Multi-Hop Retrieval-Augmented Generation for Question Answering - Group 8

Pham Minh Hieu Nguyen Nhu Giap Nguyen Vo Ngoc Khue
Phan Minh Hoa Nguyen Tri Thanh
Pham Dang Khue

Hanoi University of Science and Technology

June 11, 2025

Abstract

Retrieval-Augmented Generation (RAG) has become a key technique for mitigating large-language-model hallucinations by supplementing parametric knowledge with external evidence. However, most systems assume that all supporting facts can be retrieved in a single pass, which fails on multi-hop questions whose evidence is distributed across disparate documents. In this work, we perform a systematic comparison of four multi-hop RAG paradigms on the MultiHop-RAG benchmark: (i) Query Decomposition, where complex questions are first broken into sequential sub-queries whose individual retrievals are later aggregated; (ii) TreeHop-RAG, a lightweight embedding-space update method that refines query embeddings without LLM rewrites; (iii) Multi-Meta-RAG, which extracts source and temporal metadata from the query to pre-filter the corpus; and (iv) IR-CoT, which interleaves BM25 retrieval with chain-of-thought reasoning. Evaluated on 2,556 multi-hop questions, IR-CoT with 10 retrieved chunks per hop achieves the best trade-off (Precision = 0.66, Recall = 0.89, F1 = 0.76) while converging in only 1.2 iterations on average. Query Decomposition attains moderate recall gains with minimal sub-query granularity (F1 = 0.336 at three decompositions) but suffers diminishing returns as more sub-queries are added. TreeHop-RAG excels in one-hop precision (F1 = 0.38) yet sees steep precision drop-off in later hops, whereas Multi-Meta-RAG boosts early recall before plateauing. Our hop-conditioned F1 metric illuminates the distinct strengths and failure modes of each method, offering actionable guidance for designing efficient, accurate multi-hop QA systems.

Keywords: *RAG · Multi-hop RAG · Question Answering · Natural Language Processing · Large Language Model.*

1 Introduction

Question answering (QA) has evolved from rule-based systems, through statistically trained passage retrieval, to the current era of large language models (LLMs). Despite the remarkable breadth of knowledge encoded in modern LLMs, their parametric memory remains static, expensive to refresh, and prone to hallucinating facts absent from training data. Retrieval-Augmented Generation (RAG) mitigates these weaknesses by coupling a neural retriever with a generator: external documents are fetched at inference time and provided to the language model as additional context. This paradigm has become a de-facto standard for knowledge-intensive NLP tasks, yielding substantial gains in factual accuracy and temporal adaptability.

Most deployed RAG systems, however, operate in a single-hop regime. They issue one retrieval request driven solely by the original question and assume that all requisite evidence is co-located in the top-k documents. Such an assumption is brittle. First, information relevant to complex questions is often dispersed across multiple sources. Second, the initial query may lack the lexical hooks needed to surface later-hop evidence. Third, single-hop RAG typically fails on compositional questions that require sequential reasoning. Consider the query

"Which university is attended by the president of the country that borders Ukraine and uses the euro?"

Answering correctly entails three chained sub-problems: identify the country satisfying the geographic and currency constraints (Slovakia), retrieve the current president of that country (Zuzana Čaputová), and finally locate her alma mater (Comenius University). A single retrieval step rarely returns documents containing all three pieces, leading to propagation of partial or erroneous evidence.

These shortcomings motivate the study of multi-hop RAG, in which retrieval and reasoning are interleaved over several iterations. Recent proposals differ in how they update queries, prune search space, and integrate intermediate evidence, yet a systematic comparison of their design trade-offs remains scarce.

The present work addresses this gap. We implement and contrast three representative multi-hop RAG frameworks: (i) TreeHop-RAG, which explores a branching space of embedding-level query refinements, (ii) Multi-Meta-RAG, which exploits LLM-extracted metadata to filter the corpus before each hop, and (iii) IR-CoT, which interlaces retriever calls with chain-of-thought reasoning produced by an LLM backbone. All methods are evaluated on a benchmark suite of multi-hop questions and an accompanying evidence corpus hosted at `https://drive.google.com/drive/folders/1e2wSbCQj1tMOEinb0_cDfC0Cwc08_3HC`

The remainder of the paper details the methodological differences among these systems, describes the experimental setup, and reports empirical findings that illuminate the strengths and limitations of current multi-hop RAG technology.

2 Methodology

2.1 TreeHop-RAG

2.1.1 Overview

TreeHop-RAG introduces a novel multi-hop retrieval mechanism that operates entirely within the embedding space, effectively addressing the computational overhead associated with iterative language-model (LLM) query rewrites commonly employed in traditional multi-hop retrieval-augmented generation (RAG). Unlike conventional RAG frameworks, which alternate between expensive textual query rewriting and embedding-based retrieval, TreeHop-RAG compresses the iterative process into a lightweight embedding update operation. This eliminates the need for LLM-generated intermediate queries, drastically improving computational efficiency while retaining high retrieval accuracy.

