Accuracy or Efficiency: Analyzing the Trade-offs in Performance when Using Predicted Hop Counts in Multi-Hop RAG

Avinash Nair
an586@cornell.edu
Cornell University - Cornell Tech
New York, New York, USA
AN586

Jasper Levy
jl4537@cornell.edu
Cornell University - Cornell Tech
New York, New York, USA
JL4537

Theodore Wu
theowu1029@gmail.com
Cornell University - Cornell Tech
New York, New York, USA
TW577

Abstract

Retrieval-Augmented Generation (RAG) systems are an increasingly popular application for large language models. Multi-hop reasoning is an important class of tasks within RAG that requires reasoning over multiple documents to answer complex queries. Current RAG systems struggle with multi-hop reasoning tasks without massive amounts of context. As such, retrieval systems for completing multi-hop RAG tasks can become expensive, redundant, and inefficient at scale. This paper investigates strategies to optimize the trade-off between accuracy and efficiency in multi-hop reasoning tasks by predicting the hop count needed for the task. In addition, we introduce a metric - Accuracy Efficiency Index (AEI) - by which we can measure this trade-off. Our work shows that using the predicted hop count plus a certain amount of retrieval steps can lead to increased accuracy while minimizing retrieval costs. By analyzing performance changes as additional context is included, our work provides insight into scalable and cost-efficient implementations for RAG systems.

1 Introduction

1.1 Related Works

Our study draws inspiration from the work of Krishna et al. in "Fact, Fetch, and Reason: A Unified Evaluation of Retrieval-Augmented Generation" (Krishna et al., 2024). This paper introduces a comprehensive framework for evaluating Retrieval-Augmented Generation (RAG) systems by integrating factual retrieval, multi-hop reasoning, and generative responses into a unified assessment methodology. Furthermore, they introduce the FRAMES dataset to test multi-hop reasoning abilities for different LLMs. Krishna et al. explore the challenges inherent in retrieval-based pipelines, including retrieval accuracy, reasoning complexity, and the interplay between retrieved context and generative quality.

1.2 Problem Definition

Retrieval-Augmented Generation (RAG) systems, particularly those incorporating Large Language Models (LLMs), are integral to applications like question-answering agents, knowledge retrieval, and customer support tools. Further, multi-hop reasoning RAG tasks require retrieving and synthesizing information from multiple sources to answer complex questions accurately. However, these systems face challenges:

- **Cost Implications**: Token-based pricing structures for LLMs can lead to high operational costs.
- **Efficiency Concerns**: Large retrieval contexts increase response times, which is detrimental to real-time applications.

Building on the foundational work by Krishna and colleagues, our study utilizes their BM25-based retrieval pipeline and multi-hop reasoning approach. We extend their work by adjusting the retrieval pipeline and introducing a predictive component to dynamically adjust retrieval depth (number of hops), to measure its impact on multi-hop RAG performance. This addition allows us to explore the balance between accuracy, efficiency, and cost, further addressing scalability challenges in real-world RAG applications. To quantify this balance, we also introduce the Accuracy Efficiency Index (AEI) metric to measure the trade-off between accuracy and retrieval efficiency.

The objective of this study is to optimize the trade-off between accuracy and efficiency in multi-hop Retrieval-Augmented Generation (RAG) systems by predicting the optimal number of retrieval hops. Although generally increasing the number of retrieval hops can improve accuracy by incorporating more context, it also results in higher computational costs, increased token usage, and longer latency.

The core research questions addressed are the following.

- Can the number of retrieval hops required for accurate reasoning be predicted effectively using a machine learning model?
- How does the trade-off between accuracy, efficiency, and cost vary across different reasoning types as the number of hops changes?
- What is the optimal range of additional hops ($p + 1, p + 2, \ldots$ where p is the predicted value) to maximize accuracy improvements while minimizing inefficiencies?

As discussed in our results section below, we found that dynamically assigning the retrieval depth, or number of hops, can lead to improved accuracy while effectively managing efficiency (scoring high on our introduced AEI metric) better than hard coded retrieval depths.

2 Dataset

We based our experiments on the FRAMES dataset introduced in the "Fact, Fetch, and Reason" paper, which consists of multi-hop reasoning questions requiring factual retrieval from structured knowledge sources like Wikipedia. These questions and their associated links and answers are organized in a CSV file. Key characteristics of the dataset include:

- **Question Types**: A mix of factual, deductive, and common sense reasoning questions, each requiring varying depths of retrieval.
- **Context Retrieval**: Associated Wikipedia articles and links form a knowledge graph that supports multi-hop reasoning tasks.
- **Hop Constraints**: Questions are annotated with the number of hops needed to retrieve sufficient context for accurate reasoning.

For other pre-processing, we filtered out 7 questions which were used for by the retriever for few-shot prompting (more on this in the next section) in order to prevent contamination of the test set data.

For each question, Wikipedia articles are scraped to build a retrieval pool. Sentence summaries and embeddings of these articles are used to construct a BM25-based pipeline for dynamic retrieval.

2.1 Exploratory Data Analysis

For the scope of our experiments, we decided to modify the original dataset, which consists of 824 questions, each requiring a varied number of hops to accurately answer. In our preliminary data exploration, we found that most of the questions in the dataset (796 questions) are answerable within 2-7 hops, with the higher hop counts being outliers. As such, we created a condensed subset of the original dataset, containing only questions that required 2-7 hops to be accurately answered.

