TreeHop: Efficient Embedding-Level Query Rewriter

Anonymous ACL submission

Abstract

Retrieval-augmented generation (RAG) systems face significant challenges in multi-hop question answering (MHQA), where complex queries require synthesizing information across multiple document chunks. Existing approaches typically rely on iterative LLM-based query rewriting and routing, resulting in high computational costs due to repeated LLM invocations and multi-stage processes. To address these limitations, we propose TreeHop, an embedding-level framework without the need for LLMs in query refinement. TreeHop dynamically updates query embeddings by fusing semantic information from prior queries and retrieved documents, enabling iterative retrieval through embedding-space operations alone. This method replaces the traditional "Retrieve-Rewrite-Vectorize-Retrieve" cycle with a streamlined "Retrieve-Embed-Retrieve" loop, significantly reducing computational overhead. Moreover, a rule-based stop criterion is introduced to further prune redundant retrievals, balancing efficiency and recall rate. Experimental results show that TreeHop rivals advanced RAG methods across three open-domain MHQA datasets, achieving comparable performance with only 5%-0.4% of the model parameter size and reducing the query latency by approximately 99% compared to concurrent approaches. This makes TreeHop a faster and more cost-effective solution for deployment in a range of knowledge-intensive applications. For reproducibility purposes, codes and data are available anonymously here[1].

1 Introduction

Recent breakthroughs in Large Language Models (LLMs) (DeepSeek-AI et al., 2025; OpenAI et al., 2024) have demonstrated impressive abilities in understanding queries (Brown et al., 2020; Ouyang et al., 2022) and generating human-like language texts. Nonetheless, LLMs still face significant limitations, particularly in domain-specific (Li et al., 2024; Zhang et al., 2024) or knowledge-intensive (Kandpal et al., 2023) tasks, where they often hallucinate (Zhang et al., 2023) when dealing with queries that exceed their parametric knowledge (Muhlgay et al., 2024). To address this, Retrieval-Augmented Generation (RAG) (Lewis et al., 2021) has undergone rapid development (Gao et al., 2024), leveraging external knowledge bases to retrieve relevant document chunks and integrate them into LLMs, thereby producing more faithful (Khandelwal et al., 2020) and generalizable (Kamalloo et al., 2023) answers.

However, the conventional single-retrieval paradigm of RAG falters in multi-hop question answering (MHQA) scenarios (Yang et al., 2018; Ho et al., 2020; Tang and Yang, 2024; Trivedi et al., 2022), where answers require synthesizing information from multiple document chunks. For instance, consider the query "*Who is the grandfather of Donald Trump?*" A single retrieval might return a chunk stating "*Donald John Trump was born on June 14, 1946..., the fourth child of Fred Trump and Mary Ann Macleod Trump.*", but resolving the grandfather requires a follow-up query like "*Who is the father of Fred Trump?*". This typical multi-hop scenario reveals the need to dynamically compose new query based on information in relevant document chunk[2]. Current methods like query-rewriters (Ma et al., 2023), routers (Zhuang et al., 2024), and iterative loops (Shao et al., 2023) attempt to resolve this by iteratively refining queries with retrieved information, and drop query-irrelevant chunks. While these approaches improve retrieval, they introduce computational overhead due to repeated LLM invocations and multi-stage processes, leading to latency and complexity trade-offs.

To addresses these limitations, we propose TreeHop, a framework enabling embedding-level query updates without requiring LLM rewrites. Inspired by the semantic and structural properties of sentence embeddings (Zhu et al., 2018), TreeHop dynamically predicts next-step query embeddings by fusing prior queries and retrieved content embeddings (Step 3, Figure 1). For the aforementioned example, the initial information in query "grandfather of Donald Trump" was substituted with "father of Fred Trump", now encoded directly at the embedding level. This approach collapses the traditional "Retrieve-Rewrite-Vectorize-Retrieve" cycle into a streamlined "Retrieve-Embed-Retrieve" loop, significantly reducing computational costs. TreeHop further

[1] https://github.com/Super-Researcher/TreeHop

[2] For more discussion about the need of query rewriters for MHQA task, please refer to Appendix I.

introduces two pruning strategies to ensure computational efficiency: *redundancy pruning* terminates paths where the retrieved chunks have been seen in previous iterations, while *layer-wise top-K pruning* retains only the top-ranked retrieval candidates at each step, curbing exponential branch growth (Step 4, Figure 1).

TreeHop employs a gated cross-attention mechanism (Vaswani et al., 2023) to effectively focus on extracting salient information from retrieved chunks, making the model effective while parameterizing with only 25 million parameters. Trained with contrastive learning (Chen et al., 2020; Wu et al., 2022b), TreeHop is capable of achieving a performance comparable to computationally intensive multi-hop methods across three benchmarks while maintaining significantly lower latency. Remarkably, TreeHop reduces retrieval latency by 99% compared to LLM-based methods while sacrificing only 4.8% of the recall rate at maximum, even outperforming some advanced systems by 4.1% in deeper retrieval iterations.

In summary, our work makes three key contributions to the iterative retrieval framework:

- A novel embedding-updating mechanism that replaces LLM-driven iterative query rewrites with lightweight neural operations, enabling linear computational complexity for MHQA tasks.
- An efficient rule-based stopping criterion that controls branching factor growth while maintaining performance in retrieval iteration.
- Empirical validation demonstrating superior efficiency-accuracy trade-offs in three MHQA tasks. Our approach bridges the gap between computational efficiency and retrieval effectiveness, offering a scalable solution for diverse knowledge-intensive applications.

2 Preliminaries

2.1 Multi-hop Retrieval-Augmented Generation

The Retrieval Augmented Generation (RAG) (Lewis et al., 2021; Gao et al., 2024) fundamentally enhances the capabilities of LLMs by retrieving pertinent documents from an external knowledge base, which is made possible through the calculation of semantic similarity between user's query and document chunks. Building upon RAG, multi-hop variants have been proposed to tackle more complex tasks, such as multi-hop question answering (MHQA). Notable approaches include iterative retrieval methods (Shao et al., 2023), where the knowledge base is repeatedly searched based on the initial query and generated text, providing a more comprehensive information retrieval. Other approaches revolve around employing cooperative language models as query-rewriters (Ma et al., 2023), routers (Manias et al., 2024) or both (Zhuang et al., 2024). These models generate new queries for document chunk retrieval and filter out irrelevant chunks, ensuring the most relevant information is retained. It is worth noting that they all mentioned solutions utilize one or multiple transformer model variants (Vaswani et al., 2023; Reimers and Gurevych, 2019) for enhanced retrieval, which induces additional computational cost and significantly increases system latency.

2.2 Sentence Representation Learning and Contrastive Learning

Sentence representation learning, a technique for training retrieval model in the realm of RAG, refers to the task of encoding sentences into fixed-dimensional embeddings. Early approaches extended word-level techniques like word2vec (Mikolov et al., 2013) to sentences, such as Skip-Though (Kiros et al., 2015) and FastSent citephill-etal-2016-learning, which learned unsupervised sentence embeddings by optimizing sequential or semantic coherence objectives. Subsequent work leveraged pre-trained language models like BERT (Reimers and Gurevych, 2019), extracting sentence embeddings via the [CLS] token or mean pooling of contextualized token representations (Reimers and Gurevych, 2019; Li et al., 2020; Su et al., 2021).

To further improve the performance, contrastive learning emerged as a powerful paradigm for learning discriminative sentence representations (Zhang et al., 2020; Carlsson et al., 2021; Giorgi et al., 2021; Yan et al., 2021; Kim et al., 2021). A cornerstone in this space is SimCSE (Gao et al., 2021), which employs InfoNCE (van den Oord et al., 2019) to maximize agreement between augmented views of the same sentence. The loss function is defined as:

$$\mathcal{L}_{\text{infoNCE}} = -\sum_{i=1}^{N} \log \frac{e^{sim(f(x_i), f(x_i)')/\tau}}{\sum_{j=1}^{N} e^{sim(f(x_i), f(x_j)')/\tau}}, \tag{1}$$

where N is the batch size, τ is a temperature hyperparameter and $sim(f(x_i), f(x_j)') = \frac{f(x_i)\top f(x_j)'}{\|f(x_i)\| \cdot \|f(x_j)'\|}$ is the cosine similarity used in this work. $f(\cdot)$ is the sentence representation encoder, x_i and x_i' are a paired semantically related sentences derived from positive set $\mathcal{D} = \{(x_i, x_i')\}_{i=1}^{m}$.

Additionally, SimCSE applied a dropout as a data augmentation strategy, inspired many following works. Meanwhile, DiffCSE (Chuang et al., 2022) introduces equivariant transformations to ensure invariance to input perturbations, while PCL (Wu et al., 2022a) leverages diverse augmentation strategies to reduce bias in negative sampling. InfoCSE (Wu et al., 2022c) learns sentence representations with the ability to reconstruct the original sentence fragments, RankCSE (Liu et al., 2023) further introduce a listwise ranking objectives for learning effective sentence representations.

