TREEHOP: EFFICIENT EMBEDDING-LEVEL QUERY REWRITER

Anonymous authors
Paper under double-blind review

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 stopping criterion is introduced to further prune redundant retrievals, balancing efficiency and recall rate. Experimental results show that TreeHop rivals advanced RAG methods across four open-domain MHQA datasets, achieving comparable performance with only 2.2%-29.4% of the parameter size of concurrent solutions and reducing the query latency by 92.8%-97.8%. This makes TreeHop a faster and more cost-effective solution for low-resource or latency-sensitive deployment. For reproducibility purposes, codes and data are anonymized here[1].

1 INTRODUCTION

Large Language Models (LLMs) (DeepSeek-AI et al., 2025; Bai et al., 2025) understand queries (Ouyang et al., 2022; Brown et al., 2020) and generate fluent text, yet on domain-specific (Li et al., 2024; Zhang et al., 2024) or knowledge-intensive (Kandpal et al., 2023) tasks they hallucinate (Zhang et al., 2023) once a query exceeds their parametric knowledge (Muhlgay et al., 2024). Retrieval-Augmented Generation (RAG) (Lewis et al., 2021; Gao et al., 2024) addresses this by retrieving relevant chunks from external knowledge bases, 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 & 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]. Query-rewriters (Ma et al., 2023b), routers (Zhuang et al., 2024) and iterative loops (Shao et al., 2023) address this by refining queries with retrieved information, but the repeated LLM invocations impose substantial latency.

To address 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

[1] `https://anonymous.4open.science/r/TreeHop-D11E/`

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

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

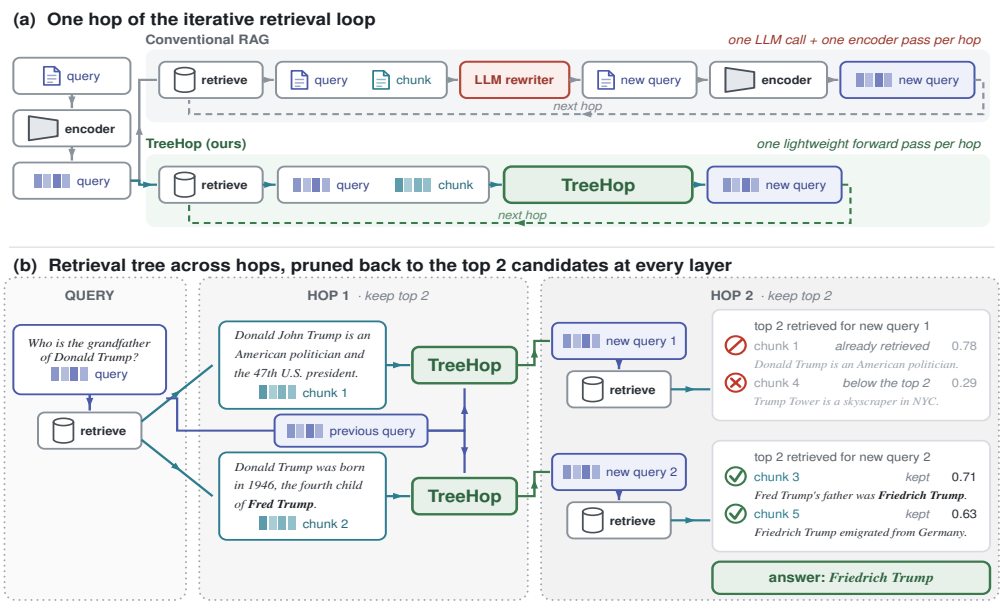


Figure 1: TreeHop replaces LLM-based query rewriting with an update carried out directly on embeddings. **(a)** Conventional iterative RAG rewrites the query with an LLM and re-embeds it at every hop; TreeHop instead fuses the previous query embedding with the retrieved chunk embedding, collapsing "Retrieve-Rewrite-Vectorize-Retrieve" into "Retrieve-Embed-Retrieve". **(b)** The question is encoded once by embedding model. Every retrieved chunk embedding spawns its own next-hop query via TreeHop model; the candidates that all branches retrieve at a layer are ranked together and cut back to K (2 shown in the figure) by *redundancy pruning* and *layer-wise top-K pruning*.

by fusing prior queries and retrieved content embeddings (Figure 1b). This approach collapses the traditional "Retrieve-Rewrite-Vectorize-Retrieve" cycle (Figure 1a) into a streamlined "Retrieve-Embed-Retrieve" loop, significantly reducing computational costs. TreeHop further 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.

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 0.18 billion parameters. Trained with contrastive learning (Chen et al., 2020; Wu et al., 2022), TreeHop is capable of achieving a performance comparable to computationally intensive multi-hop methods across four benchmarks while maintaining significantly lower latency. Remarkably, TreeHop reduces retrieval latency by 92.8%-97.8% compared to LLM-based methods while sacrificing only 0.9% of the recall rate at maximum, even outperforming some advanced systems by 9.1% in deeper retrieval iterations.

In summary, our work makes three key contributions to the class of efficient, iterative retrieval frameworks:

- 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 four 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 RAG have been proposed to tackle more complex tasks, such as multi-hop question answering (MHQA). Iter-RetGen (Shao et al., 2023) feeds the LLM's partial answer back as context for the next round, and query rewriting reformulates the query from retrieved context (Ma et al., 2023a; Vakulenko et al., 2020); both re-generate and re-encode text at every hop. Two recent MHQA systems pursue our objective from the opposite cost regime: Q-DREAM (Ye et al., 2025) also optimises the question semantic space, but through a decomposition model, a dependency optimiser and cluster-wise LoRA tuning over 7B backbones (roughly 4s per query); ChainRAG (Zhu et al., 2025) identifies "lost-in-retrieval", where key entities dropped during sub-question decomposition break the reasoning chain, and repairs it by prompting an LLM against a sentence graph. Each corroborates that recovering the bridge context is the crux of multi-hop retrieval, the premise TreeHop builds on, yet each spends an LLM generation per hop to act on it. RAPTOR (Sarthi et al., 2024) moves that cost off the query path, clustering and summarizing passages into a tree at index time so one retrieval can reach evidence that would otherwise take several hops. A separate, encoder-based works without LLM generating a rewritten query emerged: MDR (Xiong et al., 2021) conditions retrieval on earlier evidence using a single encoder and beam-search mechanism. EfficientRAG (Zhuang et al., 2024) trims this with a trained filter and rewrite encoder. To the best of our knowledge, EfficientRAG is the only stable efficient iterative retrieval model with open-source parameters ; since our goal is efficiency, we treat accuracy-focused methods—such as the model-agnostic Iter-RetGen and the tree-structured RAPTOR—as complementary rather than as baselines.

2.2 SENTENCE REPRESENTATION LEARNING AND CONTRASTIVE LEARNING

Sentence representation learning encodes a sentence into a single fixed-dimensional embedding, and contrastive objectives are the standard way to train it. SimCSE (Gao et al., 2021) employs InfoNCE (van den Oord et al., 2019) to pull two encodings of the same sentence together while pushing apart encodings of different sentences:

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

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

3 THE PROPOSED METHOD: TREEHOP

TreeHop generates the next query embedding by integrating the previous query embedding with retrieved content embeddings, in an architecture optimised for retrieval performance at a compact parameter size.

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 $r+1$ step.

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

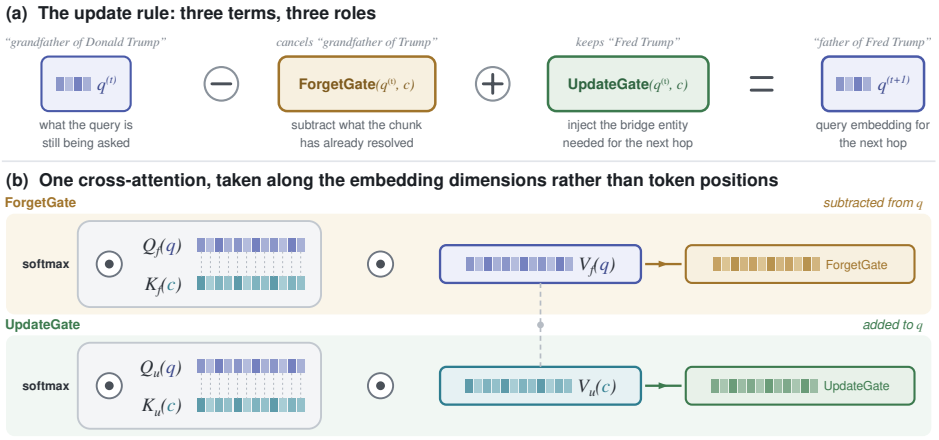


Figure 2: The TreeHop update. **(a)** The next-hop query embedding is assembled from the three terms of Equation 3, glossed on the running example of Figure 1. **(b)** The two gates are the same cross-attention and differ only in the source of their values; subsection 3.2 explains why the attention runs along the embedding dimensions.

