

UNJOIN: Enhancing Multi-Table Text-to-SQL Generation via Schema Simplification

Poojah Ganesan^{1*} Rajat Aayush Jha^{1*} Dan Roth² Vivek Gupta^{1†}

¹Arizona State University ²University of Pennsylvania ²Oracle
{pganesa4, rjha16, vgupt140}@asu.edu danroth@seas.upenn.edu

Abstract

Recent advances in large language models (LLMs) have greatly improved Text-to-SQL performance for single-table queries. But, it remains challenging in multi-table databases due to complex schema and relational operations. Existing methods often struggle with retrieving the right tables and columns, generating accurate JOINS and UNIONS, and generalizing across diverse schemas. To address these issues, we introduce UNJOIN, a two-stage framework that decouples the retrieval of schema elements from SQL logic generation. In the first stage, we merge the column names of all tables in the database into a single-table representation by prefixing each column with its table name. This allows the model to focus purely on accurate retrieval without being distracted by the need to write complex SQL logic. In the second stage, the SQL query is generated on this simplified schema and mapped back to the original schema by reconstructing JOINS, UNIONS, and relational logic. Evaluations on SPIDER and BIRD datasets show that UNJOIN matches or exceeds the state-of-the-art baselines. UNJOIN uses only schema information, which does not require data access or fine-tuning, making it scalable and adaptable across databases. Our code is available at: <https://github.com/coral-lab-asu/unjoin>

1 Introduction

Relational databases form the foundation for structured data management in domains such as finance, healthcare, and education. Accessing information from these databases typically requires writing SQL queries, a skill that demands technical expertise. Text-to-SQL, known as semantic parsing, addresses this barrier by translating natural language questions into executable SQL commands (Androustopoulos et al., 1995; Li and Jagadish, 2014; Li et al., 2023b), allowing non-experts to interact with complex databases.

*These authors contributed equally to this work.
†Primary supervisor of this work.

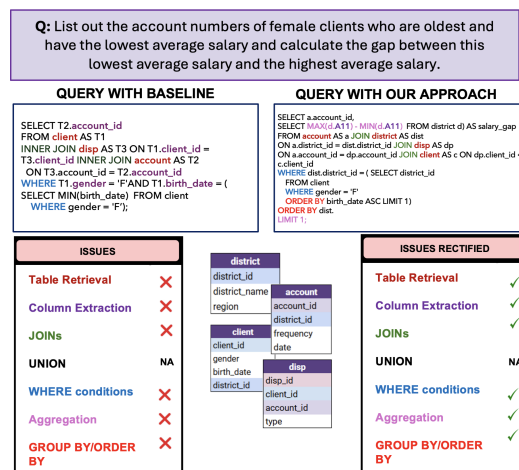


Figure 1: Baseline vs UNJOIN

Early approaches use syntax trees or query sketches to guide query generation (Xu et al., 2017; Guo et al., 2019), while more recent methods rely on sequence-to-sequence models (Raffel et al., 2020a; Scholak et al., 2021). Recent advances leverage large language models (LLMs), either through in-context learning with powerful proprietary models or by fine-tuning smaller open-source alternatives, leading to substantial performance gains (Li et al., 2024; Pourreza and Rafiei, 2024).

While recent advances in Text-to-SQL have significantly improved performance on single-table databases, relatively little attention has been given to the more challenging multi-table setting. As shown in Figure 1, multi-table SQL generation introduces additional complexities such as identifying relevant tables and columns, resolving inter-table relationships (e.g., JOINS and UNIONS), and constructing more intricate queries involving GROUP BY, aggregation, ordering, and nested subqueries. These challenges are central to building robust and generalizable Text-to-SQL systems for real-world applications. This raises a natural question: *How can the progress made in single-table SQL generation be extended to handle the challenges of multi-table querying?*

To answer this, we propose UNJOIN, a modular two-stage framework that decouples retrieval of schema elements from complex SQL generation. The key intuition is that LLMs are highly effective at generating SQL for single-table schemas, a setting they are more exposed to during training, while multi-table scenarios introduce structural complexity that is harder to handle directly. UNJOIN bridges this gap by reframing multi-table Text-to-SQL generation as a simplified single-table task that LLMs can solve more reliably. UNJOIN operates in two stages:

(a.) **Stage 1: Schema Simplification:** The multi-table schema is flattened into a single-table format by merging the column names of all tables in the database into a single-table representation by prefixing each column with its table name, without altering the underlying data.

(b.) **Stage 2: Query Generation and Translation.** A SQL query is first generated over this simplified schema and then translated back to align with the original schema by reconstructing necessary JOINS, UNIONS, and column relationships. By isolating the challenges of schema element selection and SQL logic construction, UNJOIN disambiguates the reasoning process and enables more accurate, scalable, and generalizable multi-table SQL generation. Our contributions are as follows:

1. We propose UNJOIN, a novel approach for multi-table Text-to-SQL generation based on schema simplification. By decoupling table and column retrieval from complex SQL structuring, UNJOIN reduces compounding errors and improves robustness.
2. We demonstrate that our method outperforms a wide range of baselines, including (a) standard prompting, (b) in-context learning methods, (c) supervised fine-tuning approaches, (d) end-to-end table Question Answering (QA) models, and (e) recent reasoning-focused models such as Deepseek-R1 on both open book and closed book settings.
3. Through detailed analysis, we show that SQL generated by UNJOIN achieves superior performance in both table retrieval and column selection. Additionally, when combined with various retrievers in open-domain table QA settings, UNJOIN consistently outperforms standard few-shot prompting baselines.

2 Methodology

Existing LLM-based approaches struggle with identifying relevant tables and columns, resolving inter-table relationships like JOINS and UNIONS, and constructing more complex queries involving GROUP BY, nested subqueries, aggregation, and ordering. This is due to schema complexity, ambiguous column references, and limited context size. Overcoming these challenges is crucial for developing robust and generalizable Text-to-SQL systems suited for real-world applications. In Figure 2, we present our proposed framework, UNJOIN, which addresses these issues systematically through *Schema Simplification* and *Query Generation and Translation*. In the following section, a detailed introduction of these steps are presented.

(1.) Schema Simplification To simplify complex multi-table schemas, we introduce a straightforward *Schema Simplification* step. Here, we combine columns from all tables in the database into one simplified schema, without altering the underlying data. The goal is to create a single, flat table. We prefix each column name with its corresponding table name, which helps in removing ambiguity between similar column names from different table.

Let’s consider a database with six tables, each containing ten columns. The resulting single-table representation will comprise one table, named after the database, and a total of 60 columns. Here, each column name is reformatted using the structure *TableName.ColumnName*, preserving its original context while eliminating ambiguity. The algorithm for *Schema Simplification* along with an example is shown in Appendix 4

In the simplified format, the structure no longer depends on how the original tables were connected, so there is no concept of table joins at this stage. By removing the need to reason about table-to-table relationships, this step eliminates a layer of complexity for the LLM. The transformation relies only on schema-level information, specifically, table and column names, and does not require access to row-level data. As a result, it generalizes well to databases of any size or content. This step is also fully deterministic and does not depend on LLMs, helping to reduce computational overhead and avoid hallucination errors that can occur when using LLMs.

(2.) Query Generation and Translation. This involves two sub-steps: (a.) Query Generation.