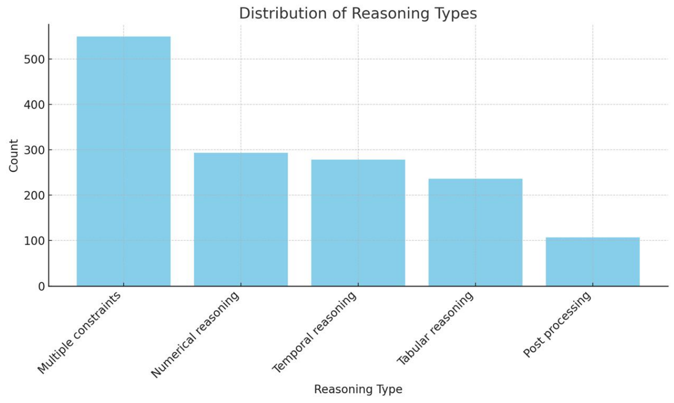

Figure 1: Distribution of reasoning types

To better understand the query distribution, we calculated key statistics:

- **Mean:** 3.37
- **Standard Deviation:** 1.99
- **Lower Bound (Mean - 2×Std):** -0.61
- **Upper Bound (Mean + 2×Std):** 7.35

Part of our analysis dives into the different reasoning types and trade-off performance between batches of reasoning types in the dataset. The reasoning types present in the data are the following:

- **Numerical Reasoning**: Performing calculations, comparisons, or counting tasks.
- **Tabular Reasoning**: Analyzing statistics or information in structured formats (i.e. Tables).
- **Multiple Constraints**: Queries with multiple conditions to reach a unique answer.
- **Temporal Reasoning**: Reasoning through timelines and considering chronological order.
- **Post-Processing**: Performing additional work on compiled context.

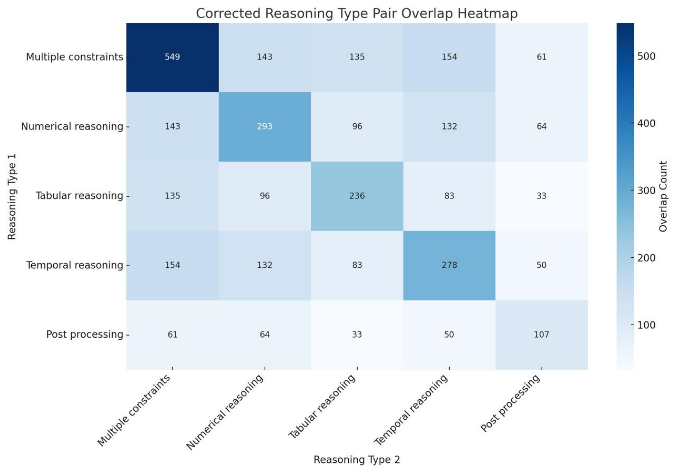

Figure 2: Heat Map of reasoning type overlap

The dataset is quite varied in regards to these reasoning types, and we see this in **figure 1**, which shows the distribution of labeled reasoning types in the dataset. Notably, as the heat map in **figure 2** shows, there is quite a bit of overlap between reasoning types as most queries are classified under multiple reasoning types.

3 Method

Our methodology consists of two primary parts. In the first part, we build a version of the BM-25 pipeline to measure how the different LLMs performed across topics and different pre-determined hop counts. In the second part, we start by first building a model to predict the necessary hop count. We then incorporate this prediction into our pipeline to measure performance and AEI scores.

BM25, or Best Match 25, is a probabilistic-based ranking function widely used for information retrieval tasks (Robertson et. al., 1994). At its core, BM25 quantifies the relevance of a document to a query by examining how well the terms in the query match those in the document while considering the frequency of the terms and the overall length of the document. BM25 considers three main factors for document ranking.

Firstly, BM25 considers the frequency of terms. BM25 assumes that terms occurring more frequently in a document are indicative of the document's relevance to the query. Adding more instances of the same term boosts the score of a document but with decreasing significance. This prevents overly frequent terms from disproportionately dominating the ranking.

Secondly, BM25 looks at inverse document frequency. Not all terms carry equal weight. BM25 prioritizes rarer terms over common ones because rare terms are better at distinguishing relevant documents. For example, a document containing the uncommon term "neural" is more likely to be about neural networks than one containing a ubiquitous term such as "consistent" (Jones, 1972).

Lastly, BM25 uses document length normalization. Longer documents naturally have more words and might seem artificially relevant because they have a higher chance of including query terms. BM25 corrects for this by penalizing documents based on their length, ensuring that relevance scores are not biased toward overly verbose content.

3.1 Part 1: Retriever Approach and Initial Evaluation

In the retriever approach, we tried to replicate the BM-25 pipeline detailed in the Krishna paper with some slight modifications. This part of the method was carried out so that we could get an idea of the baseline performance of the tested language models. We followed the basic algorithm proposed in the paper, in which context is added and the query is refined for a pre-determined number of iterations before generation of a final answer. However, we slightly modified this approach, as seen in **Algorithm 1**.

First, we include a step for initial query generation, in which we prompt the LLM to extract key terms from a given question. We do this to help guide the generation of more relevant Wikipedia searches. See **Figure 3** for a diagram of the modified BM-25 pipeline that we use.

Our modified BM-25 pipeline makes use of OpenSearch functionality of the Wikipedia API for searching and collecting relevant pages based on past queries and gathered context (MediaWiki, 2024). We decided to use the API as a lightweight solution to data retrieval. The API search action is used to query for pages containing specific terms, and the search results are ranked using Wikipedia's relevance algorithm.

Our settings use the "engine_autoselect" search profile, meaning that the search engine automatically determines the best profile to use for the query based on input. For our approach, the text of each retrieved Wikipedia article is gathered (up to a set token limit, after which text is cut off - an important limitation) and then used to rank the articles to keep and add to context.

The LLM and prompts are instructed to generate 3-5 word queries that are concise and likely to be key-words in Wikipedia articles that contain the necessary information needed for retrieval. These key-words are then used for the search. Notably, we employ a simple history-caching system to keep track of repeatedly visited documents using the Wikipedia search across multiple queries for a given question. In a production setting, employing a real cache system that stores files locally could help reduce latency of retrieval.

