Decomposition-Driven Multi-Table Retrieval and Reasoning for Numerical Question Answering

Feng Luo[1], Hai Lan[2], Hui Luo[3], Zhifeng Bao[2], Xiaoli Wang[4], J. Shane Culpepper[2], Shazia Sadiq[2]
[1]RMIT University [2]The University of Queensland [3]University of Wollongong [4]Xiamen University

Abstract*—In this paper, we study the problem of numerical multi-table question answering (MTQA) using tables in the wild (e.g., online data repositories). This task is essential in many analytical applications. Existing MTQA solutions, such as text-to-SQL or open-domain MTQA methods, are designed for databases and struggle when applied to the siloed tables in the wild. The key limitations include: (1) Limited support for complex table relationships; (2) Ineffective retrieval of relevant tables at scale; (3) Inaccurate answer generation. To overcome these limitations, we propose DMRAL, a Decomposition-driven Multi-table Retrieval and Answering framework for MTQA using tables in the wiLd, which consists of: (1) constructing a table relationship graph to capture complex relationships among tables; (2) Table-Aligned Question Decomposer and Coverage-Aware Retriever, which jointly enable the effective identification of relevant tables from large-scale corpora by enhancing the question decomposition quality and maximizing the question coverage of retrieved tables; (3) Sub-question Guided Reasoner, which produces correct answers by progressively generating and refining the reasoning program based on sub-questions. Experiments on two MTQA datasets demonstrate that DMRAL significantly outperforms existing state-of-the-art MTQA methods, with an average improvement of 24% in table retrieval and 55% in answer accuracy.

I. Introduction

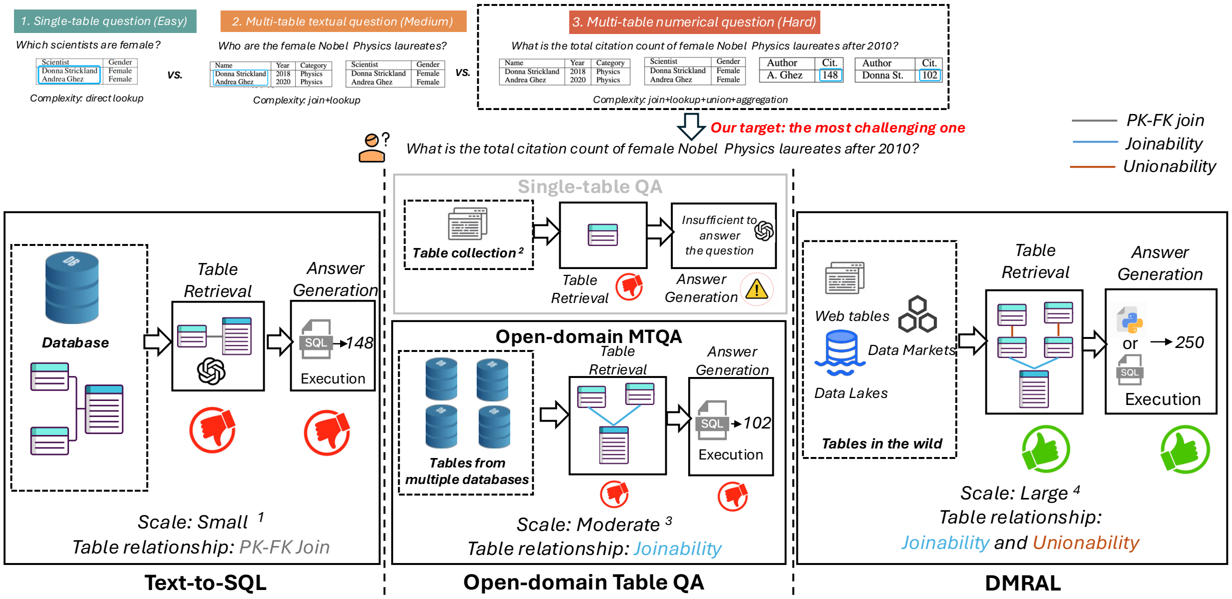
Multi-table question answering (MTQA) is a well-known task that requires identifying and integrating information from multiple tables to answer a single question [1]. Among these questions, those requiring *numerical answers* exhibit substantially greater complexity than those seeking *textual answers*. Empirical evidence indicates a pronounced performance gap: approximately 55% accuracy for numerical answers vs. 88% for textual answers, even with gold tables [2], [3], which underscores the imperative for MTQA systems to accurately support numerical questions [4]. At the same time, the growing prevalence of *tables in the wild*—such as collections of tables found on the web, in public data lakes, or available through data markets—offers rich opportunities for MTQA, where the valuable data for answering the numerical questions is often dispersed across those soiled tables. While promising, working with such tables presents unique challenges due to their *large scale* (e.g., tens of thousands of tables), potentially *incomplete metadata* (e.g., missing column headers) [5], [6], and *complex inter-table relationships*, namely unionability (tables that can be unioned on similar column headers) and joinability (tables that can be joined based on matching columns) [7], [8]. In this paper, we study numerical MTQA using tables in the wild, a challenging but increasingly realistic scenario driven by real-world data acquisition and usage trends.

Current approaches to MTQA broadly fall into two categories: *Text-to-SQL* and *Open-domain Table QA*. As shown in Figure 1, Text-to-SQL methods [9], [10] were originally developed for a single relational database, typically involving fewer than ten tables and relying on a well-defined database schema with complete table metadata and PK-FK constraints [11]. More recently, state-of-the-art methods increasingly leverage large language models (LLMs) to select the most relevant tables and generate executable SQL queries [12]. Open-domain Table QA is further divided into (i) *single-table* QA [13], [14], which assumes the answer requires information solely from a single table and is therefore incompatible with our setting, and (ii) *multi-table* QA (open-domain MTQA) [1], [15], which combines evidence from multiple tables. While open-domain MTQA is closer to our problem, it is primarily designed for modest corpora (i.e., hundreds of tables) that are aggregated from multiple relational databases. These approaches typically begin by decomposing a question into a set of sub-questions using the internal knowledge of LLMs, then retrieving relevant tables based on a relevance score between sub-questions and tables using table joinability, and finally invoking an LLM to generate SQL over the retrieved tables.

However, applying the applicable methods (i.e., Text-to-SQL and Open-domain MTQA) to the tables in the wild presents three major limitations. **(L1)** *Limited support for complex table relationships*: Both approaches overlook complex table relationships such as unionability [16], which limits their robustness in handling such table corpora. **(L2)** *Ineffective retrieval of relevant tables at scale*: With token limits, state-of-the-art LLM-based Text-to-SQL methods [10] cannot directly select relevant tables from large-scale corpora. Meanwhile, Open-domain MTQA depends on LLMs to decompose sub-questions for table retrieval, while the quality of these decompositions is often low, leading to suboptimal retrieval effectiveness. **(L3)** *Inaccurate answer generation*: While generating programs such as SQL and running them to derive answers is promising for answering numerical questions, existing approaches rarely produce fully correct programs (e.g., including incorrect joins) for execution, which result in low answer accuracy [17].

Our Contributions. To resolve these limitations, we propose DMRAL, a Decomposition-driven Multi-table Retrieval and Answering framework for numerical MTQA using tables in the wiLd. Our key contributions are as follows:

- We develop a *Preprocessing* pipeline (§II-C) that builds a



[1] We measure the scale based on the number of tables. For example, the Text-to-SQL benchmark Bird averages ~ 7 tables per database.
[2] Their evaluated datasets such as NQ-Tables and Openwiki-Tables containing only single-table questions.
[3] The datasets utilized in Open-domain MTQA study contains at most ~ 500 tables.
[4] We collected at least 73,688 tables for evaluation.

Fig. 1: A comparison of problem settings and system capabilities among existing MTQA approaches and our proposed DMRAL.

Table Relationship Graph to capture complex table relationships (i.e., addressing **L1**).

- We propose an effective decomposition-driven multi-table retrieval strategy to address **L2**, consisting of: (1) *Table-Aligned Question Decomposer* (§III), which enhances decomposition quality by identifying distinct information needs and aligning them with the underlying table structures; (2) *Coverage-Aware Retriever* (§IV), which effectively retrieves relevant tables from a large-scale table corpus by maximizing their coverage of the question through coverage scoring and verification mechanisms.
- We propose a *Sub-question Guided Reasoner* (§V) to improve answer accuracy by guiding LLMs to progressively generate and refine the program based on the decomposed sub-questions (i.e., addressing **L3**).
- To comprehensively evaluate this task, we prepare two large-scale datasets, SpiderWild and BirdWild, consisting of 73,688 and 109,949 tables curated from real-world table sources [18]–[20] (§VI). Extensive experiments demonstrate the effectiveness of DMRAL, achieving an average improvement of 24% in identifying relevant tables and 55% in producing accurate answers (§VII).

In summary, DMRAL is a robust, scalable, and traceable framework designed for numerical MTQA over tables in the wild. For each answer, DMRAL enables fine-grained tracing and verification (e.g., whether the retrieved tables are appropriate, whether the reasoning program is correct, and whether the sub-questions are well-decomposed). This traceability is crucial for MTQA, as it not only ensures transparency in how answers are derived but also provides actionable insights to diagnose and improve each component of our framework.

II. Problem, Literature, and Solution Overview

A. Problem Definition

A table T consists of a set of columns $T_c = \{c_1, \ldots, c_m\}$ and rows T_r, along with associated metadata. The metadata includes a table title T_p and column headers $T_h = \{h_1, \ldots, h_m\}$, which may be partially missing. We refer to *tables in the wild* as a table collection $\mathcal{T} = \{T^1, T^2, \ldots, T^{|\mathcal{T}|}\}$. These tables are often related through two representative inter-table relationships:

- **Joinability:** Two tables T^i and T^j are considered joinable if there exists at least one column from T^i_c and one from T^j_c such that the two columns share overlapping or semantically similar values.
- **Unionability:** Two tables T^i and T^j are considered unionable if their column headers T^i_h and T^j_h are sufficiently similar to allow a one-to-one alignment across two tables.

Definition 1 (Numerical Multi-Table Question Answering): Given a table collection $\mathcal{T}$ and a numerical question q that requires a numeric answer, it aims to compute the correct answer ans by learning the function $f : (\mathcal{T}, q) \rightarrow ans$, where the answer ans is derived by identifying a subset of relevant tables $\mathcal{T}_q \subseteq \mathcal{T}$ and performing reasoning over $\mathcal{T}_q$.

