EXPLORING THE CREATION AND OPTIMIZATION OF RAG-BASED LOCAL LARGE LANGUAGE MODELS FOR DOMAIN-SPECIFIC APPLICATIONS AND HOW IT COMPARES WITH LOCAL SEARCH ENGINES

MAGNUS BØNDERGAARD POULSEN, 202008368
AUGUST BORUP EXNER, 202005782
PATRICK ENGELBRECHT SØRENSEN, 202005459
DENNIS VABBERSGAARD, 202006200

PROJECT REPORT

January 2025

Advisor: Niels Olof Bouvin

AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

EXPLORING THE CREATION AND OPTIMIZATION OF RAG-BASED LOCAL LARGE LANGUAGE MODELS FOR DOMAIN-SPECIFIC APPLICATIONS AND HOW IT COMPARES WITH LOCAL SEARCH ENGINES

MAGNUS, AUGUST, PATRICK & DENNIS

Project Report

Department of Computer Science
Faculty of Natural Sciences
Aarhus University

January 2025

ABSTRACT

This project investigates the development and evaluation of a local Retrieval-Augmented Generation (RAG) large language model (LLM) for domain-specific applications, with a focus on the Lord of the Rings (LOTR) universe. The study explores key aspects of implementing such systems, including data scraping, system design, and qualitative and quantitative evaluations. A comparative analysis was conducted between the local RAG-based LLM and ElasticSearch-based search engine to assess their respective strengths and limitations. The LLM demonstrated user-friendly interactions and the ability to provide contextually relevant responses, though it occasionally generated overly detailed answers. In contrast, ElasticSearch offered broader data retrieval capabilities but demanded more user effort for effective search of information. These results highlight the trade-offs inherent in adopting conversational AI versus traditional search engine approaches. The project faced several challenges, such as hardware limitations that impacted response times and evolving software dependencies that required constant updates. Evaluation metrics like BERTScore and the Ragas framework were used to optimize parameters, although the scope of testing was constrained by incompatibility between systems. Overall, the findings underline both the potential and the complexities of deploying local RAG-based LLMs. The study identifies challenges and pitfalls in areas such as computational efficiency and scalable evaluation methods, paving the way for broader application of these technologies in specialized domains.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

1 INTRODUCTION

The development of large language models (LLMs) has significantly changed how users interact with information, making it easier and more natural to access knowledge through conversational and context-aware tools. However, these systems face challenges, especially in areas where privacy, accuracy, and specialized expertise are critical. Public LLMs often require users to send sensitive data to external servers, which can raise concerns about data security. Additionally, these models sometimes hallucinate, generating incorrect responses, which can be problematic in high-stake fields like healthcare, finance, or legal advisory. This project investigates locally hosted LLMs combined with Retrieval-Augmented Generation (RAG) as a way to address these issues. By focusing on processing data on a local server and using domain-specific information, this approach aims to improve both privacy and accuracy. RAG helps enhance the reliability of the system by integrating external, structured data into the model's responses, reducing the chance of hallucinations. For this study, we use the extensive Lord of the Rings (LOTR) fandom wiki as a dataset, testing the system's ability to handle detailed and domain-specific questions.

To quantitatively assess the accuracy of the RAG system, the Ragas framework, BERTScore and ChatGPT with intent to evaluate aspects such as faithfulness and answer relevance on a broader scale than what manual evaluations can achieve. To assess the system's effectiveness from a user perspective, we conducted a comparison between the RAG system and Elasticsearch, a widely used traditional search engine. The evaluation involved testing both systems with factual, complex, and open-ended queries to assess their ability to retrieve and present information. By examining their strengths and weaknesses, this study provides insights into how these technologies can be used in specialized information retrieval and decision-making tasks. This project demonstrates the potential of local LLMs for secure and specialized applications and explores how they might serve as an alternative to traditional search engines.

2 RELATED WORK

In this section we will present related work to our problem statement. Starting with an overview of Large Language Models (LLMs), their evolution and challenges. Then, briefly introducing existing LLM methods and cases. Finally, relevant evaluation considerations.

2.1 EVOLUTION

In the paper, A Survey of Large Language Models by Zhao et al. [16], the authors review and discuss some of the recent advances of LLMs by introducing the background, key and mainstream techniques and challenges when developing LLMs.

Recent advancements in LLMs have significantly impacted artificial intelligence. These models evolved from early statistical and neural approaches into pre-trained models like BERT and GPT, culminating in current LLMs such as GPT-3, GPT-4, and Gemini. The distinguishing factor for LLMs is their ability to scale, leading to emergent capabilities like in-context learning, instruction following, and multi-step reasoning. These abilities have made LLMs powerful general-purpose solvers for complex tasks across various domains. Despite their transformative potential, challenges persist in training efficiency, human alignment, and managing risks like hallucinations.