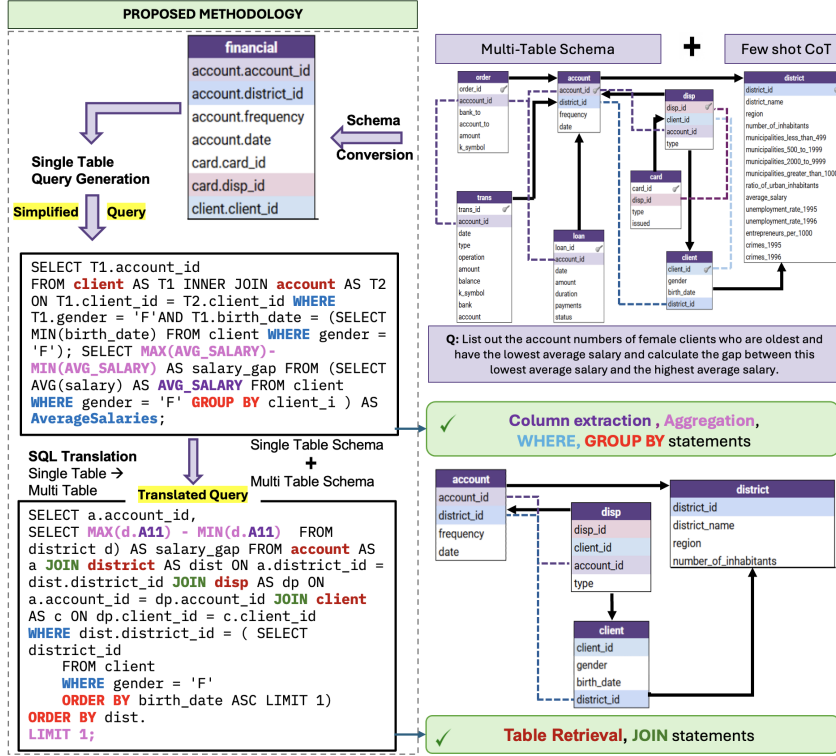


Figure 2: **Our Proposed Method.** After schema conversion, the resulting single-table schema contains a total of 54 columns obtained by combining the columns from all shown tables: *order*, *account*, *district*, *disp*, *card*, *client*, *loan*, and *trans*. Due to space constraints, the complete single-table representation is not shown.

and (b.) *Query Translation.*, which are described in detail below.

(a.) *Query Generation.* In this step, we use the *Simplified Schema* obtained from the previous step to prompt the LLM to generate a semantically accurate intermediate (simplified) SQL query. By abstracting away relational complexities, such as JOIN and UNION operations, the LLM is free to focus solely on identifying relevant tables (which are represented as columns in this case of simplified schema), as well as accurately handling SQL operations like aggregation and numeric computations. This reduces the task to a simpler single-table Text-to-SQL scenario, a setting in which contemporary LLMs typically perform very well.

To ensure robust and unbiased query generation, the LLM is provided with a carefully structured prompt that includes the *Simplified Schema*, the user’s question, detailed column descriptions and several schema-agnostic few-shot examples. These manually crafted examples remain consistent across evaluations to prevent data leakage or schema-specific biases. Importantly, as our approach never modifies the underlying table data, the resulting intermediate SQL is not directly executable; instead, its purpose is to clearly *capture user intent within a simplified schema context*. The

prompts are given in Appendix A

(b.) *Query Translation.* The intermediate SQL query generated in the previous step references a simplified single-table schema and is therefore not directly executable. The *Query Translation* step transforms this intermediate SQL into a fully executable SQL code, explicitly reintroducing necessary relational operations such as JOINS, pertaining to the original multi-table schema.

This process is fully automated, requiring only the original schema, the simplified schema, the simplified SQL generated in the previous step, and the user question as input. Since this transformation is implicit within the LLM’s reasoning capabilities, no additional rule-based logic is required. To further reduce hallucination errors, we apply an edit-distance-based correction mechanism to ensure that the generated SQL aligns with the actual schema. This post-processing step only adjusts table and column names, ensuring that any abbreviations or modifications introduced by the LLM (e.g., table name *disp* changed to *disposition*) are accurately mapped back to their valid schema counterparts. In particular, this step does not alter SQL logic or introduce external constraints; it only enhances schema consistency.

UNJOIN Variants: We explore two variations of our proposed method: (a) UNJOIN_{SP}: Performs schema simplification, query generation, and translation sequentially within a single (joint) prompt approach. (b) UNJOIN_{MP}: Separates these steps into distinct stages using a multi-prompt setup, allowing for more focused reasoning at each stage.

3 Text2SQL Approaches

We compare UNJOIN with five approaches, as follows:

Standard Prompts: We evaluate against prompting strategies like Direct Prompting with Few-Shot Chain of Thought (CoT) (Wei et al., 2022), Program of Thoughts (PoT) (Chen et al., 2023), Meta Prompting (MP) (Suzgun and Kalai, 2024), and Self-Consistency (SC) (Wang et al., 2023).

In-Context Learning (ICL): We compare UNJOIN against several recent in-context learning baselines. DIN-SQL (Pourreza and Rafiei, 2023) decomposes the task into subtasks, prompting GPT-4 separately for each. C3-SQL (Dong et al., 2023) introduces schema filtering followed by calibrated prompting with self-consistency. RSL-SQL (Cao et al., 2024) improves schema linking through bidirectional reasoning, contextual augmentation, and multi-turn self-correction. Several other recent in-context learning methods (Sheng et al., 2025; Pourreza et al., 2025; Lee et al., 2025) are not publicly available.

Supervised Fine-Tuning (SFT): Here, we compare against CodeS-7B (Li et al., 2024) and DTS-SQL (Pourreza and Rafiei, 2024).

End2End Table QA (TQA): We also evaluate UNJOIN on End-to-End QA tasks. DATER (Ye et al., 2023), TabSQLify (Nahid and Rafiei, 2024), and ReActTable (Zhang et al., 2024b) decompose the table by generating SQL, typically using sequence-to-sequence architectures or fine-tuned LLMs, and then perform QA. We also compare against non-SQL based QA models such as MultiTabQA (Pal et al., 2023) (Seq2Seq model), ResdSQL (Li et al., 2023a) and two variants of QFMTS (Zhang et al., 2024a): QFMTS_1 (summarization followed by QA), and QFMTS_2 (single-table schema summarization followed by QA).

Reasoning LLMs: This category consists of models optimized for complex reasoning tasks, including DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025), DeepSeek-R1-Distill-Llama-70B (DeepSeek-AI, 2025).

4 Experimental Evaluation.

Datasets: We evaluate our approach on two widely used large-scale cross-domain Text-to-SQL datasets: Spider (Yu et al., 2018) and BIRD (Li et al., 2023b). Spider includes 200 databases, while BIRD contains 96, both spanning a diverse range of domains. In each dataset, tables are grouped by topic, with each topic corresponding to a separate database that includes an average of 5.4 tables, along with queries answerable using that schema. Since our focus is on queries that require reasoning over multiple tables, we exclude those that involve only a single table, following a filtering strategy similar to (Chen et al., 2024). After filtering, we obtain 443 queries across 81 databases for Spider, and 1095 queries across 77 databases for BIRD.