2.1.2 Problem Formulation

Formally, given an initial query embedding $q_r \in \mathbb{R}^d$ at retrieval hop r, and a vector retriever $g(\cdot, K)$ that returns the top-K document embeddings $\{c_r^{(i)}\}_{i=1}^K$, the central objective of TreeHop-RAG is to generate a refined set of embeddings $\{q_{r+1}^{(i)}\}_{i=1}^K$ for subsequent retrieval:

$$q_{r+1}^{(i)} = \mathrm{TreeHop}(q_r, c_r^{(i)}), \quad c_r^{(i)} \in g(q_r, K).$$

This embedding-based update method forms an implicit search tree structure, with nodes represented as embeddings rather than explicit textual queries. Consequently, TreeHop efficiently explores multiple knowledge paths through embedding-based branches.

2.1.3 Embedding-Space Query Refinement

The embedding update process in TreeHop-RAG employs two complementary mechanisms: (i) **overlap suppression** and (ii) **salient-information injection**.

Overlap Suppression: To mitigate redundancy, TreeHop computes the semantic difference between the current query embedding q_r and the retrieved chunk embedding c_r. Formally, the redundancy-suppressing update is defined as:

$$\Delta q = q_r - c_r.$$

This operation explicitly removes the shared semantic information that has already been covered by previous retrieval, guiding subsequent hops towards novel knowledge regions.

Salient-Information Injection: To selectively integrate novel and useful information from the retrieved document chunk, TreeHop employs a learned single-head cross-attention mechanism, termed the **UpdateGate**, parameterized by projection matrices $Q_u, K_u, V_u \in \mathbb{R}^{d\times d}$. Given embeddings q_r (query) and c_r (chunk), the UpdateGate computes a context-dependent embedding update:

$$\mathrm{UpdateGate}(q_r, c_r) = \mathrm{softmax}\left(\frac{(Q_u q_r)(K_u c_r)^T}{\sqrt{d}}\right) V_u c_r.$$

This mechanism allows TreeHop to dynamically focus on relevant semantic dimensions of the chunk embedding that are most informative for answering the query, thus producing a refined embedding:

$$q_{r+1} = \Delta q + \mathrm{UpdateGate}(q_r, c_r).$$

Ablation experiments provided in the original paper verify the indispensability of both overlap suppression and the salient-information injection, with the absence of either component severely deteriorating retrieval performance.

2.1.4 Branching Control and Pruning Strategy

Unrestricted iterative retrieval naturally incurs exponential computational complexity ($O(K^r)$). TreeHop mitigates this combinatorial explosion via two rule-based pruning strategies applied at each hop:

1. **Redundancy Pruning**: A branch is terminated if it retrieves a chunk previously encountered within the same path, thereby avoiding cyclical or redundant retrieval paths.

2. **Layer-wise Top-K Selection**: Among all newly generated query embeddings at each subsequent hop, TreeHop retains only the top-K embeddings whose associated chunks exhibit the highest cosine similarity with respect to their parent query embeddings. Remaining embeddings and their retrieval paths are discarded.

These pruning heuristics effectively constrain the growth of the embedding search tree, empirically reducing second-hop branching from approximately 25 to about 7–9 active branches per query, thus ensuring manageable complexity without significant accuracy degradation.

2.1.5 Training Paradigm

To train the embedding-update mechanism, TreeHop-RAG leverages a large-scale dataset curated specifically for multi-hop retrieval, comprising approximately 111K multi-hop training examples distilled from standard benchmark datasets such as 2WikiMultiHop, MuSiQue, and MultiHop-RAG. Instances requiring genuine multi-hop reasoning—such as inference-based queries, compositional questions, and bridge-comparison questions—were selected to explicitly capture the challenges posed by multi-hop retrieval.

During training, initial embeddings were computed by a pretrained frozen embedding encoder (BGE-m3). TreeHop was subsequently optimized using the InfoNCE contrastive loss to encourage query embeddings to align with correct next-hop chunk embeddings while distancing irrelevant ones. Concretely, each training instance involved one positive embedding pair and five in-batch negative pairs. Optimization was carried out via the AdamW algorithm (learning rate 6×10^{-5}, batch size 64, temperature 0.15) on a single NVIDIA V100 GPU.

2.1.6 Computational Efficiency and Complexity Analysis

TreeHop-RAG exhibits significant computational advantages due to its streamlined embedding-based update operation. Each hop requires only a single matrix attention computation with complexity $O(d^2)$. Given the pruning constraints, overall retrieval complexity scales linearly with respect to the number of retrieval hops. Empirical evaluations demonstrate remarkable latency reductions: answering two-hop queries in approximately 22 milliseconds on an NVIDIA A100 GPU. This represents an approximately 99% latency reduction compared to state-of-the-art LLM-driven iterative retrieval-generation baselines (e.g., Iter-RetGen), without sacrificing retrieval accuracy—achieving competitive Recall@5 metrics across multiple benchmark datasets.