We also added a threshold step for adding retrieved documents after the BM-25 algorithm is ran to determine the most relevant documents. If the threshold is met, the top k documents that meet the threshold, or Wikipedia articles, are added to context for a given iteration. This threshold step ensures that for a given query iteration, if more than one documents are highly similar to the query terms, multiple documents can then be added to the context. This threshold is set to be a significant percentage of the top ranked document's BM-25 score. Without this threshold step, we could

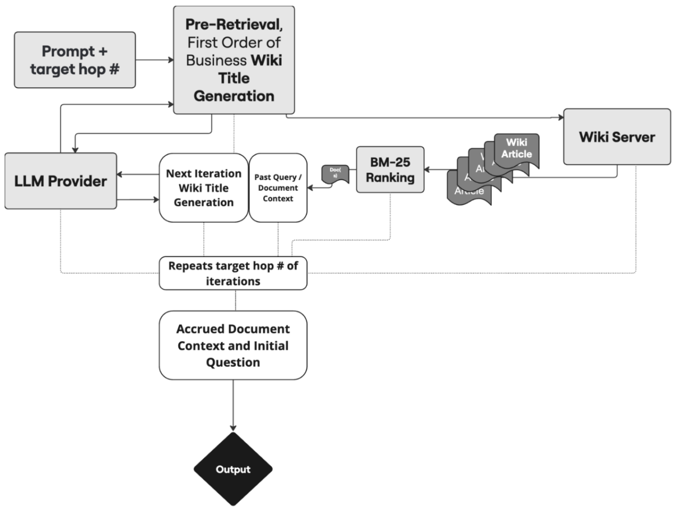
Prompt + target hop #
Pre-Retrieval, First Order of Business Wiki Title Generation
Wiki Server
LLM Provider
Next Iteration Wiki Title Generation
Past Query / Document Context
Doc(s)
BM-25 Ranking
Wiki Article
Article
Repeats target hop # of iterations
Accrued Document Context and Initial Question
Output

Figure 3: Modified BM-25 retrieval pipeline

overlook key documents as the LLM used is instructed to construct new, unique search terms for each new query in a new iteration.

With the threshold step and a condition for the maximum number of documents added to context as part of the iteration loop, we can guarantee that the retrieval uses no more than the designated number of documents (or hops) when generating an answer.

To measure the accuracy of generated answers, we generated embeddings for both the ground truth answer from the dataset and the generated LLM response. We used the sentence transformers library from Hugging Face to get embeddings for the answers. These embeddings were then compared using a cosine similarity score, with scores beating our threshold being marked as accurate generations.

After the pipeline was set up, we evaluated retrieval based on accuracy and token efficiency and computed AEI scores for each LLM at different set hop counts.

3.2 Few-Shot Prompting

Few-Shot prompting is a technique that is used in LLM applications to guide the model to provide the desired output through examples included in the prompt to the model. Few-shot prompting is known to boost performance in translation, question-answering, and other task (Brown et. al., 2020).

For our approach, we used examples of ground truth articles necessary to answer 7 search queries pulled from the FRAMES dataset. Below are two examples of few-shot prompts we use.

Example 1: Numerical Reasoning
Question: How many years earlier would Punxsutawney Phil have to be canonically alive to have made a Groundhog Day prediction in the same state as the US capitol?
Links:

- https://en.wikipedia.org/wiki/Punxsutawney_Phil
- https://en.wikipedia.org/wiki/United_States_Capitol

Algorithm 1 Modified - Multi-Hop Wikipedia Question Answering with BM25

```
Require: Question $Q$, target_docs $n$, queries_per_iteration $k$, docs_per_query $d$, score_threshold $t$, max_tokens $m$
Ensure: Answer $A$, Context History $H$
Initialize:
$H \leftarrow \{Q\}$                                  ▷ Context history
$V \leftarrow \emptyset$                                  ▷ Visited pages
$D \leftarrow \emptyset$                                  ▷ Context documents
$i \leftarrow 0$                                  ▷ Iteration counter
while $|D| < n$ and $i < 2n$ do
    $C \leftarrow$ concatenate$(H)$
    if $i = 0$ then
        $Q_i \leftarrow$ GenerateInitialQuery$(Q)$        ▷ Extract key terms
    else
        $Q_i \leftarrow$ GenerateContextQuery$(C, Q)$        ▷ Use context
    end if
    $P \leftarrow$ WikipediaSearch$(Q_i)$                  ▷ Get pages
    candidates $\leftarrow \emptyset$
    for all page $p \in P$ do
        if $p$.title $\notin V$ then
            content $\leftarrow$ GetWikiContent$(p)$
            candidates $\leftarrow$ candidates $\cup\{(p.\text{title}, \text{content})\}$
        end if
    end for
    if candidates $\neq \emptyset$ then
        ranked $\leftarrow$ BM25Rank$(Q_i$, candidates$)$
        top_score $\leftarrow$ ranked[0].score
        threshold $\leftarrow$ top_score $\times t$
        docs_to_add $\leftarrow \min(d, n - |D|)$
        for all doc in ranked do
            if doc.score $\geq$ threshold and $|D| <$ docs_to_add then
                $D \leftarrow D \cup \{\text{TruncateTokens}(\text{doc}, m)\}$
                $V \leftarrow V \cup \{\text{doc.title}\}$
                $H \leftarrow H \cup \{\text{doc.content}\}$
            end if
        end for
    end if
    $i \leftarrow i + 1$
end while
$A \leftarrow$ GenerateAnswer$(Q, H)$
return $A, H$
```

Answer: 87

Example 2: Temporal Reasoning
Question: Leonardo DiCaprio once won an Oscar for best actor. Who won the award for best costume design sixteen years earlier?
Links:
- https://en.wikipedia.org/wiki/Leonardo_DiCaprio
- https://en.wikipedia.org/wiki/List_of_Academy_Award_winners_and_nominees_for_Best_Actor
- https://en.wikipedia.org/wiki/List_of_Academy_Award_winners_and_nominees_for_Best_Costume_Design

Answer: Colleen Atwood

These prompts provide the LLM with structured examples of the question-answer task in an effort to show the model how to identify key words that are then to be used in the query to the Wikipedia API. This is all done in effort to extract meaningful information to add to the context.