Note, that the TreeHop framework defaults to the base retriever under the single-hop retrieval scenario, as no iterative query rewrite is needed. Please refer to Figure 1b for an illustration of TreeHop inference with the stopping criterion described in subsection 3.3.

3.2 Model Architecture

The core of TreeHop's query update is a pair of cross-attention gates, the $ForgetGate$ and the $UpdateGate$ (see Figure 2b), which together modulate how information from prior queries q and retrieved chunks c is discarded or retained. The intuition behind is that we only need to remove information present in both embeddings, and update information yet to be further retrieved from the retrieved chunks to form a new query embedding.

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

The $ForgetGate(q_r, c_r^i)$ term suppresses the semantics that the retrieved chunk has already resolved (Figure 2a). Rather than subtracting the chunk embedding directly, it is designed to attend over the query-chunk pair and remove only those query dimensions the chunk accounts for, which 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. Both gates revise the cross-attention mechanism (Vaswani et al., 2023) to work on embedding level, and are identical in structure except for the source of their values: $ForgetGate$ draws values from the query q, whereas $UpdateGate$ draws them from the chunk c.

$$\begin{aligned} ForgetGate(q, c) &= CrossAttn_f(q, c) \\ &= \text{softmax}(Q_f(q) \odot K_f(c)) \odot V_f(q) \\ UpdateGate(q, c) &= CrossAttn_u(q, c) \\ &= \text{softmax}(Q_u(q) \odot K_u(c)) \odot V_u(c) \end{aligned} \tag{4}$$

Where Q_f, K_f, V_f and Q_u, K_u, V_u are weight and bias matrices learned separately for the two gates. Conventional cross-attention scores one token against another and normalizes the resulting map over sequence positions. That axis is unavailable to us: the frozen encoder emits a single vector per query and per chunk, so no token positions survive for the gates to attend over. We therefore replace the dot products with element-wise products and take the softmax along the d embedding dimensions instead, so that each gate produces one weight per dimension which scales its value vector independently (Figure 2b). Attention thus selects *which coordinates of the embedding* carry the semantics to discard or to inject, which is precisely the granularity a query rewrite needs once the text has been pooled away into a single vector. See subsection 5.1 for ablations and Appendix O for the intuition behind the design. Appendix S discloses the whole model architecture.

3.3 Stopping Criterion

TreeHop produces a query embedding for every retrieved chunk, so unchecked iteration grows the retrieved set exponentially ($O(n^r)$) without proportional gains in accuracy. We therefore introduce rule-based stopping criteria that prune redundant branches so that 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 P for more explanation about this pruning.

3.4 Train Data Construction

Because TreeHop optimizes next-step retrieval via contrastive learning, flat question-answer pairs are insufficient. Training requires step-by-step trajectories structured as Directed Acyclic Graphs (DAGs), whose nodes are evidence paragraphs and edges logical dependencies, exposing the model at each hop to explicit positive targets (evidence containing the bridge entity) alongside hard negatives. To build this robust, structurally supervised dataset, we develop a three-stage pipeline designed to derive a fine-grained train dataset:

- **Multi-Hop Composition:** We construct bottom-up multi-hop queries by linking single-hop questions through shared bridge entities, mapping target paragraphs to reasoning steps using LLM.
- **Topology and Evidence Integrity Validation:** The initial LLM-generated evidence chains can be erroneous, we validate them using another LLM, followed by strict rule-based filtering. Queries with incorrect reason topologies and those with mismatched hop-to-evidence counts are either corrected or discarded.
- **Contrastive Learning Graph:** Validated chains are transformed to DGL (Wang et al., 2019) graphs and traversed layer-by-layer in training. In each hop, current-layer passage embeddings serve as positive targets while corpus-wide passage embeddings form a negative pool. Query embeddings produced by TreeHop will be fed into TreeHop again in the next hop.

For more detailed explanation about the train data construction, please refer to Appendix D.

3.5 Model Training

We use BGE-m3 (Chen et al., 2024), a multilingual embedding model covering over 100 languages, to embed the initial query and build the chunk embedding database. BGE-m3 remains frozen during training to ensure the training process focus on the TreeHop model. Prompt templates for the four datasets are in Table 11 and Table 12 in Appendix F.

We employ the $\mathcal{L}_{\text{infoNCE}}$ objective in Equation 1 with temperature τ of 0.15 and six 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 2e-5.

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 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

Retriever	HOTPOT Recall	Chain	Avg.K	Lat.	2WIKI Recall	Chain	Avg.K	Lat.	MUSIQUE Recall	Chain	Avg.K	Lat.	MULTIHOP-RAG Recall	Chain	Avg.K	Lat.
Baselines																
Direct-R@5	68.3	43.5	5	0.002	49.3	12.7	5	0.002	45.4	16.4	5	0.002	48.6	15.8	5	0.002
Direct-R@10	71.2	50.5	10	0.003	53.2	16.9	10	0.003	49.8	20.2	10	0.002	53.0	19.3	10	0.002
Direct-R@15	73.9	53.3	15	0.003	55.1	19.8	15	0.003	53.3	23.8	15	0.003	56.6	22.2	15	0.003
Advanced RAG																
Iter-RetGen(8B)@5 iter2	73.0	53.4	9.9	4.698	59.2	43.3	9.9	4.690	**52.8**	**28.6**	9.9	4.949	55.0	20.7	9.9	4.876
Iter-RetGen(8B)@5 iter3	73.7	54.8	14.9	7.288	61.9	44.8	14.7	7.278	54.1	**31.6**	14.8	7.274	57.0	25.0	14.5	7.322
Iter-RetGen(0.6B)@5 iter2	70.4	50.5	7.2	2.853	56.0	22.0	6.1	3.004	49.9	19.5	6.6	3.120	56.9	21.0	9.1	2.936
Iter-RetGen(0.6B)@5 iter3	70.7	51.3	10.0	4.511	56.6	23.5	8.1	4.701	50.2	21.6	8.7	4.888	57.2	25.1	12.9	4.674
EfficientRAG@5	71.8	51.8	4.4	2.851	60.5	43.8	3.8	2.846	46.9	19.1	6.1	2.907	51.8	19.0	4.1	2.855
RAPTOR@9 collapse	71.5	48.6	9	5.325	50.9	15.0	9	5.483	49.5	21.2	9	5.331	53.6	20.0	9	6.250
RAPTOR@14 collapse	**75.4**	**55.1**	14	5.546	53.3	17.9	14	5.709	**54.6**	25.7	14	5.536	56.0	20.8	14	6.108
Ours																
TreeHop@5 iter2	**73.6**	**53.8**	9.5	**0.117**	**62.7**	**47.3**	9.5	**0.114**	52.0	25.1	9.2	**0.113**	**60.2**	**23.0**	8.7	**0.108**
TreeHop@5 iter3	74.5	54.9	14.2	**0.202**	**66.8**	**48.1**	14.0	**0.204**	53.7	27.7	13.5	**0.200**	**66.1**	**28.9**	12.2	**0.181**

Table 1: Results of baselines, concurrent advanced RAG solutions and TreeHop on four MHQA datasets. Direct-R@5 denotes the direct retrieval at top-5 baseline retrieval setting (See subsection 4.2 for baseline and advanced RAG details). **Bold** numbers indicate the best performance among advanced retrievers at a comparable Avg.K.

MultiHop RAG (Tang & 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 7 for number of queries and size of embedding databases, and Table 8 for detailed number of queries for each selective types of question. To evaluate TreeHop's generalizability to questions that do not require multi-hop retrieval, the experimental results, Appendix I presents results obtained using the settings described in the following subsections, but without query type filtering.

4.2 Evaluation Metrics & Benchmarks

To evaluate retrieval performance, we report the standard recall rate under a top-K retrieval setting, denoted Recall@K. Since Recall@K is micro-averaged over gold passages, it gives partial credit to a broken chain that a downstream reader cannot use. We therefore also report **complete evidence-chain recall** (Chain), the fraction of queries whose gold passages are *all* retrieved by the end of an iteration, reported cumulatively with retrieval hops. Table 7 lists the average gold passages per query that this criterion demands.