2.1.7 Summary

In summary, TreeHop-RAG presents a computationally efficient, parameter-light multi-hop retrieval approach that achieves state-of-the-art retrieval accuracy without relying on expensive LLM query rewrites. Its embedding-based query refinement and controlled branching strategies enable real-time performance, making TreeHop-RAG particularly suitable for deployment in latency-sensitive, resource-constrained knowledge-intensive QA applications.

2.2 Multi-Meta-RAG

2.2.1 Motivation and Overview

Standard Retrieval-Augmented Generation (RAG) approaches rely primarily on dense vector similarity between a query and candidate chunks from a large corpus. However, in complex multi-hop question-answering scenarios, relying solely on semantic similarity frequently leads to the retrieval of irrelevant chunks that superficially match lexical or semantic cues but fail to fulfill the specific factual constraints implied by the query. Multi-Meta-RAG explicitly addresses this limitation by leveraging query-extracted metadata constraints, such as the source or publication date, to pre-filter documents prior to dense retrieval, substantially reducing irrelevant candidate chunks and boosting retrieval precision for compositional multi-hop queries.

2.2.2 Query Metadata Extraction

The fundamental innovation in Multi-Meta-RAG is a metadata-driven filtering strategy facilitated by a query-parsing mechanism. Concretely, for each query q, a helper language model h_θ (ChatGPT-3.5-Turbo) is utilized to infer explicit metadata constraints embedded in the query text. This model is invoked through few-shot prompting, which avoids the computational and data overhead associated with fine-tuning. Given the query text q, the extraction model outputs structured logical constraints on two critical metadata fields, `source` (e.g., publication name) and `published_at` (publication date), as formalized below:

$$\text{Filter}(q) = \{\text{source} : [s_1, s_2, ..., s_m], \quad \text{published\_at} : [t_1, t_2, ..., t_n]\}.$$

Empirical analysis in the original work demonstrates that metadata extraction is robust, producing non-empty source constraints in nearly 100% of the cases, and additional temporal constraints (pub-

lication dates) in approximately 16% of queries. Crucially, this step introduces minimal latency overhead, averaging approximately 0.7 seconds per query, thus remaining computationally feasible in real-time settings.

2.2.3 Metadata-Enriched Document Representation and Storage

Multi-Meta-RAG employs a specialized metadata-aware document indexing strategy. Each document in the corpus is segmented into chunks of 256 tokens with an overlap of 32 tokens to preserve continuity across segments. Subsequently, each chunk d_i is represented as a dense embedding vector $e_i = f_\varphi(d_i) \in \mathbb{R}^d$, computed using a fixed, pretrained embedding model such as **bge-large-en-v1.5**.

Critically, each chunk embedding is stored alongside structured metadata attributes:

$$\text{Chunk Storage: } (e_i, \text{meta}(d_i)),$$

where $\text{meta}(d_i)$ encapsulates properties such as the original source publication (e.g., *TechCrunch, The Verge*) and timestamp (`published_at`). These metadata-rich embeddings are indexed in a Neo4j vector database, allowing for efficient retrieval operations conditioned explicitly on structured metadata filters.

2.2.4 Metadata-Constrained Retrieval Pipeline

Given a query q, Multi-Meta-RAG executes retrieval in four rigorously defined stages:

1. **Embedding Generation**: The query is first converted into a dense vector representation $q_{\text{emb}} = f_\varphi(q)$, using the identical frozen embedding encoder applied to document chunks, ensuring consistent semantic alignment.

2. **Metadata Extraction**: The metadata constraints are inferred via the LLM extraction model as described in Section 3.2, generating the filter set $\text{Filter}(q)$.

3. **Metadata-Constrained Similarity Search**: With both the embedding q_{emb} and constraints $\text{Filter}(q)$, the retrieval system performs constrained vector similarity search within the Neo4j graph-based vector store:

$$\mathcal{C} = \arg\operatorname*{topK}_{d_i} \left(\langle q_{\text{emb}}, e_i \rangle \mid \text{meta}(d_i) \models \text{Filter}(q) \right).$$

   Here, similarity search is performed exclusively over the subset of document chunks satisfying the metadata constraints, effectively eliminating irrelevant candidates before dense similarity ranking.

4. **Neural Cross-Encoder Re-ranking**: A powerful neural cross-encoder r_ψ (e.g., **bge-reranker-large**) is subsequently employed to refine the retrieved candidate set $\mathcal{C}$. The cross-encoder computes precise interaction scores between the query and each candidate chunk, further improving precision by selecting the top-ranked K' (default $K' = 6$) chunks:

$$\mathcal{C}' = \arg\operatorname*{topK'}_{c \in \mathcal{C}} r_\psi(q, c).$$

This integrated pipeline ensures the language model generation step receives highly relevant and factually accurate context, explicitly conforming to query-imposed metadata constraints.

2.2.5 Computational Complexity and Efficiency Analysis

The computational efficiency of Multi-Meta-RAG primarily hinges on two components: metadata extraction and constrained vector retrieval. Metadata extraction introduces minimal overhead due to its single-pass LLM inference, remaining constant per query. Meanwhile, the complexity of the constrained vector search scales as $O(K \log M')$, where $M' \ll M$ represents the significantly reduced search space after metadata filtering, and K denotes retrieval depth.

