

Agentic Open-Domain Tabular Question Answering with Self-Supervised Multi-Table Retriever

Jianwei Wang¹, Mengqi Wang¹, Qing Liu², Xiwei Xu²,
Zhenchang Xing², Liming Zhu², Wenjie Zhang^{1,*}

¹University of New South Wales, Australia

²CSIRO, Australia

{jianwei.wang1, mengqi.wang4, wenjie.zhang}@unsw.edu.au

{q.liu, xiwei.xu, zhenchang.xing, liming.zhu}@csiro.au

Abstract

Open-domain tabular question answering (TQA) aims to answer natural language queries over a data lake that contains a large corpus of heterogeneous tables. Unlike closed-domain TQA, this task operates without pre-provided tables, making it particularly challenging. Existing methods usually first retrieve relevant tables and then answer the question based on the retrieved tables. However, they face challenges in effectively and efficiently preparing tables from the low-quality, heterogeneous data lake and in operationalizing table information. To address these limitations, we present ArrTab (Agentic retrieval and reasoning on Tables) for open-domain TQA. ArrTab leverages MuRe, a self-supervised Multi-table Retriever trained on questions synthesized by translating table-derived SQL queries into natural language via large language models. Based on MuRe, ArrTab proposes a multi-agent system for open-domain TQA that consists of the Table Retriever Agent, the Table Integration Agent, the Information Extraction Agent and the Answer Agent. Furthermore, to enhance accuracy, a lazy execution strategy is adopted where operations are described in natural language but executed only during the information extraction stage, allowing for globally consistent reasoning. Extensive experiments on real-world datasets demonstrate the superiority of ArrTab.

1 Introduction

Tabular Question Answering (TQA) (Zhang et al., 2024; Fan et al., 2025) has been extensively studied in real-world scenarios. Given a query and a structured table, the goal is to generate a precise answer by reasoning over the table contents. However, conventional TQA methods are typically closed-domain, assuming the relevant tables are already provided, which limits their practical applicability. In many real-world applications, such as web

search or virtual assistants, the appropriate table is not readily available and must first be retrieved from a large collection (Cafarella et al., 2008; Jin et al., 2022). To address this limitation, open-domain tabular question answering (open-domain TQA) (Herzig et al., 2021; Gu et al., 2022) has recently emerged. This task aims to answer natural language queries over a table data lake containing a large number of heterogeneous tables.

Since directly feeding the entire data lake into a model is infeasible, existing open-domain TQA methods typically adopt a retrieve-then-reason paradigm: they first employ a retriever to locate relevant tables and then leverage language models, such as Large Language Models (LLMs) (Wang et al., 2025, 2026), to infer the answer. In the early stage, methods (e.g., (Herzig et al., 2021; Kang et al., 2024)) usually first retrieve relevant tables using methods such as dense retrieval techniques, and then employ small language models, such as TAPAS (Herzig et al., 2020), to generate answers. More recently, LLM-based approaches have been widely adopted, leveraging either SQL-based query generation (Kong et al., 2024) or tool-integrated reasoning (Qiu, 2025) to enhance performance.

Despite these advancements, two key limitations remain. First, existing open-domain TQA systems typically rely on either label-intensive dense retrieval (Herzig et al., 2021) or inaccurate sparse retrieval (Kong et al., 2024). While dense retrieval embeds questions and tables into a latent space to capture semantic similarity, it requires labeled data for training and primarily focuses on single-table retrieval. On the other hand, the inaccurate sparse retriever usually relies on keyword matching (e.g., BM25 (Kong et al., 2024)), failing to capture deep semantics. Second, current methods usually operate in sequential stages, where errors from table preprocessing cascade and accumulate downstream. This leads to low accuracy, particularly in open-domain settings characterized by

*Corresponding Author.

noisy retrieval and heterogeneous table structures.

To overcome these limitations, we propose ArrTab, an **A**gentic **r**etrieval and **r**easoning framework on **T**ables for open-domain TQA. It features MuRe, a self-supervised **M**ulti-table **R**etriever designed to accurately discover relevant tables. Building upon MuRe, a multi-agent system is established to collaboratively operationalize table information and perform globally consistent reasoning via a lazy execution strategy.

MuRe establishes a self-supervised learning paradigm that constructs multi-table training samples directly from the data lake without the use of labels. To improve the quality of the synthesized training samples, MuRe adopts SQL as an intermediate representation and introduces a table-to-SQL-to-question pipeline. It first identifies value-level and schema-level overlaps in raw tables to automatically generate executable multi-table SQL queries, which are then translated into natural-language questions using LLMs. Furthermore, MuRe adopts a ranking-based relevance learning to retrieve coherent table sets that collectively support the query, thus enhancing its ability to solve complex, multi-table reasoning tasks in the open-domain settings.

Based on MuRe, a multi-agent system is built to enable end-to-end reasoning over retrieved tables. Specifically, ArrTab consists of four specialized agents: the Table Retriever Agent, the Table Integration Agent, the Information Extraction Agent, and the Answer Agent. These agents collaborate through a shared state and collectively decompose open-domain TQA into retrieval, planning, extraction, and synthesis stages. Following a lazy execution strategy, the Table Retriever Agent identifies a relevant set of tables, while the Table Integration Agent formulates high-level multi-table reasoning plans without materializing intermediate results. The Information Extraction Agent then translates these abstract plans into executable SQL to extract precise evidence, and the Answer Agent synthesizes and validates the final response. This enables efficient operation while ensuring globally consistent reasoning across multi-table evidence.

The main contributions are summarized as:

- First, we propose ArrTab, an agentic retrieval and reasoning framework for open-domain tabular question answering.
- Second, we introduce MuRe, a self-supervised multi-table retriever that first synthesizes training data from raw tables via a table-to-SQL-

to-question pipeline and then trains a relevance model by employing a ranking-based relevance objective to accurately retrieve coherent table sets without manual annotation.

- Third, we develop a multi-agent system with a lazy execution strategy, enabling efficient and globally consistent multi-table reasoning through coordinated retrieval, planning, extraction, and answer synthesis.
- Finally, extensive experiments on real-world datasets demonstrate that ArrTab significantly outperforms existing TQA systems. Compared to current open-domain TQA methods, ArrTab achieves an average performance improvement of 12.46%. It also delivers competitive results compared to closed-domain TQA baselines where the grounding tables are provided.

2 Preliminaries

We consider a tabular data lake denoted as $\mathcal{D} = \{T_1, T_2, \dots, T_N\}$, where each table $T_i = (C_i, R_i)$ consists of a set of columns $C_i = \{c_{i1}, \dots, c_{im_i}\}$ and a set of rows $R_i = \{r_{i1}, \dots, r_{in_i}\}$. Each cell x_{ijz} represents the value in row r_{ij} and column c_{iz} . We use $\mathcal{Q}$ to denote the space of natural language questions and $\mathcal{A}$ to denote the space of the answers.

Definition 1 (Open-domain Tabular Question Answering) (Herzig et al., 2021; Kang et al., 2024). Given a question ($q \in \mathcal{Q}$) and a large tabular data lake ($\mathcal{D} = \{T_1, \dots, T_N\}$), the goal of open-domain tabular question answering is to infer the correct answer ($a^* \in \mathcal{A}$), such that the answer is verifiably grounded in and factually consistent with the evidence contained in $\mathcal{D}$.

Due to the large number of tables in the data lake, feeding all tables into the model is infeasible (Chen et al., 2026). Therefore, typical open-domain TQA systems first retrieve a relevant subset of tables ($\mathcal{T}_q \subseteq \mathcal{D}$), and then reason over them to produce the correct answer (a^*). An open-domain TQA system can be represented as a composition of two functions: $a^* = f_{\text{ans}}(f_{\text{ret}}(q, \mathcal{D}), q)$, where (f_{ret}) denotes the table retrieval module that selects potentially relevant tables, and (f_{ans}) denotes the reasoning or answer generation module that derives the final answer from the retrieved tables.

