CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Real-time Relevance

RAG with Dynamic Context for Improved Natural Language Responses

Master's thesis in Computer science and engineering

Malte Landgren
Oskar Giljegård

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Real-time Relevance

RAG with Dynamic Context for Improved Natural Language Responses

Malte Landgren

Oskar Giljegård

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Real-time Relevance
RAG with Dynamic Context for Improved Natural Language Responses
Malte Landgren
Oskar Giljegård

Supervisor: Richard Johansson, CSE
Advisor: Mathias Andreasson, CPAC Systems
Examiner: Richard Johansson, CSE

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

Real-time Relevance
RAG with Dynamic Context for Improved Natural Language Responses
Malte Landgren
Oskar Giljegård
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Today's Retrieval Augmented Generation (RAG) systems often struggle when trying to answer questions that require complex multi-hop reasoning. In this thesis we investigate an autoregressive Large Language Model (LLM) architecture which can generate a real-time relevant dense search vector for every token generation step. To facilitate this we also develop a synthetic data generation technique to acquire search query vector labels on a token-by-token level, requiring only a generating LLM and a document database. We investigate the quality of the synthetic data, and provide an attention based relabeling method which decreases hallucinations, improving the correctness of the labels by 67%. The architecture is able to produce query vectors 27 times faster than a separate embedder at the cost of retrieval accuracy. Finally, we train and employ the model in an active retrieval question-answering setting.

Keywords: LLM, RAG, active retrieval, synthetic data generation, master thesis.

Acknowledgements

We would like to thank thank our academic supervisor Richard Johansson for his encouragement, valuable insights and detailed feedback throughout the project. We would also like to thank everyone at CPAC systems for letting us do this thesis under their supervision and providing both computational resources and important insights into real world use-cases of this technology. Special thanks to Mathias Andreasson and Peter Forsberg at CPAC for their continued guidance and support.

Malte Landgren & Oskar Giljegård, Gothenburg, 2024-06-10

Acronyms

AL Attention-Labeled.

AR Autoregressive.

BPE Byte Pair Encoding.

CoT Chain of Thought.

HAL H trained on AL.

HPL H trained on PL.

LLM Large Language Model.

LRU Least Recently Used.

MCRD Mean Correctly Retrieved Documents.

MLP Multi-Layer Perceptron.

NLP Natural Language Processing.

PL Prompt-Labeled.

RAG Retrieval Augmented Generation.

RNN Recurrent Neural Network.

t-SNE t-distributed Stochastic Neighbor Embedding.

Contents

List of Figures

List of Tables

1

Introduction

Large Language Models (LLMs) leverage their extensive number of parameters to not only model language, but also store knowledge [1]. However, memory stored in the models parameters (parametric knowledge) is not always sufficient to recall facts and can therefore cause "hallucinations" [2]. One research area attempting to solve this is Retrieval Augmented Generation (RAG), which lets the generating LLM ground its statements in relevant documents retrieved from a database through a retriever model [3]. Similarly to how humans can ground their statements in references, this aims to let LLMs do the same.

1.1 RAG architectures

The typical approach for a RAG system is to leverage three models (see Figure 1.1). Firstly, a document embedder model (typically a bidirectional encoder such as BERT [4–6]) embeds the documents into a representative vector. These vector embeddings act as semantically consistent keys in a searchable vector database, allowing you to retrieve similar text by nearest neighbor search. Secondly, a retriever model is trained to similarly embed the query, for instance "When was Hemingway born?", to match with the document where relevant information is contained. The query vector is used to seach a vector database, retrieving similar documents. Lastly, the query and the relevant documents are given as context to a generating LLM, that formulates an answer.

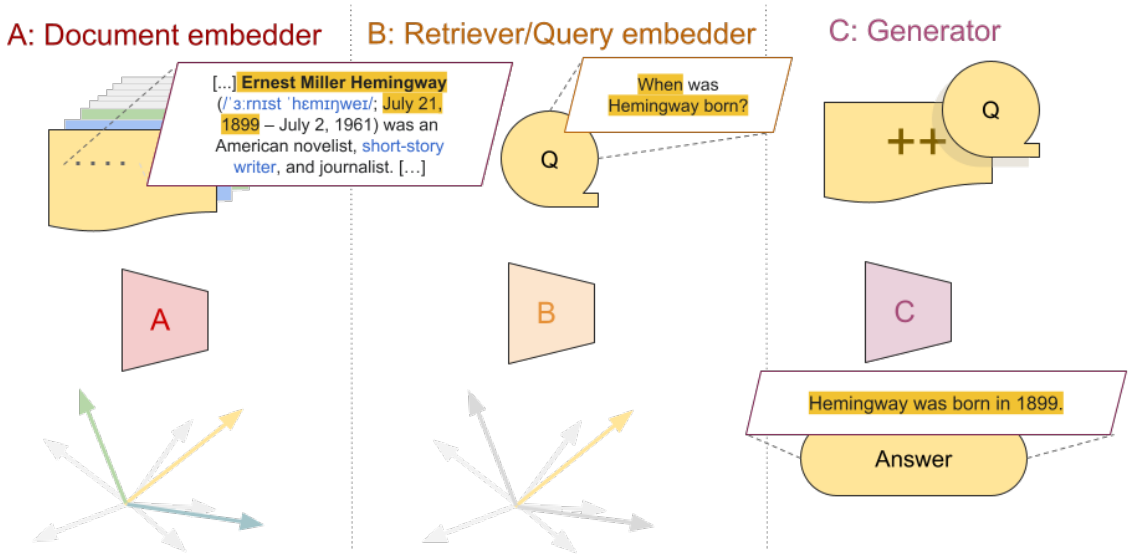


Figure 1.1: A general overview of a RAG system. **A:** Embeds the document into a representative dense vector embedding. **B:** Embeds a query into a representative embedding, trained to match with embeddings from **A** of supporting documents. **C:** The generating LLM is given both the question and relevant documents to ground its answer.

In RAG systems, difficulties arise when the question is multifaceted [7]. Furthermore, single searches may struggle when the task is exploratory; e.g. "Considering the current geopolitical landscape in europe, what party might benefit the most (politically) from increased inflation?" is a question that is hard to find a single query for and may require reflection throughout. Questions like these that require information from multiple different sources to answer are known as **multi-hop** questions [8] (see more in Section 3.3.1), and are a focus of this report. To address these issues and help answer multi-hop questions, we propose a system that is continually supplied with locally relevant information from locally relevant queries, at each generation step. These systems are considered active [9], or adaptive [10] retrieval systems.

Furthermore, in a traditional RAG system we note that the query embedder and generator model both perform some work related to extracting the semantic content of the question. This is a source of computational inefficiencies since this work is performed separately in each model, and incurs extra latency in the final system [10, 11]. This problem is further exacerbated in active retrieval systems since retrieval is performed many times over the course of the answering process.

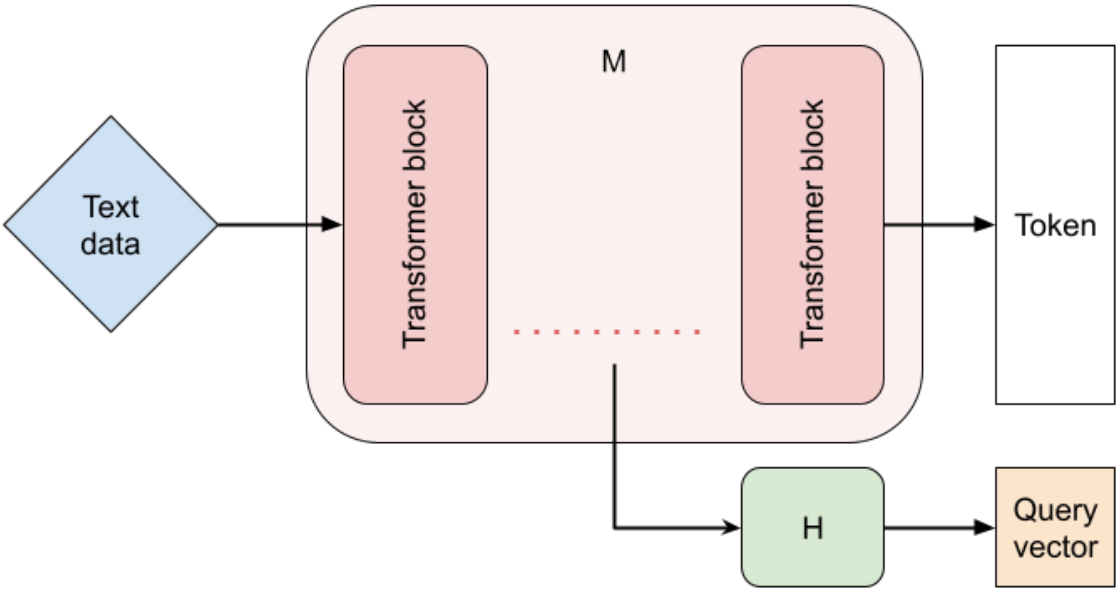


Figure 1.2: Overview of the proposed architecture. Intermediary features from the larger generator model M are fed as input into a side network H that predicts a query vector.

Our proposed architecture (see Figure 1.2) combines the query understanding and answer generation processes into one combined unit. The side network (H), which we introduce, operates as a new head to the larger autoregressive generator model (M), allowing for real-time relevance of the query vector. This replaces the traditional embedding model as a query prediction mechanism. At each token generation step, this side network utilizes intermediate features from the generator to predict a query vector that corresponds to the evolving context of the generated text. The predicted query vector is then used to retrieve the most currently relevant documents, which are fed back into the generator to inform the subsequent token generation. This creates a feedback loop where the generator is constantly informed by up-to-date, relevant information.

1.2 Synthetic data

The dataset is a central part of training a machine learning system [12]. In the case of our document retriever system we need a set of questions Q linked to a set of documents D. Since our model is going to search while generating text we allow it to first reason about the question before giving a direct answer. This is done by generating a series of Chain of Thought (CoT) steps before creating a final answer [13]. Therefore, our dataset also needs CoT steps which are labeled with the relevant document at that time. Furthermore, we want to specifically analyze multi-hop questions, meaning that the questions in our dataset should require more than one document to answer.

A popular RAG use-case for companies is to answer questions using a proprietary corpus of documents D [14]. Manually annotating data is difficult and expensive [15]. Therefore, we want to avoid manually creating Q with CoT steps for D. There does exist datasets like HotpotQA [16] which contain multi-hop questions with answers for a set of documents, but it lacks associated CoT steps. Using a pre-existing dataset with questions also removes the ability to pick D ourselves, leading to less transferable results. Therefore, we will investigate techniques to synthetically create multi-hop questions, answers, and CoT steps using an LLM. The CoT steps will be automatically labeled with references, ensuring that a system can be trained to retrieve locally relevant information.

1.3 Research Questions

Our work is centered on investigating the following two questions:

1. **Can the query prediction mechanism (H) be combined with the autoregressive generator model M to reduce the memory usage and improve inference speed?** Compared to a separate query prediction network, this method could be especially useful in the active retrieval use-case where recalculations otherwise need to be done at every token position for each query vector inference.
2. **Can multi-hop questions with answers and reference-labeled CoT reasoning be cheaply and effectively generated synthetically using an LLM?** In that case, the datasets required to train multi-hop question answering systems do not have to be manually constructed but can instead be generated based on a document database.

1.4 Limitations

The goal of this study is to investigate how a query prediction model H can utilize intermediary representations from an existing LLM. Therefore, only H will be trained, not the generating LLM. Training both H and the LLM could lead to higher accuracy [17], however it would require more computational resources. Furthermore, even though the choice of LLM could influence the retrieval accuracy, only one model will be analyzed. For the synthetic data, the goal is to create a pipeline that is applicable to any set of documents. Therefore, there will be less focus on constructing a flawless set of documents, in order to prioritize the multi-hop question generation.

2

Related Work

There have been many previous papers about RAG and about synthetic data generation. The following chapter will give an overview of some of the most relevant works, as well as detail both similarities and differences to our approach.

2.1 Active Retrieval

The RETRO model used a frozen BERT model as the retriever to fetch small (128 token) documents [6]. These were embedded using a transformer encoder and integrated with cross attention to a custom transformer block (RETRO block) to enhance an autoregressive language model. The retrieved document encoder shares ideas with our approach, as it is conditioned on intermediary features of the larger generator model. However, they differ significantly in purpose, as our retriever generates queries, rather than processing what is eventually retrieved.

FLARE is a technique for active retrieval, meaning that the model can actively infer when and what to retrieve during generation [9]. Their model, $\text{FLARE}_{\text{direct}}$, would first generate a temporary sentence. If any token in the sentence has a probability below a given threshold, the system would create a search query based on the sentence and then regenerate the sentence using the retrieved documents. To retrieve data they either used BM25 to search Wikipedia or Bing to search the web, using queries phrased in natural language in both cases. Our approach would also be classified as active retrieval, but we will instead use a query vector to retrieve relevant documents. Another difference is that FLARE bases the query on the generated output, while our approach will base it on an intermediate representation of the input text.

ReAct is an approach which combines reasoning and acting in LLMs to enhance performance in many language processing tasks [18]. It generates both reasoning traces (similar to CoT steps) and action calls which allow it to gather more information. Applied to various tasks such as question answering and fact verification, ReAct consistently outperforms baselines while also producing more interpretable results. The ReAct approach is similar to ours in that it can generate CoT steps and search for new information continuously before producing an answer. However, searching is only done if the model explicitly requests it through a `search`-action call. Furthermore, the search is entirely based on the argument generated by the model to the `search`-action. ReAct has a clear division between reasoning steps and

action steps, while our approach implicitly performs searches based on changes in embeddings as the reasoning steps are being created.

GRIT is an LLM that can both embed texts and generate the next token, leading to a 60% speed up in RAG systems [17]. The quality of the next token prediction and the embeddings matches that of systems trained individually on those tasks. GRIT differentiates between the two tasks by providing the model with either the `<|embed|>` instruction or the `<|assistant|>` instruction. The authors note that one possible downside of this approach is that it requires more compute since the model is trained with two different objective functions. GRIT is similar to our approach in that it uses a single model to produce both embeddings and tokens. However, the embedding model in our case is a separate head attached to a preexisting frozen generative LLM. Furthermore, in GRIT the model is instructed to either perform the embedding task or the generation task, while in our approach the embedding is performed implicitly whenever the model generates a token. Note that GRIT can speed up the computations of the other task by utilizing the cached computations from the first task. This is similar to how we use the cache from the generator when predicting a query vector.

2.2 Synthetic Data Generation of Multi-Hop Questions

In a related paper [19] the authors use LLaMA 65B to synthesize multi-hop questions based on pairs of documents from Wikipedia. Given a pair of documents, an answer is chosen using hyperlinks or named entity recognition. An LLM then generates both a question and two queries used to find the documents. These instances are also filtered to remove queries which retrieve incorrect documents and unanswerable questions. By letting the LLM answer the question and checking that the output is the same as the original answer, they detect whether the question is answerable. In total, the synthetic dataset they produced included 1.5M questions and 1.9M claims. They found that fine-tuning an LLM on this data leads to a significant improvement in accuracy on multi-hop benchmarks. This is similar to our approach in that they used an LLM to generate synthetic multi-hop questions to train a RAG system. However, the system they trained does not perform active retrieval, meaning that the synthetic data did not include labeled CoT steps like our approach.