3.3 Accuracy Efficiency Index

The main goal of our study is to understand and optimize the trade-off between accuracy and efficiency in multi-hop RAG. As such, we needed a way to quantify this trade-off with a metric. We thus introduce the Accuracy Efficiency Index (AEI). AEI scores at a given prediction of hop count are calculated as follows:

$$\text{AEI} = \frac{\text{Accuracy over batch}}{\text{Average \# of documents in context}}$$

If AEI scores are plotted against hop count on an axis of average number of retrieved documents for a given prediction p, the peak represents the optimal prediction (i.e. p, $p+1$, $p+2$) at which accuracy is maximized are with minimal additional retrieval overhead. Beyond this peak, increasing the number of retrieved documents in theory yields diminishing returns.

This AEI curve thus helps pinpoint where retrieval becomes inefficient to allow for optimal resource utility without performance compromises. Determining AEI scores and analyzing the curves across different batches of data can help to guide adaptive retrieval pipelines and serve to help optimally calibrate these pipelines to the LLM used.

3.4 Part 2: Hop Prediction Model

The second part of our method involves dynamically incorporating a predicted hop count into our modified BM-25 pipeline. As such, we built a model to predict the required hop counts to answer any given question.

We built a simple neural network and used Term Frequency Inverse Document Frequency (TFIDF) to vectorize questions and feed them to the network. We used the questions and their corresponding ground truth number of links from the FRAMES dataset to train our model. We chose to build a neural network for this because we needed to capture the complex, non-linear relationship between question embeddings and their associated hop counts. We leaned on the ability of neural networks to learn complex patterns in text data in building our model.

The simple neural network achieved 91% accuracy within a tolerance of ±2.0 hops. This result highlights the model's reliability in predicting approximate hop counts, forming a strong foundation for optimizing retrieval strategies in multi-hop RAG tasks. **Figure 4** shows the tight distribution of prediction errors near 0. **Figure 5** is a diagram that shows the simple, fully connected network architecture used, and **figure 6** clarifies the whole process from input vectorization to hop count prediction.

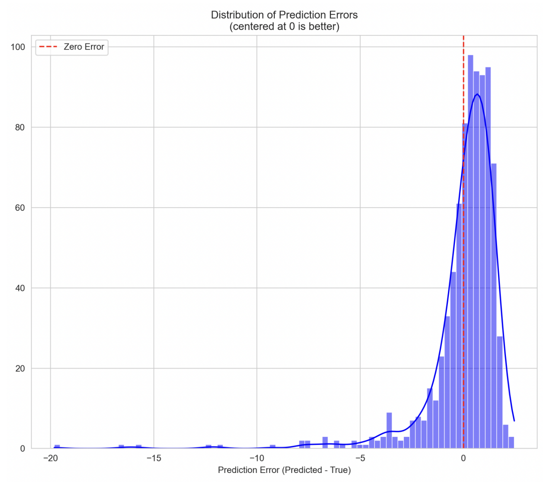

Figure 4: Error distribution for the hop prediction model.

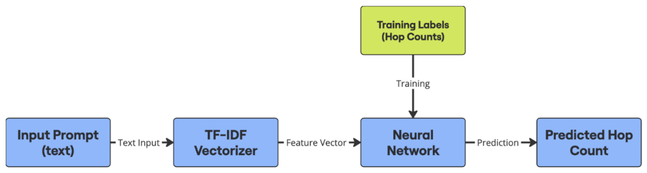

Figure 5: Predictor network architecture

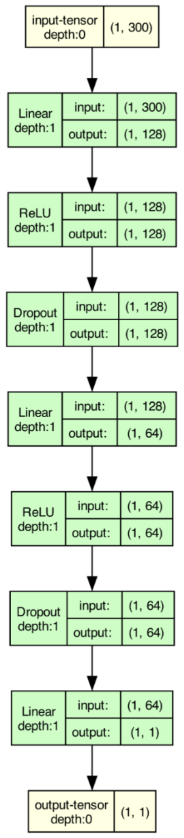

Figure 6: Hop prediction process

3.5 Predictive Hop Count Approach

The final part of our method involved using our predictive model to dynamically adjust retrieval depth in our modified BM-25 pipeline. To accomplish this, we added a step in our algorithm to first feed the question through our vectorizer and model to generate a hop count. This hop count is then returned and used as an input for our retrieve method, which is called for each question. The rest of the algorithm remains largely unchanged. Once again, as in the first part of the method, we computed AEI scores for comparison across all tested LLMs.

3.6 Alternate Approach: LangGraph Multi-Agent

The proposed modified BM-25 pipeline is used in the experiments section of this paper, but it notably has token growth issues and should not be used in a production setting. Token usage increases non-linearly across iterations. Assuming that documents are capped at 10,000 tokens, for a query requiring multi-hop reasoning:

- At iteration 1, one document (10,000 tokens) is retrieved and appended to the context, leading to a likely 10,000-token input.
- At iteration 2, a new document (10,000 tokens) is retrieved, but the previously accumulated 10,000 tokens are re-used as part of the query context. This results in a total token input of 30,000 tokens (10,000 from iteration 1, 20,000 from iteration 2)
- At each iteration, the entire cumulative context, consisting of all previously retrieved documents, is included as input to the model when making a new query.
- This token accumulation continues across iterations, with the input size for iteration i approximated, as well as the total token Input after i iterations:

$$\text{Token Input at Iteration } i = \sum_{k=1}^{i} (k \times \text{Tokens Per Doc})$$

$$\text{Total Token Input After } i \text{ Iterations} = \frac{i(i+1)}{2} \times \text{Tokens Per Doc}$$

Therefore, token growth scales quadratically with the number of iterations for this retrieval system. This leads to inefficiencies in memory and high computational cost which ultimately results in high API costs.

