TableRAG: A Retrieval Augmented Generation Framework for Heterogeneous Document Reasoning

Xiaohan Yu*, Pu Jian*, Chong Chen✉
Huawei Cloud BU, Beijing
{yuxiaohan5, jinapu2, chenchong55}@huawei.com
https://github.com/yxh-y/TableRAG

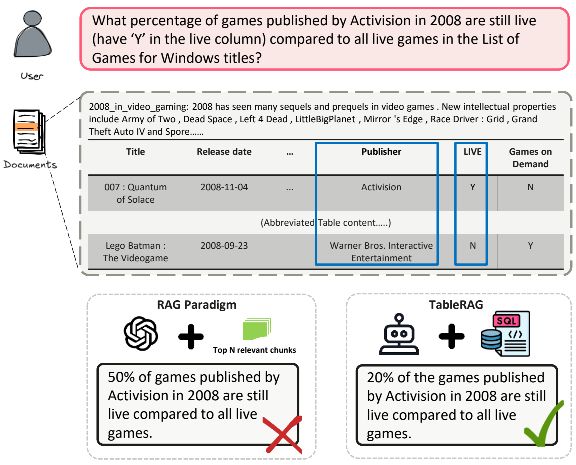
Abstract

Retrieval-Augmented Generation (RAG) has demonstrated considerable effectiveness in open-domain question answering. However, when applied to heterogeneous documents, comprising both textual and tabular components, existing RAG approaches exhibit critical limitations. The prevailing practice of flattening tables and chunking strategies disrupts the intrinsic tabular structure, leads to information loss, and undermines the reasoning capabilities of LLMs in multi-hop, global queries. To address these challenges, we propose TableRAG, an SQL-based framework that unifies textual understanding and complex manipulations over tabular data. TableRAG iteratively operates in four steps: context-sensitive query decomposition, text retrieval, SQL programming and execution, and compositional intermediate answer generation. We also develop HeteQA, a novel benchmark designed to evaluate the multi-hop heterogeneous reasoning capabilities. Experimental results demonstrate that TableRAG consistently outperforms existing baselines on both public datasets and our HeteQA, establishing a new state-of-the-art for heterogeneous document question answering.

1 Introduction

Heterogeneous document-based question answering (Chen et al., 2020), which necessitates reasoning over both unstructured text and structured tabular data, presents substantial challenges. Tables are characterized by interdependent rows and columns, while natural language texts are sequential. Bridging this divergence within a unified QA system remains a non-trivial task.

The prevailing approach extends the retrieval-augmented generation (RAG) paradigm, in which the tables are linearized into textual representations (e.g., Markdown) (Gao et al., 2023; Jin and Lu, 2023; Ye et al., 2023b). Typically, chunking strategies are employed (Finardi et al., 2024), wherein flattened tables are segmented and merged with adjacent text spans. During inference, the LLMs generate answers based on the top-N retrieved chunks. However, these methodologies are predominantly tailored to scenarios that require only surface-level comprehension of tables, such as direct answer extraction (Pasupat and Liang, 2015a; Zhu et al., 2021). When applied to extensive documents that interleave textual and tabular elements, existing RAG methodologies exhibit critical limitations:



What percentage of games published by Activision in 2008 are still live (have 'Y' in the live column) compared to all live games in the List of Games for Windows titles?

2008_in_video_gaming: 2008 has seen many sequels and prequels in video games . New intellectual properties include Army of Two , Dead Space , Left 4 Dead , LittleBigPlanet , Mirror 's Edge , Race Driver : Grid , Grand Theft Auto IV and Spore......

Title	Release date	...	Publisher	LIVE	Games on Demand
007 : Quantum of Solace	2008-11-04	...	Activision	Y	N
(Abbreviated Table content.....)					
Lego Batman : The Videogame	2008-09-23		Warner Bros. Interactive Entertainment	N	Y

50% of games published by Activision in 2008 are still live compared to all live games.

20% of the games published by Activision in 2008 are still live compared to all live games.

Figure 1: An example of the heterogeneous document based question answering task.

- Structural Information Loss: The tabular structure integrity is compromised, leading to information loss or irrelevant context that impedes downstream LLMs performance.
- Lack of Global View: Due to document fragmentation, the RAG system struggles with multi-hop global queries (Edge et al., 2024), such as aggregation, mathematical computations, and other reasoning tasks that require a holistic understanding across entire tables.

[1] *These authors contributed equally to this work.
[2] ✉ Corresponding author.

As illustrated in Figure 1, the RAG approach computes percentage over the top-N most relevant chunks rather than the full table, and thus results in an incorrect answer.

To address these limitations of existing RAG systems, we propose TableRAG, an SQL-based framework that dynamically transitions between textual understanding and complex manipulations over tabular data. TableRAG interacts with tables by leveraging SQL as an interface. Concretely, the framework operates via a two-stage process: an offline database construction phase and an online inference phase of iterative reasoning. The iterative reasoning procedure comprises four core operations: (i) context-sensitive query decomposition, (ii) text retrieval, (iii) SQL programming and execution, and (iv) intermediate answer generation. The utilization of SQL enables precise symbolic execution by treating table-related queries as indivisible reasoning units, thereby enhancing both computational efficiency and reasoning fidelity. To facilitate rigorous evaluation of multi-hop reasoning over heterogeneous documents, we introduce HeteQA, a novel benchmark consisting of 304 examples across nine diverse domains. Each example contains a composition across five distinct tabular operations. We evaluate TableRAG on both established public benchmarks and our HeteQA dataset against strong baselines, including generic RAG and program-aided approaches. Experimental results demonstrate that TableRAG consistently achieves state-of-the-art performance. Overall, our contributions are summarized as follows:

- We identify two key limitations of existing RAG approaches in the context of heterogeneous document question answering: structural information loss and lack of global view.

- We propose TableRAG, an SQL-based framework that unifies textual understanding and complex manipulations over tabular data. TableRAG comprises an offline database construction phase and a four-step online iterative reasoning process.

- We develop HeteQA, a benchmark for evaluating multi-hop heterogeneous reasoning capabilities. Experimental results show that TableRAG outperforms RAG and programmatic approaches on HeteQA and public benchmarks, establishing a state-of-the-art solution.

