An Implementation and Analysis of the ARM System

Apostolos Koukouvinis
National and Kapodistrian University of Athens
Athens, Greece
tolkoukouvinis@di.uoa.gr

Abstract

While Large Language Models (LLMs) excel at understanding complex queries, their utility in real-world systems is often undermined by their tendency to hallucinate, by feeling pretty confident about their answers while they are completely wrong, particularly when faced with private or domain-specific information. Retrieval-Augmented Generation (RAG) addresses this by grounding LLM outputs in factual data. However, the effectiveness of RAG hinges on its initial retrieval step, which often falls short when questions require synthesizing information from multiple, interconnected sources, as it lacks awareness of the data's underlying organization. This report details the reproduction of ARM (Alignment-Oriented LLM-based Retrieval Method), a system that reframes retrieval not as a single search action, but as a multi-step reasoning process. The core principle of ARM is to deeply align the user's query with both the content and the organizational structure of the available data collection. This is achieved through a multi-stage pipeline orchestrated by an LLM. The process initiates with information alignment, where the LLM decomposes the query and uses constrained decoding to find candidate data objects that match the query's key concepts. It then performs a structure alignment, a query-agnostic expansion based on the data collection relations, where it maximizes the compatibility of the objects using a Mixed-Integer Program to identify an optimally connected and relevant subgraph of data. This step allows ARM to discover necessary bridging information that simple semantic search would miss. The entire process is embedded within a beam search to explore multiple reasoning paths, culminating in a self-verification and aggregation stage where the LLM makes a final, reasoned selection of evidence. Our reproduction confirms that this alignment-oriented approach enables a more effective "retrieve-all-at-once" solution, outperforming standard and agentic RAG baselines on complex question-answering benchmarks involving both text and tables.

1 Introduction

At its core, **Information Retrieval (IR)** addresses a fundamental challenge: given a user query, find the most relevant documents from a vast collection. Traditional IR systems relied heavily on lexical similarity, employing statistical methods like BM25[42] to rank documents based on keyword frequency, or techniques like MinHash LSH [63] to efficiently identify near-duplicate texts. While highly efficient, these methods are constrained by their inability to capture deeper semantic meaning; they struggle with synonyms, paraphrasing, and conceptual understanding, often failing when the query's vocabulary does not precisely match that of the document. This limitation paved the way for a paradigm shift with the advent of Transformer [49] architectures and dense retrieval [30]. By leveraging the encoder part of the Transformer, models learned to map text into a low-dimensional vector space, or "embedding space," where semantic similarity is represented by geometric proximity[4, 15, 31]. The **dense retrieval** pipeline became the new standard: first, during an offline phase, the entire corpus is encoded into embeddings and stored in a specialized vector index. Then, at retrieval time, the user's query is embedded, and a similarity search (e.g., cosine similarity) is performed against the index to find the most semantically relevant objects[51]. Parallel to these advancements in encoders, work on the decoder side of the Transformer architecture gave rise to **Large Language Models (LLMs)**. These models exhibit remarkable capabilities in reasoning and answering complex user questions. However, their power is accompanied by significant drawbacks, including a propensity for **hallucination**, reliance on outdated knowledge from their training data, and non-transparent, untraceable reasoning processes[50]. **Retrieval-Augmented Generation (RAG)** emerged as a powerful solution, synergistically merging the intrinsic knowledge of LLMs with the factual grounding of external databases[17]. By retrieving relevant information and providing it as context, RAG enhances the accuracy and credibility of the generated output, allowing for continuous knowledge updates and the integration of domain-specific information. The evolution of embeddings did not stop at text. The field of **multimodal learning** enabled models to project different data types—such as text, images, and audio—into a shared or coordinated representation space[19, 41]. This inevitably led to the development of **Multimodal LLMs** [57] and, consequently, **Multimodal RAG** [1], where a natural language query can retrieve information from a heterogeneous collection of documents and tables. More recently, research has explored fundamentally new ways to integrate retrieval with generation by modifying the LLM's parameters. One such approach is **Generative Information Retrieval**[37, 59], where the traditional external index is replaced by the parametric knowledge of the LLM itself. During an offline training phase, the model "memorizes" the corpus, and retrieval is performed at inference time by directly generating the identifier of a relevant document. Another recent technique involves **online parameter modification**[46]. Here, individual data objects are associated with small, lightweight adapters (e.g., LoRA[22]). When an object is retrieved, its corresponding adapter is dynamically merged with the base LLM's parameters, temporarily infusing the model with specialized, fine-grained knowledge for the current query.

The final family of RAG systems in which we will focus are those **Synergizing RAG and Reasoning**[18]. It reframes the retrieval process from a single lookup into an iterative, dynamic dialogue between the LLM and the data. This approach aims to dynamically decompose problems, generate intermediate hypotheses, and iteratively refine solutions. This synergy manifests in two primary directions: **Retrieval-Augmented Reasoning**, where retrieved information grounds the model's deductive steps, and

Reasoning-Augmented Retrieval [18, 29], where the model's reasoning actively improves the quality and relevance of what is retrieved.

The ARM[7] system, which is the focus of this report, is an example of Reasoning-Augmented Retrieval. It employs an LLM to guide a multi-stage pipeline designed for jointly retrieving information from heterogeneous sources like text and tables. The process is characterized by the LLM dynamically switching between unconstrained generation for high-level reasoning (e.g., query decomposition) and constrained decoding for aligning its outputs with the actual content of the data collection. This allows ARM to first explore the data, then use techniques like Mixed-Integer Programming to re-rank candidates based on their structural relationships, and finally, perform a self-verification step to finalize the selection of evidence.

Contributions

In this report, we provide a comprehensive technical account of our reproduction of the ARM system. Our contributions are as follows:

- We provide a concise review of the landscape of Reasoning-Augmented Retrieval systems to contextualize ARM's approach.
- We detail the implementation of several baseline retrieval methods, utilizing state-of-the-art models and emphasizing efficient design patterns.
- We present a comprehensive breakdown of our implementation of the ARM retrieval pipeline, component by component.
- We conduct a series of ablation studies to analyze the contribution of each module within the ARM system and determine an optimal configuration.
- We perform an extensive experimental evaluation, benchmarking our ARM implementation against the baselines on datasets that require complex retrieval from both unstructured text and structured tables.

2 Related work

Recent work has explored various ways to integrate reasoning into the retrieval process, with different methods focusing on adaptive decision-making, structured frameworks, and iterative refinement.

One line of work enables models to adaptively decide when and how to retrieve information. **SELF-RAG**[3] proposes a framework where an LLM learns to control its own retrieval process through self-reflection. It adaptively decides when to retrieve information on-demand by generating special "reflection tokens," which are then used to critique the relevance of retrieved passages and to evaluate whether its own generated output is factually supported by the evidence, allowing the model to control and select the best generation path. Similarly, **DeepRAG**[20] models retrieval-augmented reasoning as a Markov Decision Process (MDP), where at each step of a decomposed query, it makes an "atomic decision" to either retrieve external knowledge or rely on its internal parametric knowledge. Its novelty lies in a two-stage training process that first uses a binary tree search for imitation learning, and then employs a "Chain of Calibration" to fine-tune the model on preference data, specifically enhancing its ability to recognize its own knowledge boundaries. **Adaptive-RAG**[26] presents a framework that first classifies an incoming query's complexity using a smaller, trained language model and then, based on this classification, dynamically routes the query to one of three distinct strategies: no retrieval, a single-step retrieval, or an iterative multi-step retrieval process. Expanding on this, **UAR**[13] introduces a unified active retrieval framework that determines retrieval timing based on four orthogonal criteria: intent-awareness, knowledge-awareness, time-sensitivity, and self-awareness. Its novelty is in casting each criterion as an independent, plug-and-play binary classification task handled by a lightweight classifier trained on the hidden states of a fixed LLM, with these individual judgments being integrated through a predefined decision tree to make a comprehensive and efficient final decision for the retrieval prcoess.

Another family of methods introduces hierarchical or structured frameworks to orchestrate the complex interplay between reasoning and search. **LevelRAG**[62] proposes a hierarchical retrieval framework that decouples multi-hop logic planning from retriever-specific query rewriting by introducing a high-level searcher and multiple low-level searchers. The high-level searcher decomposes the complex user query into atomic queries, while each specialized low-level searcher (e.g., sparse, dense, web) independently rewrites these atomic queries using a strategy optimized for its corresponding retriever. In a similar vein, **HARIS**[23] puts forward a retrieval framework that explicitly models the interplay between reasoning and search through a two-agent hierarchy. Its retrieval process is novel in that a high-level reasoning agent acts as a dynamic query planner, generating targeted factual questions, while a dedicated low-level search agent executes these questions through an iterative, self-refining search process. A different approach, **PlanRAG**[34], introduces an iterative plan-then-retrieval framework where an LLM first generates an explicit, multi-step plan based on the question and data schema, and then executes this plan by posing queries, with the ability to iteratively re-plan if the initial strategy proves insufficient. Focusing on the verification loop, **HiRAG**[61] offers a framework with a two-tiered hierarchical retrieval strategy that first uses sparse retrieval for documents and then dense retrieval for chunks. Its core novelty is a "rethink" mechanism that operates on single-candidate retrievals; it sequentially retrieves one chunk, uses a Filter module to verify the answer, and iteratively selects a new chunk if the previous one led to an incorrect answer. Another structured approach, **OmniThink**[54], simulates a cognitive learning process through an Information Tree and a Conceptual Pool. The framework iteratively expands the hierarchical Information Tree by retrieving information on new sub-topics and then uses a reflection step to distill this new knowledge into the Conceptual Pool, which guides subsequent rounds of expansion and finally summarizing the knowledge of all steps.

A third approach focuses specifically on using the model's own generated content to iteratively refine subsequent retrieval queries. **ITER-RETGEN**[45] proposes a method that synergizes retrieval and generation in an iterative loop where the complete generated output from one iteration is used as an informative context to guide the retrieval for the next. This process allows the model to progressively refine its answer by retrieving more relevant or corrective information. In a more granular application of this idea, **ChainRAG**[24] introduces a progressive retrieval and rewriting framework to address the "lost-in-retrieval" problem. Its core novelty is a sequential process where, after answering one sub-question, the LLM uses that answer to rewrite the subsequent sub-question by explicitly completing the missing entity, a process supported by a pre-constructed sentence graph.