It has been a longstanding research challenge to enable machines to read, write, and communicate like humans. Technically, language modeling (LM) is one of the major approaches to advancing language intelligence of machines. In general, LM aims to model the generative likelihood of word sequences, so as to predict the probabilities of future (or missing) tokens. For this there are four considered stages:

Statistical language models (SLMs): Mathematical rules from the 1990's for predicting the next word based on the most recent context, helpful at simple tasks, but struggling with complex sentences due to lacking enough data.

Neural Language Models (NMLs): With the rise of neural networks in the 2010's machines became better at learning patterns in language, understanding relationships between words in a more advanced way increasing accuracy and use cases.

Pre-trained Language models (PLMs): PLMs like BERT introduced context-aware word representations through pre-training on large corpora, followed by fine-tuning for specific tasks. The arrival of the Transformer architecture, with its self-attention mechanism, signifi-

cantly improved PLM performance, establishing the now-dominant "pre-training and fine-tuning" paradigm

Large Language Models (LLMs): Scaling up PLMs led to the development of LLMs, such as GPT-3 and GPT-4, which possess billions of parameters. These models display emergent abilities, capabilities not observed in smaller models, such as in-context learning and solving complex tasks.

2.2 CHALLENGES

Kaddour et al. [7] explore the limitations, applications, and adaptations of LLMs in the paper, Challenges and Applications of Large Language Models. The authors identify challenges such as high data requirements, latency, and limitations in contextual reasoning. They highlight the issue of hallucinations, where models produce incorrect or made-up information, emphasizing the need for retrieval-augmented generation (RAG) to ensure outputs are coming from reliable external sources. Prompt brittleness, where small variations in phrasing lead to differences in model outputs, underlines the need for robust prompting. Additionally, evaluations which rely on human-written ground truths are inadequate to capture the complexity of real-world tasks. The authors discuss the potential of LLMs in domain-specific applications, especially when enhanced with fine-tuning techniques, suggesting that with proper frameworks, LLMs can effectively bridge knowledge gaps in specialized fields.

Zhao et al. [16] highlights a lack of transparent data collection/training corpora. The quality and scale of pre-training datasets are critical to LLM performance. Several models disclose the nature of their training data, including large corpora comprising internet text, academic papers, and code repositories. However, data transparency remains an issue for many proprietary models, limiting reproducibility.

They also address that developing or deploying LLMs requires substantial resources, both in terms of technical infrastructure and accessible tools. A growing ecosystem of publicly available resources helps researchers and practitioners overcome barriers to experimentation and incremental improvement. These resources include pre-trained model checkpoints, APIs, corpora, and libraries.

The paper also discusses the limitations of LLMs in specialized domains or tasks. While LLMs are proficient in generating coherent text due to their general training, they struggle with domain-specific tasks, such as generating medical reports that require specialized terminology and knowledge. Injecting this specialized knowledge into LLMs is challenging and can lead to issues, where new training interferes with previously learned abilities. The text emphasizes the importance of developing methods to specialize LLMs for various tasks while minimizing the loss of their original capabilities.

> As Zhao et al. [16] states, "LLMs may fall short in mastering generation tasks that require domain-specific knowledge or generating structured data. It is non-trivial to inject specialized knowledge into LLMs, meanwhile maintaining the original abilities of LLMs." (p. 58).

Recent progress in developing LLMs has focused on improving how accurate their information is, which is a very important issue for using them in specialized fields. Wang et al. [13] conducted a detailed study on the accuracy of LLMs, looking at two main types: standalone models like GPT-4 and retrieval-augmented models like BingChat. The study explains that these models often make factual mistakes, especially in important fields like healthcare, finance, and law, where wrong information can have serious effects. The authors review different ways to measure accuracy, including basic rule-based methods like Exact Match and more advanced tools like BERTScore. They also highlight the need for specific benchmarks to test how well these models work in fields where accuracy is very important. Additionally, they mention strategies to reduce mistakes, like using retrieval-augmented methods and continuing to train LLMs with data from specific areas of knowledge.

2.3 RETRIEVAL AUGMENTED GENERATION (RAG)

Gao et al. [6] describes Retrieval Augmented Generation (RAG) as being a promising solution to combat some of the challenges following LLMs. RAG allows for enhanced accuracy when incorporating data from external databases. RAG can also help with outdated knowledge and untraceable reasoning. The general strategy for RAG is an initial indexing of the chosen data, splitting the text into smaller chunks that are then made into vectors stored in a vector database. This allows for efficient similarity searches. By computing the user query into a vector, it is then possible to calculate a similarity score and utilize the top-k chunks that fit the query. These chunks are then used to generate the prompt. The generated prompt can differ according to task-specific requirements. The paper also discusses parameters for optimizing different aspects of the process. The larger chunk sizes in the indexing phase can capture more content, but this comes with the consequence of noise and a longer processing time. However, smaller chunks can end up not providing enough context though having less noise.