Remark 2.1: (Open-domain TQA vs. Closed-domain TQA). The distinction between open-domain TQA and closed-domain TQA lies in the grounding tables. In closed-domain tabular QA,

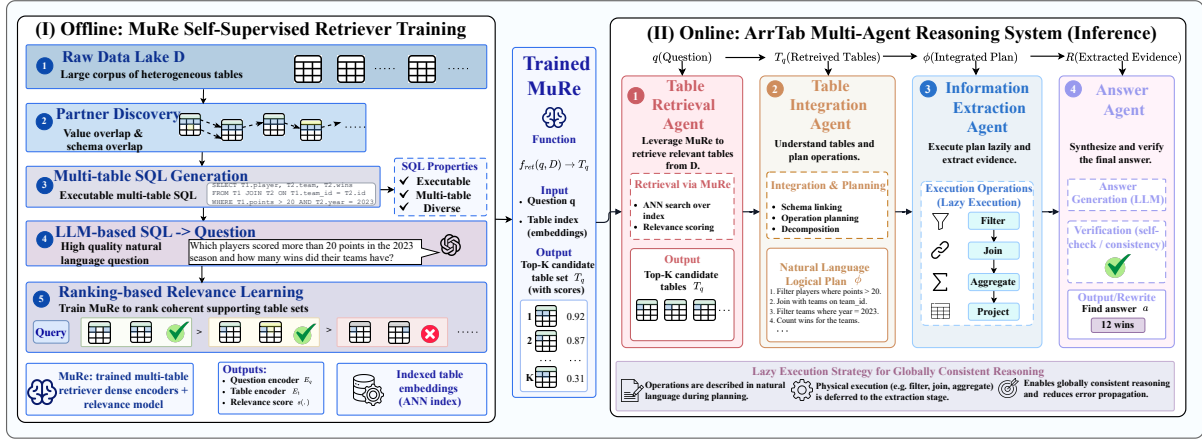


Figure 1: System architecture of ArrTab.

the question is answered with respect to given tables, assuming that the relevant data are known. In contrast, open-domain QA operates over a large and diverse tabular data lake, where relevant tables need first to be retrieved from numerous heterogeneous candidates before reasoning can be applied.

3 Overview of ArrTab

As illustrated in Figure 1, ArrTab integrates a self-supervised retriever training pipeline with a collaborative multi-agent reasoning framework. To accurately train a multi-table retriever without labels, an LLM-based Data Synthesis module first discovers table relationships through *Partner Discovery* and then generates multi-table SQL queries paired with natural language questions, which are used to optimize the retriever by *Ranking-based Relevance Learning*. Leveraging this retriever, open-domain queries are processed through a coordinated workflow of specialized agents. First, the *Table Retrieval Agent* identifies evidence tables. Then, the *Table Integration Agent* unifies them, the *Information Extraction Agent* extracts useful data, and the *Answer Agent* generates the final answer. A lazy execution strategy is used to defer the execution of logical integration plans until the extraction stage.

4 MuRe Retriever

4.1 LLM-based Data Synthesis

We first construct a self-supervised training corpus comprising natural language questions paired with ground-truth evidence sets of tables. Formally, a training example is a pair $(q, \mathcal{T}^*)$ where $\mathcal{T}^* = \{T_A, T_B, \dots\}$ denotes the minimal set of tables required to answer q . Our goal is to synthesize

such pairs directly from an unlabeled data lake $\mathcal{D}$, without manual annotation. “self-supervised” means that supervision tuples $(q, \mathcal{T}^*)$ are automatically generated from the unlabeled data lake without human annotation.

(1) Partner discovery via content overlap. MuRe needs to identify *partners* for a given anchor table, so that the resulting question logically requires combining evidence from multiple tables.

Given the data lake $\mathcal{D}$, MuRe first sample an anchor table $T_A \sim \mathcal{U}(\mathcal{D})$ and select a representative column $c_A \in T_A$. Instead of traversing an explicit schema graph, which is often unavailable in open data lakes, MuRe discovers partner tables heuristically. For each candidate T_B , MuRe computes two overlap metrics that capture value-level and schema-level relatedness. Let $V(c)$ be the set of cell values in a column c , and let $H(T)$ denote the set of column headers of table T . MuRe defines the *Value Overlap* $\mathcal{J}_{\text{val}}$ and *Schema Overlap* $\mathcal{J}_{\text{sch}}$ between T_A and T_B as

$$\mathcal{J}_{\text{val}}(T_A, T_B) = \max_{c_i \in T_A, c_j \in T_B} \frac{|V(c_i) \cap V(c_j)|}{|V(c_i) \cup V(c_j)|}, \quad (1)$$

$$\mathcal{J}_{\text{sch}}(T_A, T_B) = \frac{|H(T_A) \cap H(T_B)|}{|H(T_A) \cup H(T_B)|}. \quad (2)$$

MuRe then select as partner T_B the table that maximizes a composite score $\lambda \mathcal{J}_{\text{val}} + (1 - \lambda) \mathcal{J}_{\text{sch}}$, where λ controls the relative emphasis on shared values versus shared schema. Hyperparameters are evaluated in Section 6.4.

(2) Adaptive multi-table SQL generation. Once a pair (T_A, T_B) has been selected, MuRe generates a structured SQL query $\mathcal{S}$ whose evaluation requires simultaneous access to both tables. Instead of directly generating questions, MuRe uses

SQL as a latent semantic representation that is easier to translate to natural language than the tables themselves.

For entity overlap, when $\mathcal{J}_{\text{val}}(T_A, T_B) > \mathcal{J}_{\text{sch}}(T_A, T_B)$, MuRe instantiates a join-based template that links tuples across tables:

```
SELECT agg( $T_B$ .col) FROM  $T_A$  JOIN  $T_B$ 
ON  $T_A$ .key =  $T_B$ .key WHERE  $T_A$ .cond,
```

where agg is an aggregation operator such as MAX, MIN, or COUNT, and key and cond are instantiated from sampled rows and predicates in T_A and T_B . This template encourages questions that, for example, ask for the maximum value of an attribute in T_B conditioned on an entity filter in T_A .

For schema overlap, when $\mathcal{J}_{\text{sch}}(T_A, T_B) > \mathcal{J}_{\text{val}}(T_A, T_B)$, MuRe employs a union-based template that aggregates information coming from disjoint subsets of two structurally analogous tables:

```
SELECT agg(col) FROM  $T_A$  WHERE cond
UNION SELECT agg(col) FROM  $T_B$  WHERE cond.
```

(3) Question translation. To bridge the gap between structured logical forms and natural language, MuRe leverages the zero-shot capabilities of LLM. Instead of fine-tuning a smaller language model, MuRe employs an LLM with a small size as the translation module. We choose Gemma-2-9b-it because it offers a favorable quality-cost trade-off for SQL-to-question translation, allowing us to synthesize training supervision in batch while maintaining adequate semantic fidelity (see experiments in Sec 6.4). The prompt template is in Sec A.1.

Given a multi-table SQL query $\mathcal{S}$ and the prompt instructions $\mathcal{P}$, the model generates the corresponding question: $q = \text{LLM}(\mathcal{S}, \mathcal{P})$. The resulting training sample is the pair $(q, \mathcal{T}^*)$, where $\mathcal{T}^*$ is the set of tables involved in the SQL.

4.2 Ranking-based Relevance Learning

In multi-table retrieval, individual tables often provide *partial* evidence. If one enforces a single positive label per question or treats retrieval as independent binary classification, the model may suffer from overfitting or catastrophic forgetting of the pre-trained semantic space (Sun et al., 2022; Chen et al., 2024b). To remedy this, we cast relevance learning as a *ranking task*. Instead of forcing absolute probability targets, we optimize the relative ordering between relevant and irrelevant tables. Let $(q, \mathcal{T}^*)$ be a training instance, and let $\mathcal{C}(q) \subseteq \mathcal{D}$ be

the candidate set retrieved by the first stage. We define the set of hard negatives as $\mathcal{N}(q) = \mathcal{C}(q) \setminus \mathcal{T}^*$.

(1) Architecture. MuRe adopts a dense bi-encoder architecture following SOLO (Wang and Castro Fernandez, 2023). It consists of two Transformer encoders (Reimers and Gurevych, 2019): a question encoder $E_q(\cdot)$ and a table encoder $E_t(\cdot)$, which independently encode the input question q and candidate table T into a shared dense embedding space: $h_q = E_q(q) \in \mathbb{R}^d$, $h_T = E_t(T) \in \mathbb{R}^d$.

