IMPROVING RAG FOR MULTI-HOP QUESTION ANSWERING USING GUIDED RETRIEVAL AND ADAPTATION

AALOK PATWA (APATWA@SEAS)
AKSHAT TALREJA (TALREJA@WHARTON)
TAHMID AHAMED (AHAMEDT@WHARTON)

ABSTRACT. Due to their unprecedented generalization capabilities, large language models (LM) pre-trained on large text corpora have become the foundation of modern systems. Despite these abilities, the models falter at tasks requiring specialized knowledge and can succumb to issues including hallucinations. Retrieval Augmented Generation has emerged as a powerful solution to this problem, by augmenting a LM with non-parametric, external memory. Despite RAG's rapid adoption in production settings, it remains limited particularly for multi-hop question answering tasks due to inherent difficulty in both retrieving relevant knowledge by discovering linkages, and reasoning for generation. Our goal is to investigate and develop lean and generalizable modifications to the retrieval system to boost performance for multi-hop QA. We build new LM-guided sequential retrieval techniques that yield an 10% improvement in generation performance on a limited evaluation set (n=100) due to improved retrieval. We also construct a lean, generalizable method of adapting off-the-self embeddings specifically for multi-hop QA using a single matrix, which yields 5% improvement in retrieval hit rate across multiple top-K values. Our results are a stepping stone for further work on optimized, dynamic retrieval techniques as opposed to the monolithic one-step approaches in use today.

1. INTRODUCTION

1.1. **Background.** Large language models have risen to the forefront of natural language processing, particularly due to their ability to generate accurate and precise answers to a wide array of questions. Typically, LLMs are pre-trained in a self-supervised fashion on a huge corpus of text data and fine-tuned to produce human-like text in subsequent steps. The scale of these models and the data fed to them throughout this process give them incredible generalization properties, making them suitable as a foundation for a range of language-based downstream applications.

Despite their abilities, LLMs have suffered from several issues, namely hallucination, static memory, and false recall. [3] To mitigate these issues, Retrieval Augmented Generation (RAG) has become a popular engineering solution. RAG allows an LLM to connect to an external knowledge base, extending the LLM's capabilities to answer nuanced questions on specific data. Typically, this is done by computing vector representations of the query and the documents in the knowledge base. At inference time, the LLM will use the similarity between the vector representation of the query and the documents to retrieve the most relevant documents from the external knowledge base (typically a vector store), and use these when answering the question [2].

A large proportion of downstream applications leveraging LMs in combination with retrieval deal with questions in a multi-hop setting, where multiple pieces of interconnected evidences need to be used almost in a sequential fashion to determine an answer. This still remains a substantial issue in industry as multi-hop settings add additional layers of complexity to both retrieval and generation. [4]

Static retrieval systems often fail to capture linkages that are important in multi-hop questioning. Beyond extraction, generation systems need to be adept at reasoning and dependency identification to be successful. Furthermore, current adaptations and additions to RAG pipelines that seek to improve retrieval performance often come with tradeoffs that in turn could sacrfice performance in a production environment. Tackling the aforementioned issues is crucial to scale LM-based applications in an industry setting and further to drive at the true world view of complete information democratization.

Existing literature has aimed to tackle the problem with multiple lenses, ranging from component specific adaptations within a RAG system to end to end fine tuning to purely inference time methods. While such approaches have yielded