2 Task Formulation

In the context of the heterogeneous document question answering task, we define the task input as extensive documents, denoted as $(\mathcal{T}, \mathcal{D})$ where $\mathcal{T}$ denotes the textual contents and $\mathcal{D}$ refers to the tabular components. Given a user question q, the objective of this task is to optimize a function $\mathcal{F}$ that, given the combined textual and tabular context, can produce the correct answer $\mathcal{A}$:

$$\mathcal{F}(\mathcal{D}, \mathcal{T}, q) \rightarrow \mathcal{A}. \tag{1}$$

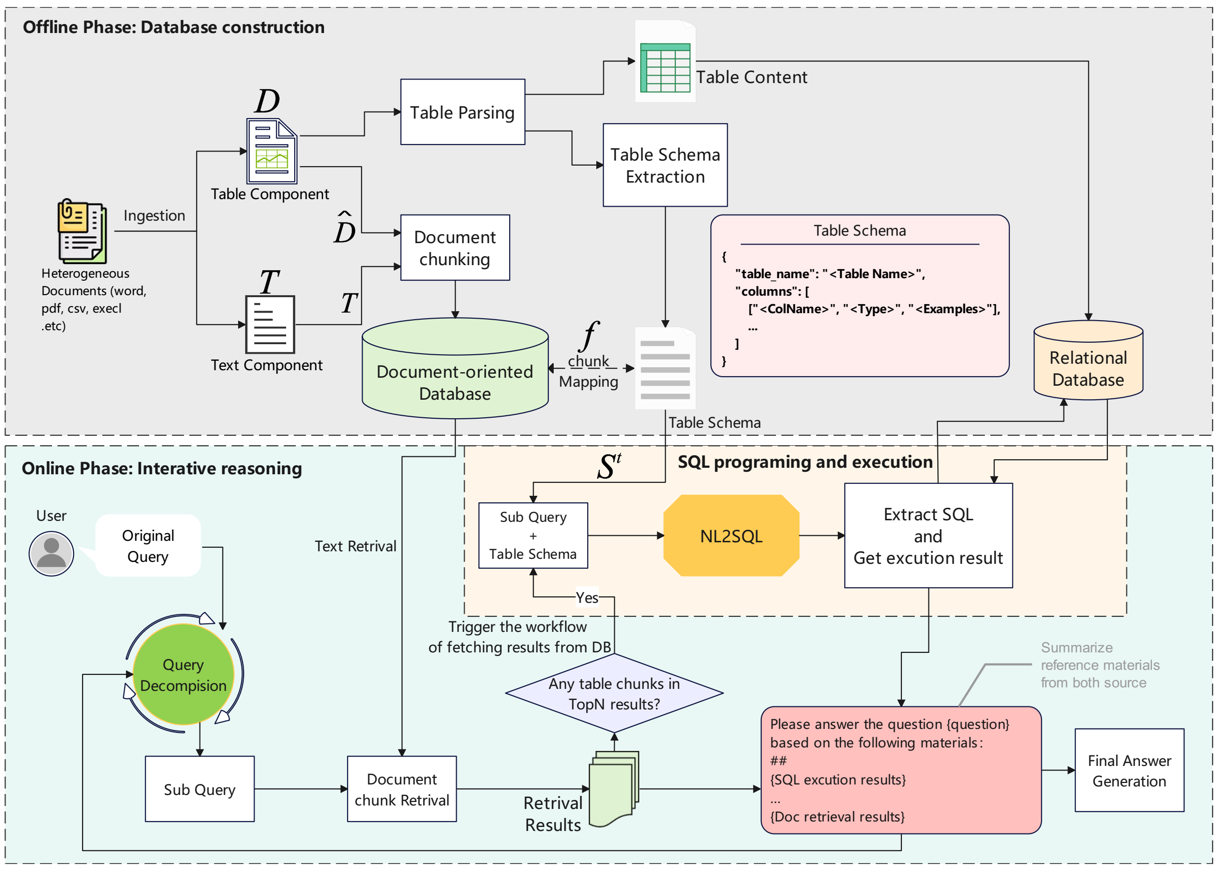
3 TableRAG Framework

3.1 Overview and Design Principles

We propose TableRAG, an SQL-based framework designed to preserve table structural integrity and facilitate heterogeneous reasoning. As depicted in Figure 2, TableRAG consists of offline and online workflows. The offline phase is tasked with database construction, while the online phase facilitates iterative reasoning. The reasoning procedure unfolds in a four-stage process: (i) Context-sensitive query decomposition, which identifies the respective roles of textual and tabular modalities within the query. (ii) Text retrieval. (iii) SQL programming and execution, which is selectively invoked for subqueries requiring tabular data reasoning. (iv) Compositional intermediate answer generation. The preferential use of SQL is motivated by its capacity to leverage the expressive strength of symbolic execution over structured data, thereby enabling tabular components within user queries to be treated as monolithic reasoning units. In contrast, other languages like Python incur substantial computational overhead when dealing with large-scale data or complex workloads (Shahrokhi et al., 2024).

3.2 Database Construction

In the offline stage, we first extract structured components from heterogeneous documents, yielding a set of tables $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$. To enable information retrieval, we construct two parallel corpora: a textual knowledge base and a tabular schema database. The textual knowledge base comprises both the raw texts $\mathcal{T}$ and the Markdown-rendered form of each table, denoted as $\hat{\mathcal{D}}$. Both $\hat{\mathcal{D}}$ and $\mathcal{T}$ are segmented into chunks, which are then embedded into dense vector representations using a pretrained language model (Chen et al., 2024a). For



Offline Phase: Database construction
Heterogeneous Documents (word, pdf, csv, execl .etc)
Ingestion
D
Table Component
T
Text Component
Table Parsing
Table Content
Table Schema Extraction
$\hat{D}$
T
Document chunking
Document-oriented Database
f
chunk Mapping
Table Schema
{ "table_name": "<Table Name>", "columns": [["<ColName>", "<Type>", "<Examples>"], ...] }
Relational Database
Table Schema
Online Phase: Interative reasoning
User
Original Query
Query Decompision
Sub Query
Text Retrival
Document chunk Retrival
S^t
SQL programing and execution
Sub Query + Table Schema
NL2SQL
Extract SQL and Get excution result
Yes
Trigger the workflow of fetching results from DB
Any table chunks in TopN results?
Retrival Results
Summarize reference materials from both source
Please answer the question {question} based on the following materials: ## {SQL excution results} ... {Doc retrieval results}
Final Answer Generation

Figure 2: The overall architecture of TableRAG.

tabular schema database construction, we represent each table $\mathcal{D}_i$ by a standardized schema description $S(\mathcal{D}_i)$, derived via a template as follows:

Table Schema Template

```
{
    "table_name": "<Table Name>",
    "columns": [
        ["<ColName>", "<Type>", "<Examples>"],
        ...
    ]
}
```

Then, we define a mapping from each flattened table chunk to its originating table schema:

$$f : \hat{D}_{i,j} \rightarrow S(\mathcal{D}_i) \qquad (2)$$

where $\hat{\mathcal{D}}_{i,j}$ denotes the j-th chunk derived from table $\hat{\mathcal{D}}_i$. This mapping ensures that local segments remain contextually anchored to the table structure from which they are derived.

The tables are also ingested in a relational database (e.g., MySQL[1]), supporting symbolic query execution in the subsequent online reasoning.

[1]https://github.com/mysql

3.3 Iterative Reasoning

To address multi-hop, global queries that require compositional reasoning over texts and tables, we introduce an iterative inference process aligned with $\mathcal{F}$ in Equation 1. This process comprises four core operations: (i) context-sensitive query decomposition, (ii) text retrieval, (iii) program and execute SQL, and (iv) compositional intermediate answer generation. Through repeated cycles of decomposition and resolution, a solution to the query is progressively constructed. Detailed prompt templates are provided in Appendix G.

Context-Sensitive Query Decomposition We explicitly delineate the respective roles of textual and tabular modalities during the reasoning process. While a table-related query may involve multiple semantic reasoning steps, its tabular resolution can collapse to a single executable operation. Consequently, an effective decomposition of global queries demands more than mere syntactic segmentation, but also structural awareness of the underlying data sources. To this end, we first retrieve the most relevant table content from the textual database and link it to its corresponding table schema description $S(\mathcal{D}^t)$ via the mapping function f. Based on this, we formulate a subquery q_t at the t-th iteration.