For table encoding, each table is first transformed into a set of semantic triples using the Row-Wise Complete Graph serialization strategy in SOLO. The serialized triples are then linearized into textual sequences and fed into the Transformer encoder to capture schema semantics and row-wise relational information. The relevance score between q and T is computed using cosine similarity:

$$s(q, T) = \langle h_q, h_T \rangle. \quad (3)$$

(2) Objective. To train the model, we minimize the pairwise margin ranking loss, enabling fine-tuning of the decision boundary for hard candidates while preserving the general capability of the pre-trained embeddings. Specifically, for each question q , we construct training pairs by taking the Cartesian product of the positive set $\mathcal{T}^*$ and the negative set $\mathcal{N}(q)$. The objective dictates that the score of a relevant table T^+ must exceed that of an irrelevant table T^- by a margin γ :

$$\mathcal{L} = \frac{1}{|\mathcal{T}^*||\mathcal{N}(q)|} \sum_{\substack{T^+ \in \mathcal{T}^* \\ T^- \in \mathcal{N}(q)}} \max(0, \beta - (s(q, T^+) - s(q, T^-))), \quad (4)$$

where β is a hyperparameter controlling the separation between positive and negative scores.

(3) Index construction and inference. The resulting dense table embeddings are stored in a compressed index using for efficient retrieval (Douze et al., 2025; Wang et al., 2024a). At inference time, given a question q , we encode it using the student encoder and perform an Approximate Nearest Neighbor (ANN) search directly over the table index. This yields a ranked list of top- K tables $\mathcal{T} = \{T_1, \dots, T_K\}$. The relevance score is defined as: $S(q, T) = \langle \mathbf{v}_q, \mathbf{v}_T \rangle$, where $\mathbf{v}_q$ and $\mathbf{v}_T$ are the dense vectors of the question and table, respectively. The retrieved set $\widehat{\mathcal{T}}(q)$ is then sent to the multi-agent system for evidence synthesis and generates the final answers.

5 The Multi-Agent TQA Framework

Build upon MuRe retriever, a multi-agent TQA framework is proposed that processes user queries through four agents: Table Retrieval Agent, Table Integration Agent, Information Extraction Agent, and Answer Agent. These agents collaborate through a shared state context, employing a *Lazy Execution* strategy to minimize overhead and error propagation when generating the final answer. Detailed prompts and communication mechanism are provided in Sec. A.

5.1 Table Retrieval Agent

Given a large and heterogeneous data lake in open-domain TQA, the goal of the Table Retrieval Agent is to efficiently narrow the search space and identify a small set of relevant candidate tables. Leveraging the vector index constructed during the offline stage, the agent encodes the natural language query q into the same semantic embedding space as tables, and retrieves the top- K tables according to the cosine similarity score $\langle q, T \rangle$. Formally, the retrieved candidate set is defined as

$$\mathcal{T}_q = \text{TopK}_{T \in \mathcal{D}} \langle q, T \rangle,$$

where $\mathcal{D}$ denotes the table corpus.

5.2 Integration Agent with Lazy Execution

In open-domain settings, answering a question typically requires integrating evidence from multiple tables. However, the retrieved tables may contain substantial noise, and the relationships among them are often ambiguous or weakly connected, making reliable multi-table reasoning particularly challenging. The Table Integration Agent addresses this by operating as a strategic logic planner. Instead of directly processing tables, it adopts a *Lazy Execution* strategy, analyzing the schemas of the candidate tables $\mathcal{T}_q$ to formulate an abstract integration plan Φ . Specifically, the agent utilizes an LLM to predict the optimal relational operation type and to identify semantic correspondences between columns. Formally, the planning process is defined as:

$$\Phi = (op, \mathcal{T}_{sel}, \mathcal{M}_{col}) = \text{LLM}_{\text{Integrator}}(q, \mathcal{T}_q) \quad (5)$$

Here, $op \in \{\text{JOIN}, \text{UNION}, \text{NONE}\}$ denotes the operation, which captures the most common types of inter-table relationships. $\mathcal{T}_{sel} \subseteq \mathcal{T}_q$ represents the subset of tables selected as relevant to the query, and $\mathcal{M}_{col}$ denotes the set of semantically aligned column pairs across tables.

Instead of directly executing table operations, the agent first constructs an abstract integration plan over table schemas. Such lazy execution provides two key advantages: (1) it avoids premature evidence composition that may amplify retrieval and schema alignment errors during physical table integration; and (2) it enables semantic-level filtering of irrelevant or weakly related tables before execution, thereby reducing the downstream reasoning space and improving answer reliability.

5.3 Information Extraction Agent with Self-reflective Text-to-SQL

The Information Extraction Agent is designed to bridge the gap between natural language planning and symbolic execution. It operationalizes the abstract integration plan Φ and the user query q by synthesizing them into a concrete, executable SQL statement Y_{SQL} . While focused on SQL, the framework is extensible to other languages like Python for complex tasks beyond relational algebra.

Leveraging the strong code generation capabilities of LLMs (Zhu et al., 2024b; Zhan et al., 2025), we further introduce a *Self-reflective Text-to-SQL mechanism* to improve execution reliability. Specifically, the generated SQL query is executed against an in-memory database constructed from the selected tables $\mathcal{T}_{sel}$. If execution fails due to syntax or schema errors, the execution feedback $E^{(t)}$ is used to iteratively refine the query:

$$Y_{SQL}^{(t+1)} = \text{LLM}_{\text{Text2SQL}}(q, \Phi, Y_{SQL}^{(t)}, E^{(t)}) \quad (6)$$

This execution-grounded loop enables the agent to progressively correct erroneous SQL and align the generated query with both the schema constraints and the semantic intent of the integration plan.

5.4 Answering Agent with Evidence Sufficiency Verification

The Answering Agent acts as the final reasoning module, synthesizing a natural language answer based on the query q and the retrieved evidence R .

A critical limitation of conventional single-pass pipelines is their vulnerability to retrieval failures, which frequently lead to unanswerable queries or hallucinations. To alleviate this issue, we introduce a *Evidence Sufficiency Verification* mechanism that explicitly evaluates the sufficiency of R with respect to q before generating the final answers. Specifically, the LLM first evaluates whether the retrieved evidence sufficiently supports answering

the query. If the evidence is deemed sufficient, the final answer is generated as $a = \text{LLM}(q, R)$. Otherwise, instead of forcing generation under missing evidence, the agent identifies unresolved information gaps and reformulates the query:

$$q' = \text{LLM}_{\text{Rewrite}}(q, R_{\text{gap}}). \quad (7)$$

The reformulated query q' is then routed back to the *Table Retrieval Agent* (Section 5.1) for another retrieval iteration. This closed-loop refinement progressively improves evidence completeness and mitigates retrieval failures.

6 Evaluation

6.1 Experiment Setup

Datasets. Following prior works (Zhang et al., 2024; Zou et al., 2025), we adopt four widely used datasets to evaluate the performance of open-domain TQA, including WikiTables (Zou et al., 2025), RETQA (Wang et al., 2024b), FeTaQA (Nan et al., 2022), and TabFact (Chen et al.). WikiTables and RETQA are natively multi-table benchmarks, while FeTaQA and TabFact are originally single-table datasets but are included due to their wide usage. Following (Zou et al., 2025), the single-table datasets are partitioned by rows, columns, or kept intact to construct the final data lake. Detailed dataset statistics and preprocessing procedures are provided in Sec. B in the Appendix.

Baselines. We consider two categories of baselines for a comprehensive evaluation: open-domain TQA and closed-domain TQA. For open-domain TQA, we include three representative methods: (1) DTR (Herzig et al., 2021), (2) OPENTAB (Kong et al., 2024), and (3) OpenTable-R1 (Qiu, 2025). For closed-domain TQA, we include two state-of-the-art methods: (4) ReacTable (Zhang et al., 2024) and (5) AutoPrep (Fan et al., 2025). For fairness, all open-domain baselines are evaluated on the same data lake, query set, and retrieval space for each dataset. We report closed-domain systems separately as upper-bound references, because they are provided with gold tables and therefore do not face the retrieval stage.