Empirical benchmarking demonstrates that this combined metadata extraction and retrieval pipeline achieves end-to-end latencies averaging approximately 200 milliseconds per query on standard hardware, marking it viable for real-time deployment in latency-sensitive environments.

2.2.6 Empirical Validation and Performance Gains

Evaluated extensively on the MultiHop-RAG benchmark, Multi-Meta-RAG demonstrates compelling empirical improvements. Specifically, employing the metadata-filtering approach achieves a substantial gain in retrieval precision, improving **Hits@4** from 0.663 (baseline RAG) to 0.792 (Multi-Meta-RAG with voyage-02 embedding), representing a 19.5% relative improvement. Consequently, this improved retrieval precision significantly enhances downstream generation accuracy, elevating PaLM-2 generated answer correctness by 25.6% over the baseline RAG configuration.

Critically, these empirical gains are achieved entirely without task-specific fine-tuning of embedding models or re-rankers, underscoring the generality and effectiveness of leveraging explicit query-imposed metadata constraints to structurally enhance multi-hop retrieval accuracy.

2.2.7 Summary

In summary, Multi-Meta-RAG's methodology provides a principled, efficient, and robust approach to enhancing retrieval accuracy in complex multi-hop question-answering tasks through structured metadata extraction and constrained retrieval. By explicitly extracting and leveraging logical constraints directly embedded in queries, Multi-Meta-RAG effectively addresses key limitations of traditional similarity-based retrieval, demonstrating significant empirical improvements while maintaining computational tractability and scalability.

2.3 Interleaving Retrieval with Chain-of-Thought Reasoning (IR-CoT)

2.3.1 Motivation and Overview

Conventional retrieval-augmented generation (RAG) systems typically rely on single-step retrieval strategies, which assume all necessary evidence can be effectively identified and extracted using a single initial query. This assumption frequently fails in multi-hop question answering tasks, where accurate reasoning requires chaining multiple facts across several documents. Interleaving Retrieval with Chain-of-Thought (IR-CoT) explicitly addresses this issue by dynamically interleaving iterative retrieval steps with intermediate reasoning steps generated through chain-of-thought (CoT) prompting. By incrementally refining the retrieval query using intermediate reasoning steps, IR-CoT significantly enhances the alignment between retrieved evidence and model-generated reasoning, thereby mitigating the hallucination problem and substantially improving retrieval accuracy for compositional questions.

2.3.2 System Architecture and Components

The IR-CoT framework consists of three core components working synergistically:

- **Base Retriever**: A lexical retriever based on the BM25 ranking algorithm implemented via Elasticsearch. Given a query q, it retrieves the top-K relevant paragraphs from the document corpus.

- **Chain-of-Thought Language Model**: A CoT-capable generative language model (h_θ) such as GPT-3 (`code-davinci-002`) or Flan-T5 (XL or XXL). Crucially, this model is employed exclusively for inference, requiring no additional fine-tuning beyond zero- or few-shot prompting.

- **In-context Demonstrations**: A small number (20 examples per dataset) of manually-crafted demonstration instances consisting of complete CoT reasonings paired with gold supporting paragraphs. These exemplars are provided as conditioning contexts at inference time to guide reasoning and retrieval behavior.

2.3.3 Detailed Algorithmic Procedure

Given a multi-hop question Q, IR-CoT employs an iterative "Retrieve-Reason-Retrieve" strategy, formally structured as follows:

Step 1: Initial Paragraph Retrieval The initial retrieval employs the original user question Q as input to the base retriever:

$$\mathcal{P}_0 = g(Q, K),$$

where g denotes the BM25 retriever and $\mathcal{P}_0$ represents the initial set of retrieved paragraphs.

Step 2: Iterative CoT-Enhanced Retrieval Loop IR-CoT then enters an iterative loop with a maximum of $R_{\max}$ (default 8) reasoning-retrieval iterations. For each iteration $r = 1, \ldots, R_{\max}$:

- **Reasoning Step**: The CoT language model is prompted with the user question, along with all currently retrieved paragraphs up to iteration $r-1$:

$$\text{Prompt}(r) = \underbrace{[\text{Title}_1 : P_1, \ldots, \text{Title}_n : P_n]}_{\text{retrieved paragraphs}} + \underbrace{Q : \text{ user question}}_{\text{query context}} + \underbrace{A : T_1, T_2, \ldots, T_{r-1}}_{\text{previous CoT reasoning steps}},$$

  where T_i denotes the generated reasoning sentence at step i. The model outputs the next reasoning sentence T_r:

$$T_r = h_\theta\left(\text{Prompt}(r)\right).$$

  Importantly, IR-CoT retains only the first generated sentence of T_r, which concisely encapsulates the next intermediate reasoning step.

