012	0.3043	0.8482	0.7526	0.6521	0.2248
	10	0.6572	0.1652	0.9009	0.8354	0.5798	0.1054
	15	0.4923	0.1169	0.9223	0.8671	0.4321	0.0556
DBCopilot	5	0.6719	0.2872	0.7337	0.6503	0.5441	0.1888
	10	0.6719	0.1933	0.8258	0.7454	0.5571	0.1149
	15	0.6719	0.1747	0.8545	0.7753	0.5574	0.1031

Table 10: Component ablation results at $K = 5$. Best value within each dataset and metric is shown in bold.

Dataset	Variant	DB R@1	Table Recall	ERC	ERJ
Spider	**Full**	0.9632	0.9942	0.9613	0.4068
	No purpose	**0.9691**	**0.9958**	**0.9671**	0.4057
	No columns	0.9091	0.9347	0.8588	**0.4235**
	Random tables	0.9642	0.9923	0.9603	0.3818
BIRD	**Full**	0.9889	**0.9640**	**0.9140**	**0.3972**
	No purpose	**0.9915**	0.9615	0.9055	0.3911
	No columns	0.9635	0.9230	0.8305	0.3722
	Random tables	0.9889	0.9513	0.8866	0.3732
BEAVER-DW	**Full**	–	**0.5975**	0.2231	0.1388
	No purpose	–	0.5923	**0.2314**	**0.1488**
	No columns	–	0.5255	0.1570	0.0942
	Random tables	–	0.4631	0.1322	0.0727
BEAVER-NW	**Full**	**0.9773**	**0.4912**	**0.1705**	**0.1017**
	No purpose	0.9432	0.4335	0.1591	0.0972
	No columns	**0.9773**	0.4760	0.1250	0.0750
	Random tables	0.9318	0.3709	0.0909	0.0386

D.1 Component Ablation

We ablate the synthetic-anchor construction pipeline to measure the effects of (i) table-purpose descriptions, (ii) column information, and (iii) relation-aware schema sampling. All variants use the same retrieval and voting procedure; only the indicated component is changed. Table 10 reports results at $K = 5$.

Purpose descriptions. Removing table-purpose descriptions has comparatively little effect on Spider, BIRD, and BEAVER-DW. On Spider, the variant without purpose text slightly improves Table Recall (0.9942 $\rightarrow$ 0.9958) and ERC (0.9613 $\rightarrow$ 0.9671), while on BEAVER-DW ERC changes only marginally (0.2231 $\rightarrow$ 0.2314). The effect is more noticeable on BEAVER-NW, where Table Recall decreases from 0.4912 to 0.4335. Overall, purpose descriptions have a modest and dataset-dependent effect, with the clearest benefit on BEAVER-NW. This suggests that such descriptions may be more useful for schemas with domain-specific or less transparent terminology.

Column information. Removing column information degrades retrieval across all four benchmarks. On Spider, Table Recall decreases from 0.9942 to 0.9347 and ERC from 0.9613 to 0.8588; on BIRD, they decrease from 0.9640 to 0.9230 and from 0.9140 to 0.8305, respectively. BEAVER-DW also shows a notable ERC drop from 0.2231 to 0.1570. These results indicate that column-level semantics provide useful evidence for recovering the complete required schema.

Relation-aware schema sampling. The effect of relation-aware sampling is substantially larger on the BEAVER benchmarks. Replacing random-walk sampling with independently sampled tables reduces Table Recall on BEAVER-DW from 0.5975 to 0.4631 and ERC from 0.2231 to 0.1322. On BEAVER-NW, Table Recall decreases from 0.4912 to 0.3709 and ERC from 0.1705 to 0.0909. Spider and BIRD are considerably less sensitive to this change, indicating that preserving table relationships becomes more important as schemas grow larger and queries require multiple related tables.

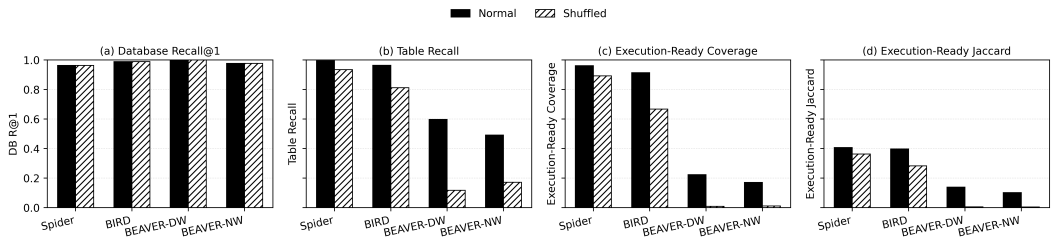


Figure 9: Effect of within-database schema reshuffling at $K = 5$.

