Web-based Retrieval Summarization

Abb, Jacob Meißner, Michel Thiel-Peters, Bahne Schümann, Lea
Zupke, Alexander

September, 30, 2024

1 Introduction

As the demand for more sophisticated and contextually aware models in natural language processing (NLP) increases, generative models encounter significant limitations. These models rely on static knowledge encoded during training which makes them less effective for applications that require dynamic access to external, up-to-date or domain-specific information.

The introduction of Retrieval-Augmented Generation (RAG) addresses these demands by combining the generative power of large language models (LLM) with an external retrieval mechanism. By augmenting the generation process with relevant data sources retrieved at inference time, RAG enhances the overall quality of the generated output (Radford and Narasimhan, 2018). This paper explores the application of RAG for processing question-answering (QA) pairs, focusing on optimizing performance through various prompting, embedding and chunking strategies within a modular architecture. We designed a flexible system to experiment with static and contextualized vector embedding methods, capturing more nuanced semantic relationships between questions and text segments. Additionally, we investigated diverse chunking approaches, adjusting text granularity and context windows to optimize retrieval effectiveness. In our work, we also experimented with various retrieval techniques to enhance the system's ability to identify and retrieve the most relevant content. To further improve reasoning capabilities, we finally implemented a Chain of Thought (CoT) model and compared its performance to our base model. Our results demonstrate that this modular RAG approach, along with a careful selection of parameters and techniques, significantly improves both retrieval accuracy and answer generation quality in QA tasks, offering insights into best practices for real-world applications.

1.1 Related Work

NLP has seen rapid advancement, driven by breakthroughs in machine learning and neural network-based approaches (Manning et al., 2008). The limitations of these approaches in capturing the nuanced semantics and context of natural language soon prompted the development of word embedding models, such as Word2Vec (Mikolov, 2013) and GloVe (Pennington et al., 2014). By representing words in a compressed form as vectors, so-called embeddings, these models capture complex semantic relationships, enabling better generalization across NLP tasks. Despite the success of word embeddings, one limitation was that they produced static representations of words, where each word had a fixed vector regardless of its context. This gap was addressed with the introduction of the transformer architecture (Vaswani et al., 2017). The transformer model allowed for dynamic, context-dependent representations of words, considering the entire sequence of input tokens.

Building upon the transformer architecture, the NLP field saw a shift with the rise of LLMs such as BERT (Devlin et al., 2019), GPT (Radford and Narasimhan, 2018) and many more over recent years. These models introduced a new paradigm, where models are first pre-trained on massive amounts of text data to learn general language representations, and then fine-tuned for specific downstream tasks. The development of LLMs represents a convergence of advancements in embedding models, transformer architectures, and efficient training techniques, leading to models that can perform well on a variety of NLP tasks with minimal additional supervision (Zubiaga, 2024). The usage of LLMs for NLP task, however, comes with its own challenges. The complexity of the models may lead to a generation of answers that are difficult to explain. When a model makes predictions that deviate from common sense or outputs false information, it is known as hallucination. RAG is a promising tool to mitigate occurrences of that behaviour (Lewis et al., 2020). Controlled experiments have shown that hallucinations in LLMs are most often induced by premise predicates which deviate a lot from the training data (McKenna et al., 2023). With the

help of chain-of-thought based patterns (Wei et al., 2023), LLM are guided to produce answers that contain multiple reasoning steps. RAG extends the knowledge base and diversifies information sources in such a way that the model does not solely rely on its training corpus.

2 Methodology

2.1 Dataset and Metrics

The dataset used for the evaluation of our model is the *CRAG dataset V3* provided by Meta, 2024. It contains 2706 samples and encompasses the following information:

interaction ID, **query time**, **domain** (Finance, Music, Sports, Movie, Open), **answer**, **alternative answer**, **validation and test split indicator** and **search results**. The search results contain up to 5 HTML pages for each query, including page name, URL, snippet, full HTML, and last modified time. These contents are likely relevant to the question, but relevance is not guaranteed.

There are several factors influencing the quality of the generation of answers. For this portion of our report, we want to focus on the data-driven aspects. The quality of the augmentation is important to consider when trying to enhance generation. On the other hand, the "difficulty" of the questions plays an important role in the evaluation of model performance. First, we inspected the HTML pages provided by the dataset. The handling of HTML layout and elements is tackled using the **Reader** segment of the model. In order to provide the LLM with information from these pages we have to identify relevant material in regards to the respective question. Sentences that read like definitions or facts are promising candidates for answering the respective questions. We thus inspected the top 10 most used URLs from the provided web pages, as listed in 1. The quality

Page name	Frequency
Wikipedia	1723
IMDB	442
Investopedia	417
ESPN	271
Britannica	213
NASDAQ	205
Billboard	151
Quora	143
Statista	142
Screenrant	139

Table 1: Most common URLs in the provided search results

of the articles can not be determined by URL alone, but it is apparent that most of the referenced URLs are Wikipedia articles, domain-specific statistics or digital news outlets. While *Britannica* for example is fact-checked, it is important to note that *Quora* is a question-answer forum and does not have to provide reliable information, even if formulated as a definition or fact.