3

Theory

The following chapter will give an overview of the theory used in this thesis. It covers LLMs, RAG and the concept of synthetically generated data.

3.1 Large Language Models

Language modeling involves estimating the distribution of language present in the training data. Processing language can be constructed as a sequence task, the base unit of which is usually tokens (words or subwords).

$$P(w_t|w_1...w_{t-1}) = f(w_1...w_{t-1}) \tag{3.1}$$

In Equation (3.1), we express the probability of the last token w_t as being dependant on only the previous tokens, w_1 through w_{t-1}, and is the formulation referred to as Autoregressive (AR) language modeling. f can be modelled using a neural network of different architectures, and historically Multi-Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs) and transformers have been used, to name a few. Using f, you can determine the probability of an entire sequence of language by applying f iteratively over it, or you can generate entire new sequences by sampling the probability of the next token. Generative AR language modeling extends the formulated next token prediction with an iterative inference scheme, where the input extended with this step's output is the input to the following step. In this way, an LLM can build large sequences of language by incrementally extending a sequence. Though, machine learning for language modeling as a field is more diverse than AR generation.

Other approaches to modeling language exist, such as BERT [4], which conditions on both the previous and subsequent context of the text sequence with a focus of extracting deeper representations. For instance, AR-language modeling may have difficulties when words have dependencies in the future as illustrated in Figure 3.1. Regardless, to do effective language modeling, LLMs learn to extract and process the semantic meaning of the sequenced tokens [20]. Besides the original task, this semantic encoding can be used for other text-related tasks such as searching for text that is semantically similar (see Section 3.2.1). Models focused on this task are generally known as embedding models.

```
['<s>', '_The', '_batter', '_swung', '_his', '_bat']
```

Figure 3.1: **"_bat"**'s meaning is dependent on "_swung" and "_batter", while **"_batter"** is dependent on "_swung" and "_bat". **"_bat"** would work well in an autoregressive manner, while **"_batter"** would be more difficult since its dependencies are in the future. I.e., "_The_batter" could be followed by either "_swung" or "_tasted_good", and its vector representation would be mixed.

3.1.1 Components of language modeling

Tokens are the base unit of a sequence in language modeling. These can be characters, words or subwords based on for example the Byte Pair Encoding (BPE) algorithm [21]. Tokenization of text is benificial in language modeling to shorten the sequences the models work on, both for performance and efficiency reasons [22]. In RNNs, reducing the sequence length is important to minimize its weakness on long dependencies, and in transformers to mitigate its quadratic memory and compute requirement in the length of the sequence. However, using tokenization requires picking which specific tokens you use, something that is known as a vocabulary of tokens. This may introduce biases and bugs into the prediction task [23].

Vocabularies refers to the set of different tokens chosen to encode the text the model works on. In BPE this is done algorithmically based on a representative sample of the larger training data, though researchers often insert special tokens into the vocabulary manually. The begin and end-of-sequence tokens (BOS, EOS) are included in vocabularies for generative models, to control the sampling of the model at inference time and delimit unrelated text documents during training [24]. Additionally, in instruction tuning (see Section 3.1.6), [INST] and [/INST] are sometimes introduced to mark the beginning and end of an instruction, and in masked language modeling a [MASK]-token is used to corrupt text [4]. The choice of vocabulary can be deceptively important, and can be the cause of large biases both in the performance and efficiency of a model between different languages and tasks [22]. This is in part because of generalisation issues when biased tokenization causes some tokens to be underrepresented in the training data. Typical vocabularies for LLMs today range between 32,000 and 256,000 different tokens with multilingual models typically exhibiting larger vocabularies (see Table 3.1).

Table 3.1: Vocabulary sizes and paper names/sources of various well known LLMs.

Model	**Vocabulary Size**	**Paper Name/Source**
Mistral-7B	32,000	*Mistral 7B [25]*
LLaMA 2	32,000	*LLaMA: Open and Efficient Foundation Language Models [26]*
LLaMA 3	128,000	*Introducing LLaMA 3: The Next Generation of Large Language Models [27]*
Gemma	256,128	*Gemma: Open models based on gemini research and technology [28]*
BERT	30,522	*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [4]*
Multilingual BERT	110,000	*BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [4]*

Training objectives can differ in models depending on the intended usecase. In generative AR language modeling, next token prediction is used to train the model to predict the next token in a sequence given the previous, as shown in Equation (3.1) and realised through a cross entropy loss L (see Equation (3.2)). The desired output given the input sequence $w_{<t}$ and the model parameters θ is a probability distribution over the model's vocabulary at token t, representative of the next token probabilities.

$$L(\theta) = -\sum_{t=1}^{T} \log p(w_t | w_{<t}; \theta) \tag{3.2}$$

BERT is trained with the Masked Language Modeling objective, which aims to reconstruct text that has been corrupted with a [MASK]-token, also formulated as a form of cross entropy loss (see Equation (3.3)). Here, M is the set of masked positions, and $w_{\setminus i}$ is the input sequence with the i-th token masked.

$$L_{\text{MLM}}(\theta) = -\sum_{i \in M} \log p(w_i | w_{\setminus i}; \theta) \tag{3.3}$$

Embedding models are often trained on sentence pairs with a contrastive loss, which aims to pull similarly labeled samples close in embedding space and push dissimilar samples further away. A common loss function for this purpose is the triplet loss, shown in Equation (3.4). Here, $\mathcal{T}$ is a set of triplets, with e_i and e_j being embeddings of similar items, e_k being an embedding of a dissimilar item, d is a distance function (e.g., Euclidean distance), and "margin" is a hyperparameter that defines how far apart dissimilar items should be.

$$L_{\text{contrastive}}(\theta) = \sum_{(i,j,k) \in \mathcal{T}} \max(0, d(e_i, e_j) - d(e_i, e_k) + \text{margin}) \tag{3.4}$$

3.1.2 Transformers

The transformer architecture was novel in its heavy reliance on self-attention mechanisms across its entire structure [29]. Unlike other architectures, each token in the

input sequence can dynamically "attend" to any other token, adjusting its perception based on the overall context provided by the entire sequence. Simply put, the attention attributed to other tokens in the sequence can be thought of as a measure of how important they are for the current token. However, this should not be seen as an *explanation* for the predicted token [30]. This model contrasts sharply with convolutional neural networks (CNNs) and multilayer perceptrons (MLPs), which apply the same transformation uniformly across all inputs. The transformer's ability to process each input token differently based on context leads to its exceptional performance on a wide range of sequence modeling tasks. Additionally, in contrast to RNNs which has a bias towards local relationships [31], every token in the transformer has the equal ability to be affected by any other token in the sequence regardless of distance.

More formally, the attention mechanism uses the learnable weight matrices, W_Q, W_K, W_V, which are all multiplied with the input sequence X to form the query, key and value matrices $XW_Q = Q$, $XW_K = K$, $XW_V = V$.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.5}$$

The Q and K matrices are multiplied and divided by the root of their dimensionality d_k, which stabilises the gradient during training. Finally, it is passed through a row-wise softmax which acts as an activation function as well as a normalizer, before being multiplied with the value matrix. This concludes the scaled dot-product attention mechanism. The rest of the transformer block consists of a residual [32] add and normalize operation [33], a feed forward layer (which for a given block is one set of weights applied identically to each token) and another add and normalize. If this is the last layer, this is referred to as the logits over the model vocabulary, which is finally followed by a softmax to get a probability distribution.

The original transformer architecture included both an encoder and a decoder module (see Figure 3.2), though for generative language modeling the decoder-only variant has become popular for various reasons mostly relating to efficiency. The advantage of this type of model is that the computation for generating the next step in the sequence is solely dependant on the calculations that came before it, which means that no recalculation has to be done for previous token positions when inferring the next. The consequence is that each token can be trained in parallel, and inferenced with a cache of the previous KV matrices (referred to as a KV-cache [34]). This is not the case for transformer variants with bidirectional attention such as the encoder, which has to recompute the whole sequence each time it is expanded. Effectively, this makes next token prediction with KV-cached decoder-only models linear in the sequence length, while encoder models with bidirectional attention would yield quadratic complexity for each new token generated.

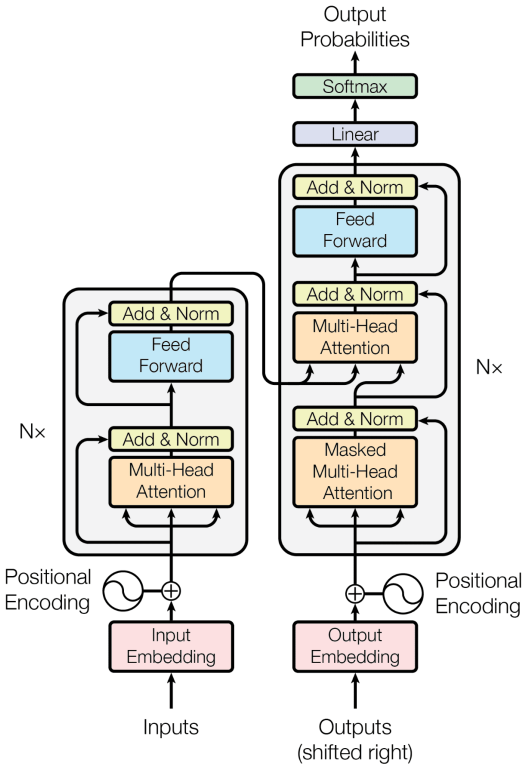


Figure 3.2: The original encoder (left) - decoder (right) transformer architecture [29]. This project features the decoder-only variant, which skips the cross attention with the encoder output.

3.1.3 Application of LLMs

Generative models are a subset of the larger LLM category designed to generate new content based on input data. They are trained on large datasets to learn its patterns, structure and content, and as such can have a wide range of applications. Chatbots, content creation, data augmentation or synthetic data generation are applications of generative models. Large generative models display some measure of reasoning, and agenic behaviour (where the model completes multi-step tasks on behalf of a user) is a current trend in the research field [35].

Embedding models refer to LLMs whose specific purpose is to capture the semantic meaning of text, and are trained to create dense vector spaces where semantically similar texts are close together [10]. These embeddings are useful for applications like information retrieval (e.g., finding relevant documents), semantic search (e.g., searching based on meaning rather than keywords), and recommendation systems (e.g., suggesting similar content).

Classification tasks include things such as spam detection, language detection, and more generally topic categorization by classifying texts into predefined topics. These models are commonly finetuned from other base models, such as BERT, on task-specific labeled datasets.

3.1.4 Hallucinations

Hallucinations in LLMs refer to the generation of text that is not grounded in the input or real-world facts, often producing incorrect or nonsensical information. Causes include lack of sufficient context, ambiguous prompts, or the model's over-reliance on learned patterns rather than factual knowledge [36]. Smaller language models often suffer more from hallucinations than their larger counterparts, likely due to their inability to store vast amounts of factual information in their limited number of parameters [37]. Several methods to mitigate hallucinations have been proposed, and include improved training datasets [38], better prompt engineering [39], and incorporating external knowledge sources as support when generating content [5, 38].

3.1.5 Quantization

Quantization refers to representing the model parameters and activations in a datatype with lesser size at the cost of precision. Research has shown that while models are often trained in float32, float16, or brainfloat16, they can be compressed down to an 8-bit float or integer representation in inference without noticeable performance loss. Some methods even allow as low as 6 or 4-bit weights [40]. Quantization has allowed models to be run more widely on consumer hardware.

3.1.6 Instruction fine-tuning

LLMs in interaction with humans, particularly in chat systems, have been found to benefit from an instruction tuning step. The purpose is to transform the model

from the raw pre training objective of predicting the most likely sequence of any text, into a responsive system that follows the user's intent [41]. This allows a user to simply describe a problem or task and the LLM will respond in the form of an attempted solution.

3.1.7 Chain of Thought reasoning

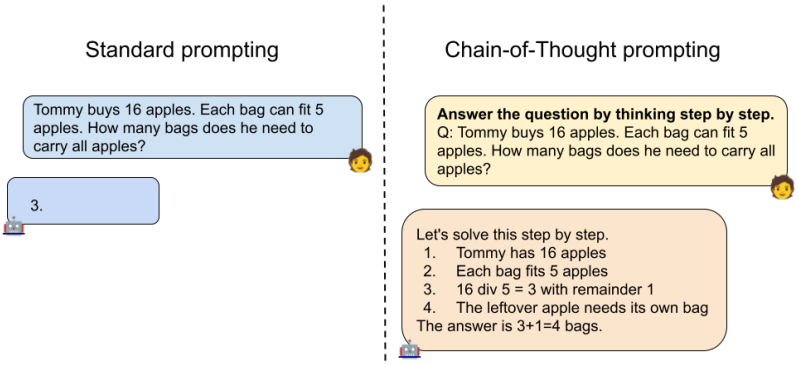


Figure 3.3: Chain of Thought prompting elicits reasoning [13]. Without step-by-step reasoning, the model has difficulty inferring non-obvious logic.

CoT reasoning is an alternative to providing the answer to a question or task outright (see Figure 3.3). Instead, the model is prompted to explain its thoughts step by step, working incrementally towards the answer in small logical steps which avoids complex logical leaps. CoT reasoning boost model performance, especially on mathematics, commonsense and reasoning benchmarks [13].

3.2 Retrieval Augmented Generation

LLMs are able to answer factual questions by storing knowledge in their parameters [3]. RAG aims to substitute LLMs reliance on parametric knowledge with retrieved documents. By searching a database, the system can find relevant information to accomplish its task, which could be used to answer the user query in the case of a question-answering system. This reduces the risk for hallucinations in the output from the LLM [42]. Information retrieval can also be used for other purposes in the field of Natural Language Processing (NLP), such as fact checking [43] or translation [44]. Generally, each entity in the database is embedded to a semantic vector representation using an embedding model [10]. A semantically related entry can then be found with a nearest-neighbor algorithm.

3.2.1 Vector databases

A vector database is a data structure that specialises in nearest-neighbor search on vectors. By some distance metric, for instance cosine similarity or euclidean distance, the vector database can retrieve the nearest neighbors of a query vector. You can for instance embed text documents into a dense vector representation, and use this

in conjunction with a vector database to find similar documents even if there is no keyword overlap. Many vector databases implement approximate nearest neighbor algorithms, which trade off correctness for speed. These can retrieve documents in O(log(T)) where T is the number of entities in the database, instead of the O(T) required for an exhaustive search. Faiss is a vector database developed by Meta's FAIR group and is free and open to use. Faiss implements both a CPU and an GPU accelerated version [45] of their database, with both exhaustive and approximate retrieval [46]. Fast retrieval is especially useful in the active retrieval use-case where repeated retrievals make latency important (see Section 3.2.2).