Text Retrieval We deploy a retrieval module that operates in two successive stages: vector-based recall followed by semantic reranking. Given an incoming query q_t, it is encoded into a shared dense embedding space alongside document chunks. We then select the top-N candidates with the highest cosine similarity to the query embedding:

$$\hat{\mathcal{T}}^{q_t}_{recall} = \text{top-}N\left(\arg\max_{\hat{\mathcal{T}}_i \in \{\hat{\mathcal{D}}, \mathcal{T}\}} cos(\mathbf{v}_{\hat{\mathcal{T}}_i}, \mathbf{v}_{q_t})\right), \tag{3}$$

In the subsequent reranking stage, the recalled candidate chunks are re-evaluated by a more expressive relevance model, yielding the final top-k selections, denoted by $\hat{\mathcal{T}}^{q_t}_{rerank}$.

SQL Programming and Execution To support accurate reasoning over tabular data, we incorporate a "program-and-execute" mechanism that is selectively invoked only when subquery reasoning involves tables. Specifically, we inspect whether any content originates from tabular sources in the retrieved results. For each chunk in the top-ranked set $\hat{\mathcal{T}}^{q_t}_{rerank}$, we apply the mapping function (in Equation 2) to extract its associated schema, yielding a table schema set:

$$S^t = \{f(\hat{\mathcal{T}}_i) \mid \hat{\mathcal{T}}_i \in \hat{\mathcal{T}}^{q_t}_{rerank}\}. \tag{4}$$

If the set S^t is empty, this module is passed. Otherwise, we derive an accurate answer with the current subquery q_t and the corresponding schema context as inputs. To achieve this, we leverage structured query execution over relational data and use SQL as the intermediate formal language. A dedicated tool f_{SQL} with LLM as backend generates executable SQL programs and applies them to the pre-constructed MySQL database, formalized as follows:

$$e_t = f_{SQL}(S^t, q_t). \tag{5}$$

Intermediate Answer Generation For the subquery q_t, TableRAG can benefit from two heterogeneous information sources: the execution result e_t over SQL database and text retrieval result $\hat{\mathcal{T}}^{q_t}_{rerank}$ from the document database. Both of the data sources provide partial or complete evidence. They introduce distinct failure modes: SQL execution may produce incorrect results or execution errors, while text retrieval may yield incomplete or misleading context. Consequently, the results from these sources may either reinforce each other or present contradictions. To address this, we adopt a compositional reasoning mechanism. The execution result e_t and the retrieved textual chunks $\hat{\mathcal{T}}^{q_t}_{rerank}$ are cross-examined to validate consistency and guide answer selection. The final answer to each subquery is derived by adaptively weighting the reliability of each source based on its evidential utility, $a_t = \mathcal{F}(e_t, \hat{\mathcal{T}}^{q_t}_{rerank})$.

Once the query decomposition module determines that no further subqueries are necessary, TableRAG terminates the iterative reasoning process, yielding the final answer $\mathcal{A} = a_T$, where T denotes the total number of iterations performed.

4 Benchmark Construction

In this section, we present HeteQA, a novel benchmark for assessing multi-hop reasoning across heterogeneous documents.

4.1 Data Collection

HeteQA necessitates advanced operations, such as arithmetic computation, nested logic, etc. To balance annotation fidelity with scalability, we adopt a human-in-the-loop collaborative strategy that integrates LLMs with human verification. The construction pipeline proceeds in three stages:

Query Generation We curate tabular sources from the Wikipedia dataset (Chen et al., 2020). To facilitate analytical depth, we restrict our selection to tables with a minimum of 20 rows and 7 columns, and apply structural deduplication to eliminate redundancy across similar schemas. For each retained table, we define a suite of advanced operations, e.g., conditional filtering, and statistical aggregation. These operations serve as primitives for constructing complex queries. Leveraging the Claude-3.7-sonnet [2], we prompt for query synthesis as compositions over these primitives. Each generated query is paired with executable code in both SQL and Python. We execute the associated code and obtain the answer. A final deduplication pass is applied over both queries and answers, promoting diversity in the dataset. Full implementation details are provided in Appendix A.

Answer Verification To ensure correctness and reliability, each instance is subjected to manual

[2]https://www.anthropic.com/claude/sonnet

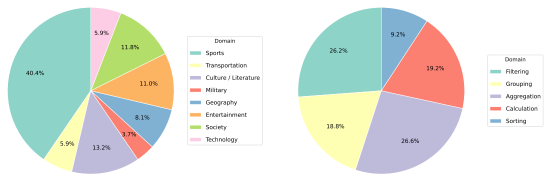


Figure 3: Domain distribution and tabular operation distribution of HeteQA.

inspection by human annotators. Their task is to verify that the execution outcome is accurate for the corresponding query. In cases where discrepancies are found, they are responsible for correcting both the underlying code and the resulting answer.

Document Reference To support queries that integrate both tabular and textual information, we augment the instance by leveraging the associated Wikipedia document. Specifically, certain entities within the query are replaced with reference-based formulations by the human annotators. For example, the query *"Which driver . . ."* can be rephrased as *"What is the nationality of the driver . . ."*. This entity substitution can either modify the subject of the question and its corresponding answer or alter the query phrasing while preserving the original answer. The annotation guidelines and annotator profiles are detailed in Appendix B.

4.2 Discussion

Each data instance in HeteQA is composed of a query, its corresponding answer, the executable SQL sentence, and the execution-derived answer. Through our data collection pipeline, we construct 304 high-quality examples whose answers are grounded in both single-source (82%) and multi-source (18%). The resulting benchmark spans 136 distinct tables and 5314 wiki knowledge entities. To characterize the dataset, we analyze its semantic domains and the types of tabular reasoning operations. As illustrated in Figure 3, HeteQA covers 9 semantically diverse domains and encompasses 5 principal categories of tabular operations. Together, HeteQA constitutes a structurally diverse and semantically broad resource for advancing question answering over heterogeneous documents.

5 Experiments

5.1 Experimental Settings

5.1.1 Datasets.

We assess the performance of TableRAG on our curated HeteQA, as well as multiple established benchmarks spanning two settings:

HybridQA (Chen et al., 2020) A multi-hop QA dataset involving both tabular and textual information. For our evaluation, we only retain data cases with tables containing more than 100 cells.

WikiTableQuestion (Pasupat and Liang, 2015b) A TableQA dataset spanning diverse domains. The queries necessitate a range of data manipulation operations, including comparison, aggregation, etc.

5.1.2 Implementation Details

In the text retrieval process, we employ the BGE-M3 series models (Chen et al., 2024a,b). During recall, we retain the top 30 candidates, from which the top 3 are subsequently selected via reranking. To manage large inputs, the text is chunked into segments of 1000 tokens, with a 200-token overlap between consecutive chunks. The iterative loop is bounded by a maximum of 5 iterations. For backbone LLMs, we utilize Claude-3.5-Sonnet as a representative closed-source LLM, while Deepseek-V3, Deepseek-R1 (Guo et al., 2025), and Qwen-2.5-72B (Yang et al., 2024) serve as the open-source counterparts. A consistent backend is maintained for all modules in the online iterative reasoning process. We use accuracy as the evaluation metric, assessed by Qwen-2.5-72B, which yields a binary score of 0 or 1. The prompt is provided in Appendix G, and the results evaluated using the exact match metric are presented in Appendix D.