To compare the efficiency among selected RAG solutions, we record the average duration per query in seconds on each dataset, denoted latency, which measures retrieval only.

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-5-mini, to evaulate end-to-end QA performance among iterative retrieval solutions. The prompt can be found in Appendix G. We adopt Llama3.1-8B-Instruct (Grattafiori et al., 2024), Qwen3.6-27B (Qwen Team, 2026) with reasoning capability, and Qwen2.5-7B-Instruct (Qwen et al., 2025) as downstream LLMs. Hyper-parameters and the QA prompt adopted for the three models are in Appendix H.

Baselines and Advanced RAG Our baseline is native top-K retrieval. All methods use BGE-m3 as the retrieval backbone for fairness. To isolate the benefit of structural optimization, we confine the comparison to TreeHop's own paradigm, efficient iterative retrieval, benchmarking against Iter-RetGen (Shao et al., 2023) and EfficientRAG (Zhuang et al., 2024) with Filter (0.3B) and Labeler (0.4B) models. We refrain from comparing against heavy LLM frameworks, whose parametric reasoning capacity would confound the comparison. We additionally compare against RAPTOR (Sarthi et al., 2024), which is similar in the tree structured reasoning graph. We report its collapse mode that performs better; Appendix J gives full mode details.

We instantiate Iter-RetGen at two backbone scales, Llama3.1-8B-Instruct (AI@Meta, 2024) and the compact Qwen3-0.6B (Yang et al., 2025), and evaluate both Iter-RetGen and TreeHop at their

Method	Downstream LM	HOTPOT EM	HOTPOT ACC	2WIKI EM	2WIKI ACC	MUSIQUE EM	MUSIQUE ACC	MULTIHOP-RAG EM	MULTIHOP-RAG ACC
Iter-RetGen(8B)@5 iter2	Qwen 3.6	62.4	66.0	52.1	54.8	**46.4**	**50.0**	98.7	99.0
	Qwen 2.5	43.6	46.5	38.0	40.6	**25.4**	**27.0**	89.8	**91.1**
	Llama 3.1	**34.2**	**37.0**	30.5	32.3	**19.1**	19.9	86.3	87.8
Iter-RetGen(8B)@5 iter3	Qwen 3.6	63.0	66.8	53.3	56.1	**49.3**	**52.3**	98.9	99.1
	Qwen 2.5	43.9	46.7	39.8	41.7	**27.8**	**29.7**	84.5	86.5
	Llama 3.1	30.9	33.7	26.2	28.5	**18.0**	**21.0**	83.9	86.4
Iter-RetGen(0.6B)@5 iter2	Qwen 3.6	58.2	61.3	48.5	51.3	23.5	26.2	98.8	99.0
	Qwen 2.5	42.2	44.4	34.4	37.3	20.5	23.1	89.9	90.6
	Llama 3.1	33.0	34.8	27.4	30.1	14.3	17.0	86.3	87.4
Iter-RetGen(0.6B)@5 iter3	Qwen 3.6	58.2	61.6	49.2	52.1	24.5	27.7	98.8	99.0
	Qwen 2.5	43.0	45.3	35.3	37.9	21.2	23.7	85.6	87.6
	Llama 3.1	28.6	31.4	23.0	25.4	14.0	16.4	84.3	86.7
EfficientRAG@5	Qwen 3.6	59.8	62.1	52.7	55.3	21.2	22.6	96.8	97.4
	Qwen 2.5	42.8	44.6	38.7	41.0	18.2	19.5	88.7	89.5
	Llama 3.1	33.4	35.7	31.2	33.4	15.9	17.8	86.0	86.9
RAPTOR@9 collapse	Qwen 3.6	57.9	60.6	47.7	49.7	35.2	38.7	97.3	98.2
	Qwen 2.5	40.4	42.5	32.6	34.8	21.0	23.4	89.1	90.4
	Llama 3.1	33.4	35.0	21.4	23.5	18.8	**20.9**	**87.5**	87.9
RAPTOR@14 collapse	Qwen 3.6	**65.1**	**68.2**	48.1	50.6	41.3	44.6	98.7	99.0
	Qwen 2.5	**45.3**	**48.1**	33.9	35.3	23.1	25.3	89.7	90.8
	Llama 3.1	30.3	34.0	20.3	23.2	17.6	19.4	**86.5**	**87.7**
TreeHop@5 iter2	Qwen 3.6	**62.7**	**66.5**	**56.3**	**58.9**	43.8	46.5	**99.0**	**99.2**
	Qwen 2.5	**43.8**	**46.7**	**41.6**	**43.1**	24.0	26.4	**90.0**	90.8
	Llama 3.1	34.0	36.2	**32.3**	**34.5**	18.8	19.8	87.0	**88.2**
TreeHop@5 iter3	Qwen 3.6	64.4	67.8	**61.9**	**64.1**	46.9	50.8	**99.0**	**99.3**
	Qwen 2.5	44.7	47.7	**45.8**	**48.3**	26.0	28.9	**91.4**	**92.1**
	Llama 3.1	32.4	34.5	**32.2**	**35.1**	17.2	18.8	86.4	87.3

Table 2: End-to-End Question Answering (QA) Results. **Bold** numbers indicate the best model answer performance among advanced iterative retrieval frameworks in a same iteration step.

second and third retrieval iterations to expose behaviour at different depths. Prompt templates for Iter-RetGen are in Table 13.

4.3 RESULTS

We report results of TreeHop and benchmarks on four datasets; Appendix Q contains a case study on the retrieval outcomes.

Retrieval Efficiency As shown in Table 1, TreeHop reduces computational overhead by more than an order of magnitude while remaining competitive on recall, at 0.1 seconds per query in the second iteration and 0.2 seconds in the third, over 2.6 seconds faster than the next best solution, EfficientRAG. This is a latency reduction of 95.9%-96.2% at the second iteration and 92.8%-93.7% at the third, of 97.2%-97.8% against Iter-RetGen(8B), and of 96.4%-98.3% against RAPTOR. It stems from TreeHop's embedding-level computation, which avoids the recursive token generation loops of LLM-based methods.

Retrieval Performance TreeHop achieves performance comparable to LLM-based frameworks across datasets while balancing efficiency and effectiveness. On the HotpotQA dataset, TreeHop's recall leads all iterative solutions at the second retrieval iteration and trails only RAPTOR at the larger budget of the third; On the 2WikiMultiHop and Multihop dataset, TreeHop surpasses Iter-RetGen backboned by Llama3.1-8B-Instruct by 3.5%-5.2% recall in the second iteration and 4.9%-9.1% recall in the third iteration, with slightly less chunks retrieved on average. This demonstrates the effectiveness of the embedding mechanism in Equation 4.

Complete Evidence-Chain Recall Direct retrieval closes the full evidence chain for only 12.7%-43.5% of queries, far below its 45.4%-68.3% micro recall. Iteration recovers that gap: by the third

Architecture	**HOTPOT** (avg.)	**2WIKI** (avg.)	**MUSIQUE** (avg.)	**MULTIHOP RAG** (avg.)
Direct-R@5	68.3	49.3	45.4	48.6
TreeHop@5 iter2	73.6	62.7	52.0	60.2
w/o. c	70.3 (3.3↓)	58.6 (4.1↓)	48.1 (3.8↓)	55.6 (6.5↓)
w/o. q	69.2 (4.4↓)	55.7 (7.7↓)	46.6 (6.4↓)	51.4 (10.7↓)
w/o. $UpdateGate$	68.3 (5.3↓)	49.3 (13.4↓)	45.4 (6.5↓)	48.6 (13.5↓)
Single-gate fusion	69.7 (3.9↓)	56.5 (6.2↓)	48.3 (3.6↓)	53.6 (8.5↓)
Residual MLP	70.4 (3.2↓)	56.9 (5.8↓)	48.5 (3.4↓)	54.1 (8.0↓)

Table 3: Ablation study on TreeHop model architecture. TreeHop experiences degraded performance when the core components q, c, or $UpdateGate$ are removed, and two capacity-matched alternatives also fall behind TreeHop despite matched parameter counts, demonstrating that the architectural design rather than parameter count drives the performance.

iteration TreeHop raises Chain by 11.3%-35.4% over direct retrieval, exceeding the corresponding micro recall gain on three of four datasets, and leads every concurrent solution on 2WikiMultiHop and MultiHop-RAG. Iter-RetGen(8B) retains its MuSiQue advantage (31.6% against 27.7%) for the synthesis reason above, and RAPTOR edges ahead on HotpotQA (55.1% against 54.9%) at a comparable Avg.K. Iterating also buys more chains than simply widening the retrieval window. At a comparable Avg.K, TreeHop@5 iter3 gains 1.6%-28.3% Chain over Direct-R@15 on all four datasets, yet its recall gain there is only 0.6% on HotpotQA and 0.4% on MuSiQue: retrieving more documents recovers isolated gold passages about as well as iterating, but it is completing the chain that iterative rewriting contributes.