3.2.2 Active retrieval

Retrieval augmented generation is in its base implementation static, with a single retrieval before the generation process. In contrast, active retrieval allows the system to retrieve new documents throughout the generation process. The intent of this approach is to increase the quality on long form tasks, where gathering new information throughout is essential [9]. However, it comes with increased latency since retrieval needs to be performed multiple times and swapping the documents may require recomputing activations that could otherwise be cached. See Chapter 2 for some methods for and implementations of active retrieval systems.

3.3 Synthetic data generation

Synthetic data generation is the process of automatically generating new training data. This is distinct from data augmentation which creates modified copies of original data [47]. However, the terminology is new and some sources consider synthetic data as a form of data augmentation [48].

Generative models such as generative LLMs can be used to generate new data. This is especially useful when training domain specific models since the specialized data required to train them may not be available [48]. For example, synthetic data from LLMs have been successfully used in a medical setting where there are major privacy concerns from using real patient information [49]. However, the synthetic data produced by LLMs can contain biases or factually incorrect statements due to hallucinations [50, 51].

3.3.1 Multi-hop questions

One use case for synthetic data generation is to build multi-hop questions, i.e. questions that require reasoning over knowledge from more than one document to answer [8]. These questions could be synthetically generated from only a set of documents, reducing the manual labor required [19]. This synthetic data can then be used to train RAG systems or fine-tune question-answering systems (see Chapter 2). One possible downside of this approach is that generating large numbers of questions with LLMs can still be expensive if a powerful LLM is being used.

4

Methods

The following chapter is divided into three parts. The first part analyzes the optimal way to construct the proposed architecture by testing it in isolation on an already established dataset. It covers experiments to establish which layer of the LLM the new head H should be attached to, as well as what architecture it should have. The second part covers the pipeline for constructing synthetic multi-hop questions. Furthermore, it details a technique to use an analysis of the transformer attention to reduce hallucinations in the constructed data. The final part covers how the final system was tested in an active retrieval setting.

4.1 Utilizing Intermediary Representations

In order to verify that it is possible to convert an intermediary representation to query vector matching a document embedding, the proposed architecture was trained on an existing dataset, in this case HotpotQA. Using HotpotQA instead of only training on a synthetic dataset ensures that the architecture is tested in isolation. This makes the results produced by the architecture easier to compare to other results in the literature.

While using HotpotQA is useful for comparability, it does come with some downsides. It does not provide reference-labeled CoT steps for the questions. Therefore, the system was trained to produce the correct search vector while generating the *answer* instead. In a full QA pipeline this would be problematic as the answers could be too short and not allow the model to find all required documents. Furthermore, each question in HotpotQA has two gold documents, i.e. two documents that are labeled as correct. Therefore the architecture was adapted to produce two search vectors for each token in the answer, instead of just one. Note that the ultimate goal is to produce one search vector for each token in a series of CoT reasoning steps.

Our architecture is a new MLP head H attached after some layer in an LLM (the best layer to use is analyzed in Section 4.1.1). Specifically H takes in the intermediary representation of the last token in the sequence produced by the MLP component in a transformer block. The LLM used was Mistral-7B [52] due to its wide adoption and good performance in many benchmarks. In order to reduce VRAM usage and speed up inference, a 4 bit quantized version of Mistral was used. The specific weights are available to download from https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-AWQ. Furthermore, the model is instruction fine-tuned in order to

work well in a question answering setting. This quantized and instruction fine-tuned model will be referred to as Mistral-7B for the sake of brevity.

The intermediary representations are created by feeding a prompt to Mistral-7B and observing the representations of the tokens for the answer. The prompt contains context (i.e. some documents with facts), a question, and the answer. The goal is that the final system should be able to swap the information in the context based on the current search vector. Therefore, the expectation must be that the documents in the context could be incorrect. 4 random documents were assigned to the context as a pre-processing step in order to simulate the fact that there may be incorrect documents retrieved to the context. If the context instead would contain the gold documents, then H might just learn to embed the documents in the context, effectively giving it privileged information.

For every token that is part of the section in the prompt where the system should be generating search vectors (in this case the answer), an intermediary representation is stored. This effectively expands each token from a couple of bytes to a 4096 long vector of 16 bit floats, requiring much more disk space to store the dataset. However, this allows the inference through Mistral-7B to be skipped at training time, since all the intermediary representations are saved. By precomputing the intermediary representations, the training time is reduced at the cost of storage.

The embedding model used was all-MiniLM-L6-v2 which is a small and fast version of the Microsoft model MiniLM [53]. The model all-MiniLM-L6-v2 (here on simply referred to as MiniLM) is available at https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. MiniLM is used to embed the gold documents which serves as the target output for H. Furthermore, it is also used as a baseline system to compare retrieval accuracy, memory usage and inference speed.

4.1.1 Layer analysis

Mistral-7B is made up of 32 transformer blocks that all produce an intermediary representation that could be used as input to H. In order to find out which is most useful when producing search vectors, H was trained on multiple different layers in order to compare their accuracy. Saving the intermediary representations for every sample in HotpotQA for every layer would require too much storage. Therefore, the data for this analysis was limited to only 10% of HotpotQA, and furthermore involved only even layers between 2 and 32. While the reduction in data will reduce the final performance, the relative performance between layers will ideally remain similar.

The documents from the HotpotQA test set were embedded and added to a Faiss vector database (see Section 3.2.1 for details on Faiss). H is evaluated by retrieving k documents from the database and counting how many of the two correct (gold) documents were retrieved. This is averaged over the test set, giving the Mean Correctly Retrieved Documents (MCRD), a score which for 2 gold documents ranges from 0 to 2. Through this analysis, an optimal layer is found and used in subsequent experiments.

4.1.2 *H* architecture experiments

Beyond the choice of which intermediary representation to use, there is also the choice of what architecture H should have. Five variants of an MLP were analyzed to see which gave the best accuracy.

Table 4.1: Table showing the architecture of the 5 models analyzed. Every model has an input size of 4096 (the shape of the intermediary representation from the LLM) and an output shape of 384×2, i.e. the two query vectors. *Small*, *normal* and *large* changes the number of neurons per layer, while the deep versions are the same as normal but with more hidden layers.

Model Name	Hidden Layers	
	Number of layers	Neurons per layer
small	1	384
normal	1	384×2
large	1	384×4
deep-1	2	384×2
deep-3	4	384×2

Similarly to Section 4.1.1 the variants of H were evaluated on the MCRD from a faiss database of 7405 HotpotQA test documents. However, since analyzing H architectures only requires one instance of the intermediary representations, this analysis uses the entire dataset instead of just 10%. This analysis was done to pick an architecture for H to use for the following tests.

4.1.3 Comparison to baseline

In order to know if intermediary representations are useful for predicting search vectors, the system has to be compared to a baseline. The baseline system involves embedding the prompt (consisting of a context and a question) with MiniLM and retrieving k documents using that embedding. However, in a system using MiniLM as a query vector predictor there would be no reason to include the context when embedding since it may include incorrect documents; it is better to just embed the question. This is not an option when using H since the (possibly incorrect) documents must be provided to the LLM in a RAG context. Therefore, two types of MCRD were analyzed, one where the context contains 4 random documents and one where the context is left empty.

Furthermore, the two approaches were investigated on their inference time and memory usage. The inference time is counted as the *additional* time needed to infer a query vector, i.e. for H this excludes the time needed to run one inference pass through the Mistral-7B model we used. In a RAG-setting, the generating model (Mistral-7B) has to run regardless. Memory usage is defined as the amount of VRAM required to run each of the models on a sample from HotpotQA.

4.2 Synthetic Multi-hop Questions

While the architecture is validated by experiments described in Section 4.1.2, this section moves on with dataset experiments. Since the goal is to be able to use this system on unlabeled datasets that only contain documents, synthetic data generation was used. The question, CoT, and answer (i.e. (Q, CoT, A)-triples) were synthetically generated for a set of documents D using Mistral-7B. The CoT steps are generated on the form `"(Document <i>): <text>"` where the LLM generates both the document label, and reasoning over the relevant information from that document. The full prompt used to generate this can be found in the Appendix A.1.

The CoT part is an essential addition to allow the model more tokens to reason about its answer, facilitating the retriever head H (which works token-by-token) to retrieve new locally relevant documents. This is also a reason for synthetically generating a dataset, since other open multi-hop QA datasets (like HotpotQA) lack document-labeled CoT reasoning steps (for more info, see Section 3.1.7 on CoT reasoning and Section 3.3 on synthetic data generation).

4.2.1 Question generation pipeline

The dataset is based on a dump of Wikipedia from dumps.wikimedia.org, specifically `enwiki-latest-pages-articles-multistream.xml.bz2` accessed 2024-02-19. Since Wikipedia is very large, the dataset had to be reduced in size. In this case only articles which link to the page **Algorithms** were kept. These articles were then cleaned up slightly by removing some formatting and metadata before being chunked using a SentenceSplitter from llama-index [54] which divides the articles into smaller text chunks with a preference for keeping sentences and paragraphs together. These text chunks were used as the set of documents D.

Building a multi-hop question requires two chunks which contain information about a similar subject. Therefore, every chunk was embedded with MiniLM to calculate their distances in embedding space. For every chunk, it was paired with its closest neighbor. Each pair was sent to Mistral-7B running with the default Ollama configuration [55], with a prompt instructing it to formulate a multi-hop question, CoT reasoning steps, and an answer. The prompt specifies that if it is impossible to construct a multi-hop question from those documents, it should return the string `INVALID`. In that case the document pair is ignored. Furthermore, the prompt contains instructions about how the output should be a JSON object with a specific structure. If the output does not follow that structure, the generation failed and is attempted again with the same document pair. Since the CoT labels are generated by the LLM only based on the prompt, this dataset is named the Prompt-Labeled (PL) dataset.

4.2.2 Attention-based relabeling

One potential issue with this approach is the way the CoT labels are generated. Letting the LLM generate the label and the summary of the relevant information

means that the model could hallucinate and generate the wrong label. Since the prompt tells the model to generate multi-hop questions, the model may label CoT steps as coming from both documents while only summarizing information from one of the documents.

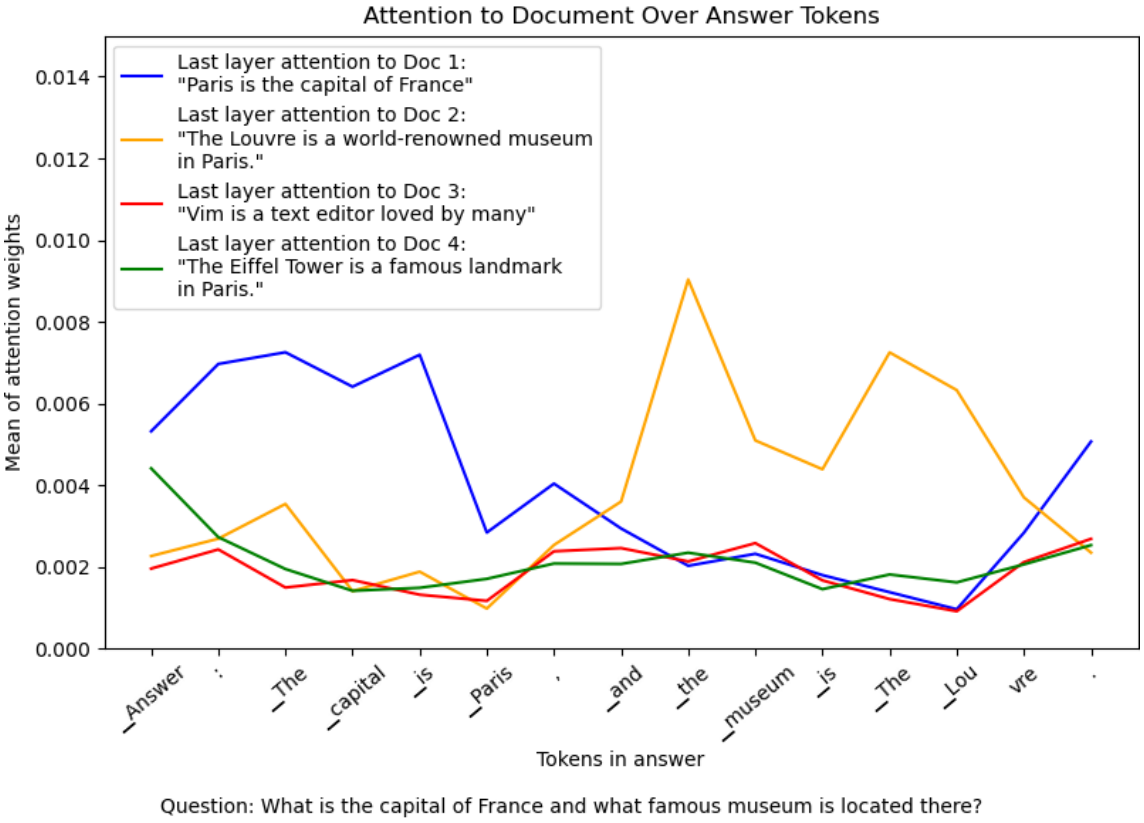


Figure 4.1: Mean attention attributed from answer tokens to document tokens over the last layer only. We can see how the model switches from attending to one document to the other at the expected times. The relevant documents (Doc1, Doc2) are contrasted with irrelevant, and related but not needed documents (Doc3, Doc4). (This pattern is replicated on problems with larger documents in the CoT reasoning. See figure 5.7).

Instead of labeling entire CoT steps based on the output from the LLM, each token could be labeled based on which document receives the most attention when generating that token (see Figure 4.1). Using the last layer's attention averaged over all attention heads, each token receives the label 1 or 2 based on which document has the largest mean attention attributed to it by that token. Using this technique to relabel the tokens defines a new dataset, called the Attention-Labeled (AL) dataset.

The main idea behind attention-based relabeling is that this would reduce the errors caused by hallucinations since the attention conveys which document was actually relevant for a given token prediction. However, the reason a document is relevant does not have to be because of factual dependencies (see Section 3.1.2). The attention from other layers was also analyzed, but was found to be less informative

for the purpose of identifying correct document labels. Furthermore, using the max attention instead of the mean attention was also tested, but it was much less stable than the mean.

4.2.3 Training *H* on the datasets

H was trained on each of the two datasets (PL and AL) in order to evaluate differences in MCRD. These questions were fed through Mistral-7B to extract the intermediary representations. Using a 90%-10% train-dev split to train H we define two models based on which dataset they were trained on: H trained on PL (HPL) and H trained on AL (HAL).