5.1.3 Baselines

We evaluate the performance of TableRAG by benchmarking it against three distinct baseline methodologies: (1) Direct answer generation with LLMs. (2) NaiveRAG, which processes tabular data as linearized Markdown formatted texts and subsequently applies a standard RAG pipeline. (3) ReAct (Yao et al., 2023), a prompt based paradigm to synergize reasoning and acting in LLMs with external knowledge sources. (4) TableGPT2 (Su et al., 2024) employs a Python-based execution module to generate code (e.g., Pandas) for answer derivation within a simulated environment. The detailed implementation of these baseline methods

Method	**Backbone**	**HybridQA**	**WikiTQ**	**HeteQA**		
		-	-	**Single-Source**	**Multi-Source**	**Overall**
Direct	Claude-3.5	9.84	6.21	10.68	8.65	10.00
	DeepSeek-R1	24.42	12.20	3.40	13.46	6.77
	DeepSeek-V3	14.75	10.39	6.80	28.85	14.19
	Qwen-2.5-72b	11.47	7.37	4.85	12.50	7.42
NaiveRAG	Claude-3.5	20.28	82.60	33.20	40.35	34.54
	DeepSeek-V3	26.56	75.40	33.60	45.61	35.85
	Qwen-2.5-72b	22.62	66.33	23.07	36.84	25.66
ReAct	Claude-3.5	43.38	69.81	26.40	44.44	29.60
	DeepSeek-V3	38.36	63.40	21.14	47.39	26.07
	Qwen-2.5-72b	37.38	53.80	16.94	35.71	20.47
TableGPT2		9.51	63.40	35.60	16.67	32.24
TableRAG	Claude-3.5	47.87	**84.62**	**44.94**	40.74	44.19
	DeepSeek-V3	47.87	80.40	43.32	**51.85**	**44.85**
	Qwen-2.5-72b	**48.52**	78.00	37.65	43.96	38.82

Table 1: Performance of TableRAG compared to baseline models across multiple benchmarks, evaluated using LLM-as-Judge accuracy. "Multi-Source" indicates questions requiring both tabular and textual information, while “Single-Source” refers to questions relying on only one source type.

is provided in Appendix C and additional baseline comparisons are presented in Appendix D.

5.2 Main Result

The main results across different LLMs backbones are presented in Table 1. Several key observations emerge: (1) The ReAct framework demonstrates advantages over naive RAG on multi-source data, but exhibits degraded performance on single-source data that requires tabular reasoning. This can be attributed to context insensitivity during multi-turn reasoning. Queries that could be resolved with single SQL execution are instead decomposed into multiple subqueries. This over-fragmentation may introduce errors or incomplete information during execution (especially when operations such as filtering, aggregation are involved), potentially leading to cascading failures. (2) TableGPT2 yields acceptable results solely on single-source queries, such as WikiTQ, underscoring its limited capacity for handling multi-source queries. This reflects a lack of generalizability in heterogeneous information environments. (3) TableRAG surpasses all baselines, achieving at least a 10% improvement over the strongest alternative. Notably, it performs robustly across both single-source and multi-source data. This performance gain is attributed to the incorporation of symbolic reasoning, which enables effective adaptation to heterogeneous documents. Moreover, the consistency in performance across different LLM backbones underscores TableRAG’s architectural generality and compatibility with a broad range of backbones.

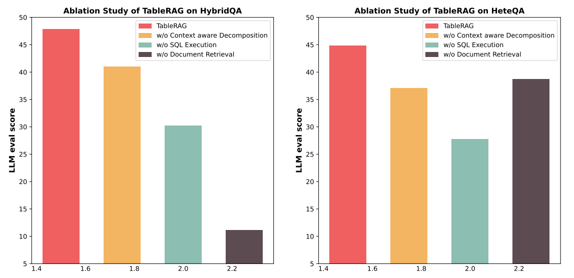


Figure 4: Ablation study on HybridQA and HeteQA benchmarks based on DeepSeek-V3 backbone.

5.3 Ablation Study

To elucidate the relative importance of each component within the TableRAG framework, we evaluate the full architecture against three ablated variants: (1) w/o Context-Sensitive Query Decomposition, where query decomposition is performed without conditioning on retrieved table schema. (2) w/o SQL Execution, which replaces the SQL programming and execution module with the markdown

Method	WikiTQ		HeteQA	
	Total Latency	**Avg. Latency / Step**	**Total Latency**	**Avg. Latency / Step**
TableRAG	13.50	5.92	24.57	7.70
ReAct	17.70	7.93	45.65	10.79

Table 2: Latency evaluation, measured in seconds.

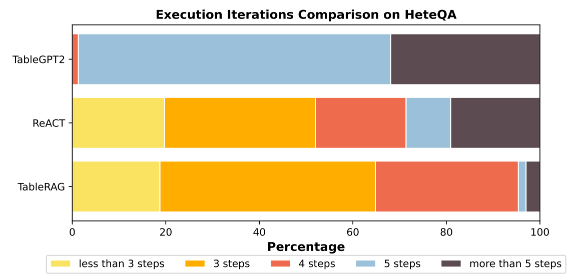


Figure 5: Comparison of the execution iterations on HeteQA between TableRAG, ReAct and TableGPT2.

table format. (3) w/o Textual Retrieval, which operates solely through table-based SQL execution, without leveraging textual resources such as Wikipedia documents. The results are summarized in Figure 4. All the modules contribute to the overall performance of TableRAG, though their relative impact varies across benchmarks. On HybridQA, document retrieval proves particularly critical, due to its emphasis on the extraction of entity-centric or numerical cues. Conversely, for HeteQA, SQL execution proves more influential, as the queries involve nested operations that benefit from SQL-based symbolic reasoning. These findings highlight the complementary design of TableRAG's textual retrieval and program-executed reasoning components.

6 Efficiency

6.1 Execution Dynamics

We evaluate the efficiency of TableRAG by examining the distribution of its execution iterations, as illustrated in Figure 5. Execution lengths are grouped into four categories: fewer than 3 steps, 3–5 steps, exactly 5 steps, and more than 5 steps. Among the evaluated methods, TableGPT2 demonstrates the highest average number of execution steps, with a modal value centered around five. In contrast, TableRAG consistently requires fewer steps, resolving approximately 63.55% of instances in fewer than five steps and an additional 30.00% precisely within five, with only a marginal proportion of cases remaining unsolved under the given iteration constraints. While ReAct exhibits a comparable distribution in execution steps, its overall performance remains markedly inferior to that of TableRAG. These results suggest that TableRAG achieves both superior efficiency in execution and outstanding reasoning accuracy. It is attributed to the incorporation of SQL-based tabular reasoning.

6.2 Latency

We assess the latency performance using Qwen-72B-Instruct as a consistent backbone model. As shown in Table 2, TableRAG incurs lower latency compared to ReAct, in both average time per step and total time efficiency. This is mainly due to: (1) TableRAG executes fewer iterations, and employs symbolic SQL execution, which is computationally efficient. (2) ReAct performs reasoning at every step, and consumes more steps before reaching a final decision, thereby incurring greater time overhead.