Metrics. Following prior work (Zhang et al., 2024), we use ROUGE (Wang et al., 2025) to evaluate performance on FeTaQA and RETQA, since their ground-truth answers are free-form. We use exact match accuracy for WikiTables and TabFact. For WikiTables, predictions are normalized for number formatting and letter casing, while TabFact uses

binary labels (True/False). For both metrics, the higher value indicates better performance.

Implementation details. We set the default number of retrieved tables to 10, use DeepSeek-V3 as the default LLM for agents, and Gemma-2-9b-it for SQL translation due to its cost efficiency. Both are among the top open-source LLMs available at the time of our experiments. The deployment details are in Sec. C. λ , which is used for balancing value overlap and schema overlap, is set to 0.6. β , which is used in the training objective, is set to 0.3. We synthesize one question per table for MuRe with a maximum of 4000. The retriever is trained for 20 epochs with batch size 4, learning rate 1×10^{-4} , and weight decay 0.1. In all experiments, the number of context examples provided to the LLM is set to 3. The maximum number of iterations is empirically set to 3. Other hyperparameters for the baseline methods follow their original papers. The codebase is available at <https://github.com/guaiyoui/ArrTab>.

6.2 Main Results

Exp-1: Effectiveness. We first present the main results across four datasets in Figure 2. The models are categorized into open-domain and closed-domain settings. We include closed-domain baselines as upper-bound references, since they are given gold tables while ArrTab needs to retrieve evidence from the full data lake first. In the open-domain setting, our method exhibits outstanding performance, significantly outperforming all baselines across all metrics. Specifically, compared to the strong baseline OPENTAB, our method achieves a substantial improvement of 31.89% and 21.61% in terms of ROUGE-1 on RETQA and TabFact accuracy, respectively. Furthermore, our method surpasses OpenTable-R1 by 30.60% on FeTaQA (ROUGE-L). These results highlight the superiority of our framework in retrieving and reasoning over tables in an open-domain context. Remarkably, despite operating in the more challenging open-domain setting (which requires retrieval), our method demonstrates competitive performance against closed-domain models. For instance, on the FeTaQA dataset, our method even outperforms ReacTable and AutoPrep by 4.67% and 6.30% in terms of ROUGE-L, respectively. The effectiveness of the model in multi-table environments and the superior quality of the retrieved evidence allow ArrTab to achieve competitive results.

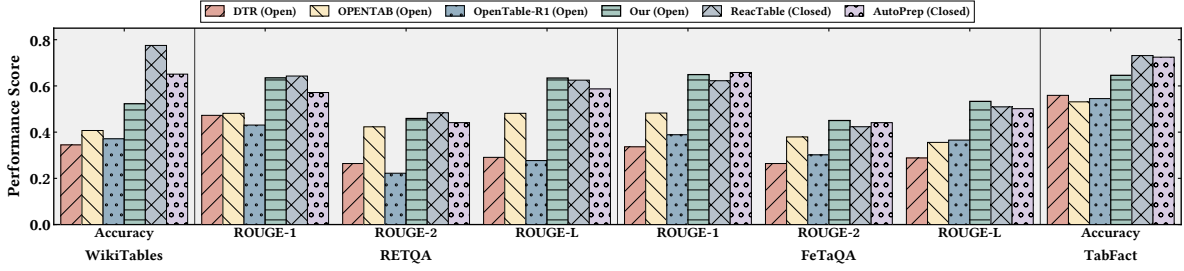


Figure 2: Results on four open-domain TQA benchmarks (WikiTables, RETQA, FeTaQA, and TabFact).

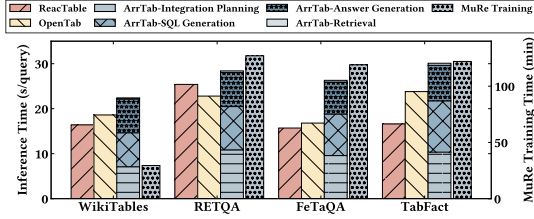


Figure 3: Efficiency study

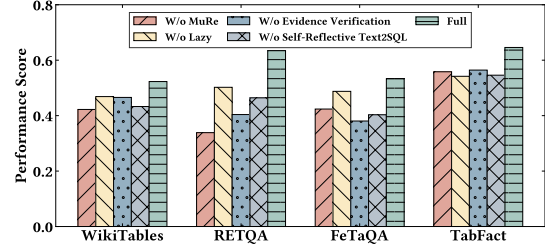


Figure 4: Ablation study

6.3 Efficiency and Ablation Study

In the following experiments, the performance scores are measured by *ROUGE-L* for RETQA and FeTaQA and *accuracy* for WikiTable and TabFact.

Exp-2: Efficiency analysis. We compare the inference latency of ArrTab with ReacTable and OPENTAB in Figure 3. ArrTab incurs moderately higher latency due to its multi-agent reasoning process, with most overhead coming from integration planning, SQL generation, and answer generation, while retrieval itself is negligible. Across datasets, ArrTab remains within a practical latency range and provides a favorable trade-off between accuracy and computational cost. We also report the one-time MuRe training cost (30–120 minutes), which is conducted offline and thus imposes no overhead on per-query inference latency.

Exp-3: Ablation study. We conduct component-level ablations to quantify the contribution of each module. As shown in Figure 4, all components contribute consistently to the final performance. Replacing MuRe with BM25 causes the largest average drop of 0.1483, with the most severe degradation on RETQA (0.6342 → 0.3386), confirming the importance of multi-table retrieval. Removing evidence verification, self-reflective Text2SQL, and lazy execution reduces performance by 0.1304, 0.1230, and 0.0839 on average, respectively. These results show that ArrTab’s gains are not driven by a single component, but by the joint effect of reliable retrieval, evidence checking, robust SQL extraction, and delayed physical execution.

6.4 Hyperparameter Study

Exp-4: Impact of retrieved table number. Figure 5(a) shows the effect of varying the retrieved table number K . Increasing K from 6 to 10 improves the average performance from 0.5374 to 0.5840, with clear gains on RETQA (0.5493 → 0.6342) and TabFact (0.5777 → 0.6459). Further increasing K to 12 slightly reduces the average score to 0.5798, indicating that extra retrieved tables may introduce noise. We therefore use $K = 10$ as the default setting.

Exp-5: Impact of LLMs. We further the impact of LLM on ArrTab, evaluating Qwen3-32B, DeepSeek-V3, and Gemini-2.5-pro in Figure 5(b). Stronger LLM backbones generally improve performance, with the average score increasing from 0.5308 with Qwen3-32B to 0.5840 with DeepSeek-V3 and 0.6028 with Gemini-2.5-pro. Gemini-2.5-pro achieves the best results on three of four datasets, including FeTaQA (0.5804) and TabFact (0.6658). These results show that ArrTab benefits from stronger reasoning backbones while preserving the same retrieval and agentic workflow.

Exp-6: Impact of retry times. We also analyze the effect of the maximum retry number Y . The results are in Figure 5(c). Increasing Y from 1 to 3 improves the average performance from 0.4536 to 0.5840, with large gains on RETQA (0.4039 → 0.6342) and FeTaQA (0.3806 → 0.5331). Further increasing Y to 4 brings no clear benefit, with the average score remaining nearly unchanged at 0.5807. We therefore set $Y = 3$ as the default setting, balancing self-correction benefits and addi-

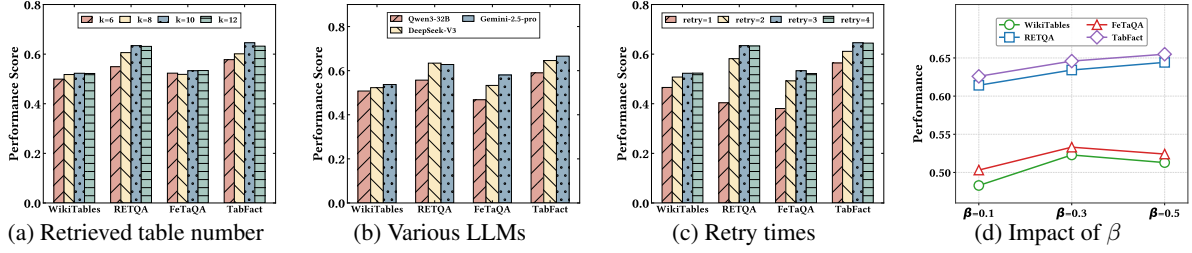


Figure 5: Hyperparameter study of retrieved table number, used LLMs, retry times and β .

tional inference cost.