Average Number of K The average number of retrieved chunks grows linearly with retrieval step, a predictability that serves as a practical deployment guideline. Our stopping criteria (see subsection 3.3) achieve this by reducing the theoretical exponential growth under top-5 retrieval from $5^2 = 25$ chunks to 8.7–9.5 in the second iteration and from $5^3 = 125$ to 12.2–14.2 in the third.

End-to-End QA performance As illustrated in Table 2, downstream LMs achieve the best end-to-end performance on 2WikiMultiHop and MultiHop-RAG with TreeHop's retrieval outcomes, while on MusiQue they work the best with Iter-Retgen's retrieval results and on HotpotQA with RAPTOR's at $K = 14$. This is correlated to the performance of upstream retrieval systems, where TreeHop also leads in chain on 2WikiMultiHop and MultiHop-RAG, rivals Iter-Retgen on HotpotQA and underperforms Iter-Retgen on MusiQue, as shown in Table 1. What the third iteration is worth to a reader depends on how much context it can absorb. Llama3.1-8B-Instruct, whose context window is the shortest of the three, loses exact match on all datasets as average K grows from the second iteration to the third, whereas both Qwen models improve on every one of them. The stopping criterion is what makes this manageable: by holding average K to 12.2–14.2 rather than the 5^3 of unpruned expansion, it keeps the retrieved context within reach of a short-context LM.

5 Ablation Study

5.1 Effectiveness of Architecture

To evaluate the necessity of each component in Equation equation 3, we ablate the query term q, chunk term c, and $UpdateGate$ in isolation; To further test whether TreeHop's structured update is preferable to a simpler transition of equal capacity, we additionally compare against two capacity-matched alternatives: a ***single-gate fusion*** that folds the update into one $CrossAttn$ module ($q + CrossAttn(q, c)$, depth-doubled to TreeHop's 176M parameters), and a ***residual MLP*** ($q+\mathrm{MLP}([q; c])$, 195M) that discards the attention and gating structure altogether. Each component-removal variant is trained 10 times with identical hyperparameters (as in subsection 3.5), the alternatives are trained under the same setup, best performant checkpoint is chosen and all variants are evaluated on the second hop. Results are shown in Table 3 and analyzed below.

Impact of Component Removal Removing $ForgetGate$ has the least impact, costing 3.3%-4.6% average recall: $UpdateGate$ still retains the critical chunk information. Training convergence time nonetheless rose by roughly 15%, as the model struggled to suppress redundant information without

	HOTPOT		2WIKI		MUSIQUE		MULTIHOP RAG	
Stopping Criterion	**Recall**	**Avg.K**	**Recall**	**Avg.K**	**Recall**	**Avg.K**	**Recall**	**Avg.K**
TreeHop@5 iter2	73.6	9.5	62.7	9.5	52.0	9.2	60.2	8.7
w/o. Redundancy Pruning	70.4	10	59.9	10	47.2	10	59.7	10
w/o. Layer-wise Top Pruning	79.1	17.0	77.2	18.4	55.6	16.6	69.3	14.2
w/o. Both	79.1	30	77.2	30	55.6	30	69.4	30

Table 4: Ablation study on stopping criterion, where redundancy pruning and layer-wise top 5 pruning are removed from post retrieval process, respectively.

the subtractive branch. Without q, the model loses critical information from query, thereby experiences significant performance degrade on all the datasets, improves only 0.9%-6.4% over the vanilla top-5 retrieval. Without $UpdateGate$, recall dropped back to the direct retrieval baseline, matching Direct-R@5 on all four datasets, confirming the gate's critical role in integrating new information. Without it, the model degenerated to a gated subtraction that can only remove information from the query, failing to refine query embeddings iteratively.

Capacity-Matched Alternatives Component removal shows each term is necessary, but not that TreeHop's structured update is preferable to a simpler transition of equal capacity. Both alternatives (Table 3) trail TreeHop@5 iter2 on all four datasets despite matched parameter counts, with the largest gaps on 2WikiMultiHop and MultiHop-RAG. The gain therefore stems from the specific dual-gate parameterization rather than from capacity or a generically expressive learned map.

5.2 Effectiveness of Stopping Criterion

Below we examine the performance without the presence of Redundancy Pruning and Layer-wise Top Pruning, as shown in Table 4.

Removing redundancy pruning, which prevents revisiting retrieved chunks, raises the average retrieved chunks K to 10 while lowering recall by 0.5–4.8%; removing layer-wise top-K pruning, which caps branching at the top-5 chunks per layer, conversely improves recall by 3.6–14.5% but inflates K by 63–93%, an efficiency–recall trade-off.

Combined, the two prunings are complementary: on MultiHop-RAG, applying both yields 62.1% recall at K=8.7, whereas disabling both balloons K to 30 (a 349% increase) for only a 7.2% recall gain. Layer-wise pruning curbs the exponential path growth of iterative retrieval while redundancy pruning prevents recall loss from duplicated paths, keeping TreeHop efficient without sacrificing competitiveness.

6 Conclusion

We introduce TreeHop, a lightweight query embedding model that refines query embeddings across retrieval iterations without invoking an additional LLM or a rewrite component. Its core mechanism is a pair of embedding-level cross-attention gates: the $UpdateGate$ selectively integrates information from retrieved chunks, and the $ForgetGate$ discards the query semantics those chunks have already resolved. Paired with a simple rule-based stopping criterion, this keeps the model at 0.18 billion parameters while remaining competitive on four MHQA benchmarks. Future work could explore more effective model architectures, adaptive stopping criteria, or extensions to lengthy, structural, or multi-modal inputs. Embedding-centric rewriting offers a practical balance between retrieval performance and computational cost, paving the way for real-world multi-hop reasoning in industrial applications.

7 AI USE STATEMENT

The research content of this paper was conceived, designed, and written by the authors. An AI assistant was used only to polish author-drafted material for language and clarity; it did not contribute research ideas, experiments, analyses, or results. Specifically, Figure 1 and Figure 2 were created by the authors and refined with AI assistance, and the experiment and the discussion of MDR's exclusion from our benchmark (Appendix L) together with Appendix S were drafted by the authors and subsequently polished with AI assistance. The authors reviewed all such content and take full responsibility for it.

8 REPRODUCIBILITY STATEMENT

We release our code and model checkpoint anonymously at `https://anonymous.4open.science/r/TreeHop-D11E`. The exact architecture as implemented, the released checkpoint's configuration, and its full parameter breakdown are documented in Appendix S, reconciling the simplified description in subsection 3.2. The training objective, optimizer, and hyperparameters are given in subsection 3.5, the training-data construction in Appendix D, and the rule-based stopping criterion in subsection 3.3. All retrieval and latency measurements are reported together with the hardware and software environment in subsection 4.2; latency is the average per-query wall-clock time on that single machine. Owing to the file-size limit of the anonymous hosting service, the training and benchmark datasets are not bundled with the anonymized repository and will be released publicly upon acceptance.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL `https://arxiv.org/abs/2511.21631`.

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, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

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

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020. URL `https://arxiv.org/abs/2002.05709`.

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, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao,

Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Hady Elsahar, Pavlos Vougiouklis, Arslen Remaci, Christophe Gravier, Jonathon Hare, Frederique Laforest, and Elena Simperl. T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6894–6910, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.552. URL https://aclanthology.org/2021.emnlp-main.552/.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren

Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin

Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. URL https://www.aclweb.org/anthology/2020.coling-main.580.

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

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

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

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

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

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. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

Zhonghao Li, Xuming Hu, Aiwei Liu, Kening Zheng, Sirui Huang, and Hui Xiong. Refiner: Restructure retrieval content efficiently to advance question-answering capabilities, 2024. URL https://arxiv.org/abs/2406.11357.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023a.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023b. URL https://arxiv.org/abs/2305.14283.

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

Evren Ayberk Munis, Deniz Yılmaz, Arianna Muti, and Çağrı Toraman. Fiber: A multilingual evaluation resource for factual inference bias, 2025. URL https://arxiv.org/abs/2512.11110.

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. Training language models to follow instructions with human feedback, 2022.

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, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016. URL https://arxiv.org/abs/1606.05250.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy, 2023. URL https://arxiv.org/abs/2305.15294.