7 Analysis

We provide a comprehensive analysis of TableRAG in this section, with additional results presented in Appendix E.

7.1 Error Analysis

In addition to evaluating the overall performance of TableRAG against established baselines, we performed a detailed error analysis to characterize the nature of prediction failures. Broadly, the incorrect outputs fall into two primary categories: (1) reasoning failures, attributable to errors in SQL execution or flawed intermediate query decomposition, and (2) task incompletion, typically manifesting as refusals to answer or termination upon exceeding the maximum iteration limit. The prediction distribution is shown in Figure 6. Notably, TableGPT2 exhibits the highest frequency of such failures, largely due to its limited capacity to integrate contextual cues from the wiki documents. This constraint frequently results in the model either explicitly re-

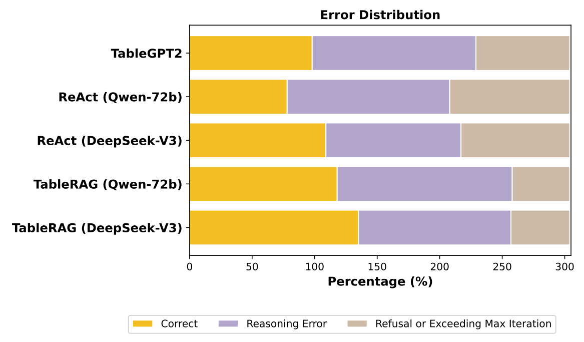


Figure 6: Error analysis of TableRAG, TableGPT2 and ReAct with DeepSeek-V3 and Qwen-2.5-72b as backbones on HeteQA.

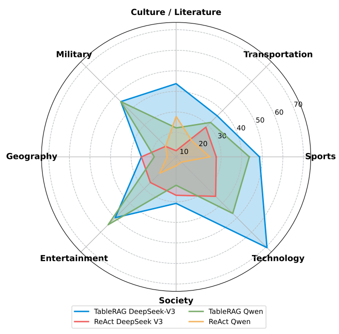


Figure 7: Performance distribution of TableRAG and ReAct across different domains.

fusing to respond or acknowledging its inability to do so. In contrast, ReAct, which lacks mechanisms for context-aware query decomposition and code execution simulation, often engages in unnecessarily elaborate reasoning steps for problems that could be addressed via a single structured inquiry. TableRAG demonstrates the lowest failure rate among the methodologies assessed. Its consistent ability to yield valid responses within five iterations highlights the efficacy of its design — particularly its use of context-aware query decomposition and selective SQL-based execution planning.

7.2 Prediction across Domains

Figure 7 presents a comparative evaluation of TableRAG, instantiated with various backbone LLMs, against the ReAct framework across various domains. The results reveal that TableRAG consistently outperforms ReAct in the majority of domains, demonstrating its effectiveness in heterogeneous document question answering. Only certain domains, such as Culture, exhibit comparatively weaker performance on TableRAG with Qwen backbone. A closer inspection of the data distribution suggests that this degradation may stem from the sparsity of domain-specific instances.

8 Related Work

8.1 Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) has emerged as a robust paradigm for mitigating hallucination (Zhang et al., 2023a) and enhancing the reliability of Large Language Models (LLMs) generated responses (Lewis et al., 2020; Guu et al., 2020). The RAG approaches retrieve from the knowledge base, and the most relevant document chunks are subsequently incorporated into the generation process (Gao et al., 2023; Zhu et al., 2024; Borgeaud et al., 2022). However, this straightforward retrieval process often yields noisy chunks that may lack critical details, thereby diminishing the quality of the subsequent generation. Recent advancements have thus focused on task-adaptive retrieval mechanisms. Notable frameworks in this regard include Self-RAG (Asai et al., 2023), RQ-RAG (Chan et al., 2024), etc. Despite these innovations, RAG still faces challenges when dealing with heterogeneous contexts (Satpute et al., 2024).

8.2 Table Reasoning via Large Language Models

Table reasoning refers to the development of a system that provides responses to user queries based on tabular data (Lu et al., 2025). The mainstream approaches to table reasoning can be broadly classified into two categories. The first category revolves around leveraging LLMs through prompt engineering. For instance, Tab-CoT (Jin and Lu, 2023) applies chain-of-thought (CoT) reasoning to establish a tabular structured reasoning process. Similarly, Chain-of-Table (Wang et al., 2024) extends the CoT methodology to the tabular setting, enabling a multi-step reasoning process for more complex table-based queries. The second category involves utilizing programs to process tabular data. Tabsqlify (Nahid and Rafiei, 2024) employs a text-to-SQL approach to decompose tables into smaller, contextually relevant sub-tables. DATER (Ye et al., 2023a) adopts a few-shot prompting strategy to reduce large tables into more manageable sub-tables, using a parsing-execution-filling technique that generates intermediate SQL queries. BINDER (Cheng et al., 2022; Zhang et al., 2023b) integrates both Python and SQL code to derive answers from tables. InfiAgent-DABench (Hu et al., 2024) utilizes an LLM-based agent that plans, writes code, interacts with a Python sandbox, and synthesizes results to solve table-based questions.

9 Conclusion

We address the limitations of existing RAG approaches in handling heterogeneous documents that combine textual and tabular data. Current approaches compromise the structural integrity of tables, resulting in information loss and degraded performance in global, multi-hop reasoning tasks. To overcome these issues, we introduce TableRAG, an SQL-driven framework that integrates textual understanding with precise tabular manipulation. To rigorously assess the capabilities of our approach, we also present a new benchmark HeteQA. Experimental evaluations across public datasets and HeteQA reveal that TableRAG significantly outperforms existing baseline approaches.

Limitations

While TableRAG demonstrates strong performance, several limitations merit consideration: 1. The effectiveness of TableRAG is closely tied to the capabilities of the underlying LLMs. Our implementation leverages high-capacity models such as Claude, DeepSeek-v3, and Qwen-72B-Instruct, which possess strong generalization abilities. Smaller models that lack specialized instruction tuning may exhibit a marked degradation in performance. This suggests that achieving competitive results may necessitate substantial computational resources. 2. The HeteQA benchmark is restricted to English. This limitation arises from the difficulty in curating high-quality heterogeneous sources across multiple languages. As a result, cross-lingual generalization remains unexplored. In future work, we aim to extend HeteQA to a multilingual setting, thereby broadening the applicability and robustness of our evaluation framework.

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K Reddy. 2024. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *arXiv preprint arXiv:2407.05952*.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024a. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024b. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Paulo Finardi, Leonardo Avila, Rodrigo Castaldoni, Pedro Gengo, Celio Larcher, Marcos Piau, Pablo Costa, and Vinicius Caridá. 2024. The chronicles of rag: The retriever, the chunk and the generator. *arXiv preprint arXiv:2401.07883*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Ziqi Jin and Wei Lu. 2023. Tab-cot: Zero-shot tabular chain of thought. *arXiv preprint arXiv:2305.17812*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Weizheng Lu, Jing Zhang, Ju Fan, Zihao Fu, Yueguo Chen, and Xiaoyong Du. 2025. Large language model for table processing: A survey. *Frontiers of Computer Science*, 19(2):192350.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. *arXiv preprint arXiv:2404.10150*.