- **Retrieval Step**: The newly generated intermediate reasoning sentence T_r is then immediately employed as a query to retrieve additional evidence paragraphs:

$$\mathcal{P}_r = g(T_r, K).$$

  The retrieved set $\mathcal{P}_r$ is appended to the cumulative set of retrieved paragraphs:

$$\mathcal{P} = \bigcup_{i=0}^{r} \mathcal{P}_i.$$

- **Termination Criterion**: The iterative process continues until either the CoT-generated sentence explicitly produces a terminating lexical pattern (e.g., "The answer is:"), or the predefined maximum number of reasoning iterations $R_{\max}$ is reached.

Step 3: Final Paragraph Selection and QA Generation At the conclusion of the iterative procedure, IR-CoT compiles a final evidence set by retaining up to a maximum budget of 15 paragraphs:

$$|\mathcal{P}| \leq 15,$$

ensuring adherence to context window limitations. This final paragraph set $\mathcal{P}$ is then provided as input context to a subsequent QA reading module (a separate LLM inference), which produces the final answer to the original user question.

2.3.4 Prompt Engineering and In-Context Learning

Effective prompt engineering plays a critical role in IR-CoT's success. Specifically, in-context examples provided as prompts contain complete CoT reasoning steps paired explicitly with correct supporting paragraphs. To improve model robustness against distractor documents, each exemplar contains M randomly selected distractor paragraphs interspersed among the correct paragraphs, where $M \in \{1, 2, 3\}$. Furthermore, during inference, the prompt is carefully structured, presenting each retrieved paragraph with explicit provenance information (title) to aid the language model's interpretability and reasoning accuracy.

2.3.5 Hyperparameter Settings and Configuration

IR-CoT's performance is sensitive to three key hyperparameters, empirically tuned using development-set recall metrics:

- **Retrieval Fan-Out (K)**: Defines the number of paragraphs retrieved at each iterative step, evaluated across $\{2, 4, 6, 8\}$, typically set to maximize recall.
- **Maximum Iterations ($R_{\max}$)**: Capped at 8 to balance comprehensive multi-hop coverage with computational tractability.
- **Final Paragraph Budget**: Limited to a maximum of 15 paragraphs to conform to practical LLM context window constraints.

2.3.6 Computational Complexity and Efficiency Analysis

Each IR-CoT iteration incurs exactly one retrieval step (BM25 query) and one LLM inference step (CoT reasoning). Thus, the overall computational complexity scales linearly with the number of iterations $R_{\max}$. Practically, IR-CoT achieves competitive latency performance on commodity GPUs, completing two-hop retrieval within approximately 200 milliseconds per query, demonstrating its viability for real-time QA applications.

2.3.7 Empirical Validation and Comparative Performance

Empirical evaluations conducted across prominent multi-hop benchmarks (e.g., HotpotQA, 2WikiMultiHopQA, MuSiQue, IIRC) showcase significant performance advantages of IR-CoT. Relative to standard single-step retrieval methods, IR-CoT consistently improves retrieval recall metrics (Hits@K) by margins ranging from 11 to 22 percentage points, correspondingly enhancing downstream QA accuracy metrics (exact-match and F1 scores) by up to 15 points. Notably, IR-CoT's iterative retrieval-reasoning interleaving strategy reduces factual errors (hallucinations) within generated answers by approximately 50% compared to baselines, and remains robust when employing relatively compact LLMs ($\leq$ 3 billion parameters).

2.3.8 Summary

In summary, IR-CoT contributes a robust, efficient, and effective approach to multi-hop question answering by explicitly interleaving intermediate retrieval steps with incremental CoT reasoning. Its methodical integration of retrieval and reasoning steps substantially enhances retrieval precision and downstream QA accuracy, addressing key limitations of traditional single-hop retrieval. IR-CoT's straightforward integration into existing RAG pipelines, minimal computational overhead, and significant empirical improvements establish it as a potent baseline for future research in multi-hop retrieval-augmented QA systems.

2.4 Query Decomposition for Multi-hop Retrieval-Augmented Generation

2.4.1 Motivation and Overview

Standard Retrieval-Augmented Generation (RAG) approaches typically rely on dense vector similarity between a user query and candidate chunks from a large corpus. While effective for simple queries, these methods often struggle in multi-hop scenarios, where answering a query requires gathering and reasoning over information from multiple interconnected documents. In such cases, a single-pass retrieval strategy may fail to retrieve all the necessary supporting documents.

To overcome this limitation, we adopt a *Query Decomposition* strategy. The central idea is to treat the original query as a complex problem that can be broken down into simpler, more manageable sub-queries. By addressing each sub-query individually and later aggregating their results, we increase the likelihood of retrieving all relevant pieces of information required to answer the original query accurately.

2.4.2 Algorithmic Procedure

Our approach consists of the following steps:

1. **Corpus Preparation:** The entire corpus is first segmented into chunks. Each chunk is then embedded using a powerful Large Language Model (LLM), such as GPT-4. The resulting dense vector representations are stored in a vector database to facilitate efficient similarity search.