3 The Proposed Method: TreeHop

Our proposed model, TreeHop, is designed to generate the next query embedding by integrating previous query

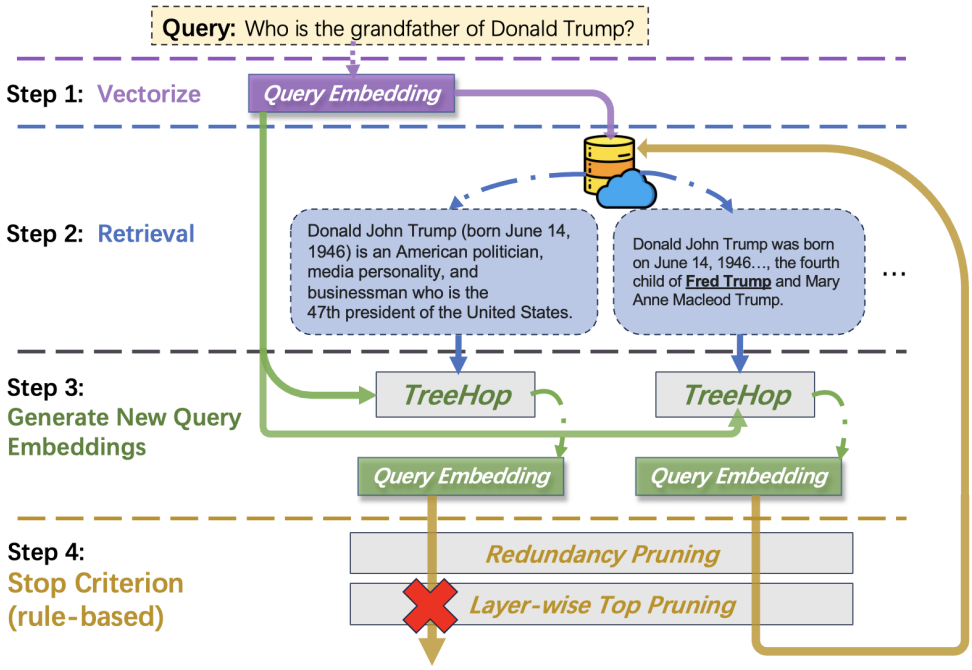

Figure 1: The TreeHop model utilizes query and content chunk embeddings to predict new query embeddings, which are subsequently filtered with similarity and ranking thresholds, thereby streamlines the conventional "Retrieve-Rewrite-Vectorize-Retrieve" into a "Retrieve-Embed-Retrieve" loop.

embeddings and retrieved content embeddings. We have optimized the architecture to achieve high retrieval performance while ensuring a compact parameter size. In the following sections, we first formally define the problem, then detail the model architecture and stopping criterion that contribute to its computational efficiency and effectiveness. Finally, we explain the construction of the train data.

3.1 Problem Formulation

At retrieval iteration step r, given query embedding q, a set of the top K document chunk embeddings, $\mathcal{T} = \{c^i\}_{i=1}^{K}$, is retrieved using the retriever $g(q_r, K)$. The TreeHop model then produces the corresponding query embedding set $\mathcal{Q}_1 = \{q_{r+1}^i\}_{i=1}^{K}$ for the subsequent $r+1$ retrieval step.

$$q_{r+1}^i = \text{TreeHop}(q_r, c_r^i), c_r^i \in \mathcal{T}_r \quad (2)$$

Note, that the TreeHop framework defaults to the base retriever under the single-hop retrieval scenario, as no iterative query refinement is needed. Please refer to Figure 1 for detailed TreeHop inference steps with the stop criterion included in subsection 3.3.

3.2 Model Architecture

TreeHop's architecture is tailored to be effective in performance while maintain a small parameter size. The core of TreeHop's query update is the $UpdateGate$ (see Figure 2), which modulates how information from prior queries q and retrieved chunks c is retained or discarded. The intuition behind is that we only need to remove information presents in both embeddings, and update information yet to be further retrieved from the retrieved chunks to form a new query embedding.

$$\text{TreeHop}(q_r, c_r^i) = q_r - c_r^i + UpdateGate(q_r, c_r^i) \quad (3)$$

The term $q_r - c_r^i$ suppresses semantic overlap between the current query and chunk embeddings. This prevents redundant retrieval of information already captured. (e.g., if the chunk confirms "Fred Trump is Donald's father," the model avoids re-searching for Fred Trump in subsequent hops). The $UpdateGate(q_r, c_r^i)$ selectively incorporates new information from c_r^i to form the next query. We implement cross-attention mechanism (Vaswani et al., 2023) for $UpdateGate$.

$$\begin{aligned} UpdateGate(q, c) &= CrossAttn_u(q, c) \\ &= \text{softmax}\left(\frac{Q_u(q) \odot K_u(c)}{\sqrt{d}}\right) \odot V_u(c) \end{aligned} \quad (4)$$

Where d is the number of embedding dimension, Q_u, K_u and V_u are three weight and bias matrices for $UpdateGate$. Together, information to be maintained in the chunk embeddings is selectively extracted through comparing the information in q_r and c_r^i, and new information is added through $UpdateGate$. This architectural design is based on empirical experiments for improving the model performance (see subsection 5.1 for ablation details). Please also refer to Appendix J for our explanation to the intuition behind the architecture.

3.3 Stopping Criterion

The TreeHop iteratively produces query embedding for every retrieved document chunk, this risks excessive computational costs if every query proceeds to subsequent hops. Unchecked, this approach could lead to

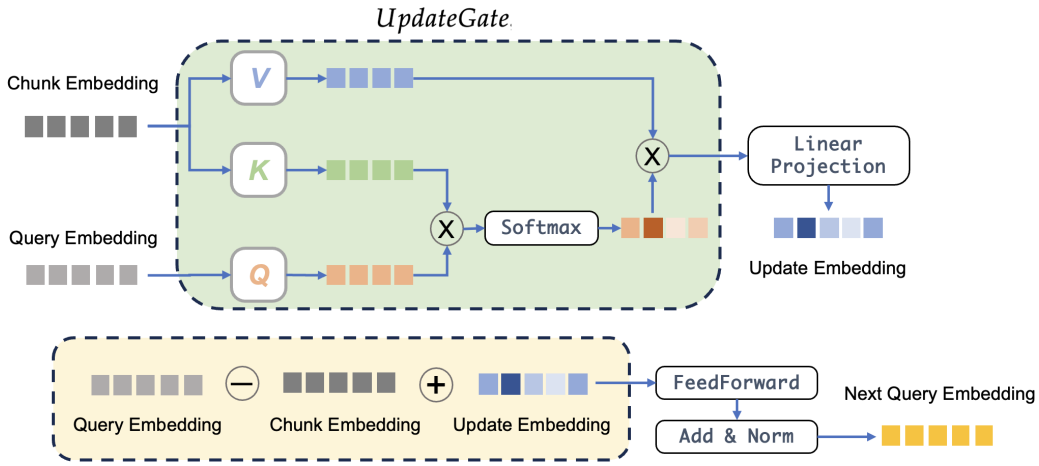

Figure 2: The model architecture of TreeHop. The *UpdateGate*, using cross-attention, updates embeddings via selectively incorporating new information from chunk embeddings. The output is combined with the difference between the previous query and chunk embeddings to form the next query embedding.

an exponential increase in retrieved chunks ($O(n^r)$), degrading efficiency without proportional gains in accuracy. To address this, we introduce a set of rule-based stop criterion that dynamically prunes irrelevant or redundant retrieval branches to ensure only promising paths advance.

Redundancy Pruning We terminate branches where the document chunk has been retrieved in the previous iterations, as depicted in line 8, Figure 3.

Layer-wise Top-K Pruning At each retrieval layer, we retain only the top-K chunks with highest similarity scores across all generated query embeddings. This reduces the branching factor from $O(n^r)$ to $O(nr)$ by focusing computation on the most promising paths, as shown in line 12, Figure 3. Please refer to Appendix L for more explanation about this pruning.

3.4 Train Data Construction

To train the TreeHop model, we require a dataset that explicitly captures the multi-step knowledge retrieval path for MHQA. The 2WikiMultiHop-train dataset (Ho et al., 2020) provides an ideal foundation due to its explicit decomposition of complex questions into intermediate steps, with each step linked to corresponding evidence chunks from Wikipedia. We attempted to construct retrieval paths on Musique/Multihop-RAG train datasets, but found their lack of explicit multi-hop paths. 2WikiMultiHop-train is the only dataset that provides concrete and verifiable multi-hop retrieval paths, a.k.a, question decompositions. Multihop-RAG-train does not come with question decomposition, whatsoever; while MuSiQue-train provides question decompositions, it tends to revise or summarize words from original chunks, impeding us to reliably match the corresponding chunks in scale, ultimately forced us to abandon its training data.

To obtain a fine-grained training dataset, the following processes are implemented to clean the dataset:

Input: Initial user query text x, embedding model $f(\cdot)$ that produces embeddings for input text, total number of hops $N \geq 1$, retriever $g(\cdot)$ that takes one query embedding and outputs top K text chunks $\{c^i\}_{i=1}^K$ and respective embeddings $\{v^i\}_{i=1}^K$, cosine similarity scores $\{s^i\}_{i=1}^K$.