For the sake of this paper and the goal of examining trade-offs with predicted hops, we will overlook these inefficiencies as this pipeline is experimental and built for analysis rather than production RAG.

Another viable approach to this problem that would solve this complex issue is to build a custom LangGraph Multi-Agent system tailored to our retrieval problem. LangGraph is an agent framework for graph-based reasoning with agents. Using an established and more robust framework offers many different benefits including compact context storage, more advanced planning and reasoning, and parallelism using a supervisor agent.

The LangGraph Multi-Agent System can tackle the quadratic growth of token usage gracefully by using compact context storage. Instead of storing entire (or large portions) of retrieved documents, LangGraph can maintain a concise context by storing only planned subqueries generated and query results containing answers and some brief relevant information. This serves to drastically reduce necessary context lengths and free up memory, reduce latency, and lower API call costs.

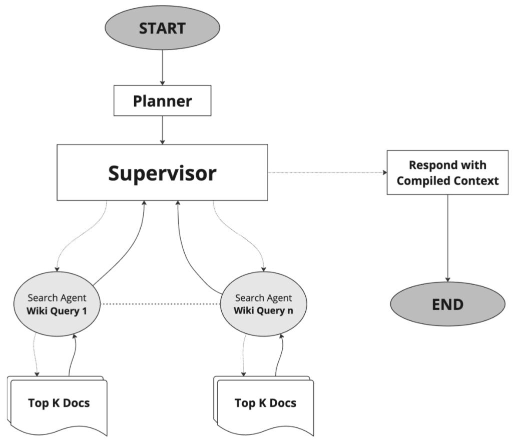
START
Planner
Supervisor
Respond with Compiled Context
END
Search Agent Wiki Query 1
Search Agent Wiki Query n
Top K Docs
Top K Docs

Figure 7: Proposed Multi-Agent system

LangGraph's flexibility of in developing interacting agents also allows for more in-depth reasoning of multi-hop searches through planning agent mechaninisms. By leveraging a "plan-and-execute" approach, high-level planning can be done prior to execution of retrieval for accurate initial query decomposition (LangChain, 2024).

Multi-Agent systems can also make use of "supervisor" agents to help "delegate tasks" or "choose the next worker node or finish processing" in LangGraph (LangChain-AI, 2024). This supervisor approach can open a broad range of opportunities for improving a RAG system. For instance, independent subqueries without contextual dependencies - meaning queries that are not contextually dependent on other queries - can be executed in parallel by multiple search agents that report to the supervisor agent. An example of contextually dependent queries is:

- Query 1: *"What year did Company X acquire Company Y?"*
  - Output: *2010.*
- Query 2: *"Who was the CEO of Company X in 2010?"*
  - Depends on the answer to Query 1.

The supervisor agent would ensure that these "dependent" queries are executed in the proper order by injecting previous answers into dependent queries, whereas for independent queries, search agents can process them in parallel.

This can all be accomplished through a dependency list and search execution ordering (along with higher-level threading) managed and updated by the supervisor agent. In practice, this solution can drastically reduce latency, memory usage, and API call costs.

The architecture of such a Multi-Agent RAG system similar to the adjusted BM-25 pipeline we propose is shown in **figure 7**. This LangGraph architecture is inspired by James Briggs' LangGraph/Pinecone research agent, though it is adjusted to include a planner agent, Wikipedia search agents, and a supervisor for all the benefits in the aforementioned paragraphs (Briggs, 2024).

4 Experiments

4.1 Evaluation Metrics

To evaluate the performance of the proposed hop-prediction model and retrieval configurations, we use the following metrics:

- **Accuracy**: Accuracy measures proportion of correctly answered questions across different retrieval depths in a given sample, and it is determined by using the library sentence-transformers and computing the cosine similarity between embeddings with a threshold value. For cases of poor performance of the retriever, average cosine similarity is used in place of our accuracy percentage over the given test batch. This is done to allow us to continue to observe improvements towards accurate answers as hops are increased.
- **Efficiency**: Efficiency measures the computational resources required, measured by the number of retrieved documents added to context. Other factors were considered in devising a metric for efficiency, but since the assumption held that all retrieved documents contain at least the maximum number of tokens parsed, this level of granularity was negligible and we omitted considering token usage. We also considered adding latency as a factor, but since this retriever is inefficient and not intended for production use-cases, we figured that it was unnecessary to include.
  - **# of Docs in Context**: The number of retrieved documents added to context during the retrieval
  - **Trade-Off Analysis**: Visualizing accuracy improvements against efficiency (# of docs in context) to identify points of diminishing returns in retrieval depth.

4.2 Prediction Model Training

Prediction Accuracy of Predictor Model: Evaluating the hop-prediction model is done using metrics such as Mean Absolute Error (MAE) and R-squared (R^2). We use **Mean Squared Error (MSE)** as the loss function for weight calculation during backpropagation, and this guides the model to minimize the average squared differences between predicted and true values. The MSE is defined as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

For training the prediction model:

- Each "prompt" was vectorized using Term Frequency-Inverse Document Frequency (TFIDF).
- The corresponding `"query_count"` was used as the label for supervised training.