Finally, we enhanced our evaluation process by including a filter for different question types. The amount of samples with a specific question type can be seen in Figure 1. We differentiate between questions of type **simple**, **simple with condition** (simple_wc), **comparison** (comp), **aggregation** (agg), **false premise** (false_pr), **set**, **multi-hop** (m-hop) and **post-processing** (post-pr).

Another aspect is the rate of change expected for a QA pair. This is especially relevant in the domains finance and sports. The corresponding distribution is shown in Figure 2. We compute the **cosine similarity** during our retrieval process in the **Retriever** module of the model. Cosine similarity quantifies how alike two vectors are within an inner product space. It is calculated by finding the cosine of the angle between the vectors, indicating how closely they align in direction. This method is commonly applied in text analysis to assess the similarity between documents.

The metrics used for the **evaluation** were adapted from the Meta challenge, as provided for local evaluation. The score of the model is calculated as follows:

$$score = \frac{(2 \cdot n\_correct + n\_miss)}{n} - 1,$$
$$exact\_accuracy = \frac{n\_correct\_exact}{n},$$
$$accuracy = \frac{n\_correct}{n},$$
$$hallucination = \frac{(n - n\_correct - n\_miss)}{n} \text{ and}$$

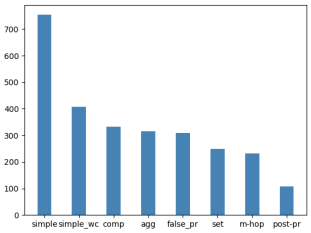

Figure 1: The distribution of available question types.

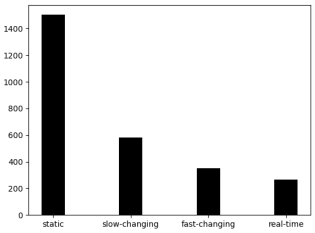

Figure 2: The expected rate of change for the given questions.

$$missing = \frac{n\_miss}{n},$$

where *exact_ acurracy* is the amount of word by word correct answers and counts towards *n_ correct*, the number of correct answers total. The correctness of an answer (True/False) is evaluated by the LLM, returning if the question has been answered, when compared to the provided gold standard answer. The *miss* rate considers the answers of the LLM of the kind "I don't know", as a response due to missing information or false premises. The *hallucination* rate considers the responses that do not answer the question or wrong responses.

2.2 Architecture

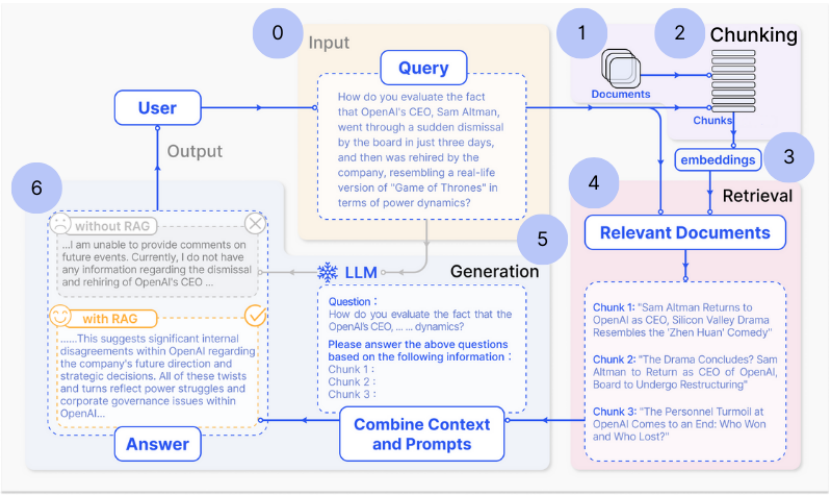

Figure 3: Naive RAG flow. Step 0: The system receives a query from the user, along with a timestamp. Step 1: The already given documents are pre-processed and transformed into a suitable format. Step 2: The pre-processed documents are split into smaller text parts called "chunks." Step 3: The chunks are embedded into compact, comparable representations. Step 4: The system retrieves/selects the chunks most similar to the query. Step 5: A prompt is constructed using the query and selected chunk texts as references. Step 6: The LLM generates a final answer using the references, giving a more informative answer. Modified image from Y. Gao et al., 2024

We have found the survey paper by Y. Gao et al., 2024 and the open-source RAG system described by Slocum and Schmuhl, 2024 to be useful resources for guiding the design of our architecture and providing a comprehensive review of the topic. Building upon the insights from these works, the general RAG process is illustrated in Figure 3. Initially, the system is given a query by a user which automatically includes a timestamp. This may be relevant for a time-critical question, such as "What is the current gold price?". Then the system searches documents relevant to that query. In real-world scenarios, this would be done by searching the web or retrieving suitable documents from databases. In the challenge scenario, the documents are already provided as HTML web pages. Chunking the documents is necessary, due to the limited context size of LLMs. We embed these chunks to give them a compact and comparable representation. This enables the system to retrieve the chunks that are most similar to the query, which in turn are more likely to be relevant.