Output: Retrieved text chunk set $\mathcal{C}$.

1: $\mathcal{C} = \emptyset$
$\mathcal{Q} = \{f(x)\}$ // Query embedding set
2: **for** $r \in \{1, \ldots, N\}$ **do**
3: $\mathcal{S} = \emptyset$
4: **for** q in $\mathcal{Q}$ **do**
5: // Retrieve and iterate over top K chunks, embeddings and similarity scores
6: **for** c, v, s in $g(q_i, K)$ **do**
7: **if** $c \notin \mathcal{C}$ **then**
8: // Include only distinct chunks.
$\mathcal{S} \leftarrow [q, c, v, s]$
9: $\mathcal{Q} = \emptyset$
t = **TopSimilarityScore**$(\mathcal{S}, K)$ // Get layer-wise K-th similarity score t from set $\mathcal{S}$.
10: **for** q, c, v, s in $\mathcal{S}$ **do**
11: **if** $s \geq t$ **then**
12: $\mathcal{C} \leftarrow c$
$\mathcal{Q} \leftarrow$ **TreeHop**(q, v)

Figure 3: Multi-hop inference steps and rule-based stop criterion for TreeHop.

Question Type Selection We focus on inference, compositional and bridge comparison questions, as they strictly require the model to synthesize information across multiple hops (e.g., deriving a grandfather's identity by first retrieving a father's name). In contrast, comparison questions rely more on direct factual retrieval without requiring iterative information interaction. See Appendix A for more information about question types.

Query Type Integrity Check We filter the dataset to retain only instances where the provided query decompositions align precisely with the multi-step reasoning required by the question type.

Through these curation processes, we obtained 111,239 samples. For more detailed explanation about the train data construction, please refer to Appendix C.

3.5 Model Training

We utilize BGE-m3 (Chen et al., 2024), a multilingual embedding model that supports more than 100 languages, to generate dense embeddings for the initial query and construct a document chunk embedding database. This gives our trained TreeHop model the potential to be versatile and applicable to a wide range of languages and use cases[3]. Note that BGE-m3 remains frozen during training to ensure the training process focus on the TreeHop model. For detailed prompt templates for generating embeddings on three datasets, please refer to Table 9 and Table 10 in Appendix D.

Following previous work (van den Oord et al., 2019), we adopt contrastive learning framework to train TreeHop to produce embeddings that maximizes the similarities with their corresponding positive chunk embeddings while minimizing similarity with negative ones. Specifically, we employ the $\mathcal{L}_{\text{infoNCE}}$ objective in Equation 1 with temperature τ of 0.15 and five negatives sampled from embedding database. The model is compact enough to be trained on a single Nvidia V100 GPU, with batch size of 64, AdamW optimizer and learning rate of 6e-5. Inspired by SimCSE (Gao et al., 2021), we add a dropout layer after the hidden representations for data augmentation.

4 Experiments and Results

To examine TreeHop, experiments are conducted regarding its retrieval performance and efficiency. Below, we introduce the selective evaluation datasets, evaluate metrics, baselines, concurrent advanced RAG solutions and downstream LLMs that involve in the experiments for comparison.

4.1 Datasets

We benchmark TreeHop on four widely used MHQA datasets in the literature: HotpotQA (Yang et al., 2018), 2WikiMultiHop (Ho et al., 2020), MuSiQue answerable (Trivedi et al., 2022), and MultiHop RAG (Tang and Yang, 2024). Some of their questions do not challenge multi-hop retrieval performance but require LLMs to deduce from multiple documents. Since we only want to test the performance of iterative retrieval systems, we focus on question types requiring multi-step retrieval. Specifically, in 2WikiMultiHop, we filter to inference, compositional and bridge comparison questions (9,536 records), while for HotpotQA and MuSiQue, all answerable questions are used (7,405 and 2,417 records, respectively). MultiHop RAG's inference questions (816 records) are included. See Appendix A for detailed introduction to the types of question among the evaluate datasets, Table 1 for number of queries and size of embedding databases, and Table 8 for detailed number of queries for each selective types of question. To ensure the query type filtering does not affect our contribution, the experimental results, obtained with the same settings described in the following subsections, but without any query type filtering, are presented in Appendix G.

4.2 Evaluation Metrics & Benchmarks

To evaluate retrieval performance, we use the standard evaluation metric, the recall rate, to test the retrieval performance, specifically in the top K retrieval setting, denoted Recall@K. It measures whether the relevant documents are present among the top K retrieved documents. Higher Recall@K values indicate better retrieval performance. To compare the efficiency among selected RAG solutions, we record the average durations for each query in seconds on each dataset, denoted latency. The whole evaluation process is conducted on a single Nvidia A100 GPU and 64 GB of RAM.

End-to-End QA Evaluation Following Efficient RAG and Iter-Retgen, we employ exact match (EM), as well as accuracy (ACC), a metric that evaluates model answers by GPT-3.5, to evaulate end-to-end QA performance among iterative retrieval solutions. The prompt can be found in Appendix E. We adopt llama3-8B-Instruct (AI@Meta, 2024) and Qwen2.5-7B-Instruct (Qwen et al., 2025) as downstream LLMs. For hyper-parameters, we set top-p, top-k, and repetition penalty as 1, max new token as 1024 for the two models, temperature as 0.8 for Llama3.1-8B-Instruct and 0.6 for Qwen2.5-7B-Instruct, respectively. QA prompt adpoted to the two models can be found in Appendix F.

Baselines and Advanced RAG We evaluate the performance of TreeHop by comparing it to a native top retrieval method using the BGE-m3 embedding model as the baseline. In addition, we include two advanced iterative retrieval-augmented generation (RAG) methods: Iter-RetGen (Shao et al., 2023) and EfficientRAG (Zhuang et al., 2024) to assess both performance and latency. For Iter-RetGen, we use the vanilla Meta Llama3-8B-Instruct model (AI@Meta, 2024) for inference. Additionally, we test Iter-RetGen and TreeHop under the second and third retrieval iterations, respectively, to evaluate their performance across different steps of the retrieval process. For the prompt templates used in Iter-RetGen, please refer to Table 11 in Appendix D.

[3]Please find our discussion about TreeHop's multilingual capability in Appendix K.

Dataset	Query	Embedding Database
HotpotQA-dev-distractor	7,405	66,635
2WikiMultihop-dev	9,536	56,709
MuSiQue-dev	2,417	21,100
Multihop-RAG-dev	816	609

Table 1: Descriptive statistics of datasets in terms of the number of queries and sizes of the embedding database.

	HOTPOT			2WIKI			MUSIQUE			MULTIHOP-RAG		
Retriever	**Recall**	**Avg. K**	**Latency**	**Recall**	**Avg. K**	**Latency**	**Recall**	**Avg. K**	**Latency**	**Recall**	**Avg. K**	**Latency**
Baselines												
Direct-R@5	68.3	5	0.002	49.3	5	0.002	45.4	5	0.002	48.6	5	0.019
Direct-R@10	75.2	10	0.003	53.2	10	0.003	53.8	10	0.002	67.8	10	0.019
Advanced RAG												
Iter-RetGen@5 iter2	**73.0**	9.9	4.698	59.2	9.9	4.690	**52.8**	9.9	4.949	55.0	9.9	4.876
Iter-RetGen@5 iter3	73.7	14.9	7.288	61.9	14.7	7.278	**54.1**	14.8	7.274	57.0	14.5	7.322
EfficientRAG@5	71.8	4.4	2.851	60.5	3.8	2.846	46.9	6.1	2.907	51.8	4.1	2.855
Ours												
TreeHop@5 iter2	72.4	8.2	**0.022**	**61.6**	8.6	**0.022**	48.0	8.1	**0.023**	**57.9**	7.0	**0.023**
TreeHop@5 iter3	**73.9**	11.3	**0.064**	**65.4**	11.8	**0.067**	48.1	11.0	**0.064**	**61.1**	8.4	**0.062**
TreeHop@10 iter2	76.4	16.2	0.059	57.9	17.2	0.062	55.7	15.3	0.056	72.8	13.1	0.049

Table 2: We report results of baselines, concurrent advanced RAG solutions and TreeHop on three MHQA datasets. Direct-R@5 denotes the direct retrieval at top-5 baseline retrieval setting (See subsection 4.2 for selected retriever details). **Bold** numbers indicate the best performance in the same iteration among retrievers.

4.3 Results

Below, we present the experimental results of TreeHop and benchmarks on four datasets, including retrieval efficiency, recall, and end-to-end QA performance. For case study on TreeHop's retrieval outcomes, please refer to Appendix N.

Retrieval Efficiency As shown in Table 2, TreeHop significantly reduces computational overhead while maintaining competitive retrieval performance. It achieves latencies of 0.02 seconds per query in the second iteration and 0.06 seconds in the third, outperforming the next best solution, EfficientRAG, by over 2.9 seconds. This significant reduction of 99.2%–99.6% in latency is attributed to TreeHop's embedding-level computation, which avoids the recursive token generation loops required by LLM-based methods. This is confirmed by examining the number of retrieved document chunks, which is proportional to the lantency.