2. **Query Decomposition:** Given a complex query q, an LLM is used to decompose q into a set of N sub-queries $\{q_1, q_2, \ldots, q_N\}$, where N is a predefined parameter reflecting the expected decomposition granularity.

3. **Retrieval:** For each sub-query q_i, we retrieve the top-K most relevant document chunks based on vector similarity. This yields a set of potentially overlapping document sets for each q_i.

4. **Aggregation and Generation:** The retrieved documents from all sub-queries are aggregated into a single context. This aggregated context is then passed to the LLM, which synthesizes a final answer to the original query q.

This query decomposition-based retrieval method enhances the ability of the system to perform multi-hop reasoning by explicitly guiding the retrieval process through intermediate sub-queries, thereby improving coverage and relevance of the retrieved context.

3 Experimental Results

3.1 Dataset

3.1.1 Corpus

The external knowledge store supplied to all models is a heterogeneous news-centric corpus distributed as **multihoprag_corpus.txt** (6.4 MB). Each entry contains two explicit textual fields

```
Title:   <headline>
Passage: <full paragraph-level content>
```

and is delimited by the sentinel tag `<endofpassage>`. A simple lexical scan identifies **609** titled documents spanning technology, finance, sports and culture. All articles are contemporary (September–December 2023) and originate from more than fifty distinct media outlets (e.g., *TechCrunch, The Verge, Fortune, Sporting News*). Document length ranges from short bulletins ($\approx$120 tokens) to long-form reportage ($>$1,000 tokens), ensuring retrieval difficulty is not trivially correlated with passage size.

3.1.2 Question–Answer Set

The supervision file **MultiHopRAG.json** provides the multi-hop evaluation suite. Each record comprises a natural-language `query`, a gold-standard `answer`, a machine-readable `question_type`, and an `evidence_list` of source snippets. Parsing the JSON reveals:

Statistic	Value
Total QA pairs	**2,556**
Avg. evidence chunks / question	**2.38**
Max. evidence chunks / question	4
Distinct sources in evidence	58

The annotation schema distinguishes four reasoning categories:

Question type	Count	Share
`comparison_query`	856	33.5 %
`inference_query`	816	31.9 %
`temporal_query`	583	22.8 %
`null_query`*	301	11.8 %

*`null_query` instances intentionally lack sufficient evidence in the corpus, forcing the system to abstain.

Evidence provenance is highly skewed toward technology-oriented outlets, *TechCrunch* (2,199 citations) and *The Verge* (841) dominate, yet non-tech domains (sports, finance, entertainment) remain well represented. Publication dates cluster tightly between **2023-09-26** and **2023-12-25**, enabling controlled evaluation of temporal-reasoning abilities.

3.1.3 Multi-Hop Characteristics

Each non-null query was curated to require **at least two distinct evidence items drawn from different documents**. Manual inspection confirms that retrieved passages seldom co-occur in a single article, enforcing genuine multi-hop retrieval. Typical phenomena include:

- **Cross-source synthesis** – combining facts from, say, a *Fortune* business report and a *TechCrunch* technology brief.

- **Temporal anchoring** – verifying chronology (e.g., "earlier than", "after the lawsuit").

- **Logical comparison** – determining consistency or contradiction across outlets.

Consequently, single-shot retrieval baselines achieve less than 0.40 Hits@4, underscoring the dataset's difficulty and its suitability for benchmarking the advanced multi-hop RAG strategies introduced in this work.

3.2 Evaluation Metric

To assess each retrieval pipeline we adopt a hop-conditioned F_1 framework that jointly captures retrieval quality and the number of retrieval rounds required to attain it. Let

- $R(q) \subset \mathcal{C}$ be the set of gold-standard evidence chunks for query q,

- $D_r(q) = \bigcup_{i=1}^{r} G_i(q)$ be the accumulated set of chunks returned by the system after the first r retrieval hops, where $G_i(q)$ denotes the chunks fetched at hop i.

For every hop $r \in \{1, \ldots, R_{\max}\}$ we compute

$$\begin{aligned} \text{Precision}_r(q) &= \frac{|D_r(q) \cap R(q)|}{|D_r(q)|}, \\ \text{Recall}_r(q) &= \frac{|D_r(q) \cap R(q)|}{|R(q)|}, \\ F_{1,r}(q) &= \frac{2 \cdot \text{Precision}_r(q) \cdot \text{Recall}_r(q)}{\text{Precision}_r(q) + \text{Recall}_r(q)}. \end{aligned}$$

System-level scores are obtained by macro-averaging over the full set $\mathcal{Q}$ of 2,556 evaluation questions:

$$F_{1,r} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} F_{1,r}(q).$$

Accumulated retrieval Because $D_r(q) \supseteq D_{r-1}(q)$, each additional hop introduces strictly more candidate chunks. Consequently

- $\text{Recall}_r(q)$ is monotone non-decreasing in r (the numerator can only grow, the denominator is fixed),

- $\text{Precision}_r(q)$ is non-increasing (the denominator grows whenever new, possibly irrelevant, chunks are added),

- $F_{1,r}$ may rise or fall, reflecting the trade-off between gaining relevant evidence and accumulating false positives.