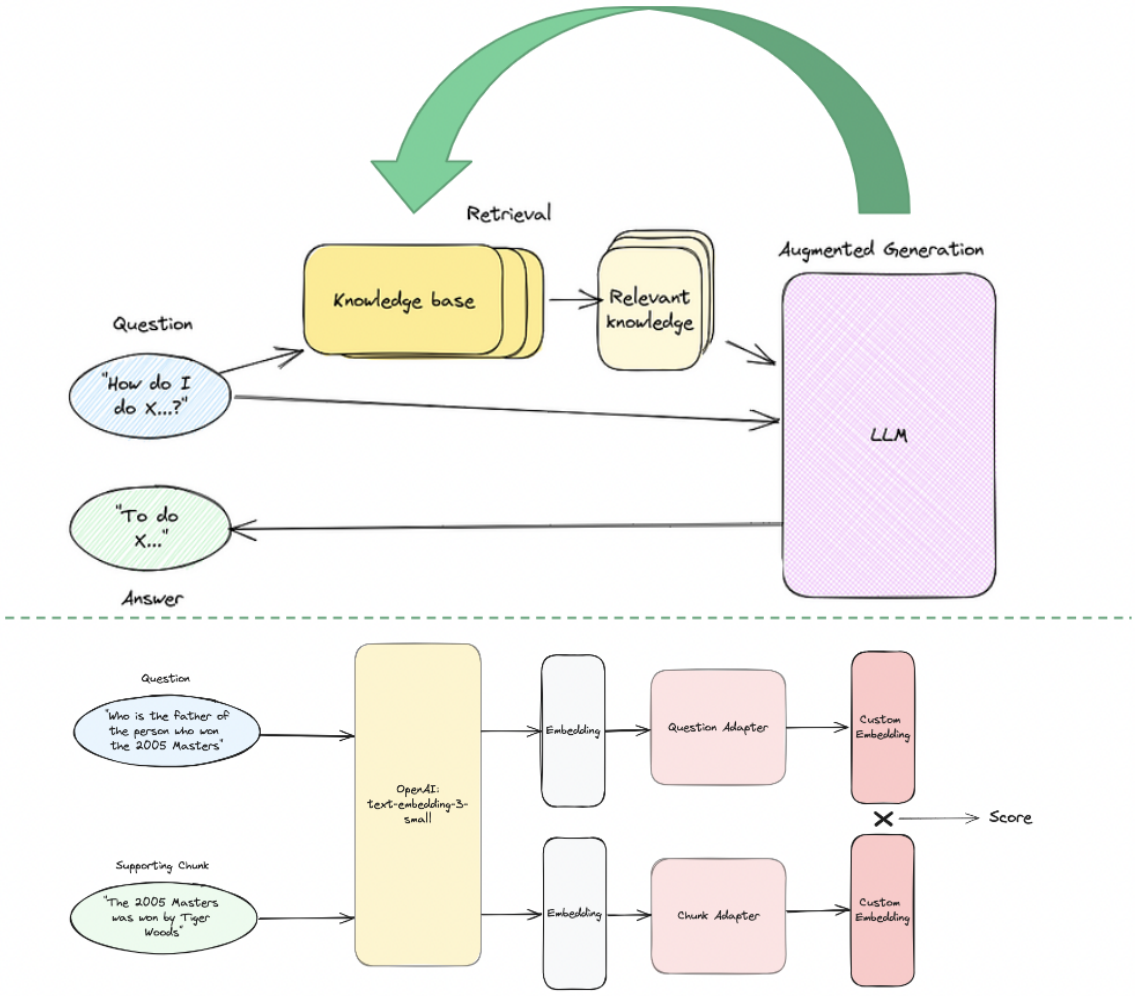

FIGURE 1. Lean retrieval adaptations to maximize MultiHopQA performance [1]

improvements, we posit that they are either high variance and lack domain transferability or are resource intensive.

In this paper, we aim to study lean adaptations to the RAG workflow that can drive performance improvements for multi-hop question answering tasks. Our goal is to demonstrate proof of concept for generalizable techniques that could potentially work for a variety of domains and tasks.

We focus on improving the retrieval of relevant chunks from the knowledge base. Given the correct context, out-of-the-box generator models are quite good at synthesizing and outputting the correct answer to a multi-hop question. But, existing retrieval pipelines that rely on cosine similarity between language model-generated embeddings of the query and context chunks fail to find the correct information for a certain question. If retrieval for multi-hop questions were perfect, the performance gains would be significant.

1.2. **Contributions.** Our core contribution from this paper is an exploration of lean strategies to modify or augment retrieval in a way that improves performance on multi-hop question answering tasks. In particular, we make the following contributions:

(1) A new sequential retrieval approach guided by the language model that improves both retrieval and downstream generation performance.

(2) A generalizable method for adapting off-the-shelf text embeddings to specific tasks/domains using contrastive learning that improves retrieval performance.
(3) An exploration into the applications of rerankers in Multi Hop QA, and how retrieval outcomes can be improved using smaller Cross Encoder models.

1.3. **Related Work.** Existing literature has explored RAG, particularly in the context of multi-hop question answering, in multiple ways. The first part of literature has focused on component specific evaluations, focusing on specific sub-systems, e.g. the generator or the retriever. Examples of this work include RAFT [10], which in part looks at tuning the generator to ignore irrelevant pieces of context. The second lens in literature focuses on end to end tuning, for instance in Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. [2] While not immediately linked to multi-hop QA, this approach has been popularized in industry (e.g., Contextual AI) and shown to provide benefits in multi-hop QA tasks such as HotPotQA. Finally, we see engineering focused approaches such as IRCoT [5], ReAct [8] which focus on prompting techniques, tool use, or other inference oriented methods to improve performance on multi-hop tasks.

Our core focus lies in the first and third lens of work. End to end approaches tend to be highly resource intensive and often either an overkill or inaccessible to smaller teams.

In optimizing individual sub-systems of RAG, we focus on improving the quality of the retriever by adapting embeddings taken from black-box models (such as OpenAI's). We choose this route because we hypothesize that significant improvements can be made, and because we see it as the most likely to succeed in the real world. We identify two shortcomings of existing embedding-finetuning methods. First, some approaches rely on fine-tuning the underlying large embedding model, i.e. billion-parameter language models. [6] We believe this is unnecessary computational workload and unlikely to be feasible for the whole host of tasks a user might need. Second, other approaches like MAFIN have mostly been applied to domain transfer, not task transfer. [9] Multi-hop QA, in particular, has not been solved by pure embedding-based adaptation yet.

As for engineering focused approaches, we posit that those such as IRCoT rely on techniques like in context learning and prompting making them fairly high variance and making their translateability more of a question. If we can build systems out of highly generalized reasoning abilities (e.g., question decomposition, entailment understanding), which themselves are something of a debate, then we can yield approaches that can easily be mapped out of the box to other domains.

To further elaborate on potential improvements to RAG from an engineering perspective, the inclusion of Rerankers within the RAG pipeline has been seen by many as a significant boon to the performance of semantically relevant documents. [1] However, the biggest drawback when it comes to the introduction of rerankers in RAG pipelines is the additional inference time overhead as a result of the models often being heavy weight transformer based Cross Encoders. Advances in Knowledge distillation however imply that it is possible to teach more lightweight / smaller models from larger teacher models to achieve similar performance despite having fewer weights / architectural complexities. [3] This presents a potential way of alleviating the performance overhead incurred by neural reranking models. Thus, we explore the possibility of distilling down knowledge in a large reranker models to smaller models in the hopes that it would be able to achieve equivalent or better performance than the large model in a MultiHopQA setting.

1.4. **Design.** Since we are looking at multi-hop question answering, we make use of the standard HotPotQA dataset [7] which includes Wikipedia passage based multi-hop questions. Each question has a set of 2 supporting evidences. More specifically, we use a subset of the database i.e. 28k context passages from the database for our vector store from which retrieval is performed. We further use 100 examples for manual evaluation to check generation performance. Of these 100 examples, 28 are classified as "easy" difficulty, 56 are "medium", and 16 are "hard."

To evaluate our approaches, we make use of 2 streams of metrics. To evaluate retrieval, we make use of a metric we call Hits @ K which is effectively whether or not we retrieve a relevant passage in K retrievals. We define a relaxed version of this metric as whether or not there is at least one relevant passage in the first K and a strict version which checks for whether all relevant passages are present. For our case, we primarily focus on the strict metric as the nature of the task is such that all relevant passages should ideally be retrieved for an answer, and it is this metric that we report in the paper going forward in our evaluations.