Taken together, the ablation shows that the components play different roles: column information provides useful semantic evidence across benchmarks, whereas relation-aware sampling becomes especially important on the larger BEAVER schemas. Purpose descriptions have a smaller and more dataset-dependent effect.

D.2 Semantic Alignment

SQuAR's gains could potentially arise from coarse database priors or frequently sampled tables rather than from the semantic correspondence between a synthetic question and the schema from which it was generated. We therefore perform a controlled within-database reshuffling experiment.

For each synthetic anchor, we keep the question embedding fixed and randomly permute its schema payload among anchors from the same database. This preserves database assignments and the within-database distribution of sampled schemas and tables, while breaking the original question–schema correspondence. We then apply the same retrieval and voting procedure without any other changes. We report this analysis at the primary operating point $K = 5$, consistent with the component ablation and representing the tightest retrieval budget.

Figure 9 shows that disrupting question–schema alignment substantially degrades table retrieval while leaving DB R@1 unchanged by construction. On Spider, Table Recall decreases from 0.9942 to 0.9346 and ERC from 0.9613 to 0.8926. The degradation is larger on BIRD, where Table Recall falls from 0.9640 to 0.8126 and ERC from 0.9140 to 0.6675.

The effect is particularly pronounced on the BEAVER benchmarks. On BEAVER-DW, Table Recall falls from 0.5975 to 0.1171 and ERC from 0.2231 to 0.0083. On BEAVER-NW, Table Recall decreases from 0.4912 to 0.1720 and ERC from 0.1705 to 0.0114. Thus, after breaking the association between each anchor and its original schema, complete, correctly routed retrieval becomes rare on both BEAVER benchmarks.

Because reshuffling is restricted to anchors from the same database, it preserves database priors as well as schema and table frequencies. The observed degradation therefore cannot be explained by database routing or frequency effects alone. Instead, it shows that fine-grained semantic alignment between synthetic questions and their associated schema subsets is an important source of SQuAR's table-retrieval performance. The larger degradation on BEAVER is also consistent with semantic alignment becoming more valuable when the candidate schema contains many plausible table combinations.

D.3 Workload-Aware Anchor Generation

We evaluate whether historical SQL workloads can improve synthetic-anchor generation using the workload-aware sampler described in Section B.1. We use a separate approximately 80/20 workload-oriented train/test split for each dataset, with workload statistics computed exclusively from the training portion. We compare a 10K-anchor *Normal* store against a 10K-anchor *Adaptive Workload* store while keeping question generation, embedding, indexing, retrieval, and voting unchanged. We evaluate $K \in \{5, 10, 15\}$.

Figure 10 shows that the adaptive mixture varies substantially across datasets. Workload-aware anchors account for 96.2% of Spider and 94.2% of BIRD, compared with 25.4% for BEAVER-DW

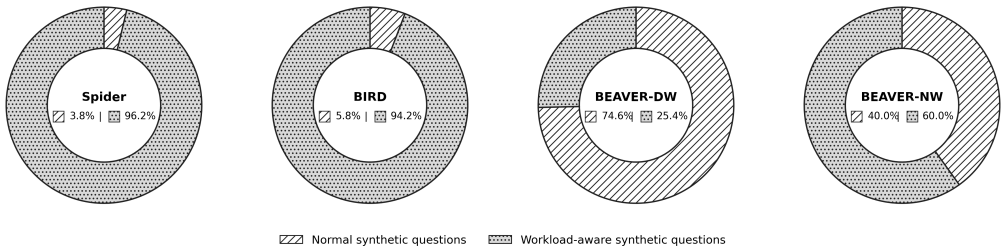


Figure 10: Adaptive composition of the 10K-anchor stores across datasets.