B. Related Work

Text-to-SQL. Text-to-SQL is a long-standing task that aims to convert natural language (NL) questions into executable SQL queries over relational databases. Existing Text-to-SQL approaches fall into three paradigms: rule-based, neural, and LLM-based. Rule-based methods use hand-crafted grammars and semantic parsers to map NL to SQL [21]–[23], but they require heavy manual engineering and generalize poorly to new domains [12]. Neural methods leverage deep models—encoder–decoder architectures [24], pretrained LMs [25], and GNNs [26]—to improve schema understanding and SQL generation, yet remain limited by fixed model capacity and the availability of high-quality training data. LLM-based methods generate SQL via prompt engineering [27], [28], supervised fine-tuning [29], and tool-augmented/agentic reasoning [9], [30]–[32], achieving state-of-the-art results [12].

Despite their success, these methods are not directly applicable to MTQA over tables in the wild, due to two key limitations. First, they assume explicit table relationships via PK-FK joins [33] and require complete metadata for schema linking [34], both of which are unavailable in real-world table collections. Second, the SOTA LLM-based methods are limited by the context length of LLMs, and do not scale for large-scale table corpora [35]. Given these limitations, we do not consider Text-to-SQL methods as primary baselines, while we provide an experimental evaluation to compare our effectiveness with Text-to-SQL methods in identifying relevant tables for SQL generation (§VII-E).

Open-domain MTQA. Open-domain MTQA operates over table collections by first retrieving the relevant tables using the decomposed sub-questions, and then generating SQL queries directly using LLMs [36]. To identify the relevant tables, MMQA [1] formulates the problem as a multi-hop retrieval problem. In contrast, JAR [15] formulates the problem as an optimization problem by jointly considering sub-question–table relevance and overall table relevance. However, both approaches are designed for relational database tables, which limits their ability to support identifying unionable tables from tables in the wild. Moreover, MMQA is vulnerable to cascading errors introduced by early retrieval mistakes [37], reducing its *effectiveness*. Conversely, JAR suffers from a high computational overhead due to the optimization algorithm used, limiting *efficiency*.

Other loosely related work. Open-domain Single-Table QA [13], [14] aims to answer NL questions using only the single, most relevant table retrieved from a collection of diverse tables, relying on dedicated single-table retrieval solutions such as [38] without considering inter-table relationships. This fundamental assumption makes them incompatible for integrating information from multiple tables to answer a question, thus we omit them from comparison.

Another related area is classical data mediator systems [39], [40], which process user queries over heterogeneous data sources, such as relational or graph databases. These systems define a unified global schema and establish formal schema mappings to the underlying sources. The system then utilizes these mapping relations to logically rewrite the query for execution, ensuring deductive correctness. However, this framework is strictly schema-first, requiring an explicit, defined global schema that is inherently unavailable for tables in the wild.

C. Solution Overview

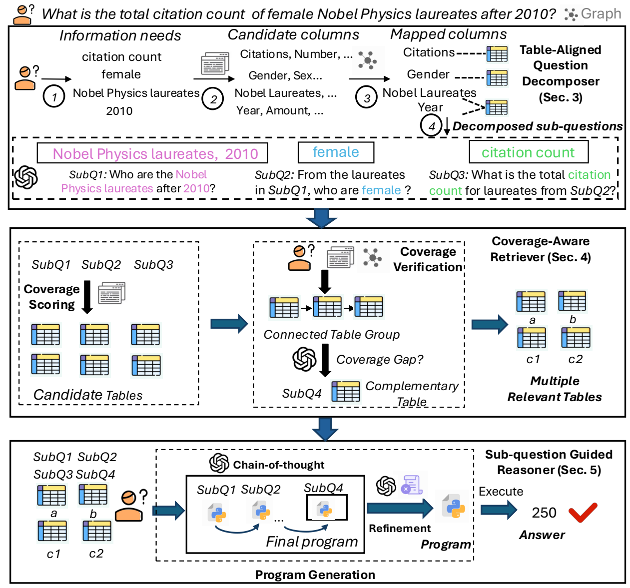


Fig. 2: MTQA processing with DMRAL, in which intermediate results at each step can be traced and refined.

DMRAL begins with a *Preprocessing* pipeline that constructs a *Table Relationship Graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to efficiently capture complex table relationships. In this graph, each node $v \in \mathcal{V}$ represents a cluster of unionable tables, identified using the unionability from [41]. Edges $e \in \mathcal{E}$ connect these clusters if any pair of tables in the clusters is joinable, using joinability defined in [15]. Built upon this preprocessed graph, DMRAL answers questions using three core modules, as illustrated in Figure 2:

Table-Aligned Question Decomposer (§III) decomposes the input question into multiple sub-questions. To improve decomposition quality, it proposes a four-step decomposition approach: (1) extract the information needs from the question, (2) align each information need to the columns, (3) select the most promising mapping between information needs and columns, and (4) group contextually coherent information needs to generate subquestions using LLM-based decomposition.

Coverage-Aware Retriever (§IV) retrieves multiple relevant tables based on the decomposed subquestions. To support effective retrieval over large scale of tables, it first introduces a learning-based coverage scoring function to retrieve candidate tables for each sub-question. Then it proposes a coverage verification function to first construct the connected table group to collectively answer the question, and then identify complementary tables to fill any potential coverage gap.

Sub-question Guided Reasoner (§V) generates executable programs (e.g., SQL or Python) over the retrieved tables to derive the final answer. To improve answer reliability, it first uses chain-of-thought prompting to incrementally generate intermediate programs using a sequence of decomposed sub-questions. It then applies an execution-guided refinement mechanism to verify and iteratively revise the generated program.

III. TABLE-ALIGNED QUESTION DECOMPOSER

A. Principles of Effective Decomposition

Correct decomposition of complex user questions is essential for MTQA, as it enables precise retrieval [15]. Given a question q, we define the information needs contained as a set $\mathcal{I}(q)$, where each element $I_q \in \mathcal{I}(q)$ represents a specific concept, entity, or condition included in q that is required to derive the correct answer.

We argue that high-quality question decomposition should satisfy these three guiding principles: (1) **Completeness**: The generated sub-questions must collectively cover all of the information needs contained in q. Missing any aspect of a question may result in an incomplete answer that fails to fully meet the user's intent; (2) **Non-redundancy**: There should be no overlap between the information needs in each sub-question. Redundant information requires unnecessary computation, increases retrieval latency, and can create misleading input for the reasoning; (3) **Table-specificity**: Each sub-question targets a single table or a group of unionable tables. This reduces the complexity of multi-table joins and ensures that each sub-question is answerable within a coherent context.

While LLMs have shown promise when decomposing questions [1], [42], [43] and improved decomposition accuracy compared to previous approaches [44], [45], our empirical analysis (§VII-B) reveals that using LLMs directly to decompose questions often fails to meet the above principles, leading to incorrect answers.

B. From Question to Table-Specific Sub-questions

To address these limitations and satisfy the principles outlined above, we propose a four-step decomposition approach that consistently aligns sub-questions with the structural information of the table collection.

1) Identifying Information Needs: To extract the most salient information needs from each question, we first parse a question into a constituency tree [1]. Then, we extract all the noun phrases, which represent the core concepts and entities required, and include adjective phrases and verb phrases that may represent the conditions, based on the syntactic labels [46]. This produces a comprehensive set of "information needs" to ensure **Completeness**.

2) Hybrid Column Matching: To ensure alignment between information needs and columns, we construct a text snippet to represent each column c_j in table T by concatenating the table title T_p, column header h_j, and distinct cell values from c_j, separated by spaces. Since information needs can differ from table contents both lexically and semantically, we use an M3-Embedding (M3) [47], a unified embedding model designed to encode both lexical and semantic features of text.

We then encode all column snippets in the table collection $\mathcal{T}$ using M3 and index them with FAISS to support efficient similarity search. For each information need I_q, we encode an M3 embedding and retrieve the top-30 column snippets based on their similarity scores. A retrieval depth of 30 was chosen empirically to balance relevance against noise (irrelevant columns).

3) Context-Aware Column Disambiguation: For each information need I_q, we identify the candidate columns $\mathcal{C}_{I_q}$, along with their similarity scores $\text{sim}(I_q, c_{I_q})$ for each $c_{I_q} \in \mathcal{C}_{I_q}$. To guide sub-question generation, we select the most promising mapping between the information needs and columns to ensure contextual coherence across information needs.

Since the information needs for a question q may be interrelated, they should refer to columns from a single table or from a set of joinable tables. To achieve this, we use a table cluster graph $\mathcal{G}$ that contains any required contextual knowledge, and define a "contextual relevance score" that jointly scores all candidate columns for each information need.

Contextual Relevance Score. Given a specific mapping M that assigns a single column from $\mathcal{C}_{I_q}$ to I_q for each $I_q \in \mathcal{I}_q$, the *context relevance score* $R(M, \mathcal{I}_q)$ is defined as $\sum_{(I_q, c_{I_q}) \in M} \text{sim}(I_q, c_{I_q})$ if $\mathbb{I}\left(\{T(c_{I_q}) \mid I_q \in \mathcal{I}(q)\}, \mathcal{G}\right) = 1$. Otherwise, $R(M, \mathcal{I}_q) = 0$. Here, $T(c_{I_q})$ denotes the table containing column c_{I_q}, and $\mathbb{I}(\cdot, \mathcal{G})$ is an indicator function returning 1 if all selected tables belong to nodes forming a connected component in $\mathcal{G}$.

Our objective is to identify a mapping M that achieves the highest contextual relevance score for all possible mappings. However, the number of candidate mappings grows exponentially with the number of information needs, which can be large. So, exhaustive search is not computationally efficient. Therefore, we apply a greedy strategy on $\mathcal{G}$. The strategy consists of the following steps.

Step 1: Ranking Information Needs. We rank all information needs in a descending order based on the maximal similarity between each information need and its candidate columns.

Step 2: Initializing the Mapping. We initialize the mapping M by selecting the candidate column with the highest similarity score for the top-ranked information need.

Step 3: Expanding the Mapping Progressively. We iterate through the remaining sorted information needs one by one. For each information need, we select the candidate column that (1) belongs to a table that is from the same connected component in $\mathcal{G}$ with the tables already included in the current mapping M, and (2) with the highest similarity score among these connected candidates. If no candidate satisfies these requirements, the current expansion is considered unsuccessful. In such cases, we backtrack to Step 2 to retry the process by selecting the next-best candidate column based on the similarity scores.

The resulting mapping M is obtained until all information needs have been successfully processed.

4) Question Decomposition: Once the optimal column selection is obtained, we group the information needs based on the tables containing the columns selected. These table-aligned groups are then sent to an LLM to generate a single sub-question for each group. This step **reduces redundancy** by ensuring that each sub-question corresponds to a disjoint set of

[1] We use the Stanford Stanza toolkit. See https://stanfordnlp.github.io/stanza/.

targeted needs, and leads to more consistent **Table-specificity** by limiting the scope of each sub-question.