Finally, distinct from methods that adapt the retrieval process itself, **REASONIR**[44] concentrates on training the retriever model to be inherently better at reasoning tasks. Its main novelty lies in a synthetic data generation pipeline, REASONIR-SYNTHESIZER, designed specifically for the contrastive training of its bi-encoder retriever. This pipeline generates two types of training data to enhance retrieval capabilities: (1) varied-length queries and corresponding documents to extend the retriever's effective context length for complex, information-rich inputs, and (2) reasoning-intensive hard queries generated from high-quality seed documents. Furthermore, it introduces a novel multi-turn generation process for hard negatives, synthesizing documents that are challenging and superficially relevant to the query but ultimately unhelpful, thereby improving the retriever's discriminative power during training.

3 Background

3.1 Table representation

Representing, or serializing, a table into a format suitable for language models is an open research problem [8]. A significant body of work focuses on encoding tables into embeddings that capture their semantic meaning while preserving key structural characteristics, such as the permutation invariance of rows and columns[28]. For instance, **TABERT**[56] addresses this by first selecting a small, relevant subset of rows called a "content snapshot", linearizing each row by concatenating its column names and cell values, and then applying vertical self-attention across the row-wise representations to aggregate column information. In contrast, **TAPAS**[21] flattens the entire table into a single sequence of tokens and injects structural context by adding specialized embeddings for row, column, and numerical rank IDs to each token's representation. Building on this, **UTP**[5] adopts the TAPAS architecture but introduces a pre-training strategy that explicitly handles three different input formats—text-only, table-only, and joint table-text—to create a more versatile model for diverse downstream tasks.

On the other hand, given the capabilities of general-purpose language models, many approaches attempt to serialize tables into textual strings for direct processing[27]. Yet, similarly to unstructured text, the optimal granularity of this serialization is an unresolved problem[10]. The strategies explored in the literature vary in the granularity level they use. For instance, some systems rely solely on the table's schema, such as its name, column headers, and a few sample values [33]. Other approaches serialize each row individually, resulting in a distinct embedding for each row in the table [39], while a more granular method involves creating individual embeddings for each distinct cell value [9, 32, 35, 47, 52]. Finally, some systems enrich the serialization offline by using Large Language Models (LLMs) to generate more descriptive, knowledge-rich text from the table's contents prior to the encoding phase [6, 60].

3.2 Constrained decoding

Leveraging the powerful reasoning and memorization capabilities of modern Large Language Models (LLMs), a significant area of research has emerged in constrained generation[1]. In contrast to standard autoregressive decoding, where the next token is predicted from the entire vocabulary based on the preceding sequence, constrained decoding restricts the potential output at each step to a predefined subset of valid tokens, effectively setting the probability of all other tokens to zero. This paradigm is used to enforce specific output formats, such as a valid JSON structure or correct Python indentation, or to ensure that the generated text is grounded in an underlying data source. In the domain of **Generative Recommender Systems (Gen-RecSys)**, this approach is used to directly generate sequences of item identifiers as recommendations[14]. The LLM is prompted with user history, and its output is constrained to a vocabulary of known item IDs, ensuring that every recommendation corresponds to a valid product in the catalog. This allows for the end-to-end generation of recommendation lists without intermediate retrieval steps, relying on the model's learned understanding of user preferences and the underlying items. Building on constrained generation, **RICHES**[25] unifies retrieval and generation into a single autoregressive process. It achieves this by directly decoding retrieval keys—such as verbatim text from a corpus—by constraining the LLM's output at each step using an FM-Index[16]. This technique allows the model to seamlessly interleave unconstrained "thought" generation with constrained, verifiable evidence retrieval within a single decoding pass, enabling complex multi-hop reasoning.

3.3 Table-to-Table relevance

A challenge in multi-table retrieval is ensuring table compatibility, as retrieving a set of individually relevant tables is insufficient if they cannot be joined to answer a query. The work by Chen et al. [8] addresses this by reformulating multi-table retrieval as a join-aware re-ranking problem. Their method moves beyond standard query-table relevance by introducing a second component: **table-table relevance**, which models the joinability between candidate tables. This joinability is inferred by calculating a compatibility score between pairs of columns across different tables. The score is a composite measure that considers both column relevance—determined by the semantic similarity of schemas and the Jaccard similarity of their instances—and the likelihood of a key-foreign-key relationship. This key-foreign-key likelihood is approximated by computing the uniqueness of values within each column, thereby encouraging joins where at least one of the columns exhibits the high-uniqueness characteristic of a primary key. To synthesize these signals, the authors formulate the re-ranking task as a Mixed-Integer Program (MIP). The objective function of this MIP maximizes a unified score that incorporates coarse- and fine-grained query-table relevance alongside table-table compatibility scores. This formulation is used to select the optimal set of tables by balancing the trade-offs between individual table relevance and inter-table joinability under a series of defined constraints.

4 Baseline Implementations

In this section, we detail the implementation of the baseline systems and the ARM retriever system. We provide a concise description of the algorithms used when applicable and the information about the

[1]https://github.com/Saibo-creator/Awesome-LLM-Constrained-Decoding

technologies used. For our systems, since they are all essentially retrievers, they follow a high-level pipeline: during an offline phase, they access the data to calculate and store necessary data structures; during the inference phase, given one or more user queries, they use these pre-computed structures to retrieve relevant documents. To formalize this, since our programming language is Python, we define an abstract base class that all our retrievers implement. Following the pipeline described above, it defines two functions to be implemented: one for the offline phase, `index()`, and one for the inference phase, `retrieve()`.

The following sections offers a conceptual and technical overview of our implementation. Rather than detailing specific Python code or class interfaces, this section focuses on the core algorithms, architectural decisions, and key technologies employed in our system. For a practical guide to the codebase, we refer the reader to the comprehensive README in the project's GitHub repository. The documentation provides a complete guide for reproducing our experiments (including data download and script execution) and for adapting the framework for custom use (detailing class interfaces and providing instructions for indexing and retrieving over new datasets).

4.1 BM25 Sparse Index

The first retriever is a sparse index that utilizes the BM25 scoring function. BM25 is a ranking function used by search engines to estimate the relevance of documents to a given search query. It is based on the probabilistic relevance framework and extends the TF-IDF model by considering document length and term frequency saturation. The implementation uses Pyserini[2], a Python toolkit built on top of Anserini and Lucene. To our knowledge, it is the fastest open-source implementation of a BM25 sparse index.

4.2 Dense Index

The second retriever is a dense index that stores the embeddings of the underlying objects. The implementation is designed to, given a corpus of objects, compute their embeddings using open-source models compatible with the Hugging Face `sentence-transformers`[3] library. For efficient storage and retrieval, we make use of Faiss[4], a library developed by Facebook AI Research for efficient similarity search and clustering of dense vectors. While Faiss supports multiple specialized indexes for Approximate Nearest Neighbor (ANN) search, for our data size, the GPU implementation of its exact search index is more than sufficient for speed while offering perfect recall. To accelerate the computation of embeddings during the offline phase, we utilize `infinity`[5], a high-throughput, low-latency REST API for serving text-embedding models.

4.3 Dense Index with Reranker

This system augments the dense index with a reranking stage. For the reranker, we use a cross-encoder architecture. Unlike bi-encoders, which create independent vector representations for the query and documents, a cross-encoder processes a query and a document together as a single input. This allows for deeper attention-based interactions between their tokens, resulting in a more accurate relevance score. However, this approach is computationally expensive and cannot be applied to the entire corpus. We therefore employ a coarse-to-fine retrieval strategy: the dense index first retrieves an initial candidate set of size $K \times L$, where K is the number of final documents required and L is a scaling factor. These candidates are then reranked by the cross-encoder, and the top K are returned. For the implementation, we use the open-source `mixedbread-ai/mxbai-rerank-large-v2`[6] model, which, along with an efficient implementation provided by its authors, balances accuracy and speed, topping public leaderboards.

4.4 Dense Index with Query Decomposition

This approach first decomposes a complex query into simpler, more focused subqueries before retrieving documents from the dense index. This allows the system to capture multiple facets of the original query that might be missed by a single vector representation. In our implementation, we retain the original query alongside the generated subqueries for retrieval, as this technique improved results. After decomposition, each subquery is used to retrieve a list of documents from the dense index. The result sets are then fused using Reciprocal Rank Fusion (RRF), a simple and effective rank-based method that combines results by summing the reciprocal of each document's rank across lists, which we found to outperform naive voting based on occurances. For the decomposition step, we use open-source LLMs available on Hugging Face, with two inference providers: Ollama[7] and vLLM[8]. While Ollama is simple to use and suitable for low-volume workloads, vLLM's ability to batch requests and use PagedAttention for efficient KV cache management makes it orders of magnitude faster for high-throughput scenarios, making it better suited for benchmarks. The prompt used for decomposition consists of clear instructions and three few-shot examples.

4.5 Dense Index with Query Decomposition and Reranker

This final system combines the technologies and algorithms presented in the previous subsections. It follows the query decomposition pipeline to generate and execute subqueries against the dense index. However, instead of using Reciprocal Rank Fusion (RRF) to merge the retrieved document lists, it employs the cross-encoder reranker on the union of the candidate documents to produce the final, more accurately ranked result set.

4.6 ReAct

ReAct (Reason and Act) [55] is a baseline that performs reasoning over retrieval. It operates on an iterative "Thought, Action, Observation" cycle guided by an LLM. At each step, the model first generates a "Thought" to reason about the problem, then executes an "Action"—either a `search` action against the dense index with a self-generated query, or a `finish` action to output its final answer. The result of the action, the "Observation," is then fed back into

[2]https://github.com/castorini/pyserini
[3]https://huggingface.co/sentence-transformers
[4]https://github.com/facebookresearch/faiss
[5]https://github.com/michaelfeil/infinity
[6]https://huggingface.co/mixedbread-ai/mxbai-rerank-large-v1
[7]https://ollama.com/
[8]https://github.com/vllm-project/vllm

the model's context to inform its next thought. This loop allows the model to progressively gather and synthesize information until it determines it can conclude with a final answer. Thus, both the number of retrieval steps and the specific queries for the dense index at each step are dynamically defined by the LLM.