To ensure effective comparison, we build a Vanilla RAG pipeline that involves one-step retrieval from a vector store followed by generation, and use the performance of this to compare to our approaches.

2. Approach

2.1. **Guided Retrieval.** Static, one-shot retrieval systems are largely inept for multi-hop question answering tasks due their inability to consider linkages in paragraphs that are crucial for multi-hop tasks. Our aim in this section is to explore the use of retrieval approaches that make use of this nature of multi-hop tasks. More specifically, we see if the language model can be used to mimic human-like reasoning patterns and further guide the retriever to improve both retrieval and generation performance.

The core idea of our approaches is based on analyzing whether an LM can determine (1) when additional information is needed and (2) what additional information is needed to answer the question. The presence of these abilities, which we feel are somewhat analogous to planning and reasoning abilities, that have been a topic of much debate recently, would be a great stepping stone in building highly generalizable and transferrable retrieval techniques that can be translated across domains.

Building on this, we experiment the use of 2 main techniques:

(1) Naive Iteration: Prompt the LLM to determine additional information needed post retrieval. The output of this, framed as a follow-up question, is then used to guide further retrieval in a one-step setting. We then take 2 passages from the first round and 2 from the second as the updated context in the current one-step formulation
(2) Guided Iteration: Same as naive iteration but with LLM guided prompting on whether the pre-iteration answer is entailed by given evidence. This aims to assess if an LM can determine whether its current generation is correct or appropriately supported by the retrieved evidence (which is a variant of the textual entailment problem). It further aims to solve the "over-correction" issue (the observation that when prompted to correct, LMs often convert correct answers to wrong) that is often seen in self-correction literature and was observed by us empirically in Naive Iteration. At the end, we either take 2 passages from each of the first and second step is a claim is considered to be not validated by the LM based guider or just use the original context.
(3) Tree Retrieval: Construct new context passages through evidence chains created by iterative retrieval on each individual context retrieved from one-step retrieval; use the new passages for generation. This is an advanced version of naive iteration where we do sequential retrieval until each evidence chain is self-sufficient. For now, we implement this one-step for simplicity.

Our goal is to evaluate the aforementioned approaches with static, one-shot retrieval and thus, we compare each of them to a Vanilla RAG formulation, which uses one-step retrieval from a vector database as opposed to a more involved approach. We further ran experiments with a re-ranker to allow for comparisons with a stronger baseline, however, we did not improve performance significantly and thus, decided to not include it in the paper.

2.2. **Adapter-based Finetuning of Embeddings.** We hypothesize that static retrieval systems that use off-the-shelf embeddings from black-box models fail to capture the nuance required for multi-hop QA tasks. In a multi-hop setting, each query q has multiple "pieces" $p_1...p_i$ required to solve it. However, each piece individually might not seem to be answering the question. An example query is "What is the birth city of the player who won the 2018 US Open in Tennis?" The corresponding documents usually contain pieces of the answer, i.e. one document that says "Novak Djokovic won the 2018 US Open," and "Novak Djokovic was born in Belgrade." The latter document might not appear to directly answer the question, at first glance.

We experiment with taking off-the-shelf embeddings e_{bb} from a black-box model, OpenAI's text-embedding-3-small, and adapting them for MultiHopQA. We specifically pursue three different architectures for this adaptation:

(1) Linear Transformation: We apply a simple linear transformation to e_{bb}. Our adapter consists of two linear layers, one for adapting embeddings of queries and the other for adapting the embeddings of context chunks. Each linear layer has a $D \times D$ weight matrix W and D bias terms b. Then, the output embeddings for context become $W_c e_{bb} + b$ and the output embeddings for queries become $W_q e_{bb} + b$. At the start of training, weights are initialized to identity plus some noise while bias are initialized to zero plus some noise, to preserve most of the initial data in the embedding.

(2) Multi-layer perceptron adapters: Instead of just using a single linear layer to transform the chunk and query embeddings, we use an arbitrary number l of linear layers. We particularly test out the special case of $l = 2$, and a small hidden dimension, which essentially becomes similar to a low-rank adapter. We pass the base embeddings through this network and then add the result to the initial embeddings.
(3) Concatenation: We apply linear transformations for the chunk embeddings and query embeddings, but with a very small output dimension. Then, we concatenate this output to the original embedding. Call the concatenated embeddings $e_{q,conc}$ and $e_{q,conc}$. $e_{c,conc} \cdot e_{c,conc} = e_{q,bb} \cdot e_{c,bb} + e_{q,a} \cdot e_{c,a}$. That is, the dot product of the concatenated embeddings decompose into the dot product of the black-box embeddings and the output of the adapter.

In each case, we train the adapters using a similar contrastive loss:

$$L = M \sum_{i=1}^{B/2} s(e_i^q, e_i^{c,p}) - \sum_{j=1}^{B/2} s(e_j^q, e_j^{c,n})$$

where s is cosine similarity, B is batch size, M is a margin used to soften the loss, e_i^q are the queries in the batch, and $e_i^{c,p}$ and $e_i^{c,n}$ are the embeddings for positively-associated and negatively-associated chunks, respectively.

2.3. **Distilling Rerankers.** Rerankers have found great success in contributing to the improved performance of RAG systems. With their ability to further filter down comes the tradeoff of increased time and memory overhead. In this section, we aim to see if it is possible to train a more lightweight model that is able to rerank documents to the same or approximately the same degree as larger Reranker models. The core idea behind this lies in Knowledge Distillation, and how it can be applied to Reranker models in this metric learning task. We thus hypothesize that the inclusion of a distilled reranker model will help improve retrieval outcomes while simultaneously improving the time performance of the RAG system.