We therefore report results as ordered pairs $(F_{1,r}, r)$ for $r = 1, \ldots, R_{\max}$. A method is preferred when it achieves a higher F_1 at the same or a smaller hop count, indicating either better early precision or faster convergence to the full gold evidence set. This metric aligns naturally with our practical objective: maximise answer-supporting recall while minimising the number of expensive retrieval–generation cycles.

3.3 Retrieval Evaluation Results

This section reports detailed retrieval evaluation results obtained from comprehensive experiments conducted on the MultiHop-RAG dataset. Four distinct retrieval methods were systematically evaluated, namely Query Decomposition, Multi-Meta-RAG, IR-CoT, and TreeHop-RAG, using a retrieval-specific metric (Precision, Recall, and F_1 accumulated per hop or decomposition step).

3.3.1 Query Decomposition

The Query Decomposition strategy was tested on a subset of 50 randomly selected test samples, utilizing OpenAI embeddings with 300-token chunks and a 60-token overlap. Each query was decomposed into multiple simpler sub-queries, retrieving the top-5 documents per decomposition. The evaluation outcomes are summarized in Table 1.

Table 1: Retrieval performance for Query Decomposition.

Number of Decompositions	**Precision**	**Recall**	F_1
3	0.242	0.663	0.336
4	0.214	0.660	0.301
5	0.182	0.675	0.265

We observe that as the number of decompositions increases from 3 to 5, Recall marginally improves (0.663 $\rightarrow$ 0.675). However, the substantial drop in Precision (0.242 $\rightarrow$ 0.182) dominates this effect, resulting in progressively declining F_1-scores. Thus, optimal performance is achieved at fewer decomposition steps, highlighting the diminishing returns of further query decompositions.

3.3.2 Multi-Meta-RAG

Using Multi-Meta-RAG with a single iteration, we compared two embedding backbones-`Voyage-2` and `BAAI's BGE-large-en-v1.5`-and observed that BGE-large-en-v1.5 slightly outperformed `Voyage-2` (0.2476 vs. 0.2388) at F1@10 metric. In our setup, for each query we initially retrieve the top 20 candidates, then apply the BAAI/bge-reranker-large model to rerank them, saving the top 10 results. Document chunks are sized to 256 tokens with an overlap of 32 tokens to ensure contextual continuity. These results suggest that, under identical retrieval and reranking conditions, the BGE-large-en-v1.5 embeddings yield a modest but consistent gain in our RAG pipeline.

3.3.3 IR-CoT (Interleaved Retrieval with Chain-of-Thought Reasoning)

The IR-CoT strategy underwent extensive evaluation across three distinct experimental configurations, summarized in Tables 3, 4, and 5.

Configuration 1 Using the `all-MiniLM-L6-v2` embedding, a chunk size of 1000 tokens (200-token overlap), top-5 documents per decomposition, and 20 test samples. The comparative performance of two models (`DeepSeek R1-0528` and `MistralAI/Devstral-small`) is shown in Table 3.

Both models demonstrated identical retrieval performance (F_1 = 0.47), indicating consistent robustness across varying underlying language model architectures, with DeepSeek slightly requiring more iterations on average (1.7 vs. 1.4).

Table 2: IR-CoT results (top-5 documents per decomposition).

Model	Precision	Recall	F_1	Accuracy	Avg. Iterations
`DeepSeek R1 0528`	0.39	0.64	0.47	0.64	1.7
`mistralai/devstral-small`	0.39	0.64	0.47	0.64	1.4

Table 3: IR-CoT results (top-10 documents per decomposition).

Model	Precision	Recall	F_1	Accuracy	Avg. Iterations
`DeepSeek R1 0528`	0.66	0.89	0.76	0.89	1.2

Configuration 2 Increasing retrieval depth to top-10 documents per decomposition (same chunking and embedding strategy), results using `DeepSeek R1 0528` are shown in Table 4.

Increasing the retrieval depth significantly boosted Precision (0.39 $\rightarrow$ 0.66), Recall (0.64 $\rightarrow$ 0.89), and consequently the overall F_1-score (0.47 $\rightarrow$ 0.76), demonstrating the critical role of retrieval depth for IR-CoT's effectiveness.

Configuration 3 Switching to `BAAI/bge-m3` embeddings (other parameters unchanged), the results for `DeepSeek R1 0528` appear in Table 5.

Table 4: IR-CoT results (`BAAI/bge-m3` embeddings, top-10 documents).

Model	Precision	Recall	F_1	Accuracy	Avg. Iterations
`DeepSeek R1 0528`	0.39	0.61249	0.46091	0.61250	1.5

Interestingly, changing embeddings resulted in decreased performance compared to the previous embedding configuration, highlighting embedding model sensitivity in IR-CoT pipelines.

3.3.4 TreeHop-RAG

Finally, the TreeHop-RAG model was evaluated using the default configurations provided by its official implementation, tested directly on the `multihop_rag_dev_processed.jsonl` dataset. No hyperparameter adjustments were made to maintain original reproducibility. Performance across five retrieval hops is reported in Table 6.