IV. Coverage-Aware Retriever

Retrieving complete and relevant sets of tables is a key requirement for accurate multi-table question answering. However, existing approaches either incur high computational costs [15] or include overly rigid multi-hop pipelines that increase error rate [1]. To address this, we introduce Coverage-Aware Retriever to support more efficient multi-hop retrieval, which relies on two key innovations: (1) A coverage scoring function that prioritizes question semantic coverage to improve the effectiveness of early-stage candidate selection (§IV-A); (2) A coverage verification function that detects and corrects any missing information using *residual sub-questions* (§IV-B).

A. Maximizing Question Coverage via Learning-based Scoring

Our approach uses a two-stage pipeline. First, we perform candidate retrieval using FAISS and M3 table embeddings. Then, we apply a more fine-grained reranking model to increase the precision.

Coarse Retrieval. We represent each table cluster in $\mathcal{G}$ as a document by concatenating the table metadata and encoding it using M3. The embeddings produced are then indexed using FAISS. For each sub-question sq_i, we retrieve all of the candidate clusters by querying the index.

Learned Scoring. Coarse retrieval often introduces irrelevant documents due to superficial semantic similarity, which can propagate errors in later stages. To increase precision during candidate selection, we train a scoring function $f_\theta(q, T_q)$ that estimates the *semantic coverage* of a candidate table T_q w.r.t a question q. We use ColBERTv2 [48] to create f_θ due to its effectiveness in capturing contextual interactions.

To construct the training data, we include single-table QA datasets [18], [19]. Each training instance consists of a question q, a positive table T^+ (i.e., the ground-truth table containing the correct answer), and a hard negative table T^- created by removing the answer-bearing column from T^+ to simulate partial relevance. We train f_θ using the following margin-based ranking loss:

$$\mathcal{L} = \sum_{(q,T^+,T^-)} \max(0, 1 - f_\theta(q, T^+) + f_\theta(q, T^-)).$$

At inference time, f_θ is used to rerank the retrieved candidates based on semantic coverage.

B. Ensuring Completeness using Coverage Verification

Although reranking improves the candidate quality, non-relevant information in the retrieval stage may still lead to only partial coverage of the information needs. To address this problem, we introduce a coverage verification function using a gap detection and refinement algorithm.

Connected Table Group Construction. We construct connected table groups by selecting a single table for each sub-question such that the corresponding clusters form a connected component in $\mathcal{G}$. Each group represents a candidate set of tables that collectively cover the full question. Each group is assigned a score by concatenating all of the tables involved and applying f_θ on the concatenated content and question.

Gap Detection and Refinement. If the score of the top-ranked group is less than a predefined threshold, we assume the coverage is incomplete. Next, we use an LLM to generate a residual sub-question following [49]. We then retrieve and rerank candidate tables using this new sub-question and select an alternative table—connected to the current group—that maximizes the joint coverage score.

We finally assign each table a score based on the best-performing group containing it, and select the top-k tables for answer generation. This design enables higher-precision retrieval that is effective and scalable for large-scale tables.

Remark. While Table-Aligned Question Decomposer provides initial column-level alignments, this mapping is insufficient for robust table retrieval. Our empirical analysis shows that relying on it for retrieval leads to inaccurate mappings (60%), primarily due to semantic ambiguities in tables in the wild, or incomplete mappings (40%), where critical tables are missed because no single column strongly matches an information need. Therefore, a dedicated Table-Aligned Question Decomposer is essential to identify a complete and relevant set of tables, ensuring accurate MTQA.

V. Sub-question Guided Reasoner

Once the relevant tables are retrieved, a common *answer generation* strategy is to perform the reasoning over tables directly using a sequence-to-sequence model [3]. However, this strategy performs poorly on numerical reasoning tasks, often failing to model necessary arithmetic operations. Program-based reasoning [16] offers a structured alternative – by first generating an executable program (e.g., SQL or Python) that is then executed over multiple tables to produce the final answer. However, existing program generators are unreliable when applied to a collection of soiled tables as they frequently select non-relevant tables, fail to infer which tables can be joined, or contain incorrect reasoning steps, leading to low-quality results [17].

To address these challenges, we propose **Sub-question Guided Reasoner**. Instead of generating the entire program in one shot, our approach incrementally constructs the program using a sequence of decomposed sub-questions. Each sub-question can be reliably answered using a single table or a unionable group, and inter-sub-question dependencies determine which intermediate results should be joined to get a complete reasoning program. The resulting program is then executed to derive the final answer. This design leads to a more modular, interpretable, and accurate reasoning pipeline, capable of supporting diverse types of reasoning programs over siloed retrieved tables.

A. Program Generation

Given the original question, a set of decomposed sub-questions, and the retrieved tables, our reasoner generates a program in two stages:

CoT-Guided Multi-step Program Generation. Following recent advances in multi-hop reasoning [50], we integrate chain-of-thought (CoT) prompting to generate each program step by step based on the sequence of sub-questions. The process begins by generating an initial sub-program that uses a unionable group of tables relevant to the initial sub-question. Then, for each subsequent sub-question, the program is incrementally improved by joining the current intermediate program with a new sub-program generated for that sub-question. The final executable program is obtained when all sub-questions have been processed.

This step-by-step process provides two benefits: (1) It explicitly encodes reasoning to solve high complexity tasks using table dependencies as input; (2) It enables more robust program construction by constructing each reasoning step based on smaller, coherent subsets of the table collection. To handle column-level inconsistencies when joining unionable tables, we also include a fuzzy-join operator [51] when necessary.

Execution-guided Refinement. Despite the use of carefully constructed prompts, the generated program may still contain errors (e.g., syntax errors). To improve robustness, we introduce an *execution-guided refinement* step in our solution. We first execute the program using the retrieved tables and check the output for any failures. If errors are detected, we re-prompt the LLM and include the error message to help refine the program. This process is repeated until a valid program is generated or a maximum retry limit is reached.

VI. Data Preparation for Evaluation

In this section, we first examine the limitations of existing benchmarks for our problem setting. Then, we introduce our designed solutions to prepare the evaluation datasets.

A. Motivation

To the best of our knowledge, there are no existing benchmarks that support the evaluation of numerical MTQA over tables in the wild. First, datasets commonly adapted for MTQA evaluation [1], [15] are directly sourced from text-to-SQL benchmarks, such as Spider [18], and Bird [19]. These original and derived benchmarks typically consist of tables from one or multiple databases, with limited scale, complete metadata, and focus only on explicit PK-FK joins. Thus, they fail to reflect the *scalability*, *incomplete metadata*, and *complex table relationships* for tables in the wild. Second, popular table QA benchmarks, such as NQ-Tables [13] and Open-WikiTables [52], are designed for open-domain single QA, where each question can be answered using a single table. Thus, they lack the multi-table question–answer pairs necessary to evaluate our MTQA problem.

B. Preparation Process

To address these gaps, our data preparation follows two steps. First, we collect real-world question–answer pairs that are explicitly grounded in multiple relevant tables. Second, we expand these grounded tables into a large-scale table repository, ensuring the *scalability*, *incomplete metadata*, and *complex table relationships* characteristic of tables in the wild.

To realize this, we repurpose the question–answer pairs and their grounded tables from existing text-to-SQL benchmarks, Spider [18] and Bird [19]. The resulting datasets are referred to as SpiderWild and BirdWild. In the next, we first introduce how we prepare tables in the wild by repurposing grounded tables and introducing external tables, and then make the question annotation.

1) Repurposing Grounded Tables: For each text-to-SQL benchmark, we begin by aggregating all tables from the original databases as the ground tables into a centralized repository. To reflect the characteristics of tables in the wild, we then apply a three-stage table transformation pipeline:

Stage 1: Table Decomposition. To overcome the limited scale and inter-table relationship diversity of the original datasets, we adopt the idea of table decomposition inspired by [20]. Different from their random decomposition, our decomposition is designed to produce more semantically meaningful tables. Specifically, we decompose large tables (i.e., those with more than 5 columns and 50 rows) into multiple disjoint subtables using both column-wise and row-wise splitting strategies. This design is motivated by common organizational patterns observed in data lakes, where tables are often constructed using semantically related columns (e.g., `Year`, `Month`, `Day`) or partitioned by value ranges or categories [41], [53].

For the column-wise splitting strategy, we first identify key columns (i.e., columns with all distinct values) and use the LLM [2] to group the remaining non-key columns into semantically related column subsets. To achieve this, we design a custom prompt to instruct the model to cluster columns by topic or theme based on their column headers. Each column group is then combined with a randomly selected key column to form a new subtable. To ensure the decomposed subtables can be joined together to reconstruct the original table (i.e., values from the same row in the original table remain correctly aligned), we verify whether any subtable contains all key columns. If no such subtable exists, an additional subtable containing all key columns is created. Since the original database tables only provide the PK-FK joins, we aim to introduce additional joinability of semantic joins. Thus, we choose subtables derived from the same key column that does not participate in PK-FK joins to create joinability.

Row-wise splitting strategy is applied after the column-wise step, which further partitions each subtable based on the distribution of a randomly selected non-key column. If a numerical column is chosen, we partition the rows into a random number of buckets (between 5 and 20) based on value ranges (e.g., splitting a `sales` table by ranges of `sale amounts`). If a categorical column is chosen, we randomly divide the table into 2 to 20 tables, each containing rows for a specific group of categories (e.g., splitting a `sales` table by `region` might yield separate tables for "North America Sales", "Europe Sales" etc.). This row-wise splitting produces

[2]The LLM utilized here is GPT-4o-mini.

unionable tables that share the same column headers but cover different table content, which mirrors the common real-world occurrence of unionable tables, as seen in the data versioning of Wikipedia tables [54].

Stage 2: Metadata Incompleteness Simulation. To mimic the incomplete metadata, we randomly select 20% of the decomposed tables and mask 50% of their column headers and table titles, using the placeholder `MASK` based on the incomplete metadata statistics reported in [5], [6]. For example, an `employee` table with columns `emp_id`, `department`, `salary`, and `role` may be transformed into `MASK` table with columns `MASK`, `department`, `MASK`, and `role`.

Stage 3: Joinability Simulation. Among various types of joinability, fuzzy joins (where values differ slightly in spelling or format) are particularly common in real-world data lakes [55]–[57]. To simulate this, we inject value-level variations into the textual key columns of decomposed tables. Specifically, following [57], we introduce perturbations such as typographical errors and character deletions into 20% of the cell values. This transformation creates realistic join scenarios where approximate string matching is required (e.g., `"New York"` vs. `"Nw York"`).

2) Incorporating External Tables: While our table decomposition strategy increases the number of tables from hundreds to thousands, it remains insufficient to reflect the large-scale settings typically encountered in practice, such as the massive collections of web tables available online. To further enhance scalability, we incorporate additional tables from the widely used table corpora WebTables and OpenData [20], considering two dimensions: (1) tables relevant to the questions—to ensure that the tables are useful for answering the question, and (2) tables from a similar domain as existing tables, where domain relevance is estimated using the relevance of table titles–to simulate realistic distractor tables. Specifically, for each collected question or decomposed table, we retrieve the top-N candidate tables from these corpora using BM25 [58] over both the question and the table names. This enrichment significantly enlarges the table repository. For example, with N=100, the number of tables increases from 2,210 to 73,688 on SpiderWild, and from 5,136 to 109,949 on BirdWild.