Exp-6: Impact of β . We analyze the effect of the balance parameter β in Figure 5(d). Increasing β from 0.1 to 0.3 improves the average performance from 0.5565 to 0.5840, indicating that stronger partner-discovery weighting improves multi-table evidence construction. Further increasing β to 0.5 slightly reduces the average score to 0.5840, with mixed effects across datasets. We therefore set $\beta = 0.3$ as the default setting, which provides the most stable performance across datasets.

6.5 Additional Experiments

We add more additional reproducibility details in Appendix C, including: (1) *LLM deployment*, (2) *Retriever architecture and training*, (3) *Inter-agent state transfer*, and (4) *Context budgeting*. To further validate ArrTab, we conduct the following additional experiments in Appendix C: (1) *SQL-to-question translation ablation*: comparing Gemma-family models for MuRe data synthesis. (2) *Stage-wise latency breakdown*: profiling the inference cost of each agent. (3) *MuRe training data analysis*: analyzing the statistics of augmented training data. (4) *Case study*: illustrating the self-correction behavior of the pipeline on a representative messy-context example.

7 Related Works

Our work is closely related to both open-domain and closed-domain TQA.

Closed-domain TQA. With the provided grounding tables, closed-domain TQA aims to accurately answer natural language queries by reasoning over the structured data and relationships. Recent approaches are predominantly language-model-based, particularly leveraging LLMs. These methods can be broadly categorized into two groups: *training-based models* and *code-generation-based models*. The training-based approaches directly train or fine-tune custom models to generate answers from input tables, typically requiring ground-truth labels

for supervision. Representative models include TAPAS (Herzig et al., 2020), TAPEx (Liu et al.), TACUBE (Zhou et al., 2022), and OMNITAB (Jiang et al., 2022). With the rapid development of LLMs (Tan et al., 2025b, 2026, 2025a), recent work has focused on instructing models to generate executable code, such as SQL or Python, to manipulate tables and reason over structured data more effectively. Representative studies include REACTABLE (Zhang et al., 2024), AUTOTQA (Zhu et al., 2024a), and AUTOPREP (Fan et al., 2025). Recent studies explore wider settings, such as multi-table question answering (Pal et al., 2023) and multi-modal question answering (Talmor et al.; Chen et al., 2020).

Open-domain TQA. Since open-domain TQA operates over a large data lake, it typically involves a retrieval stage that selects a subset of relevant tables (Chen et al., 2024a; Zhang and Balog, 2020; Deng et al., 2019). Once the relevant tables are retrieved, the task reduces to a regular TQA problem. Traditional pipelines rely on static retrieval methods (e.g., TF-IDF, BM25) followed by closed-domain semantic parsing (Herzig et al., 2020). More recently, researchers have explored leveraging LLM (Kong et al., 2024) and tool-integrated reasoning frameworks (Qiu, 2025) to enhance the performance of open-domain TQA.

8 Conclusion

In this work, we introduced ArrTab, an agentic open-domain TQA framework with a self-supervised multi-table retriever MuRe. MuRe is trained in a self-supervised manner with LLM-augmented questions and the table-to-SQL-to-question pipeline. Based on MuRe, a multi-agent collaboration paradigm is proposed, which achieves both high efficiency and robust reasoning. The proposed lazy execution strategy further enhances accuracy through globally consistent reasoning. Experimental results on real-world datasets validate the superiority of ArrTab.

9 Limitations

While the proposed framework demonstrates significant improvements in open-domain tabular question answering, it involves certain limitations. The coordination of multiple agents and the use of iterative feedback loops introduce additional computational overhead, resulting in higher inference latency compared to single-pass pipelines. Moreover, although the simulated data lake effectively captures challenges such as schema heterogeneity, further research is necessary to evaluate the robustness of the framework in even more complex, large-scale real-world environments.

10 Acknowledgment

This work is supported by the Australian Research Council Industry Transformation Research Hub for Next Generation Mining Methods and the Australian Research Council Centre of Excellence for Mathematical Modelling of Cellular Systems (MACSYS). Moreover, we thank the anonymous reviewers, area chairs, senior area chairs, and program chairs for their valuable service.

References

- Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549.
- Mengshi Chen, Yuxiang Sun, Tengchao Li, Jianwei Wang, Kai Wang, Xuemin Lin, Ying Zhang, and Wenjie Zhang. 2026. Empowering tabular data preparation with language models: Why and how? *ACL*.
- Peter Baile Chen, Yi Zhang, and Dan Roth. 2024a. 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)*, pages 2687–2699.
- Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439*.
- Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyu Zhou, and William Yang Wang. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.
- Yanxi Chen, Chunxiao Li, Xinyang Dai, Jinhuan Li, Weiyu Sun, Yiming Wang, Renyuan Zhang, Tinghe Zhang, and Bo Wang. 2024b. Boosting single positive multi-label classification with generalized robust loss. *arXiv preprint arXiv:2405.03501*.
- Li Deng, Shuo Zhang, and Krisztian Balog. 2019. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1029–1032.
- Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *IEEE Transactions on Big Data*.
- Meihao Fan, Ju Fan, Nan Tang, Lei Cao, Guoliang Li, and Xiaoyong Du. 2025. Autoprep: Natural language question-aware data preparation with a multi-agent framework. *Proc. VLDB Endow.*, 18(10):3504–3517.
- Zihui Gu, Ruixue Fan, Xiaoman Zhao, Meihui Zhang, Ju Fan, and Xiaoyong Du. 2022. Opentfv: An open domain table-based fact verification system. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2405–2408.
- Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open domain question answering over tables via dense retrieval. *arXiv preprint arXiv:2103.12011*.
- Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.
- Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942.
- Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: recent advances. In *China Conference on Knowledge Graph and Semantic Computing*, pages 174–186. Springer.
- Deokhyung Kang, Baikjin Jung, Yunsu Kim, and Gary Geunbae Lee. 2024. Denoising table-text retrieval for open-domain question answering. European Language Resources Association (ELRA).
- Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. 2024. Opentab: Advancing large language models as open-domain table reasoners. In *The Twelfth International Conference on Learning Representations*.

- Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations*.
- Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailley Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.
- Vaishali Pal, Andrew Yates, Evangelos Kanoulas, and Maarten de Rijke. 2023. 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)*, pages 6322–6334.
- Zipeng Qiu. 2025. Opentable-r1: A reinforcement learning augmented tool agent for open-domain table question answering. *arXiv preprint arXiv:2507.03018*.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-bert: Sentence embeddings using siamese bert-networks](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Si Sun, Chenyan Xiong, Yue Yu, Arnold Overwijk, Zhiyuan Liu, and Jie Bao. 2022. Reduce catastrophic forgetting of dense retrieval training with teleportation negatives. *arXiv preprint arXiv:2210.17167*.
- Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hananeh Hajishirzi, and Jonathan Berant. Multi-modalqa: complex question answering over text, tables and images. In *International Conference on Learning Representations*.
- Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, and Wenjie Zhang. 2025a. Paths-over-graph: Knowledge graph empowered large language model reasoning. In *Proceedings of the ACM on Web Conference 2025*, pages 3505–3522.
- Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, Liming Zhu, and Wenjie Zhang. 2025b. Hydrarag: Structured cross-source enhanced large language model reasoning. *EMNLP*.
- Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, Liming Zhu, and Wenjie Zhang. 2026. Skillzip: Contract-preserving graph compression for scalable agent skill libraries. *arXiv preprint arXiv:2608.05604*.
- Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT — Building open translation services for the World. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.
- Jianwei Wang, Kai Wang, Ying Zhang, Wenjie Zhang, Xiwei Xu, and Xuemin Lin. 2025. [On llm-enhanced mixed-type data imputation with high-order message passing](#). *Proc. VLDB Endow.*, 18(10):3421–3434.
- Mengqi Wang, Jianwei Wang, Qing Liu, Xiwei Xu, Zhenchang Xing, Liming Zhu, Michael Bain, and Wenjie Zhang. 2026. [Ensembling llm-induced decision trees for explainable and robust error detection](#). In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, page 4963–4974, New York, NY, USA. Association for Computing Machinery.
- Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data discovery using natural language questions via a self-supervised approach. *Proceedings of the ACM on Management of Data*, 1(4):1–27.
- Yiqi Wang, Long Yuan, Wenjie Zhang, Xuemin Lin, Zi Chen, and Qing Liu. 2024a. Simpler is more: Efficient top-k nearest neighbors search on large road networks. *Proceedings of the VLDB Endowment*, 17(13):4683–4695.
- Zhensheng Wang, Wenmian Yang, Kun Zhou, Yiquan Zhang, and Weijia Jia. 2024b. [RETQA: A large-scale open-domain tabular question answering dataset for real estate sector](#). In *The 39th Annual AAAI Conference on Artificial Intelligence*.
- Guosong Zhan, Ge Shi, Xiaoguo Liu, and Nan Wang. 2025. Natural language to sql using large language models. In *2025 21st International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 495–502. IEEE.
- Shuo Zhang and Krisztian Balog. 2020. Web table extraction, retrieval, and augmentation: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(2):1–35.
- Yunjia Zhang, Jordan Henkel, Avriela Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2024. Reactable: enhancing react for table question answering. *Proceedings of the VLDB Endowment*, 17(8):1981–1994.
- Fan Zhou, Mengkang Hu, Haoyu Dong, Zhoujun Cheng, Fan Cheng, Shi Han, and Dongmei Zhang. 2022. Tacube: Pre-computing data cubes for answering numerical-reasoning questions over tabular data. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2278–2291.
- Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2024a. Autotqa: Towards autonomous tabular question answering through multi-agent large language models. *Proceedings of the VLDB Endowment*, 17(12):3920–3933.
- Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024b. Large language model enhanced text-to-sql generation: A survey. *arXiv preprint arXiv:2410.06011*.

Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. 2025. Rag over tables: Hierarchical memory index, multi-stage retrieval, and benchmarking. *arXiv preprint arXiv:2504.01346*.

A Implementation Details

This section provides implementation details for the main modules in our framework, including the SQL-to-text translation procedure and the prompt designs of the four specialized agents. All agents operate within a unified state-space, passing intermediate results to subsequent modules.

A.1 SQL-to-Text Translation Implementation

The SQL-to-text module is used during the synthetic data construction process of MuRe. Given a generated multi-table SQL query, the module uses an LLM to convert the formal query into a natural-language question, which then serves as weak supervision for retriever training. Rather than directly verbalizing table or column identifiers, the prompt encourages the model to replace raw schema names with content-grounded descriptions and to express SQL operators in natural language.

Prompt Design: SQL-to-Question

Input. Multi-table SQL query S .

Guidance. Translate the query into a natural English question. Ensure raw identifiers are replaced with content-based descriptions and operators are contextualized for readability.

Output. The natural question q only.

A.2 Table Retrieval Agent

The Table Retrieval Agent is the entry point of the pipeline. It filters the massive data lake and retrieves a small set of candidate tables that are most relevant to the user’s query. In our implementation, we utilize MuRe to score the relevance between the natural language query q and all available tables in the data lake. The agent selects the top- k (specifically, $k = 10$ in our experiments) tables based on these scores. These retrieved tables, denoted as $\mathcal{T}_{retrieved}$, are then loaded into the system state for subsequent integration analysis.

A.3 Table Integration Agent

The Table Integration Agent is designed for understanding the relationships between the retrieved tables and the query. It performs two critical tasks: (1) Operation Decision, determining whether the tables should be combined via a *Join* (vertical) or *Union* (horizontal) operation, or if no combination is needed; and (2) Schema Matching, identi-

Table 1: Datasets Details

Dataset	# of Tables	# of Questions	Output	Origin	Description
WikiTables	560	500	Short answer	Native multi-table	Wikipedia web-table QA with 1–3 related evidence tables.
RETQA	4932	20762	Free-form text	Native multi-table	Multi-hop reasoning over the Chinese real-estate sector.
FeTaQA	17639	10330	Free-form text	Simulated multi-table	Abstractive reasoning requiring aggregation on Wikipedia tables.
TabFact	199078	117884	Binary label (True/False)	Simulated multi-table	Large-scale fact verification on open-domain Wikipedia tables.

finding semantically equivalent column pairs across different tables (e.g., matching “born_date” with “dob”). This agent takes the schema information of $\mathcal{T}_{retrieved}$ as input and outputs a structured integration plan. The prompt design is in Box A.4.

A.4 Information Extraction Agent

Once the data is integrated, the Information Extraction Agent generates an executable SQL query to extract the precise answer. This agent receives the integration instructions (e.g., which columns to align) and the serialized content of the selected tables. To ensure robustness against dirty data, the prompt explicitly instructs the model to handle ‘NULL’ values, perform fuzzy matching for entity names (using ‘LIKE’), and cast data types correctly before arithmetic operations. The generated SQL is executed against an in-memory SQLite database.

Prompt Design: Table Integration Agent

Input. User query q and Schema Information $S = \{(T_{name}, [c_1, \dots, c_m])\}$.

Guidance. You are a data integration expert. Analyze the table schemas and the query to determine the optimal combination strategy.

- Identify the operation type: “Join” (column-wise) or “Union” (row-wise).
- Select table subset relevant to the query.
- Identify matched column pairs for schema alignment.
- If the tables lack sufficient information, output “No need”.

Output. A JSON object containing join_union_type, selected_tables, and matched_pairs.

retrieved evidence is sufficient to support answering the query. The prompt design for this iterative reasoning process is illustrated in Box A.5. Concretely, the verification step is invoked when the extracted evidence is insufficient for answering the question, including cases where SQL execution returns an empty result or the returned result lacks the necessary factual information.

Information Extraction Agent

Input. Query q , Integration Instructions, and Table Samples.

Guidance. You are a SQL writing assistant. Generate a SQL query to extract the answer. (1) *Data Cleaning*: Filter out rows where key numeric fields are NULL or ‘nan’. (2) *Fuzzy Matching*: Use ‘LIKE’ operators for entity names to handle potential typos or formatting differences. (3) *Type Casting*: Ensure string values are cast to numeric types before mathematical calculation. (4) *Syntax*: Use double quotes for column names with special characters.

Output. The executable SQL query string.

Answer Agent

Input. User query q and Retrieved Evidence R (SQL Execution Results).

Guidance. You are a rigorous answer agent. Analyze whether the retrieved evidence R provides a complete and factual basis to answer q without hallucination. Based on this verification, strictly adhere to the following branching logic: if R is sufficient, synthesize the final natural language answer a ; conversely, if critical information is missing or ambiguous, diagnose the information gap and generate a reformulated query q' aimed at retrieving the specific missing details.

Output. A JSON object containing the verification status (“sufficient” or “insufficient”) and the corresponding content (either the answer or the reformulated query).

A.5 Answering Agent

The Answering Agent acts as the final reasoning module, synthesizing a natural language answer based on the query q and the retrieved evidence R . We adopt a *Evidence Sufficiency Verification* step that explicitly evaluates the sufficiency of R with respect to q before generating the final answers. The LLM internally assesses whether the

A.6 Communication Mechanism

The agents communicate through a shared structured state rather than free-form dialogue. Each agent reads the fields relevant to its role, writes its intermediate output back to the state, and passes the updated state to the next agent. This design makes inter-agent communication explicit, reduces redundant context, and keeps each stage grounded in the outputs of previous stages.

Concretely, the integration agent receives schema-level inputs represented as $\{(T_{\text{name}}, [c_1, \dots, c_m])\}$ and writes an integration plan specifying selected tables, relevant columns, and potential join or union relationships. The SQL agent then reads this plan together with the selected table names, full column lists, and up to six sampled rows rendered by `pandas.DataFrame.to_string(index=False)`, and writes executable SQL queries and extraction results. Finally, the answer agent receives the extracted evidence as compact JSON-style records with column names and at most eight rows, verifies evidence sufficiency, and produces the final answer or a rewrite request when necessary.