Panupong Pasupat and Percy Liang. 2015a. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Panupong Pasupat and Percy Liang. 2015b. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Ankit Satpute, Noah Gießing, André Greiner-Petter, Moritz Schubotz, Olaf Teschke, Akiko Aizawa, and Bela Gipp. 2024. Can llms master math? investigating large language models on math stack exchange. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 2316–2320.

Hesam Shahrokhi, Amirali Kaboli, Mahdi Ghorbani, and Amir Shaikhha. 2024. Pytond: Efficient python data science on the shoulders of databases. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 423–435. IEEE.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, et al. 2024. Tablegpt2: A large multimodal model with tabular data integration. *arXiv preprint arXiv:2411.02059*.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2023. Tap4llm: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. *arXiv preprint arXiv:2312.09039*.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023a. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *arXiv preprint arXiv:2301.13808*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023b. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023a. Siren's song in the ai ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. 2023b. Reactable: Enhancing react for table question answering. *arXiv preprint arXiv:2310.00815*.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*.

Yinghao Zhu, Changyu Ren, Shiyun Xie, Shukai Liu, Hangyuan Ji, Zixiang Wang, Tao Sun, Long He, Zhoujun Li, Xi Zhu, et al. 2024. Realm: Rag-driven enhancement of multimodal electronic health records analysis via large language models. *arXiv preprint arXiv:2402.07016*.

A HeteQA

A.1 Table Collection

To enable complex reasoning over tabular structures, we curate a subset of extensive tables from the Wikipedia-based corpus. Specifically, we retain

table	List_of_Australian_films_of_2012_0
query	Who wrote and starred the comedy film released in the second half of 2012 (July-December) that had the highest number of cast members in the List of Australian films of 2012?
sql_query	Which comedy film released in the second half of 2012 (July-December) had the highest number of cast members in the List of Australian films of 2012?

sql

```
SELECT
  title,
  LENGTH(cast_subject_of_documentary) - LENGTH(
    REPLACE(
      cast_subject_of_documentary, ',',
      ''
    )
  ) + 1 AS cast_count
FROM
  `list_of_australian_films_of_2012_0_sheet1`
WHERE
  genre LIKE '%Comedy%'
  AND (
    release_date LIKE '%July%'
    OR release_date LIKE '%August%'
    OR release_date LIKE '%September%'
    OR release_date LIKE '%October%'
    OR release_date LIKE '%November%'
    OR release_date LIKE '\ufffdcember%'
  )
ORDER BY
  cast_count DESC
LIMIT
  1;
```

sql_ans	Kath & Kimderella
answer	Riley, Turner, and Magda Szubanski

Table 3: An example of HeteQA.

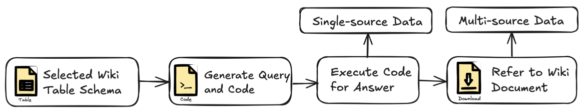


Figure 8: The dataset construction pipeline of HeteQA.

only tables containing more than 20 rows and at least 7 columns. This filtering process reduced the initial collection from 15,314 tables to 1,345 candidates that meet the criteria for structural richness. To further enhance query diversity and reduce redundancy in the dataset, we apply a deduplication step based on schema similarity. In cases where multiple tables share identical column structures — for example, the entries for `1947 BAA draft` and `1949 BAA draft` — only a single representative instance is preserved. This procedure yielded a final dataset comprising 155 unique tables.

A.2 Data Collection

As illustrated in Figure 8, each data instance is constructed through a three-stage pipeline, facilitated via the Claude 3.7 Sonnet API[3].

To construct the dataset, we prompt the LLM to generate SQL or `pandas` code associated with queries conditioned on a provided table schema. The generated code is then executed against the table to obtain the corresponding answer. Instances that fail to produce an executable result are discarded. To promote diversity within the dataset and avoid redundancy, we eliminate any instances exhibiting duplicate queries or identical answers. This filtering is performed using regular expressions to detect lexical or semantic repetition. Additionally, we discard cases where the execution result is ambiguous, such as queries seeking the top-ranked entity when multiple candidates are tied. Following automated filtering, all remaining instances undergo a manual verification process. Two human annotators independently assess each query, the corresponding code, and the resulting answer. In cases of inconsistency or error, annotators revise the SQL or Python code and correct the associated answer to ensure accuracy and coherence.

B Guidelines for HeteQA Annotations

B.1 Annotator Profiles

The two annotators are volunteers whose native language is Chinese and who possess fluent English proficiency. Both annotators are professional software engineers with substantial experience in SQL and Python programming.

B.2 Guidelines for Answer Verification

Objective
Your task is to verify whether the provided answer correctly and completely addresses the natural language query, based on the associated code and table. You will also identify and handle ambiguous or non-unique cases.

Annotation Steps
1. Verify the Answer
Read the natural language query and examine the accompanying Excel table, SQL statements, and Python (pandas) code. Verify whether the output aligns with the semantics of the original query and the underlying data. To this end, you are permitted to execute multiple operations on the provided Excel files.
2. Correction Tasks (if necessary)
If the answer is incorrect, modify both the code and the answer so that they correctly fulfill the query. Ensure the modified code is minimal, clean, and logically sound.
3. Ambiguity and Tie Cases
In instances where the query yields no unique resolution - whether due to tied outcomes, under specified conditions, or semantic ambiguity in the formulation - you can employ principled strategies to ensure meaningful processing:
- Option A: Modify the query to resolve the ambiguity (e.g., make clarification).
- Option B: If the ambiguity is irreparable or the answer depends on arbitrary choices, discard the case.

Additional Guidelines
Maintain consistency between the query, code, and answer. Ensure your corrections do not introduce new ambiguities or assumptions not grounded in the table or query.

B.3 Guidelines for Document Reference

Objective
Your task is to add an additional reasoning hop to the original query using the provided Wikipedia entities and content. This will help transform the original query into a more complex, multi-hop question that requires deeper reasoning.

Input Data

[3] https://www.anthropic.com/claude/sonnet

Each data case consists of:

- Original Query
- Answer
- A set of Wikipedia entities relevant to the query or the answer
- Corresponding Wikipedia content for each entity

Task Overview
For each data case, check if the original answer is mentioned in the provided Wiki entities:
- If not mentioned or ambiguous (e.g., "Jordan" for both basketball star and sports brand), do not change the query and answer. Just mark the case as "No modification needed".
- If the answer entity exists in the Wikipedia content, perform the following steps:

1. Identify key factual descriptions about the answer entity from its Wiki content.
2. Add a reasoning hop to the original query that leads to the answer via this key fact.
3. Generate two candidate (Query, Answer) pairs by rewriting the query to incorporate this extra reasoning step and updating the answer accordingly.

Example
Original Query: Which album takes first place on the Billboard leaderboard in 2013?
Original Answer: ArtPop
Wiki Entity: ArtPop (album)
A key description: It was released on November 6, 2013, by Streamline and Interscope Records.
Modified Query Candidates:
Who released the album that takes the first place on the Billboard leaderboard in 2013? $\rightarrow$ Answer: Streamline and Interscope Records

C Implementation Details

This section provides a detailed account of the implementation procedures for the baseline methodologies and TableRAG to ensure a fair and reproducible evaluation.