Metrics: A generated SQL query may be structurally valid and executable, yet still return an incorrect answer. To evaluate both correctness and execution, we use two key metrics: (a.) *Query Execution Accuracy (QE)* measures the percentage of generated queries that execute successfully without runtime errors. It checks only whether the query runs, not whether the result is correct, (b.) *Exact Match (EM)* captures the percentage of generated queries that not only execute successfully but also return the correct answer. Since EM requires both successful execution and correct output, it is a stricter metric and always a subset of QE. A query may be valid and count toward QE, yet yield an incorrect result, leading to a lower EM.

LLM’s: We evaluate the performance of Prompting based and Table-QA baselines (SQL-based) across three language models: GPT-4o (OpenAI, 2024) (*gpt-4o-2024-08-06*), Gemini 1.5 Flash (Gemini Team et al., 2024), and Llama 3.3 (70B)¹. Table-QA baselines (non SQL-based), ICL and SFT baselines are evaluated using GPT-4o. For further analysis about table and column retrieval and on variations of our approach baseline, we expand our evaluation to include additional models: GPT-4o Mini² (*gpt-4o-mini-2024-07-18*), Llama 3.1 (3B), SQLCoder (34B), Mixtral 7x8B (Jiang et al., 2024) and CodeLlama (Rozière et al., 2023). This broader evaluation provides insight into the impact of model size and architecture on schema-aware SQL generation.

¹ <https://github.com/meta-llama/llama-models>

² <https://openai.com/index/gpt-4o-system-card/>

Evaluation Settings. We evaluate our approach under two distinct settings: (a) Closed-book setting — The relevant database is provided in advance. Our method, UNJOIN, directly generates the final SQL query over the given multi-table schema, and (b) Open-book setting — Relevant tables must first be retrieved using state-of-the-art retrievers. Then, UNJOIN is applied to generate the final SQL query over the retrieved tables.

In the closed-book setting, the full database schema is available, enabling SQL generation without retrieval. In contrast, the open-book setting requires a retrieval step, where tables are selected based on both table-query similarity and table-table (i.e., joinability) similarity. This setting is more challenging because (i) the retrieval process may miss relevant tables (i.e., recall is less than 100%), and (ii) the retrieved tables may come from different databases, leading to invalid or spurious joins.

4.1 Results: Closed Book Settings

Standard Prompts vs UNJOIN: As shown in Table 1, prompt-based baselines achieve high QE (e.g., CoT with 98.87 QE on SPIDER) but fall short in EM (75.28 EM), reflecting errors in table and column selection. In contrast, UNJOIN (UNJOIN_{MP}) maintains a strong QE (99.9) while significantly improving EM (77.13), demonstrating better schema understanding. A similar trend appears on BIRD (See Appendix 10), where UNJOIN_{MP} surpasses the best baseline in EM (50.36 vs. 44.38).

| Method | GPT-4o | | Gemini | | LLAMA 70B | |
|-----------------------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | QE | EM | QE | EM | QE | EM |
| <i>Standard Prompts</i> | | | | | | |
| CoT | 93.70 | 63.10 | 95.25 | 65.23 | 98.87 | 75.28 |
| PoT | 93.77 | 67.39 | 94.40 | 72.89 | 96.71 | 68.61 |
| MP | 94.14 | 67.76 | 92.91 | 67.39 | 90.88 | 63.86 |
| SC | 94.14 | 67.76 | 94.48 | 72.89 | 96.35 | 67.21 |
| <i>Table QA (SQL-based)</i> | | | | | | |
| DATER | 19.30 | 07.80 | 20.10 | 08.90 | 17.77 | 06.57 |
| TabSQLify | 93.38 | 64.10 | 90.07 | 69.96 | 93.80 | 68.61 |
| ReActTable | 02.00 | 00.20 | 03.00 | 00.20 | 02.80 | 00.30 |
| UNJOIN _{SP} | 96.35 | 76.13 | 94.89 | 73.36 | 94.49 | 69.62 |
| UNJOIN _{MP} | 94.89 | 76.00 | 95.99 | 75.57 | 99.90 | 77.13 |

Table 1: QE and EM scores on SPIDER dataset.

ICL vs UNJOIN : The results in Table 2 show that RSL-SQL achieves the highest overall performance on both SPIDER and BIRD, particularly with strong generalization to BIRD (QE: 90.8, EM: 54.3). However, UNJOIN performs competitively, on SPIDER, where UNJOIN_{SP} achieves an EM of 76.13, slightly surpassing RSL-SQL. On BIRD, both variants of UNJOIN outperform DIN-SQL and C3, showing robust cross-domain performance.

| ICL Text-to-SQL | SPIDER | | BIRD | |
|-------------------------------|--------------|--------------|--------------|--------------|
| | QE | EM | QE | EM |
| DIN-SQL + GPT-4o | 95.97 | 75.56 | 87.53 | 49.32 |
| C3 + GPT-4o | 94.50 | 71.42 | - | - |
| RSL-SQL | 96.40 | 76.04 | 90.80 | 54.30 |
| UNJOIN _{SP} (GPT-4o) | 96.35 | 76.13 | 88.55 | 51.74 |
| UNJOIN _{MP} (GPT-4o) | 94.89 | 76.00 | 89.75 | 50.36 |

Table 2: ICL models vs UNJOIN.

End-to-end Table QA vs UNJOIN: From Tables 1 and 3 it can be seen that UNJOIN outperforms all table QA baselines in both QE and EM by a substantial margin. This proves its efficiency and usefulness in end-to-end table QA tasks, apart from Text-to-SQL. It also scales efficiently on larger databases like BIRD, where other baselines struggle (see Appendix 10). MultiTabQA’s near-zero EM (0.03) suggests possible overfitting to single-table queries in SPIDER. We did not evaluate multi-table baselines on BIRD due to its large dataset size exceeding LLM input limits and the high cost of fine-tuning these models on larger datasets. As highlighted earlier, fine-tuned models often suffer from limited generalizability and become overly domain-specific, increasing costs while reducing their applicability to diverse scenarios. For these reasons, we restricted our experimentation to the SPIDER dataset.

| Table QA (non SQL-based) baselines | EM |
|------------------------------------|--------------|
| MultiTabQA (Seq2Seq) | 00.03 |
| QFMTS_1 (GPT-4o) | 25.28 |
| QFMTS_2 (GPT-4o) | 25.55 |
| ResdSQL | 28.87 |
| UNJOIN _{SP} (GPT-4o) | 76.13 |
| UNJOIN _{MP} (GPT-4o) | 76.00 |

Table 3: Performance comparison on SPIDER dataset.

SFT vs UNJOIN : Across both datasets, UNJOIN demonstrates strong performance (Table 4), outperforming existing SFT baselines on SPIDER, and showing notable generalization on BIRD, highlighting the effectiveness of our schema simplification and decomposition strategy in improving multi-table Text-to-SQL generation.

| SFT Text-to-SQL | SPIDER | | BIRD | |
|--------------------------------|--------------|--------------|--------------|--------------|
| | QE | EM | QE | EM |
| CodeS-7B | 94.14 | 74.72 | 89.65 | 51.14 |
| DTS-SQL _{DeepSeek 7B} | 87.50 | 69.23 | - | - |
| UNJOIN _{SP} (GPT-4o) | 96.35 | 76.13 | 88.55 | 51.74 |
| UNJOIN _{MP} (GPT-4o) | 94.89 | 76.00 | 89.75 | 50.36 |

Table 4: SFT models vs UNJOIN.