For the implementation, we use Guidance[9], a framework that enables constrained decoding over large language models. The Guidance framework is ideal for this ReAct implementation due to two key advantages. First, it enforces strict structural control by compelling the model to follow the "Thought-Action-Observation" template and select only from a list of valid actions, which prevents generation errors and simplifies parsing. Second, its stateful control manages the entire interaction as a single, coherent program. Because the input-output history is stored in this state, the model only needs to compute attention over newly added information—such as an observation—instead of re-processing the entire context at each turn, accelerating the iterative process. The prompt consists of clear instructions and a one-shot example, and since we use Guidance, we can guarantee that the outputs conform to the expected format.

4.7 Semantic Chunking

Since we also deal with text documents that can become large, a common approach is to chunk the document into smaller, more manageable pieces. Naive chunking methods, such as fixed-size or overlapping splits, often break sentences mid-thought or separate semantically related content, which can degrade the quality of embeddings and subsequent retrieval performance. To address this, semantic chunking aims to divide documents into coherent, self-contained segments. This is typically achieved by using an embedding model to identify natural semantic boundaries within the text, ensuring that the resulting chunks are contextually meaningful.

For our implementation, we use `chonky`[10], a library that performs semantic chunking using a specially fine-tuned transformer model built on top of `answerdotai/ModernBERT-large`[11]. This framework has been shown to outperform other semantic chunking approaches.

4.8 Evaluation Framework

An important aspect of our evaluation framework is the distinction between a *source document* and a *retrieved object*. A source document refers to the original, complete entity, such as a large PDF file or a full database table. In contrast, a retrieved object is the unit of retrieval that has been indexed, such as a semantic chunk from a document or a single row from a table. This distinction is important because a single query may require multiple retrieved objects, potentially from the same source document, to be fully answered. Our evaluation must account for this one-to-many relationship.

To standardize and automate our evaluation, we have implemented a comprehensive `EvaluationMetrics` class. This class is designed to calculate and visualize standard retrieval metrics across a set of user-defined top-k values, denoted as `@n`. It computes Precision, Recall, and F1-score. Additionally, we incorporate **Perfect Recall (PR)**, a stricter metric that measures the percentage of queries for which *all* ground truth objects were successfully retrieved within the top `n` results. This is important for complex queries where partial evidence is insufficient and we need the full context. The evaluation correctly handles duplicates by operating on the set of unique ground truth and predicted object IDs. The class also calculates and reports the percentage of instances where the number of unique retrieved objects is less than `n`, meaning that not enough objects were retrieved. The primary method, `calculate_metrics`, processes lists of ground truth and predicted IDs to generate a pandas DataFrame summarizing the performance at each specified `n`.

It is important to note a practical implication of this source-object distinction on our evaluation methodology. Specifically, when measuring performance at the source level for a given cutoff '@k', retrieving exactly 'k' objects is often insufficient, as multiple retrieved objects may belong to the same source document. Consequently, after deduplicating the retrieved objects to obtain a list of unique sources, the resulting list may contain fewer than 'k' entries, precluding a fair evaluation at that cutoff. To mitigate this, our evaluation scripts adopt a strategy of over-retrieval. In all our experiments, we multiply the target cutoff 'k' by a constant factor to retrieve a larger initial set of objects, increasing the likelihood that, after source-level deduplication, we have a sufficient number of unique sources to conduct a meaningful evaluation at the desired '@k'.

5 ARM Retriever Implementation

In this section, we detail the implementation of our primary system, the Alignment-Oriented Retrieval Method (ARM)[7]. We describe the retrieval pipeline step-by-step, detailing the algorithms and technologies used. The first step involves constructing both a sparse and a dense index using the implementations described in the previous section. For the serialization of tables, we adopt a row-grained approach, where each row becomes a distinct object in our corpus; consequently, a single table may be represented by multiple objects.

5.1 Information Alignment: Keyword Generation and Rewriting

The first stage of the ARM pipeline involves generating relevant keywords from a given query using beam search. We implement this using vLLM, which natively supports beam search. The number of beams and the maximum number of keywords to retain are configurable parameters of the retriever. If more keywords are generated than the specified maximum, a voting mechanism based on occurrence frequency is used to select the most common ones.

The next step is to rewrite these keywords using constrained beam decoding to align them with the indexed corpus. For this to happen, we first generate all possible n-grams (1-3 tokens) from our corpus. We then adapt the method introduced by RecLM-cgen[38], which constrains the output of an LLM to map to actual item IDs from a recommendation domain. In our case, the place item IDs is taken by n-grams from the corpus, we are recommending n-grams. The RecLM-cgen method is designed for generative recommendation, ensuring the model's output for a query strictly adheres to an entry within a predefined corpus. This is achieved by first

[9] https://github.com/guidance-ai/guidance
[10] https://github.com/mirth/chonky
[11] https://huggingface.co/answerdotai/ModernBERT-large

constructing a prefix tree (trie) from all valid text entries. During inference, the LLM's generation is constrained by masking its vocabulary at each step, allowing only tokens that form a valid path within the trie. This token-by-token validation guarantees the final generated sequence is a verbatim match to an entry in the corpus. Our implementation supports beam search, allowing the number of keyword rewritings to be a configurable parameter.

5.2 Structure Alignment: MIP-based Object Selection

After generating and aligning the keywords, we perform retrieval using the pre-built indexes. We also include the initial user query in the retrieval process, as our experiments showed this improves performance. While the original paper mentions using BM25, our implementation defaults to using the dense Faiss-GPU index, which proved to be both faster and more accurate for our datasets. The implementation allows switching between sparse and dense retrieval via a configuration flag.

The core of the structure alignment component is a Mixed-Integer Programming (MIP) solver, as detailed in the source paper. We have implemented this using the PuLP[12] library. The objective function maximizes a weighted sum of object-query relevance (R_i) and pairwise object compatibility (C_{ij}). The compatibility scores are calculated based on object type: table-table compatibility uses column similarity (header semantics and Jaccard similarity of values), while table-passage and passage-passage compatibility use a mix of cosine similarity and token overlap. The weights for semantic and exact-match similarity are configurable.

To ensure computational efficiency, our `MIPSolver` employs several optimizations. It first aggregates all unique textual components from candidate objects and queries the pre-computed Faiss index. We found that approximately 40% of the required embeddings could be retrieved directly from the index via exact text match. The remaining components are then encoded in batches using a secondary cache. This batching and caching strategy significantly reduces redundant calls to the embedding model. Our solver is also designed with a fallback mechanism, prioritizing the high-performance commercial Gurobi solver[13] and switching to the open-source CBC solver[14] if Gurobi is unavailable.

5.3 Candidate Expansion

The expansion method proposed in the original paper, which involves a pairwise compatibility calculation across all objects in the corpus, proved to be computationally infeasible due to its $O(N^2)$ complexity. We therefore implemented a simpler, more efficient strategy. Our approach uses the initially retrieved objects as new queries against the existing indexes to expand the candidate set. The number of expansion steps and the quantity of objects to consider at each step are configurable parameters.

[12]https://github.com/coin-or/pulp

[13]https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/

[14]https://github.com/coin-or/Cbc

5.4 Alternative Alignment: Reranker-based Selection

Given the strong performance of the dense-index-with-reranker baseline, we designed the ARM implementation to be modular. With a single argument change during initialization, the MIP solver can be replaced with the cross-encoder reranker for the structure alignment stage, which is typically faster and turns out more effective.

5.5 Self-Verification and Final Output Generation

Once the final set of objects is selected (either by the MIP solver or the reranker), they are prepared for the final LLM, which is served by vLLM. To prevent hallucinations that can arise from prompting with raw document content, we generate `pseudo_ids` (e.g., `table_x`, `sentence_y`) for each object. Objects originating from the same source document (e.g., multiple rows from the same table or sentences from the same passage) are grouped under a common pseudo-id. This strategy proved highly effective, dropping the generation of out-of-context IDs to zero in our tests and removing the need for constrained decoding at this final stage. A beam search is then performed over the vocabulary of pseudo-ids to generate the final selection and map them to the actual object IDs. The objects from all beams are gathered, deduplicated, and returned as the final retrieval result.

6 Experiments

In this section, we describe the datasets used for benchmarking the different retrieval systems, present the empirical results, and conclude with a series of ablation studies designed to identify and justify the optimal configuration for the ARM retriever.

6.1 Datasets

The following datasets were used to evaluate our retrieval systems, chosen to cover a range of tasks including multi-table retrieval, joint table-and-text retrieval, and text-only retrieval.

- **TableBench [53]:** A benchmark for Table Question Answering (TableQA) using complex tables from industrial and financial scenarios. It is designed to test multi-hop reasoning, including fact-checking, numerical analysis, and data forecasting. The retrieval task is exclusively focused on **multi-table retrieval**, where the system must identify and retrieve a set of joinable tables to answer complex, insight-driven questions.
- **Spider [58]:** A large, cross-domain benchmark for text-to-SQL parsing over 200 relational databases. Questions frequently require complex SQL constructs like JOINs and nested subqueries, making the task inherently multi-hop. The retrieval challenge is **multi-table retrieval**, requiring a system to select the correct set of tables from a database schema based on a natural language question.
- **TabFact [12]:** A large-scale benchmark for verifying textual statements against Wikipedia tables. The task is to classify a statement as ENTAILED or REFUTED, often requiring reasoning across multiple rows. This benchmark

evaluates a system's ability to perform accurate **single-table retrieval** by finding the one correct table that can verify or refute a given statement.

- **OTT-QA [11]:** An open-domain question answering dataset that requires reasoning over both Wikipedia tables and unstructured text passages. Questions are designed to necessitate linking information between these heterogeneous sources, thus evaluating a system's capability for multi-hop **joint table and text retrieval** by finding the relevant table(s) and text passage(s) needed for synthesis.
- **MultiHop-RAG [48]:** A benchmark built from English news articles to specifically evaluate multi-hop Retrieval-Augmented Generation (RAG) systems. Questions are designed to require evidence synthesis from multiple documents, making the task an evaluation of **multi-document text retrieval**.
- **FeTaQA [40]:** A dataset for free-form, generative TableQA, where the goal is to produce long, explanatory answers by integrating multiple facts from a single Wikipedia table. This tests a system's performance on **single-table retrieval**, where it must find the table containing all the necessary information for a coherent, long-form answer.
- **FEVEROUS [2]:** A fact verification benchmark requiring evidence from a mix of Wikipedia sentences and tables. A single claim may need a combination of evidence from both types, thereby testing multi-hop **joint table and text retrieval**.
- **MT-RAIG [43]:** A large-scale benchmark for generating deep, analytical insights by integrating information from multiple distinct tables. Every query is designed to be answered by reasoning over a set of tables, making this an inherent test of **multi-table retrieval**.
- **BIRD [36]:** A large-scale text-to-SQL benchmark featuring massive, noisy databases from real-world domains. It tests a model's ability to handle dirty data and complex values when generating SQL, focusing the retrieval task on **multi-table retrieval** from large, noisy databases.