Retrieval Performance TreeHop achieves strong performance across datasets while balancing efficiency and effectiveness. On the HotpotQA dataset, TreeHop's recall falls behind the second best solution, Iter-RetGen, at the second retrieval iteration and surpasses all concurrent solutions at the third; On the 2WikiMultiHop and Multihop dataset, TreeHop surpasses Iter-RetGen by 2.4%-2.9% recall in the second iteration and 3.5%-4.1% recall in the third iteration, with 3.1 less chunks retrieved on average. This demonstrates the effectiveness of the embedding mechanism in Equation 4. For the MuSiQue dataset, recall is 4.8% lower than Iter-RetGen, likely due to the dataset's unique requirement for synthesizing information from multiple chunks (e.g., branching and converging paths in Table 7), which TreeHop's current architecture addresses less effectively than iterative LLM-based approaches. TreeHop's current design focuses on predicting embeddings from query-chunk pairs, limiting its ability to synthesize information across multiple chunks simultaneously. The performance degradation aligns with EfficientRAG solution, which also struggles with this dataset, suggesting a limitation common to query-chunk-pair strategy.

Average Number of K Overall, our TreeHop's average number of retrieved document chunks falls in the middle of the advanced RAG solutions. This is contributed by stop criteria, which drastically curtails computational overhead. For top-5 retrieval, it reduces the theoretical exponential growth of chunks, $5^2 = 25$ chunks in second iteration, to 7.1–8.8 chunks, and $5^3 = 125$ chunks to 8.3–12.1 chunks in the third iteration. For top-10 retrieval, this scales linearly to 13.8–17.9 chunks, versus 10^2 chunks without pruning.

End-to-End QA performance As illustrated in Table 3, downstream LLMs achieve the best end-to-end performance on HotpotQA, 2WikiMultiHop and MultiHop-RAG with TreeHop's retrieval outcomes, while on MusiQue they work the best with Iter-Retgen's retrieval results. This performance is correlated to the performance of upstream retrieval systems, where TreeHop also leads in recall and average number of K on 2WikiMultiHop and MultiHop-RAG, rivals Iter-Retgen on HotpotQA and underperforms Iter-Retgen on MusiQue, as shown in Table 2. Additionally, weaker LLMs tend to be more sensitive to average K, as Llama3.1-8B-Instruct struggles with the lengthy context introduced by high average K at the third iteration, pronouncing the role of stopping criterion as proposed in our framework. More powerful LLMs like Qwen3.1-7B-Instruct benefit from TreeHop on more iterations, as

Downstream LM	HOTPOT				2WIKI				MUSIQUE				MULTIHOP-RAG			
	Qwen2.5		Llama3		Qwen2.5		Llama3		Qwen2.5		Llama3		Qwen2.5		Llama3	
Metric	EM	ACC	EM	ACC	EM	ACC	EM	ACC	EM	ACC	EM	ACC	EM	ACC	EM	ACC
Baselines																
Direct-R@5	34.4	33.0	30.9	29.8	31.3	30.6	27.7	26.7	18.0	16.2	15.4	14.0	87.2	85.3	86.3	84.4
Direct-R@10	34.6	33.3	30.0	29.2	32.8	31.2	27.0	25.9	20.2	18.6	11.8	10.6	90.3	88.6	86.1	85.0
Advanced RAG																
Iter-RetGen@5 iter2	**43.6**	**42.8**	34.2	33.7	38.0	37.6	30.5	30.1	25.4	24.8	17.1	**17.1**	89.8	88.5	86.3	85.3
Iter-RetGen@5 iter3	43.9	43.0	30.9	30.4	39.8	37.9	26.2	25.5	**27.8**	**25.7**	**18.0**	16.6	84.5	82.8	83.9	83.0
EfficientRAG@5	41.8	40.9	34.4	33.7	38.7	38.0	31.2	30.4	18.2	17.5	15.9	14.8	88.7	87.7	86.0	84.4
Ours																
TreeHop@5 iter2	43.3	42.6	**35.0**	**34.4**	38.6	38.1	**31.3**	**30.8**	19.0	18.8	16.6	16.5	90.0	88.8	**87.0**	**86.2**
TreeHop@5 iter3	**44.7**	**44.0**	32.4	31.5	**39.8**	**38.3**	28.9	28.6	19.0	18.9	17.2	16.8	**91.4**	**90.3**	86.4	85.3

Table 3: End-to-End Question Answering (QA) Results. **Bold** numbers indicate the best model answer performance among iterative retrieval frameworks in a same retrieval iteration step.

they are more capable of reasoning with longer context, leading to more consistent performance increases with the retrieval iterations. We also notice the exact match exceeds the accuracy on some datasets, please find the related discuss in Appendix M.

5 Ablation Study

5.1 Effectiveness of Architecture

To evaluate the necessity of each component in Equation (3), we ablated the query term q, chunk term c, and $UpdateGate$ in isolation. Each variant was trained 10 times with identical hyperparameters (as in subsection 3.5), and performance was evaluated on the second hop. Results are shown in Table 4 and analyzed below.

Impact of Component Removal The impact of structure without c is the minimal, with a decrease of 0.9%-6.0% of average recall rates across datasets. The $UpdateGate$ mitigated information loss by selectively retaining critical chunk information, without c, it keeps the information to the extent that still makes the model effective. However, average training convergence time increased by approximately 15% on average, as the model struggled to suppress redundant information without the $q - c$ structure.

Without q, the model loses critical information from query, thereby experiences significant performance degrade on the three datasets, achieves only 0.09%-5.3% improvement of average recall rates comparing to the vanilla top-5 retrieval.

Without $UpdateGate$, recall dropped to near baseline retrieval performance (within 0.1% of random retrieval), confirming the gate's critical role in integrating new information. Without it, the model degenerated to a simple vector difference, failing to refine query embeddings iteratively.

Dataset-Specific Insights The four datasets exhibit different extents of performance decay when the same components are removed. The MuSiQue dataset decays the least without c, q, and $UpdateGate$, this is due to the inherent deficiency of TreeHop on multihop queries with converging paths, making it perform inferior to the other datasets. The Multihop RAG dataset experiences the greatest negative impact without q, due to its complex queries that mention more than three entities for the retrieval model to gather each piece of information. Without q, TreeHop cannot navigate the missing information. The 2WikiMultihop dataset influences less than Multihop RAG without c and q, possibly because of its less challenging queries and query decomposition paths.

5.2 Effectiveness of Stop Criterion

The stop criterion serves for filtering query paths to reduce computational cost without sacrificing too much performance. Below we examine the performance without the presence of Redundancy Pruning and Layer-wise Top Pruning, illustrated in Table 5.

Redundancy Pruning Redundancy pruning prevents revisiting previously retrieved chunks. Table 5 shows that removing this pruning increases the average number of retrieved chunks K to 10, but reduces recall by 4.2 points (e.g., 61.6 $\rightarrow$ 57.4 on 2WikiMultihop). This occurs because when cooperate with layer-wise top pruning, redundancy pruning further ensures only unique, informative paths are pursued, thereby maintaining recall performance. Without redundancy pruning, more duplicated information take the place, resulting in degraded performance.

Layer-wise Top Pruning This pruning strategy selects top-5 chunks at each layer to control branching. Removing this strategy results in a maximum increase of 113% in the average number of retrieved chunks (8.6 $\rightarrow$ 18.4) on the 2WikiMultihop dataset, but yields a 18.6% improvement in recall (61.6% $\rightarrow$ 80.2%). This suggests that the pruning introduces a trade-off between computational efficiency and recall performance. In contrast, for the Multihop RAG dataset, the recall improves by 5.6% with a 79% increase in average retrieved chunks, a significantly lower increase compared to 2WikiMultihop. This disparity among datasets is correlated with the size of the corresponding embedding database. Specifically, in large databases such as 2WikiMultihop, featuring 56,709 chunk embeddings, the model benefits from exploring more paths (higher K) due to the large

Architecture	**HOTPOT** (avg.)	**2WIKI** (avg.)	**MUSIQUE** (avg.)	**MULTIHOP RAG** (avg.)
Direct-R@5	68.3	49.3	45.4	48.6
TreeHop@5 iter2	72.4	61.6	48.0	57.9
w/o. c	70.2 (2.2↓)	57.5 (4.1↓)	47.1 (0.9↓)	51.9 (6.0↓)
w/o. q	69.3 (3.1↓)	54.6 (*6.0*↓)	46.3 (*2.5*↓)	50.8 (*7.1*↓)
w/o. $UpdateGate$	68.3 (*4.1*↓)	49.3 (*12.3*↓)	45.4 (*3.4*↓)	48.6 (*9.3*↓)

Table 4: Ablation study on TreeHop model architecture. The TreeHop experiences degraded performances when core components q, c and $UpdateGate$ are removed from its architecture, demonstrating their functionalities to the model performance.