4.2.4 Manual evaluation on a random sample

In order to evaluate if the labels in AL are more correct than the ones in PL, they were compared to a small subset of manual labels. 50 questions were randomly selected as a test set. A human annotator was then given the question, answer, documents and CoT steps and was tasked to label which document each CoT step derives from. To avoid bias, the human annotator did not have access to the labels in either AL or PL. For each CoT step they could assign one of the following labels:

- **1**: The step references document 1
- **2**: The step references document 2
- **12**: The step references both documents
- **n**: The step references neither document

Using these manual labels as the ground truth, PL and AL were evaluated on their correctness by counting the number of questions for which the CoT steps have the same labels as the ground truth. Furthermore, the labels are used to divide the questions into multi-hop and single-hop to evaluate which kinds of questions the datasets tends to label incorrectly.

4.3 Active Retrieval

The H-model was designed with the active retrieval use-case in mind, and such this section details the experiments in building an active retrieval augmented generation system.

4.3.1 Inference with Active Retrieval

Active retrieval involves fetching new documents actively while generating text. The simplest implementation of this scheme was used in these experiments, where a new retrieval is made every t tokens generated. For the HotpotQA experiments, $t = 20$ was chosen as the answers were observed to be 40-60 tokens long at the lower end, which would still allow the model 2-3 searches for new information during the

answering process. The documents used to generate the AL and PL datasets were on average 512 tokens long. Hence, to not fill up the context length of the LLM excessively, documents retrieved per search was limited to $k = 4$. These retrieved documents replace the documents that were previously occupying the context.

Inference of the H-model was implemented in the same way training data was collected, by attaching a pytorch hook to extract the intermediary representation of the best layer found in Section 4.1.1. When the generating LLM Mistral-7B predicts the next token in the answer, an intermediary representation is passed through the H-model to infer a query vector. And in accordance with the retrieval schedule outlined above, every 20th query vector (starting with the first) is used to perform a query and switch the context documents.

4.3.2 Experiment on HotpotQA

The HotpotQA dev set has 7405 questions, but due to time constraints only 2736 were evaluated. The inference and retrieval scheme that was employed is described in the previous section. The H-model variant used was the *normal* architecture described in Section 4.1.2, trained on the AL-dataset from Section 4.2.2. The answers were extracted from the raw text by prompting the LLM to format its response in a json format.

The metrics tracked for the answer are F1 (overlap) and EM (Exact Match). MCRD was used to track retrieval quality. The predictions were collected using the custom inference and retrieval scheme described in Section 4.3.1 with relevant data saved to file. These files were then evaluated using the official HotpotQA evaluation script available on https://github.com/hotpotqa/hotpot.

4.3.3 Analysis of Query Convergence

To evaluate the efficacy of the active retrieval approach using our system, an additional experiment was conducted to evaluate if the documents retrieved over time converged or diverged in relation to the correct documents. As the documents used at each step were recorded in the inference process (see Section 4.3.1), it was possible to embed these with MiniLM to obtain their vector representations. These embeddings were compared on the measures of euclidean distance and cosine similarity.

Two subexperiments were performed:

- **Tracking euclidean and cosine distance to target documents during active retrieval inference**. The results are shown in Figure 5.12b and Figure 5.12a, respectively.
- **t-distributed Stochastic Neighbor Embedding (t-SNE)-projection of the trajectory of predicted query vectors.** A 2d projection of the high dimensional space was made to view the trajectory of the predicted query vectors during active retrieval for 4 questions. The results can be viewed in Figure 5.13.

5

Results

The following result section mirrors that of the Methods (Chapter 4). Firstly, the results of our experiment into the utilizing the intermediary representations of the generating LLM are displayed in Section 5.1. Secondly, the results of our synthetic data generation of a CoT-labeled multi-hop question-answering dataset are shown in Section 5.2. Lastly, the results of our active retrieval system can be viewed in Section 5.3.

5.1 Utilizing Intermediary Representations

The following section contains results from testing the architecture on HotpotQA. It details the best layer, the best architecture, and how the system compares to a baseline.

5.1.1 Layer analysis

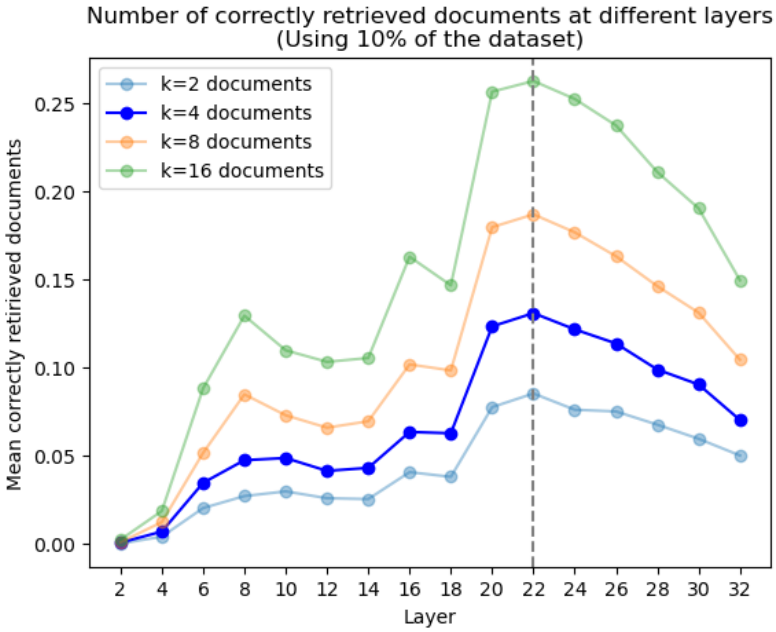


Figure 5.1: Plot showing how the MCRD of H changes depending on what layer the intermediary representation is extracted from. The different lines represent the number of documents retrieved during evaluation (k). Due to storage constraints, this analysis only used 10% of the total data when training each instance of H. The graph shows that using layer 22 provides the greatest MCRD.

Figure 5.1 shows that there is a clear difference in the MCRD depending on the layer chosen. We observe early layers slowly increasing from 0 MCRD reaching a maximum at layer 22 and then slowly decreasing afterwards. There is also a large increase from layer 18 to 20. Increasing the number of documents that are retrieved during evaluation (k) will predictably increase the MCRD.

5.1.2 *H* architecture experiments

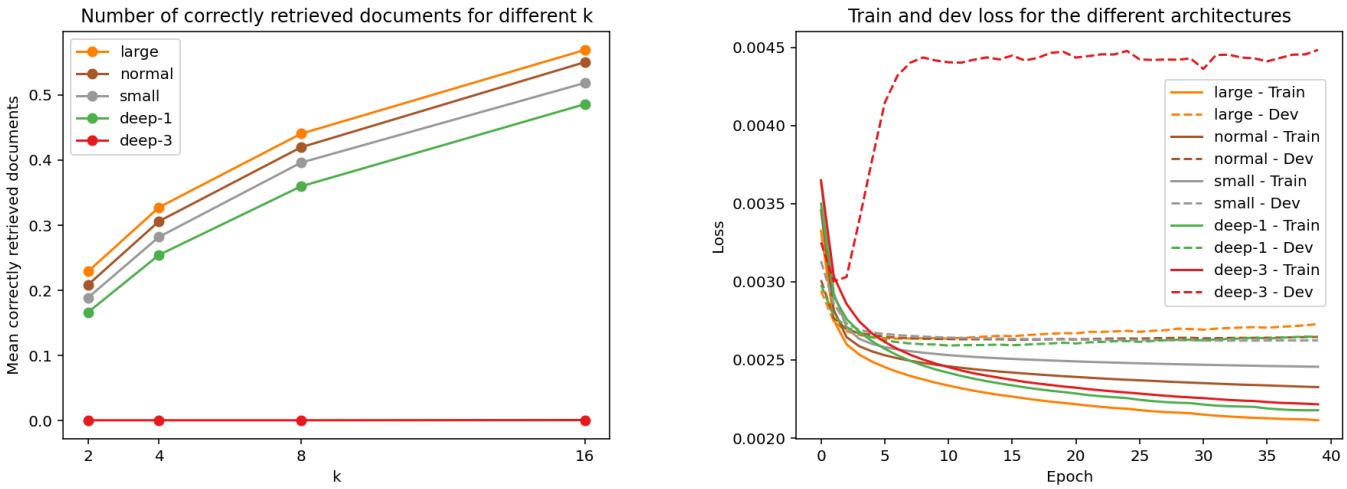


Figure 5.2: Plots showing the MCRD and loss when using different architectures for H. The results are based on data from layer 22. Going from *small*, to *normal*, to *large* improves it very slightly. However, increasing the number of hidden layers led to worse MCRD. In the case of *deep-3*, the model overfit to the training data to such an extent that the accuracy on the dev set became 0. The *large* model achieves the highest MCRD although *normal* is very close.

The left graph in Figure 5.2 shows how the *large* model achieves the greatest MCRD, however, the *normal* model also performs well. Since the training time grew with the size of the model, the *normal* model was chosen for subsequent experiments. Interestingly, even though the *large* model had the highest MCRD on the dev set, the dev loss was worse than that of the *small*, *normal*, and *deep-1* models.

5.1.3 Comparison to baseline

The results in Figure 5.3 show that H reaches roughly a third the MCRD of the baseline. Furthermore, the MCRD is higher in the cases when there are no documents in the context compared to the case with 4 random documents. As expected, the baseline suffers more from the random documents while H has an MCRD that is very similar in both cases.

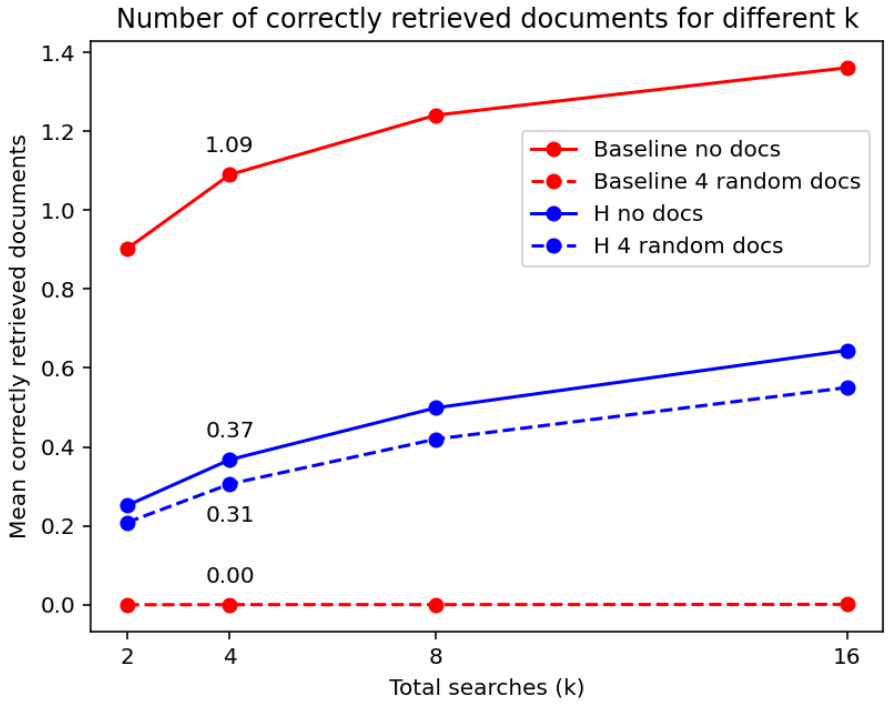
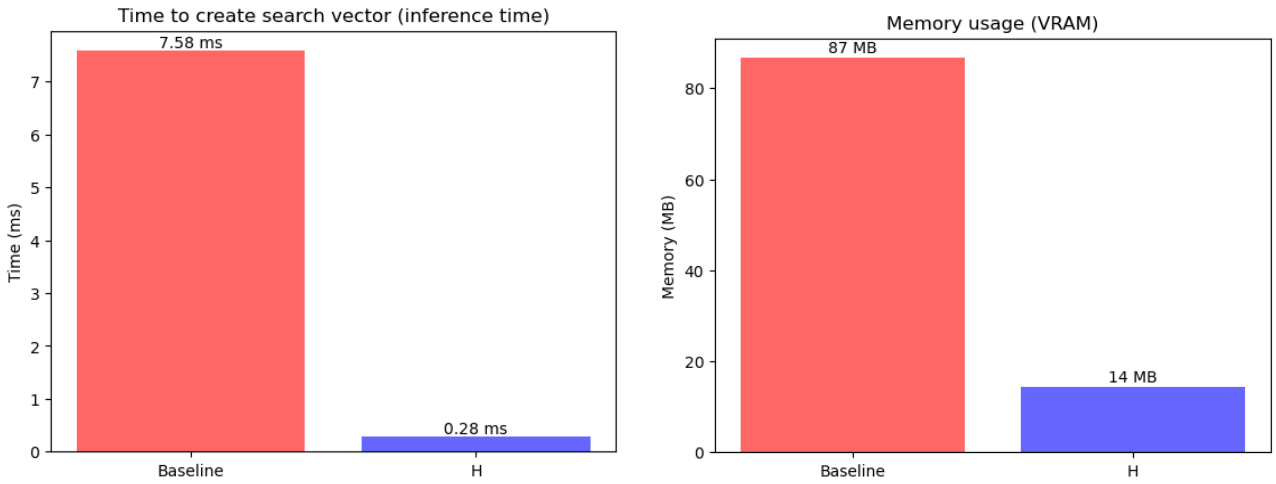


Figure 5.3: Graph showing how many of the two correct documents are retrieved when fetching a total of k documents (averaged over the test set). The figure compares the results of our model with the baseline. It also shows how the scores change depending on if the prompt includes no documents or 4 random documents.



a) Comparing the inference speed

b) Comparing the VRAM usage

Figure 5.4: Plots showing how the inference time and memory usage compares between the baseline (MiniLM) and H when running on a sample from HotpotQA. The results show that H is both much faster and much smaller compared to the baseline. Generating a search vector with H is essentially free time and memory-wise when running inference on the LLM.

Looking at the results from Figure 5.4, we see how H is much faster and smaller. This means that in a RAG-setting, we can generate a search vector for active retrieval for every new token at almost no cost. There is a trade-off since the produced search vector will not be as accurate as one produced by a dedicated model (as seen in Figure 5.3). On the other hand, even a small model like MiniLM takes 27 times longer to run and uses 6 times more memory. For reference, an inference pass through Mistral-7B took roughly 10ms when evaluating QA on HotpotQA.

5.2 Synthetic Multi-hop Questions

The following section shows the results from synthetically generating multi-hop questions. We also show how the questions changed through attention-based relabeling and how the correctness of AL and PL compare.