Prompting the LLM with these chunks, the LLM should be able to generate a more informative answer than without the given references.

As we can observe from this flow, the following components exist in our system: a **reading** component, a **chunking** component, an **embedding** component, a **retrieval** component and a **generation** component. Though the presented process flow is mainly sequential, the components have a modular design so that they can be used in a non-sequential way. For instance, the chunking component can make use of the embedding component or the generator can be set up such that it can iteratively follow the process of: Retrieval → Generation → Retrieval → Generation $\cdots$. More details regarding these setups are presented in 2.3.

2.3 Approach

Evaluation
All of our evaluations use a sample of 100 queries to assess their performance in the RAG architecture. In addition, we used the following configuration: BasicReader, QueryExpansionManager, SemanticChunkerManager, LlamaEmbeddingManager, BasicRetrieverManager and BasicGenerator (local_evaluation.py). We set the percentile for semantic chunking to 77 and limited the retrieval to the top 10 chunks.

Reading Manager
The reading manager converts HTML web pages into plain text, preserving the semantic structure while removing tags and extracting table content. It uses BeautifulSoup [1] to parse various HTML elements, such as paragraphs, lists, and both traditional, as well as div-based tables.

Chunking Manager
The chunking part of our approach is used to split the content of the given documents into several chunks, which are later passed to the LLM. Therefore, we made use of three different chunking methods. In **Basic Chunking (BC)**, the text is split into sentences, and a sliding window technique is employed, where the window moves over the text and merges adjacent sentences together into one chunk. Two parameters are used during that process. The *Step Size* dictates how far the window moves with each iteration. To specify the number of sentences included in each chunk, the *Chunk Size* is used. A larger *Chunk Size* can capture more context but may lead to challenges in maintaining coherence. For a *Step Size* smaller than the *Chunk Size*, decreasing the *Step Size* results in an increase of the overlap of two chunks, which can provide richer contextual information but at the cost of increased redundancy.
To assess the performance of BC, we conducted two different evaluations.

Step Size	Chunk Size	Score	Accuracy	Hallucination	Missing
10	20	**0.06**	**0.32**	**0.26**	**0.42**
20	30	0	0.28	0.28	0.42

Table 2: Evaluation metrics for two different parameter combinations of *Basic Chunking*.

The results in Table 2 indicate that the combination of a *Step Size* of 10 and a *Chunk Size* of 20 yielded the highest evaluation score of 0.06. The tests indicate that smaller *Step Sizes* enhance contextual overlap, which improves output quality. A smaller *Chunk Size* combined with a reasonable *Step Size* effectively retains context and minimizes redundancy. Larger *Chunk Sizes* do not necessarily yield better performance, suggesting diminishing returns on information capture. Beyond a certain threshold, increased *Chunk Sizes* can introduce noise, undermining the effectiveness of retrieved information (Wu et al., 2024).

We also explored a more advanced chunking technique called **Semantic Chunking (SC)**, which is designed to group text into meaningful segments based on semantic similarity. Unlike *BC*, *SC* leverages embeddings and cosine similarity to determine the natural breakpoints between semantically distinct sections of the text. This ensures that the content within each chunk is contextually cohesive. The core of *SC* is driven by sentence embeddings. The text is first split into sentences and then the semantic similarity between adjacent sentence groups is calculated using cosine similarity. By analyzing the cosine distances, breakpoints are determined where the semantic gap between sentences is largest, thus allowing the model to chunk the text based on natural boundaries in meaning, rather than fixed intervals. To determine these semantic gaps, the percentile of cosine distances between sentence groups is calculated. A higher percentile leads to

[1] BeautifulSoup is a Python library used for parsing HTML and XML documents: https://pypi.org/project/beautifulsoup4/

fewer, larger chunks, while a lower percentile results in more, smaller chunks (Santhosh, 2024).