Reasoning LLM’s vs UNJOIN: The results in Table 5 highlight the strong performance of UNJOIN compared to reasoning model baselines in both QE and EM, demonstrating the versatility of our approach across different datasets.

| Reasoning LM | SPIDER | | BIRD | |
|-------------------------------|--------------|--------------|--------------|--------------|
| | QE | EM | QE | EM |
| DeepSeek Qwen 7B | 91.21 | 57.82 | 66.25 | 25.01 |
| DeepSeek Llama 70B | 95.26 | 61.08 | 84.12 | 44.54 |
| UNJOIN _{SP} (GPT-4o) | 96.35 | 76.13 | 88.55 | 51.74 |
| UNJOIN _{MP} (GPT-4o) | 94.89 | 76.00 | 89.75 | 50.36 |

Table 5: Reasoning models vs UNJOIN.

4.2 Results: Open Book Settings

Table 6 presents the impact of UNJOIN on the EM accuracy of various retriever-based models. Across all retrievers—ranging from Contriever³ and DTR (Herzig et al., 2021) to their enhanced counterparts in the JAR (Chen et al., 2024) framework—UNJOIN consistently yields substantial improvements in SQL generation performance. Notably, the EM scores increase by 14.90% to 19.57%, demonstrating the robustness and generalizability of our approach.

| Retriever | EM(%): CoT → UNJOIN (Δ) |
|------------------|----------------------------------|
| ARM | 31.7 → 51.27 (+19.57) |
| Contriever | 29.7 → 47.99 (+18.29) |
| DTR | 30.4 → 49.85 (+19.45) |
| JAR (DTR) | 36.9 → 52.63 (+15.73) |
| JAR (Contriever) | 36.2 → 51.10 (+14.90) |

Table 6: Exact Match (EM) scores with UNJOIN in open book settings. Δ in parentheses shows the improvement.

End2End Table Extraction. Table 7 presents a detailed comparison of multiple retriever models evaluated on precision (P) and recall (R) metrics, with the integration of our proposed UNJOIN framework for End2End table extraction settings. The results indicate a consistent and substantial improvement in retrieval performance across all retrievers when augmented with UNJOIN.

Specifically, both the DTR and Contriever retrievers exhibit significant performance gains, improving their precision by approximately 25% and 27%, and their recall by around 22% and 26% respectively. Even retrievers with relatively strong baseline performance, such as JAR (DTR) and JAR (Contriever), benefit from UNJOIN, achieving gains in precision and recall ranging from 4.6% to 6.7%.

These results highlight the core strength of our UNJOIN method—its ability to consistently enhance SQL generation across a range of retrievers. By simplifying schemas and structuring query decomposition, UNJOIN acts as a plug-in module that improves both base retrievers (e.g., Contriever, DTR) and advanced systems like JAR and ARM (Chen et al., 2025). This demonstrates its broad applicability and effectiveness in handling multi-table Text-to-SQL challenges.

³ <https://huggingface.co/facebook/contriever-msmarco>

5 Discussion

In this section, we further present ablation studies and additional insights from our UNJOIN approach for the closed book settings.

How does UNJOIN benefit End2End table extraction? We analyze End2End table extraction performance where we assess how UNJOIN approach enhances table retrieval and column extraction, along with its impact on query accuracy.

Table 8 presents the precision and recall scores for table and column selection across different methods: standard CoT prompting, two variants of our approach—UNJOIN_{SP} and UNJOIN_{MP}. Additionally, we introduce a new baseline, CoT-SS (CoT on Simplified Schema), which directly generates the final multi-table SQL (including JOINS, etc.) from the intermediate simplified schema using few-shot chain-of-thought prompting. This comparison highlights the effectiveness of decoupling schema reasoning from SQL generation. Furthermore, we observe that UNJOIN_{SP} and UNJOIN_{MP} consistently outperform other baselines (CoT, CoT-SS) in end-to-end table and column extraction.

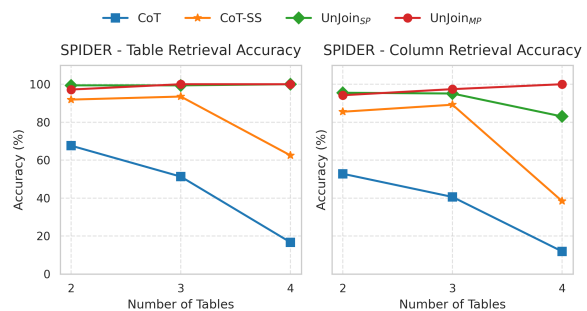


Figure 3: Retrieval accuracy Performance with Increasing Number of Tables

How robust is UNJOIN with increasing relevant tables? We analyze how table and column retrieval accuracy is affected as the number of relevant tables needed to answer the query increases. As shown in Figure 3, baseline methods exhibit a sharp decline in performance as query complexity grows, whereas UNJOIN sustains high retrieval accuracy. This demonstrates the scalability and robustness of our approach in handling complex multi-table queries. Baselines often fail on multi-operation queries as the number of tables increases, while UNJOIN’s separate translation phase handles these systematically.

How does schema simplification benefit End2End table extraction? From table 9, we see that CoT-SS suffers significant drops in QE and EM, but from table 8, we can see that it

| Retriever | ARM | | JAR (DTR) | | JAR (Contriever) | | DTR | | Contriever | |
|-----------|---------------|---------------|---------------|---------------|------------------|---------------|---------------|---------------|---------------|---------------|
| | P | R | P | R | P | R | P | R | P | R |
| CoT | 32.88 | 74.47 | 65.60 | 66.10 | 64.70 | 65.40 | 59.50 | 59.20 | 55.90 | 55.50 |
| UNJOIN | 84.30 | 83.20 | 72.31 | 70.79 | 70.84 | 70.00 | 84.55 | 81.75 | 82.97 | 81.88 |
| Δ | +51.42 | +08.73 | +06.71 | +04.69 | +06.14 | +04.60 | +25.05 | +22.55 | +27.07 | +26.38 |

Table 7: Precision (P) and Recall (R) for all retriever configurations with UNJOIN for open book settings. Δ indicates the absolute improvement.