B Datasets Details

B.1 Dataset Statistics

We evaluate ArrTab on four table-centric benchmarks covering both native and simulated multi-table settings, as summarized in Table 1. RETQA and WikiTables are native multi-table benchmarks, while FeTaQA and TabFact are converted into simulated multi-table data lakes through table fragmentation. This combination allows us to evaluate naturally occurring cross-table reasoning as well as in open-domain retrieval settings.

WikiTables is a native multi-table extension of WikiTableQuestions built from Wikipedia web tables. The dataset is from (Zou et al., 2025). We convert each unique source table into a CSV file in the data lake, and preserve the benchmark-provided question-answer pairs and oracle evidence-table groupings. Our processed version contains 560 unique tables and 500 questions, where each question may require evidence from 1–3 related tables and approximately 45% of questions require more than one table.

RETQA contains 4,932 tables and 20,762 questions from real-world Chinese real-estate data. It is naturally multi-table, where answering a question often requires integrating evidence across related

tables, and the output is free-form text.

FeTaQA is an abstractive table QA benchmark built on Wikipedia tables, containing 17,639 tables and 10,330 questions. Its answers are free-form text and typically require aggregation, comparison, or summarization over tabular content. TabFact contains 199,078 tables and 117,884 statements for table-based fact verification, where each output is a binary label. For FeTaQA and TabFact, we construct simulated multi-table settings from their original single-table supervision to study the scalability of open-domain TQA (the detailed processing process is in Sec. B.2).

We further quantify retrieval-space heterogeneity using three statistics in Table 2. First, schema heterogeneity is defined as

$$H_{\text{schema}}(\mathcal{D}) = \frac{|\bigcup_{T \in \mathcal{D}} \text{NormCols}(T)|}{\sum_{T \in \mathcal{D}} |\text{Cols}(T)|}, \quad (8)$$

where $\text{NormCols}(T)$ denotes the normalized column-name set of table T after lower-casing and trimming. Second, for each table T with cell set $\text{Cells}(T)$, the missing rate is

$$\text{MR}(T) = \frac{\sum_{c \in \text{Cells}(T)} \mathbb{I}[c \text{ is missing}]}{|\text{Cells}(T)|}, \quad (9)$$

and the average missing rate of data lake $\mathcal{D}$ is

$$\text{AvgMR}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{T \in \mathcal{D}} \text{MR}(T). \quad (10)$$

Finally, we define the percentage of noisy tables as

$$\text{NoisePct}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{T \in \mathcal{D}} \mathbb{I}[\text{MR}(T) > 0.5] \times 100. \quad (11)$$

As shown in Table 2, RETQA has a relatively homogeneous schema space but still requires cross-table evidence composition, TabFact provides the largest retrieval space, and WikiTables exhibits the highest schema heterogeneity. These complementary properties stress different aspects of open-domain tabular QA, including evidence retrieval, schema variation, and noisy-table filtering.

B.2 Dataset Processing and Construction

For native multi-table benchmarks, we preserve the released retrieval space and table groupings, applying only format normalization needed by our pipeline. For originally single-table benchmarks, we construct simulated multi-table data

Table 2: Retrieval-space statistics of the four evaluation data lakes.

Dataset	# Tables	Avg. Rows	Avg. Cols	Schema Heterogeneity	Avg. Missing Rate (%)	Noisy Tables (%)
WikiTables	560	13.45	6.28	0.2805	4.36	0.18
RETQA	4,932	22.51	10.13	0.0007	5.46	0.00
FeTaQA	17,639	10.36	4.92	0.1501	6.21	0.01
TabFact	199,078	9.48	5.48	0.0178	7.16	0.00

lakes through controlled table partition and noise injection. This design follows the common practice of adapting single-table TQA benchmarks to multi-table datasets to evaluate cross-table reasoning capabilities (Zou et al., 2025).

For native multi-table datasets, we avoid artificial repartitioning. RETQA is evaluated in its original open-domain form: the retrieval space consists of the 4,932 real-estate tables released with the dataset, and the native table boundaries are preserved. Since the raw RETQA data is in Chinese, we apply *Helsinki-NLP/opus-mt-zh-en* (Tiedemann and Thottingal, 2020) to translate table names, column schemas, cell values, questions, and answers into English, followed by format normalization. WikiTables is also used as a native multi-table benchmark. We build it from (Zou et al., 2025), convert each unique source table into a CSV file, and preserve the benchmark-provided question-answer pairs and oracle evidence-table groupings. Thus, RETQA and WikiTables evaluate ArrTab under naturally multi-table retrieval spaces rather than simulated tables.

For single-table datasets, namely FeTaQA and TabFact, we follow a controlled simulation strategy to create fragmented data lakes from their original table-level supervision. Since each original instance is paired with a single table, we follow the widely common practice (Zou et al., 2025) and apply vertical or horizontal partitioning to decompose the table into multiple related fragments. Vertical partitioning creates join-style evidence by splitting columns while preserving key columns as join paths. Horizontal partitioning creates union-style evidence by splitting rows and further injecting missingness with the missing completely at random (MCAR) mechanism and LLM-based schema rewriting. This construction is intended to model common table fragmentation and schema variation in real-world data lakes, rather than to simplify retrieval. The full table processing procedure is formalized in Algorithm 1, and the schema-rewriting prompt is shown below.

Algorithm 1: Simulation Pipeline

Input : Source Dataset $\mathcal{D}$, Agent $\mathcal{A}_{\text{schema}}$

Output : Processed Data Lake $\mathcal{D}_{\text{lake}}$

```

1  $\mathcal{D}_{\text{lake}} \leftarrow \emptyset$ 
2 for each  $(T, q, a) \in \mathcal{D}$  do
3    $s \leftarrow \text{Random}(\{\text{Vertical}, \text{Horizontal}\})$ 
4   if  $s = \text{Vertical}$  then
5     // Decompose columns
5     // preserving join keys
5      $\{T_A, T_B\}, \mathcal{J} \leftarrow \text{Decompose}(T)$ 
6      $S_{\text{meta}} \leftarrow \{\text{tables} : [T_A, T_B], \text{Join}\}$ 
7   end
8   else if  $s = \text{Horizontal}$  then
9     // Row-wise split with noise
9     // injection
9      $T_{\text{top}}, T_{\text{bottom}} \leftarrow \text{SplitRows}(T, \rho = 0.6)$ 
10     $T_{\text{top}} \leftarrow \text{MaskMCAR}(T_{\text{top}})$ 
11     $C_{\text{orig}} \leftarrow \text{GetColumns}(T_{\text{bottom}})$ 
12     $C_{\text{alt}} \leftarrow \mathcal{A}_{\text{schema}}(\mathbf{P}_{\text{schema}}, C_{\text{orig}})$ 
13     $T_{\text{bottom}}.\text{cols} \leftarrow C_{\text{alt}}$ 
14     $S_{\text{meta}} \leftarrow \{\text{tables} :$ 
14     $\quad [T_{\text{top}}, T_{\text{bottom}}], \text{Union}\}$ 
15  end
16   $\mathcal{D}_{\text{lake}}.\text{append}(\{q, a, S_{\text{meta}}\})$ 
17 end
18 return  $\mathcal{D}_{\text{lake}}$ 

```

Prompt Design: Schema Heterogeneity

Input. Original table columns C_{orig} and sample rows.

Guidance. You are an expert data engineer. Simulate schema heterogeneity in a data lake by generating *alternative column names* for the provided table. The new names must be semantically equivalent to the originals but lexically different (e.g., utilizing synonyms or abbreviations), while maintaining the original column order.

Output. A JSON object containing the list ‘alternative_column_names’.

C Additional Experiments

C.1 Additional Reproducibility Details

(1) LLM deployment. In our evaluation, we use API-based LLMs for the agentic reasoning pipeline for SQL-to-question translation in MuRe data synthesis. In the main experiments, the default reasoning backbone is deepseek-ai/DeepSeek-V3, accessed through the Nebius AI Studio OpenAI-compatible API. DeepSeek-V3 is the leading open-source LLM when doing the experiments. Note that, during the revision of this paper, the DeepSeek-V3 access on Nebius expired, so we switched to OpenRouter. For backbone comparison, we additionally evaluate Qwen3-32B (from Nebius) and Gemini-2.5-pro (from the official provider) through their corresponding OpenAI-compatible endpoints. For SQL-to-question translation, we use Gemma-family models from Nebius, with Gemma-2-9B-it as the default setting and smaller/larger variants used for sensitivity analysis.