Yixuan Tang and Yi Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries, 2024. URL https://arxiv.org/abs/2401.15391.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Svitlana Vakulenko, Shayne Longpre, Zhucheng Tu, and Raviteja Anantha. Question rewriting for conversational question answering, 2020.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019. URL https://arxiv.org/abs/1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Xing Wu, Chaochen Gao, Zijia Lin, Jizhong Han, Zhongyuan Wang, and Songlin Hu. InfoCSE: Information-aggregated contrastive learning of sentence embeddings. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 3060–3070, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.223. URL https://aclanthology.org/2022.findings-emnlp.223/.

Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen tau Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oğuz. Answering complex open-domain questions with multi-hop dense retrieval, 2021. URL https://arxiv.org/abs/2009.12756.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Linhao Ye, Lang Yu, Zhikai Lei, Qin Chen, Jie Zhou, and Liang He. Optimizing question semantic space for dynamic retrieval-augmented multi-hop question answering. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17814–17824, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.871. URL https://aclanthology.org/2025.acl-long.871/.

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

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. Siren's song in the ai ocean: A survey on hallucination in large language models, 2023.

Rongzhi Zhu, Xiangyu Liu, Zequn Sun, Yiwei Wang, and Wei Hu. Mitigating lost-in-retrieval problems in retrieval augmented multi-hop question answering. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22362–22375, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1089. URL https://aclanthology.org/2025.acl-long.1089/.

Xunjie Zhu, Tingfeng Li, and Gerard de Melo. Exploring semantic properties of sentence embeddings. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 632–637, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2100. URL https://aclanthology.org/P18-2100/.

Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Efficientrag: Efficient retriever for multi-hop question answering, 2024. URL https://arxiv.org/abs/2408.04259.

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.	
	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 5: Dataset Cards.

Dataset	Question Type	Require Synthesize
MuSiQue	**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.	✓
	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 6: Dataset Cards.

B Details on Evaluate Datasets

We illustrate number of records in all inclusive datasets in the following table.

Dataset	Query	Embedding Database	Avg. Gold Passages
HotpotQA-dev-distractor	7,405	66,635	2.00
2WikiMultihop-dev	9,536	56,709	2.58
MuSiQue-dev	2,417	21,100	2.60
Multihop-RAG-dev	816	609	2.47

Table 7: Descriptive statistics of datasets in terms of the number of queries, sizes of the embedding database, and the average number of gold supporting passages per query, which is the number of passages a retriever must recover to close the evidence chain.

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

All retrieval and latency measurements are conducted on a single NVIDIA A100-SXM4-40 GB GPU in a dual-socket AMD EPYC 7763 (128 threads) server with 100 GB of RAM, running Red Hat Enterprise Linux 8.10 (kernel 4.18) with NVIDIA driver 595.71.05 (CUDA 13.2) and the CUDA 12.4 toolkit. Only one exception: Qwen3.6-27B as downstream LM for end-to-end QA experiment, every environment settings are the same except that there are two NVIDIA A100-SXM4-40 GB GPU.

D Details on Train Data

Due to the space limitations, we refrained from introducing low-level construction steps on our train data in the main text; however, this process is non-trivial. The TreeHop model fundamentally uses contrastive learning to learn next query embedding given a query-chunk embedding pair. Therefore, training TreeHop requires embedding level dataset structured using multi-hop reasoning steps, with each step provides the model with positive-negative passage pairs, where positive passages contain evidence steer toward ground truths. This results in directed acyclic graphs (DAGs) that mirrors the reasoning path. Below, we describes the sources of initial questions, the derivation of directed evidence chains, the pipeline for turning raw data into DGL training graphs, the augmentation strategy for comparison queries, and cleaning stages.

D.1 From Single-hop Question to Multi-hop Question

We started with collecting single-hop questions from four English reading comprehension datasets: SQuAD (Rajpurkar et al., 2016), Natural Questions (Kwiatkowski et al., 2019), MLQA (en-en) (Munis et al., 2025) and T-REx (Elsahar et al., 2018). Multi-hop questions can be viewed as a series of single-hop sub-questions and are constructed bottom-up from a sequence of corresponding reasoning steps, where answer from one reasoning step was used to identify the next reasoning step. Following (Trivedi et al., 2022), we adopt the same methodology to derive multi-hop questions, with each multihop question Q has an underlying DAG G_Q whose nodes are single-hop sub-questions $q_1, q_2, \ldots, q_n$ and where a directed edge (q_j, q_i) indicates that q_i depends on the answer of q_j via a shared bridge entity (Trivedi et al., 2022). For example, the multi-hop question *"Who succeeded the first President of Namibia?"*, is composited by two single-hop sub-questions: *"Who was the first President of Namibia?"* and "Who succeeded ..." where the answer of the first question is the bridge entity, we annotate paragraphs in which the bridge entities reside as the supporting paragraphs.

D.2 Evidence Chain Validation

After generating questions and their supporting paragraphs, we employ LLM-based inference to validate the correct reasoning order among the supporting paragraphs. This is crucial because they were generated by LLMs and are not 100% reliable. Concretely, for each sample we extract the graph type from its sample ID, look up the expected reasoning topology, and construct a structured prompt containing the question, its answer, and the set of supporting paragraphs in random order. A reasoning model is then prompted to output a JSON list of `{title, next}` pairs specifying how the supporting paragraphs chain together. Structured output constraints ensure the model produces valid JSON conforming to the expected schema, and a maximum of 3 retries are scheduled to recover from parsing failures.

The LLM responses are post-processed with fuzzy title matching to resolve minor discrepancies between the model's output and the actual paragraph titles. Each parsed chain is then converted into evidence triplets $(p_j, \cdot, p_i)$—where paragraph p_j leads to paragraph p_i—and validated: we verify that the resulting directed graph is acyclic, has the correct number of nodes for the expected topology, and that every source title appears in the sample's context paragraphs. Samples that fail validation are retried with increased temperature; those that still fail after all retries are discarded.

The prompt template for validating the evidence chain, and a corresponding instantiated example can be found in Table 9 and Table 10, respectively.

D.3 VECTORIZATION

We employ a multilingual embedding model, BGE-m3 (Chen et al., 2024), to vectorize queries and document chunks. The BGE-m3 is frozen throughout our training, isolating the unique contribution of performance improvement by our framework. We adopt Faiss (Johnson et al., 2019) IndexFlatIP for vector database construction and retrieval similarity searching.

D.4 GRAPH CONSTRUCTION PIPELINE

For each training sample we parse its annotated evidence chain into a directed graph $G = (V, E)$ whose nodes are passages and whose edges encode hop-level dependencies. We verify that G is acyclic and that its topology matches the expected question type (e.g., a single entry node for compositional questions, parallel entry nodes for comparison questions); samples that fail these checks are discarded. We then traverse G in breadth-first order: at each layer the current-layer passages serve as positive next-hop targets, while a set of embeddings randomly drawn from the corpus pool (excluding the current and future positive nodes) serve as negatives. This layer-wise decomposition yields one contrastive training group per hop, in direct correspondence with the hop-indexed retrieval at inference time.

On a higher level, this graph formulation mimics the DAG, i.e., the inference-time hop sequence, enabling contrastive training with structural and scalable supervision.

D.5 COMPARISON QUESTION AUGMENTATION

Comparison questions are questions that compare two or more entities (see Appendix A). More generally, for a comparison question with two first-layer passages P_a and P_b, we generate two training graphs:

1. query $\rightarrow P_a \rightarrow P_b$: first retrieves P_a, then (conditioned on the rewritten query) retrieves P_b as a leaf node.
2. query $\rightarrow P_b \rightarrow P_a$: the retrieval order is reversed.

Each variant is an independent training graph with its own negative samples drawn per hop. This augmentation doubles the number of training examples originating from comparison questions and ensures that TreeHop learns to sequentially retrieve both passages regardless of the order in which they appear.

We summarize the goals and the remaining data record for each stages in the following table.

Cleaning Stage	**Goal**	**Remaining Records**
Initial Train Data		233,487
Topology Validation	Ensure the presence (need) of a follow-up retrieval step and the graph is acyclic.	215,378
Evidence Integrity Check	Verify that the generated reasoning topology matches the evidence chain. (e.g., a "three-hop" question can only have exactly two follow-up queries)	178,394
Comparison Question Augmentation	Enrich trainable data	249,133

E PROMPT TEMPLATE FOR EVIDENCE CHAIN VALIDATION

We present prompt templates for validating evidence chain. This ensures the evidence chain matches the question composition path, which is crucial for a fine-grained train dataset.