| Model | CoT | CoT-SS | UNJOIN _{SP} | UNJOIN _{MP} | CoT | CoT-SS | UNJOIN _{SP} | UNJOIN _{MP} |
|------------------|----------------------|--------------|----------------------|----------------------|-------------------|--------------|----------------------|----------------------|
| | Table Precision (%) | | | | Table Recall (%) | | | |
| GPT-4o | 59.30 | 60.90 | 84.10 | 83.00 | 58.90 | 61.50 | 86.70 | 84.10 |
| GPT-4o Mini | 64.00 | 80.00 | 96.50 | 96.60 | 63.00 | 80.00 | 97.60 | 98.20 |
| Gemini 1.5 Flash | 55.10 | 76.90 | 72.90 | 78.10 | 54.80 | 74.40 | 75.90 | 78.60 |
| Mixtral | 45.90 | 53.50 | 71.70 | 71.00 | 40.50 | 53.30 | 78.30 | 74.70 |
| SQLCoder-7B | 52.90 | 64.00 | 48.40 | 65.00 | 53.60 | 69.30 | 53.00 | 70.20 |
| SQLCoder-34B | 54.80 | 67.50 | 68.10 | 58.10 | 55.40 | 67.10 | 73.90 | 62.00 |
| Llama 3.1 (3B) | 55.40 | 65.90 | 52.90 | 49.70 | 56.00 | 65.80 | 62.00 | 48.30 |
| Llama 3.3 (70B) | 40.80 | 74.90 | 77.10 | 72.70 | 41.20 | 74.00 | 79.10 | 75.50 |
| | Column Precision (%) | | | | Column Recall (%) | | | |
| GPT-4o | 50.70 | 67.40 | 88.10 | 88.60 | 53.70 | 71.10 | 95.50 | 95.30 |
| GPT-4o Mini | 46.70 | 71.00 | 87.70 | 87.70 | 49.80 | 75.20 | 94.60 | 95.40 |
| Gemini 1.5 Flash | 53.90 | 90.40 | 88.60 | 89.10 | 55.60 | 86.10 | 96.20 | 93.20 |
| Mixtral | 40.00 | 65.80 | 80.10 | 87.40 | 37.40 | 57.00 | 89.50 | 93.00 |
| SQLCoder-7B | 50.00 | 63.30 | 43.60 | 75.00 | 54.10 | 70.90 | 54.00 | 83.50 |
| SQLCoder-34B | 45.20 | 70.70 | 81.20 | 75.40 | 48.80 | 60.00 | 93.20 | 77.00 |
| Llama 3.1 (3B) | 50.60 | 58.90 | 43.40 | 52.00 | 54.70 | 58.90 | 66.70 | 54.30 |
| Llama 3.3 (70B) | 38.90 | 84.50 | 86.70 | 87.50 | 42.60 | 89.60 | 96.00 | 96.60 |

Table 8: Evaluation of Table/Column Precision and Recall (in %) on SPIDER dataset. For BIRD Dataset results, please refer to Appendix D.

| Dataset | Model | Query Execution (QE) | | | Exact Match (EM) | | |
|---------|-----------------|----------------------|----------------------|----------------------|------------------|----------------------|----------------------|
| | | CoT-SS. | UNJOIN _{SP} | UNJOIN _{MP} | CoT-SS. | UNJOIN _{SP} | UNJOIN _{MP} |
| Spider | GPT-4o | 14.59 | 96.35 | 94.89 | 11.68 | 76.13 | 76.00 |
| | Gemini 1.5 | 81.39 | 94.89 | 95.99 | 59.23 | 73.36 | 75.57 |
| | Llama 3.3 (70B) | 91.91 | 94.49 | 99.90 | 58.05 | 69.62 | 77.13 |
| | SQLCoder (34B) | 05.12 | 83.75 | 51.74 | 04.03 | 53.05 | 25.36 |
| | SQLCoder (7B) | 16.48 | 26.00 | 58.80 | 11.83 | 06.67 | 35.46 |
| BIRD | GPT-4o | 16.86 | 88.55 | 89.75 | 11.24 | 51.74 | 50.36 |
| | Gemini 1.5 | 67.40 | 78.51 | 81.73 | 38.58 | 44.73 | 48.40 |
| | Llama 3.3 (70B) | 82.61 | 91.97 | 96.28 | 49.40 | 52.10 | 56.35 |
| | SQLCoder (34B) | 11.24 | 64.75 | 31.75 | 08.03 | 28.55 | 19.70 |
| | SQLCoder (7B) | 08.53 | 20.01 | 35.34 | 04.70 | 06.00 | 20.70 |

Table 9: QE and EM on SPIDER and BIRD datasets.

shows strong table/column retrieval performance. This shows that schema simplification helps in retrieving correct table and column names, improving precision and recall. For instance, moving from CoT to CoT-SS often increases recall (e.g., Llama 70B table recall on SPIDER goes from 0.41 to 0.74), showing that a simplified schema aids in finding relevant tables/columns.

Why does CoT-SS fails across all models? One other interesting thing to note is that the performance of CoT-SS on EM is generally poor across all the models. This highlights the importance of query translation step in our approach. Since CoT-SS directly generates the final SQL from the intermediate Simplified Schema, skipping the query translation step, it doesn't focus on integrating the

correct compositional SQL operations, and thus, it omits necessary JOIN paths or introduce invalid syntax. This shows that query translation is essential in ensuring correct table linking and reducing structural inconsistencies.

How does our approach perform across different model sizes? From Table 9, we observe that our approach UNJOIN_{MP} performs reasonably well with different model sizes. But our approach is most effective with larger LLMs due to their superior instruction-following capability. One of the exceptions to the above trend is SQLCoder (34B), which, despite its size, does not perform well with UNJOIN. One reason can be that unlike general-purpose LLMs, SQLCoder is highly specialized for SQL generation and lacks strong natural language

understanding and instruction-following capabilities, outside generating SQL. To gain a deeper understanding of UNJOIN’s impact across different SQLCoder model sizes and its sensitivity to End2End extraction, please refer to Appendix E.

Where do UNJOIN fails? Detailed inspections of failure cases reveal three major error sources: **(a.) Misaligned Column Names:** Without the deterministic schema mapping, LLMs hallucinate or rename columns, leading to partial matches. **(b.) Ambiguity in Natural Language Queries:** Queries with unclear phrasing or multiple possible meanings often result in incorrect SQL generation. For instance, in the query "What are the names and release years for all the songs of the youngest singer?", the presence of similar column names (Song Name and Name) in the database creates ambiguity, leading to errors in column selection. **(c.) Unconventional Query Phrasing:** Users frequently use informal or unconventional phrasing in queries, making it challenging for the model to accurately interpret their intent. For example, in the query "For each singer name, what is the total sales for their songs?" the ordering condition needs adjustment to conform to proper SQL syntax.

6 Comparison with Related Work

Seq2Seq methods. Early Text-to-SQL systems like Seq2SQL (Zhong et al., 2017) and SQLNet (Xu et al., 2017) used Seq2Seq architectures but struggled with multi-table queries due to limited schema understanding. Schema-aware methods like IRNet (Guo et al., 2019), RAT-SQL (Wang et al., 2020), and RESDSQL (Li et al., 2023a) improved performance by modeling schema relationships more explicitly. However, these methods often rely on manual schema linking and struggle with generalizing to unseen databases.

PLMs based approaches. Advancements in pre-trained transformer models (PLMs) like T5 (Raffel et al., 2020b) and BART (Lewis et al., 2020) significantly boosted Text-to-SQL accuracy. Transformer-based models such as PICARD-T5 (Scholak et al., 2021), TABSQLify (Nahid and Rafiei, 2024), and BASE-SQL (Sheng et al., 2025) incorporated symbolic execution and constrained decoding to enhance multi-table reasoning. Still, their dependence on fine-tuning and pipeline complexity limits adaptability and scalability.

LLMs prompts. Prompt-based LLM methods like C3-SQL (Dong et al., 2023), DAIL-SQL (Gao et al., 2024), and MCS-SQL (Lee et al., 2025) by-

pass fine-tuning and use stepwise reasoning and prompt ensembling to improve performance. Yet, these models can be brittle, prompt-sensitive, and prone to producing invalid or incomplete SQL. Other systems tackle schema complexity via query decomposition and retrieval (e.g., DATER (Ye et al., 2023), ReActTable (Zhang et al., 2024b), CHASE-SQL (Pourreza et al., 2025)), or modular pipelines like DIN-SQL (Pourreza and Rafiei, 2023) and MAC-SQL (Wang et al., 2024) that assign subtasks to dedicated agents. These approaches improve robustness, but add orchestration complexity, and are hard to generalize.