4.3 Experimental Setup

At a higher level, to systematically analyze the problem we aim to:

- Compare the performance of various LLMs ($M_1, M_2, \ldots$)
- Train an encoder model to predict the number of required hops for each question based on its content.
- Evaluate retrieval pipelines using $p, p+1, p+2, \ldots$ hops to assess accuracy-efficiency trade-offs and identify optimal configurations.
- Analyze how these trade-offs vary for different reasoning tasks and frontier models

In more detail, we perform the following:

(1) **Task-Agnostic Evaluation:**
This task-agnostic evaluation involves taking a random sample from the processed, condensed dataset and running the retriever over this random sample using the predicted value of hops from our predictor model (this is the input for the retriever). We set a random seed to ensure that the same sample is pulled from the data for each iteration.
We run this for 4 iterations, using $p, p+1, p+2$, and $p+3$ to vary retrieval depth. For each retrieval iteration, we calculate the following metrics:
  - Total number of queries,
  - Average number of retrieved documents,
  - Average number of iterations (for one retrieval),
  - Average answer similarity,
  - Accuracy rate over the sample.

These metrics are then used to perform the AEI trade-off evaluation.

(2) **Task-Specific Evaluation:**
The task-specific evaluation follow a similar process to the task-agnostic evaluation, but it is ran for 5 iterations and includes $p, p+1, p+2, p+3$, and $p+4$. Crucially, instead of randomly sampling from the whole dataset, we sample based on each reasoning type and perform the same evaluation process for each reasoning type present in the dataset. Using this data, we perform the AEI trade-off evaluation per reasoning type.

4.4 Trade Off Adjustment

The main goal of the experiment is to evaluate the AEI trade offs differ for reasoning tasks such as those described in the exploratory data analysis section (**2.1**). For cases in which accuracies are low, the vector similarity score is used in place of accuracy to allow for analysis in improvements *towards* accuracy as hops are increased by increments on the prediction.

For this case of low or zero accuracy, the modified AEI looks as follows:

$$\text{AEI}_{\text{sim}} = \frac{\text{Average similarity over batch}}{\text{Average \# of documents in context}}$$

The result of this metric is a similar trade off metric from the original AEI. As hop counts are incremented, even if accuracies remain at zero, we can use AEI_{sim} to observe improvements towards accuracy. This means that we can see improvements towards accuracy as the context grows, and we can plot this trade off similarly.

4.5 Compared Language Models

We will compare multiple LLMs on their ability to generate hops for multi-hop questions. We will specifically be using:

- Llama 3.2 (3B)
- Gemini 1.5 Pro

4.6 Results

Note that the sample size in the task-agnostic (54) evaluation differs from that of the task-specific evaluation (36). Notably, we used similarity scores for AEI calculations in numeric reasoning and post processing tasks (due to low and often 0 test accuracies).

The reason for there being different sample sizes is due to the method of balanced sampling that we utilized. For each test, we first built a balanced subset of the full dataset by adjusting the number of queries per ground truth hop count to be roughly equal. More precisely, we sampled around 20 questions for prompts that required 2 hops to answer, and repeated this process for all existing hop counts, up to 23 hops. It should be noted that not all of these hop counts actually contained 20 questions each, but this value allowed for us to build a large enough sample for testing. After sampling from the original dataset to be more balanced, we then filtered based on reasoning type. The dataset is also imbalanced in reasoning type, which is why there is a large range in the sample sizes for individual tests.

To measure accuracy, for each prompt, the generated answer provided by the LLM was transformed into an embedding using the Sentence Transformers library from Hugging Face. The specific transformer model was the "all-MiniLM-L6-v2". Each prompt's ground truth answer was also converted into an embedding. We then compared the cosine similarity between the ground truth embedding and the generated response embedding. We then determined if an accurate response was generated if the cosine similarity passed our threshold of 0.8. For similarity scores, we just stored the cosine similarity of the embeddings.

We presume the low accuracy for individual reasoning types are partially due to the lower, balanced sample size as well as strengths of the model in some areas over others.

4.7 Gemini 1.5 Pro

In our task-agnostic evaluation for Gemini 1.5 Pro, we achieved interesting results. Most notably, we found that the greatest accuracy of 0.148 was achieved when using the predicted hop count + 1 (p + 1). This was on a sample obtained using our previously described sampling method without filtering on reasoning type and with at least 5 queries per hop count.

Our task-specific evaluation yielded less accuracy across the board, but provided useful information through the collected AEI scores. Notably, different tasks have different optimal p values that maximize AEI.

Table 1 shows our results for the optimal prediction increment for various tasks using the Gemini 1.5 Pro model with the retriever.