Data Preprocessing and Serialization. After gathering the datasets, we converted all of them into a unified format through a standardized serialization process. This process distinguishes between tabular and textual data sources. For all datasets containing tables, we adopted a row-grained serialization strategy where each individual row from every table was treated as a distinct object. For datasets composed of unstructured text, we employed the semantic chunking methodology described in the previous section, with each resulting chunk considered a single object. To preserve context, if an object's source had an associated title, this title was appended to the serialized content, separated by a special `[SEP]` token. The serialization format for a table row specifically followed the structure: `table_name [SEP] [H] col1_name : val1 [H] col2_name : val2 [H] ...`. The final count of objects generated for each dataset, along with the average word count per object, is presented in Table 1.

Benchmarks. Each dataset is accompanied by its own benchmark for evaluation. However, given the number of datasets and the computational time required by agentic retrievers, evaluating on the full set of queries for each benchmark was impractical. To create a representative yet manageable evaluation set, we subsampled the queries from each benchmark. To ensure diversity and avoid the biases of naive random sampling, we employed K-means clustering on the TF-IDF representations of the queries. We then selected one random query from each of the resulting clusters. For our experiments, we set the number of clusters, k, to 500, yielding a final benchmark of 500 queries per dataset. The average number of source documents required to answer a query for each of these subsampled benchmarks is detailed in Table 2.

Benchmark	# Objects	Avg. Words per Object
BIRD	3,932,735	61.54
FeTaQA	28,358	32.96
FEVEROUS	391,605	28.92
MT-RAIG BENCH	206,138	38.18
MultiHop-RAG	14,895	85.06
OTT-QA	9,782	33.34
Spider	245,574	12.57
TabFact	22,768	33.80
TableBench	9,663	34.97

Table 1: Dataset statistics

Benchmark	Avg. Sources per Query
BIRD	1.91
FeTaQA	1.00
FEVEROUS	1.96
MT-RAIG BENCH	3.10
MultiHop-RAG	2.64
OTT-QA	1.00
Spider	1.61
TabFact	1.00
TableBench	1.00

Table 2: Average Number of Source Documents Required per Query in Subsampled Benchmarks.

6.2 Experimental Setup

Evaluation Metrics. To evaluate the performance of our retrieval systems, we use a set of metrics calculated at retrieval cutoffs of $k \in \{1, 3, 5\}$.

- **Precision@k**: Measures the proportion of retrieved objects that are relevant within the top-k results. It answers the question: "Of the k items the system returned, how many were correct?"
- **Recall@k**: Measures the proportion of all relevant ground truth objects that are successfully found within the top-k results. This metric answers: "Of all the items that should have been returned, how many did the system find?"
- **F1-Score@k**: The harmonic mean of Precision and Recall, providing a single score that balances the trade-off between the two. It is useful for a consolidated performance view, especially when the number of relevant items varies.

- **Perfect Recall@k (PR@k)**: A stricter version of recall that measures the percentage of queries for which *all* required ground truth objects were retrieved within the top-k. This is important for evaluating performance on multi-hop queries where partial evidence is insufficient to form a correct answer.
- **Queries Per Second (QPS)**: Measures the throughput of the retrieval system by dividing the total number of queries processed by the total time taken. This metric is used to evaluate the inference speed and efficiency of each approach.

Hardware and Software Environment. All experiments were conducted on Ubuntu 22.04.3 LTS with:

- **CPU**: Intel Xeon Gold 5318Y, 2 sockets × 24 cores/socket, hyperthreading enabled (96 logical cores), base 2.10 GHz.
- **Memory**: 377 GiB RAM.
- **GPUs**: 2× NVIDIA RTX A6000 (46 GB VRAM each), CUDA 12.2.

6.3 Results

Table Serialization. In this first experiment, we evaluate the impact of different table serialization strategies on retrieval performance using the **FEVEROUS** dataset. We compare three granularities: serializing the entire table as a single document, each row as an individual object, and each distinct cell as an object. To isolate the effect of the representation itself, we limit our evaluation to three core retrieval architectures: a sparse BM25 index, a dense embedding-based index, and a dense index augmented with a cross-encoder reranker. For all dense retrieval methods, embeddings were generated using the `WhereIsAI/UAE-Large-V1`[15] model.

The results, summarized in Table 3, indicate that row-level serialization provides the most effective representation, particularly for the BM25 and base dense index methods, where it yields the highest recall and perfect recall. This suggests that representing each row as a self-contained unit strikes a good balance between providing sufficient context and avoiding the noise of a full-table serialization. While the sparse BM25 method remains highly competitive, often outperforming the base dense index, the addition of a reranker consistently boosts performance across all serialization strategies, especially for F1-Score and Perfect Recall at k=5. Cell-level serialization performs on-par with row representation and sometimes outperforms it by a bit, however given the extra cost of computing the embedding for each distinct value compared to the row encoding, we will keep the row representation.

Embedding Models. To ensure our experiments use a top-performant embedding model, we conducted a comprehensive evaluation to select an optimal off-the-shelf embedding model. We began by surveying top-performing open-source models (under 2 billion parameters) from the Massive Text Embedding Benchmark (MTEB) leaderboard[16], a standard framework for evaluating the quality of text embeddings across diverse tasks. However, since the MTEB primarily focuses on unstructured text retrieval, and our task involves the unique structure of serialized tables, direct evaluation to our datasets was necessary to determine which model best handles this specific data format. We selected 12 prominent models, computed embeddings for all our datasets, indexed them using Faiss, and performed retrieval, reporting Recall@5 as the key performance indicator.

The results of this evaluation are presented in Table 4. Several key insights emerge from this analysis. Firstly, no single model dominates across all datasets, indicating that performance is highly dependent on the specific data characteristics of each benchmark. While `inf-retriever-v1-1.5b` achieves the highest average recall, the 'older' `bge-m3` is a very close second and demonstrates remarkable consistency. Secondly, a clear trade-off exists between retrieval performance and inference speed (QPS). The fastest models, such as `snowflake-arctic-embed-s`, exhibit a noticeable drop in recall compared to the top performers. Lastly, the strong performance of multilingual models like `bge-m3` suggests that their training on diverse linguistic and structural data may provide an advantage in interpreting the structured syntax of our serialized table rows.

Serialization	Method	@k	P	R	F	PR
Table	BM25	1	0.598	0.334	0.403	0.161
		3	0.408	0.633	0.466	0.463
		5	0.288	0.730	0.391	0.593
	Dense	1	0.542	0.296	0.360	0.135
		3	0.393	0.619	0.451	0.443
		5	0.285	0.733	0.388	0.582
	D+R	1	0.578	0.312	0.381	0.139
		3	0.448	0.702	0.513	0.540
		5	0.319	0.816	0.433	0.699
Row	BM25	1	0.616	0.351	0.420	0.178
		3	0.429	0.678	0.493	0.510
		5	0.304	0.779	0.413	0.648
	Dense	1	0.612	0.353	0.421	0.183
		3	0.420	0.675	0.486	0.501
		5	0.301	0.784	0.412	0.641
	D+R	1	0.563	0.297	0.366	0.125
		3	0.440	0.684	0.503	0.520
		5	0.316	0.805	0.429	0.687
Cell	BM25	1	0.597	0.335	0.403	0.165
		3	0.426	0.666	0.488	0.497
		5	0.311	0.770	0.418	0.636
	Dense	1	0.608	0.353	0.419	0.186
		3	0.428	0.666	0.489	0.489
		5	0.326	0.769	0.430	0.622
	D+R	1	0.589	0.324	0.393	0.151
		3	0.451	0.696	0.514	0.531
		5	0.336	0.810	0.447	0.690

Table 3: Retrieval performance across different table serialization strategies. We evaluate using a sparse (BM25) and two dense (Dense Index, Dense + Reranker) retrieval methods.

[15]https://huggingface.co/WhereIsAI/UAE-Large-V1
[16]https://huggingface.co/spaces/mteb/leaderboard

Model	BIRD	FETAQA	FEVEROUS	MT RAIG	Multi-Hop RAG	OTTQA	SPIDER	TabFact	Table-Bench	Avg.	QPS
inf-retriever-v1-1.5b	**0.8241**	**0.9630**	0.8001	<u>0.9030</u>	**0.7001**	0.8950	0.6437	0.6750	**0.5203**	**0.7694**	60.19
bge-m3	0.7731	**0.9630**	**0.8235**	**0.9101**	<u>0.6997</u>	0.8600	0.6293	**0.6860**	**0.5203**	<u>0.7628</u>	66.93
UAE-Large-V1	<u>0.7822</u>	0.9590	<u>0.8123</u>	0.8975	0.6988	**0.9260**	0.6023	0.6730	<u>0.5023</u>	0.7615	65.16
gte-modernbert-base	0.7802	0.9510	0.7590	0.8852	0.6867	0.8700	**0.6508**	0.6620	0.4932	0.7487	61.86
multilingual-e5-large-instruct	0.7732	0.9600	0.7904	0.8893	0.6469	<u>0.9030</u>	0.6188	0.6680	0.4740	0.7471	63.50
gte-multilingual-base	0.7482	<u>0.9610</u>	0.7321	0.8873	0.6753	0.8720	<u>0.6467</u>	0.6780	0.4989	0.7444	74.63
snowflake-arctic-embed-l-v2.0	0.7476	0.9560	0.7912	0.8866	0.6812	0.8710	0.5766	<u>0.6820</u>	0.4932	0.7428	72.17
nomic-embed-text-v2-moe	0.7174	<u>0.9610</u>	0.7568	0.8684	0.6737	0.8520	0.5859	0.6680	0.4639	0.7275	71.49
jina-embeddings-v3	0.7358	0.9520	0.7512	0.8285	0.6713	0.8430	0.6003	0.6120	0.4436	0.7153	75.54
modernbert-embed-base	0.7031	0.9480	0.7296	0.7741	0.6580	0.7960	0.6278	0.6530	0.4729	0.7069	<u>75.89</u>
snowflake-arctic-embed-s	0.6299	0.8950	0.7836	0.8213	0.5736	0.7370	0.5162	0.5830	0.4345	0.6638	**79.05**
Qwen3-Embedding-0.6B	0.7811	0.9600	0.7828	0.8407	0.6838	0.8630	0.4837	0.6090	0.4910	0.7216	64.35

Table 4: Performance of various embedding models across datasets. The best-performing model for each dataset is in bold, and the second-best is <u>underlined</u>. Model names are linked to their Hugging Face repositories. We report Recall@5 for each experiment.