C.1 ReAct

For the ReAct framework, we preprocess tabular data by converting it into markdown-formatted plain text. To ensure consistency in experimental conditions, we adopt the same chunking and retrieval configurations as those employed in our TableRAG model. We build upon the publicly available ReAct implementation[4]. The framework addresses user queries through an iterative reasoning loop consisting of `Thought`, `Action`, and `Observation` steps, culminating in a final answer via the `Finish` operation. The max iteration is set to 5, the same as TableRAG.

C.2 TableGPT2

We evaluate TableGPT2 using its officially released TableGPT Agent implementation [5]. As specified in its API documentation, we provide both the document content and the tabular data as inputs to the `HumanMessage` class, ensuring adherence to the intended usage of the model:

```python
from typing import TypedDict
from langchain_core.messages import HumanMessage

class Attachment(TypedDict):
    """Contains at least one dictionary with the key filename."""
    filename: str

attachment_msg = HumanMessage(
    content="",
    # Please make sure your iPython kernel can access your filename.
    additional_kwargs={"attachments": [Attachment(filename="titanic.csv")]},
)
```

C.3 TableRAG

In this section, we provide additional details about the offline stage in TableRAG. To mitigate the potential effects of errors during this stage, we apply various engineering techniques to minimize preprocessing mistakes:

- Missing Value Imputation: we systematically assign default missing column names or missing elements to preserve table integrity.
- Column Type Mapping: we perform data type inference for each column based on type consistency across its values.
- Representative Sample Selection: during schema extraction, we carefully control the length of selected samples while promoting value diversity.

TableRAG is designed to degrade gracefully in the presence of offline errors (most commonly happens in the tabular processing phase). TableRAG

[4]https://github.com/ysymyth/ReAct
[5]https://github.com/tablegpt/tablegpt-agent

Method	Backbone	HybridQA	WikiTQ	HeteQA
ReAct	Claude-3.5	41.4	72.6	19.4
	DeepSeek-V3	35.4	58.4	15.5
	Qwen-2.5-72b	29.5	52.6	12.5
TableGPT2		11.8	66.8	15.1
TableRAG	Claude-3.5	42.6	86.4	27.8
	DeepSeek-V3	33.1	79.8	21.7
	Qwen-2.5-72b	44.6	76.0	25.5

Table 4: Performance of TableRAG compared to baseline models across multiple benchmarks, measured by exact match.

Method	WikiTQ	HybridQA
ReAct	53.80	37.38
TableRAG	78.00	48.52
TAP4LLM	-	23.28
H-Star	67.35	-

Table 5: Supplementary baseline comparison with Qwen2.5-72B-Instruct as backend.

bypasses the SQL execution pathway and leverages the text retrieval module to extract and reason over markdown-formatted table content.

D Experiments

D.1 Exact Match Results

We present the experimental results with exact match as evaluation metrics in Table 4.

D.2 Supplementary Baseline Comparison

We further compare TableRAG with additional baseline methods that target specific domain scenarios:

- TAP4LLM[6] (Sui et al., 2023): a comprehensive pre-processing toolkit designed to enhance in Tabular reasoning with LLMs. It employs three main strategies: table sampling, table augmentation and table packing.

- H-Star[7] (Abhyankar et al., 2024): a hybrid approach that operates in two stages: Table Extraction, which identifies and isolates relevant table sections based on the query, and Adaptive Reasoning, which dynamically selects between symbolic and semantic reasoning strategies depending on the question type.

[6]https://github.com/Y-Sui/archive_code
[7]https://github.com/nikhilsab/H-STAR

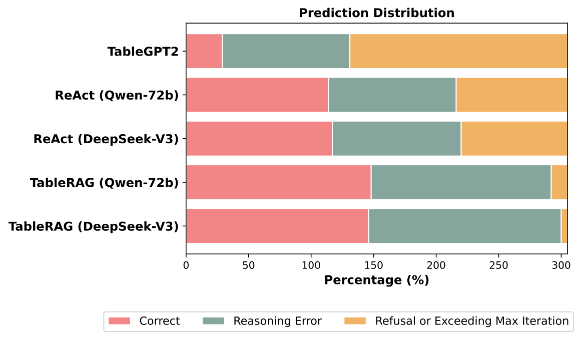


Figure 9: Prediction distribution of TableRAG, TableGPT2 and ReAct on HybridQA.

The results are summarized in Table 5. As shown, TAP4LLM performs worse than both TableRAG and ReAct, our primary baseline, indicating that agentic frameworks are generally more effective than pipeline-style approaches. In the TableQA scenario, TableRAG also surpasses H-Star. Overall, TableRAG consistently delivers strong performance, further supporting our claims.

E Analysis

We present a series of auxiliary analyses conducted on both our proposed benchmark, HETEQA, and the publicly available HYBRIDQA dataset. These analyses offer further insight into the behavior and limitations of TableRAG beyond the primary evaluation metrics.

E.1 Analysis on HybridQA

We extend our investigation to the HYBRIDQA dataset by examining the performance distribution on data domains and conducting a detailed error

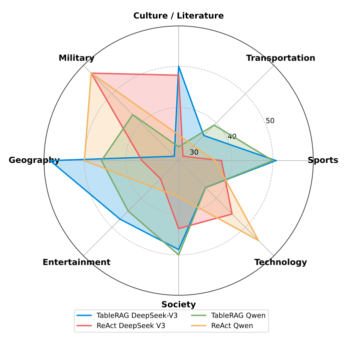


Figure 10: Model performance of different domains on HybridQA.

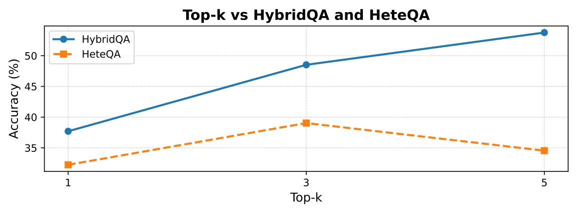


Figure 11: Hyperparameter top-k analysis of TableRAG on HeteQA.

analysis, using DEEPSEEK-V3 as the backbone model. The results, summarized in Figures 10 and 9, reveal patterns consistent with those observed on HETEQA. This parallel further validates the generality of our observations across HeteQA settings.

E.2 HyperParameter Analysis

To elucidate the influence of the top-k retrieval parameter on the performance of TableRAG, we undertook a systematic sensitivity analysis. While our main experimental setup fixed $k = 3$, we expanded our investigation on the HeteQA dataset, utilizing DeepSeek-V3 as the retrieval backbone, and varied k across the set $1, 3, 5$. The resulting performance metrics are illustrated in Figure 11.

Our observations reveal distinct behaviors across benchmarks with respect to the choice of k. Notably, HybridQA exhibits superior performance at higher k values. This effect is plausibly attributable to the lack of deduplication within its tables, whereby overlapping or similar information from multiple tables or Wikipedia documents contributes constructively during retrieval. In aggregate, setting $k = 3$ strikes an effective balance, yielding robust performance across both benchmarks while maintaining computational efficiency. This efficiency gain arises from the direct relationship between the top-k retrieval size and the subsequent LLM context length, underscoring the practical importance of this hyperparameter in optimizing the trade-off between accuracy and resource consumption.