Table 5: TreeHop-RAG performance across retrieval hops.

Hop	Precision	Recall	F_1
1	0.3115	0.4860	0.3797
2	0.1836	0.5728	0.2781
3	0.1299	0.6080	0.2141
4	0.0985	0.6149	0.1698
5	0.0793	0.6184	0.1405

The highest F_1-score (0.3797) was achieved at hop 1, indicating rapid retrieval of relevant documents initially. Subsequent hops improved Recall marginally (up to 0.6184 at hop 5), but dramatically decreased Precision (0.3115 $\rightarrow$ 0.0793), thus steadily deteriorating F_1. This behavior highlights that while iterative hops gather more correct evidence, the cost of false positives becomes increasingly significant over additional retrieval rounds.

3.3.5 Summary and Observations

Overall, these comprehensive retrieval evaluations underscore the intrinsic trade-offs among Precision, Recall, and iterative retrieval steps in multi-hop RAG frameworks. While deeper retrieval (e.g., IR-CoT's top-10 retrieval setting) effectively improves Recall and F_1, it risks precision loss unless retrieval mechanisms (like metadata filtering in Multi-Meta-RAG or embedding selection in IR-CoT) are carefully tuned. TreeHop's results emphasize the importance of controlling precision losses across iterative retrieval. The findings clearly indicate that embedding selection, retrieval depth, and decomposition granularity critically influence overall retrieval efficacy in complex multi-hop scenarios.

4 Conclusion

We present the first unified head-to-head evaluation of four representative multi-hop RAG strategies under a common protocol and hop-aware F1 metric. Query Decomposition demonstrates that breaking a complex question into three sequential sub-queries can yield solid early recall (0.663) with an F1 of 0.336, but further decomposition leads to precision erosion (F1 drops to 0.265 at five sub-queries). TreeHop-RAG offers sub-22 ms one-hop retrieval with strong initial precision (F1 = 0.38) but accumulates false positives in deeper hops. Multi-Meta-RAG leverages inexpensive LLM-extracted metadata to deliver high early recall without retriever retraining, though its benefits plateau once filter coverage is exhausted. IR-CoT stands out as the most balanced approach—interleaving chain-of-thought reasoning drives retrieval to achieve near-perfect recall by two hops (0.89) while maintaining high precision (0.66). Together, these findings highlight that (1) explicit query decomposition can effectively guide multi-hop retrieval if granularity is carefully tuned; (2) structured metadata constraints and CoT feedback are powerful for pruning irrelevant evidence; and (3) embedding-space updates can curb latency when LLM calls are costly. Beyond QA, these insights extend to any iterative evidence-gathering task—such as fact-checking or scientific review—paving the way for adaptive hop scheduling, joint retriever-generator training, and graph-based reasoning to further narrow the precision–recall gap in future RAG systems.

References

[1] Cheng-Kuan Ting, Chien-Kuang Wang, Ke-Han Lu, Hung-Yi Lee, and Wen-Lian Hsu. "TreeHop: A Lightweight, Interpretable, and Verifiable Search-in-Embedding Method for Multi-Hop Question Answering." *arXiv preprint arXiv:2504.20114*, 2025. https://arxiv.org/abs/2504.20114.

[2] Xinyu Zhang, Zhaowei Zhang, Ruijun Huang, Xinyi Li, and Dong-Kyu Chae. "Multi-Meta-RAG: A Multi-Meta-Path-based Retrieval-Augmented Generation for Multi-hop Question Answering." *arXiv preprint arXiv:2406.13213*, 2024. https://arxiv.org/abs/2406.13213.

[3] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. "Interleaving Retrieval with Chain-of-Thought Reasoning for Multi-Hop Question Answering." In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3377–3393, Singapore, December 2023. Association for Computational Linguistics. https://aclanthology.org/2023.findings-emnlp.224.

[4] Zhilin Yang, Peng Qi, Saizheng Zhang, et al. "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering." In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2369–2380, Brussels, Belgium, October 2018. Association for Computational Linguistics. https://aclanthology.org/D18-1259.

[5] Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. "A Survey on Multi-hop Question Answering." *ACM Computing Surveys*, 56(3):72:1–72:38, March 2024. https://doi.org/10.1145/3636531.

[6] Ofir Press, Murilo Ribeiro, Adam Roberts, et al. "Self-Ask: Measuring and Improving Language Models' Ability to Decompose Reasoning Steps." In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 12607–12620, Singapore, December 2023. Association for Computational Linguistics. https://aclanthology.org/2023.emnlp-main.782.

[7] Yixuan Tang and Yi Yang. "MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries." *arXiv preprint arXiv:2401.15391*, 2024. https://arxiv.org/abs/2401.15391.

[8] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. "A Survey on Retrieval-Augmented Text Generation." *arXiv preprint arXiv:2202.01110*, 2022. https://arxiv.org/abs/2202.01110.