SFT End2End methods. SFT-based methods (Li et al., 2024; Yang et al., 2024; Pourreza and Rafiei, 2024; Talaei et al., 2024; Gorti et al., 2025; Sheng et al., 2025) fine-tune open-source LLMs to improve SQL generation. Hybrid approaches like CHESS (Talaei et al., 2024), XiYan-SQL (Gao et al., 2025) and MSc-SQL (Gorti et al., 2025) combined prompt-based strategies with targeted fine-tuning, often leveraging smaller open-source models. Comprehensive surveys provide an overview of this evolving landscape (Qin et al., 2022; Katsogiannis-Meimarakis and Koutrika, 2023; Shi et al., 2024).

Despite these advancements, challenges remain in generalization, scalability, and correctness for complex multi-table queries. Our framework, UNJOIN, addresses these gaps via schema simplification and decoupled query translation, achieving strong performance on Spider and BIRD.

7 Conclusion and Future Work

In this work, we propose UNJOIN, a two-step framework for multi-table Text-to-SQL that simplifies schemas for improved retrieval, generates an intermediate SQL query, and maps it back to the original schema. This modular approach reduces structural and retrieval errors, outperforming most baselines, including prompting-based, ICL, SFT techniques, and reasoning models like Distilled-DeepSeek Llama 70B. When layered over strong retrievers like ARM, JAR, Contriever, and DTR, UNJOIN significantly boosts end-to-end retrieval and SQL generation accuracy. By processing only schema information, UNJOIN ensures scalability and generalizability without pre-training or fine-tuning. Future work includes extending UNJOIN to handle hierarchical, semi-structured, and deeply nested schemas for broader real-world applicability.

Limitations

Our approach consistently outperforms existing baselines on BIRD and SPIDER while remaining scalable across various schemas and large databases. However, it has certain limitations. The effectiveness of UNJOIN depends on the LLM’s ability to accurately follow instructions, which may be less reliable with smaller or less capable models. Moreover, UNJOIN currently focuses on structural correctness rather than cell-level content extraction (e.g., resolving ambiguous entity names like “M. Obama” vs. “Michelle Obama”). As a result, it is less suited for tasks requiring precise entity disambiguation or record-level analysis. UNJOIN also does not work on multimodal/unstructured/semi-structured tables.

Ethics Statement

We affirm that our work upholds the highest ethical standards in research and publication. Ethical considerations have been carefully addressed to ensure the responsible use of computational linguistics methodologies. To support reproducibility, we provide detailed information, including publicly available code, datasets, and relevant resources, all compliant with their respective ethical guidelines. Our claims are backed by experimental results, acknowledging minor variations due to the stochastic nature of black-box LLMs, which we mitigate by using a fixed temperature. Additionally, we outline dataset splits, model configurations, and prompting strategies to ensure transparency and reproducibility. AI was utilized in both experimentation and paper writing to enhance analysis, streamline result interpretation, and improve overall presentation clarity. Automated tools assisted in structuring content, refining language, and ensuring coherence, making the findings effectively communicated.

Acknowledgments

We gratefully acknowledge the Complex Data Analysis and Reasoning Lab (CoRAL) at Arizona State University and the Cognitive Computation Group at the University of Pennsylvania for their resources and computational support. We also thank the anonymous reviewers for their constructive feedback that helped improve this paper.

References

- Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. [Natural language interfaces to databases - an introduction](#). *CoRR*, cmp-lg/9503016.
- Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust schema linking in text-to-SQL generation. *arXiv preprint arXiv:2411.00073*.
- Peter Baile Chen, Yi Zhang, Michael Cafarella, and Dan Roth. 2025. Can we retrieve everything all at once? ARM: An alignment-oriented LLM-based retrieval method. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.
- Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. 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*. Association for Computational Linguistics.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. In *Transactions on Machine Learning Research*.
- DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-SQL with ChatGPT. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.
- 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 preprint arXiv:2411.08599*.
- Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Bunnell, Libin Bai, Anmol Gulati, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.
- Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2025. MSc-SQL: Multi-sample critiquing small language models for text-to-SQL translation. *arXiv preprint arXiv:2410.12916*.

- Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. [Towards complex text-to-SQL in cross-domain database with intermediate representation](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.
- Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Eisenschlos. 2021. [Open domain question answering over tables via dense retrieval](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 512–519, Online. Association for Computational Linguistics.
- Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L elio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. *Mixtral of experts*. *arXiv preprint arXiv:2401.04088*.
- George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, 32(4):905–936.
- Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. [MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353, Abu Dhabi, UAE. Association for Computational Linguistics.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics.
- Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.
- Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.
- Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards building open-source language models for text-to-SQL. In *Proceedings of the ACM on Management of Data*, volume 2, pages 1–28. ACM.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi 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. 2023b. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems*, volume 36.
- Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tab-SQLify: Enhancing reasoning capabilities of LLMs through table decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 3426–3440. Association for Computational Linguistics.
- OpenAI. 2024. GPT-4o system card. *arXiv preprint arXiv:2410.21276*.
- Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. [Multitabqqa: Generating tabular answers for multi-table question answering](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6322–6334, Toronto, Canada. Association for Computational Linguistics.
- Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2025. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *Proceedings of the 13th International Conference on Learning Representations*.
- Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, volume 36.
- Mohammadreza Pourreza and Davood Rafiei. 2024. [DTS-SQL: Decomposed text-to-SQL with small large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.
- Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A survey on text-to-SQL parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020a. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020b. Exploring the limits of transfer learning with a unified text-to-text

- transformer. *Journal of Machine Learning Research*, 21(140):1–67.
- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.
- Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. **PICARD: Parsing incrementally for constrained auto-regressive decoding from language models**. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Lei Sheng, Shuai-Shuai Xu, and Wei Xie. 2025. BASE-SQL: A powerful open source text-to-SQL baseline approach. *arXiv preprint arXiv:2502.10739*.
- Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2024. A survey on employing large language models for text-to-SQL tasks. *arXiv preprint arXiv:2407.15186*.
- Mirac Suzgun and Adam Tauman Kalai. 2024. Meta-prompting: Enhancing language models with task-agnostic scaffolding. In *Proceedings of the 12th International Conference on Learning Representations*.
- Shayan Taleai, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHES: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578. Association for Computational Linguistics.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 11th International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.
- Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQL-Net: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.
- Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics.
- Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 174–184.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. **Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task**. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.
- Weijia Zhang, Vaishali Pal, Jia-Hong Huang, Evangelos Kanoulas, and Maarten de Rijke. 2024a. **QFMTS: Generating query-focused summaries over multi-table inputs**. In *Proceedings of the 27th European Conference on Artificial Intelligence (ECAI-2024)*, Stockholm, Sweden. IOS Press.
- Yunjia Zhang, Jordan Henkel, Avriella Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024b. **Reactable: Enhancing react for table question answering**. *Proceedings of the VLDB Endowment*, 17(8):1981–1994.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Appendix: LLM Prompts and Examples

A.1 Prompt for UNJOIN SP

Listing 1: SQL Query Generation Prompt for UNJOIN SP

You are an expert at semantic parsing. You have to follow two steps in order to complete your task.