(1) **Task-Agnostic Evaluation**

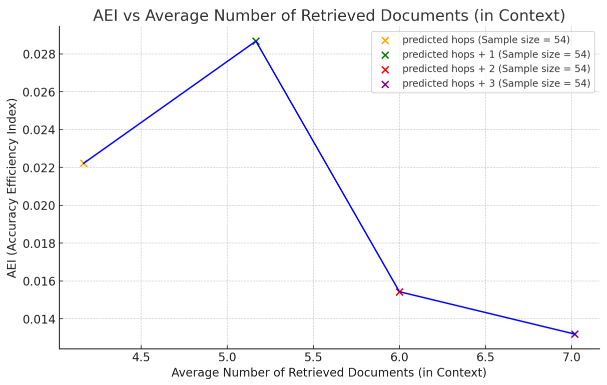

Figure 8

(2) **Task-Specific Evaluation That Use AEI (accuracy-based)**

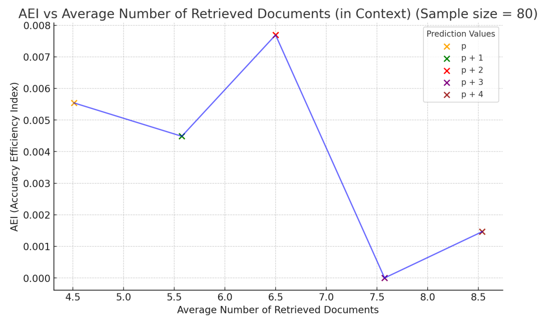

Figure 9: Multiple Constraints

(3) **Task-Specific Evaluation That Use AEI$_{\text{sim}}$**

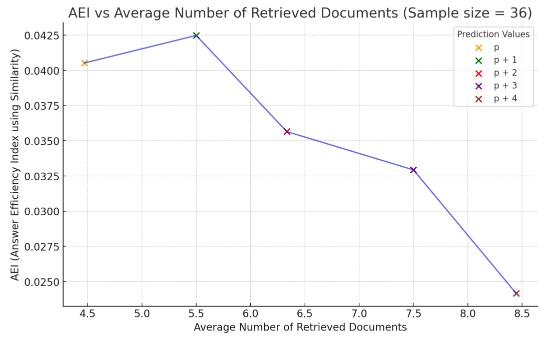

Figure 10: Numerical Reasoning.

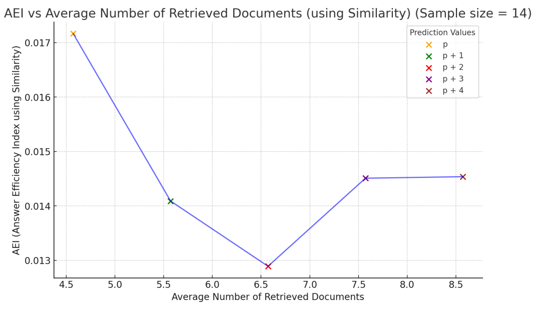

Figure 11: Post Processing

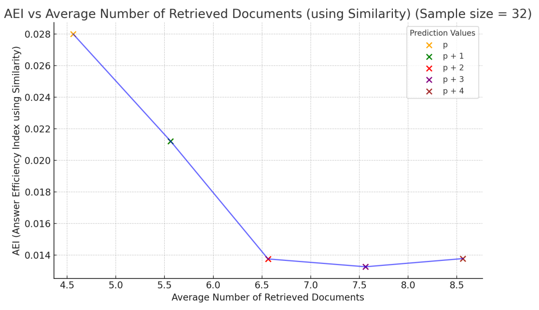

Figure 12: Tabular Reasoning

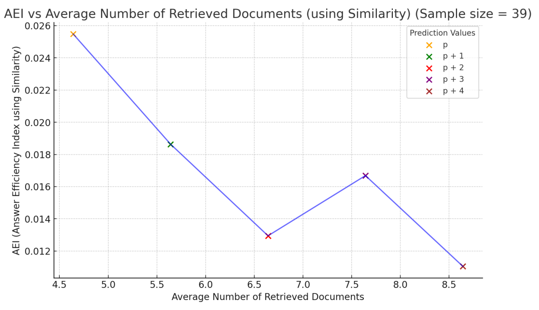

Figure 13: Temporal Reasoning

Table 1: Optimal Prediction Increment Analysis by Task

Using AEI		
Task	**Sample Size**	**Optimal Increment**
Task-Agnostic	54	$p + 1$
Multiple Constraints	80	$p + 2$
Using AEI$_{\text{sim}}$		
Task	**Sample Size**	**Optimal Increment**
Numerical Reasoning	36	$p + 1$
Post Processing	14	p
Tabular Reasoning	32	p
Temporal Reasoning	39	p

4.8 Llama 3.2 (3B)

For LLama 3.2 (3B), the results were dismal across the board, and that includes random samples of the entire condensed dataset as well as samples from specific reasoning types. Clearly, the retrieval task on the FRAMES dataset is difficult in general. The task was even difficult for Gemini 1.5 Pro for which we resorted to using similarity scores for an accuracy-efficiency metric when accuracies were low. We do not include visual results for the Llama model due to its incredibly poor performance and uninterpretable AEI plots.

The lack of successful performance for the Llama 3.2 model is likely a result of the models significantly smaller size of 3 billion parameters compared to the larger Gemini 1.5 Pro. Google does not disclose the parameter size of the Gemini 1.5 Pro model, but estimates vary and it is generally presumed to be much larger than 3 billion parameters (Lacy, 2024).

Additionally, we did not have the compute power or resources to run Llama 3.3 (70B), which would likely boast performance improvements from its significantly smaller counterpart.

5 Conclusion

This study formalizes the accuracy-efficiency trade-off in multi-hop RAG systems, providing a framework for cost-efficient retrieval configurations. By leveraging hop-prediction models and systematically analyzing performance drop-offs, the findings contribute to the topic of building scalable and practical enterprise RAG implementations by offering insights for optimizing LLM-based pipelines in various domains.

The introduction of the Accuracy Efficiency Index (AEI) provides a novel metric for balancing accuracy and efficiency, enabling a more granular analysis of retrieval strategies. Our experiments highlight the effectiveness of dynamically adjusting retrieval depth based on predicted hop counts.

Furthermore, the predictive hop count model's ability to achieve 91% accuracy (within a tolerance of ±2 hops) underscores its potential as a valuable tool for enhancing real-world RAG applications.

Multi-hop tasks in RAG remain a difficult challenge, and many recent studies explore improving performance in such tasks. Our retriever, built using the BM-25 algorithm and the wikipedia API, might not be as well suited to achieve high accuracy in such tasks as opposed to systems that might utilize multiple agents. With a more robust pipeline, we expect that we could have collected more compelling data to apply our AEI metric toward.

Despite our pipeline's limitations, we still had interesting findings. Our highest accuracy of 0.148 was achieved when using p + 1 hops on a random subset of the FRAMES dataset. This also was the peak of our AEI metric, suggesting that in general, using our predicted value of hops, plus 1, optimizes the trade-off between accuracy and efficiency.