2.4 FINE TUNING

Vulpescu and Beldean [12] explores how fine-tuning pre-trained LLMs such as Llama can enhance their correctness and accuracy for spe-

cialized tasks. The study shows that fine-tuning when done right can enhance the overall accuracy of LLMs. LLMs like Llama excel in tasks like text generation and categorization, but face challenges in data-limited scenarios, such as overfitting and hallucinations. Fine-tuning with methods like transfer learning, data augmentation, and regularization helps improve performance even with minimal data. Few-shot learning and active learning further optimize limited datasets. Hallucinations, where models generate incorrect outputs, are reduced using techniques like beam search, top-k sampling, and fact-checking. Human feedback and reinforcement learning also enhance reliability, though scalability can be an issue. Compared to traditional models, fine-tuned LLMs achieve superior accuracy and robustness but require more computational resources. These advancements make LLMs more practical and effective for specialized natural language processing (NLP) tasks.

2.5 DOMAIN SPECIFIC

Domain-specific applications of LLMs increasingly rely on specialized techniques to adapt general-purpose models to niche areas of expertise. Kaddour et al. [7] illustrate this with the use of impersonation prompts, allowing models to emulate domain experts for more accurate performance. This technique has shown promise in fields like law and medicine, where LLMs have demonstrated their potential for handling complex, specialized information. There is also a growing interest in using LLMs for broader knowledge tasks while ensuring accuracy in specialized domains. In-context learning further enhances this adaptability by enabling LLMs to integrate targeted training data, making them valuable tools for advancing domain-specific research and knowledge.

Zhang et al. [15] propose a framework to bridge the gap between domain-specific models and LLMs for personalized recommendations. They highlight that domain-specific models excel in capturing community behavior patterns, but struggle with data quantity. Opposite to domain-specific models, LLMs bring general knowledge and reasoning capabilities but lack specific domain information. The proposed framework makes use of mutual learning, allowing LLMs and domain-specific models to share information bidirectionally, enhancing both systems' performance.

2.6 EVALUATION

2.6.1 *Overview*

Chang et al. [2] provide a comprehensive overview of how to evaluate LLMs, focusing on what to evaluate, where to evaluate, and how to

carry out these assessments. They explore LLM applications in various fields like natural sciences, social sciences, and ethics, highlighting both successes and failures. In particular, they stress the importance of testing LLMs' abilities in Natural Language Understanding (NLU) tasks, such as sentiment analysis and text classification, to see how well they understand language. They also focus on reasoning tasks, which challenge models to make logical conclusions and handle commonsense problems. The paper considers ethical issues, evaluating how LLMs deal with biases and ethical questions across different social and cultural contexts. The authors highlight the need to assess a model's robustness to unexpected inputs, ensuring it can handle challenging scenarios beyond its typical training. Key challenges include improving evaluation methods and benchmarks, particularly in specialized fields, to address weaknesses like prompt sensitivity and factual accuracy. They advocate for diverse evaluation strategies that include domain-specific benchmarks and multilingual datasets, helping to test LLMs' performance across a broad range of tasks and contexts.

2.6.2 *Benchmarks and tools*

Papineni et al. [9] introduced BLEU, a metric created to automatically evaluate machine translation (MT) systems. BLEU measures the quality of translations by comparing them to reference translations, using modified n-gram precision to account for differences in word choice and order. To prevent over-counting repeated words, BLEU uses clipping, which limits how often words in a candidate translation can match those in the reference. It also includes a brevity penalty to discourage overly short translations and encourage outputs that are closer in length to the reference. Designed to be efficient, scalable, and applicable across languages, BLEU allows developers to quickly test and improve MT systems. It has been shown to align well with human evaluations on large datasets, making it a reliable tool for frequent assessments.

Lin [8] introduced ROUGE, a metric designed to automatically evaluate text generation systems, especially for summarization. ROUGE compares generated summaries to human-written ones using methods like n-gram overlaps, longest common subsequence (LCS), and skip-bigram statistics. These techniques allow flexibility by recognizing both exact matches and similar phrasing, making it useful for measuring quality even when word order varies. ROUGE is efficient, scalable, and offers a reliable alternative to manual evaluations, showing strong agreement with human judgments, particularly in single-document summarization. It has also been widely used in areas like machine translation and multi-document summarization, proving its usefulness for improving text generation systems.