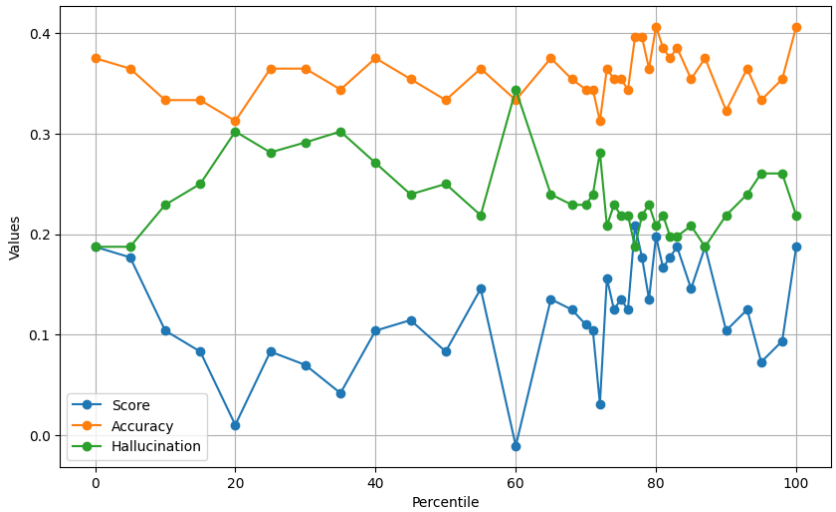

Figure 4: Evaluation metrics across different percentiles for *Semantic Chunking*, showing the variations of performance based on chunk size.

Figure 4 demonstrates the model's performance across varying percentiles, which dictate the average chunk size. At very low percentiles, where chunk sizes are minimal, the model may rely heavily on its internal knowledge due to insufficient context. This can result in higher hallucination rates and lower overall performance. In general, the model's performance exhibits some instability, which is caused by the non-deterministic nature of the LLM model, and a usage of only 100 samples per evaluation run. The best performance is observed when the percentile falls between 70 and 90, which seems to represent an optimal balance between context and chunk size. This range allows the model to access sufficient relevant information without overwhelming it with excessive or irrelevant context. Beyond the 90th percentile chunk sizes grow too large, potentially introducing noise or unnecessary details, which may dilute the effectiveness of the information retrieved. This leads to a decrease in score and a higher incidence of hallucinations.

Additionally, we explored the use of **K-Means Clustering (KMC)** for chunking (similar to Kang et al., 2024), an approach that groups semantically similar sentences into clusters based on their sentence embeddings. Unlike fixed-size chunking methods, KMC identifies natural groupings of sentences without considering sentence order, which can capture deeper semantic relationships across the text. In this method, sentences are first split and then embedded. These embeddings serve as input to the K-Means algorithm, which clusters the sentences into a predefined number of clusters, k.

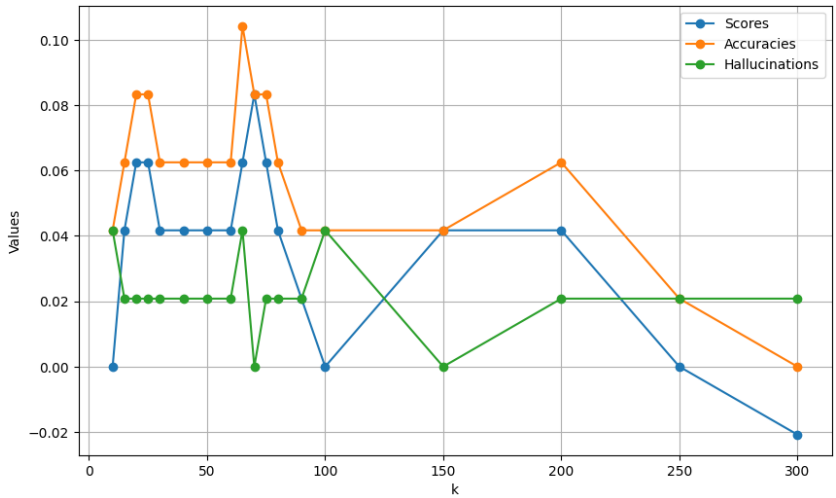

Figure 5: Evaluation metrics across different k's for *K-Means Clustering*, showing variations in performance based on chunk count.

Each cluster then forms a chunk of text, where the sentences share strong semantic similarity. One potential downside of this method is the loss of sentence ordering, which may confuse models that depend on coherent text flow. However, this method benefits from creating chunks that are more semantically cohesive than those generated by simple fixed-size approaches.

Figure 5 illustrates the performance across various values of k. At lower values of k (between 20 and 50), we observe relatively stable performance in terms of both scores and accuracies, with low hallucination rates. The peak performance, in terms of the score, is observed around $k = 70$. This suggests that clustering at this level captures sufficient context for effective retrieval without overwhelming the model. As k increases beyond 100, both scores and accuracies begin to decline, with fluctuations in hallucination rates. At high values of k, such as 200 and above, the performance drops significantly, likely due to small chunk sizes leading to low information in each chunk. Overall, an optimal k range appears to be between 40 and 100, balancing chunk cohesion and information richness without overwhelming the model.