	HOTPOT (avg.)		**2WIKI** (avg.)		**MUSIQUE** (avg.)		**MULTIHOP RAG** (avg.)	
Stop Criterion	**Recall**	**Avg.** K	**Recall**	**Avg.** K	**Recall**	**Avg.** K	**Recall**	**Avg.** K
TreeHop@5 iter2	72.4	8.2	61.6	8.6	48.0	8.1	57.9	7.0
w/o. Redundancy Pruning	70.1	10	57.4	10	46.4	10	52.8	10
w/o. Layer-wise Top Pruning	75.3	15.2	80.2	18.4	53.6	14.5	61.3	9.7
w/o. Both	75.3	30	80.2	30	53.6	30	61.3	30

Table 5: Ablation study on stop criterion, where redundancy pruning and layer-wise top 5 pruning are removed from post retrieval process, respectively. The results indicate that the recall rate does not exhibit a considerable enhancement, despite a substantial grow in the number of average K.

information pool. Conversely, in smaller databases like Multihop RAG with 609 chunk embeddings, iterative retrieval tends to introduce more redundant chunks, making layer-wise top pruning filter out more useful chunks.

Combined Effects The synergistic effect of combining redundancy pruning and layer-wise top pruning is critical to achieving TreeHop's efficiency gains without excessive recall loss. Take Multihop RAG for example, When both criteria are applied, they achieve a recall of 57.9% with an average number of retrieved chunks 7.0. When both criteria are disabled, the system's computational complexity balloons to 30 chunks in the second iteration, a 329% increase, while yielding only a marginal 3.4% recall improvement. This demonstrates that layer-wise top pruning is essential for limiting branching factor, while redundancy pruning prevents recall degradation from redundant paths. Their combined use ensures that TreeHop avoids the exponential retrieval path explosion inherent to iterative systems while maintaining competitive performance.

6 Conclusion

This work presents a novel paradigm for Retrieval-Augmented Generation (RAG), introducing TreeHop, a lightweight query embedding generator that dynamically refines query embeddings through iterative retrieval without relying on additional LLMs or complex rewrite components, thereby enhancing the efficiency of RAG system. Its core mechanism, the $UpdateGate$, employs cross-attention to selectively integrate information from retrieved chunks while discarding redundant information, enables a compact model size of 25 million parameters while maintaining competitive performance on three MHQA benchmarks when integrated with simple rule-based stop criterion. Future work could explore more effective model architecture, adaptive stop criteria or extensions to handle lengthy, structural, or multi-modal inputs. Our approach underscores the potential of embedding-centric strategies to enhance retrieval process for RAG systems, offering a practical balance between performance and computational efficiency, paving the way for solutions to real-world multi-hop reasoning challenges in industrial applications.

7 Limitation

This study seeks to enhance the efficiency of multihop question answering in the realm of retrieval-augmented generation, a shortcoming that continues to hamper its practical applications. Notwithstanding our method have demonstrated efficacy, it relies on uniform document chunk embeddings generated with a specific prompt template using a specific embedding model. Once trained, the model strictly bonds it usage with the embedding model. Thus, the impact of diverse embedding models and prompt templates remain unclear and requires further investigation. Moreover, our model is trained to retrieve information from open domain documents, its robustness cannot be guaranteed on alternative input structures such as tabular data or other non-textual formats, which raises our concerns about potential misuse.

We note that our TreeHop's query-rewriting mechanism depends on query-chunk pair embeddings, limiting its capability to synthesize information across multiple chunks simultaneously. This design choice prioritizes efficiency and system complexity, as synthesizing arbitrary numbers of chunks requires additional tools or models to repel irrelevant chunks that can potentially contaminate the produced query embeddings.

We also note that our TreeHop has the potential to further improve its performance, please find our analysis on the scale of train dataset in Appendix H.

References

AI@Meta. 2024. Llama 3 model card.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Fredrik Carlsson, Amaru Cuba Gyllensten, Evangelia Gogoulou, Erik Ylipää Hellqvist, and Magnus Sahlgren. 2021. Semantic re-tuning with contrastive tension. In *International Conference on Learning Representations*.

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

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. *Preprint*, arXiv:2002.05709.

Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljacic, Shang-Wen Li, Scott Yih, Yoon Kim, and James Glass. 2022. DiffCSE: Difference-based contrastive learning for sentence embeddings. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4207–4218, Seattle, United States. Association for Computational Linguistics.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

John Giorgi, Osvald Nitski, Bo Wang, and Gary Bader. 2021. DeCLUTR: Deep contrastive learning for unsupervised textual representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 879–895, Online. Association for Computational Linguistics.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Ehsan Kamalloo, Nouha Dziri, Charles L. A. Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. *Preprint*, arXiv:2305.06984.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. *Preprint*, arXiv:2211.08411.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. *Preprint*, arXiv:1911.00172.

Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. 2021. Self-guided contrastive learning for BERT sentence representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2528–2540, Online. Association for Computational Linguistics.

Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Bohan Li, Hao Zhou, Junxian He, Mingxuan Wang, Yiming Yang, and Lei Li. 2020. On the sentence embeddings from pre-trained language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9119–9130, Online. Association for Computational Linguistics.

Zhonghao Li, Xuming Hu, Aiwei Liu, Kening Zheng, Sirui Huang, and Hui Xiong. 2024. Refiner: Restructure retrieval content efficiently to advance question-answering capabilities. *Preprint*, arXiv:2406.11357.

Jiduan Liu, Jiahao Liu, Qifan Wang, Jingang Wang, Wei Wu, Yunsen Xian, Dongyan Zhao, Kai Chen, and Rui Yan. 2023. Rankcse: Unsupervised sentence representations learning via learning to rank. *Preprint*, arXiv:2305.16726.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *Preprint*, arXiv:2305.14283.

Dimitrios Michael Manias, Ali Chouman, and Abdallah Shami. 2024. Semantic routing for enhanced performance of llm-assisted intent-based 5g core network management and orchestration. *Preprint*, arXiv:2404.15869.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Dor Muhlgay, Ori Ram, Inbal Magar, Yoav Levine, Nir Ratner, Yonatan Belinkov, Omri Abend, Kevin Leyton-Brown, Amnon Shashua, and Yoav Shoham. 2024. Generating benchmarks for factuality evaluation of language models. *Preprint*, arXiv:2307.06908.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

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. *Preprint*, arXiv:2305.15294.

Jianlin Su, Jiarun Cao, Weijie Liu, and Yangyiwen Ou. 2021. Whitening sentence representations for better semantics and faster retrieval. *Preprint*, arXiv:2103.15316.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *Preprint*, arXiv:2401.15391.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Svitlana Vakulenko, Shayne Longpre, Zhucheng Tu, and Raviteja Anantha. 2020. Question rewriting for conversational question answering. *Preprint*, arXiv:2004.14652.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation learning with contrastive predictive coding. *Preprint*, arXiv:1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Qiyu Wu, Chongyang Tao, Tao Shen, Can Xu, Xiubo Geng, and Daxin Jiang. 2022a. Pcl: Peer-contrastive learning with diverse augmentations for unsupervised sentence embeddings. *Preprint*, arXiv:2201.12093.

Xing Wu, Chaochen Gao, Zijia Lin, Jizhong Han, Zhongyuan Wang, and Songlin Hu. 2022b. InfoCSE: Information-aggregated contrastive learning of sentence embeddings. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3060–3070, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xing Wu, Chaochen Gao, Zijia Lin, Jizhong Han, Zhongyuan Wang, and Songlin Hu. 2022c. Infocse: Information-aggregated contrastive learning of sentence embeddings. *Preprint*, arXiv:2210.06432.

Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu, and Weiran Xu. 2021. ConSERT: A contrastive framework for self-supervised sentence representation transfer. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5065–5075, Online. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset

for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tianjun Zhang, Shishir G. Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E. Gonzalez. 2024. Raft: Adapting language model to domain specific rag. *Preprint*, arXiv:2403.10131.

Yan Zhang, Ruidan He, Zuozhu Liu, Kwan Hui Lim, and Lidong Bing. 2020. An unsupervised sentence embedding method by mutual information maximization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1601–1610, Online. Association for Computational Linguistics.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. 2023. Siren's song in the ai ocean: A survey on hallucination in large language models. *Preprint*, arXiv:2309.01219.

Xunjie Zhu, Tingfeng Li, and Gerard de Melo. 2018. Exploring semantic properties of sentence embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 632–637, Melbourne, Australia. Association for Computational Linguistics.

Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. Efficientrag: Efficient retriever for multi-hop question answering. *Preprint*, arXiv:2408.04259.

Part I
Appendix

Table of Contents

A Dataset Cards 13

B Details on Evaluate Dataset Question Types 15

C Details on Train Data 16

D Iter-RetGen Prompt Templates 17

E Accuracy Prompt for GPT 3.5 19

F QA Prompt for End-to-End Evaluation 19

G Experiment on All Query Types 20

H Analysis on The Size of Train Dataset 21

I Why do we need Query Rewriters Like TreeHop for Multi-hop Question Answering? 22

J Intuition Behind TreeHop's Architecture 23

K Multilingual Capability of TreeHop 23

L More Explanation About Layer-Wise Top-K Pruning 23

M The Phenomenon of Exact Match Exceeding Accuracy in End-to-End QA Performance 23

N Case Study on TreeHop's Retrieval 24

A Dataset Cards

Below illustrates datasets inclusive in our work, the question types for evaluation are selected to ensure synthesizing information from query and retrieved document chunks are mandatory for multihop retriever.