(2) Retriever architecture and training. MuRe follows a transformer-based dual-encoder design with separate question and table encoders. Questions are truncated to 40–50 tokens, table passages to 200–300 tokens, and the retriever is trained for 20 epochs with batch size 4, learning rate 1×10^{-4} , and weight decay 0.1.

(3) Inter-agent state transfer. Agents communicate through a shared structured state dictionary with fixed fields. Each agent reads the fields required by its role and writes its output back to the same state:

```
{
  "query",      "table_names",    "tables",
  "selected_tables", "integration_plan",
  "sql_tables",  "sql",          "evidence",
  "evidence_sufficient", "rewritten_query",
  "final_answer" }
```

This explicit state interface supports lazy execution, evidence verification, and retry control in the online question answering pipeline.

(4) Context budgeting. To keep prompts within a manageable length, we retrieve the top-K=10 candidate tables, select at most three for SQL generation, and serialize only a small number of representative rows per table (six rows drawn at random). This ensures context length is controlled through principled table selection and row sampling, rather than late-stage prompt truncation.

C.2 Ablation of SQL-to-Question

We evaluate the impact of the SQL-to-question translation model used in MuRe’s synthetic data

Table 3: SQL-to-question translation model ablation on FeTaQA. Latency and token usage are measured during offline SQL-to-question generation, while ROUGE-L reports downstream QA performance.

SQL-to-NL Model	Avg. Latency (s)	Avg. Total Tokens	FeTaQA ROUGE-L
Gemma-2-2B	0.61	328.5	0.5193
Gemma-2-9B-it	0.69	305.2	0.5331
Gemma-3-27B	0.85	252.0	0.5345

Table 4: Stage-wise latency and token usage on FeTaQA with max retries = 3 and the default backbone.

Stage	Latency (s)	Prompt Tokens	Completion Tokens	Total Tokens
Retrieval	0.37	0	0	0
Integration Planning	9.579	4863	42	4905
SQL Generation	9.367	2573	104	2677
Answer Generation	7.245	2195	31	2226
Total	26.235	9631	177	9808

construction. As shown in Table 3, smaller and larger Gemma-family translators differ in generation cost, while the downstream FeTaQA ROUGE-L measures how the resulting synthetic questions affect final QA performance. We use Gemma-2-9B-it as the default translator because it provides a practical balance between translation quality and offline generation cost.

C.3 Stage-wise Latency

Table 4 reports the stage-wise inference cost of ArTab on FeTaQA under the default setting. The overall latency is 26.235 seconds, with most of the cost coming from integration planning and SQL generation, which take 9.579 and 9.367 seconds, respectively. These two stages also account for the majority of token usage, consuming 7,582 of the 9,808 total tokens. In contrast, retrieval is lightweight, taking only 0.37 seconds and requiring no LLM tokens. Answer generation adds 7.245 seconds and 2,226 tokens, reflecting the final evidence-grounded reasoning step. These stage-wise results show that overhead is primarily introduced by structured multi-agent reasoning.

C.4 MuRe Training Data Analysis

Table 5 summarizes the scale and composition of the synthesized MuRe training corpus. The corpus contains 4,420 training samples with broad coverage of filtered aggregation queries, including MAX, MIN, COUNT, SUM, and AVG operations. The SQL operation counts are non-exclusive, since a query may combine aggregation, filtering, and join or union structures. A manual inspection of 20 sampled multi-table pairs estimates the obvious partner-noise rate at 20.0%.

Table 5: Summary of the synthesized MuRe training corpus on FeTaQA.

Statistic	Value
Training samples	4,420
Degenerate single-table fraction	66.2%
Distinct-1 / Distinct-2	0.0278 / 0.0891
JOIN-containing SQL	1,401 (31.7%)
UNION-containing SQL	95 (2.1%)
Aggregation ops	4,420
WHERE clauses	4,420 (100%)
Manual partner-noise estimate	4/20 (20.0%)

C.5 Case Study

We present a representative messy-context success case. The question is: *“When did Andy Karl win the Olivier Award and for which of his work?”*

Step 1: Retrieval context. The retrieved neighborhood contains the correct table, `0_2275_0_Andy_Karl_Awards` and nominations. It also contains one distractor table tagged as messy because its headers are raw cell-like values, such as 1964, Aileen Ward, and John Keats: The Making of a Poet. Thus, the instance is labeled messy at the retrieval-context level, even though the target table itself has a clear schema.

Step 2: First extraction. The integration agent selects the correct Andy Karl awards table. However, the SQL generator produces an overly restrictive predicate due to limited context:

```
WHERE Award LIKE '%Olivier%'
AND Result LIKE '%Won%'.
```

The query is syntactically valid but returns an empty result, leaving insufficient evidence for answer generation.

Step 3: Verification and rewrite. The verifier detects that SQL execution succeeded but the extracted evidence is empty. It rewrites the original query as follows:

“When did Andy Karl win the Olivier Award and for which of his work?”

into a more evidence-oriented request:

“Provide a table of Andy Karl’s Olivier Awards, listing the award year and the production/work for which he won, or a table of Olivier Award winners in categories you think Andy Karl appears in.”

Step 4: Second extraction. The system re-enters extraction over the same oracle table set. The second SQL broadens the predicate to include alternative Olivier-related and winning-result matches,

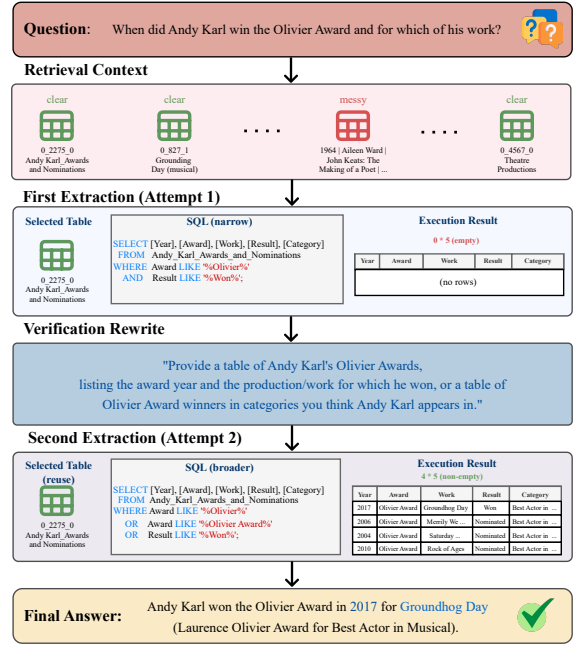


Figure 6: Verification-guided recovery in the Andy Karl messy-context case. The verifier detects that the first SQL execution is valid but insufficient, rewrites the query, and enables a broader second extraction that exposes answer-bearing rows.

rather than requiring both conditions simultaneously. This revised extraction returns non-empty answer-bearing rows.

Step 5: Final answer. Given the recovered evidence, the answer agent produces the grounded answer that Andy Karl won in 2017 for *Groundhog Day*, with the category identified as the Laurence Olivier Award for Best Actor in a Musical. This case shows that verification helps by repairing insufficient evidence through broader second-pass extraction, rather than by retrieving new tables.

D LLM Usage

To enhance clarity and readability, we use LLMs as general-purpose writing assistants during manuscript polishing. Specifically, after drafting the manuscript ourselves, we use LLMs to: 1) suggest alternative phrasing to reduce repetitiveness while maintaining consistency; 2) check consistency in tense, notation, and other grammatical aspects; and 3) recommend more standard terminology or abbreviations where appropriate. We carefully reviewed and edited all LLM-generated text to ensure accuracy and faithfulness, and no text was incorporated without verification by the authors. LLMs were not used for research ideation, experimental design, or data analysis. All technical

claims, equations, proofs, and conclusions were written and verified by the authors. We take full responsibility for the content of this paper.