We particularly look at various Cross Encoder models as the student models. The core knowledge distillation setup is to utilize the relevance scores provided by Cohere's ReRanker API to establish a ground truth relevance score to each query document pair within HotPotQA. The student Cross Encoders would then be trained with these relevance scores, specifically in a setup that would include query - positive document and query - negative document pairs. Effectvely, we want the student models to learn to distinguish between relevant and irrelevant documents as well as the teacher Cohere ReReanker. The steps we take to explore this are as follows:

(1) Mine Positive and Negative query document pairs using Cohere Reranker relevance score as ground truth scoring metric. This was done by using Cohere's rerank API with default parameters (model being cohere-rerank-englishv3.0). Positive and Negative query document pairs are generated from HotPotQA by including supporting documents as positive relevance pairs and all contexts that do not include the supporting documents for a given query as negative relevance documents.
(2) Train student models using Cohere predicted relevance scores to distill knowledge using MSELoss.
(3) For each query, retrieve a set of documents from the vector databse, and compare the Cohere Reranker performance compared to the student models in reranking each of the retrieved documents for a given query.
    (a) Relevance Scores would be calculated using the training split of HotPotQA, and the student models would be trained on the training split query and document pairs.
    (b) Reranking performance would be evaluated based on ndcg scores (how well the student model is able to rerank a given set of documents for a query) as well as average time to execute reranking. Along with this, we would also evaluate the Retrieval Performance using Percent Hit, and Hits @ K values.

3. Experimental Results

3.1. **Perfect Retrieval Baseline.** To isolate the impact that better retrieval would have on performance, we evaluated the ability of GPT-3.5 to answer questions from HotpotQA when it was provided the exact correct context in the prompt. Results are shown in Table 1. Perfect retrieval improves performance considerably, leading to 20% improvement in overall accuracy, and 26.3% improvement in accuracy on medium-difficulty questions, which make up the bulk of the dataset. The gains on hard-difficulty questions are more modest, at 12.5%, but still considerable.

Method	Overall Accuracy	Medium-Difficulty Accuracy	Hard-Difficulty Accuracy
Baseline	93.0%	96.5%	75.0%
Vanilla RAG	73.0%	70.2%	62.5%

TABLE 1. Baseline generation performance on HotpotQA, as manually graded, when all context is provided.

3.2. **Guided Retrieval.** As mentioned in Section 2, we evaluate 3 main techniques: Naive Iteration, Guided Iteration, and Tree Retrieval and compare them against the vanilla RAG baseline across both retrieval and generation metrics.

Figure 2 shows retrieval performance (measured using the strict variant of the Hits@K metric) for each of the techniques under questions split by difficulty level (easy, medium, or hard). We make use of 4 total retrieved passages for this metric (note that this gets a bit muddied for tree retrieval which we will explian when looking at generation results). Overall, we see evidence for retrieval performance gains across each difficulty level. Majority of the gain is concentrated on medium difficulty questions, which are also the majority of the 100 examples used for manual evaluaton. We see that naive iteration leads to a worsening of performance compared to vanilla RAG for easy questions but slight improvements for medium and hard questions, indicating that directly prompting the LLM for what additional information it needs leads to something analogous to the "over-correction" problem in self-correction literature. Reflective iteration solves this issue in the case of easy samples where performance improves, however, it has no additional benefit for medium questions and worsens retrieval for hard questions, indicating the presence of a tradeoff. Nevertheless, our tree retrieval strategy of constructing chains of passages through LM guided retrieval leads to better retrieval across all difficulties, cementing our belief in the approach's adequacy for the task at hand.

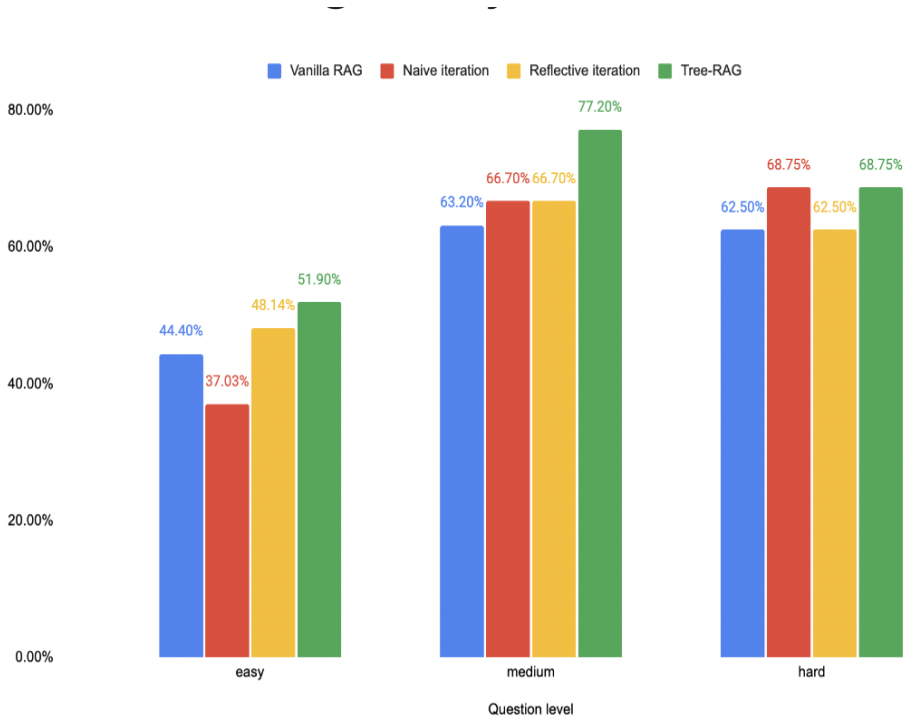

FIGURE 2. Retrieval Performance by question level using proposed dynamic retrieval strategies

Figure 3 shows the generation performance using the passages retrieved based on each of the three approaches. Once again, we see promising results as generation performance largely improves under our given approaches, with the overall best performance achieved by the tree retrieval strategy. We see most of the gains in medium and hard questions, which is intuitive as these questions are largely those that rely more on multi passage inference. Overall, our performance improves from 72% overall in the case of vanilla RAG to 82% in the case of the tree retrieval formulation, which we consider to be a fairly good improvement. One aside in this analysis is the fairness of this comparison as while for each of the other approaches, 4 retrievals are used, tree RAG uses 3 new context passages but each are