Based on these results, we selected **`bge-m3`** as the embedding model for all subsequent experiments. Although `inf-retriever` shows a slightly higher average recall, we chose `bge-m3`, since it's roughly 3x times smaller than the `inf-retriever` and performs on par with it.

***Main Results*.** In this section, we present the main evaluation results for all retrieval methods across the nine benchmarks. For the agentic and alignment-oriented retrievers, namely ReAct and ARM, we employed the `Qwen/Qwen2.5-72B-Instruct-AWQ`[17] model as the core Large Language Model. The final hyperparameter configuration for the ARM retriever is detailed in Table 5.

Parameter	Value
`keyword_extraction_beams`	5
`generate_n_grams`	False
`keyword_rewriting_beams`	0
`alignment_retrieval_k`	5
`use_reranker_instead_of_mip`	True
`expansion_steps`	1
`expansion_k_compatible`	5
`compatibility_semantic_weight`	0.99
`compatibility_exact_weight`	0.01
`dense_instead_of_sparse`	True
`final_llm_selection_beams`	5

Table 5: Hyperparameter Configuration for the ARM Retriever.

This configuration was established after careful experimentation with the primary goal of optimizing the trade-off between retrieval accuracy and inference speed. We chose to use the cross-encoder reranker instead of the MIP solver for structure alignment, as it consistently delivered superior empirical performance and faster execution. Keyword rewriting was disabled, as the high-quality keywords generated by the 72B parameter LLM made this step redundant. In fact, in most cases, the keywords were exact matches with the n-grams existing in the corpus, combined with the really good performance of the embedding model, rewriting the keywords was not necessary. We should note, though, that in real-world scenario, not in benchmarks, where the user should express themselves more freely, without knowledge of the underlying data, this step would potentially be more fruitful. Similarly, a single candidate expansion step was found to be sufficient for augmenting the initial retrieval set without introducing excessive latency. The configuration prioritizes semantic similarity (0.99 weight) over exact-match (0.01 weight) when calculating compatibility, leveraging the strength of the dense embeddings, which turned out to be superior compared to the Jaccard similarity. Finally, the internal retrieval used the dense index instead of the sparse index proposed initially by the paper. The justification for these choices is explored in greater detail in the subsequent ablation studies.

The experimental results, detailed in Table 6, reveal several performance trends across the baseline retrieval systems. First, dense retrieval (DR) generally surpasses the sparse BM25 baseline in recall, indicating that semantic matching captures relevant information missed by lexical overlap. Second, the addition of a cross-encoder reranker (DRR) further increases performance over the base dense retriever on most datasets by refining the initial candidate set. An exception is the BIRD dataset, where the base dense retriever shows higher recall. Third, query decomposition alone (DR-D) does not uniformly improve results and, in some cases, degrades performance compared to standard dense retrieval, suggesting that the fusion of sub-query results can introduce noise or that queries lose important holistic context when split. Finally, the combination of query decomposition and reranking (DRR-D) frequently yields the highest performance, especially on multi-hop benchmarks such as BIRD, MT-RAIG, and MultiHop-RAG.

To make a more direct comparison and simplify the interpretation of our results, we present a consolidated view in Table 7. This table reports the performance metrics for all baseline systems exclusively at a retrieval cutoff of k=5. We selected this value because it most closely aligns with the average number of objects retrieved by the reasoning-based retrievers, enabling a fairer comparison. It is important to note that in some cases, recall may increase between methods while precision decreases, even at a fixed k=5. This occurs when a method retrieves fewer than five unique objects, a scenario that can arise due to our row-grained serialization, where multiple predictions might map to the same source document. In any case,

[17] https://huggingface.co/Qwen/Qwen2.5-72B-Instruct-AWQ

	BM25				Dense retrieval (DR)				DR + reranker (DRR)				DR-D				DRR-D			
	P	R	F1	PR	P	R	F1	PR	P	R	F1	PR	P	R	F1	PR	P	R	F1	PR
BIRD																				
@1	0.750	0.417	0.521	0.120	0.820	0.488	0.593	0.190	0.750	0.413	0.520	0.110	0.820	0.472	0.581	0.160	0.820	0.462	0.575	0.140
@3	0.523	0.682	0.556	0.450	0.503	0.825	0.609	0.650	0.463	0.768	0.563	0.570	0.480	0.785	0.580	0.590	0.497	0.817	0.602	0.630
@5	0.435	0.733	0.492	0.540	0.328	0.884	0.469	0.770	0.326	0.879	0.466	0.750	0.324	0.877	0.464	0.760	0.342	0.917	0.488	0.820
FeTaQA																				
@1	0.590	0.590	0.590	0.590	0.800	0.800	0.800	0.800	0.840	0.840	0.840	0.840	0.620	0.620	0.620	0.620	0.830	0.830	0.830	0.830
@3	0.237	0.700	0.353	0.700	0.332	0.900	0.477	0.900	0.330	0.900	0.475	0.900	0.323	0.880	0.465	0.880	0.328	0.890	0.472	0.890
@5	0.165	0.770	0.270	0.770	0.234	0.910	0.356	0.910	0.229	0.900	0.349	0.900	0.231	0.910	0.353	0.910	0.232	0.900	0.353	0.900
FEVEROUS																				
@1	0.580	0.312	0.386	0.130	0.640	0.359	0.438	0.160	0.580	0.309	0.384	0.120	0.520	0.283	0.351	0.110	0.560	0.299	0.371	0.120
@3	0.417	0.662	0.485	0.490	0.443	0.680	0.510	0.500	0.473	0.740	0.549	0.590	0.400	0.651	0.469	0.450	0.460	0.713	0.531	0.560
@5	0.298	0.775	0.411	0.640	0.320	0.815	0.439	0.690	0.334	0.866	0.460	0.760	0.294	0.767	0.406	0.620	0.328	0.849	0.451	0.750
MT-RAIG BENCH																				
@1	0.540	0.196	0.283	0.000	0.870	0.308	0.448	0.000	0.890	0.310	0.452	0.000	0.840	0.296	0.430	0.000	0.900	0.315	0.458	0.000
@3	0.500	0.510	0.495	0.300	0.693	0.703	0.683	0.380	0.777	0.767	0.754	0.460	0.700	0.712	0.690	0.400	0.780	0.774	0.759	0.460
@5	0.411	0.622	0.482	0.460	0.556	0.851	0.655	0.680	0.632	0.865	0.701	0.670	0.539	0.833	0.638	0.630	0.640	0.882	0.714	0.690
MultiHop-RAG																				
@1	0.710	0.300	0.417	0.000	0.680	0.295	0.407	0.000	0.800	0.337	0.469	0.000	0.540	0.231	0.320	0.000	0.800	0.340	0.472	0.000
@3	0.503	0.607	0.540	0.300	0.473	0.589	0.517	0.310	0.587	0.708	0.630	0.400	0.440	0.540	0.477	0.250	0.583	0.711	0.630	0.430
@5	0.406	0.749	0.514	0.490	0.377	0.696	0.480	0.470	0.451	0.808	0.563	0.570	0.364	0.697	0.469	0.460	0.445	0.810	0.562	0.560
OTT-QA																				
@1	0.450	0.450	0.450	0.450	0.580	0.580	0.580	0.580	0.700	0.700	0.700	0.700	0.430	0.430	0.430	0.430	0.720	0.720	0.720	0.720
@3	0.202	0.590	0.300	0.590	0.247	0.740	0.370	0.740	0.300	0.890	0.448	0.890	0.233	0.700	0.350	0.700	0.292	0.860	0.435	0.860
@5	0.147	0.650	0.237	0.650	0.179	0.840	0.294	0.840	0.198	0.920	0.324	0.920	0.168	0.800	0.277	0.800	0.205	0.930	0.333	0.930
Spider																				
@1	0.390	0.295	0.323	0.220	0.510	0.371	0.412	0.260	0.560	0.401	0.449	0.270	0.440	0.323	0.357	0.230	0.570	0.406	0.456	0.270
@3	0.228	0.458	0.294	0.390	0.300	0.593	0.382	0.520	0.310	0.607	0.394	0.530	0.293	0.584	0.374	0.490	0.310	0.602	0.393	0.520
@5	0.174	0.548	0.257	0.470	0.196	0.641	0.289	0.570	0.204	0.656	0.300	0.600	0.198	0.646	0.292	0.580	0.204	0.656	0.300	0.600
TabFact																				
@1	0.240	0.240	0.240	0.240	0.290	0.290	0.290	0.290	0.420	0.420	0.420	0.420	0.220	0.220	0.220	0.220	0.370	0.370	0.370	0.370
@3	0.125	0.370	0.187	0.370	0.132	0.390	0.197	0.390	0.178	0.530	0.267	0.530	0.115	0.340	0.172	0.340	0.175	0.520	0.262	0.520
@5	0.104	0.480	0.170	0.480	0.092	0.440	0.151	0.440	0.123	0.580	0.202	0.580	0.082	0.390	0.134	0.390	0.108	0.520	0.178	0.520
TableBench																				
@1	0.220	0.220	0.220	0.220	0.220	0.220	0.220	0.220	0.270	0.270	0.270	0.270	0.160	0.160	0.160	0.160	0.290	0.290	0.290	0.290
@3	0.115	0.320	0.167	0.320	0.128	0.360	0.187	0.360	0.147	0.410	0.213	0.410	0.110	0.330	0.165	0.330	0.158	0.440	0.230	0.440
@5	0.095	0.350	0.144	0.350	0.102	0.410	0.158	0.410	0.117	0.450	0.180	0.450	0.087	0.390	0.142	0.390	0.117	0.460	0.180	0.460

Table 6: Retrieval performance of baselines. PR refers to Perfect Recall. X-D refers to method X with query decomposition.

our primary comparison will focus on the interplay between precision, recall, and QPS, for which this observation does not affect the analysis.