Zhang et al. [14] introduced BERTScore, an automatic evaluation metric for text generation tasks. Unlike older methods like BLEU or METEOR, which focus on matching exact words or phrases, BERTScore uses contextual embeddings from pre-trained models like BERT to calculate similarity between words based on their meanings. This helps address problems with traditional metrics, such as not recognizing paraphrases or capturing semantic equivalence. Tests on tasks like machine translation and image captioning showed that BERTScore aligns more closely with human judgments, especially in difficult cases like detecting paraphrased text. Its ability to measure meaning accurately and handle complex text structures makes BERTScore a useful tool for evaluating natural language generation.

Chen et al. [3] introduced the RGB benchmark, a framework designed to evaluate how well retrieval-augmented generation (RAG) systems perform in large language models (LLMs). RGB focuses on four key areas: noise robustness, which looks at how well models handle irrelevant or noisy information; negative rejection, where models should avoid answering if relevant information is missing; information integration, the ability to combine details from multiple sources; and counterfactual robustness, which checks if models can detect and correct false information. While RGB shows that LLMs can handle noisy data to some extent, it also highlights ongoing challenges with integrating and verifying retrieved knowledge, as models often rely on incorrect external information over their own. This benchmark provides a clear way to identify these issues and improve RAG systems for knowledge-heavy tasks.

Roychowdhury et al. [10] evaluate how well different metrics work for assessing Retrieval-Augmented Generation (RAG) systems, focusing on domain-specific question answering (QA) tasks in the telecom industry. They point out that general-purpose metrics like BLEU and ROUGE, which rely on surface-level text matching, struggle to capture deeper semantic meaning in RAG outputs. To overcome this, the study looks at Ragas, a framework that uses metrics such as Faithfulness, Context Relevance, and Factual Correctness to evaluate results. Their experiments show that Faithfulness and Factual Correctness align more closely with expert evaluations for technical QA tasks, especially when language models are adapted to the specific domain. However, they also criticize metrics like Answer Relevance, which use cosine similarity from embeddings, as they can give unreliable results in specialized fields.

Recent progress in evaluating retrieval-augmented generation (RAG) has led to the development of frameworks that make automated assessments of RAG systems easier. Es et al. [5] introduced Ragas, a framework that focuses on faithfulness, answer relevance, and context relevance, using large language models (LLMs) like GPT-3.5 for scoring and breaking down tasks. Ragas works well with tools like

LangChain and LlamaIndex, making evaluation more efficient and closely matching human judgments, especially in faithfulness, where it outperforms baseline metrics like GPTScore. Building on this, Saad-Falcon et al. [11] introduced ARES, a similar framework that also evaluates context relevance, answer faithfulness, and answer relevance but uses prediction-powered inference (PPI) to improve confidence and accuracy while needing fewer annotations. ARES uses fine-tuned, lightweight LLMs trained on synthetic datasets, showing better efficiency and reliability across different domains and datasets like KILT and SuperGLUE, surpassing Ragas in both accuracy and efficiency. Both frameworks represent important advances in automating RAG evaluation, providing useful and reliable tools for developers.

2.6.3 *Large Language Model vs Search Engine*

Recent studies have examined how LLMs and search engines serve different purposes in information retrieval and learning. Divekar et al. [4] found that higher education students preferred LLMs for tasks that needed structured synthesis and simple explanations, while search engines were better for finding a wide range of detailed information. This shows how each tool fits different needs, with LLMs being useful for creating summaries or writing and search engines working well for deeper exploration. Similarly, Caramancion [1] looked at user preferences across different tasks and found that search engines were preferred for fact-based queries, while LLMs were better at handling more complex and interpretive tasks. The research highlights how search engines offer broad access to information, whereas LLMs provide detailed, conversational responses. Together, these studies suggest that combining the strengths of both tools could improve how users access and learn from information in specific subject areas.

3 ANALYSIS

Based on the insights from the related work, this analysis highlights key findings and challenges that inform the development and evaluation of local large language model (LLM).

3.1 EVOLUTION AND CHALLENGES OF LLMS

LLMs, such as GPT-4 and Gemini, have transformed natural language understanding and generation through advancements in pre-training techniques and scaling methodologies. As Zhao et al. [16] highlight, the historical progression from Statistical Language Models to Pre-trained Language Models (PLMs) and LLMs illustrates several key advancements, including the introduction of the Transformer architecture and self-attention mechanisms. These innovations enable LLMs to possess abilities such as in-context learning, instruction-following, and multi-step reasoning, positioning them as powerful general-purpose tools capable of solving complex problems across diverse domains.