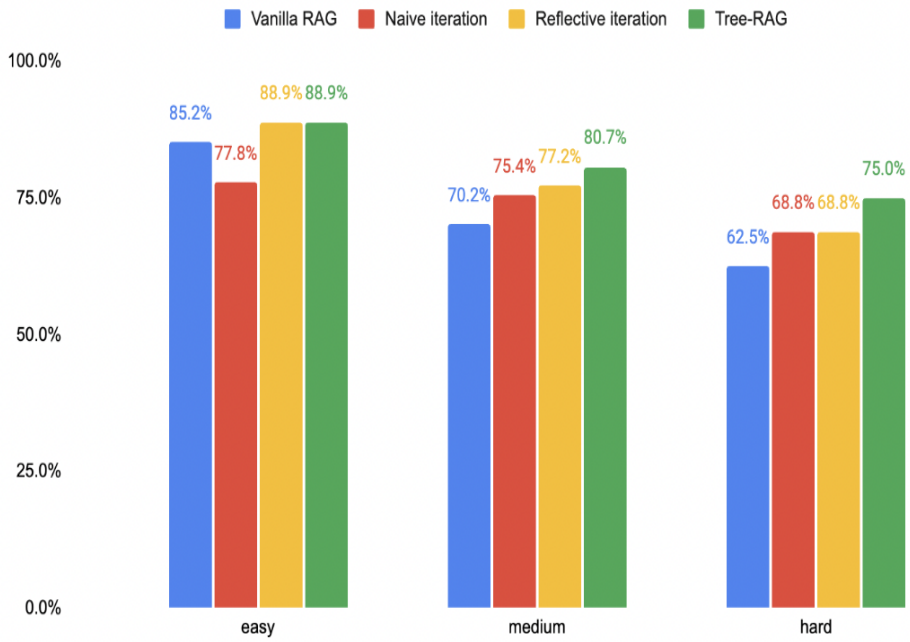

FIGURE 3. Generation Performance by question level using proposed dynamic retrieval strategies

constructed using 3 passages themselves. To study this briefly, we ran vanilla RAG with 10 retrievals and only obtained a performance of 76%, still preserving the improvements in tree retrieval that we posit happen due to the inductive bias of reasoning based retrieval. Naturally, such comparison can be extended to the guided and naive iteration, but tree retrieval is anyways an extension of those or a more calculated approach to navigating multiple contexts, and thus, we feel like that comparison is not imperative as the core takeaway from this is that the inclusion of reasoning patterns in retrieval can drive retrieval and generation performance. The aforementioned over-correction issue is once again seen within easy questions. Ultimately, the results show evidence that more guided and dynamic retrieval strategies make for systems more optimized for multi-hop QA

3.3. **Adapter Finetuning of Embeddings.** We created a dataset using 3,000 ”hard” queries from HotpotQA. For each query, we obtained the two Wikipedia articles used for answering the question and two Wikipedia articles that contained tangentially relevant information, or ”distractors.” We treated the relevant Wikipedia articles as positively associated with the queries, and the distractors as negatively associated. We additionally generated one additional synthetic negative per query by randomly assigning it a Wikipedia article used to answer a completely different question. This gave us 3,000 queries and 15,000 pieces of context in our dataset. We obtained embeddings for the queries and context chunks using OpenAI’s text-embedding-3-small, and then trained networks to produce adapted embeddings that were more distinct between positive and negative examples.

3.3.1. *Metrics for Separation.* We used a few custom metrics for assessing the degree to which the embeddings for positive and negative examples (i.e. relevant and not relevant context chunks) were separated.

(1) Prediction accuracy using cosine similarity. This metric assesses how well the cosine similarity between a query and a chunk can predict whether they are positively or negatively associated. We would hope this metric to be 1.0, i.e. there was perfect separation.
(2) Correlation coefficient. We would simply calculate Pearson’s r between the positive/negative label and the cosine similarity of query/chunk pairs. The closer this is to 1, the better.

3.3.2. *Baseline Separation.* We computed the above metrics for the OpenAI embeddings to create a baseline to compare our adaptations to. We found there was already some separation, but it was not perfect (Figure 4). We obtained a baseline prediction accuracy of 80.9%, and an $r = 0.636$.

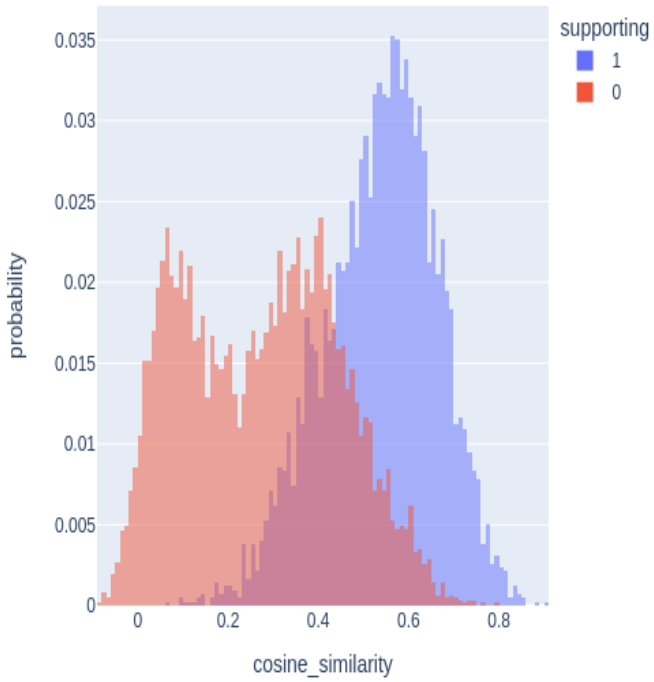

FIGURE 4. Separation between embeddings of positive and negative examples using base OpenAI Embeddings. Accuracy=80.5%, $r = 0.643$

3.3.3. *Linear Transformation.* We trained the linear transformation model for 70 epochs. During training, the biggest issue we faced was overfitting. To achieve good performance, we had to add a regularization term equivalent to the L2 norm of the adapted embeddings minus the original OpenAI embeddings. Essentially, this regularization penalizes large changes during the transformation, adding more stability.