The data in Table 7 shows distinct performance characteristics for each retrieval method across the various benchmarks. On multi-table datasets such as **BIRD** and **Spider**, the ARM retriever acihieved the highest values for precision, recall, and F1-score. For instance, on BIRD, ARM's F1-score of 0.79 is substantially higher than the 0.49 recorded by the best-performing baseline, DRR-D. The ReAct retriever's performance on these datasets was lower than ARM and, in some cases, lower than the non-agentic baselines.

For benchmarks requiring joint text and table retrieval, such as **OTT-QA** and **FEVEROUS**, the results are more varied. On OTT-QA, ARM achieved the highest F1-score (0.50), while its perfect recall (0.93) was tied with the DRR-D baseline. On FEVEROUS, the DRR baseline recorded the highest F1-score (0.46), while ARM achieved the highest recall (0.90) and perfect recall (0.85). The ReAct retriever recorded the highest precision on this dataset (0.36).

On single-table QA datasets, ARM again achieved the highest performance. For **FeTaQA**, ARM's F1-score (0.76) and perfect recall (0.91) exceeded the other methods. Similarly, on **TabFact**, ARM achieved the highest F1-score (0.21) and perfect recall (0.65).

Regarding efficiency, a clear trade-off is evident. The BM25 retriever consistently demonstrates the highest throughput, with a QPS of 1373 on Spider. The dense retrieval methods (DR, DRR, DR-D, DRR-D) are significantly slower but still orders of magnitude faster than the reasoning-based retrievers. Both ReAct and ARM have a QPS of less than 1.0 across all datasets, with ARM generally being faster than ReAct.

When averaged across all nine datasets, as shown in Table 8, the performance trends become even more distinct. The ARM retriever achieves the highest average scores across all four retrieval quality metrics: Precision (0.43), Recall (0.81), F1-Score (0.51), and Perfect Recall (0.77). The ReAct retriever records the second-highest average Precision and F1-score, while the DRR and DRR-D baselines got the second-highest average Recall and Perfect Recall. This aggregation underscores a clear trade-off between retrieval quality and speed. The methods that do not involve LLM-based reasoning (BM25, DR, DRR, DR-D, DRR-D) operate at a much higher QPS, with BM25 being the fastest. In contrast, the reasoning-based retrievers

(ReAct and ARM) demonstrate higher retrieval effectiveness, particularly in precision and F1-score, at the cost of significantly lower throughput.

6.4 Ablation Studies

In this section, we conduct a series of ablation studies to analyze the performance of our system and justify the final hyperparameter configuration of the ARM retriever.

Our initial experiments on text-to-SQL datasets like BIRD revealed a significant challenge with row-level serialization. Even when retrieving up to 2,048 objects—the maximum supported by Faiss—we observed that for over 80% of queries, the system failed to retrieve the complete set of required tables. The retrieved objects were predominantly rows from a small number of tables, indicating a strong relevance signal between query terms and the table or column names, which caused the retriever to repeatedly return objects from the same source. This observation suggested that for text-to-SQL tasks, which deal with large tables, a metadata-only representation might be more effective.

To test this hypothesis, we conducted a targeted ablation study comparing two retrieval strategies on the Spider dataset: one using only table metadata (table name and column names) as the retrieval object, and the other using the full row-level serialization. The results are presented in Table 9.

Row-level vs. Metadata-only Retrieval for Text-to-SQL. The results confirm that metadata-only serialization is a more effective strategy for text-to-SQL tasks. While ARM performs comparably with both serialization methods, the baseline retrievers show a marked decrease in performance with row-level serialization, particularly in recall and perfect recall. This is because metadata-only serialization forces the retriever to evaluate at the table level, preventing it from getting stuck on multiple rows from a single, highly relevant table. In addition to the performance benefits, this approach is more efficient in both search space and storage. Indexing only table metadata drastically reduces the number of objects in the corpus, leading to smaller index sizes and faster retrieval times. Based on these findings, all subsequent experiments on text-to-SQL datasets (BIRD and Spider) utilize the metadata-only serialization.

Keyword Rewriting. This study evaluates the utility of the keyword rewriting step in the information alignment phase, where an LLM rewrites query keywords to better match the corpus. To isolate its effect, we measured the set of candidate objects passed to the MIP solver, both with and without rewriting enabled, using a sample of 10 queries from each dataset. The results showed that the set of input objects for the MIP solver was identical in both configurations. However, enabling the rewriter introduced a significant latency overhead of approximately 10 seconds per query. This suggests that either the keywords generated by the base LLM already exist verbatim in the corpus, or that the dense index used for internal retrieval is sufficiently robust to handle semantic variations without needing exact lexical matches. Given the considerable cost for no performance gain, we disabled keyword rewriting in our final configuration, though further study on its utility in different datasets should be performed.

Dataset	Method	P	R	F1	PR	QPS
	BM25	0.44	0.73	0.49	0.54	**1373.00**
	DR	0.33	0.88	0.47	0.77	64.00
	DRR	0.33	0.88	0.47	0.75	2.00
BIRD	DR-D	0.32	0.88	0.46	0.76	7.00
	DRR-D	0.34	0.92	0.49	0.82	1.26
	ReAct	0.32	0.85	0.45	0.73	0.01
	ARM	**0.73**	**0.96**	**0.79**	**0.89**	0.10
	BM25	0.17	0.77	0.27	0.77	**132.00**
	DR	0.23	0.91	0.36	**0.91**	49.00
	DRR	0.23	0.90	0.35	0.90	1.00
FeTaQA	DR-D	0.23	0.91	0.35	**0.91**	2.00
	DRR-D	0.23	0.90	0.35	0.90	0.50
	ReAct	0.62	0.90	0.68	0.90	0.01
	ARM	**0.73**	**0.92**	**0.76**	**0.91**	0.05
	BM25	0.30	0.78	0.41	0.64	**84.00**
	DR	0.32	0.82	0.44	0.69	7.00
	DRR	0.33	0.87	**0.46**	0.76	1.00
FEVEROUS	DR-D	0.29	0.77	0.41	0.62	3.00
	DRR-D	0.33	0.85	0.45	0.75	0.50
	ReAct	**0.36**	0.76	0.45	0.57	0.01
	ARM	0.27	**0.90**	0.38	**0.85**	0.05
	BM25	0.41	0.62	0.48	0.46	**137.00**
	DR	0.56	0.85	0.66	0.68	14.00
	DRR	0.63	0.87	0.70	0.67	1.00
MT-RAIG	DR-D	0.54	0.83	0.64	0.63	1.00
	DRR-D	0.64	0.88	0.71	0.69	0.50
	ReAct	**0.71**	0.87	**0.76**	0.75	-
	ARM	0.64	**0.93**	0.72	**0.83**	0.05
	BM25	0.41	0.75	0.51	0.49	**213.00**
	DR	0.38	0.70	0.48	0.47	18.00
	DRR	0.45	0.81	0.56	0.57	1.00
MultiHop	DR-D	0.36	0.70	0.47	0.46	1.00
	DRR-D	0.45	0.81	0.56	0.56	0.50
	ReAct	0.49	0.78	0.56	0.55	0.01
	ARM	**0.52**	**0.90**	**0.61**	**0.75**	0.05
	BM25	0.15	0.65	0.24	0.65	**329.00**
	DR	0.18	0.84	0.29	0.84	56.00
	DRR	0.20	0.92	0.32	0.92	1.00
OTT-QA	DR-D	0.17	0.80	0.28	0.80	2.00
	DRR-D	0.21	**0.93**	0.33	**0.93**	0.50
	ReAct	0.23	0.82	0.32	0.82	0.01
	ARM	**0.41**	0.91	**0.50**	**0.93**	0.10
	BM25	0.17	0.55	0.26	0.47	**1391.00**
	DR	0.20	0.64	0.29	0.57	81.00
	DRR	0.20	0.66	0.30	0.60	4.00
Spider	DR-D	0.20	0.65	0.29	0.58	33.00
	DRR-D	0.20	0.66	0.30	0.60	2.00
	ReAct	0.15	0.63	0.23	0.55	0.01
	ARM	**0.26**	**0.68**	**0.40**	**0.65**	0.10
	BM25	0.10	0.48	0.17	0.48	**303.00**
	DR	0.09	0.44	0.15	0.44	50.00
	DRR	0.12	0.58	0.20	0.58	1.00
TabFact	DR-D	0.08	0.39	0.13	0.39	10.00
	DRR-D	0.11	0.52	0.18	0.52	1.00
	ReAct	0.11	0.45	0.17	0.45	0.01
	ARM	**0.15**	**0.65**	**0.21**	**0.65**	0.10
	BM25	0.10	0.35	0.14	0.35	**392.00**
	DR	0.10	0.41	0.16	0.41	41.00
	DRR	0.12	0.45	0.18	0.45	1.00
TableBench	DR-D	0.09	0.39	0.14	0.39	6.00
	DRR-D	0.12	0.46	0.18	0.46	1.00
	ReAct	**0.19**	0.39	**0.23**	0.39	0.01
	ARM	0.17	**0.47**	0.22	**0.47**	0.10

Table 7: Retrieval performance of baselines at @5 and results for reasoning retrievers. The best value per column for each dataset is in bold, and the second best is underlined. P, R, F1, and PR are precision, recall, F1-score, and passage recall, respectively. QPS is queries per second.

Method	P	R	F1	PR	QPS
BM25	0.25	0.63	0.33	0.54	**483.78**
DR	0.27	0.72	0.37	0.64	42.22
DRR	0.29	0.77	0.39	0.69	1.44
DR-D	0.25	0.70	0.35	0.62	7.22
DRR-D	0.29	0.77	0.39	0.69	0.86
ReAct	0.35	0.72	0.43	0.63	0.01
ARM	**0.43**	**0.81**	**0.51**	**0.77**	0.08

Table 8: Average retrieval performance across all datasets. The best value per column is in bold, and the second best is underlined.

Serialization	Method	P	R	F1	PR
Metadata-only	BM25	0.170	0.550	0.260	0.470
	DR	0.200	0.640	0.290	0.570
	DRR	0.200	0.660	0.300	0.600
	DR-D	0.200	0.650	0.290	0.580
	DRR-D	0.200	0.660	0.300	0.600
	ARM	0.410	0.580	0.450	0.510
Row-level	BM25	0.158	0.342	0.203	0.250
	DR	0.232	0.588	0.315	0.510
	DRR	0.254	0.605	0.333	0.540
	DR-D	0.229	0.608	0.314	0.520
	DRR-D	0.265	0.648	0.353	0.590
	ARM	0.409	0.584	0.451	0.600

Table 9: Comparison of Metadata-only vs. Row-level serialization on the Spider dataset, measured at k=5.