Despite these advancements, LLMs face several challenges. One major issue, as discussed by Kaddour et al. [7], is the substantial computational resources required for their development and deployment, which include powerful infrastructure and large-scale datasets. This high cost limits accessibility and experimentation, particularly for smaller research groups. Zhao et al. [16] also emphasize the lack of transparency in data collection and training processes, which undermines reproducibility and hampers efforts to evaluate and improve these systems. Additionally, data privacy becomes a concern when users need to share sensitive information with these models and thereby the companies.

Another critical challenge, as noted by both Zhao et al. [16] and Kaddour et al. [7], is the tendency of LLMs to produce hallucinations, generating incorrect or fabricated information. This issue is particularly problematic in high-stakes fields such as healthcare, finance, and law, where accuracy is paramount. To address this, strategies like retrieval-augmented generation (RAG), as discussed by Wang et al. [13], showcase a promising approach for mitigating these errors by improving factual reliability through the integration of external, reliable sources.

3.2 RETRIEVAL-AUGMENTED GENERATION (RAG)

Retrieval-Augmented Generation (RAG) is a fitting approach to handle key challenges in LLMs, particularly for factual accuracy and avoiding hallucination. Gao et al. [6] describes RAG as a method for integrating external knowledge to enhance LLM outputs. By indexing data into vectors stored in a vector database, RAG makes efficient similarity searches available that retrieves from the most relevant chunks of information for prompt generation.

This can be used for various tasks and makes optimization of chunk size and overlap a critical factor. With larger chunks the contextual coverage broadens but with this noise may become a problem which introduces the risk of irrelevant context. For domain-specific applications, methods like impersonation prompts and in-context learning, discussed by Kaddour et al. [7] and frameworks for mutual learning between LLMs and specialized models proposed by Zhang et al. [15] illustrate how these systems can bridge general and niche knowledge areas. Together, these findings underline the potential of RAG to improve LLM reliability and adaptability for specialized knowledge tasks.

3.3 EVALUATION METRICS AND COMPARISONS

Evaluation frameworks are crucial in judging the performance of LLMs and RAG systems. Early metrics such as BLEU by Papineni et al. [9] and ROUGE by Lin [8] offered efficient and scalable ways to evaluate machine translations and summarization tasks by comparing generated text to human references based on word and n-gram overlaps. While these metrics have been effective they remain unsuccessful in accounting for semantic similarity. Further improvements to these metrics were made by Zhang et al. [14] who presented BERTScore which uses contextual embeddings from models like BERT to capture deeper meaning. Specifically for RAG evaluations improvements have been made with Ragas by Es et al. [5] and ARES by Saad-Falcon et al. [11], which focus on metrics such as faithfulness, relevance and context alignment. These frameworks use advanced scoring techniques to improve accuracy and efficiency, especially for domain-specific tasks. Together these tools provide a comprehensive evaluation framework for LLM and RAG systems.

3.4 LLMS VS. SEARCH ENGINES

The studies made by Divekar et al. [4] and Caramancion [1] reveal that there are different strengths and weaknesses when comparing the use of LLMs and search engines. LLM excels in giving conversational responses and explanations, such as summaries or understanding com-

plex topics. However, search engines are often favored when searching more in depth for uncovering a wide range of detailed information, or for fact based queries. These findings underline some of the strengths and weaknesses between LLMs and search engines and give a focus towards the need for LLMs to improve when searching for domain specific knowledge where search engines are still favored for more in depth searches.

This analysis serves as a foundation for the implementation and testing of a local LLM tailored to a specific knowledge domain, ensuring its performance aligns with both user needs and evaluation benchmarks.

3.5 TECHNICAL DECISION JUSTIFICATION

In analyzing the related work described in chapter 2, it becomes clear that a significant problem regarding LLMs is their ability to hallucinate. This provides concerns for domain specific fields when relying on the answers provided. Hence we would like to further explore the proposed RAG method to be able to eliminate the chance of hallucination. Furthermore custom LLMs require a high level of knowledge and extreme processing power to train the model to utilize the desired data.

Although some of the related work shows how fine-tuning an LLM can enhance the overall accuracy, this does not eliminate the hallucination. Therefore, a promising solution, as suggested by Gao et al. [6], is to mitigate hallucination by employing the Retrieval-Augmented Generation (RAG) method. Additionally, RAG offers advantages over traditional fine-tuning, including lower computational costs and greater simplicity. This simplicity makes it more accessible, enabling more people to create or utilize domain-specific LLMs effectively.

Additionally, analyzing the evaluation benchmark and tools, it is evident that simple evaluation of the machine translations are not enough to cover the complexity of evaluating a RAG LLM. Therefore, we have chosen to utilize the latest standard of BERTScore for machine translations together with newer specified RAG evaluation methods of Ragas or ARES which will give the most comprehensive evaluation of such a system.