This model led to the best results out of the three approaches. We were able to combat overfitting and obtain consistent reductions in the contrastive loss of the embeddings (Figure 5). We were able to achieve improved separation between positive and negative examples on the test set, leading to a test accuracy of 83.0% and $r = 0.664$. After training, we applied the linear transformation to the entire set of chunks in the dataset and the queries in the test set to see how the retrieval performance changed. We noticed that the adapted embeddings led to a much higher strict hit rate, meaning that more often, both of the chunks necessary to answer queries were present in the retrieved context (Figure 6). We see gains of as much as 8% on K=3 and K=4, with modest gains of at least 5% for all tested Ks.

3.3.4. *MLPs.* We experimented with several different MLP adapter architectures, varying both the number of hidden layers and the hidden dimension. The results showed that anything more than 3 layers was overkill, and even 3 layers led to some overfitting. Overall, we achieved the best results with a simple 2-layer MLP with a hidden dimension of 256. This is similar to the low-rank adaptation used to fine-tune large language models.

However, the results were still not excellent. We found it difficult to get the contrastive loss to decrease on the test, which points to persistent overfitting. We were not able to improve the level of separation on the test set, only mustering a test accuracy of 79.8% and $r = 0.631$. When we tested the downstream retrieval, we saw that the performance had actually degraded.

3.3.5. *Concatenation.* We trained the concatenator model for 60 epochs using a learning rate scheduler that starting at $lr = 0.0005$ and decreased by a factor of 0.95 every epoch. We experimented with different sizes of the augmented embedding, from 32 to 1536, the original dimension of the OpenAI embeddings. However, the results were not good in any case. We suffered from consistent overfitting issues from this model, even more than compared to the other

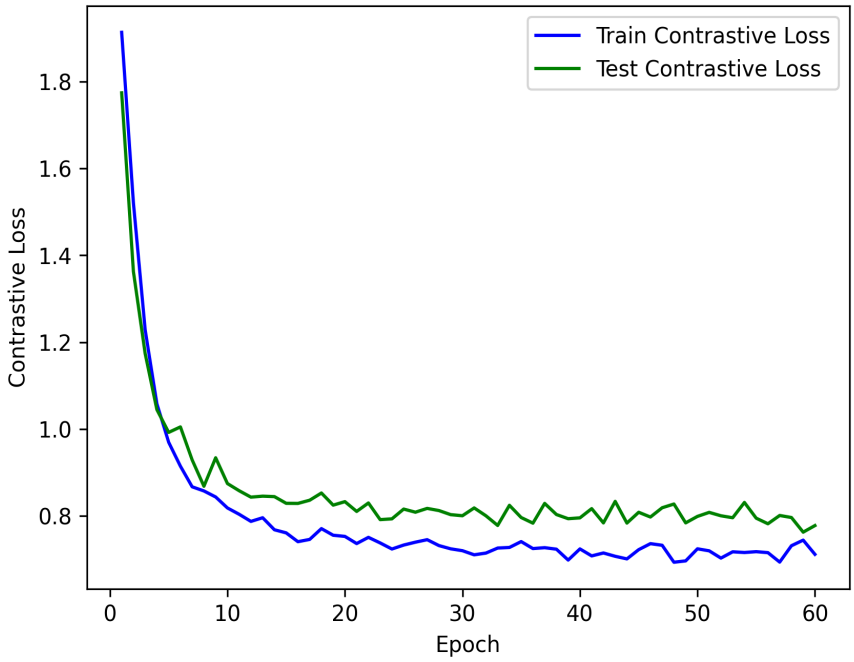

FIGURE 5. Training and Test Loss of Linear Transformation

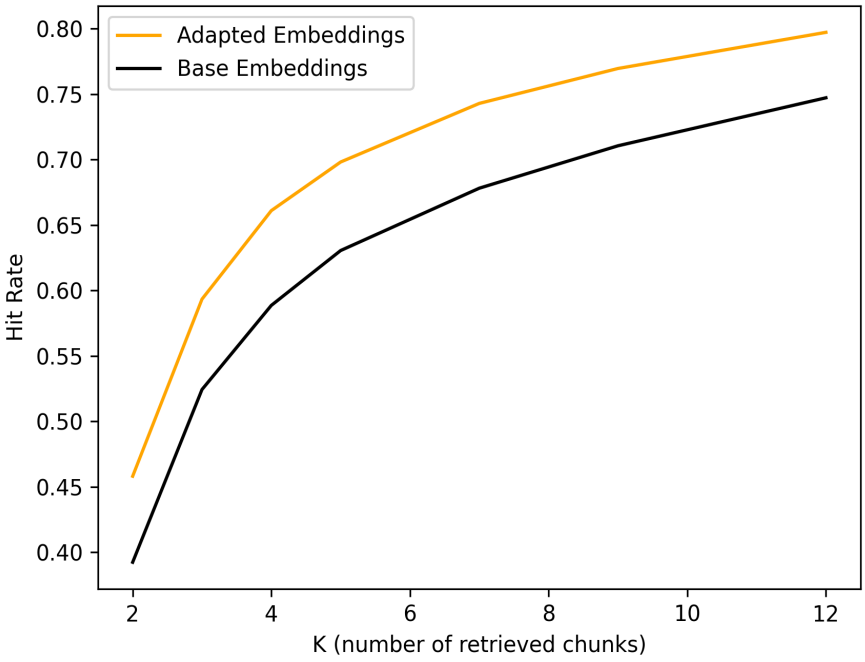

FIGURE 6. Change in Strict Hit Rate between Base and Adapted Embeddings

models (Figure 7). The separation between positive and negative examples collapsed on the test set, resulting in a poor prediction accuracy of only 71.5% and a correlation coefficient $r = 0.558$. We did not attempt to calculate the downstream retrieval performance given these poor results.