Embedding Manager

For our RAG implementation, we utilized two key embedding techniques: Word2Vec- and BERT Embeddings with the goal to maximize the semantic understanding of the text in order to successfully answer difficult questions.

WordNet is a large lexical database that groups words into sets of synonyms. By using Word2Vec embeddings, we leverage the relational knowledge between words to improve word sense disambiguation to help capture hierarchical relations like hypernyms and hyponyms. This enables our model to better understand semantic relationships in the query and context. We also employed *BERT* (Bidirectional Encoder Representations from Transformers), which is one of the state-of-the-art models for contextual word embedding. BERT captures deep contextual relationships in the text and allows the model to generate embeddings that account for both left and right contexts. This is particularly valuable in our RAG framework, as it enables the retrieval component to identify the most relevant passages based on nuanced, contextual understanding of the query. As shown in recent advancements, BERT embeddings significantly enhance performance in various NLP tasks, including information retrieval and RAG tasks (Devlin et al., 2019).

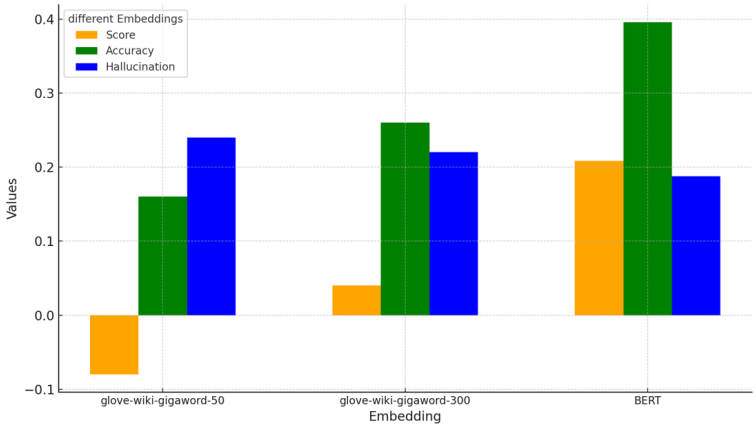

Figure 6: Comparison of score, accuracy and hallucination of embedding models - BERT and Word2Vec.

As illustrated in Figure 6, we compared three embedding models: GloVe (glove-wiki-gigaword-50), GloVe (glove-wiki-gigaword-300) and BERT and focused on three key metrics: Score, accuracy, and hallucination. BERT outperforms the GloVe models on both score and accuracy and proves to be the most effective embedding for our RAG system. Importantly, BERT also shows lower hallucination compared to both GloVe models. It indicates that the model generates fewer irrelevant or misleading results, making it more reliable in maintaining the quality of retrieved information. On the other hand, while the GloVe embeddings, particularly the 300-dimensional version, perform well in terms of accuracy, they exhibit higher hallucination, which diminishes their overall utility. This comparison underscores the importance of using deep contextual embeddings like BERT to enhance the retrieval component in RAG frameworks.

Retriever Manager

The retrieval component in the RAG architecture determines which chunks of information from the source documents are used as context for the generation model. This section discusses the different retrieval methods we evaluated, as summarized in Table 3, along with their respective

strengths and weaknesses in the context of our system.

In the default configuration of our RAG system, we relied on cosine similarity as the primary retrieval method, measuring the semantic similarity between document chunks and the query. The evaluation results indicated a moderate performance with a score of 0.167, accuracy of 0.375 and a hallucination of 0.208, but it was associated with relatively high missing rate (0.417).

We also experimented with the BM25 algorithm, a keyword-based approach that works directly with raw text and considers term frequency, document length, and inverse document frequency to calculate relevance scores. BM25 scored 0.073 in our tests, with an accuracy of 0.33. It showed a slightly worse performance in reducing hallucinations (0.25) and the highest missing rate of 0.43. As the least effective method in our evaluation, these results highlight the limitations of relying solely on exact keyword matches to retrieve the relevant chunks.

We implemented Hypothetical Document Embedding (HyDE) (L. Gao et al., 2022), a method that leverages language models to generate hypothetical answers for queries, which are then used to retrieve relevant chunks. In our tests, HyDE achieved a score of 0.094 and an accuracy of 0.354, with a hallucination rate of 0.26 and a missing rate of 0.385. While HyDE can improve retrieval by capturing query intent, the results suggest it may not be optimal for our RAG system, which requires high accuracy in QA tasks.