5.2.1 Question generation pipeline

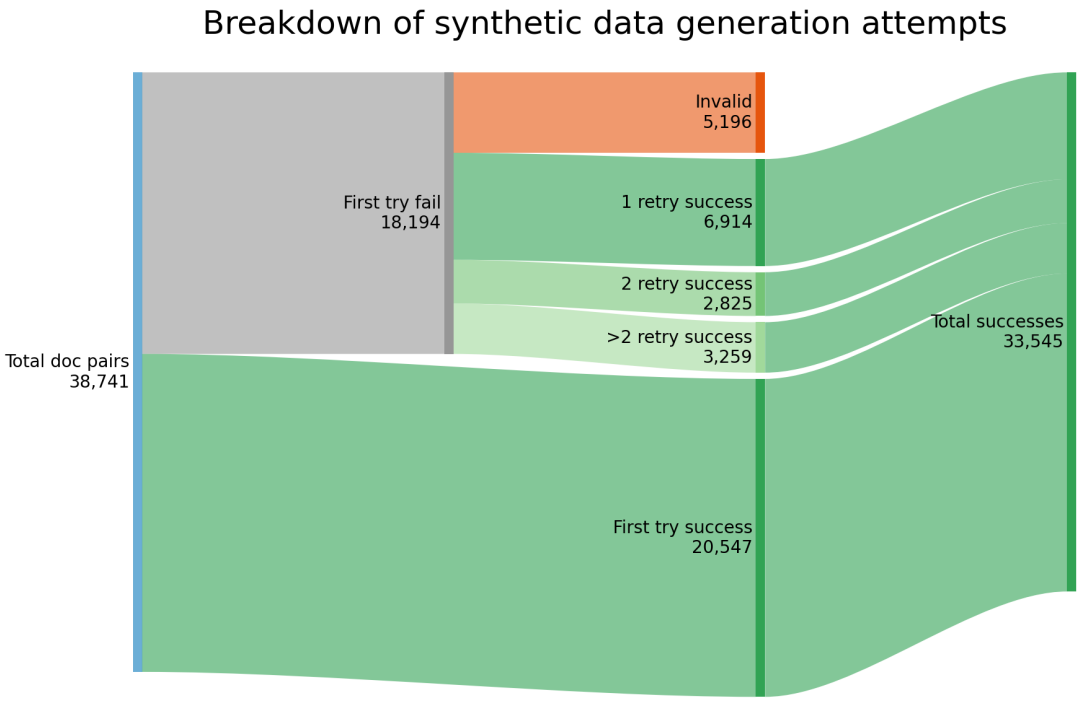


Figure 5.5: Figure showing a breakdown of how many of the document pairs resulted in a successful synthetic data generation attempt. It starts with 38741 document pairs which results in 33545 successful generation attempts. However, many of the document pairs required more than one attempt to create a valid output. For 5196 of the pairs, the LLM returned that they were invalid.

Figure 5.5 shows a breakdown of the generation attempts when making multi-hop questions from the Wikipedia dataset. This gave us 33545 successfully parseable

generation attempts. The prompt (see Appendix A.1) instructed the LLM to generate 3 questions for each pair, although the exact number of questions actually generated would sometimes vary. From the 33545 successful generation attempts we got 100689 synthetic questions. Figure 5.6 shows an example of a generated question.

Synthetic data example

Document 1

[...] Cancelling a sent e-mail on Gmail [...] to cancel a sent e-mail is impossible [...]

Document 2

[...] on many decent forum software, the PM(Private Messaging) does provide ability to retract a massage before the recipient read it [...]

Question
Compare the methods for cancelling or retracting emails between gmail and forum software mentioned in the documents.
Chain of Thought steps
(Document 1): The sender could not cancel a sent email in gmail.
(Document 2): A PM in forum software allows the sender to retract an unread message.
Answer
gmail does not allow users to cancel or retract sent emails, while forum software permits it for private messages before they're read.

Figure 5.6: Example of a good synthetically generated question. The question, CoT steps, and answer are all generated based on the provided documents. See Section 6.2.1.1 for a discussion on different aspects of this example.

5.2.2 Attention-based relabeling

Figure 5.7 shows the attention for two CoT steps which were generated. It illustrates the problem with the PL dataset and how attention-based relabeling can reduce it. The model generated one CoT step for each document, but only uses information from document 1. In this case looking at the attention shows which document is actually being referred to.

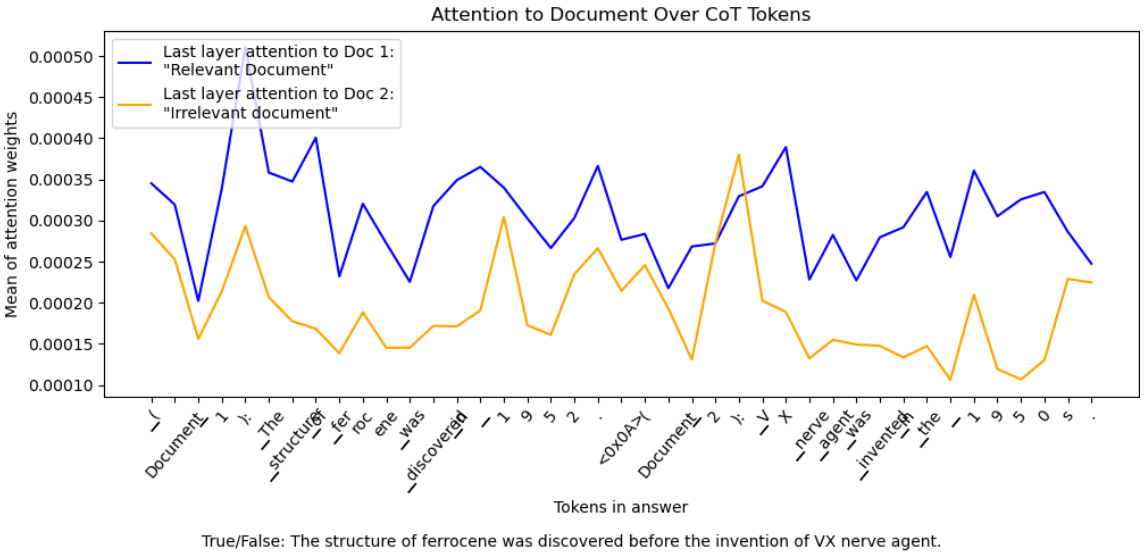


Figure 5.7: Mean attention attributed from answer tokens to document tokens. The model incorrectly labels CoT-step 2 as being derived from document 2. However, when analysing the attention from the CoT-steps to the relevant and irrelevant document, we can successfully extract the correct label.

In order to compare the two datasets we define a CoT step in AL as having the label of the majority of the tokens in that step. This allows us to check if a question in AL is multi-hop (the CoT steps reference both docs) or single-hop (all CoT steps reference the same doc). We can also check if a question in PL had a CoT step whose label was changed through the attention relabeling step. Figure 5.8 shows a breakdown of the questions resulted in the two datasets. As we can see, many of the multi-hop questions in PL became single-hop questions after attention relabeling. This further suggests that the model hallucinates the labels in PL.

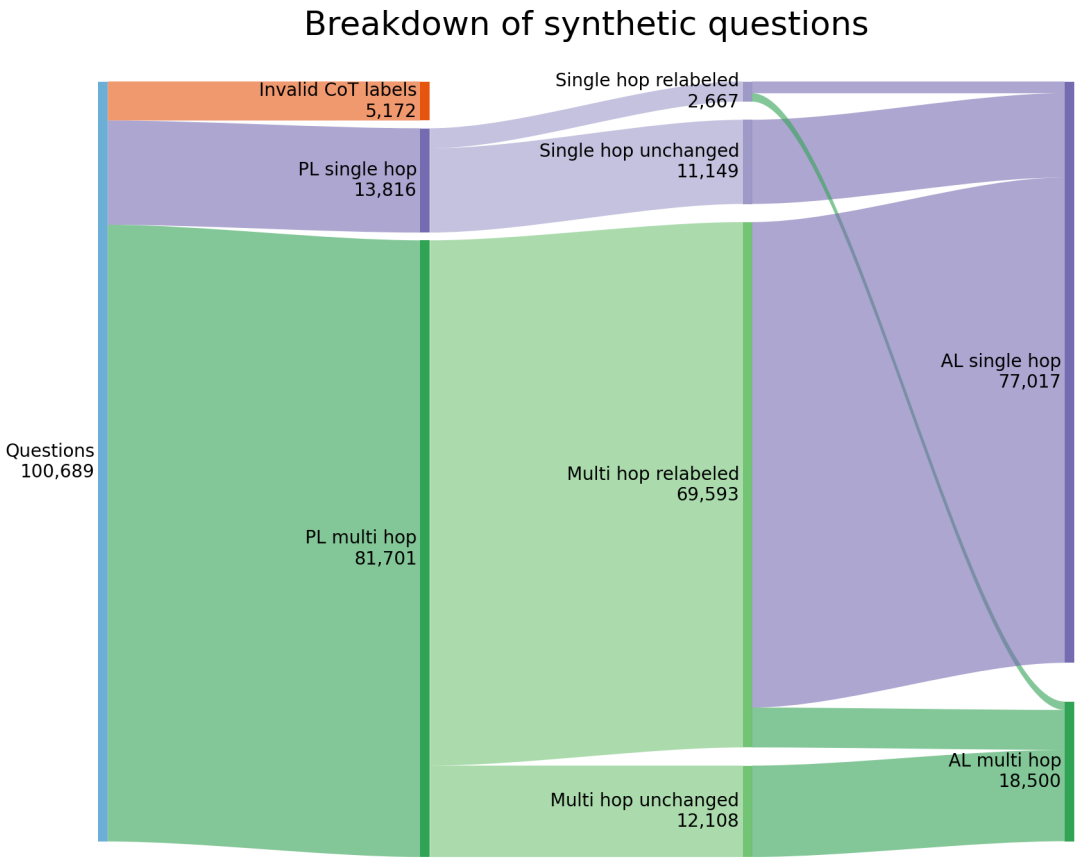


Figure 5.8: Breakdown of the 100689 questions and how they resulted in the PL dataset and the AL dataset. As we can see, many of the questions first believed to be multi-hop turn out to be single-hop when using attention labeling. There are also a small number of questions where the labels in a chain of thought step were unparseable.

5.2.3 Training H on the datasets

The results from Figure 5.9 may suggest that the attention-based relabeling step fixed some of the incorrect labels in the PL dataset, thereby reducing the noise in the training data making it easier for the model to achieve a higher accuracy.

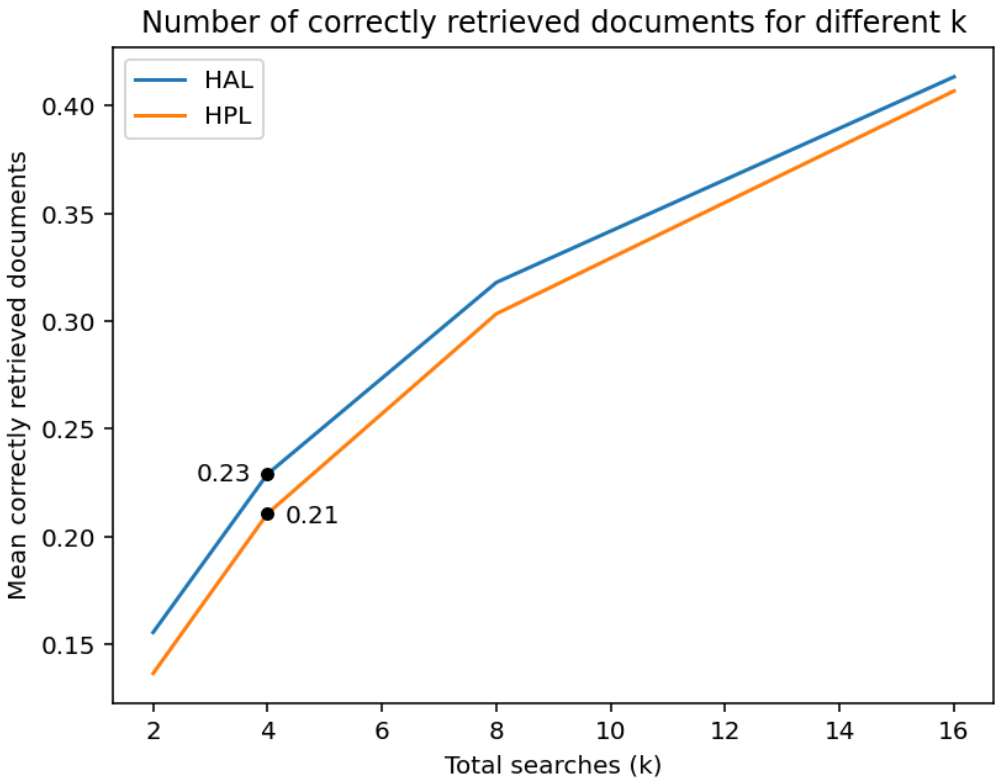


Figure 5.9: Plot showing the retrieval accuracy of the two H-models on their respective dev sets. When retrieving the 4 closest documents the HAL-model has 0.23 MCRD while the HPL-model only has 0.21, showing a slight difference in performance.

5.2.4 Manual evaluation on a random sample

Figure 5.10 shows three pie charts that break down the labels from the 50 randomly selected samples. Since AL and PL do not contain CoT steps labeled as **12** or **n**, questions that contain those labels cannot be compared. Therefore we report a manually labeled sample as invalid if it contains a CoT step with one of those labels. Note that even though there are exactly as many single-hop questions in the PL sample as there are multi-hop questions in the AL sample, this is due to random chance and does not imply that they are the same questions. We see that PL has a bias towards assigning questions as multi-hop when compared to the ground truth. AL is better, but it does have the opposite problem of assigning too many questions as single-hop.

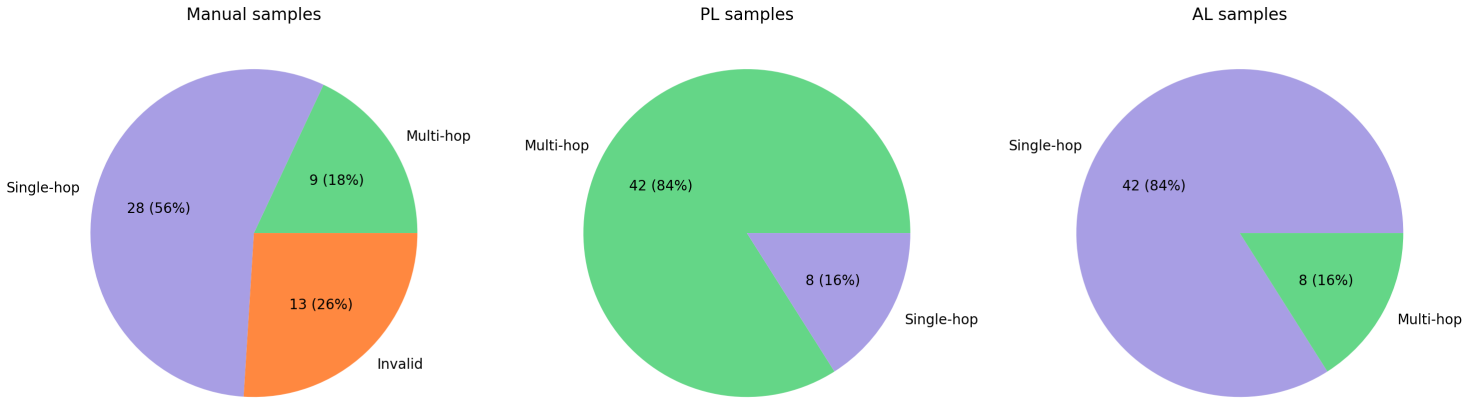


Figure 5.10: Distribution of how the 50 randomly selected samples are labeled in PL, AL, and manually by a human. When the human evaluator labeled a single CoT step as having either both or neither document as reference the sample is considered invalid. PL is too biased towards multi-hop questions and AL is too biased towards single-hop questions when compared to the manually labeled samples.