Prompt Template for Evidence Chain Validation

SYSTEM PROMPT:

Given a multi-hop question, its answer, and a set of supporting paragraphs, determine the correct reasoning order that connects the paragraphs into a directed acyclic graph (DAG) matching the specified topology.

You must output ONLY a JSON object with a list of dictionaries where each dictionary has a "title" key and a "next" key. The "next" key should contain the title of the next paragraph in the correct reasoning order.
The first title should be the starting point of reasoning, and the last title should be the paragraph that leads to the final answer.

IMPORTANT: Output ONLY the JSON object, nothing else. Do not include any explanation, markdown formatting, or extra text. Example: if the reasoning order is P2 $\rightarrow$ P1, P3 $\rightarrow$ P1, P1 $\rightarrow$ P4, and the titles of P1, P2, P3, P4 are "Title1", "Title2", "Title3", "Title4" respectively,
then you should output:

```
["title": "Title2", "next": "Title1", "title":
"Title3", "next": "Title1", "title": "Title1", "next":
"Title4", "title": "Title4", "next": "ANSWER"]
```

USER PROMPT:

```
Graph type: {graph_type}
Number of supporting paragraphs: {num_support}

Question: {question}
Answer: {answer}

Supporting paragraphs (in random order):
  {paragraphs_text}

Now, output the correct reasoning chain in the specified
JSON format.
```

Table 9: Prompt template for validating evidence chain.

Example Instantiated Prompt

```
Graph type: 2-hop
Number of supporting paragraphs: 2

Question: Who is the mayor of the city where the Battle of Gettysburg took place?
Answer: John Smith

Supporting paragraphs (in random order):
  -- Paragraph 1 --
  Title: Gettysburg, Pennsylvania
  Text: Gettysburg is a borough and the county seat of Adams County...
  -- Paragraph 2 --
  Title: Battle of Gettysburg
  Text: The Battle of Gettysburg was fought July 1–3, 1863, in and around the
town...
Now, output the correct reasoning chain in the specified
JSON format.
```

Expected LLM Output:

```
[{"title": "Battle of Gettysburg", "next":
"Gettysburg, Pennsylvania"},
  {"title": "Gettysburg, Pennsylvania", "next":
"ANSWER"}]
```

Table 10: A concrete example of the fully instantiated prompt for a 2-hop question. The LLM receives the system-level instruction followed by the user prompt (Table 9), and is expected to produce the JSON output shown at the bottom.

F 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 four evaluate datasets for Iter-RetGen.

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

Table 11: 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 12: 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 13: Prompt template for Iter-RetGen on 2Wiki, MuSiQue and MultiHop-RAG evaluate datasets.

G ACCURACY PROMPT FOR GPT 5 MINI

Following previous work (Zhuang et al., 2024), we disclose accuracy prompt for evaluating end-to-end QA performance from Table 2.

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 14: Accuracy Prompt for Evaluating end-to-end QA performance

H QA PROMPT FOR END-TO-END EVALUATION

For the downstream LLMs (Llama3.1-8B-Instruct, Qwen3.6-27B 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 three LLMs below, where retrieved chunks are naively concatenated in descending order of cosine similarity, then further concatenated in retrieval iteration order.

For decoding, we set top-p, top-k and repetition penalty to 1 and the maximum number of new tokens to 1024 for all three models, and set temperature to 0.8 for Llama3.1-8B-Instruct, 0.3 for Qwen3.6-27B and 0.6 for Qwen2.5-7B-Instruct.

QA Prompt
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 15: Downstream LLM Prompt for Evaluating end-to-end QA performance

I EXPERIMENT ON ALL QUERY TYPES

In subsection 4.1, we explicitly selected iterative retrieval questions for our benchmark, filtering out questions that do not require iterative retrieval to gather all necessary information. However, some may raise concerns regarding TreeHop's generalizability when questions that do not demand a retrieval system's multi-hop capability are also considered. To address this, we present a revision of benchmarks reported in Table 1, where we test TreeHop's performance with all query types included. All experimental settings are identical to those described in subsection 4.2, 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**	**Chain**	**Avg.K**	**Latency**	**Recall@5**	**Chain**	**Avg.K**	**Latency**
Baselines								
Direct-R@5	57.5	29.4	5	0.003	54.6	36.3	5	0.002
Advanced RAG								
Iter-RetGen(8B)@5 iter2	66.4	57.8	9.9	4.713	60.3	40.6	9.9	4.888
Iter-RetGen(8B)@5 iter3	69.8	60.9	14.8	7.286	63.5	42.9	14.8	7.357
Iter-RetGen(0.6B)@5 iter2	63.1	53.6	6.4	2.996	58.7	39.4	9.5	2.904
Iter-RetGen(0.6B)@5 iter3	65.0	56.5	8.8	4.790	59.4	43.0	13.7	4.695
EfficientRAG@5	63.1	53.8	4.4	2.802	53.9	38.0	5.7	2.859
Ours								
TreeHop@5 iter2	**69.2**	**61.0**	9.2	**0.103**	**61.6**	**43.0**	7.7	**0.115**
TreeHop@5 iter3	**72.8**	**62.4**	13.1	**0.202**	**64.3**	**46.9**	11.3	**0.180**

Table 16: TreeHop performance when all query types are included. **Chain** is the complete evidence-chain recall defined in subsection 4.2. **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.

J Comparison with Hierarchical Summarization Retrieval

RAPTOR (Sarthi et al., 2024) indexes a corpus by recursively clustering and summarizing passages, so retrieval draws on a pool holding both original passages and LLM-written summaries. We evaluate both of its modes under the protocol of subsection 4.2: *collapse* flattens the tree into a single top-K, while *traversal* descends layer by layer, taking top-5 at each. Only leaf nodes carry an evidence title, so only they can close a chain; at $K = 5$ collapse spends 1.07–1.16 of its five slots on summary nodes, and traversal reaches leaves only at its final iteration.

	HOTPOT				2WIKI				MUSIQUE				MULTIHOP-RAG			
Retriever	**Recall**	**Chain**	**Avg.K**	**Lat.**	**Recall**	**Chain**	**Avg.K**	**Lat.**	**Recall**	**Chain**	**Avg.K**	**Lat.**	**Recall**	**Chain**	**Avg.K**	**Lat.**
Baselines																
Direct-R@5	68.3	43.5	5	0.002	49.3	12.7	5	0.002	45.4	16.4	5	0.002	48.6	15.8	5	0.002
Direct-R@10	71.2	50.5	10	0.003	53.2	16.9	10	0.003	49.8	20.2	10	0.002	53.0	19.3	10	0.002
Hierarchical Summarisation																
RAPTOR@5 collapse	65.2	38.8	5	5.438	47.1	10.9	5	5.640	42.5	14.1	5	5.231	47.5	14.5	5	6.372
RAPTOR@10 collapse	72.4	50.2	10	5.489	51.5	15.7	10	5.910	50.8	21.9	10	5.165	54.8	20.4	10	6.309
RAPTOR@15 collapse	**76.0**	**56.0**	15	5.347	53.6	18.3	15	5.721	**55.2**	26.2	15	5.004	56.2	22.6	15	6.256
RAPTOR@5 traversal iter2	40.9	19.7	10	6.843	23.4	8.1	10	6.969	27.4	9.7	10	6.417	28.6	8.7	10	7.777
RAPTOR@5 traversal iter3	21.6	8.4	15	7.095	11.0	4.2	15	7.259	18.0	5.6	15	6.647	10.1	5.2	15	8.419
Ours																
TreeHop@5 iter3	74.5	54.9	14.2	**0.202**	**66.8**	**48.1**	14.0	**0.204**	53.7	**27.7**	13.5	**0.200**	**66.1**	**27.9**	12.2	**0.181**

Table 17: RAPTOR against direct retrieval and TreeHop. **Bold** numbers indicate the best performance at a comparable Avg.K (≈14–15).

At a matched Avg.K of 5 RAPTOR trails direct retrieval by 1.1%-3.1% recall and 1.3%-4.7% Chain, the summary slots costing it more than the enriched pool returns. A larger Avg.K recovers this: at $K = 15$, comparable to TreeHop's 12.2–14.2 chunks at the third iteration, RAPTOR reaches a Chain of 56.0% on HotpotQA against TreeHop's 54.9%, and 26.2% on MuSiQue against 27.7%. On 2WikiMultiHop and MultiHop-RAG it reaches only 18.3% and 22.6%, against 48.1% and 28.9%. The gap tracks how far the evidence chain must bridge: summarisation enriches the pool that the first passages are drawn from, but it never uses a retrieved passage to reformulate the query, so the later hops of a 2WikiMultiHop chain stay out of reach. Traversal is weaker on every dataset, since expansion is confined to children of already selected nodes and a passage becomes unreachable once its ancestor is pruned. Latency compounds the picture: collapse costs 5.0–6.4 s per query and traversal, which searches once per layer, 6.4–8.4 s, between 25 and 47 times TreeHop's at a comparable Avg.K. Restructuring the index therefore buys no efficiency advantage over rewriting the query, on top of requiring an LLM summarisation pass over every cluster at every layer to build.