3) Question Annotation: Finally, we curate a set of numerical questions from the original questions that require reasoning over multiple tables. For each selected question, we then construct annotations consisting of both the ground-truth answer and the set of relevant tables to answer this question. Specifically, we first execute the original SQL query on their provided databases to obtain the ground-truth answer. Next, to determine the relevant tables, we analyze how the original SQL query draws information from multiple tables. To achieve this, we manually rewrite the query to extract the values of the primary key and other relevant columns (e.g., those mentioned in WHERE or GROUP BY clauses) for each relevant table. We then identify the specific table rows and columns containing these values. These records are then mapped back to the decomposed tables in the table repository by locating the tables that contain the corresponding rows and columns. The subset of decomposed tables that jointly cover all these records is regarded as the set of relevant tables for that question.

Property	SpiderWild	BirdWild
# Tables	73,688	109,949
% Joinability	9%	13%
Avg. # Columns	4.4	5.0
Avg. # Rows	1,384	8,629
# Numerical Questions	274	461
% Easy/Moderate/Hard	52%/43%/5%	10%/84%/6%
% Relevant Table (2 / >=3)	67% / 33%	64% / 36%
% Incomplete Metadata	42%	48%
% Requiring Unionability	13%	56%

TABLE I: Dataset statistics.

VII. Experimental Evaluation

Evaluation Goals.

Primary Evaluation Goals. We aim to assess the effectiveness, efficiency, scalability, and robustness of our proposed method in realistic settings of MTQA over tables in the wild. Specifically, we answer the following four questions:

- **Q1: How well does our method retrieve multiple relevant tables and answer numerical questions?** Unlike prior work in open-domain MTQA, which struggles to retrieve relevant tables at scale and generate reliable answers (§II-B), our framework aims to address such limitations. We evaluate the effectiveness and efficiency, in both table retrieval and answer accuracy, against strong baselines. (§ VII-A)
- **Q2: What is the contribution of our key design choices and parameter settings to overall performance?** We perform ablation studies to assess the impact of core components (i.e., preprocessing, decomposer, retriever, and reasoner) and analyze the sensitivity of key parameters such as table joinability and unionability thresholds. (§ VII-B)
- **Q3: How well does our method scale with the size of the table corpus?** We evaluate whether DMRAL maintains high effectiveness and low latency as the number of tables in the table corpora increases. (§ VII-C)
- **Q4: How robust is our method under different challenging scenarios?** In particular, we evaluate its robustness w.r.t. the varying number of involved tables, their unionability, and degrees of completeness of metadata. (§ VII-D)

Secondary Evaluation Goal: Comparison with Text-to-SQL Methods in Multi-Table Retrieval. As discussed in §II-B, text-to-SQL methods are designed for relational databases, where complete metadata and explicitly defined table relationships are available. This setting provides advantages that are not present in our scenario. Nevertheless, both our approach and text-to-SQL methods involve a common step, namely, the identification of relevant tables. Accordingly, we assess the effectiveness of this table identification process.

A. Evaluation on Table Retrieval and Answer Generation

1) Experimental Setup: We evaluate DMRAL on the prepared SpiderWild and BirdWild datasets, with the construction details in §VI. There are three primary factors that directly

Top-k	Method	SpiderWild									BirdWild								
		Easy			Moderate			Hard			Easy			Moderate			Hard		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Top-3	JARUnion	46.5	69.7	55.8	45.2	60.6	51.3	44.8	59.8	50.7	39.7	55.0	45.8	39.2	53.7	45.7	45.0	53.7	44.8
	MMQAUnion	47.2	70.8	65.6	37.9	52.4	43.7	37.9	52.1	43.5	41.0	56.6	49.1	36.7	49.7	41.8	35.9	48.8	41.0
	DMRAL	**54.9**	**82.4**	**65.9**	**52.9**	**71.3**	**60.2**	**52.3**	**70.3**	**59.4**	**52.2**	**71.0**	**59.6**	**52.7**	**70.7**	**59.8**	**51.5**	**69.5**	**58.6**
Top-5	JARUnion	31.3	78.2	44.7	32.3	71.5	44.1	31.9	70.4	43.5	28.6	65.6	39.5	28.4	64.4	39.1	28.1	64.1	38.8
	MMQAUnion	31.1	77.8	44.5	26.4	60.3	36.5	26.2	59.7	36.2	26.7	60.9	36.9	26.9	60.7	37.0	26.3	59.6	36.2
	DMRAL	**34.2**	**85.6**	**48.9**	**33.1**	**74.4**	**45.4**	**32.7**	**73.5**	**45.0**	**33.3**	**74.8**	**45.4**	**33.3**	**74.7**	**45.7**	**32.8**	**73.7**	**45.0**

TABLE II: Comparison of table retrieval effectiveness on SpiderWild and BirdWild datasets.

impact MTQA effectiveness: (1) the number of table joins, (2) the presence of union operations, and (3) whether the relevant tables contain incomplete metadata. To facilitate a more fine-grained evaluation over these factors, we categorize the questions in both datasets into three levels of complexity—*Easy*, *Moderate*, and *Hard*—based on the number of table joins and unions required to derive the correct answer, as well as whether incomplete metadata is involved in the associated tables.

Specifically, *Easy* questions involve only two table joins, require no union, and rely on complete metadata. By contrast, *Hard* questions require more than two joins, at least one union, and involve incomplete metadata in their associated tables. Questions that do not fully meet the criteria for either the Easy or Hard category are classified as *Moderate*. Table I summarizes the dataset statistics.

Evaluation Metrics. Following previous work [1], [15], we use Precision@k (P@k), Recall@k (R@k), and F1@k to evaluate the effectiveness for multi-table retrieval. To measure answer accuracy on numerical questions, we adopt Arithmetic Exact Match (EM@k) from [59], which measures the percentage of answers obtained by executing the generated program using top-k retrieved tables that match the ground truth. We choose 3 and 5 for k following the previous study [1].

Implementation & Hardware. We use the BGE-M3 [3] to encode the questions and tables, due to its effectiveness in retrieval tasks [60]. We set the unionability and joinability thresholds to 0.9 and 0.5, respectively. We include `GPT-4.1 mini` as the primary LLM model. The prompts used by our framework are provided in §B of our technical report [61].

All experiments were conducted on a server running Red Hat 7.9, equipped with an Intel(R) Xeon(R) E5-2690 CPU, 512GB RAM, and an NVIDIA Tesla P100 GPU with 16GB of memory. Our source code is available at [61].

Competitors. We evaluate our approach against JARUnion and MMQAUnion, which are adapted from the state-of-the-art open-domain MTQA systems JAR [15] and MMQA [1], respectively. Both methods were originally designed for databases with complete metadata and do not consider table unionability. To ensure a fair comparison, we apply their original table retrieval strategies on our processed metadata-complete tables and augment the retrieved tables by incorporating unionability. Specifically, for each retrieved table, we expand it by merging all tables in its cluster from our constructed graph to form the final retrieved tables. We then use their methods on these retrieved tables to obtain answers.

2) Effectiveness for Multi-table Retrieval: Table II presents the table retrieval effectiveness results across varying levels of question complexity. We highlight two key observations: (1) DMRAL consistently outperforms all baseline methods across both datasets and all complexity levels. For example, in terms of R@3, it achieves an average relative improvement of 17.8% on SpiderWild and 30.1% on BirdWild compared to the second-best baseline JARUnion. (2) Almost all methods exhibit a noticeable performance drop as question complexity increases. For example, on SpiderWild, the R@3 recall for Easy questions is up to 17% higher than for Hard ones, and on BirdWild the gap is also around 2%. These trends highlight an increased difficulty of retrieving relevant tables when questions require more joinable and unionable tables, especially in the presence of incomplete metadata.

3) Effectiveness for Answer Generation: Table III presents the answer accuracy using the top-k retrieved results. We report EM scores under both Top-3 and Top-5 retrieval settings, stratified by question complexity. We make several key observations: (1) DMRAL consistently outperforms all baselines across both datasets and all complexity levels. For example, under the Top-3 setting, DMRAL achieves relative EM improvements of 40%, 51%, and 100% over the second-best baseline JARUnion on Easy, Moderate, and Hard questions in the SpiderWild dataset, respectively. (2) As question complexity increases, answer accuracy also drops significantly on both datasets. This suggests that more complex questions—those involving more joins, unions, and incomplete metadata present greater challenges not only in identifying the correct tables but also in generating correct answers via reasoning. (3) Interestingly, the drop in answer accuracy across complexity levels is much larger than that of multi-table retrieval. For example, on BirdWild, DMRAL shows only a 2% drop in table retrieval performance R@3 from Easy to Hard questions, but EM@3 drops by 70%. This indicates that even with accurate table retrieval, reasoning over multiple tables becomes significantly more challenging as the complexity of joins, unions, and incomplete metadata increases.

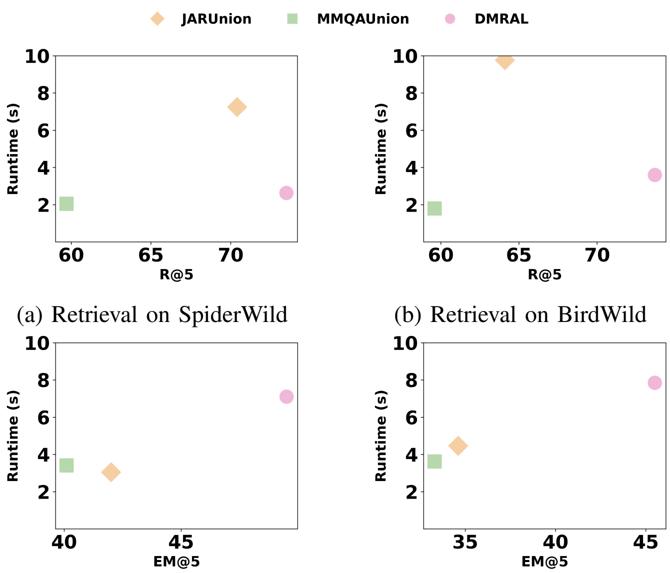
4) Efficiency for Multi-table Retrieval and Answer Generation: To evaluate the efficiency of MTQA methods, we

[3] https://bge-model.com/bge/bge_m3.html

[4] The method offering the optimal trade-off appears toward the bottom-right in each plot.