Finally, by analyzing the LLM versus search engine section it becomes apparent that there is space for improving LLMs, specifically in domain-specific contexts, when searching for deeper knowledge. As this project specifically focuses on the domain-specific area we intend to test a domain-specific RAG LLM against a domain-specific search engine, to find and improve upon these deficiencies.

4

DATASET

Prior research on domain-specific LLMs often highlights their application in specialized fields such as law and medicine. For this project, the specific domain of the dataset is less significant. The decision to use a Fandom as the dataset was driven by its accessibility and the abundance of content it offers.

The Lord of the Rings (LOTR) Fandom was chosen due to its extensive range of material, spanning books, movies, and games. This ensures a broad and diverse dataset, distributed across numerous pages on the fandom's wiki. Another advantage of selecting LOTR is the familiarity many people have with its universe, which makes it easier to identify instances of hallucination by the LLM.

The LOTR Fandom Wiki, comprising 6,842 URL pages as identified through web scraping, allows for exploration across a variety of topics within Tolkien's universe from the main page and its branches.

5 DESIGN

In this chapter we write about the system design of our Local LLM, how it is intended to work to be able to solve the problems from the introduction.

5.1 CHATBOT DESIGN

LLMs are becoming increasingly popular and are being introduced in a lot of different applications, therefore it is necessary to avoid the LLMs making hallucinations or sharing data that is private with corporations.

With this system design we strive for a LLM which we can communicate with privately but also a robust system that limits the amount of hallucinations coming from the LLM.

5.1.1 *Local LLM*

To keep the LLM private we intend to download an existing LLM which has already been trained. A locally hosted LLM that allows for entirely on-device processing. This approach guarantees that all interactions remain private, eliminating the need to transmit data to external servers. By running the LLM locally, we create a controlled environment where the model can be optimized to reduce hallucinations effectively.

Our intention is to provide the LLM with accurate and relevant information by collecting domain-specific data through web scraping. We plan to process and store this data in a structured JSON format. A dedicated module will be designed to fetch the necessary data from these JSON files and convert it into documents, ensuring that only relevant information is passed to the database.

A vector database will be used to store semantic embeddings of the processed data, enabling efficient retrieval during interactions. By narrowing the scope of information available to the LLM to only the most relevant content, this database will play a critical role in reducing hallucinations. To further ensure accuracy, a prompt template will be designed to explicitly instruct the LLM to rely solely on the data retrieved from the database. This targeted prompt engineering should align the model's responses with the provided context, reducing un-

supported claims from the LLM.

To enhance accessibility, the system's functionality will be accessible through an API, enabling interaction between external applications and the LLM. The API serves as a bridge for integration with other platforms, ensuring smooth and reliable performance across various use cases. Additionally, this approach opens the possibility of hosting the LLM on a server, which can help overcome computational power limitations.

Finally, the user interface will be designed for simplicity and ease of use, offering a simple question-and-answer experience. Users will be able to interact with the LLM in real-time, and easily ask questions about the specific domain of which the LLM has been set for. See figure 1 for a UML diagram.

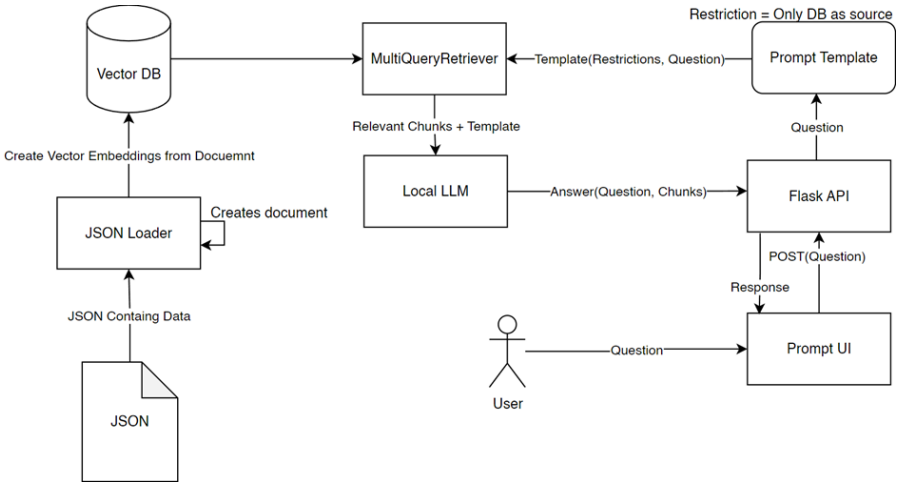

Figure 1: UML of planned RAG system design

5.2 SEARCH ENGINE DESIGN

To create a search engine that we can compare up against the local LLM, we need it to be based on somewhat the same techniques and architecture. Therefore, we plan to create a search engine that we can give the same data as we want to train the LLM upon. Elasticsearch can handle this idea of making a local search engine that is solely based on domain specific data.