Dataset	Question Type	Require Synthesize
2WikiMultiHop	**Comparison question**: Questions requiring direct comparison of attributes between entities within the same category. *Example Question*: Who was born first, Albert Einstein or Abraham Lincoln?	
	Inference question: Questions requiring derivation of implicit relationships by combining triples from a knowledge graph. *Example Question*: Who is the maternal grandfather of Abraham Lincoln? *Triples*: (Abraham Lincoln, mother, Nancy Hanks Lincoln); (Nancy Hanks Lincoln, father, James Hanks).	✓
	Compositional question: Questions requiring multi-step relational reasoning across non-explicitly linked triples. *Example Question*: Who founded the distributor of *La La Land*? *Triples*: (La La Land, distributor, Summit Entertainment); (Summit Entertainment, founded by, Bernd Eichinger).	✓
	Bridge-comparison question: Questions requiring both bridging to intermediate entities and comparative reasoning. *Example Question*: Which movie has the director born first, *La La Land* or *Tenet*? *Steps*: 1. Find directors: *La La Land* → Damien Chazelle; *Tenet* → Christopher Nolan. 2. Compare birth years: Damien Chazelle (1985) vs. Christopher Nolan (1970).	✓
MuSiQue	**Unanswerable**: Questions with potential support paragraphs are partially removed, making the reasoning infeasible or unable to arrive at the correct answer.	
	2-Hop Reasoning (Linear Path): A single, straightforward logical path connecting two facts. *Example Question*: Who succeeded the first President of Namibia? *steps*: 1. Identify the first President of Namibia. 2. Determine who succeeded them.	✓
	3-Hop Reasoning (Linear Path) A sequential, three-step logical connection. *Example Question*: What currency is used where Billy Giles died? *steps*: 1. Find the location of Billy Giles' death. 2. Locate the region this place belongs to. 3. Identify the currency used in that region.	✓
	3-Hop Reasoning (Branching-Converging Path) Begins with a single inquiry but diverges into different, branching sub-questions, then converges. *Example Question*: When was the first establishment that McDonaldization is named after, opened in the country Horndean is located? *steps*: 1. Determine what McDonaldization refers to. 2. Identify the country where Horndean is located. 3. Find the date the first establishment opened in that country.	✓

Table 6: Dataset Cards.

Dataset	Question Type	Require Synthesize
MuSiQue	**4-Hop Reasoning (Linear Path)** A continuous, four-step logical progression. *Example Question*: When did Napoleon occupy the city where the mother of the woman who brought Louis XVI style to the court died? *steps*: 1. Identify who introduced Louis XVI style. 2. Find their mother. 3. Determine the city of the mother's death. 4. Discover when Napoleon occupied that city.	✓
	4-Hop Reasoning (Branching-Converging Path) Starts with a single query, splits into multiple paths, and then converges. *Example Question*: How many Germans live in the colonial holding in Aruba's continent that was governed by Prazeres's country? *steps*: 1. Locate Aruba's continent. 2. Identify Prazeres' country. 3. Determine the colonial holding governed by that country in Aruba's continent. 4. Find the number of Germans there.	✓
	4-Hop Reasoning (Converging Path): Multiple distinct lines of reasoning that eventually converge on the answer. *Example Question*: When did the people who captured Malakoff come to the region where Philipsburg is located? *steps*: 1. Determine Philipsburg's location. 2. Identify the terrain feature it belongs to. 3. Find who captured Malakoff. 4. Determine when those people came to that terrain.	✓
MultiHop RAG	**Inference Query**: Questions requiring derivation of implicit relationships by combining triples from a knowledge graph. *Example Question*: Who is the maternal grandfather of Abraham Lincoln? *Triples*: (Abraham Lincoln, mother, Nancy Hanks Lincoln); (Nancy Hanks Lincoln, father, James Hanks).	✓
	Comparison query: Questions requiring direct comparison of attributes between entities within the same category. *Example Question*: Did Netflix or Google report higher revenue for the year 2023?	
	Temporal query: Question that requires an analysis of the temporal information of the retrieved chunks *Example Question*: Did Apple introduce the AirTag tracking device before or after the launch of the 5th generation iPad Pro?	
	Null query: Question whose answer cannot be derived from the retrieved set.This is purposely for testing the issue of hallucination. The LLM should produce a null response instead of hallucinating an answer. *Example Question*: What are the sales of company ABCD as reported in its 2022 and 2023 annual reports?	

Table 7: Dataset Cards.

B Details on Evaluate Dataset Question Types

Below, we provide detailed number of questions for each question type in our evaluate datasets. Please refer to Appendix A for introduction to the types.

Dataset	Question Type	Count
HotpotQA	-	7,405
2WikiMultiHop	Compositional	5,236
	Bridge Comparison	2,751
	Inference	1,549
MuSiQue	2-hop reasoning (linear path)	1,252
	3-Hop Reasoning (Linear Path)	568
	3-Hop Reasoning (Branching Path)	192
	4-Hop Reasoning (Linear Path)	246
	4-Hop Reasoning (Branching Path)	64
	4-Hop Reasoning (Converging Path)	95
Multihop RAG	Inference	816

Table 8: Evaluate data statistics on number of queries and sizes of embedding database.

C Details on Train Data

Due to the space limitations, we refrained from introducing low-level cleaning steps on our train data in the main text; however, this process is non-trivial, here we elaborate more details on the construction of our train data.

To start with, let's consider what data structure do we need for training a model like TreeHop. The TreeHop model fundamentally use contrastive learning to learn next query embedding given a query-chunk embedding pair. We therefore need, for every hop:

- At least one follow-up retrieval step for TreeHop to learn from. The initial step does not count, because the query embedding is produced directly by the base retriever (BGE-m3), and there exists no previous positive chunk embedding.
- A query decomposition that links the current query to its positive chunk. Without this decomposition we cannot build a correct multi-hop graph for training.

If either of these components is missing, the instance cannot be used for training.

In subsection 3.4 we briefly introduced two cleaning stages. Their purpose is exactly to guarantee the two conditions above. Below, we summarize the goals, the removed queries and the remaining data record for each cleaning stages in the following table.

Cleaning Stage	Goal	Instance to Remove	Remaining Records
Initial Train Data			167,454
Query Type Selection	Ensure the presence (need) of a follow-up retrieval step.	Query types that never have a follow-up (e.g., comparison questions).	115,491
Query Type Integrity Check	Verify that the provided decomposition matches the query type.	Instances where the decomposition is inconsistent with the declared hop count or query type (e.g., a "three-hop" label but only one follow-up query is provided).	111,239

These checks are crucial because the decompositions provided by the 2WikiMultihop dataset were generated by LLMs and are not 100% reliable.

D Iter-RetGen Prompt Templates

Below we illustrate prompt templates for generating embedding and Iter-RetGen. Templates for 2WikiMultihop and MuSiQue are identical, while for MultiHop-RAG we add source of content as many of its question decomposition revolve around this. Following previous work (Zhuang et al., 2024), we adopt the same prompt template on three evaluate datasets for Iter-RetGen.

Document Chunk Prompt Template for 2WikiMultihop and MuSiQue
Title: *[doc title]*
Context: *[doc text]*

Table 9: Prompt template for generating embedding using BGE-m3 embedding model on 2Wiki and MuSiQue train and evaluate datasets.

Document Chunk Prompt Template for MultiHop-RAG
Title: *[doc title]*
Source: *[doc source]*
Context: *[doc text]*

Table 10: Prompt template for generating embedding using BGE-m3 embedding model on MultiHop-RAG evaluate dataset.

Iter-RetGen Prompt Template for 2WikiMultihop, MuSiQue and MultiHop-RAG

You should think step by step and answer the question after <Question> based on given knowledge embraced with <doc> and </doc>. Your answer should be after <Answer> in JSON format with key "thought" and "answer", their value should be string.
Here are some examples for you to refer to:
<doc>
{{KNOWLEDGE FOR THE QUESTION}}
</doc>
<Question>: In which year did the publisher of In Cold Blood form?
Let's think step by step.
<Answer>:
``` json
{{ "thought": "In Cold Blood was first published in book form by Random House. Random House was form in 2001.", "answer": "2011" }}
```
<doc>
{{KNOWLEDGE FOR THE QUESTION}}
</doc>
<Question>: Who was in charge of the city where The Killing of a Sacred Deer was filmed?
Let's think step by step.
<Answer>:
``` json
{{ "thought": "The Killing of a Sacred Deer was filmed in Cincinnati. The present Mayor of Cincinnati is John Cranley. Therefore, John Cranley is in charge of the city.", "answer": "John Cranley" }}
```
<doc>
{{KNOWLEDGE FOR THE QUESTION}}
</doc>
<Question>: Where on the Avalon Peninsula is the city that Signal Hill overlooks?
Let's think step by step.
<Answer>:
``` json
{{ "thought": "Signal Hill is a hill which overlooks the city of St. John's. St. John's is located on the eastern tip of the Avalon Peninsula.", "answer": "eastern tip" }}
```
Now based on the given doc, answer the question after <Question>.
<doc>
{documents}
</doc>
<Question>: {question}
Let's think step by step.
<Answer>:

Table 11: Prompt template for Iter-RetGen on 2Wiki, MuSiQue and MultiHop-RAG evaluate datasets.