F Check List

Harmful information And Privacy We propose a new RAG solution to address multi-hop problems related to heterogeneous data, without involving any harmful information or privacy. In addition, we provide a heterogeneous benchmark containing tables and texts sourced from Wikipedia. All of the data is publicly available, contains no personal information, and involves no harmful content.

License and Intend We provide the license we used here:

- Claude 3.5 Sonnet (`https://www.anthropic.com/legal/aup`)
- Qwen2.5-72B-Instruct (`https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/LICENSE`)
- DeepSeek-V3 (`https://huggingface.co/deepseek-ai/DeepSeek-V3/blob/main/LICENSE-MODEL`)
- TableGPT Agent (Apache License 2.0) and TableGPT2-7B (`https://huggingface.co/tablegpt/TableGPT2-7B/blob/main/LICENSE`)

Our use of these existing artifacts was consistent with their intended use.

Documentation of the artifacts We propose a novel Retrieval-Augmented Generation (RAG) framework that integrates traditional document retrieval with structured data querying via SQL, aiming to enhance performance in table-related question answering tasks. The framework comprises two fundamental stages: offline database construction and online interactive reasoning. Compared to conventional document-centric RAG approaches, our architecture offers additional reliable SQL execution results—when table fragments are involved in the retrieval process. This supplementary source mitigates the limitations of generative models in handling structured tabular data.

To facilitate a more comprehensive evaluation of our framework, we further construct a heterogeneous benchmark named HeteQA. This benchmark aims to evaluate the capability to handle multi-hop reasoning tasks across heterogeneous documents. The benchmark instances are initially generated by LLMs, and then rigorously validated by experienced programmers and database administrators to ensure correctness and realism.

G Prompts

The prompts for TableRAG are presented in this section.

.

Prompt Template for SQL generation

```
# Multi-Hop Table Reasoning Query Generator

## Task
Generate a genuine multi-hop reasoning query based on the provided markdown table, along with SQL and pandas solutions in a structured JSON format.

## Input
A markdown formatted table schema.

## Output Requirements
Provide exactly ONE multi-hop reasoning query that:
- Requires sequential analytical operations where each step depends on the previous result
- Cannot be broken down into separate independent questions
- Is solvable using both SQL and pandas
- Is relevant to the data domain in the table
- Mention the table name in query to indicate the source table file

## Operations to Consider in Your Sequential Reasoning Chain
- Group or Aggregate
- Filtering subsets
- Calculating percentages or ratios between groups
- Comparing specific subgroups
- Rank or order
- Finding extremes (max/min) of aggregated values
- Computing difference or sum
- Finding correlations

## Example Operations Combinations
- Filter → Group → Rank in groups
- Group → Sum → Compare
- Filter → Calculate percentage → Rank
- Group → Aggregate → Filter on aggregate

## Guidance for True Multi-Hop Queries
A proper multi-hop query requires sequential operations where each step builds on the result of the previous step. For example:

GOOD (True multi-hop): "What was the average lap time among the top 5 ranked drivers in the team which had the best average lap time?"
- This requires first finding the team with best average lap time
- Then identifying the top 5 drivers in that specific team
- Finally calculating the average lap time of just those drivers

BAD (Separable questions): "Which team had the best average lap time, and what was that average among the top 5 ranked drivers?"
- This could be answered as two separate questions

Format your response as a single JSON object with this structure:
{
"query": "Clear natural language question requiring true multi-hop reasoning",
"operations_used": ["List operations used, such as: filtering, aggregation, grouping, sorting, etc."],
"sql_solution": "Complete executable SQL query that solves the question",
"pandas_solution": "Complete executable pandas code that solves the question",
"result_type: "The type of the result, must be either number or entity."
}
Ensure your query truly requires chained reasoning where later steps must use results from earlier steps and generates one answer.
```

Prompt Template for Query Decomposition ???

```
tools = [{
    "type": "function",
    "function": {
        "name": "solve_subquery",
        "description": "Return answer for the decomposed subquery",
        "parameters": {
            "type": "object",
            "properties": {
                "subquery": {
                    "type": "string",
                    "description": "The subquery to be solved"
                }
            },
            "required": [
                "subquery"
            ],
            "additionalProperties": False
        },
        "strict": True
    }
}]
```

Next, You will complete a table-related question answering task. Based on the provided materials such as the table content (in Markdown format), you need to analyze the Question. And try to decide whether the Question should be broken down into subquerys. After you have collected sufficient information, you need to generate comprehensive answers.
You have a "solve_subquery" tool that can execute SQL-like operations on the table data. It accepts natural language questions as input.

Instructions:
1. Carefully analyze the user query through step-by-step reasoning.
2. If the query requires multiple pieces of information, more than the given table content:
- Decompose the query into subqueries
- Process one subquery at a time
- Use "solve_subquery" tool to retrieve answers for each subquery
3. If a query can be answered by table content, do not decompose it. And directly put the origin query into the "solve_subquery" tool.
The "solve_subquery" tool can solve complex subquery on table via one tool call.
4. Generate exactly ONE subquery at a time.
5. Write out all terms completely - avoid using abbreviations.
6. When you have sufficient information, provide the final answer in this format:
<Answer>: [your complete response]

Table Content: {table_content}
Question: {query}
Please start!

Prompt Template for Intermediate Answer Reasoning

```
You are about to complete a table-based question answering task using the following two types of
reference materials:

# Content 1: Original content (table content is provided in Markdown format)
{docs}

# Content 2: NL2SQL related information and SQL execution results in the database
# the user given table schema
{schema}

# SQL generated based on the schema and the user question:
{nl2sql_model_response}

# SQL execution results
{sql_execute_result}

Please answer the user's question based on the materials above.
User question: {query}

Note:
1. The markdown table content in Content 1 may be not complete.
2. You should cross-validate the given two materials:
- if the answers are same, you may directly output the answer.
- If the SQL shows error, such as "SQL execution results", try to answer solely based on Content 1.
- If the two material shows conflict, carefully evaluate both sources, explain the discrepancy, and
provide your best assessment.
```

Prompt Template for Answer Evaluation

We would like to request your feedback on the performance of the AI assistant in response to the user question displayed above according to the gold answer. Please use the following listed aspects and their descriptions as evaluation criteria:
- Accuracy and Hallucinations: The assistant's answer is semantically consistent with the gold answer; The numerical value and order need to be accurate, and there should be no hallucinations.
- Completeness: Referring to the reference answers, the assistant's answer should contain all the key points needed to answer the user's question; further elaboration on these key points can be omitted.

Please rate whether this answer is suitable for the question. Please note that the gold answer can be considered as a correct answer to the question.

The assistant receives an overall score on a scale of 0 OR 1, where 0 means wrong and 1 means correct.
Dirctly output a line indicating the score of the Assistant.

PLEASE OUTPUT WITH THE FOLLOWING FORMAT, WHERE THE SCORE IS 0 OR 1 BY STRICTLY FOLLOWING THIS FORMAT: "[[score]]", FOR EXAMPLE "Rating: [[1]]":
<start output>
Rating: [[score]]
<end output>

[Question]
question

[Gold Answer]
golden

[The Start of Assistant's Predicted Answer]
{gen}