To make sure the two technologies are on the same playing field, we want the search engine to have more than just a full-text search, where the search engine only matches text against text and how well it matches. It should be capable of searching for text in vector space, as this aligns with our vision for how the LLM will retrieve answers. Thus, we want to make a search engine that can hybrid search. It should be able to take the queries and match directly with title, context text and URL and calculate a score based on this match. If the question

"Who is Gandalf?" is asked, a title text of "Gandalf" should give a good score. Furthermore, it should give a higher score upon matching "wizard" with "magician" even when the questions are not asked in the specific wording of our data, we still get the most relevant search results.

To conduct a user test, we aim to design a user interface that makes both the LLM and the search engine easy to use. We plan to create a simple UI which provides a query input field, a list of the most relevant pages to the query, which will be ranked based on a scoring system. The most relevant pages should have a link to see the full text for further inspection. This should provide users with a functional search engine, allowing them to ask and answer questions for comparison with the LLM.

5.3 DESIGN OF THE EVALUATIONS

In the following section, we will present our intended design of evaluations. Both how users interact with LLMs and an automated evaluation of the LLM.

5.3.1 *Automated evaluation*

When evaluating a LLM, automated testing is essential. The volume of data is too extensive for a single person to manually evaluate and accurately distinguish correct answers from incorrect ones.

5.3.1.1 *Ragas*

RAG systems are generally considered a complex and sophisticated way of providing answers about a specific topic while having the answers grounded in external data in a faithful manner. Evaluating RAG systems can be challenging because the evaluation strategy depends on the language model and its probabilities, which are inaccessible for certain closed models [5].

We intend to use the Ragas framework for automated evaluations of our RAG system. Ragas offers a reference-free evaluation framework tailored for such systems, focusing on three core quality dimensions: faithfulness, answer relevance, and context relevance.

Faithfulness: Faithfulness ensures that generated answers are grounded in the retrieved context, avoiding hallucinations. Ragas uses LLMs to extract claims from generated answers and verifies their alignment with the retrieved context, scoring the proportion of claims supported by the context. The faithfulness score is calculated as $F = V/S$. Where "V" is the number of statements supported by retrieved context, "S" is

the total number of claims identified.

Answer Relevancy: This measures whether the answer directly and adequately addresses the query. Ragas assesses this by generating alternative questions derived from the answer and comparing their semantic similarity to the original query. Higher similarity scores indicate more relevant answers.

Example: if the answer is "The Eiffel Tower is in Paris," the AI might generate questions like, "Where is the Eiffel Tower?" or "What city is the Eiffel Tower located in?" By turning the newly generated questions into embeddings we can compare how close they are to the original question. If these questions align closely with each other, the answer must be relevant to the original question.

Context Relevancy: The context is considered relevant to the extent that it exclusively contains information that is needed to answer the question. The AI assesses whether it believes if the question can be answered from the given context.

Irrelevant or redundant context can reduce efficiency and effectiveness. Ragas computes a score by comparing the proportion of essential sentences in the context against the total sentences retrieved.

5.3.1.2 *BERT Score*

Alongside the Ragas framework, BERTScore is introduced as an additional metric to evaluate the performance of the RAG system.

As outlined earlier in section 2.6.2, BERTScore is a Natural Language Processing evaluation metric that assesses the quality of generated text by measuring its semantic similarity to a reference text. Unlike traditional metrics such as BLEU or ROUGE, which rely on surface-level word matching, BERTScore utilizes contextual embeddings from models like BERT to capture deeper semantic meaning.

The similarity between these embeddings is measured using cosine similarity. The cosine similarity measures the distance in vector spaces between these two embeddings. If this distance is small, there will be a high degree of similarity, but when the distance is large, there will be a low degree of similarity.

5.3.2 *Local Chatbot vs Elasticsearch*

To evaluate the effectiveness of our local LLM augmented with Retrieval Augmented Generation (RAG) compared to a more traditional search engine (Elasticsearch), we intend to perform a cognitive walk-through that incorporates a structured approach to assess both systems across various user scenarios. The walk-through will involve participants engaging with each system to answer a variety of questions under controlled conditions, allowing the collection of qualitative data.