K Experiment on Different Base Embedding Model and Prompt Template

TreeHop is fundamentally trained upon embeddings generated by BGE-m3. This raises question on whether the same model architecture can adapt to other base models and other prompt templates. To validate this, we replaced BGE-m3 with Qwen3-Embedding-4B model (Yang et al., 2025), trained and benchmarked the TreeHop model variant using the identical train dataset, evaluate datasets and metrics demonstrated in subsection 3.4, subsection 4.1 and subsection 4.2. All hyper-parameters are the same as in subsection 3.5. Prompt templates for both BGE-m3 embedding and Qwen3-Embedding-4B are shown in Table 18 results are shown in Table 19.

As shown in Table 19, the TreeHop architecture transfers effectively to the Qwen3-Embedding-4B base model, yielding substantial improvements over both the direct retrieval baseline and the BGE-m3-based TreeHop variant across all four benchmarks.

First, comparing the two direct retrieval baselines, Direct-R(Qwen)@5 already outperforms Direct-R(BGE)@5 by a considerable margin (e.g., 80.8 vs. 68.3 on HotpotQA, 64.3 vs. 49.3 on 2Wiki), confirming that the stronger base embeddings of Qwen3-Embedding-4B provide a higher-quality starting point. Critically, TreeHop preserves and amplifies this advantage: TreeHop(Qwen)@5 at iteration 2 achieves 84.1 recall on HotpotQA and 81.0 on 2Wiki, representing absolute gains of 10.5 and 18.3 points over TreeHop(BGE)@5 iter2, respectively. On MuSiQue and MultiHop-RAG, the Qwen-based variant similarly leads by 7.4 and 6.1 points at the second iteration. These consistent improvements across datasets of varying difficulty indicate that TreeHop's iterative tree-expansion mechanism is not overfitted to BGE-m3's embedding space but generalizes to a different encoder architecture.

Question Prompt for BGE-m3
[question]
Document Chunk Prompt for BGE-m3 # Paragraph title: *[title]* ### Content *[content]*
Question Prompt for Qwen3-Embedding-4B Given a search query, retrieve relevant passages that answer the query. Here is the query: *[question]*
Document Chunk Prompt for Qwen3-Embedding-4B Title: *[title]* *[content]* ...

Table 18: Prompt templates for BGE-m3 and Qwen3-embedding-4B.

Second, the iterative refinement behavior is preserved under the new base model. Moving iteration from the second to the third, TreeHop(Qwen)@5 continues to gain recall on all four benchmarks (e.g., +0.3% on HotpotQA, +0.5% on 2Wiki, +1.3% on MuSiQue, +3.2% on MultiHop-RAG), mirroring the pattern observed with the BGE variant.

Third, the latency profile reflects the expected cost of using a larger base encoder. TreeHop(Qwen)@5 iter2 incurs roughly 0.16-0.17s per query compared to 0.11s for TreeHop(BGE)@5 iter2, an approximately 1.5 times increase attributable to the 4B-parameter Qwen embedder (embeds sentences with 2560 dimensions) versus the lighter BGE-m3 (embeds sentences with 1024 dimensions). This trade-off is consistent across datasets and iterations, and the absolute latency remains practical for offline or batch retrieval scenarios. Users can therefore choose between the BGE variant for latency-sensitive deployments and the Qwen variant when recall is the primary objective.

Retriever	HOTPOT Recall	HOTPOT Avg.K	HOTPOT Latency	2WIKI Recall	2WIKI Avg.K	2WIKI Latency	MUSIQUE Recall	MUSIQUE Avg.K	MUSIQUE Latency	MULTIHOP-RAG Recall	MULTIHOP-RAG Avg.K	MULTIHOP-RAG Latency
Baselines												
Direct-R(BGE)@5	68.3	5	0.002	49.3	5	0.002	45.4	5	0.002	48.6	5	0.002
Direct-R(Qwen)@5	80.8	5	0.002	64.3	5	0.002	54.5	5	0.002	57.9	5	0.002
Ours												
TreeHop(BGE)@5 iter2	73.6	9.5	0.117	62.7	9.5	0.114	52.0	9.2	0.113	60.2	8.7	0.108
TreeHop(BGE)@5 iter3	74.5	14.2	0.202	66.8	14.0	0.204	53.7	13.5	0.200	66.1	12.2	0.181
TreeHop(Qwen)@5 iter2	84.1	9.3	0.167	81.0	9.4	0.160	59.4	9.1	0.160	66.3	8.2	0.163
TreeHop(Qwen)@5 iter3	84.4	13.9	0.282	81.5	13.9	0.281	60.7	13.4	0.263	69.5	11.3	0.246

Table 19: Results of baselines, concurrent advanced RAG solutions and TreeHop variant on four MHQA datasets. Direct-R@5 denotes the direct retrieval at top-5 baseline retrieval setting (See subsection 4.2 for baseline and advanced RAG details).

L On the Exclusion of MDR from Our Benchmark

Multi-hop Dense Retrieval (Xiong et al., 2021) is a LLM-free iterative dense retriever that conditions each retrieval step on previously retrieved evidence without generating a rewritten textual query, and is therefore a natural point of comparison for TreeHop. We nonetheless report it separately from our main benchmark for a methodological reason rather than for convenience.

MDR learns its own query and passage encoders end-to-end, so its retrieval quality is naturally coupled to the corpus those encoders are trained on—here, the HotpotQA training set. Table 20 makes this concrete. MDR is a strong in-distribution retriever: on the HotpotQA distractor benchmark it reaches 92.3% recall and 88.0% complete evidence-chain recall by the second hop, the highest of any method we examine. On MultiHop-RAG, a different domain, the same checkpoint reaches 32.7% recall and 8.0% chain at the second hop, below the Direct-R@5 baseline computed on the shared BGE-m3 encoder (48.6% / 15.8%). This spread reflects the encoder's training corpus rather than a property of iterative retrieval as such, and it is orthogonal to the question we study, which isolates the query-update mechanism over a shared, frozen encoder.

Every method in our main benchmark is compared over the same frozen BGE-m3 embeddings and under matched retrieval budgets across all four datasets, precisely so that the comparison isolates multi-hop query-update behaviour from encoder- and corpus-specific fitting. Because MDR's released checkpoint carries its own encoder trained on one of our evaluation corpora, placing its numbers directly in the main table would compare encoder training as much as multi-hop retrieval. We therefore report MDR here rather than in the main benchmark, and leave a controlled, retrained, shared-encoder comparison—under which the encoder coupling would be normalised—to future work.

	HOTPOT				MULTIHOP-RAG			
Retriever	**Recall**	**Chain**	**Avg.K**	**Lat.**	**Recall**	**Chain**	**Avg.K**	**Lat.**
Baseline (BGE-m3)								
Direct-R@5	68.3	43.5	5	0.002	48.6	15.8	5	0.002
MDR (own end-to-end-trained encoder)								
MDR hop 1	65.4	36.5	5.0	0.002	26.1	4.4	5.0	0.002
MDR hop 2	92.3	88.0	8.3	0.148	32.7	8.0	7.2	0.167
MDR hop 3	93.3	89.6	10.2	0.288	34.8	9.2	8.2	0.360
Ours								
TreeHop@5 iter3	74.5	54.9	14.2	0.202	66.1	28.9	12.2	0.181

Table 20: Cross-dataset behaviour of MDR's publicly released checkpoint, which carries its own end-to-end-trained encoder, alongside the shared-BGE-m3 Direct-R@5 baseline and TreeHop. *hop 1* is a single retrieval with MDR's encoder; Recall is micro-averaged and Chain is complete evidence-chain recall. MDR attains the highest in-distribution scores on HotpotQA—the corpus its encoder was trained on—but falls below the Direct-R@5 baseline on the out-of-domain MultiHop-RAG corpus, reflecting encoder–corpus coupling rather than multi-hop retrieval capability. All methods use top-5 retrieval.