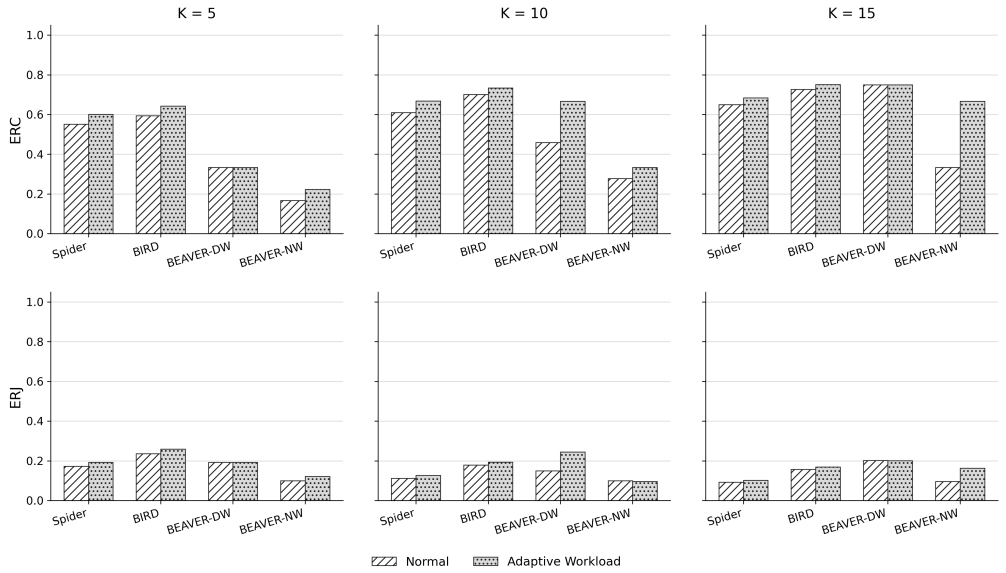


Figure 11: ERC and ERJ for Normal and Adaptive Workload stores at $K \in \{5, 10, 15\}$.

and 60.0% for BEAVER-NW. Thus, the adaptive procedure uses workload guidance heavily when recurring training structures are common, while retaining a larger schema-driven component when recurrence is weaker.

Figure 11 shows consistent ERC improvements on Spider and BIRD across all three retrieval budgets, together with smaller ERJ gains. On BEAVER-DW, the two methods are similar at $K = 5$, while workload guidance provides a substantial improvement at $K = 10$ before becoming comparable again at $K = 15$. BEAVER-NW also improves in ERC, with the largest gain at $K = 15$. Overall, the results show that historical workload information can improve complete-schema retrieval, while adaptive mixing controls how strongly the synthetic store relies on that signal.

D.4 Sensitivity to Retrieval Store Size

We study how retrieval performance changes with the number of synthetic anchors available to the retriever. We vary the store size from 1,000 to 100,000 anchors per benchmark while keeping the full configuration—purpose descriptions, relation-aware random-walk sampling, and column information—fixed. Figure 12 reports Table Recall and ERC at $K = 5$ separately for each benchmark.

The largest gains occur when increasing the store from 1,000 to approximately 10,000 anchors. On Spider, Table Recall increases from 0.9414 at 1K anchors to 0.9942 at 10K, while on BIRD it increases from 0.9252 to 0.9640. Both benchmarks subsequently remain close to saturation, indicating that relatively small synthetic stores already provide broad coverage of their schemas.

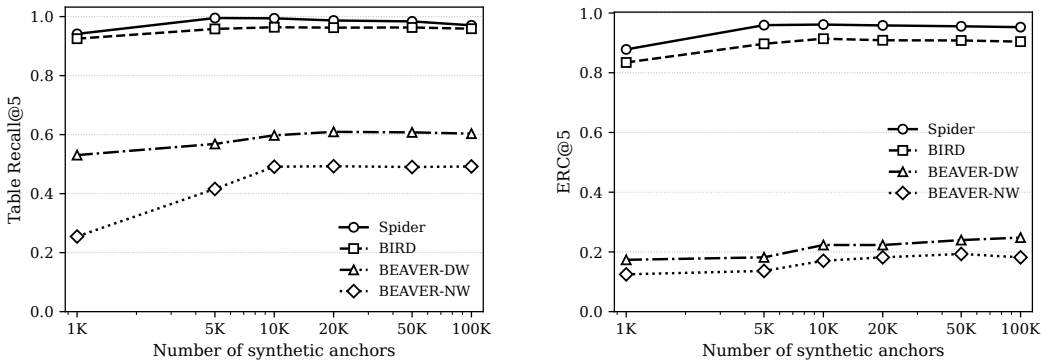


Figure 12: Effect of retrieval-store size at $K = 5$. Left: Table Recall. Right: Execution-Ready Coverage (ERC).