Candidate Expansion. Here, we investigate the impact of expanding the initial candidate set by retrieving compatible neighbors for already-retrieved objects. We tested four configurations: no expansion, a single expansion step retrieving 3 or 5 neighbors, and two expansion steps retrieving 3 neighbors each. The results indicate that expansion provides a performance uplift, with a 1-step, 5-neighbor expansion yielding a 3% improvement in the quality of the candidate set for an overhead of 6 seconds. Increasing the expansion to two steps did not improve performance further but nearly doubled the latency, demonstrating diminishing returns.

Internal Index: Sparse vs. Dense. This experiment compares the use of a sparse (BM25) versus a dense (Faiss) index for the internal retrieval step that gathers candidates for the alignment stage. By measuring the quality of the input set provided to the MIP solver, we found that using the dense index provided a substantial 11.3% performance boost over the sparse index. This improvement came at a negligible cost, adding only 0.1 seconds of latency per query.

Compatibility Weights. This study examines the optimal balance between lexical (Jaccard) and semantic (embedding cosine) similarity when calculating the compatibility score (C_{ij}) within the MIP solver. We tested five weighting schemes for the lexical vs. semantic components, measuring the quality of the final object set produced by the solver. The results show a clear trend: performance increases as more weight is given to semantic similarity. A configuration that almost exclusively relies on semantic similarity (0.01 lexical vs. 0.99 semantic) improved the output quality by 4.4% compared to an equal-weight baseline. This indicates that for determining the structural relationships between objects, understanding the semantic context is significantly more important than simple token overlap.

Alignment Method: MIP Solver vs. Reranker. We compare the effectiveness of the two potential methods for the structure alignment stage: the constraint-based MIP solver and the same cross-encoder used for reranking in dense retrieval. We measured the quality of the final set of objects selected by each method. The experiment revealed that replacing the MIP solver with the cross-encoder reranker improved the final retrieval performance by 2.6%. Furthermore, the reranker was, on average, 1.5 seconds faster per query. This suggests that for our task, the deep, contextualized relevance scoring of the reranker is more effective at identifying the best evidence combination than the explicit, constraint-based optimization of the MIP solver, while also being more computationally efficient.

Candidate Set Size for LLM Self-Verification. This study investigates the optimal number of candidate objects to present to the LLM for the final self-verification step. We tested feeding the top 10, 30, 50, and 100 objects from the alignment stage (MIP solver or reranker) to the LLM and measured the final recall of its selected objects. The results show that performance increases as the candidate set grows from 10 to 50 objects, with a 7% uplift in recall for a set of 50. However, increasing the set further to 100 objects resulted in a slight performance degradation. This suggests that while providing a larger context is beneficial, excessively large inputs may hinder the LLM's ability to focus on the most relevant evidence, a known limitation of models with finite attention spans. Based on this, we selected a candidate set size of 50 as the optimal balance between providing comprehensive context and avoiding attentional overload.

LLM Input Serialization Method. Here, we evaluate different methods for serializing the candidate objects before they are passed to the LLM for self-verification. We compared three approaches: 1) serializing the full object content directly, 2) assigning a unique 'pseudo_id' to each object and prompting the LLM to select from these IDs, and 3) grouping all objects from the same source document (e.g., all rows from one table) under a single 'pseudo_id'. The results show a clear advantage for using identifiers. The baseline of serializing full content resulted in a 13% hallucination rate, where the LLM generated non-existent or malformed content. Switching to object-level 'pseudo_ids' reduced hallucinations to just 1% and increased accuracy by 10%. The best performance was achieved by grouping objects by their source and using source-level 'pseudo_ids', which eliminated hallucinations entirely (0%) and increased accuracy by 12%. Since our evaluation is performed at the source level, this grouping strategy is not only more efficient but also aligns perfectly with the final evaluation metric.

Final Selection Beam Search. This final study assesses the impact of using beam search during the LLM's self-verification step, where it generates the sequence of 'pseudo_ids' for the final answer. We compared greedy decoding (a beam size of 1) with a beam search using 5 beams. The results indicate that using a beam size of 5 improves final recall by 1.5% and decreases precision by 3%,

Component Under Study	Configuration	Performance Change	Latency Overhead (s/query)
Keyword Rewriting	Enabled (Constrained Decoding)	0.0%	~10.0
	Disabled	**Baseline**	**0.0**
Candidate Expansion	No Expansion	Baseline	0.0
	1 Step, 3 Neighbors	+2.0%	~5.0
	1 Step, 5 Neighbors	**+3.0%**	**~6.0**
	2 Steps, 3 Neighbors	+3.0%	~10.0
Internal Index Type	Sparse (BM25)	Baseline	0.0
	Dense (Faiss)	**+11.3%**	**~0.1**
Compatibility Weights (Lexical vs. Semantic)	0.50 vs. 0.50	Baseline	0.0
	0.99 vs. 0.01	-3.0%	0.0
	0.75 vs. 0.25	+0.5%	0.0
	0.25 vs. 0.75	+1.2%	0.0
	0.01 vs. 0.99	**+4.4%**	**0.0**
Alignment Method	MIP Solver	Baseline	~1.5
	Cross-Encoder Reranker	**+2.6%**	**0.0**

Table 10: Summary of Ablation Studies for ARM Retriever Components regarding the MIP solver. Performance change is relative to the baseline configuration for each specific study.

Component Under Study	Configuration	Performance Change	Overhead (s/query)
Candidate Set Size for LLM	10 Objects (Baseline)	Baseline	0.0
	30 Objects	+5.0%	~0.5
	50 Objects	**+7.0%**	**~1.0**
	100 Objects	+6.0%	~1.4
LLM Input Serialization	Full Object Content (Baseline)	Baseline (13% Hallucination)	0.0
	Object Pseudo-IDs	+10.0% (1% Hallucination)	0.0
	Grouped Source Pseudo-IDs	**+12.0% (0% Hallucination)**	**0.0**
Final Selection Beam Size	Greedy (Beam Size = 1)	Baseline	0.0
	Beam Search (Beam Size = 5)	**+1.5% Recall, -3.0% Precision**	**~1.0**

Table 11: Summary of Ablation Studies for ARM's Final Selection and Generation Stage. Performance change is relative to the baseline configuration for each study.

with a modest latency overhead of 1 second per query. Given our objective to maximize retrieval recall, the small decrease in precision and the increase in latency is an acceptable trade-off for the gain in performance, leading us to adopt a beam size of 5 for the final selection process.

7 Conclusion

In this technical report, we presented a detailed implementation of ARM, an alignment-oriented retrieval method designed to address complex, multi-source information needs. By deeply integrating Large Language Model reasoning with structured evidence alignment, ARM moves beyond conventional retrieval paradigms that treat retrievers as separate, black-box components. Through extensive benchmarking across nine diverse datasets, our experiments demonstrate that ARM consistently outperforms established baseline methods, including sparse, dense, and reranking retrievers, in both precision and recall.

A core contribution of this work is an implementation that is not only performant but also highly configurable. As detailed in our ablation studies, users can tune the retriever for specific use cases by enabling or disabling features like keyword rewriting, swapping core alignment components such as the MIP solver for a more efficient cross-encoder reranker, or even injecting external domain knowledge directly into the LLM's reasoning prompt. This flexibility allows for a precise balancing of retrieval accuracy, inference speed, and computational cost, making the framework adaptable to different operational constraints. All in all, our work shows that ARM is a promising approach that effectively bridges the gap between the reasoning capabilities of LLMs and the structured reality of complex data collections.

References

[1] Mohammad Mahdi Abootorabi, Amirhosein Zobeiri, Mahdi Dehghani, Mohammadali Mohammadkhani, Bardia Mohammadi, Omid Ghahroodi, Mahdieh Soleymani Baghshah, and Ehsaneddin Asgari. 2025. Ask in Any Modality: A Comprehensive Survey on Multimodal Retrieval-Augmented Generation. arXiv:arXiv:2502.08826

[2] Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. FEVEROUS: Fact Extraction and VERification Over Unstructured and structured

information. (June 2021). arXiv:2106.05707 [cs.CL]

[3] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In The Twelfth International Conference on Learning Representations. https://openreview.net/forum?id=hSyW5go0v8

[4] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In Findings of the Association for Computational Linguistics: ACL 2024, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2318–2335. doi:10.18653/v1/2024.findings-acl.137

[5] Nuo Chen, Linjun Shou, Ming Gong, Jian Pei, Chenyu You, Jianhui Chang, Daxin Jiang, and Jia Li. 2023. Bridge the Gap between Language models and Tabular Understanding. ArXiv abs/2302.09302 (2023). https://api.semanticscholar.org/CorpusID:257038497

[6] Peter Baile Chen, Tomer Wolfson, Michael Cafarella, and Dan Roth. 2025. EnrichIndex: Using LLMs to enrich retrieval indices offline. (April 2025). arXiv:2504.03598 [cs.CL]

[7] Peter Baile Chen, Yi Zhang, Michael Cafarella, and Dan Roth. 2025. Can we retrieve everything all at once? ARM: An alignment-oriented LLM-based retrieval method. (Jan. 2025). arXiv:2501.18539 [cs.CL]

[8] Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is Table Retrieval a Solved Problem? Exploring Join-Aware Multi-Table Retrieval. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2687–2699. doi:10.18653/v1/2024.acl-long.148

[9] Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. TableRAG: Million-token table understanding with language models. (Oct. 2024). arXiv:2410.04739 [cs.CL]

[10] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2024. Dense X Retrieval: What Retrieval Granularity Should We Use?. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 15159–15177. doi:10.18653/v1/2024.emnlp-main.845

[11] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020. Open question answering over tables and text. (Oct. 2020). arXiv:2010.10439 [cs.CL]

[12] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. TabFact: A large-scale dataset for Table-based fact verification. (Sept. 2019). arXiv:1909.02164 [cs.CL]

[13] Qinyuan Cheng, Xiaonan Li, Shimin Li, Qin Zhu, Zhangyue Yin, Yunfan Shao, Linyang Li, Tianxiang Sun, Hang Yan, and Xipeng Qiu. 2024. Unified Active Retrieval for Retrieval Augmented Generation. In Findings of the Association for Computational Linguistics: EMNLP 2024, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 17153–17166. doi:10.18653/v1/2024.findings-emnlp.999

[14] Yashar Deldjoo, Zhankui He, Julian McAuley, Anton Korikov, Scott Sanner, Arnau Ramisa, René Vidal, Maheswaran Sathiamoorthy, Atoosa Kasirzadeh, and Silvia Milano. 2024. A Review of Modern Recommender Systems Using Generative Models (Gen-RecSys). In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (Barcelona, Spain) (KDD '24). Association for Computing Machinery, New York, NY, USA, 6448–6458. doi:10.1145/3637528.3671474

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In North American Chapter of the Association for Computational Linguistics. https://api.semanticscholar.org/CorpusID:52967399

[16] P. Ferragina and G. Manzini. 2000. Opportunistic data structures with applications. In Proceedings 41st Annual Symposium on Foundations of Computer Science. 390–398. doi:10.1109/SFCS.2000.892127

[17] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:arXiv:2312.10997

[18] Yunfan Gao, Yun Xiong, Yijie Zhong, Yuxi Bi, Ming Xue, and Haofen Wang. 2025. Synergizing RAG and Reasoning: A Systematic Review. arXiv:arXiv:2504.15909

[19] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. ImageBind: One Embedding Space To Bind Them All. In CVPR.