Step 1: Getting the simplified query
You will be given a simplified schema which has only one table and multiple columns and a question. Please return the sql query with the correct format and syntax pertaining to that question. Remember to focus on getting the correct column extraction, and where clauses.
DO NOT do any join operations. Treat this as a single table. The resulting query is the simplified query.

Example Table: bank_data

| Column Name | Description |
|----------------------|----------------------------------------------|
| customer.customer_id | Unique identifier for each customer |
| customer.name | Name of the customer |
| customer.gender | Gender of the customer |
| account.account_id | Unique identifier for accounts |
| account.balance | Current balance of the account |
| loan.loan_id | Unique identifier for loans |
| loan.amount | Loan amount |
| loan.status | Status of the loan (e.g., Approved/Rejected) |

Question 1:
"Which customers have an account balance greater than 10,000?"
SQL Query:
SELECT customer.customer_id, customer.name, account.balance
FROM bank_data
WHERE account.balance > 10000;

Question 2:
"List all customers with an approved loan."
SQL Query:
SELECT customer.customer_id, customer.name, loan.loan_id, loan.amount
FROM bank_data
WHERE loan.status = 'Approved';

Question 3:
"How many male customers have a loan?"
SQL Query:
SELECT COUNT(*) **AS** male_customers_with_loans
FROM bank_data

```
WHERE customer.gender = 'M' AND loan.loan_id IS NOT NULL;
```

Step 2: Getting the final query
Once you get the simplified query, you will have the following:

- A question: A natural language description of the desired query result.
- A simplified schema: A virtual table with columns in the format table_name.column_name, combining data from multiple original tables.
- A simplified query: A SQL query written against the simplified schema.
- An original schema: A set of multiple related tables where the actual data resides.

Your task: Translate the simplified query into a query compatible with the original schema.

Ensure the translated query aligns with the intent described in the question

Follow these steps for translation:

1. Understand the Core Objective from the Question:
 - Identify the goal of the query (e.g., aggregate data, filter specific rows, join information across tables).
2. Map Simplified Schema Columns to the Original Schema:
 - Identify how the columns in the simplified schema correspond to tables and columns in the original schema.
3. Construct Necessary Joins:
 - If the original schema splits data across multiple tables, determine the joins needed to recreate the relationships.
4. Translate Filters and Conditions:
 - Map **WHERE** clauses, conditions, and filters in the simplified query to the original schema.
5. Adapt Query Logic (Aggregation, Sorting, etc.):
 - Match aggregations, grouping, or ordering logic from the simplified query to the original schema.
6. Validate the Final Query Against the Question:
 - Review the final query to ensure it satisfies the question and produces the intended result.

< examples >

Key Considerations:

- Use the Question as a Guide:
Align the query logic with the intent expressed in the question.
- Simplified Schema as a Mapping Tool:
Treat the simplified schema as a bridge between the question and the original schema.
- Validation:
Ensure the translated query runs against the original schema and produces the intended result.

A.2 Prompt for UNJOIN_{MP} : Step1 : Query Generation on Simplified Schema

Listing 2: SQL Query Generation Prompt: UNJOIN_{MP} Step 1

You are an expert at semantic parsing. You will be given a schema which has only one table and multiple columns and a question. Please return the SQL query with the correct format and syntax pertaining to that question. Remember to focus on getting the correct column extraction and **WHERE** clauses. DO NOT perform any join operations. Treat this as a single table.

<examples>

Instructions:

1. The query should strictly adhere to the schema provided.
2. Ensure correct SQL syntax with **SELECT**, **FROM**, **WHERE**, **GROUP BY**, and **ORDER BY** clauses as needed.
3. The output query must be structured, readable, and executable in a standard SQL database.

Output:

A.3 Prompt for UNJOIN_{MP} : Step2 : Query Translation

Listing 3: SQL Query Generation Prompt: UNJOIN_{MP} Step2

You are an expert at semantic parsing. You will be provided:

- A question: A natural language description of the desired query result.
- A simplified schema: A virtual table with columns in the format `table_name.column_name`, combining data from multiple original tables.
- A simplified query: A SQL query written against the simplified schema.

- An original schema: A set of multiple related tables where the actual data resides.

Your task:

1. Generate a SQL query based on a simplified schema.
2. Translate the simplified query into a query compatible with the original schema.
3. Ensure the translated query aligns with the intent described in the question.

Steps for Translation:

1. ****Understand the Core Objective from the Question****:
 - Identify the goal of the query (e.g. ., aggregate data, filter specific rows, join information across tables).
2. ****Map Simplified Schema Columns to the Original Schema****:
 - Identify how the columns in the simplified schema correspond to tables and columns in the original schema.
3. ****Construct Necessary Joins****:
 - If the original schema splits data across multiple tables, determine the joins needed to recreate the relationships.
4. ****Translate Filters and Conditions****:
 - Map **WHERE** clauses, conditions, and filters in the simplified query to the original schema.
5. ****Adapt Query Logic (Aggregation, Sorting, etc.)****:
 - Match aggregations, grouping, or ordering logic from the simplified query to the original schema.
6. ****Validate the Final Query Against the Question****:
 - Review the final query to ensure it satisfies the question and produces the intended result.

<examples>

Key Considerations:

- ****Use the Question as a Guide****:
 - Align the query logic with the intent expressed in the question.
 - The question may highlight details (e.g., time ranges, specific groups) that must be included in the query.
- ****Simplified Schema as a Mapping Tool****:

```

- Treat the simplified schema as a
  bridge between the question and
  the original schema.
- Focus on accurately mapping the
  simplified columns to the original
  schema.

- Validation:
- Ensure the translated query runs
  against the original schema and
  produces the intended result.

Output:

```

B Schema Simplification Algorithm and Example

Algorithm 1 Schema Simplification

```

1: Input: Set of databases  $D$ 
2: Output: Simplified schema dictionary  $S$ 
3:  $S \leftarrow \{\}$   $\triangleright$  Initialize empty dictionary
4: for all database  $d$  in  $D$  do
5:    $S[d.name] \leftarrow []$   $\triangleright$  Initialize list for each
     database
6:   for all table  $t$  in  $d.tables$  do
7:     for all column name  $c$  in  $t.columns$  do
8:        $s \leftarrow t.name + '.' + c$ 
9:       Append  $s$  to  $S[d.name]$ 
10:    end for
11:  end for
12: end for
13: return  $S$ 

```

C Appendix: Baseline Results

| BIRD Dataset | | | | | | |
|----------------------|---------------------------------------|--------------|--------------|--------------|--------------|--------------|
| Method | GPT-4o | | Gemini | | LLAMA 70B | |
| | QE | EM | QE | EM | QE | EM |
| | <i>Standard Prompting Baselines</i> | | | | | |
| CoT | 87.95 | 44.38 | 73.09 | 42.40 | 91.27 | 53.55 |
| PoT | 72.49 | 43.49 | 71.76 | 46.67 | 83.09 | 51.47 |
| MP | 73.19 | 44.54 | 71.50 | 44.78 | 82.89 | 51.92 |
| SC | 73.79 | 42.86 | 70.95 | 45.19 | 86.22 | 52.06 |
| | <i>Table QA (SQL-based) baselines</i> | | | | | |
| TabSQLify | 66.83 | 42.41 | 70.32 | 44.12 | 77.78 | 49.94 |
| | UNJOIN | | | | | |
| UNJOIN _{SP} | 88.55 | 51.74 | 78.51 | 44.73 | 91.97 | 52.10 |
| UNJOIN _{MP} | 89.75 | 50.36 | 81.73 | 48.40 | 96.28 | 56.35 |

Table 10: QE and EM scores on BIRD dataset.