The effect is substantially larger on the enterprise benchmarks. BEAVER-DW improves from 0.5306 Table Recall at 1K anchors to 0.5975 at 10K, while BEAVER-NW increases from 0.2547 to 0.4912 over the same range, with ERC showing the same general pattern. Beyond approximately 10K–20K anchors, however, the gains become small and non-monotonic. For example, BEAVER-NW Table Recall remains close to 0.49 from 10K through 100K anchors, while Spider and BIRD show little additional improvement after reaching their earlier saturation points. Thus, simply increasing the retrieval store does not continuously improve retrieval quality.

These results motivate our use of 10,000 anchors per benchmark in the main experiments. This operating point captures most of the improvement obtained from increasing the synthetic store while avoiding the additional offline generation, embedding, and storage cost of substantially larger indices.

D.5 Retrieval Metrics and Downstream SQL Generation

We further analyze how the proposed retrieval metrics relate to downstream SQL-generation quality. Using the same fixed one-shot SQL generator as in the main-paper downstream experiment, we compare Execution-Ready Coverage (ERC) and Execution-Ready Jaccard (ERJ) against execution accuracy (EA) across retrieval methods and budgets. Table 11 reports the underlying values, and Figure 13 visualizes the two relationships.

For Spider and BIRD, downstream generation is evaluated at $K \in \{5, 10, 15\}$. For BEAVER-DW, we report downstream results only at $K \in \{10, 15\}$. At $K = 5$, 9.1% of BEAVER-DW queries require more than five gold tables, so complete schema recovery is impossible for those queries under the retrieval budget. We therefore omit $K = 5$ from the downstream comparison to avoid conflating retrieval quality with this hard output-capacity constraint. BEAVER-NW is excluded because the downstream generator achieves zero correct executions on its non-empty evaluation queries even when provided with the Gold Schema.

ERC and downstream execution accuracy. Figure 13 shows a consistent relationship between complete schema recovery and downstream SQL-generation performance. On Spider, SQuAR achieves approximately 96–97% ERC across retrieval budgets and 77–78% EA, placing it closer to the Gold Schema condition than the evaluated baselines. On BIRD, increasing SQuAR's ERC from 91.40% at $K = 5$ to approximately 97.5% at $K = 10$ and $K = 15$ is accompanied by an increase in EA from 43.74% to approximately 46%. The same broad pattern is visible across the baselines: configurations with substantially lower ERC generally also exhibit lower downstream execution accuracy.

BEAVER-DW remains substantially more difficult in absolute terms. Nevertheless, among the evaluated retrieved-schema conditions, SQuAR achieves the highest ERC at both $K = 10$ and $K = 15$, reaching 49.59% and 66.12%, respectively, and also obtains the highest corresponding EA, increasing from 5.79% to 6.61%. These results support ERC as a retrieval metric that captures whether the downstream generator receives the complete schema required to express the target SQL.

Table 11: Retrieval quality and downstream execution accuracy. Best retrieval-method values are bold.

		Spider			BIRD			BEAVER-DW		
Method	**K**	**ERC**	**ERJ**	**EA**	**ERC**	**ERJ**	**EA**	**ERC**	**ERJ**	**EA**
SQuAR	5	**96.1**	40.7	**77.4**	**91.4**	39.7	**43.7**	**22.3**	**13.9**	–
	10	**96.8**	38.6	**78.2**	**97.5**	34.8	**46.3**	**49.6**	**17.4**	**5.8**
	15	**96.8**	38.5	**78.1**	**97.5**	34.4	**45.9**	**66.1**	**17.1**	**6.6**
CORE-T	5	87.5	**52.0**	72.1	79.3	**54.0**	40.8	10.7	8.8	–
	10	88.1	**49.2**	72.5	84.8	**54.5**	42.0	16.5	11.7	4.1
	15	87.7	**48.6**	71.7	85.7	**54.2**	42.0	16.5	8.8	3.3
QGpT	5	85.0	26.4	69.1	82.0	29.9	39.7	15.7	9.8	–
	10	75.7	13.6	59.6	87.0	18.9	41.3	27.3	8.7	2.5
	15	75.6	13.6	60.4	86.9	18.8	41.6	34.7	7.7	3.3
DBCopilot	5	76.2	23.8	64.6	68.4	25.4	34.0	8.3	4.6	–
	10	76.2	14.5	64.6	69.9	14.3	34.8	19.8	6.2	2.5
	15	76.2	13.6	64.1	69.9	12.4	34.0	28.9	8.0	5.0
IterJAR	5	85.7	26.3	69.2	85.3	31.6	42.2	20.7	12.6	–
	10	69.3	10.5	56.1	87.6	16.8	40.3	28.9	9.1	4.1
	15	60.4	6.2	50.0	70.9	9.3	35.5	41.3	9.2	5.0
Gold Schema	–	100.0	100.0	82.5	100.0	100.0	49.9	100.0	100.0	8.3