[20] Xinyan Guan, Jiali Zeng, Fandong Meng, Chunlei Xin, Yaojie Lu, Hongyu Lin, Xianpei Han, Le Sun, and Jie Zhou. 2025. DeepRAG: Thinking to retrieve step by step for large Language Models. (Feb. 2025). arXiv:2502.01142 [cs.AI]

[21] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, 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, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 4320–4333. doi:10.18653/v1/2020.acl-main.398

[22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In International Conference on Learning Representations. https://openreview.net/forum?id=nZeVKeeFYf9

[23] Qisheng Hu, Quanyu Long, and Wenya Wang. 2025. Coordinating search-informed reasoning and reasoning-guided search in claim verification. (June 2025). arXiv:2506.07528 [cs.AI]

[24] Qisheng Hu, Quanyu Long, and Wenya Wang. 2025. Coordinating search-informed reasoning and reasoning-guided search in claim verification. (June 2025). arXiv:2506.07528 [cs.AI]

[25] Palak Jain, Livio Baldini Soares, and Tom Kwiatkowski. 2024. From RAG to Riches: Retrieval Interlaced with Sequence Generation. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 8887–8904. doi:10.18653/v1/2024.emnlp-main.502

[26] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 7036–7050. doi:10.18653/v1/2024.naacl-long.389

[27] Xingyu Ji, Parker Glenn, Aditya G Parameswaran, and Madelon Hulsebos. 2025. TARGET: Benchmarking TAble Retrieval for GEnerative Tasks. (May 2025). arXiv:2505.11545 [cs.IR]

[28] Jun-Peng Jiang, Si-Yang Liu, Hao-Run Cai, Qile Zhou, and Han-Jia Ye. 2025. Representation Learning for Tabular Data: A Comprehensive Survey. arXiv preprint arXiv:2504.16109 (2025).

[29] Liang Jintao, Su Gang, Lin Huifeng, Wu You, Zhao Rui, and Li Ziyue. 2025. Reasoning RAG via system 1 or system 2: A survey on Reasoning Agentic retrieval-Augmented Generation for industry challenges. (June 2025). arXiv:2506.10408 [cs.AI]

[30] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550

[31] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (Virtual Event, China) (SIGIR '20). Association for Computing Machinery, New York, NY, USA, 39–48. doi:10.1145/3397271.3401075

[32] Enas Khwaileh and Yannis Velegrakis. 2025. Dataset Discovery using Semantic Matching. In Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero 0001, and Petar Jovanovic 0001 (Eds.). OpenProceedings.org, 649–660. doi:10.48786/edbt.2025.52

[33] Pratyush Kumar, Kuber Vijaykumar Bellad, Bharat Vadlamudi, and Aman Chadha. 2024. RoundTable: Leveraging dynamic schema and contextual autocomplete for enhanced query precision in tabular question answering. (Aug. 2024). arXiv:2408.12369 [cs.AI]

[34] Myeonghwa Lee, Seonho An, and Min-Soo Kim. 2024. PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 6537–6555. doi:10.18653/v1/2024.naacl-long.364

[35] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. Proc. ACM Manag. Data 2, 3, Article 127 (May 2024), 28 pages. doi:10.1145/3654930

[36] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C C Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as A database interface? A BIg bench for large-scale database grounded text-to-SQLs. (May 2023). arXiv:2305.03111 [cs.CL]

[37] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. 2025. From Matching to Generation: A Survey on Generative Information Retrieval. ACM Trans. Inf. Syst. 43, 3, Article 83 (May 2025), 62 pages. doi:10.1145/3722552

[38] Hao Liao, Wensheng Lu, Jianxun Lian, Mingqi Wu, Shuo Wang, Yong Zhang, Yitian Huang, Mingyang Zhou, and Xing Xie. 2025. Avoid recommending out-of-domain items: Constrained generative recommendation with LLMs. (May 2025).

arXiv:2505.03336 [cs.IR]

[39] Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing Text-to-SQL Parsing through Question Rewriting and Execution-Guided Refinement. In Findings of the Association for Computational Linguistics: ACL 2024, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2009–2024. doi:10.18653/v1/2024.findings-acl.120

[40] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2021. FeTaQA: Free-form table question answering. (April 2021). arXiv:2104.00369 [cs.CL]

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In International Conference on Machine Learning. https://api.semanticscholar.org/CorpusID:231591445

[42] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. Found. Trends Inf. Retr. 3, 4 (April 2009), 333–389. doi:10.1561/1500000019

[43] Kwangwook Seo, Donguk Kwon, and Dongha Lee. 2025. MT-RAIG: Novel benchmark and evaluation framework for Retrieval-Augmented Insight Generation over multiple tables. (Feb. 2025). arXiv:2502.11735 [cs.CL]

[44] Rulin Shao, Rui Qiao, Varsha Kishore, Niklas Muennighoff, Xi Victoria Lin, Daniela Rus, Bryan Kian Hsiang Low, Sewon Min, Wen tau Yih, Pang Wei Koh, and Luke Zettlemoyer. 2025. ReasonIR: Training Retrievers for Reasoning Tasks. arXiv preprint arXiv:2504.20595 (2025). https://arxiv.org/abs/2504.20595

[45] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. In Findings of the Association for Computational Linguistics: EMNLP 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 9248–9274. doi:10.18653/v1/2023.findings-emnlp.620

[46] Weihang Su, Qingyao Ai, Jingtao Zhan, Qian Dong, and Yiqun Liu. 2025. Dynamic and Parametric Retrieval-Augmented Generation. arXiv:arXiv:2506.06704

[47] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual harnessing for efficient SQL synthesis. (May 2024). arXiv:2405.16755 [cs.LG]

[48] Yixuan Tang and Yi Yang. 2024. MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. (Jan. 2024). arXiv:2401.15391 [cs.CL]

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In Advances in Neural Information Processing Systems, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In Advances in Neural Information Processing Systems, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[51] Jiajia Wang, Jimmy Xiangji Huang, Xinhui Tu, Junmei Wang, Angela Jennifer Huang, Md Tahmid Rahman Laskar, and Amran Bhuiyan. 2024. Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges. ACM Comput. Surv. 56, 7, Article 185 (April 2024), 33 pages. doi:10.1145/3648471

[52] Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data Discovery Using Natural Language Questions Via A Self-Supervised Approach. Proc. ACM Manag. Data 1, 4, Article 262 (Dec. 2023), 27 pages. doi:10.1145/3626756

[53] Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Guanglin Niu, Tongliang Li, and Zhoujun Li. 2024. TableBench: A comprehensive and complex benchmark for table question answering. (Aug. 2024). arXiv:2408.09174 [cs.CL]

[54] Zekun Xi, Wenbiao Yin, Jizhan Fang, Jialong Wu, Runnan Fang, Ningyu Zhang, Jiang Yong, Pengjun Xie, Fei Huang, and Huajun Chen. 2025. OmniThink: Expanding Knowledge Boundaries in Machine Writing through Thinking. arXiv:2501.09751 [cs.CL] https://arxiv.org/abs/2501.09751

[55] 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).

[56] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 8413–8426. doi:10.18653/v1/2020.acl-main.745

[57] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. 2024. A survey on multimodal large language models. National Science Review 11, 12 (11 2024), nwae403. arXiv:https://academic.oup.com/nsr/article-pdf/11/12/nwae403/61201557/nwae403.pdf doi:10.1093/nsr/nwae403

[58] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. (Sept. 2018). arXiv:1809.08887 [cs.CL]

[59] Hansi Zeng, Chen Luo, Bowen Jin, Sheikh Muhammad Sarwar, Tianxin Wei, and Hamed Zamani. 2024. Scalable and Effective Generative Information Retrieval. In Proceedings of the ACM Web Conference 2024 (Singapore, Singapore) (WWW '24). Association for Computing Machinery, New York, NY, USA, 1441–1452. doi:10.1145/3589334.3645477

[60] Haoxiang Zhang, Yurong Liu, Wei-Lun, Hung, Aécio Santos, and Juliana Freire. 2025. AutoDDG: Automated dataset description generation using large language models. (Feb. 2025). arXiv:2502.01050 [cs.DB]

[61] Xiaoming Zhang, Ming Wang, Xiaocui Yang, Daling Wang, Shi Feng, and Yifei Zhang. 2024. Hierarchical Retrieval-Augmented Generation Model with Rethink for multi-hop Question Answering. (Aug. 2024). arXiv:2408.11875 [cs.CL]

[62] Zhuocheng Zhang, Yang Feng, and Min Zhang. 2025. LevelRAG: Enhancing Retrieval-Augmented Generation with Multi-hop Logic Planning over Rewriting Augmented Searchers. arXiv:2502.18139 [cs.CL] https://arxiv.org/abs/2502.18139

[63] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH ensemble: internet-scale domain search. Proc. VLDB Endow. 9, 12 (Aug. 2016), 1185–1196. doi:10.14778/2994509.2994534