Hybrid search combines an embedding-based and a keyword-based approach, which in our implementation uses cosine similarity and BM25, respectively. We investigated various configurations of hybrid search, adjusting the α parameter (higher α favoring cosine similarity, lower α favoring BM25) to balance semantic understanding and keyword matching. The hybrid search with $\alpha = 0.75$ emerged as the most successful approach in our evaluations, achieving the highest score of 0.24 and accuracy of 0.448. This setting successfully balanced the semantic relevance with the lowest missing information rate (0.344) and a comparatively low hallucination rate (0.208). Conversely, the settings with $\alpha = 0.5$ and $\alpha = 0.25$ showed less efficacy, scoring 0.115 and 0.188, respectively. It can still be noted that the configuration using $\alpha = 0.25$ showed the lowest hallucination rate, meaning a heavier reliance on exact keyword matches may have a positive impact on the hallucination in our RAG system. These evaluations demonstrate the significant impact of retrieval strategies on the performance of the RAG system. Hybrid search with $\alpha = 0.75$ notably provided the best balance between accuracy and the comprehensiveness of retrieved information, making it a promising candidate for further development and refinement in our architecture.

Approach	**Score**	**Accuracy**	**Hallucination**	**Missing**
Cosine similarity	0.167	0.375	0.208	0.417
BM25	0.073	0.33	0.25	0.43
HyDE	0.094	0.354	0.26	0.385
Hybrid search ($\alpha = 0.75$)	**0.24**	**0.448**	0.208	**0.344**
Hybrid search ($\alpha = 0.5$)	0.115	0.365	0.25	0.385
Hybrid search ($\alpha = 0.25$)	0.188	0.385	**0.2**	0.42

Table 3: Evaluation of Retrieval Methods.

Generation Manager

The role of the generation manager is to call the LLM and format the prompt. The model is instructed to generate content using only the references and respond with "I don´t know", if it cannot give a reliable answer. As only the Llama models were allowed in the Meta KDD challenge, we chose Llama3-8b (AI@Meta, 2024). The lower parameter count is necessary to suit our setup of two Nvidia Tesla T4s.

Query Expansion

Futhermore, we explored the effect of query expansion by leveraging WordNet. Query expansion helps in enriching the input queries by adding semantically related words. It improves the retrieval process by aligning with various forms of the input query. This process enriches the query corpus, hoping to boost accuracy and becomes particularly valuable when users input short or ambiguous queries, which may not directly match the indexed documents.

As shown in Figure 7 we compared the default approach without query expansion to the WordNet-enhanced approach. In the *default approach*, we observe the highest accuracy and a moderate score, coupled with low hallucination. This indicates that the model performs well in precise retrieval tasks but may miss out on semantically related, broader content. In contrast, the *WordNet expansion approach* shows a slight decrease in accuracy, with a significant increase in hallucination, suggesting that the introduction of WordNet leads to more irrelevant results.

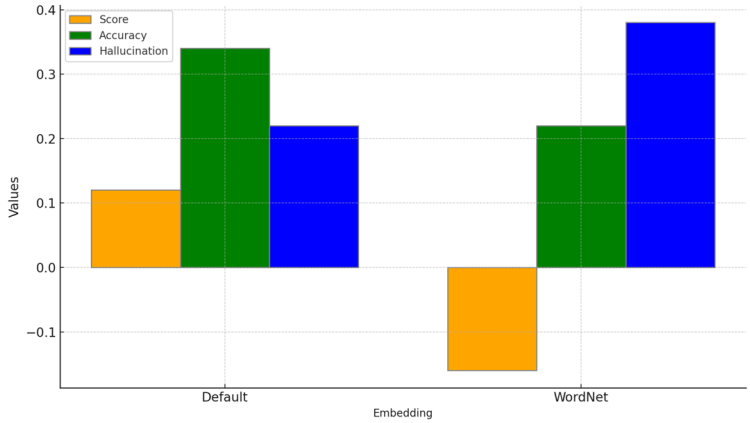

Figure 7: Comparison of score, accuracy and hallucination - with and without query expansion.

Additionally, the score decreases marginally, reflecting that query expansion may introduce noise, rather than improving retrieval performance. In summary, while the default approach provides greater precision, WordNet expansion increases the likelihood of retrieving broader, yet sometimes irrelevant, content.

We also explored the potential of integrating LLama for query expansion to further enhance retrieval capabilities by generating more contextually rich queries. LLama's generative abilities could provide a flexible way to expand input queries. Due to architectural limitations, we were able to fully implement, but not to test this approach within our current setup. This would be a task for future work.

Chain of Thought

Question types that demands complex reasoning, like multi-hop or simple-with-condition questions, are challenging for LLMs. Wei et al., 2023 method **chain-of-thought** tackle this problem by making use of the few-shot learning ability of LLMs. This involves adding example questions with answers that include reasoning steps, guiding the model through the problem-solving process. The addition of reasoning steps is crucial, as it forces the LLM to answer the question step-by-step. The idea is to mimic the human thought process, which consists of multiple stages of reasoning instead of coming up with an answer immediately. By combining RAG with chain-of-thought, we get the "Interleaving Retrieval with Chain-of-Thought Reasoning" pattern, short IRCoT of (Trivedi et al., 2023).