Top-k	Method	SpiderWild			BirdWild		
		Easy	Moderate	Hard	Easy	Moderate	Hard
Top-3	JARUnion	40.8	28.9	10.0	28.8	24.9	1.2
	MMQAUnion	40.8	25.6	14.0	26.7	24.9	3.8
	DMRAL	**57.0**	**43.8**	**20.0**	**45.7**	**43.3**	**26.9**
Top-5	JARUnion	46.5	38.0	30.0	41.7	36.0	3.8
	MMQAUnion	46.5	33.9	28.0	35.1	31.9	11.5
	DMRAL	**57.7**	**41.3**	**35.0**	**47.5**	**44.6**	**19.2**

TABLE III: Answer accuracy comparison using top-k retrieved tables .



(a) Retrieval on SpiderWild

(b) Retrieval on BirdWild

(c) Answer Generation on SpiderWild

(d) Answer Generation on BirdWild

Fig. 3: Efficiency breakdown of table retrieval and answer generation, measured by the average runtime per question[4].

measure the average runtime per question for both multi-table retrieval and answer generation across all questions in each dataset. Note that for the runtime of multi-table retrieval of DMRAL, we include the time consumed by both the question decomposer and retriever modules. Figure 3 presents a comparative analysis of efficiency, along with the corresponding effectiveness. From the results, we observe: (1) For table retrieval, DMRAL achieves a strong balance between effectiveness and latency. Compared to the most efficient baseline, MMQAUnion, it yields an average of 15% higher R@5 on both datasets, incurring only a 1.6× increase in runtime. (2) For answer generation, DMRAL delivers substantial accuracy gains—achieving 18% and 31% higher EM@5 scores on SpiderWild and BirdWild, respectively, compared to the second-best method JARUnion, at the cost of an average 3.7 seconds per question. This additional time cost stems from two reasoning-oriented mechanisms incorporated in the reasoning component of DMRAL to enhance program robustness: (i) a step-by-step CoT prompting strategy to generate the reasoning program incrementally, and (ii) a refinement mechanism that verifies and improves the generated program before execution.

Method	SpiderWild				BirdWild			
	P	R	F1	EM	P	R	F1	EM
Top-3								
No-fill	42.5	55.8	46.0	42.6	40.9	57.0	47.6	28.4
Direct LLM	50.3	67.0	57.4	44.5	48.6	66.2	56.0	40.3
DMRAL	52.3	70.3	59.4	49.0	51.5	69.5	58.6	44.3
GT	59.1	78.0	67.3	51.9	57.2	76.5	65.4	45.2
Top-5								
No-fill	25.3	59.8	35.7	43.0	26.7	58.5	36.6	29.2
Direct LLM	31.4	70.0	43.4	45.0	30.9	70.2	43.0	41.4
DMRAL	32.7	73.5	45.0	49.5	32.8	73.7	45.0	45.5
GT	37.0	81.6	50.9	52.5	36.4	81.1	50.2	46.4

TABLE IV: The impact of metadata quality to the effectiveness of table retrieval and answer generation.

B. Ablation Study and Parameter Study

1) Ablation Study: We conduct a comprehensive ablation study to evaluate the impact of our key designs – namely the preprocessing (§II-C), decomposer (§III), retriever (§IV), and reasoner (§V) components, for the effectiveness.

Preprocessing. Since the real-world table corpora may contain the incomplete metadata, which can distort unionability calculation during graph construction, we also introduce a metadata inference module. It leverages LLMs to infer missing column headers by incorporating intra-table context, inspired by [62] (see §A of our technical report). In the following, we will evaluate the inference quality of this module against a naive baseline, *Direct LLM*, which directly prompts an LLM to complete missing metadata without using the intra-table context. Following prior work [36], we use *BERT-F1* score to assess quality: for each missing column header, we compute the F1 score between the predicted and ground-truth header based on contextual embedding similarity, and report the average across all the missing headers. Our approach consistently outperforms the *Direct LLM*: on SpiderWild, BERT-F1 increases from 0.602 to 0.697, and on BirdWild from 0.602 to 0.652. These gains indicate that exploiting intra-table context materially improves metadata inference accuracy.

We further evaluate how metadata quality affects downstream retrieval and answer accuracy by comparing four configurations: (1) *No-fill*, which leaves incomplete metadata unchanged; (2) *Direct LLM*; (3) DMRAL, which is our proposed metadata inference module; (4) *GT*, which uses the corresponding ground-truth metadata to replace all incomplete metadata. The results are presented in Table IV, covering both Top-3 and Top-5 retrieval settings. From the results, we observe that DMRAL consistently improves over the *No-fill* and *Direct LLM* baselines across all evaluation metrics on both SpiderWild and BirdWild datasets. These results highlight the importance of completing missing metadata and the effectiveness of using intra-table context for metadata recovery. Moreover, the performance gap between DMRAL and *GT* setting is relatively narrow, as compared to *No-fill*. This demonstrates that our inference strategy closely approximates gold metadata and significantly contributes to

Method	SpiderWild			BirdWild		
	IRR (↑)	SR (↓)	SAR (↑)	IRR (↑)	SR (↓)	SAR (↑)
Direct LLM	93%	0.589	57%	94%	0.600	59%
DMRAL	96%	0.521	71%	96%	0.522	70%

TABLE V: Comparison of decomposition quality. ↑: higher is better, ↓: lower is better.

Method	SpiderWild				BirdWild			
	P	R	F1	EM	P	R	F1	EM
Top-3								
Direct LLM	50.9	68.3	57.8	45.4	49.5	67.2	56.5	38.9
DMRAL	**52.3**	**70.3**	**59.4**	**49.0**	**51.5**	**69.5**	**58.6**	**44.3**
Top-5								
Direct LLM	31.5	70.7	43.2	49.1	32.0	72.1	43.9	39.6
DMRAL	**32.7**	**73.5**	**45.0**	**49.5**	**32.8**	**73.7**	**45.0**	**45.5**

TABLE VI: Effectiveness comparison using our decomposer vs. direct LLM-decomposer.

both table retrieval and answer generation in MTQA.

Table-Aligned Question Decomposer. We first compare our decomposition strategy against a *Direct LLM* generation approach, which uses an LLM to generate the sub-questions without any structural alignment information. To assess the quality of decomposition, we use three metrics corresponding to the criteria defined in §III – (i) *Information Retention Rate (IRR)* to measure completeness: The proportion of questions whose decompositions successfully preserve all information needs required; (ii) *Subquestion Redundancy (SR)* to measure redundancy: The average pairwise semantic similarity [5]; (iii) *Subquestion-Table Alignment Rate (SAR)* to measure table-specificity: Determine if the number of sub-questions matches the number of relevant tables required to answer the question.

From Table V, we observe that our table-aligned question decomposer consistently improves all three metrics using both datasets. These results demonstrate the effectiveness of our decomposition strategy which leverages the structure of the table corpus. An additional case study illustrating the improvements is shown in §C of our technical report [61].

Next, we deepen our study of the impact of decomposition on both retrieval effectiveness and final answer accuracy in Table VI. Observe that our decomposition consistently achieves better performance on Top-3 and Top-5 retrieval settings. These results demonstrate that DMRAL improves both retrieval effectiveness and final answer accuracy by improving the quality of decomposed sub-questions.

Coverage-Aware Retriever. Now, we evaluate the retrieval effectiveness of the coverage scoring function and coverage verification submodule using two ablations: (1) *NaiveScoring* is used to replace our trained coverage scoring function using a simple baseline which uses the sum of the individual embedding similarities computed between sub-questions and candidate tables, obtained using our coarse retrieval method described previously. (2) *w/o Verification* disables our residual

[5]The similarity is computed using embeddings generated from a pretrained Sentence-BERT model.

Method	SpiderWild		BirdWild	
	R@3	R@5	R@3	R@5
DMRAL	**70.3**	**73.5**	**69.5**	**73.7**
NaiveScoring	62.6 (−11)	66.7 (−9)	66.4 (−4)	71.3 (−3)
w/o Verification	56.3 (−20)	60.7 (−17)	55.9 (−20)	64.7 (−12)

TABLE VII: Ablation study of Coverage-Aware Retriever[6].

Method	SpiderWild		BirdWild	
	EM@3	EM@5	EM@3	EM@5
DMRAL	**49.0**	**49.5**	**44.3**	**45.5**
w/o CoT	44.8 (−10)	45.2 (−10)	34.1 (−23)	37.0 (−19)
w/o Refinement	48.3 (−3)	49.7 (−1)	37.8 (−10)	42.0 (−8)

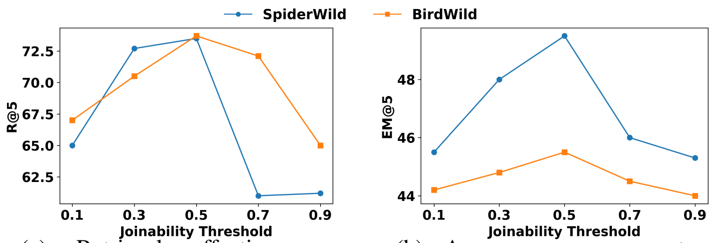
TABLE VIII: Ablation study of Sub-question Guided Reasoner.

sub-question generation that is used to retrieve complementary tables. In Table VII, we find: (1) Our trained coverage scoring function provides more effective retrieval performance by identifying subsets of tables that cover the question intent more fully. (2) Our coverage verification submodule enhances retrieval performance by filling potential coverage gaps introduced by non-relevant tables –those that appear to be relevant to the question but cannot provide a complete answer.

Sub-question Guided Reasoner. Assessing the benefit of our reasoning module is achieved by comparing it to two other variants: (1) A *w/o CoT* baseline that prompts the LLM to generate a program in a single shot using the original question and the most similar tables retrieved; (2) A *w/o Refinement* executes the first program generated with no further refinement. The answer accuracy is shown in Table VIII, which illustrates that: (1) Our chain-of-thought generation substantially improves answer quality, achieving up to 19% better EM scores. This result demonstrates the benefits of modeling the reasoning process using sub-questions, which enables the LLM to correctly infer table relationships for generating an accurate program, thus producing reliable answers. (2) Our execution-guided refinement further improves answer quality by ensuring the correctness of the generated program.

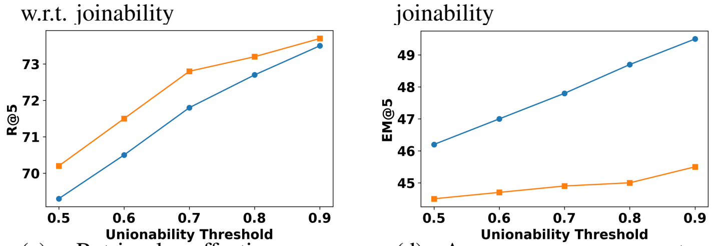
2) Parameter Study: We conduct a parameter study to investigate the impact of different thresholds for *joinability* and *unionability* when modeling table relationships (§ II-C). We set the threshold ranges—[0.1, 0.3, 0.5, 0.7, 0.9] for joinability and [0.5, 0.6, 0.7, 0.8, 0.9] for unionability. The lower bounds are selected based on their strong pruning effects observed across all table pairs, where they effectively filter out a large portion of low-quality or spurious relationships across tables. Figure 4 illustrates the retrieval and answering effectiveness under varying threshold settings. Observe that: (1) For joinability, both performances improve as the threshold increases from 0.1 to 0.5, then decline. This is because very low thresholds (e.g., 0.1) admit many noisy or spurious links between unrelated tables, which dilute the quality of

[6]Values in parentheses indicate relative performance drops compared to our full method.