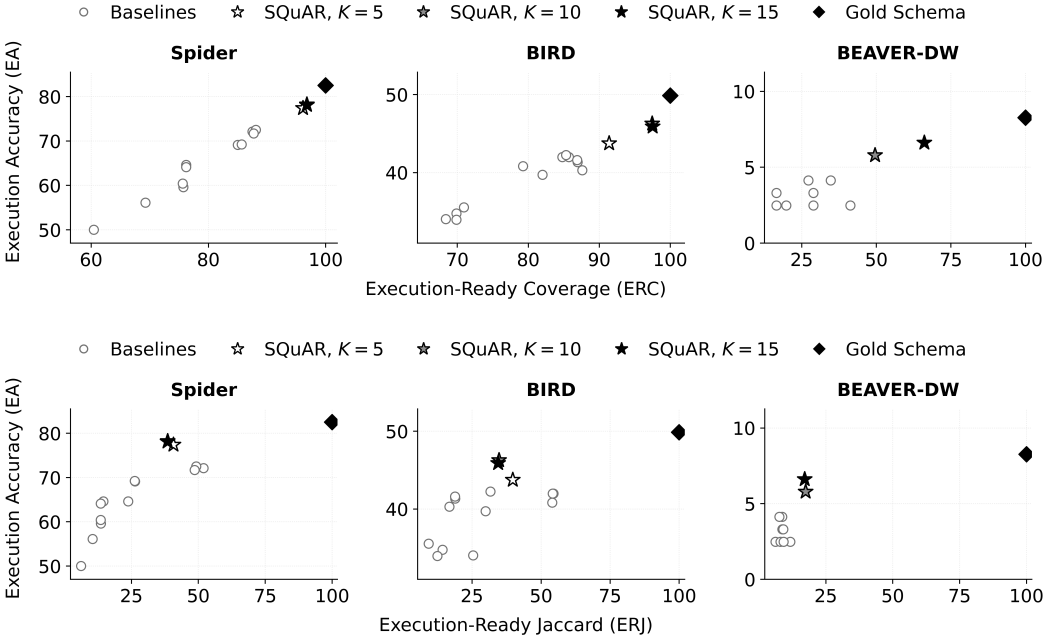


Figure 13: Relationship between retrieval quality and downstream execution accuracy (EA). *Top:* Execution-Ready Coverage (ERC) versus EA. *Bottom:* Execution-Ready Jaccard (ERJ) versus EA. Baseline points correspond to the evaluated retrieval methods and budgets; SQuAR results are shown separately for each K, and the Gold Schema condition provides the reference obtained when the generator receives the ground-truth schema. For BEAVER-DW, downstream results are shown only for $K \in \{10, 15\}$.

ERJ and downstream execution accuracy. ERJ provides a complementary view because it additionally penalizes unnecessary retrieved tables. However, its relationship with downstream EA is less direct than that of ERC. In particular, CORE-T achieves higher ERJ than SQuAR on Spider and BIRD, reflecting its more selective retrieved schemas, while SQuAR achieves higher downstream execution accuracy. For example, at $K = 5$ on BIRD, CORE-T reaches an ERJ of 54.02% compared with 39.72% for SQuAR, yet their corresponding EA values are 40.81% and 43.74%, respectively.

This distinction reflects the different properties captured by the two metrics. ERC measures whether the correct database and every required table are available to the downstream generator, whereas

ERJ additionally rewards excluding unnecessary tables. The downstream results indicate that complete required-table recovery is particularly important: a more selective retrieved schema does not compensate for missing a required table. ERJ remains useful for characterizing over-retrieval among complete-schema outputs, while ERC shows a clearer relationship with whether the retrieved schema supports successful downstream SQL generation.