In IRCoT, chain-of-thought reasoning guides retrieval, and vice versa. The prompt includes an example question, five references, a multi-step reasoning answer, and the target query. The LLM produces an output, or "thought," which becomes new content for the prompt. At each step, the prompt is updated by adding this thought and the ten most similar retrieved chunks.

This helps for instance for a question such as "Which was the first movie adaptation of the author of the book *Misery* and who directed this movie?". During the first step , the LLM figures out that Stephen King wrote *Misery*. Using this thought, the retriever can know find a Wikipedia article listing all movie adaptations of Stephen King´s books. Then, using this new piece of information, we find out that *Carrie* is the first movie. Now, we can retrieve an article about the movie *Carrie*, which states that the director of that movie is Brian De Palma. Having collected all of this information, the LLM can give an informed answer like for example: "*Carrie* is the first movie adaptation of Stephen King, the author of *Misery*. *Carrie* has been directed by Brian De Palma.".

This example describes an ideal case, but how well this approach works in practice is investigated by evaluating our base model against the model which uses IRCoT. The motivation to implement the IRCoT model can be supported by the evaluation in Table 4. Naive RAG achieves higher accuracy on for example the *simple* question type, in comparison to *multi-hop* questions. As we can see in the first two entries of the table, the naive model has a better score than the IRCoT one on miscellaneous question types. Though naive RAG achieves lower accuracy, it has a better awareness when not to answer, as its hallucination and missing score are relatively lower than those of IRCoT. The higher accuracy of the IRCoT model is expected due to calling the LLM more often. It remains unclear, if some alterations to the prompt can improve the other metrics too. The results are not convincing enough to follow this approach any further. The higher computational cost of the IRCoT model do not justify it, as it performs too similar to the more efficient naive approach.

Model	Question Type	Score	Accuracy	Hallucination	Missing
Naive RAG	all	**0.15**	0.36	**0.21**	**0.43**
IRCoT	all	0.09	**0.41**	0.32	0.27
Naive RAG	simple	**0.07**	**0.30**	0.23	**0.47**
Naive RAG	multi-hop	0.04	0.26	**0.17**	0.57

Table 4: Top half: Evaluation of metrics for naive RAG and IRCoT model respectively. Bottom half: Evaluation of naive RAG on dataset entries with question type simple, compared to entries with question type multi-hop. All runs conducted with a sample size of 100 samples.

3 Conclusion and Future Work

In the following, we are going to discuss the results of our implementation and possible future work. We developed a RAG system for QA tasks using web pages as additional sources. The system achieved its best performance with a score of 0.24 and an accuracy of 44,8%. This result was obtained using hybrid search with $\alpha = 0.75$ and the following configuration: BasicReader for extracting document text, QueryExpansionManager to enhance query relevance, SemanticChunkerManager to optimize context selection, LlamaEmbeddingManager for embedding creation, and BasicGenerator for generating the final output. While promising, these results open up several potential directions for future work.

In our RAG system, we explored the potential of integrating **Llama for query expansion** to further enhance retrieval capabilities by generating more contextually rich queries. However, due to architectural limitations, we were only able to fully implement, but not test this approach within our current setup. The evaluation of its effectiveness in query expansion tasks can be explored in future work.

We experimented with the chain-of-thought based method IRCoT to improve the capability of our model regarding questions that require multi-step reasoning, like multi-hop questions. This turned out to not lead to a significant improvement compared to our naive approach. Future work could be investigating other patterns, such as **Tree-of-Thought** (Yao et al., 2023) or trying out specifically fine-tuned models like **Self-RAG** (Asai et al., 2023).

For improved accuracy, **Sentence-BERT (SBERT)** can be employed as an embedding model in future work as well. While both models are based on the transformer architecture, BERT is not optimized for comparing entire sentences or generating fixed-size sentence embeddings for tasks like semantic similarity or sentence clustering. SBERT was developed to generate sentence-level embeddings, which are directly useful for tasks regarding semantic similarity, clustering, and ranking. It allows for a quick comparison of sentences by providing fixed-size embeddings that can be contrasted using distance metrics such as cosine similarity.

To address the limitations observed with *BC* and *SC*, we propose the application of a more refined algorithm called **Semantic Double-pass Merging**. This method builds upon classical semantic chunking by introducing a secondary pass. This pass enables a more intelligent merging of chunks based on semantic relevance beyond direct sentence adjacency. In the first pass, chunks are generated similar as in *SC*. In the second pass, the algorithm looks ahead and compares chunks that are separated by an intervening segment. If the semantic similarity between non-adjacent chunks exceeds a threshold, the algorithm merges these chunks, preserving relevant context that may be disrupted by intermediate elements such as formulas, code, or quotations. This technique enhances chunk cohesion and reduces information fragmentation, especially in texts with complex structures (Rucinski, 2024).