(a) Retrieval effectiveness w.r.t. joinability

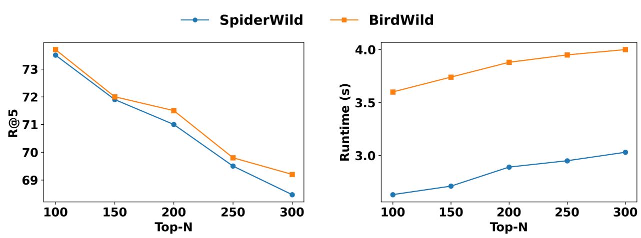
(b) Answer accuracy w.r.t. joinability



(c) Retrieval effectiveness w.r.t. unionability

(d) Answer accuracy w.r.t. unionability

Fig. 4: Impact of joinability and unionability thresholds on retrieval and answer accuracy.



(a) Retrieval effectiveness w.r.t. table scale

(b) Retrieval efficiency w.r.t. table scale

Fig. 5: Retrieval effectiveness and efficiency as the number of tables increases for DMRAL.

retrieval and introduce irrelevant candidates for reasoning. While very high thresholds (e.g., 0.9) are overly restrictive, they exclude moderately joinable tables, which are still necessary to derive the correct answer. (2) For unionability, both performance consistently improves as the threshold increases. This is because stricter thresholds ensure that only highly unionable tables—those with strong semantic similarity—are grouped together. This reduces the inclusion of semantically misaligned tables in union groups, thereby enhancing both retrieval precision and the grounding quality for reasoning.

C. Scalability Study

1) Experimental Setup: To evaluate the scalability of DMRAL under varying sizes of table corpus, we vary the number of top-N external tables retrieved per question/table (§ VI-B2) with $N \in \{100, 150, 200, 250, 300\}$. These settings yield table corpora containing approximately 109K, 149K, 183K, 214K, and 243K tables, respectively. Since the scale of the table corpus largely influences the table retrieval effectiveness and efficiency, we are mainly focusing on the retrieval effectiveness measured by R@5, and the retrieval efficiency measured by the retrieval time cost per question.

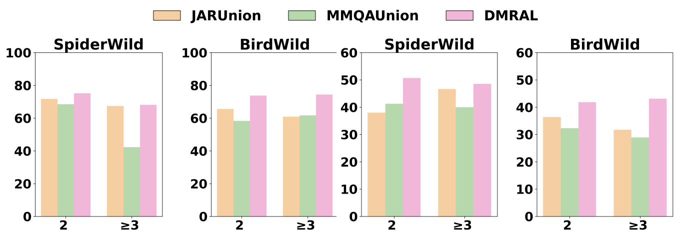
2) Main results: Figure 5 presents the results. As the size of the table corpus increases, we observe that DMRAL remains robust in retrieval effectiveness—exhibiting only a modest 5% drop in R@5 despite more than doubling the number of tables. In terms of efficiency, the query time grows gradually with scale, indicating that the method remains computationally efficient even under a larger table corpus. For the offline graph construction efficiency, using the largest 243K tables on both datasets requires approximately 4.2 days to compute joinability (with 5 parallel workers), while unionability takes about one day since it operates over table pairs which are fewer than the column pairs required for joinability. Looking ahead, we plan to reduce this preprocessing cost via distributed engineering techniques (e.g., sharding and parallel similarity joins) [63]. We also plan to support scalable, low-cost dynamic updates so that new tables and relationships can be incorporated incrementally without full recomputation.

D. Robustness Cross Challenge Scenarios

Robustness on Questions Requiring a Varying Number of Relevant Tables. To evaluate robustness across questions involving different numbers of tables, we group questions by the number of relevant tables retrieved (2 vs. $\geq$3) and assess both retrieval effectiveness and answer accuracy. As shown in Figure 6, DMRAL consistently outperforms all other baselines for both groupings, demonstrating the robustness of our solution on questions of higher cognitive complexity. This is because: (1) Our retriever explicitly optimizes for question coverage, ensuring that the necessary tables are selected even as the required set grows; (2) Our reasoner generates programs that are guided using decomposed sub-questions, which enables accurate inference of table relationships for questions of increasing difficulty.

Robustness of Questions Involving Incomplete Metadata. We also consider the robustness of our method by comparing the performance on questions with relevant tables, which include incomplete metadata, with the complete metadata case. The results are shown in Figure 7. Observe that: (1) All methods have degraded performance when introducing questions that require tables that have incomplete metadata. This highlights the challenges that incomplete metadata can introduce. (2) DMRAL is consistently the most effective approach for retrieval and answer effectiveness across both datasets, with notably less performance degradation when compared to the baselines (e.g., an average EM@5 drop of 26% for DMRAL vs. 62% for the baselines). These results demonstrate DMRAL is more robust when incomplete metadata exists.

Robustness on Questions that Require Unionable Tables. We now compare the performance of our method using questions that require unionable tables to answer versus those that do not. The results are shown in Figure 8. Once again, DMRAL is consistently the most effective retrieval model and produces higher-quality answers for both question types and datasets. While existing baselines have notable performance degradation when answering questions that require one or more union operations, DMRAL is consistently good in both



(a) Retrieval: R@5 (b) Answer: EM@5

Fig. 6: Effect of the number of relevant tables (2 vs. $\geq$3) on effectiveness.

(a) Retrieval: R@5 (b) Answer: EM@5

Fig. 7: Evaluation on questions involving tables with incomplete vs. complete metadata.

scenarios. The advantage stems from our design decisions, which explicitly model table unionability using a graph and groups unionable tables into clusters during the retrieval and reasoning stage. In contrast, the extended baselines assume that each table is independent and fail to incorporate unionability into the retrieval model, reducing their ability to locate multiple joinable tables and use them for reasoning.

E. Comparison with Text-to-SQL Methods

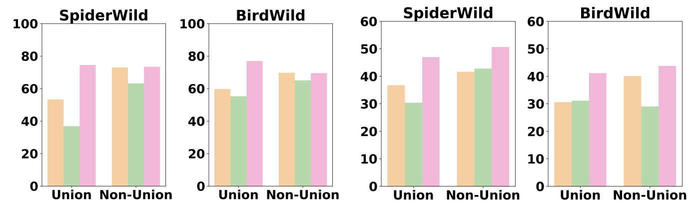
1) Experimental Setup: Text-to-SQL methods typically assume a simplified MTQA setting, where all tables reside within a relational database, with complete metadata and explicit table relationships (i.e., PK-FK constraints). We evaluate these methods under their standard setup, where all the tables and structured information are available for SQL generation.

In contrast, DMRAL is designed for the scenario involving tables in the wild, where explicit intra-table relationships and complete metadata are unavailable. To enable the comparison using DMRAL, we construct a localized table corpus for each question by aggregating all tables from the corresponding relational database. We then apply our full pipeline, treating SQL as the final target reasoning program.

Dataset. We use the Bird development set [19], and focus on the curated subset of 45 numerical questions that require integration and reasoning over multiple tables.

Competitors. We compare against two resource-efficient, high performing Text-to-SQL systems on Bird benchmark [64]:

- **CHESS** [32]: A multi-agent LLM-based framework that decomposes SQL generation into four stages—table retrieval, schema pruning, candidate generation, and query validation.
- **OpenSearch-SQL** [28]: A lightweight and modular pipeline that improves SQL generation via structured CoT prompting, SQL-Like intermediate representation, and consistency alignment.



(a) Retrieval: R@5 (b) Answer: EM@5

Fig. 8: Evaluation on questions that require integrating unionable tables (Union) versus those that do not (Non-Union).

Method	Table Selection Accuracy	Execution Accuracy
CHESS	0.733	0.733
OpenSearch-SQL	**0.880**	**0.867**
DMRAL	0.867	0.778

TABLE IX: Comparison with Text-to-SQL baselines.

Evaluation Metrics. We adopt two standard metrics commonly used in Text-to-SQL [18], [19], with table selection accuracy as the primary metric for table identification:

- **Table Selection Accuracy.** This metric is computed as the percentage of the predicted SQL query references exactly the ground-truth relevant tables.
- **Execution Accuracy.** This metric is computed as the percentage of predicted SQL query yields the correct answer when executed on the database, matching the executed result using the ground-truth SQL query.

2) Main Results: Table IX reports the results on table selection and SQL execution accuracy. From the results, we observe that DMRAL achieves competitive table selection accuracy to the best-performing baseline, OpenSearch-SQL, and clearly outperforms CHESS. This demonstrates its effectiveness in identifying the correct table subset, despite lacking access to explicit PK-FK constraints. In terms of execution accuracy, DMRAL achieves reasonable performance, with most failures (57%) stemming from value errors (i.e., the values used in the predicted SQL query do not align with the values in the database) due to our reasoner is not specifically optimized for SQL generation.

VIII. Conclusion

In this paper, we propose DMRAL, a novel decomposition-driven multi-table retrieval and answering framework designed for numerical MTQA over tables in the wild. DMRAL operates with four modules: Preprocessing, which constructs a graph to capture complex relationships among tables; Table-Aligned Question Decomposer and Coverage-Aware Retriever, which jointly improve retrieval by enhancing the quality of decomposition and maximizing the coverage; Sub-question Guided Reasoner improves the answer quality using guided LLMs to generate an accurate executable program based on decomposed sub-questions. Experiments using two prepared datasets demonstrate that DMRAL significantly outperforms existing state-of-the-art MTQA methods, achieving an average improvement of 24% in table retrieval and 55% in answer accuracy.

IX. AI-Generated Content Acknowledgement

The authors used the AI system solely for grammatical correction and writing refinement. No AI tools were employed in data analysis, experimentation, or the formulation of conclusions. We acknowledge the assistance of AI in enhancing the writing process while maintaining full academic integrity.

References

[1] J. Wu, L. Yang, D. Li, Y. Ji, M. Okumura, and Y. Zhang, "Mmqa: Evaluating llms with multi-table multi-hop complex questions," in *The Thirteenth International Conference on Learning Representations*, 2025, p. 1.

[2] Z. Qiu, Y. Peng, G. He, B. Yuan, and C. Wang, "Tqa-bench: Evaluating llms for multi-table question answering with scalable context and symbolic extension," *arXiv preprint arXiv:2411.19504*, 2024.

[3] V. Pal, A. Yates, E. Kanoulas, and M. de Rijke, "Multitabqa: 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)*, 2023, pp. 6322–6334.