E Accuracy Prompt for GPT 3.5

Following previous work (Zhuang et al., 2024), we disclose accuracy prompt for evaluating end-to-end QA performance from Table 3.

Accuracy Prompt
You are an experienced linguist who is responsible for evaluating the correctness of the generated responses. You are provided with question, the generated responses and the corresponding ground truth answer. Your task is to compare the generated responses with the ground truth responses and evaluate the correctness of the generated responses. Response in JSON format with key "response" and value "yes" or "no". Question: *[question]* Prediction: *[prediction]* Ground-truth Answer: *[answer]*

Table 12: Accuracy Prompt for Evaluating end-to-end QA performance

F QA Prompt for End-to-End Evaluation

For the downstream LLMs (Llama3.1-8B-Instruct and Qwen2.5-7B-Instruct) in our end-to-end QA performance evaluation, we applied default chat templates, directly attached questions to the user prompt, and adopted the same system prompt for the two LLMs below, where retrieved chunks are naively concatenated in descending order of cosine similarity, then further concatenated in retrieval iteration order.

Answer the question based on the context below. Respond "Unsure about answer" if not sure about the answer. Here is the context: Title: *[title1]* *[content1]* Title: *[title2]* *[content2]* ...

Table 13: Downstream LLM Prompt for Evaluating end-to-end QA performance

G Experiment on All Query Types

In subsection 4.1, we explicitly selected iterative questions for benchmark, i.e., questions that do not require iterative retrieval to gather all necessary information. Some my raise concerns on TreeHop's overall performance when other questions are also considered. Below we present a revision of benchmarks reported in Table 2 of our paper. All experimental settings are identical to those described in subsection 4.2 of the manuscript, except that comparison queries have been added back to the benchmarks. Note, that the results for the HotpotQA and MuSiQue datasets are omitted, as no question types were excluded from them in our original experiment.

	2WIKI			MULTIHOP RAG		
Retriever	**Recall@5**	**Avg.K**	**Latency**	**Recall@5**	**Avg.K**	**Latency**
Baselines						
Direct-R@5	57.5	5	0.003	54.6	5	0.002
Direct-R@10	61.5	10	0.004	71.4	10	0.003
Advanced RAG						
Iter-RetGen@5 iter2	66.4	9.9	4.713	60.3	9.9	4.888
Iter-RetGen@5 iter3	69.8	14.8	7.286	63.5	14.8	7.357
EfficientRAG@5	67.1	3.3	2.802	59.9	3.7	2.859
Ours						
TreeHop@5 iter2	**68.0**	8.8	**0.022**	**61.6**	7.1	**0.023**
TreeHop@5 iter3	**71.6**	12.1	**0.067**	**64.3**	8.7	**0.062**
TreeHop@10 iter2	70.7	17.5	0.062	75.4	13.5	0.049

Table 14: TreeHop performance when all query types are included. **Bold** numbers indicate the best performance in the same iteration among retrievers.

Overall, the existence of comparison queries does not alter the conclusion in subsection 4.3 that TreeHop was both performant and efficient among advanced iterative retrieval approaches. Surprisingly, TreeHop's ability to retrieve complementary information still improves recall for comparison queries, likely because the previous retrieval step sometimes fails to retrieve all necessary information.

H Analysis on The Size of Train Dataset

To explore the possible room of further improving TreeHop by increasing the size of training dataset, we have randomly sampled our current training dataset and trained TreeHop upon 75%, 50% and 25% of the training dataset, respectively. Throughout the experiment, we adopted the environment and hyper-parameters introduced in subsection 3.5.

	2WIKI		**MUSIQUE**		**MULTIHOP RAG**	
Subsamble Size	**Recall@K**	**K**	**Recall@K**	**K**	**Recall@K**	**K**
Direct-R@5	49.3	5	45.4	5	48.6	5
100% of Training Dataset						
TreeHop@5 iter2	61.6	8.6	48.0	8.1	57.9	7.0
TreeHop@5 iter3	65.4	11.8	48.1	11.0	61.1	8.4
75% of Training Dataset						
TreeHop@5 iter2	60.9(0.7↓)	8.7	47.9(0.1↓)	8.2	57.9(0.5↓)	7.0
TreeHop@5 iter3	64.7(0.7↓)	11.9	48.1(0.0↓)	11.1	61.1(0.1↓)	8.5
50% of Training Dataset						
TreeHop@5 iter2	60.1(1.5↓)	8.6	46.8(1.2↓)	8.0	56.0(1.9↓)	6.7
TreeHop@5 iter3	63.3(2.1↓)	11.8	47.2(0.9↓)	10.8	58.8(2.3↓)	7.9
25% of Training Dataset						
TreeHop@5 iter2	57.3(3.8↓)	8.4	46.3(1.7↓)	7.9	55.3(2.6↓)	6.6
TreeHop@5 iter3	60.0(5.3↓)	11.4	46.9(1.2↓)	11.3	58.1(2.9↓)	7.7

Table 15: TreeHop performance trained upon subsampled dataset.

Our experiment reveals that TreeHop's performance decays as the training dataset size decreases. Although the recall improvement is marginal when increasing from 75% to 100% on MuSiQue and MultiHop RAG, we observe a notable improvement on 2WikiMultihop, indicating that there is still room for improvement if we further scale up the training dataset. This suggests that TreeHop has the potential to benefit from larger embedding spaces.

I Why do we need Query Rewriters Like TreeHop for Multi-hop Question Answering?

Though there exists various query rewriters (Zhuang et al., 2024; Shao et al., 2023; Vakulenko et al., 2020; Ma et al., 2023; Vakulenko et al., 2020), readers may be confounded by the significant increase of recall as the top-K increases from 5 to 10 in Table 2, and raise doubt in the necessity of adopting query rewriters for multi-hop question answering (MHQA).

At first glance, one could argue that simply retrieving more documents should be sufficient: a larger candidate set is more likely to contain the facts needed for a multi-step answer, and the iterative refinement that TreeHop performs would appear unnecessary. we argue that query rewriters is still necessary to tasks like MHQA, and there are two pivotal aspects support our belief.

Prompt Length Versus End-to-End Performance After retrieval, the selected passages are concatenated with the original question and fed to a LLM for answer generation. A larger number of retrieved chunks induces extremely lengthy prompt that could easily elicit LLM's lost-in-the-middle syndrome. Therefore, the number of retrieved documents is also critical to the accuracy and efficiency.

The importance of K (the number of retrieved passages) is evident in the end-to-end QA results shown in Table 3. For the weaker Llama-3.1-8B-Instruct model, performance actually **decreases** when we move from Direct-R@5 to Direct-R@10 on 2WikiMultiHop and MuSiQue benchmarks, indicating that the negative impact of longer prompt surpasses the positive impact of increased recall to the model. This observation underscores why an iterative approach such as TreeHop— which keeps the prompt short while progressively increasing recall— is necessary.

Queries that are Intrinsically Unanswerable by a Single Retrieval Consider the question: *"What province shares a border with the province where Lago District is located?"* A naïve, one-shot retrieval with the original query will only return documents that mention Lago District (e.g., its location, demographics, etc.). None of these passages contain the name of the bordering province, because that information resides in a different document that is only reachable after we first learn that Lago District belongs to Niassa Province. A query rewriter reads the initial passage, infer the intermediate entity, and issue a second, more specific query such as: *"Which provinces border Niassa Province?"*. The second retrieval step is now targeted at the missing piece of the reasoning chain, dramatically increasing the probability of finding the final answer.

This two-step reasoning pattern is typical of real-world user queries, where the required facts are scattered across multiple, loosely connected documents. Without an intermediate rewriting stage, the system would either have to retrieve an impractically large K, (thereby inflating the prompt) or fail to retrieve the necessary evidence altogether.

On a higher level, the overarching goal of retrieval system is to reduce the information entropy in the prompt by minimizing irrelevant retrieved documents while still reaching a high recall, while direct retrieval often introduces huge number of irrelevant information with the recall. Query rewriters address this problem by re-formulating the query to target the next missing piece of knowledge, thereby steering the retrieval toward more focused, low-entropy results.

J Intuition Behind TreeHop's Architecture

To clarify the underlying intuition in subsection 3.2, let us revisit the example presented in the introduction: For the question *"Who is the grandfather of Donald Trump?"*, suppose the first retrieval yields a chunk stating *"Donald Trump's father is Fred Trump."* Traditional systems use LLMs to rewrite the query as *"Who is the father of Fred Trump?"* for the next hop. Our TreeHop operates at the embedding level to achieve this rewrite dynamically, thereby achieves efficiency. Under the hood, the model is fed with query-chunk embedding pair, it then replaces *"grandfather of Donald Trump"* information in the query embedding with *"father of Fred Trump"* information in the retrieved chunk embedding. The $q_r - c_r^i$ term in the Equation 3 serves for this replacing purpose by removing overlapping semantics between the query and chunk embeddings. Meanwhile, $UpdateGate$ is adopted to selectively integrate new information from the retrieved chunk to form the next query embedding (Equation 3). The $UpdateGate$ is a cross attention module that selectively maintain only information that is necessary for the next retrieval from chunk embedding (Equation 4), e.g., *"Fred Trump"*.