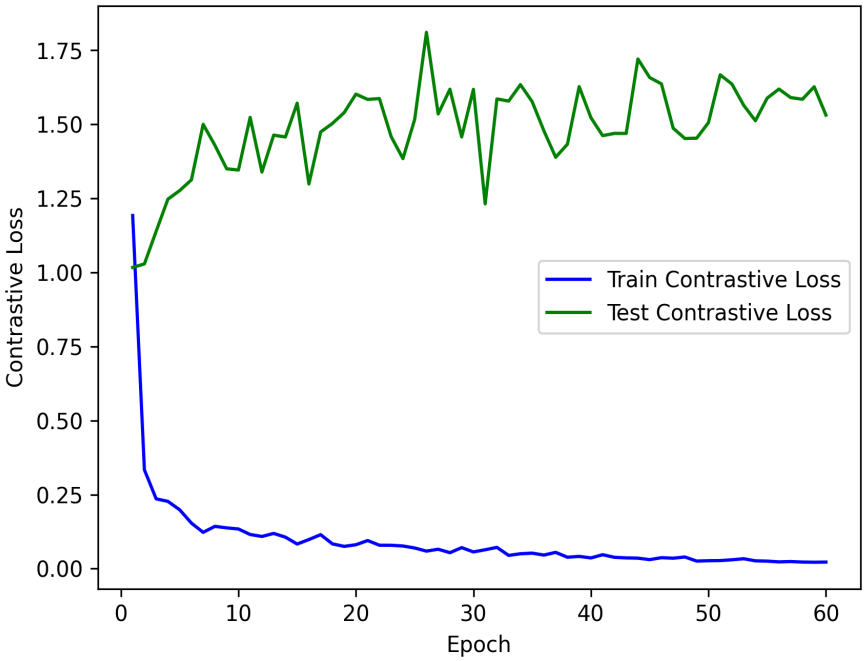

FIGURE 7. Training and Test Loss of Concatenator Model

3.4. **Distilling Rerankers.** For this experiment, we mainly evaluate 3 rerankers that serve as student models that learn from the relevance scores predicted by the Cohere reranker. These three models are the ms-marco-TinyBERT-L-2 model (TinyBERT), ms-marco-MiniLM-L-6-v2 (MiniLM) and MiniLM-L12-H384-uncased (MiniLM - uncased). The Baseline of comparison for these models would simply be the Cohere Reranker model. The baseline performance of this model has been included in the results where applicable (NGDC scores for the baseline reranker would simply be 1 in this case since we are using the relevance scores calculated by the Cohere Reranker as the prediction labels for each query - document pair). We also include metrics for our Hits @ K metrics, MRR, and Percentage Hit. Note that for our test split, we evaluate average NDCG scores by sampling 500 queries along with their corresponding set of contexts from the HotPotQA dataset. Average Time is simply the amount of time taken to retrieve the documents from the vector store and to rerank them with each model, which was averaged over the number of total queries in the test set. There were apporixmately 7400 queries within the test set, each with their own set of contexts.

Metric	Value
Hits @ K [relaxed]	97%
Hits @ K [strict]	63%
Perc. hit	80%
MRR @ K	95.5%

FIGURE 8. Baseline RAG Performance with Reranker

Analyzing the results shown above, we see a few things. For the training split of the HotPotQA data, there seems to be very high performance based on the NDGC scores for all three student models, which indicate that their reranking capabilities closely match the reranking capabilities of the Cohere Reranker for a given query and set of contexts. However, on the test split, we see a degradation in average Reranking performance, going from nearly 0.98-0.99 to

Metric	Value
Hits @ K [relaxed]	97.7%
Hits @ K [strict]	60.7%
Perc. hit	97%
MRR @ K	91%

FIGURE 9. TinyBERT RAG Performance

Metric	Value
Hits @ K [relaxed]	95.7%
Hits @ K [strict]	59.2%
Perc. hit	81%
MRR @ K	91%

FIGURE 10. MiniLM RAG Performance

Metric	Value
Hits @ K [relaxed]	96.3%
Hits @ K [strict]	61.2%
Perc. hit	85.3%
MRR @ K	94%

FIGURE 11. MiniLM uncased RAG Performance

Model	Average Time (s)
cohere-english-v2.0	147 +/- 6.56
cross-encoder/ms-marco-TinyBERT-L-2	133 +/- 5.32
cross-encoder/ms-marco-MiniLM-L-6-v2	171 +/- 11
microsoft/MiniLM-L12-H384-uncased	184 +/- 10.2

FIGURE 12. Average Time for RAG Pipeline by Model for Test Split

0.71. Given the training schema used as well as how these average scores were calculated, what this could imply is that the student models reranking capabilties underperform the teacher's on the test split which is to be expected

Model	Average NDCG Score
cross-encoder/ms-marco-TinyBERT-L-2	0.9970413
cross-encoder/ms-marco-MiniLM-L-6-v2	0.9982514
microsoft/MiniLM-L12-H384-uncased	0.9819677

FIGURE 13. Average NGDC Scores for Train Split

Model	Average Test NDCG Score
cross-encoder/ms-marco-TinyBERT-L-2	0.7127
cross-encoder/ms-marco-MiniLM-L-6-v2	0.7130
microsoft/MiniLM-L12-H384-uncased	0.70982

FIGURE 14. Average NGDC Scores for Test Split

given the smaller size and complexities within each of the student models. Now, taking a look at the Average Time for RAG on the test split, we notice a few interesting things. The teacher model actually ended up having haveing a lower average time in the RAG pipeline compared to all models except the ms-marco-TinyBERT model. Given the fact that we were further using the Cohere API, this time realistically could be even smaller if we discount the time for API communications. This could imply that the Cohere model we used as a teacher model simply is more efficient by the architecture it uses in reranking, and thus is able to rerank much faster on average compared to the other models. Now, evaluating based on the various Hit @ K and MRR metrics for each of the models compared to the Baseline, we see that while having comparable performance with the Hits @ K [relaxed] metric, each model does underpeform the baseline when it comes to the Hits @ K [strict] metric. There are varying degrees of success with the Percent Hit metrics, with TinyBert surprisingly having the highest percent hit among the three models.