[4] R. Nararatwong, C.-C. Chen, N. Kertkeidkachorn, H. Takamura, and R. Ichise, "Dbqr-qa: A question answering dataset on a hybrid of database querying and reasoning," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 15 169–15 182.

[5] Q. Wang and R. C. Fernandez, "Solo: Data discovery using natural language questions via A self-supervised approach," *Proc. ACM Manag. Data*, vol. 1, no. 4, pp. 262:1–262:27, 2023.

[6] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller, and M. Riedewald, "Santos: Relationship-based semantic table union search," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–25, 2023.

[7] G. Fan, J. Wang, Y. Li, D. Zhang, and R. J. Miller, "Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning," *Proceedings of the VLDB Endowment*, vol. 16, no. 7, pp. 1726–1739, 2023.

[8] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto, and M. Oyamada, "Deepjoin: Joinable table discovery with pre-trained language models," *Proceedings of the VLDB Endowment*, vol. 16, no. 10, pp. 2458–2470, 2023.

[9] B. Li, J. Zhang, J. Fan, Y. Xu, C. Chen, N. Tang, and Y. Luo, "Alpha-sql: Zero-shot text-to-sql using monte carlo tree search," *arXiv preprint arXiv:2502.17248*, 2025.

[10] A. Mohammadjafari, A. S. Maida, and R. Gottumukkala, "From natural language to sql: Review of llm-based text-to-sql systems," *arXiv preprint arXiv:2410.01066*, 2024.

[11] P. B. Chen, F. Wenz, Y. Zhang, D. Yang, J. Choi, N. Tatbul, M. Cafarella, Ç. Demiralp, and M. Stonebraker, "Beaver: an enterprise benchmark for text-to-sql," *arXiv preprint arXiv:2409.02038*, 2024.

[12] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Zhang, J. Fan, G. Li, N. Tang, and Y. Luo, "A survey of text-to-sql in the era of llms: Where are we, and where are we going?" *IEEE Transactions on Knowledge & Data Engineering*, no. 01, pp. 1–20, 2025.

[13] J. Herzig, T. Mueller, S. Krichene, and J. Eisenschlos, "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*, 2021, pp. 512–519.

[14] F. Pan, M. Canim, M. Glass, A. Gliozzo, and J. Hendler, "End-to-end table question answering via retrieval-augmented generation," *arXiv preprint arXiv:2203.16714*, 2022.

[15] P. B. Chen, Y. Zhang, and D. 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)*, 2024, pp. 2687–2699.

[16] J.-P. Zhu, P. Cai, K. Xu, L. Li, Y. Sun, S. Zhou, H. Su, L. Tang, and Q. Liu, "Autotqa: Towards autonomous tabular question answering through multi-agent large language models," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 3920–3933, 2024.

[17] J. Shen, C. Wan, R. Qiao, J. Zou, H. Xu, Y. Shao, Y. Zhang, W. Miao, and G. Pu, "A study of in-context-learning-based text-to-sql errors," *CoRR*, vol. abs/2501.09310, 2025.

[18] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *EMNLP*. Association for Computational Linguistics, 2018, pp. 3911–3921.

[19] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Cao, R. Geng *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *arXiv preprint arXiv:2305.03111*, 2023.

[20] Y. Deng, C. Chai, L. Cao, Q. Yuan, S. Chen, Y. Yu, Z. Sun, J. Wang, J. Li, Z. Cao *et al.*, "Lakebench: A benchmark for discovering joinable and unionable tables in data lakes," *Proceedings of the VLDB Endowment*, vol. 17, no. 8, pp. 1925–1938, 2024.

[21] F. Li and H. V. Jagadish, "Nalir: an interactive natural language interface for querying relational databases," in *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, C. E. Dyreson, F. Li, and M. T. Özsu, Eds. ACM, 2014, pp. 709–712.

[22] H. Fu, C. Liu, B. Wu, F. Li, J. Tan, and J. Sun, "Catsql: Towards real world natural language to SQL applications," *Proc. VLDB Endow.*, vol. 16, no. 6, pp. 1534–1547, 2023.

[23] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, "Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task," *arXiv preprint arXiv:1810.05237*, 2018.

[24] C. Xiao, M. Dymetman, and C. Gardent, "Sequence-based structured prediction for semantic parsing," in *Annual meeting of the Association for Computational Linguistics (ACL)*, 2016, pp. 1341–1350.

[25] P. Yin, G. Neubig, W. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds. Association for Computational Linguistics, 2020, pp. 8413–8426.

[26] B. Bogin, J. Berant, and M. Gardner, "Representing schema structure with graph neural networks for text-to-sql parsing," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, A. Korhonen, D. R. Traum, and L. Màrquez, Eds. Association for Computational Linguistics, 2019, pp. 4560–4565.

[27] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, pp. 36 339–36 348, 2023.

[28] X. Xie, G. Xu, L. Zhao, and R. Guo, "Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–24, 2025.

[29] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, "The dawn of natural language to SQL: are we fully ready? [experiment, analysis & benchmark]," *Proc. VLDB Endow.*, vol. 17, no. 11, pp. 3318–3331, 2024.

[30] K. Chen, Y. Chen, N. Koudas, and X. Yu, "Reliable text-to-sql with adaptive abstention," *Proc. ACM Manag. Data*, vol. 3, no. 1, pp. 69:1–69:30, 2025.

[31] H. Zhang, R. Cao, L. Chen, H. Xu, and K. Yu, "ACT-SQL: in-context learning for text-to-sql with automatically-generated chain-of-thought," in *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 3501–3532.

[32] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "Chess: Contextual harnessing for efficient sql synthesis," *arXiv preprint arXiv:2405.16755*, 2024.

[33] J. Fürst, C. Kosten, F. Nooralahzadeh, Y. Zhang, and K. Stockinger, "Evaluating the data model robustness of text-to-sql systems based on real user queries," *arXiv preprint arXiv:2402.08349*, 2024.

[34] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "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*, 2020, pp. 7567–7578.

[35] X. Wu, J. Yang, L. Chai, G. Zhang, J. Liu, X. Du, D. Liang, D. Shu, X. Cheng, T. Sun *et al.*, "Tablebench: A comprehensive and complex benchmark for table question answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 24, 2025, pp. 25 497–25 506.

[36] J. Zhang, Z. Shen, B. Srinivasan, S. Wang, H. Rangwala, and G. Karypis, "Nameguess: Column name expansion for tabular data," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 13 276–13 290.

[37] H. Lee, S. Yang, H. Oh, and M. Seo, "Generative multi-hop retrieval," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022*. Association for Computational Linguistics, 2022, pp. 1417–1436.

[38] M. I. L. Balaka, D. Alexander, Q. Wang, Y. Gong, A. Krisnadhi, and R. 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*, vol. 3, no. 3, pp. 1–28, 2025.

[39] Z. Shen, C. Hu, and Z. Zhao, "Lynx: A graph query framework for multiple heterogeneous data sources," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3926–3929, 2023.

[40] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi, "Query processing under GLAV mappings for relational and graph databases," *Proc. VLDB Endow.*, vol. 6, no. 2, pp. 61–72, 2012.

[41] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables." in *SIGMOD Conference*, vol. 10, 2012, pp. 2 213 836–2 213 962.

[42] K. Zhang, J. Zeng, F. Meng, Y. Wang, S. Sun, L. Bai, H. Shen, and J. Zhou, "Tree-of-reasoning question decomposition for complex question answering with large language models," in *Proceedings of the AAAI Conference on artificial intelligence*, vol. 38, no. 17, 2024, pp. 19 560–19 568.

[43] Y. Peng, Q. Wang, L. Zhang, Y. Liu, and Z. Mao, "Chain-of-question: A progressive question decomposition approach for complex knowledge base question answering," in *Findings of the Association for Computational Linguistics, ACL 2024*. Association for Computational Linguistics, 2024, pp. 4763–4776.

[44] S. Min, V. Zhong, L. Zettlemoyer, and H. Hajishirzi, "Multi-hop reading comprehension through question decomposition and rescoring," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 6097–6109.

[45] X. Huang, S. Cheng, Y. Shu, Y. Bao, and Y. Qu, "Question decomposition tree for answering complex questions over knowledge bases," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, 2023, pp. 12 924–12 932.

[46] Z. Lan and S. Li, "Ps-sql: Phrase-based schema-linking with pre-trained language models for text-to-sql parsing," in *2024 6th International Conference on Natural Language Processing (ICNLP)*. IEEE, 2024, pp. 31–35.

[47] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 2318–2335.

[48] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, and M. Zaharia, "Colbertv2: Effective and efficient retrieval via lightweight late interaction," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 3715–3734.

[49] X. Zhang, D. Wang, L. Dou, Q. Zhu, and W. Che, "Murre: Multi-hop table retrieval with removal for open-domain text-to-sql," in *Proceedings of the 31st International Conference on Computational Linguistics*, 2025, pp. 5789–5806.

[50] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[51] A. Khatiwada, R. Shraga, and R. J. Miller, "Fuzzy integration of data lake tables," *arXiv preprint arXiv:2501.09211*, 2025.

[52] S. Kweon, Y. Kwon, S. Cho, Y. Jo, and E. Choi, "Open-wikitable: Dataset for open domain question answering with complex reasoning over table," in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 8285–8297.

[53] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller, "Organizing data lakes for navigation," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1939–1950.

[54] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "The secret life of wikipedia tables," in *Proceedings of the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA-Data 2021) co-located with 47th International Conference on Very Large Data Bases (VLDB 2021), Copenhagen, Denmark, August 20, 2021*, ser. CEUR Workshop Proceedings, vol. 2929. CEUR-WS.org, 2021, pp. 20–26.

[55] E. Zhu, Y. He, and S. Chaudhuri, "Auto-join: Joining tables by leveraging transformations," *Proceedings of the VLDB Endowment*, vol. 10, no. 10, pp. 1034–1045, 2017.

[56] P. Li, X. Cheng, X. Chu, Y. He, and S. Chaudhuri, "Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 1064–1076.

[57] X. Hu, C. Lei, X. Qin, A. Katsifodimos, C. Faloutsos, and H. Rangwala, "Polyjoin: Semantic multi-key joinable table search in data lakes," in *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025, pp. 384–395.

[58] S. E. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, no. 4, pp. 333–389, 2009.

[59] D. Wang, L. Dou, X. Zhang, Q. Zhu, and W. Che, "Enhancing numerical reasoning with the guidance of reliable reasoning processes," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 10 812–10 828.

[60] R. Zhu, X. Liu, Z. Sun, Y. Wang, and W. Hu, "Mitigating lost-in-retrieval problems in retrieval augmented multi-hop question answering," *arXiv preprint arXiv:2502.14245*, 2025.