Building upon our results, integrating **Hybrid Search with Hypothetical Document Embeddings (HyDE)** into the embedding-based component would be an interesting direction for future work. While Hybrid Search with $\alpha = 0.75$ achieved the best performance, HyDE has the potential of capturing query intent despite lower overall scores. This combination could leverage the strengths of both approaches, potentially enhancing retrieval performance. Notably, a previous study has already explored this combination with promising results (Wang et al., 2024).

List of Contributions

Chapter	Jacob Abb	Michel Meißner	Lea Schümann	Bahne Thiel-Peters	Alexander Zupke
Introduction (1)	X	X	X	X	X
Related Work (1.1)			X		
Architecture (2.2)					X
Dataset and Metrics (2.2)			X		
Reading Manager (2.3)		X			
Chunking Manager (2.3)				X	
Embedding Manager (2.3)	X				
Retriever Manager (2.3)		X			
Query Expansion (2.3)	X				
Generation Manager (2.3)					X
Chain of Thought (2.3)			X		X
Conclusion and Future Work(3)	X	X	X	X	X

Table 5: Contribution

References

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

Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2023). Self-rag: Learning to retrieve, generate, and critique through self-reflection. https://arxiv.org/abs/2310.11511

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gao, L., Ma, X., Lin, J., & Callan, J. (2022). Precise zero-shot dense retrieval without relevance labels. https://arxiv.org/abs/2212.10496

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024, March 27). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv: 2312.10997 [cs]. https://doi.org/10.48550/arXiv.2312.10997

Kang, H., Zhu, Y., Zhong, Y., & Wang, K. (2024). Implementing streaming algorithm and k-means clusters to rag. https://arxiv.org/pdf/2407.21300

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 9459–9474, Vol. 33). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

McKenna, N., Li, T., Cheng, L., Hosseini, M. J., Johnson, M., & Steedman, M. (2023). Sources of hallucination by large language models on inference tasks. https://arxiv.org/abs/2305.14552

Meta. (2024). Meta comprehensive rag benchmark: Kdd cup 2024: Challenges. https://www.aicrowd.com/challenges/meta-comprehensive-rag-benchmark-kdd-cup-2024

Mikolov, T. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Radford, A., & Narasimhan, K. (2018). Improving language understanding by generative pre-training. https://api.semanticscholar.org/CorpusID:49313245

Rucinski, K. (2024). Chunking methods in rag: Comparison. *BitPeak*. https://bitpeak.com/chunking-methods-in-rag-methods-comparison/#:~:text=Semantic%20double-pass%20merging%20and%20propositions-based%20chunking%20using%20gpt-4%20performed%20admirably,

Santhosh, S. (2024). Enhancing rag efficiency: The power of semantic chunking. *Medium*. https://medium.com/@sthanikamsanthosh1994/enhancing-rag-efficiency-the-power-of-semantic-chunking-844f9cfbdd0b#:~:text=By%20condensing%20data%20into%20logically%20connected%20segments,%20semantic%20chunking%20facilitates

Slocum, V., & Schmuhl, E. (2024, March 7). *Verba: Building an Open Source, Modular RAG Application | Weaviate*. Retrieved September 26, 2024, from https://weaviate.io/blog/verba-open-source-rag-app

Trivedi, H., Balasubramanian, N., Khot, T., & Sabharwal, A. (2023, June 22). *Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions*. arXiv: 2212.10509 [cs]. https://doi.org/10.48550/arXiv.2212.10509

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. http://arxiv.org/abs/1706.03762

Wang, X., Wang, Z., Gao, X., Zhang, F., Wu, Y., Xu, Z., Shi, T., Wang, Z., Li, S., Qian, Q., Yin, R., Lv, C., Zheng, X., & Huang, X. (2024). Searching for best practices in retrieval-augmented generation. https://arxiv.org/abs/2407.01219

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023, January 10). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv: 2201.11903 [cs]. https://doi.org/10.48550/arXiv.2201.11903

Wu, S., Xiong, Y., Cui, Y., Wu, H., Chen, C., Yuan, Y., Huang, L., Liu, X., Kuo, T.-W., Guan, N., & Xue, C. J. (2024). Retrieval-augmented generation for natural language processing: A survey. https://arxiv.org/pdf/2407.13193

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: Deliberate problem solving with large language models. https://arxiv.org/abs/2305.10601

Zubiaga, A. (2024). Natural language processing in the era of large language models. *Frontiers in Artificial Intelligence*, *6*. https://doi.org/10.3389/frai.2023.1350306