At the reasoning type level, we found that most categories - Post Processing, Tabular Reasoning, and Temporal Reasoning - maximized their AEI_{sim} scores with a hop count of p. The only categories that did not fit this were Multiple Constraints, which maximized AEI at p + 2, and Numerical Reasoning, which maximized its AEI_{sim} at p + 1.

5.1 Limitations

There are a few key limitations of this paper, some of which carry more significance than others. One notable omission is the exclusion of exploring steps less than the prediction p (i.e. $p-1$, $p-2$). including and testing these sub-prediction level hop values could provide a better picture of the problem landscape.

Another very clear limitation in this study, and perhaps the largest, is the limited document size given the context windows we are working with. In our study, the context per document is limited by a maximum token capacity to prevent the context from expanding beyond the context window limits, and this restricts the systems ability to handle large-scale retrievals. Primarily, since we mostly end up restricting context to 10,000 tokens, we cut off anything beyond this token limit in Wikipedia articles, which often means we miss significant information and cannot effectively produce an answer for a sub-query. This is the biggest limitation of this paper and the biggest bottleneck for our accuracy results. While techniques such as document chunking and embedding could be deployed to reduce context size, time constraints prevented their implementation.

Issues in our accuracy calculations can originate from many sources. One possible issue is the contamination of the training data, meaning that some of the question-answer pairs may be included in the LLM's training data - leading to unreliable accuracies. Another possible issue is our use of cosine similarity to determine answer accuracy. Cosine similarity ensures with high-thresholds can ensure that predicted outputs align well with the ground truth, but it lacks in certainty. The auto-rating LLM based technique that is used in the FRAMES paper also lacks certainty in correctness, so we ultimately decided to use cosine similarity with a threshold for simplicity (Krishna et. al., 2024). Perhaps using BLEU scores, F-1 scores, or an exact match calculation (with better behaved answer outputs) would improve the validation of our results and should be considered for a longer-term study without time constraints.

Furthermore, the BM-25-based retrieval approach and query generation method may have structural limitations that hinder accuracy and flexibility - particularly with context growth and handling. This could potentially be solved with an entirely different approach, such as using a multi-agent system.

Finally, while the AEI metric is novel and offers valuable insight into the accuracy efficiency trade off, it could be extended to include token count for a more granular understanding of the trade-off. In our paper, we assume that, as was largely true in this experiment, that all documents include at least the maximum number of tokens (i.e. 10,000). An extension to include more granularity with token counts would improve the metric's ability to assess efficiency and scalability.

5.2 Future Work

Future work could explore expanding this approach to other datasets, incorporating adaptive retrieval mechanisms with real-time feedback, and testing across a wider range of LLMs to validate generalization. Ultimately, this research provides a pathway for developing scalable, cost-efficient, and high-performing RAG systems through the introduced hop prediction and AEI metric, paving the way for more practical and sustainable implementations in diverse applications.

Additionally, the use of multi-agent systems in AI-enabled research is already getting plenty of attention and has considerable future potential, especially in terms of planning and reasoning before sending out agents to divide and conquer separate tasks that comprise a larger problem. This approach would be a drastic improvement over the simple and clunky modified BM-25 system, which was primarily used for testing hop prediction and AEI metric but would be unsuitable for production due to inefficiencies and lack of scalability.

Further exploring dynamic coordination among agents and enhancing supervisor agents and agent networks can likely lead to significant improvements in performance on the FRAMES benchmark. These performance improvements can be seen not only in terms of accuracy but also for achieving more efficient and scalable solutions in terms of the AEI metric across the board. Thus, we can strive to create better systems that balance computational cost with performance and seek to optimize both.

6 Appendix

The code for our project and README can be found at the following repository:

https://github.com/Theod0reWu/Deep-Learning-Multi-Hop-QA

References

[1] Google. “Frames Benchmark: Multi-hop QA Dataset,” *arXiv preprint*, 2024. Available: https://arxiv.org/pdf/2409.12941
[2] Krishna et al. “Fact, Fetch, and Reason: A Unified Evaluation of Retrieval-Augmented Generation,” *arXiv preprint*, 2024. Available: https://arxiv.org/html/2409.12941
[3] Yu, Z., Liu, T., Wei, Z., et al. “RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMs,” *arXiv preprint*, 2023. Available: https://arxiv.org/abs/2407.02485
[4] Stephen E. Robertson; Steve Walker; Susan Jones; Micheline Hancock-Beaulieu Mike Gatford (November 1994). Okapi at TREC-3. Proceedings of the Third Text REtrieval Conference (TREC 1994). Gaithersburg, USA.
[5] MediaWiki. “API:Opensearch,” 2024. Available: https://www.mediawiki.org/wiki/API:Opensearch
[6] LangChain. “Planning Agents,” 2024. Available: https://blog.langchain.dev/planning-agents/
[7] LangChain. “Agent Supervisor: Create Tools,” 2024. Available: https://langchain-ai.github.io/langgraph/tutorials/multi_agent/agent_supervisor/#create-tools
[8] J. Briggs. “LangGraph Research Agent,” 2024. Available: https://www.pinecone.io/learn/langgraph-research-agent/
[9] 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, Dario Amodei, “Language Models are Few-Shot Learners,” *arXiv preprint*, 2020. Available: https://doi.org/10.48550/arXiv.2005.14165.
[10] L. Lacy, “GPT-4.0 and Gemini 1.5 Pro: How the New AI Models Compare,” *CNET*, May 25, 2024. Available: https://www.cnet.com/tech/services-and-software/gpt-4o-and-gemini-1-5-pro-how-the-new-ai-models-compare/
[11] Sparck Jones, K. (1972), "A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL", *Journal of Documentation*, Vol. 28 No. 1, pp. 11–21. Available: https://doi.org/10.1108/eb026526.