[61] "Our source code and technical report," https://github.com/JrJessyLuo/multitab_qa_dmral/.

[62] Y. Sun, H. Xin, and L. Chen, "Reca: Related tables enhanced column semantic type annotation framework," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1319–1331, 2023.

[63] "Parallel computing," https://en.wikipedia.org/wiki/Parallel_computing.

[64] "Bird-sql leaderboard," https://bird-bench.github.io/.

[65] X. Zeng, P. Wang, Y. Mao, L. Chen, X. Liu, and Y. Gao, "Multiem: Efficient and effective unsupervised multi-table entity matching," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 3421–3434.

APPENDIX A
METADATA INFERENCE MODULE

As incomplete metadata (i.e., missing column headers) may hinder accurate unionability computation while constructing the graph, we introduce a *metadata inference* module that leverages LLMs to infer and complete the missing headers. A naive approach here would sample a few rows (e.g., 10) from the table as the whole context and feed them to an LLM to infer missing headers. However, this may include irrelevant or loosely correlated columns, which dilute the context and reduce effectiveness [65]. To mitigate this, our module adopts a targeted approach inspired by [62]. The core idea is to utilize intra-table column grouping to establish a concise and semantically coherent context for the LLM, which enhances inference by exclusively focusing the LLM on small, meaningful groups of related columns. Specifically, we first apply the column semantics discovery algorithm [6] to partition each table into semantically coherent column groups, based on the column values. For example, given a table with columns C = {Name, Street Address, City, Product ID, Price, Date of Sale}, the algorithm might partition them into two semantically coherent groups: {Name, Street Address, City} to represent the context of "Location", and {Product ID, Price, Date of Sale} under the context of "Transaction". Then, for any column with an incomplete header, we identify its corresponding column group. We then extract only the columns belonging to this group, along with their sampled rows, to construct a subtable. The LLM is subsequently prompted for the header inference using only this highly relevant sub-context (i.e., sampled rows from the partitioned subtable). The metadata inference prompt is shown in Figure 9.

APPENDIX B
PROMPTS

In this section, we supplement the main prompts used in this paper. The question decomposition prompt (§III-B4) is shown in Figure 10. The residual sub-question generation prompt (§IV-B) is shown in Figure 12. The prompts for multi-step program generation and execution-guided refinement (§V-A) are shown in Figure 13 and Figure 14, respectively. The prompt used to prepare our evaluation datasets by grouping non-key columns into column subsets (§VI-B) is shown in Figure 11.

APPENDIX C
CASE ANALYSIS

We conduct a qualitative comparison between the subquestions produced by the *Direct LLM* approach and the table-aligned question decomposer in DMRAL. Table X presents representative examples highlighting the major decomposition issues commonly observed with the *Direct LLM* approach, alongside the improvements achieved by DMRAL. We categorize these issues into three types:

Missing Key Information: Critical elements required to fully specify the question are omitted. For example, in the first case, the *Direct LLM* decomposition misses the mention of *"account opened"*, which is essential to correctly identify the relevant table `account`. In contrast, DMRAL explicitly retains this information in the second sub-question (i.e., better completeness).

Redundant Decomposition: The decomposition produces overlapping or repetitive sub-questions. In the second case, both sub-questions generated by *Direct LLM* redundantly reference *"superheroes"*, resulting in duplicate query intent for table *superhero*. In contrast, DMRAL avoids such redundancy (i.e., better non-redundancy).

Entangled Sub-question: The decomposition produces a sub-question that entangles multiple distinct information needs into a single sub-question, increasing the answering complexity. In the third case, *Direct LLM* merges *"segment SME"* and *"year 2013"* into a single sub-question, even though both elements are less likely to co-occur in the same context. In contrast, DMRAL separates them into two sub-questions (i.e., better table-specificity).

You are an expert in metadata inference for table-based data understanding.

Task: You are provided with a table containing partially missing metadata, such as incomplete or masked table titles and column headers. Your goal is to infer and recover the missing metadata based on the provided table structure and inter-table context (i.e., sampled rows).

Important Instructions:
– You must return exactly the same number of column headers as provided in the "Column Headers".
– Do not add, remove, or reorder any columns—only replace missing ones with your best inference.
– Maintain the original header order.

Response Format:

```
{
  "updated_title": "...",
  "updated_headers": ["...", "...", ...]
}
```

Now perform the metadata inference on the provided table:

– [Table Title] {`table_title`}
– [Column Headers] {`headers`}
– [Sample Rows (10 randomly sampled rows)] {`sampled_rows`}

Fig. 9: Metadata Inference Prompt.

You are an expert in multi-hop question decomposition for table-based question answering.
Task: Decompose a complex question into a sequence of simpler, minimal sub-questions. Each sub-question must satisfy the following guidelines:
– Use the key phrases grouped together in each entry from a provided list called "information needs".
– Preserve the full meaning and intent of the original question without omitting important details.
– Ensure the sub-questions are minimal, non-redundant, and natural.
Response Format:

```
{"Sub-questions": [...]}
```

Example: [Question] Among the schools with the average Math score over 560 in the SAT test, how many schools are directly charter-funded?
[Information Needs] `[['schools', 'SAT test', 'charter funded'], ['Math score over 560']]`
Output:

```
{"Sub-questions": [ "Which schools have an
average Math score over 560?",
"How many of the schools from #1 are
directly charter-funded in the SAT test?"]}
```

Now decompose the following question:
– [Question] {`question`}
– [Information Needs] {`table-aligned groups`}

Fig. 10: Question Decomposition Prompt.

Issue	Question	*Direct LLM*	DMRAL
Missing Key Information	*Among the account opened, how many female customers born before 1950 and stayed in Sokolov?*	SubQ1: Which female customers were born before 1950? SubQ2: How many of #1 stayed in Sokolov?	SubQ1: Which female customers were born before 1950 and stayed in Sokolov? SubQ2: How many of #1 opened an account?
Redundant Decomposition	*How many superheroes have brown eyes?*	SubQ1: Which superheroes have brown eyes? SubQ2: How many of the superheroes from #1?	SubQ1: Which are brown eyes? SubQ2: How many superheroes have #1?
Entangled Sub-question	*What was the average monthly consumption of customers in segment SME for the year 2013?*	SubQ1: Which customers are in segment SME for the year 2013? SubQ2: What was the average monthly consumption for #1?	SubQ1: Which customers are in segment SME? SubQ2: What was the average monthly consumption for #1 in year 2013?

TABLE X: Examples of decomposition issues observed in *Direct LLM* vs. DMRAL.

You are an expert in organizing tables by grouping related columns into smaller, meaningful subtables.

Task: You are provided with a cluttered table and asked to reorganize its columns into subtables.

Reorganization Guidelines:
– Group columns that naturally belong together.
– Do not rename, remove, or reorder the columns within each group.

Response Format:

```
{
  "Tables": [
    {
      "Table Title": "...",
      "Column Headers": ["...", "...", ...]
    },
    ...
  ]
}
```

Now perform the table reorganization based on the provided table context.

– Table Title: {`table_title`}
– Column Headers: {`column_headers`}

Fig. 11: Semantic Column Grouping Prompt.

You are an expert in identifying semantic gaps in query decomposition.

Task: Identify any missing tables needed to fully answer the question as a residual sub-question, given a user question and a set of currently available tables. If all required tables are already provided, return `None`.

Response Format:

```
{"Residual Sub-question": ...}
```

Examples:

[Question] How many female clients opened their accounts in Jesenik branch?
[Provided Tables] `financial.client(client_id, gender, birth date, district_id)`
`financial.disp(disposition_id, client_id, account_id, type)`
Output:

```
{"Residual Sub-question": "What is the
district name of
the Jesenik branch?"}
```

[Question] Among the atoms that contain element carbon, which one does not contain compound carcinogenic?
[Provided Tables] `toxicology.atom(atom_id, molecule_id, element)`
`toxicology.molecule(molecule_id, label)`
Output:

```
{"Residual Sub-question": None}
```

Now perform the task on the following input:

– [Question] {`question`}
– [Provided Tables] {`tables`}

Fig. 12: Residual Sub-question Generation Prompt.

You are an expert in SQL program synthesis for multi-table question answering over structured tabular data.

Task: Your objective is to reason step by step to generate the correct SQL program. You are provided with:
– A natural language question.
– A set of retrieved tables with metadata.
– A set of decomposed sub-questions (which may be inaccurate or incomplete).
– Optional external knowledge (e.g., mappings between question phrases and columns).

Reasoning Process:
– **Step 1:** Carefully read the question and table metadata to understand the full requirements.
– **Step 2:** Evaluate the decomposed sub-questions. Revise or rewrite them if needed.
– **Step 3:** For each revised sub-question, reason step by step to write the corresponding SQL query.
– **Step 4:** After processing all sub-questions, check if the final sub-question's SQL fully answers the original question.

Response Format:

```
{
  "reasoning": "...",
  "Final SQL": "..."
}
```

Example:

[Question] List the names of female students who have enrolled in more than 3 courses.
[Retrieved Tables] `enrollment(enrollment_id, student_id, course_id) student(student_id, name, age, gender)`
[External Knowledge] ”female” refers to gender = 'F'
[Decomposed Sub-questions] `['Which students have enrolled in more than 3 courses?', 'What are the names of female students from #1?']`
Output:

```
{
  "reasoning": "Let's think step by step.
  Step 1: Revised sub-question 1:
  Which students have enrolled in more than 3 courses?
  SQL: SELECT student_id FROM enrollment GROUP BY student_id HAVING COUNT(course_id) > 3.

  Step 2: Revised sub-question 2: What are the names of female students from #1?
  SQL: SELECT DISTINCT name FROM student WHERE student_id IN (...) AND gender = 'F'."

  "Final SQL": "SELECT DISTINCT name FROM student WHERE student_id IN (SELECT student_id
  FROM enrollment GROUP BY student_id HAVING COUNT(course_id) > 3) AND gender = 'F'"
}
```

Now perform the reasoning and SQL program generation on the provided input:

– [Question] {`question`}
– [Retrieved Tables] {`tables`}
– [External Knowledge] {`external_knowledge`}
– [Decomposed Sub-questions] {`decomposed sub-questions`}

Fig. 13: Chain-of-Thought Guided Program Generation Prompt.

You are an expert SQL correction assistant for multi-table question answering over structured tabular data.

Task: Given a natural language question, a set of related tables with metadata, a faulty SQL query that failed with a SQL execution error, and optional external knowledge (e.g., mappings between question phrases and columns), your task is to correct the SQL query by fixing the identified error.

Response Format:

```
{
  "SQL": "..."
}
```

Now perform the SQL correction based on the provided context:

– [Question] {`question`}
– [Tables] {`tables`}
– [Error SQL] {`sql`}
– [SQL Error] {`sql_error`}
– [External Knowledge] {`external_knowledge`}

Fig. 14: Program Refinement Prompt.