Figure 5.11 shows how correct the labels are in PL and AL when compared to the manually evaluated ground truth. Looking at just the 37 questions which were given valid labels when manually evaluating, we see that AL is more likely to have the correct labels. AL has 25 correct labels while PL only has 15, representing a 67% improvement in correctness, i.e. how well it agrees with human labeler ground truth. Furthermore, almost all of the incorrect labels in PL are of the MH-SH variety, meaning that they were considered multi-hop in the PL dataset but were actually single-hop questions according to the human annotator. Looking at the 12 incorrect questions in AL they are instead much more evenly distributed between the 4 possible categories. This shows that PL labeled many single-hop questions as multi-hop. However, AL would instead label more multi-hop questions as single-hop, as it only labeled 3 multi-hop questions correct while PL got 8.

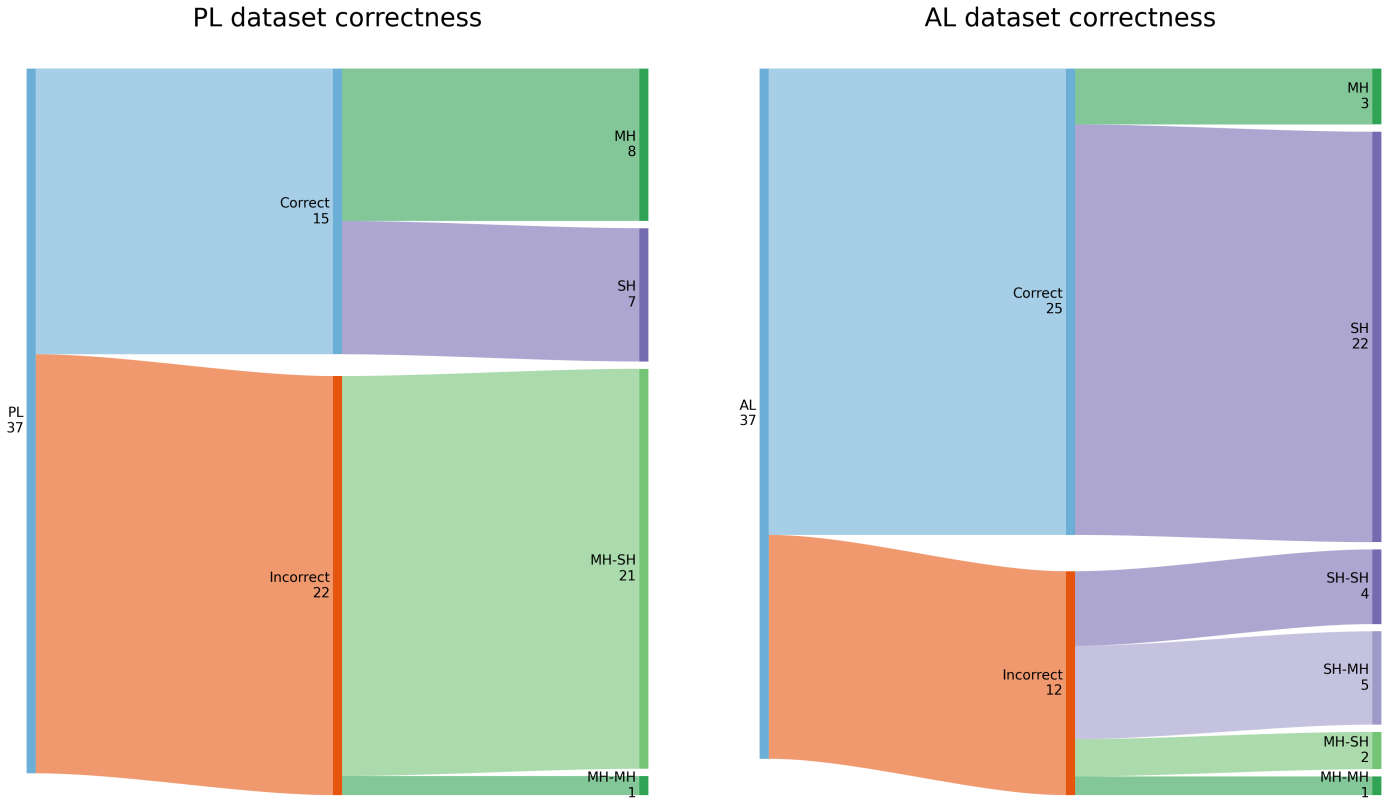


Figure 5.11: Overview of how well the AL and PL samples agree with the manual labels. The 37 samples which were not manually labeled invalid are divided into *Correct* or *Incorrect* depending on if the dataset labels are equal to the manual labels. These cases are further divided into categories like MH-SH which signify that it was labeled as a multi-hop question in the dataset but the manual label was single-hop.

5.3 Active Retrieval

This section shows our results for testing an active retrieval implementation based on Mistral-7B as the LLM, MiniLM as document embedder, and the HAL-model as retriever. The evaluations were done on a subset of the HotpotQA dev set. Further details on the method can be found in Section 4.3.

The results in this experiment largely do not align with expectations based on the results in Section 5.2.3, indicating the retrieval implementation may code-wise be incorrect. This should be taken into consideration when viewing the scores on HotpotQA in Section 5.3.1. An analysis into the retrieval performance and active retrieval behaviour can be viewed in Section 5.3.2.

5.3.1 Experiment on HotpotQA

Table 5.1: HotpotQA performance of Mistral-7B paired with HAL in an active retrieval system implementation.

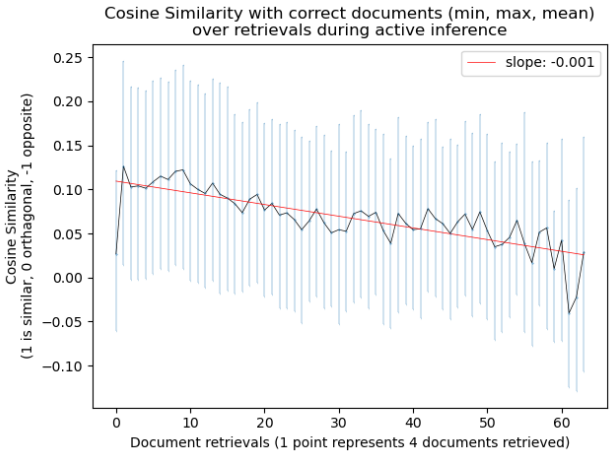
Metric	**Mistral-7B+HAL**	**Random**
em	0.02156	-
f1	0.03995	-
MCRD	0.00028	0.00004

As seen in Table 5.1, the results for the HotpotQA evaluation of the active retrieval system are poor. A column labeled "Random" has been added, where the value for MCRD is the probability to retrieve the correct document uniformly at random given the same amount of retrievals as recorded by the active retrieval system. Results from other teams can be found on the website https://hotpotqa.github.io/.

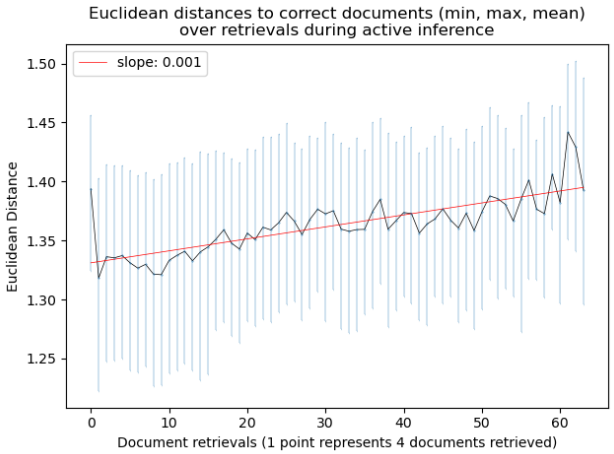
5.3.2 Analysis of Query Convergence

This experiment was performed to investigate if the documents retrieved over time converged or diverged in relation to the correct document. The experiment analysed and compared the embeddings of the retrieved documents with those of the gold documents. In Figure 5.12, we can see that the retrieved documents are poor at the start, and get worse as the generation goes on. In Figure 5.13, we can see that the distribution of the predicted query vectors is misaligned with the distribution of the embedded documents.

5.3.2.1 Tracking euclidean and cosine distance to target documents during active retrieval inference



a) Cosine Similarity of retrieved documents to the correct document, over time.



b) Euclidean Distance of retrieved documents to the correct document, over time.

Figure 5.12: The black line indicates the mean value for the 4 retrieved documents in one retrieval. The upper bound indicates the max value, and the lower bound the minimum value. We can see that the retrieved documents start off as a poor match in relation to the correct document, and diverge slightly over time to an irrelevant retrieval.

5.3.2.2 t-SNE-projection of the trajectory of predicted query vectors

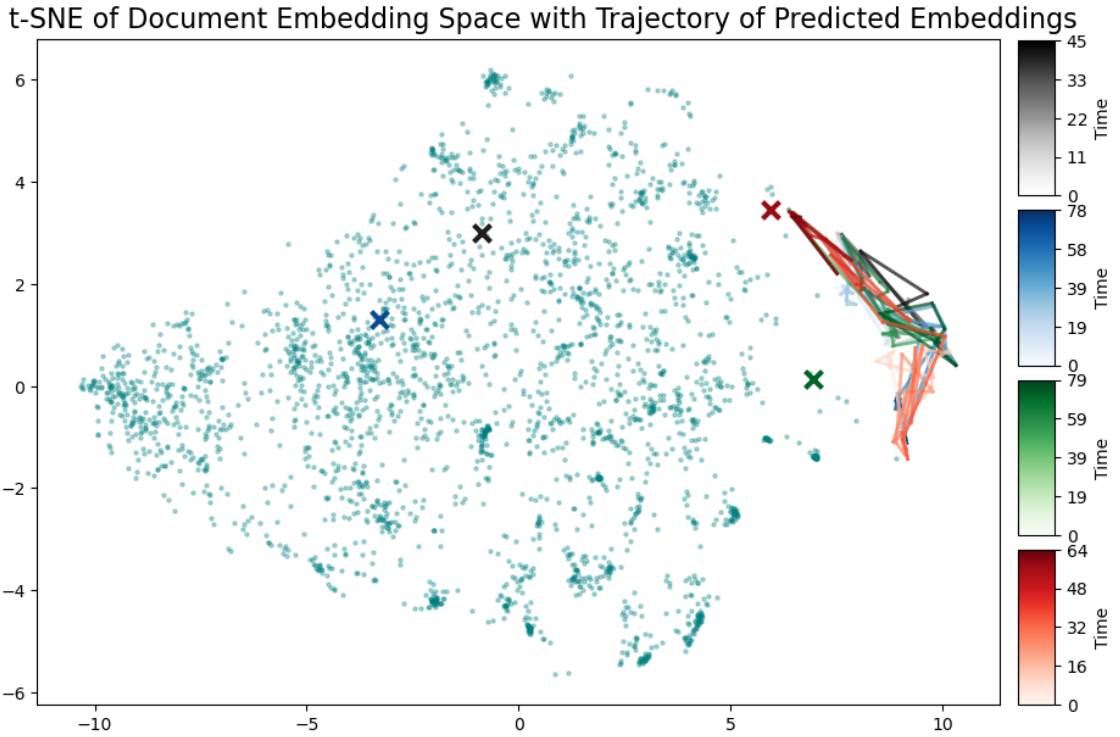


Figure 5.13: A two dimension projection (t-SNE) of embeddings from a subset of documents from HotpotQA and 4 query vector trajectories recorded during active retrieval on 4 questions. The document embeddings were made with MiniLM, which also supplied the target embeddings when training HAL that generated the trajectories. The colored trajectories correspond with the colored cross marks marking one of the correct gold document for that question. From the figure we can see that the predicted query vectors match poorly with the underlying distribution of embeddings of documents, being shifted away.

6

Discussion

The following chapter will discuss the results from Chapter 5 as well as draw conclusions and propose future work.

6.1 Utilizing Intermediary Representations

The results from these experiments suggest a good implementation of the approach described in Figure 1.2, i.e. layer 22 and the *normal* architecture. Furthermore, the fact that the results only rely on HotpotQA and not a synthetic dataset, makes it easier for these results to be compared to other systems. However, the results come with some caveats which will be discussed in the following section.

6.1.1 Layer analysis

Looking at Figure 5.1 there is a clear difference in MCRD depending on which layer H is attached to. It is interesting to see that both the early layers and the later layers have reduced MCRD, implying that there is an important balance required to find the optimum. The early layers having poor performance is reasonable, since the attention mechanism may not have been able to combine enough information from different parts of the document. H only uses the representation of the last token, but creating a search vector likely requires information from many parts of the document. Towards the later layers the LLM might be aligning itself to the probability distribution of the next token prediction, instead of creating a representation for the entire text. However, the effects of this alignment does not seem to decrease the MCRD too much, since layers 24-32 are generally better than 2-20. This implies that the most important factor when selecting a layer is to avoid picking one that is too early.

Other than the general trends there are also some peculiar sharp changes in MCRD between pairs of layers. For instance, the MCRD increases quickly between layer 18 to 20. This implies that those layers do something important for the task of embedding. One might have expected that the middle layer (i.e. layer 16) would have give the best MCRD, but due to this large increase it actually performs worse than almost all subsequent layers. For larger values of k there are also some decreases in MCRD at layer 8 to 10 and 16 to 18. It does make intuitive sense that the MCRD could fluctuate between layers, since the objective of the underlying LLM

is to produce a next token prediction, not to embed the document. However, we assume these increases and decreases in MCRD is a product of the specific LLM used; using an LLM other than Mistral-7B will probably change where they appear.

It is difficult to conclude how transferable these results would be to other datasets and environments. Due to storage restraints the training was only done on 10% of the HotpotQA training set. It is clear that this decreased the overall MCRD, since $k = 4$ only reaches a MCRD of 0.13, while it reaches 0.31 in Figure 5.3 where all the data was used. However, the relative differences in MCRD between layers could still be the same. They could change slightly, since specific layers could be more susceptible to overfitting which would be alleviated with more data. Furthermore, the optimal layer could be dependant on the specific type of tokens the query vector is inferred from; In this experiment the answer tokens were used but in the rest of the report H's training data consists of CoT-tokens.

Regardless of changes in the MCRD in specific layers, it is reasonable to assume that the optimum would be *near* layer 22. This is both due to the empirical results in this thesis, and the intuitive justification that early layers may not provide enough attention between tokens to take the whole document into consideration.