4. CONCLUSION

In this research, we have implemented and evaluated several different approaches from improving the performance of retrieval-augmented generation on multi-hop question answering tasks. Specifically, we have improved the performance of retrieval, which is the source of most errors / hallucinations.

Within guided retrieval, we observe that our system yields better retrieval and generation performance, showcasing potential for a more dynamic approach than one-step retrieval. Another benefit of our system we posit is the fact that it relies on generalized guiding surrounding the LMs ability to identify gaps in the current evidence to guide future retrieval. We hypothesize that this is more robust than prompting oriented sequential retrieval techniques such as IRCoT as they can be high variance and further need to be customized for each domain.

We obtained impressive results using single linear transformations to project off-the-shelf embeddings of queries and chunks into a new space. We achieved better separation between the cosine similarities of relevant and irrelevant context, leading to improvements of about 5-8% in retrieval hit-rate. This is a testament to the fact that off-the-shelf embeddings from black box models might not properly tie questions to their answers, especially for multi-hop tasks. Therefore, adapting those embeddings into a new space with lightweight networks is a viable strategy for improving performance of RAG systems. We saw the most success from directly transforming the base embeddings with a single

linear layer, rather than using MLPs or concatenating extra information to the base embeddings.

The results of the Reranker distillation experiments give us a bit of insight into the potential improvements we can make to RAG pipelines using smaller reranker models. In particular, it would suggest that models that are smaller (such as in the case of the TinyBERT Cross Encoder) could allow for faster Reranking times without sacrificing retrieval performance. In the context of MultiHopQA, this could mean a major improvement in the use of rerankers for datasets like HotPotQA, or any instance of a Retrieval task that requires information from multiple, potentially large documents. The results also suggest that there is potential for improvement in the retrieval time and retrieval accuracy based on the type of model architecture chosen for the reranker. Exploring different types of Cross Encoders, or even other types of Transformer models could assist in making improvements in Reranking.

4.1. **Limitations.** With guided retrieval, the core limitation of our analysis lies in the difficulty of evaluating. The approach at its core suffers from the difficulty associated with any kind of anthropomorphizing. The two core ideas leveraged here are an LMs understanding of some kind of entailment (i.e. when we prompt it to determine whether a piece of context is enough to imply the answer as in the case of the guided retrieval approach) or of what additional information is needed. The reality is that even if an LM doesn't have these abilities, even some wrong answers can lead to textual patterns that lead to better retrieval. This is almost a confounding reason for improved performance. More robust evaluation of the base system's capabilities is necessary to truly understand whether this system is truly robustly scalable.

The evaluation at the moment is further limited due to the need to do manual evaluation. In addition, the choice of dataset is further an issue as while HotPotQA is a popular choice, it does not precisely mimic the nature of data or the nature of questions in a real-world setting.

Our embedding adaption experiments require additional research to confirm the feasibility of these approaches. Throughout this project, we ran into significant training stability and overfitting issues, which could be the source of our poor performance. More research needs to be conducted to see what performance gains can be achieved if overfitting is controlled.

Regarding the Reranker experiments, one limitation that could have impacted the performance of each of the student models on this dataset is the dependence on the pure relevance scores. Given the fact that we did not have access to Cohere's model itself and rather only it's API endpoint that provides the scoring and reranking capabilities, the distillation of the internal representation within Cohere's models into our student Cross Encoders is much more difficult. Being able to access this internal representation could have increased the performance of the student models on retrieval. Another limitation were the types of models tested. While we primarily looked into knowledge distillation for Cross Encoder architectures, which is a common architecture for Rerankers, we initially hypothesized that training even simpler models such as Feedforward models could potentially be used to achieve similar performance. However, we encountered significant training instabilities with these types of models, often prodcuing non sensical scores that would not yield any significant performance gains. More research and effort would have to be done to explore non transformer models that can potentially be used as Neural Rerankers.

4.2. **Next steps.** A few key next steps emerge for the guided retrieval line of approaches. At first, we hope to develop robust evaluation standards to test the base characteristics used to construct the overall strategies (entailment, idea for additional information, etc.). We further hope to scale evaluation not just with HotPotQA, but also with real-world messier data to investigate the out-of-the-box transferrability of this approach to other domains. Naturally, the final step would be to productionize and ship a stable version of this that can be used out of the box but we leave that as an implementation concern as opposed to one that we are focused on as researchers and investigators.

For the other experiments, we hope to apply our approaches to a different domain/task to assess whether the techniques are truly generalizable. We also hope to experiment with different loss functions and forms of regularization to optimize the training process. Furthermore, we would also like to explore the impacts of different architecture choices as well as combinations of different architectures and obtain a better understand what can be used to increase performance in RAG pipelines.

REFERENCES

[1] Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. Few-shot reranking for multi-hop qa via language model prompting, 2023.
[2] 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.
[3] Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. The dawn after the dark: An empirical study on factuality hallucination in large language models, 2024.
[4] Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. A survey on multi-hop question answering and generation, 2022.
[5] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions, 2023.
[6] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models, 2023.
[7] 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.
[8] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.
[9] Mingtian Zhang, Shawn Lan, Peter Hayes, and David Barber. Mafin: Enhancing black-box embeddings with model augmented fine-tuning, 2024.
[10] 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.

5. CODE

Code can be found at https://github.com/at3001/ImprovingRetrievalMultiHopQA.