M 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	62.7	9.5	52.0	9.2	60.2	8.7
TreeHop@5 iter3	66.8	14.0	53.7	13.5	66.1	12.2
75% of Training Dataset						
TreeHop@5 iter2	61.8(0.9↓)	9.4	48.8(3.2↓)	9.1	59.3(0.9↓)	8.5
TreeHop@5 iter3	65.6(1.2↓)	13.7	49.7(4.0↓)	13.3	63.6(2.5↓)	11.9
50% of Training Dataset						
TreeHop@5 iter2	60.4(1.4↓)	9.2	47.0(1.8↓)	8.9	56.0(3.3↓)	7.8
TreeHop@5 iter3	62.3(3.3↓)	11.9	47.2(2.5↓)	12.4	58.8(4.8↓)	10.9
25% of Training Dataset						
TreeHop@5 iter2	57.1(3.3↓)	8.4	46.0(1.0↓)	7.9	53.1(2.9↓)	6.6
TreeHop@5 iter3	59.3(3.0↓)	11.4	46.4(0.8↓)	11.3	55.6(3.2↓)	8.7

Table 21: TreeHop performance trained upon subsampled dataset.

Our experiment reveals that TreeHop's performance decays as the training dataset size decreases. We observe a notable improvement on all benchmarks, indicating that there is still room for improvement if we further scale up the training dataset.

N 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., 2023b; Vakulenko et al., 2020), readers may still 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.

Role of Prompt Length 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 K is also critical to the accuracy and efficiency. Even we have tools like reranker models, they do not preclude irrelevant context. Rewriters like TreeHop connect semantically distant documents without extremely large number of K.

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 all the required facts cannot be gathered without multi-step reasoning. 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.

O 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 $ForgetGate$ term in the Equation 3 serves for this replacing purpose by attending over the query-chunk pair and subtracting the query semantics that the chunk has already resolved. 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"*.

P 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.

Q 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.

R TREEHOP INFERENCE PROCEDURE

The full multi-hop inference loop, including both pruning rules of subsection 3.3, is given in Figure 3.

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 $\mathcal{C}$.

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

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

S Full Model Architecture as Implemented

Out subsection 3.2 presents TreeHop through its outer residual form (Equation 3) and a simplified cross-attention gate (Equation 4) for clarity of exposition. This appendix discloses the *complete* architecture exactly as realised in the released implementation, so that the mathematical description, the released checkpoint, and the parameter count reported in subsection 3.2 can be reconciled unambiguously. Throughout, d denotes the frozen BGE-m3 embedding dimension ($d = 1024$), a the attention (gate) width, m the width of the residual MLP, H the number of attention heads, and L the number of stacked cross-attention blocks per gate. $\odot$ is the element-wise product and $\mathrm{Norm}(\cdot)$ the inter-block RMS normalization.

S.1 Outer Update: Two Learned Gates

The implemented per-hop update fuses the previous query q_r and a retrieved chunk c_r^i through *two* learned gates rather than the single gate of Equation 4:

$$\text{TreeHop}(q_r, c_r^i) = q_r - ForgetGate(q_r, c_r^i) + UpdateGate(q_r, c_r^i). \tag{5}$$

Both gates are instances of the same *stacked cross-attention* module $CrossAttn$ (subsection S.2), which extends the $CrossAttn_u$ of Equation 4; they differ only in their value stream:

$$UpdateGate(q, c) = CrossAttn_u(q, c), \tag{6}$$

$$ForgetGate(q, c) = CrossAttn_f(q, c). \tag{7}$$

As in Equation 4, $CrossAttn_u$ applies its value projection V_u to the chunk c. $CrossAttn_f$ instead applies its value projection V_f to the query q.

S.2 Stacked Cross-Attention Block

Each gate $g \in \{u, f\}$ stacks L multi-head cross-attention layers MHA_g with residual connections and projects every layer back to the residual width d (the value source is c for $g{=}u$ and q for $g{=}f$; see below):

$$x^{(1)} = W_p^{(1)}\, MHA_g^{(1)}(q, c), \tag{8}$$

$$x^{(l)} = x^{(l-1)} + W_p^{(l)}\, MHA_g^{(l)}\big(\mathrm{Norm}(x^{(l-1)}), c\big), \quad l = 2, \ldots, L, \tag{9}$$

$$CrossAttn_g(q, c) = x^{(L)}, \tag{10}$$

where $W_p^{(l)} \in \mathbb{R}^{d \times aH}$ are bias-free projections and Norm is applied only to the $L{-}1$ residual inputs (so a single-layer stack reduces to a bare attention). We use RMS normalization because TreeHop's output is consumed only as a *direction* (every downstream consumer L_2-normalizes it and ranks by cosine), so a pure rescale leaves the retrieval geometry intact, whereas layer normalization would subtract a per-vector mean and rotate the embedding.

S.3 Multi-Head Layer with an Output Gate

Writing x for the query-side input, c for the chunk, and v for the gate's value source ($v{=}c$ for $g{=}u$, $v{=}q$ for $g{=}f$), each layer MHA_g concatenates H heads and applies an additional learned *output gate*:

$$MHA_g(x, c) = a_{\text{cat}} \odot \text{softmax}\big(W_o\, a_{\text{cat}}\big), \quad a_{\text{cat}} = \Big\|_{h=1}^{H} head_h(x, c), \tag{11}$$

with output-gate matrix $W_o \in \mathbb{R}^{aH \times aH}$. Each head reuses the projection form of Equation 4, with learned projections $Q_g, K_g, V_g \in \mathbb{R}^{a \times d}$, followed by a residual MLP:

$$\alpha = \text{softmax}\big(Q_g(x) \odot K_g(c)\big), \tag{12}$$

$$a_{\text{head}} = \text{dropout}(\alpha) \odot V_g(v), \tag{13}$$

$$head(x, c) = W_s\, \text{MLP}(a_{\text{head}}) + a_{\text{head}}, \qquad W_s \in \mathbb{R}^{a \times m}. \tag{14}$$

As in Equation 4, the conventional scaled dot product is replaced by the element-wise product $Q_g(x) \odot K_g(c)$ so that attention acts *per dimension* of the embedding rather than across a token sequence. The residual MLP is a stack of n_{mlp} post-norm residual blocks,

$$\text{ResNet}(z) = z + \text{ReLU}\big(W\, \text{LayerNorm}(z)\big), \qquad W \in \mathbb{R}^{m \times m}, \tag{15}$$

optionally preceded by a single $a \to m$ projection when $a \neq m$ (not needed for the released configuration, where $a = m$).

S.4 Released Configuration, Parameter Count, and Checkpoint

All main results (Table 1–Table 4) use a single checkpoint with the configuration in Table 22. BGE-m3 is frozen; the 176.3M parameters below are TreeHop's alone.

Hyper-parameter	**Value**
Embedding dim. d (`x_size`)	1024
Gate / attention width a (`g_size`)	2048
Residual-MLP width m (`mlp_size`)	2048
Residual blocks per head n_{mlp}	3
Attention heads H (`n_head`)	1
Stacked blocks per gate L (`n_layer`)	3
Inter-block norm	RMS
Dropout (outer / within gates)	0.1 / 0.0
Loss	InfoNCE
Negatives n_{neg}	6
Temperature τ	0.15
Learning rate	2×10^{-5}
Seed	1307
Selected epoch	2
Total trainable parameters	**176,336,896**

Table 22: Exact configuration of the released TreeHop checkpoint used for all main experiments (Table 1–Table 4). These values match the hyper-parameters encoded in the released checkpoint filename.

Table 23 breaks the parameter count down by module. The model comprises two structurally identical gates ($\mathrm{ForgetGate}$ and $\mathrm{UpdateGate}$), each contributing $88{,}168{,}448$ parameters.

Module	**Params**
Per cross-attention layer $\mathrm{MHA}^{(l)}$ *(*$\times L$=3 *per gate)*	
$Q/K/V$ projections ($3 \times 1024 \to 2048$)	6,297,600
Residual MLP ($3 \times \mathrm{ResNet}_{2048}$)	12,601,344
MLP scale W_s ($2048 \to 2048$)	4,196,352
Output gate W_g ($2048 \to 2048$)	4,196,352
Layer subtotal	**27,291,648**
Per gate	
Cross-attention layers ($\times 3$)	81,874,944
Inter-block projections W_p ($\times 3$, $2048 \to 1024$)	6,291,456
RMS norms ($\times 2$)	2,048
Gate subtotal	**88,168,448**
TreeHop total (2 gates)	**176,336,896**

Table 23: Parameter breakdown of the released TreeHop configuration (Table 22). Both gates share the same architecture; BGE-m3 is frozen and excluded.