6.1.2 *H* architecture experiments

Figure 5.2 shows how the MCRD increases from *small* to *medium* to *large*, i.e. as the number of neurons increases in the single hidden layer. Therefore, it could be interesting to try a model that is even bigger. Imagine a *huge* model which has 384×8 hidden neurons, double the size of the *large* model. Since the input dimension is 4096 this would result in over $4096 \times 384 \times 8 \approx 12\text{M}$ trainable parameters which may be excessive for a dataset only containing $\sim 100k$ questions. This could make the model susceptible to overfitting.

Looking at the loss plot in Figure 5.2 the *large* model which had the highest MCRD does not have the lowest dev loss. It did have the lowest train loss, meaning that there are signs of overfitting. Since MCRD is the most important metric to optimize, it is interesting that it doesn't follow the same ordering as the dev losses. Simply put, the loss (mean squared error loss) measures the distance in vector space from the gold query vector, while the MCRD measures *if* the gold vector was found when looking at the k closest vectors to the search. These metrics measure the performance of the model in slightly different ways, meaning that it would be reasonable for them to differ slightly. One explanation could be that the vector space of embedded documents have dense clusters of highly related documents. A model that is slightly incorrect when searching in these clusters could still have a very low loss, but since there are many documents close together it often returns the wrong documents. With this interpretation, the discrepancy in MCRD and loss values may indicate what kinds of documents the model gets incorrect.

This thesis used a MLP attached to the last token at one particular layer. There are many other possible approaches, such as using multiple layers or using more than one token through some sequence model. One thing that was attempted was to design

H as a single transformer block that could use every representation in the sequence at layer 22. The transformer approach was abandoned because it gave worse results compared to a simple MLP while simultaneously being more difficult to work with. However, it is closer to GRIT (see Chapter 2) or running two separate models, in the sense that training the embedder now also involves modifying transformer block parameters.

6.1.3 Comparison to baseline

The results in Figure 5.3 show that H reaches roughly a third the MCRD of the baseline. The case with 4 random documents at $k = 4$ reaches 0.31 MCRD, while giving 4 random documents to the baseline predictably gives 0 MCRD, indicating that our method can ignore the irrelevant context and infer from the question alone which documents should be retrieved. This would be a requirement for active retrieval since such a system would have to search based on the recent topic and ignore the old tokens. If active retrieval were to be implemented with MiniLM it would require a hypervariable specifying how many of the latest tokens to include in the embedding. However, in the scenario where only the question is embedded to retrieve the correct documents, the baseline outperforms our approach. This implies that our approach should not be used in non active retrieval systems.

Considering research question 1, these results show that it is possible to create search vectors by combining H (the query predictor) with M (the LLM), but the retrieval performance is going to be worse. However, Figure 5.4 indicate that the inference time and memory usage will decrease.

One approach that could mitigate the reduced MCRD in an active retrieval system would be to use the output from H to instead indicate that a new search is required, but to perform the actual search with a separate embedder. The MCRD of H is reasonable and it is essentially free to run as long as tokens are being generated. Therefore, it may be the case that as the LLM changes subject, the output from H also changes rapidly. These changes could be measured by comparing the distance to the previous search vector (or rolling average of multiple vectors) to check if a new search should be performed. The benefit of such a system would be that the MCRD would be the same as the separate embedder, while analyzing if the LLM changed topic and requires new information, would be very cheap to evaluate. However, there would still need to be some hypervariable deciding how much of the current text to provide to the separate embedder for every search.

Even though the inference time is much faster on H, this may not be a problem since both models could be run in parallel, provided that there are enough computational resources to do so. However, the baseline would have to recalculate the entire sequence for every embedding, while H can utilize the cache in Mistral-7B, thereby reducing the time complexity from quadratic to linear. Therefore, the difference in inference time would increase as the size of the documents increase.

Similarly, the memory usage of H is also much smaller than that of the baseline, but they are both very small in comparison to an LLM like Mistral-7B. A reduction

from 87 MB to 14 MB is probably not too important when analyzing memory at the scale of LLMs. However, it should be noted that MiniLM is a rather small embedder model which is designed for English text. It would be interesting to analyze the retrieval performance on more difficult, multi-lingual datasets. It could be the case that intermediary representations from Mistral-7B could handle multiple languages (since it is a much bigger model) while MiniLM would have to be exchanged for a larger multi-lingual embedder.

While HotpotQA is beneficial because it allows the architecture to be tested in isolation, the fact that there are two gold documents does cause some issues. The architecture is supposed to search based on CoT steps since HotpotQA does not contain labeled CoT steps the search is done based on the answer tokens. As described in Section 4.1 the architecture was adapted to produce two search vectors. Since the two gold documents could be slightly different there should ideally be a way to search twice using different internal representations. Basically, these experiments are a bit more analogous to a one-time retrieval system, while the goal is to use the system for active retrieval.

6.2 Synthetic Multi-hop Questions

Creating synthetic multi-hop questions with an LLM will inevitably require many choices which will affect the final result, such as the prompt template, strategy for picking and cleaning documents, and the method for pairing documents. This section will discuss the results of the data generation pipelines, as well as analyze the benefits and drawbacks of our methods.

6.2.1 Question generation pipeline

As mentioned in Section 1.4 the main focus of this thesis was not to create a flawless set of documents. Therefore, Wikipedia articles linking to **Algorithms** were used. However, some of the articles in the dataset turned out to be discussions between users, which is a different type of document compared to regular, factual articles on Wikipedia. This meant that some of the questions generated (such as the example from Figure 5.6) were based on user discussions. In order to improve the questions, more time could have been spent on cleaning up the initial set of documents. However, the quality of the questions or answers are not the most important part for training H. Training H only requires correctly labeled CoT steps. Having a question that is relevant is likely also beneficial since it is included in the prompt, but the factuality of the documents or the truthfulness of the answer is probably not as important.

Creating a good multi-hop question requires (at least) two documents with information that is sufficiently related. The strategy to pair documents was to use their distances in vector space, which does not guarantee that they contain facts that can be used for a multi-hop question. This explains why some of the questions turned out to be single-hop questions (as seen in Figure 5.11); sometimes the documents provided make it impossible to create a multi-hop question. One attempted remedy

for this was to allow the model to respond with `INVALID`, which happened in 5196 out of 38741 cases as seen in Figure 5.5. However, due to the complexity of the task and the relatively small size of the LLM, this did not solve all those cases.

This data generation method is similar to a previous study [19] described in Chapter 2. Note that their paper was not released at the start of this thesis, which means that our approach is not based on theirs. Looking at just the quality of the questions and answers (ignoring CoT steps) their approach contain two key differences which could improve the quality of the data. Firstly, they pick an answer before generating the question using a more complex method involving a separate named entity recognition model. This can be used to ensure that the answer is simple and factual instead of nuanced and exploratory, making it more useful in evaluation; it is easier to grade short answers than long answers. Secondly, by double checking that answering the question produces the original answer, the questions and answers are more likely to be correct. Taking their work in relation to ours, we note that theirs does not contain reference-labeled CoT steps. It would be interesting to extend their approach with this, to create an active retrieval-oriented dataset like ours (AL, PL).

6.2.1.1 Analysis of example question

The quality of the synthetic questions were varied, with some questions having good quality and some suffering from hallucinations. Looking at the example question in Figure 5.6, there are some issues with the question. First off, the question explicitly mentions "[...] in the documents" which is an artifact from how our prompt is structured. This is not something that should be needed; the system should still search for documents regardless of if it is prompted or not. The question also asks for emails specifically but the second document writes about private messages. However, the system has correctly produced a question that relies on information from both documents. Furthermore, the CoT steps do distill the important information from the documents and the answer does follow from those steps. Since the most important part for training H is the labeled CoT steps, this question would work well for that purpose.

6.2.2 Attention-based relabeling

Considering Figure 5.8, there are a large number of multi-hop questions in PL that became single-hop questions in AL. This shows a major discrepancy between the labels in the two approaches. One explanation for this is that PL hallucinates and AL fixes those hallucinations. However, an LLM can have attention to a token for reasons other than factual dependencies. One possibility is that the model has attention to a document for reasons relating to style or structure, instead of facts. Furthermore, it may be more likely to attend to earlier parts of the prompt, resulting in document 1 receiving disproportionately more attention (or vice versa). It would be interesting to analyze whether swapping the order of the documents has any effect on the attention labels.

Another issue with AL is that it may be impossible to correctly source the early tokens in a CoT step. For example, consider the documents and the question from

Figure 5.6. If a CoT step starts with "Canceling something sent through [...]", it is impossible to know which of the documents those tokens should relate to. The phrase which differentiates the two documents, in this case "email" or "PM", may not appear at the start of the CoT step. Therefore it would be interesting to analyze techniques to improve this, possibly by ignoring tokens where the attention seems too evenly distributed between the documents.

6.2.3 Training H on the datasets

Considering the results from Figure 5.9, the results are still fairly close. Even though AL results in higher MCRD, it could simply be due to random chance. Furthermore, it should be noted that the evaluation is done on two separate dev sets, one based on AL and one based on PL. This means that the comparison is not entirely fair. While these results cannot be used to conclude that AL is more correct, they do show that AL is easier for the system to learn from. This may imply that AL has a reduction in noise and contains more predictable labels compared to PL.

6.2.4 Manual evaluation on a random sample

While the human evaluation only contained a small sample size of 50 questions, it does show a clear difference between the correctness of the two datasets with AL and PL getting 25 and 15 questions correct respectively. However, looking at Figure 5.11 there is a clear trade-off between the types of errors the datasets contain. AL is better at correctly labeling single-hop questions as single-hop, while PL is better at labeling multi-hop questions as multi-hop.

The trade-off between two types of errors is similar to the trade-off in classification systems between sensitivity and specificity. If a model outputs a probability, but a binary answer is required, a probability threshold can be picked which makes it possible to bias the system towards one type of answer. It may be possible to implement a similar bias for attention-based relabeling, letting the developer choose between getting more multi-hop questions correct or getting more single-hop questions correct. However, the outputs produced by attention-based relabeling do not represent multi-hop vs single-hop. Instead, they represent how much attention is given to a specific document at a given token. By artificially tweaking those outputs it would be possible to add a bias towards one of the documents. A system could therefore be constructed to intelligently bias towards different documents at different CoT steps creating an overall bias towards multi-hop or single-hop questions.

Considering research question 2, the results show that it is possible to generate a large number of questions based on only a document database. However, our approach did not guarantee that every generated question was a multi-hop question, resulting in many single-hop questions being generated. Furthermore, getting the correct reference-labels for the CoT steps was difficult through just prompt engineering, but was drastically improved through attention-based relabeling. Still, the reference-labels are not perfect, since AL agrees with a human annotator 68% of the time.

6.3 Active Retrieval

This section discusses the results from the experiments in building an active retrieval system from Mistral-7B and the H-model as a retriever.

6.3.1 Inference with Active Retrieval

In large our experiments in building an active retrieval system based on our architecture are inconclusive. As shown in the results in Section 5.3 (and discussed more in the following subsections), there could be a bug in the implementation that did not allow the system to reach the retrieval performance expected from the results in Section 5.2.3.

It is unlikely that the poor retrieval performance was caused by the specific retrieval schedule (retrieving every $t = 20$ tokens generated), since the retrieved documents are as poor of a match even in the first retrieval when only one token has been generated (see Figure 5.12a and Figure 5.12b). In the same way, it is unlikely that the Mistral-7B inference implementation was the cause of the poor HotpotQA results. As the generating LLM was rarely provided with the correct documents, it was unable to ground its answers.

Had the retrieval performance been as expected, though, it would still have been worse than the baseline as shown in Section 5.1.3 in the search-once, static retrieval case. The proposed benefit of our architecture for the active retrieval use-case would be the lack hyperparameters for the retriever. Other methods like FLARE, RETRO, all require a hyperparameter that sets the size of the window of past tokens used to retrieve to base the similarity search on.

6.3.2 Experiment on HotpotQA

To give our active retrieval implementation a better opportunity to formulate its answers, and search for new documents, we used CoT-prompting. Additionally, we gave the model an instruction to give a short 1-3 word answer in json format for easy extraction from the raw response. It is uncertain if this method yields the best answer accuracy, for the same reason the other results in this section are inconclusive - low retrieval quality. It is possible that the active retrieval use-case favors some other kind of prompting strategy, for instance one which notes facts as if on a scratch card when exploring the space of documents to use. Either way, the results on HotpotQA should not be taken as a larger indication of the efficacy of active retrieval, rather as a sign that to answer HotpotQA well you need an accurate retriever.

6.3.3 Analysis of Query Convergence

These experiments were performed to analyze the trajectory of predicted query vectors over time, though largely instead became indicators for possibly buggy behaviour.

6.3.3.1 Tracking euclidean and cosine distance to target documents during active retrieval inference

It can be seen from Figure 5.12 that the retrieved documents start off as a poor match in relation to the correct documents. Furthermore, as the AR generation progresses and documents are switched out for new ones, the distance between the retrieved documents and the correct documents increases. This could be an indication that active retrieval does not work, though a more likely explanation is faulty code in the implementation of the retriever. Related work has shown that active retrieval is possible and beneficial, and results from Section 5.2.3 suggest better possible performance than what was observed in this experiment. While the results on our own AL dev set had 0.23 MCRD, this experiment achieved 0.00028 MCRD, 3 orders of magnitudes off and close to random chance at 0.00004 MCRD, supporting the bug hypothesis. Hence, it cannot be concluded if the queries predicted and retrieved documents would converge or diverge in relation to a reference solution over time in the generated answer.

6.3.3.2 t-SNE-projection of the trajectory of predicted query vectors

Figure 5.13 shows a large distributional shift between the predicted query vectors (the trajectories) and the embeddings of the correct documents (cross marks). The embedding model used for both the documents and acting as target for training the H-model was MiniLM. Hence, the discrepancy between the distributions of the predicted query vectors and document embeddings is not simply because they are and should be different distributions. In the ideal (and trained for) case, the distribution of predicted query vectors should mimic the distribution of document embeddings. However, the HAL-model used in this experiment only involved training data from the AL dataset. Hence, the HotpotQA data seen during evaluation acts as a test set which could explain some distributional shift, especially since the synthetic data is limited to topics on algorithms as mentioned in Section 4.2.1. Though, given the extent of the distributional shift this is likely not the sole explanation.

6.4 Conclusion

In this thesis we have investigated an approach for combining the embedder and generator in a RAG system, as well as how to generate synthetic data to train that system. The following section will cover how our findings relate to the research questions in Section 1.3, what the implications of these results are, as well as provide some avenues for future work.

6.4.1 Summary of findings

Research question 1 asks whether the query prediction mechanism H can be combined with the generator to reduce memory usage and inference time. Our results show that this is possible; intermediary representations in Mistral-7B do contain relevant information for constructing embeddings. Furthermore, our system is 27

times faster and uses 6 times less memory when running on a sample from HotpotQA. More importantly, for active retrieval systems our approach reduces the computational complexity of generating query vectors from quadratic to linear by utilizing the cache from the LLM. Furthermore, if we consider the intermediary representation free, as the llm will be run in a RAG system anyway, the computational complexity of H is constant in respects to the sequence length. However, our approach also leads to a 3 times reduction in MCRD making it difficult to deploy in an active retrieval system.