K Multilingual Capability of TreeHop

Since TreeHop is trained on embeddings produced by BGE-m3 and, during the inference phase, is also fed with query and chunk embeddings outputted by BGE-m3 (rather than raw text), we believe that TreeHop's multilingual capability is inherited from BGE-m3. This is because if two queries written in different languages are semantically close, BGE-m3 will output two similar embeddings, which will be close to each other in terms of cosine distance. Consequently, if the two query embeddings are similar, TreeHop will produce identical next query embeddings given the same chunk embedding. TreeHop works exclusively in the embedding level, it doesn't see the language difference; it only sees the semantic patterns produced by BGE-m3, which are language-agnostic.

L More Explanation About Layer-Wise Top-K Pruning

To begin with, K is a hyper-parameter that controls the number of chunks to be retrieved given a query. However, this could result in an explosion in number of retrieved chunks in multi-hop retrieval. During initial retrieval step, a user query is vectorized and K chunks with the highest similarities to the query are retrieved. Subsequently, TreeHop would generate K embeddings for each of the query-chunk pair. In the next retrieval, we would retrieve top-K chunks again for **every** generated embedding, resulting in a total of K^2 retrieved chunks. Generally, given the number of retrieval iterations r, the chunks grow exponentially as K^r, most of which are irrelevant. To address this overhead, at each retrieval iteration, our Layer-Wise Top-K Pruning retains only the top-K chunks by cosine similarity to the generated query embeddings, pruning the rest. This reduces the branching factor, prevents redundant retrievals and minimizes prompt overload for downstream LLMs.

M The Phenomenon of Exact Match Exceeding Accuracy in End-to-End QA Performance

The exact match metric is a naive evaluation metric that only checks for the presence of the answer string in the text generated by LLMs. Therefore, if a LLM repeats the answer string but arrives at the wrong conclusion or uses flawed reasoning, the GPT-3.5 model may detect these issues and mark the answer as incorrect. This can result in the exact match score exceeding the accuracy score. On the other hand, exact match is also a strict metric, as it may fail to recognize two semantically close sentences written in different ways. Therefore, exact match can either exceed or underperform accuracy.

N Case Study on TreeHop's Retrieval

To enhance readers' understanding on how TreeHop works, below we present a retrieval case from the 2WikiMultiHop benchmark, with the retrieval settings being: two-hop query, top-5 retrieval. The cosine distances are directly obtained from the KNN search procedure presented by the Fiass library (Johnson et al., 2019). For readability we use a hierarchical chunk identifier: "Chunk i.j" means chunk j retrieved in hop i (rank j among the top-5). We highlighted the true-positive chunks that actually contain pertinent information in the annotation column. Other annotations indicate why a chunk is not expanded further by our proposed stopping criterion:

- **Redundancy-pruned**: the chunk has already appeared in the current or a previous hop.
- **Layer-wise Top-K pruned:** the chunk's cosine similarity falls outside the top-5 for that hop.

Query: *"What is the place of birth of the director of the film Beowulf & Grendel?"*

The first retrieval iteration (by direct retrieval framework, BGE-m3)

Rank	Annotation	Retrieved Chunk	Cosine Distance
1	*True Positive*	"Beowulf & Grendel is a 2005 Canadian-Icelandic fantasy-adventure film directed by **Sturla Gunnarsson** . . . "	0.071
2	-	"Wrath of Gods . . . , directed by Jon Gustafsson . . . "	0.073
3	-	"Keith Bearden (born . . .) is an American screenwriter and director . . . "	0.077
4	-	"Wolf Gremm (26 Feb 1942 – 14 Jul 2015) was a German film director . . . "	0.080
5	-	"Werner Herzog (born 5 Sep 1942) is a German film director . . . "	0.081

The second retrieval iteration (by TreeHop framework)

Rank	Annotation	Retrieved Chunk	Cosine Distance
1.1	*True Positive*	"Sturla Gunnarsson (born 30 Aug 1951) is an Icelandic-Canadian film and television director and producer. . . **Gunnarsson was born in Reykjavík in 1951**.""	1.531
1.2	Layer-wise top-5 pruned	"Ágúst Guðmundsson (born 29 Jun 1947) is an Icelandic film director and screenwriter."	1.689
1.3	Layer-wise top-5 pruned	"Ágúst Guðmundsson studied French and Icelandic in Reykjavík..."	1.709
1.4	Redundancy pruned	"Beowulf & Grendel is a 2005 Canadian-Icelandic fantasy-adventure film directed by Sturla Gunnarsson..."	1.716
1.5	Layer-wise top-5 pruned	"Keith Bearden (born in Middletown, Connecticut) is an American screenwriter and director."	1.729
2.1	Redundancy pruned	"Wrath of Gods (2006 documentary) tells the story of the dramatic circumstances Canadian director Sturla Gunnarsson and his crew had to go through during the making of the film 'Beowulf & Grendel'."	1.583
2.2	Redundancy pruned	"Beowulf & Grendel is a 2005 Canadian-Icelandic fantasy-adventure film directed by Sturla Gunnarsson."	1.627

2.3	Redundancy pruned	"Sturla Gunnarsson (born 30 Aug 1951) is an Icelandic-Canadian film and television director and producer… Gunnarsson was born in Reykjavík in 1951."	1.674
2.4	Layer-wise top-5 pruned	"Mikael Håfström (born 1 Jul 1960) is a Swedish film director and screenwriter."	1.729
2.5	Layer-wise top-5 pruned	"Ágúst Guðmundsson studied French and Icelandic in Reykjavík and filmmaking at the National Film School in London."	1.733
3.1	Redundancy pruned	"Keith Bearden (born in Middletown, Connecticut) is an American screenwriter and director."	1.538
3.2	Layer-wise top-5 pruned	"Mike Binder (born 2 Jun 1958) is an award-winning American film director, screenwriter, producer and actor."	1.670
3.3	Layer-wise top-5 pruned	"Edward Bernds (12 Jul 1905 – 20 May 2000) was an American screenwriter and director..."	1.706
3.4	Layer-wise top-5 pruned	"Seth Gordon (born 15 Jul 1976) is an American film director, producer, screenwriter and film editor."	1.773
3.5	Layer-wise top-5 pruned	"David Frankel (born 2 Apr 1959) is an American film director, screenwriter and producer."	1.781
4.1	Redundancy pruned	"Wolf Gremm (26 Feb 1942 – 14 Jul 2015) was a German film director and screenwriter."	1.507
4.2	Layer-wise top-5 pruned	"Franz Josef Gottlieb (1 Nov 1930 – 23 Jul 2006) was an Austrian film director and screenwriter."	1.719
4.3	Layer-wise top-5 pruned	"Rainer Werner Fassbinder (31 May 1945 – 10 Jun 1982), sometimes credited as R.W. Fassbinder, was a West German filmmaker..."	1.730
4.4	Layer-wise top-5 pruned	"Hans Billian (born Hans Joachim Hubert Backe, 15 Apr 1918 in Breslau (now Wrocław, Poland) – 18 Dec 2007 in Gräfelfing, Bavaria) was a German film directo..."	1.730
4.5	Redundancy pruned	"Werner Herzog (born 5 Sep 1942) is a German film director, screenwriter, author, actor and opera director."	1.733
5.1	Redundancy pruned	"Werner Herzog (born 5 Sep 1942) is a German film director, screenwriter, author, actor and opera director."	1.341
5.2	-	"Rainer Werner Fassbinder (31 May 1945 – 10 Jun 1982), sometimes credited as R.W. Fassbinder, was a West German filmmaker..."	1.629
5.3	Redundancy pruned	"Wolf Gremm (26 Feb 1942 – 14 Jul 2015) was a German film director and screenwriter."	1.734
5.4	Layer-wise top-5 pruned	"Georg Tressler (25 Jan 1917 – 6 Jan 2007) was a Vienna-born German film actor and film director."	1.737
5.5	Layer-wise top-5 pruned	"Werner W. Wallroth (28 Feb 1930 – 9 Aug 2011) was a German film director and screenwriter."	1.744

In conclusion, the true-positive chunks (1 and 1.1) are retrieved at ranks 1 and 1.1 during the first and second retrieval iterations, respectively, confirming that TreeHop effectively propagates the query embedding and refines the retrieval in the second hop. At the second iteration, the TreeHop framework retrieves chunks 1.1 and 5.2, while all other chunks are filtered out by the proposed stopping criterion. Notably, although redundant chunks 2.1, 2.2, and 5.1 exhibit lower cosine distances, they are disregarded because layer-wise ranking is performed after removing redundant chunks.