D Evaluation Metrics across different models

E Further Discussion

Reason: SQLCoder 34B and 7B Underperformance. SQLCoder 34B’s best table recall in UNJOIN_{MP} (60.3 on BIRD) is lower than the best performing version of smaller general models like Llama 3.1 (62.0 on BIRD) (see Table 11) despite its larger size. This suggests that SQLCoder struggles not just with instruction following, but also with schema reasoning, since it is primarily trained on SQL generation rather than structured multi-table reasoning.

End2End Tables Extraction Sensitivity. Across models, the gap between the best and worst-performing baselines is more pronounced for column precision and recall than for table precision/recall. For instance, in SPIDER, GPT-4o’s column recall (95.5 in UNJOIN_{SP}) is significantly higher than SQLCoder-7B (54.1 in UNJOIN_{SP}), whereas the table recall gap is smaller. Similarly, we can observe this behavior within the same model with different sizes. For example, on the SPIDER dataset in the UNJOIN_{SP} baseline, Llama 3.3 (70B) achieves a table recall of 74.0, whereas Llama 3.1 (3B) scores 65.9, showing a difference of only 8.1. However, the column recall for Llama 70B is 89.6, while Llama 3B achieves only 62, resulting in a much larger gap of 27.6. This trend indicates that smaller models struggle significantly more with precise column selection than table selection, likely due to their weaker contextual understanding and reasoning capabilities.

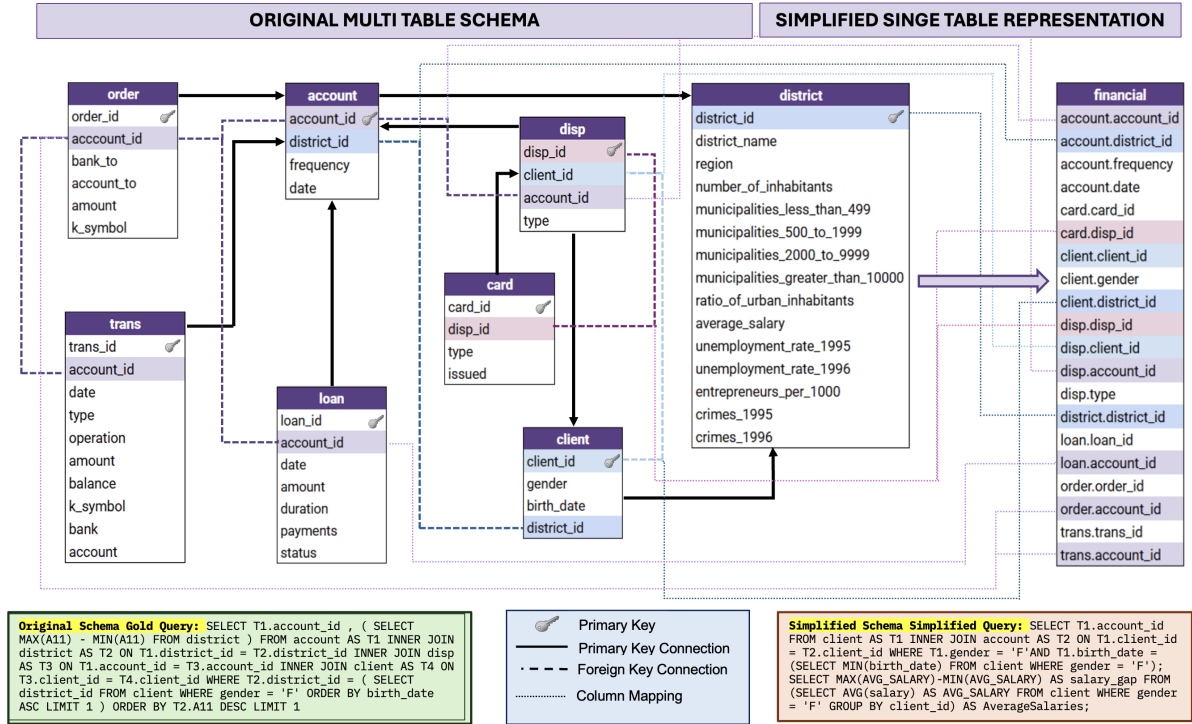


Figure 4: Schema Simplification

| Model | Table Precision | | | | Table Recall | | | |
|------------------|------------------|--------------|----------------------|----------------------|---------------|--------------|----------------------|----------------------|
| | CoT | CoT-SS | UNJOIN _{SP} | UNJOIN _{MP} | CoT | CoT-SS | UNJOIN _{SP} | UNJOIN _{MP} |
| GPT-4o | 59.30 | 57.20 | 73.60 | 74.70 | 58.60 | 58.00 | 73.40 | 74.20 |
| GPT-4o Mini | 57.40 | 58.80 | 75.00 | 74.80 | 56.00 | 58.30 | 72.40 | 73.30 |
| Gemini 1.5 Flash | 53.00 | 62.90 | 65.30 | 66.20 | 52.40 | 61.00 | 65.30 | 65.70 |
| Mixtral | 36.00 | 62.10 | 57.00 | 66.00 | 34.30 | 60.80 | 60.00 | 67.00 |
| SQLCoder-7B | 56.00 | 62.60 | 22.10 | 60.60 | 50.70 | 63.10 | 23.30 | 58.60 |
| SQLCoder-34B | 51.80 | 51.60 | 70.10 | 60.80 | 47.80 | 47.80 | 71.60 | 60.30 |
| Llama 3.1 (3B) | 53.30 | 53.80 | 53.00 | 50.00 | 52.30 | 51.90 | 62.00 | 49.00 |
| Llama 3.3 (70B) | 67.10 | 74.30 | 75.30 | 76.80 | 67.20 | 71.20 | 74.80 | 75.00 |
| | Column Precision | | | | Column Recall | | | |
| GPT-4o | 68.80 | 68.00 | 86.30 | 86.60 | 68.20 | 65.80 | 85.40 | 85.00 |
| GPT-4o Mini | 67.20 | 69.40 | 83.10 | 85.00 | 66.00 | 65.10 | 80.00 | 83.00 |
| Gemini 1.5 Flash | 54.40 | 88.60 | 82.80 | 83.80 | 53.30 | 85.40 | 82.90 | 82.30 |
| Mixtral | 42.00 | 71.50 | 62.20 | 75.00 | 40.00 | 54.00 | 62.20 | 73.00 |
| SQLCoder-7B | 55.60 | 69.80 | 17.10 | 62.50 | 52.60 | 66.40 | 20.20 | 61.10 |
| SQLCoder-34B | 55.40 | 56.10 | 75.50 | 67.20 | 51.80 | 44.60 | 76.80 | 63.90 |
| Llama 3.1 (3B) | 59.40 | 53.70 | 43.40 | 52.00 | 58.00 | 47.00 | 66.70 | 54.30 |
| Llama 3.3 (70B) | 81.80 | 84.80 | 85.00 | 86.00 | 81.60 | 82.90 | 84.90 | 84.90 |

Table 11: Evaluation of Table/Column Precision and Recall on **BIRD** dataset.