Research question 2 asks whether multi-hop questions with answers and reference-labeled CoT reasoning can be generated with an LLM as opposed to manually constructed. We have shown that such data can be generated synthetically, however there were issues with the correctness of the reference-labels when these were obtained through prompting the LLM. The labels could be improved through attention-based relabeling which resulted in more correct labels according to human-labeled ground truth. Being able to generate these questions synthetically removes the cost of human annotators.

6.4.2 Implications

By analyzing how the intermediary representations at different layers result in varying MCRD, we provide some insight into how LLMs operate internally. This may be relevant for the interpretability of LLMs since it shows how the internal representation becomes more and less similar to the more pure representational embedding space of an embedding model.

Our approach did reduce the MCRD compared to the baseline, but GRIT (see Chapter 2) showed that accurate embedding models can be created while still being combined with a generating model. GRIT trained both the embedding and generation simultaneously while we limited us to only training the embedder H. The discrepancy in accuracy may imply that in order to combine the models successfully, you have to back-propagate though all weights rather than only through an additional attached head. The fact that the MCRD changed dramatically depending on the layer used shows that Mistral-7B does something similar to an embedding model internally. However, it may not be enough to achieve a high MCRD without allowing the transformer blocks to also be trained for embedding.

Our attention-based relabeling approach showed success in decreasing hallucinations and finding the true reference-labels. This shows that analyzing the attention of an LLM can be a useful tool to find sources for some claim. However, the reference-labels are not perfect, illustrated by the fact that AL assigns the same labels to a question as a human annotator 68% of the time. Regardless, we believe that analyzing attention can be a useful tool to develop LLMs that generate more factual and interpretable answers.

6.4.3 Future Work

A system that can cheaply generate a search vector for every token allows for greater flexibility in deciding when documents should be retrieved. Instead of searching after a fixed number of tokens or sentences, the system could dynamically decide when to retrieve documents based on changes in the search vector. An active retrieval system could be developed which performs retrieval whenever the current search vector is sufficiently different from the previous, as this may indicate a change in topic. As an alternative, a larger separate model could be used to perform the actual search after the need for new information has been indicated with the smaller H-model.

The synthetic data generation from both us the related work [19] described in Chapter 2 used Wikipedia, but both approaches could be applied to other sets of documents. The text in factual articles on Wikipedia may be different from that of guides, legal documents, or internal company documentation. Applying these approaches to other document sets may reveal unique challenges caused by changes in style or format. An analysis could be done studying how well data generation techniques that work on Wikipedia transfer to other domains.

The data from our synthetic data generation pipelines could also be used to fine-tune an existing embedding model. For example, applying it to proprietary company documentation could generate questions, answers and CoT steps for that dataset. By fine-tuning an separate embedding model (e.g. MiniLM or similar) on this data, a more traditional active retrieval system could be constructed. The system would still be able to retrieve based on the CoT steps and the embedding model may be more accurate when it comes to company specific terms, but the embedding model and generating model would remain separate. This system would not benefit from the decreased memory usage or inference speed, but may achieve higher retrieval accuracy and thereby provide better answers.

Studying attention may be a useful tool for document re-ranking in RAG systems. As has been shown in this thesis, attention can be analyzed to provide insight into which document is most relevant for a claim. Therefore, a system could be constructed which ranks documents based on how relevant they are to a claim, allowing a RAG system to only receive the most relevant documents. It would be interesting to analyze how this approach compares to other methods of re-ranking.

Attention could be used to decide which documents should be exchanged for new ones in an active retrieval system. If a document receives low attention it may no longer be relevant for the current topic, meaning that it should be exchanged for a new one. This idea could be used to develop an active retrieval system with a Least Recently Used (LRU) cache, ensuring that irrelevant documents get removed and relevant documents are kept in the context.

Bibliography

[1] X. Hu *et al.*, *Do large language models know about facts?* 2023. arXiv: 2310.05177 [cs.CL].

[2] Z. Xu, S. Jain, and M. Kankanhalli, *Hallucination is inevitable: An innate limitation of large language models*, 2024. arXiv: 2401.11817 [cs.CL].

[3] P. Lewis *et al.*, *Retrieval-augmented generation for knowledge-intensive nlp tasks*, 2021. arXiv: 2005.11401 [cs.CL].

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[6] S. Borgeaud *et al.*, "Improving language models by retrieving from trillions of tokens," in *International conference on machine learning*, PMLR, 2022, pp. 2206–2240.

[7] G. Chen, W. Yu, and L. Sha, *Unlocking multi-view insights in knowledge-dense retrieval-augmented generation*, 2024. arXiv: 2404.12879 [cs.CL].

[8] Y. Tang and Y. Yang, *Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries*, 2024. arXiv: 2401.15391 [cs.CL].

[9] Z. Jiang *et al.*, *Active retrieval augmented generation*, 2023. arXiv: 2305.06983 [cs.CL].

[10] Y. Gao *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[11] P. Zhao *et al.*, "Retrieval-augmented generation for ai-generated content: A survey," *arXiv preprint arXiv:2402.19473*, 2024.

[12] A. K. Heger, L. B. Marquis, M. Vorvoreanu, H. Wallach, and J. W. Vaughan, *Understanding machine learning practitioners' data documentation perceptions, needs, challenges, and desiderata*, 2022. arXiv: 2206.02923 [cs.HC].

[13] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[14] C. S. Krishna, *Prompt generate train (pgt): Few-shot domain adaption of retrieval augmented generation models for open book question-answering*, 2023. arXiv: 2307.05915 [cs.LG].

[15] O. Honovich, T. Scialom, O. Levy, and T. Schick, *Unnatural instructions: Tuning language models with (almost) no human labor*, 2022. arXiv: 2212.09689 [cs.CL].

[16] Z. Yang *et al.*, *Hotpotqa: A dataset for diverse, explainable multi-hop question answering*, 2018. arXiv: 1809.09600 [cs.CL].

[17] N. Muennighoff *et al.*, *Generative representational instruction tuning*, 2024. arXiv: 2402.09906 [cs.CL].

[18] S. Yao *et al.*, *React: Synergizing reasoning and acting in language models*, 2023. arXiv: 2210.03629 [cs.CL].

[19] M. Chen, X. Chen, and W.-t. Yih, *Few-shot data synthesis for open domain multi-hop question answering*, 2024. arXiv: 2305.13691 [cs.CL].

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[21] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[22] A. Karpathy, *Let's build the gpt tokenizer*, YouTube video, 2024. [Online]. Available: https://www.youtube.com/watch?v=zduSFxRajkE (visited on 05/13/2024).

[23] S. Land and M. Bartolo, "Fishing for magikarp: Automatically detecting under-trained tokens in large language models," *arXiv preprint arXiv:2405.05417*, 2024. [Online]. Available: https://arxiv.org/pdf/2405.05417.

[24] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[25] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[26] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[27] M. AI, *Introducing llama 3: The next generation of large language models*, https://ai.meta.com/blog/meta-llama-3/, Accessed: 2024-05-14, 2024.

[28] G. Team *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[29] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] S. Jain and B. C. Wallace, *Attention is not explanation*, 2019. arXiv: 1902.10186 [cs.CL].

[31] C. Zhang, S. Li, M. Ye, C. Zhu, and X. Li, *Learning various length dependence by dual recurrent neural networks*, 2020. arXiv: 2005.13867 [cs.NE].

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[33] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[34] W. Kwon *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23, , Koblenz, Germany, Association for Computing Machinery, 2023, pp. 611–626. DOI: 10.1145/3600006.3613165. [Online]. Available: https://doi.org/10.1145/3600006.3613165.

[35] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, *Generative agents: Interactive simulacra of human behavior*, 2023. arXiv: 2304.03442 [cs.HC].

[36] V. Rawte, A. Sheth, and A. Das, "A survey of hallucination in large foundation models," *arXiv preprint arXiv:2309.05922*, 2023.

[37] M. Elaraby *et al.*, *Halo: Estimation and reduction of hallucinations in open-source weak large language models*, 2023. arXiv: 2308.11764 [cs.CL].

[38] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, "Retrieval augmented language model pre-training," in *International conference on machine learning*, PMLR, 2020, pp. 3929–3938.

[39] S. Jha, S. K. Jha, P. Lincoln, N. D. Bastian, A. Velasquez, and S. Neema, "Dehallucinating large language models using formal methods guided iterative prompting," in *2023 IEEE International Conference on Assured Autonomy (ICAA)*, IEEE, 2023, pp. 149–152.

[40] Z. Liu *et al.*, "Llm-qat: Data-free quantization aware training for large language models," *arXiv preprint arXiv:2305.17888*, 2023.

[41] S. Zhang *et al.*, "Instruction tuning for large language models: A survey," *arXiv preprint arXiv:2308.10792*, 2023.

[42] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, *Retrieval augmentation reduces hallucination in conversation*, 2021. arXiv: 2104.07567 [cs.CL].

[43] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, *Fever: A large-scale dataset for fact extraction and verification*, 2018. arXiv: 1803.05355 [cs.CL].

[44] J. Gu, Y. Wang, K. Cho, and V. O. Li, "Search engine guided neural machine translation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. DOI: 10.1609/aaai.v32i1.12013. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12013.

[45] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[46] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[47] IBM, *What is synthetic data?* Accessed: 2024-05-15, 2024. [Online]. Available: https://www.ibm.com/topics/synthetic-data.

[48] X. Guo and Y. Chen, *Generative ai for synthetic data generation: Methods, challenges and the future*, 2024. arXiv: 2403.04190 [cs.LG].

[49] R. Tang, X. Han, X. Jiang, and X. Hu, *Does synthetic data generation of llms help clinical text mining?* 2023. arXiv: 2303.04360 [cs.CL].

[50] P. P. Liang, C. Wu, L.-P. Morency, and R. Salakhutdinov, *Towards understanding and mitigating social biases in language models*, 2021. arXiv: 2106.13219 [cs.CL].

[51] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, Mar. 2023, ISSN: 1557-7341. DOI: 10.1145/3571730. [Online]. Available: http://dx.doi.org/10.1145/3571730.

[52] A. Q. Jiang *et al.*, *Mistral 7b*, 2023. arXiv: 2310.06825 [cs.CL].

[53] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, *Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers*, 2020. arXiv: 2002.10957 [cs.CL].

[54] llama-index, *Sentence splitter - llamaindex*, 2024. [Online]. Available: https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/sentence_splitter/ (visited on 05/17/2024).

[55] Ollama, *Ollama*, Accessed: 2024-06-10, 2024. [Online]. Available: https://www.ollama.com/.

A

Appendix 1

A.1 Synthetic Data Generation Prompt

These are two gold documents:
Document 1 on "{title1}":
−−−−−−−−−−−−−−−−−−−−
{content1}
−−−−−−−−−−−−−−−−−−−−
Document 2 on "{title2}":
−−−−−−−−−−−−−−−−−−−−
{content2}
−−−−−−−−−−−−−−−−−−−−
You are a Teacher/Professor.
Your task is to write 3 questions for an upcoming quiz/examination.
The students' task is to both search for the documents and answer the question.
The questions should be Multi−hop, i.e require reasoning over both gold documents above.
These are some examples of Multi−hop questions:
Type of question | Question | Answer |
Bridge Entity−based (temporal entity) | Who was the president of United States in the year in ↪ which Mike Tyson declared his retirement? | George W. Bush |
Bridge Entity−based (geographical entity) | What is the national bird of the nation that has a ↪ negative carbon footprint? | The Raven |
"type": "Comparison",
"question": "Who doted more on Ada Lovelace: Lady Byron or Lady Milbanke?",
"chainOfThought": [
"(Document 1): Ada was often left in the care of her maternal grandmother, who doted on ↪ her.",
"(Document 2): Lady Byron wrote anxious letters to Lady Milbanke about her daughter's ↪ welfare and referred to her as 'it', implying a lack of emotional connection."
],
"answer": "Lady Milbanke"

"type": "True/False",
"question": "Ada Lovelace's notes on the Analytical Engine contain the first computer program ↪ and she foresaw the capability of computers to go beyond mere calculating.",
"chainOfThought": [
"(Document 1): Ada Lovelace's notes on the engine include what is considered the first ↪ computer program.",
"(Document 2): Ada Lovelace also foresaw the capability of computers to go beyond mere ↪ calculating."
],
"answer": "True"

"type": "Analysis",
"question": "Analyze the numerical stability of Chan's method for estimating the mean when ↪ n_A is close to n_B and both are large. Compare it with the weighted incremental ↪ variance algorithm suggested by West.",
"chainOfThought": [
"(Document 1): Chan's method for estimating the mean is numerically unstable when n_A ↪ is close to n_B and both are large.",
"(Document 2): The weighted incremental variance algorithm suggested by West uses a ↪ different formula for computing population variance, sample frequency variance, and ↪ sample reliability variance.",
"(Document 1): In Chan's method, the numerical error in delta = mean_Y − mean_X is ↪ not scaled down when n_A is close to n_B and both are large, leading to potential ↪ instability.",

```
    "(Document 2): The weighted incremental variance algorithm suggested by West uses Bessel'
        ↪ s correction for population variance, which may help improve numerical stability."
],
"answer": "Chan's method for estimating the mean is numerically unstable when n_A is close to
    ↪ n_B and both are large due to the lack of scaling in the delta calculation. The weighted
    ↪ incremental variance algorithm suggested by West uses Bessel's correction, which may
    ↪ help improve numerical stability."


Generate 3 multi−hop questions that fit the given documents, or answer "INVALID" if no
    ↪ question can be made.
For each question, structure the JSON object as follows:\n
{
    "questions": [
        {
            "id": 1,
            "type": "<type of question>",
            "question": "<question text>",
            "chainOfThought": [
                "<reasoning step 1 with citations e.g., '(Document 1): ...' or '(Document 2):
                    ↪ ...'>",
                "<reasoning step 2 with citations e.g., '(Document 1): ...' or '(Document 2):
                    ↪ ...'>"
            ],
            "answer": "<short 1−3 word answer if possible>"
        },
        {
            "id": 2,
            "type": "<type of question for question 2>",
            "question": "<question text for question 2>",
            "chainOfThought": [
                "<reasoning step 1 with citations for question 2>",
                "<reasoning step 2 with citations for question 2>"
            ],
            "answer": "<short 1−3 word answer if possible>"
        },
        {
            "id": 3,
            "type": "<type of question for question 3>",
            "question": "<question text for question 3>",
            "chainOfThought": [
                "<reasoning step 1 with citations for question 3>",
                "<reasoning step 2 with citations for question 3>"
            ],
            "answer": "<short 1−3 word answer if possible>"
        }
    ]
}
```