5.3.2.1 *Question Categories*

The evaluation will involve three types of questions to comprehensively test the capabilities of the systems. Factual questions will demand straightforward, precise answers, such as *"Who created the One Ring?"* These are designed to test the systems' accuracy in retrieving unambiguous information. And complex questions, such as *"Why are the hobbits the ones to carry the ring?"*, will assess the systems' ability to provide nuanced and explanatory answers. Lastly, open-ended questions, like *"Is Gollum good or evil?"*, will explore how well each system can handle subjective or interpretive queries that require a broader contextual understanding.

5.3.2.2 *Participant Selection*

To ensure a well-rounded evaluation, participants will be divided into two groups based on their experience with information retrieval. The first group will consist of expert users, such as researchers or professionals accustomed to efficiently locating and evaluating information. The second group will include novice users, individuals with limited experience in systematic information-seeking. This diversity will help gauge how accessible and effective each system is for users of varying expertise levels.

5.3.2.3 *Test Scenarios*

The testing process will begin with a brief introduction where participants will be informed about the purpose of the evaluation and given a demonstration of both systems. This ensures that all participants have a baseline understanding of how to interact with the tools. Each participant will then use the RAG-based system to answer a set of five questions, followed by using Elasticsearch to address another set of five questions. To minimize bias, the order in which participants interact with the two systems will be alternated.

5.3.2.4 *Data Collection*

To comprehensively evaluate the systems, we will collect both quantitative and qualitative data. Quantitative metrics will focus on measurable performance indicators. We will record the time taken by participants to find answers, evaluate the accuracy of their retrieved responses, and compare the interaction effort by analyzing the number of searches and documents clicked (Elasticsearch) versus the number of prompts (RAG).These metrics will provide insights into the efficiency and effectiveness of each system.

Qualitative data will capture user perceptions and experiences. After completing the tasks, participants will rate their experience with each system using a 5-point Likert scale, providing feedback on ease of use,

satisfaction, and relevance of results. Additionally, semi-structured interviews will allow participants to share detailed thoughts about the strengths and weaknesses of each system, as well as suggestions for improvement. This qualitative feedback should assist in understanding how users perceive and engage with each tool.

By incorporating structured scenarios, a range of user perspectives, and diverse data collection methods, this test design seeks to evaluate the Local LLM with RAG up against a search engine such as Elasticsearch. The results aim to provide a balanced understanding of the strengths and limitations of both systems, offering insights into their potential applications and suitability for different information retrieval contexts.

6 ARCHITECTURE & IMPLEMENTATION

In this chapter we will explain and go more in depth with the architecture and implementation of the solutions we have used for scraping, RAG and Elasticsearch.

6.1 SCRAPING

To effectively collect an appropriate sized dataset, we decided to scrape the pages of The Lord of the Rings Wiki fandom website. This was done to collect information from various different sources within the LOTR universe, such as books, movies, games etc. As this would give our LLM a comprehensive overview of the subject. Furthermore, we intent to scrape different depths of data to test our LLM with different amounts of knowledge, to further understand how much data is required to give comprehensive answers.

The scraping has been done with the framework Scrapy, which is a fast and powerful tool written in Python, that crawls websites and extracts large amounts of data. The scraping is done by creating and activating what Scrapy calls "Spiders", it is a base class specialized to extract data from websites by defining custom crawling and parsing behavior for specific sites. The scraping process starts with generation of initial requests, that targets the URLs which have been specified in the spider's "start_urls" attribute, for our specific case[1]. Each request includes a callback function, which handles the response received from the web page. The page content is parsed using tools like Scrapy's selectors, to extract structured data. This data is then encapsulated as item objects and is processed further in Scrapy's item pipeline. Here, we write it to a JSON file, to get the appropriate format for later use. Furthermore, the callback may as well generate new requests for additional pages, repeating the cycle until the entire site or the defined number of pages or depth is scraped.

The wiki fandom pages are not all just raw readable text where everything is relevant for a LLM. So we have to sort and filter a lot of what would actually be scraped from just extracting all the data from the LOTR fandom wiki. So this requires a deep dive into understanding how the pages of the website wiki.fandom is actually set up. From our start URL, Main Page, we navigate to different link categories, to understand which groups of pages we do not want to extract the

1 https://lotr.fandom.com/wiki/Main_Page

data from. When making a LLM that should be accurate about LOTR lore we do not want blogs, forums, posts, talks etc. as this might lead to false information as people speculate about theories. Additionally, wiki fandom keeps the old versions of URLs when they have been altered, this might also make confusions as we would scrape the same information more than once. Furthermore, if scraping the full site, we extract other pages that do not make sense to the LLM, when given in JSON file format, such as edits, users, word list etc. We have therefore decided to exclude all of these URLs in the scrape to get a dataset that is as true as possible to the lore that is written on the LOTR wiki fandom website. Lastly, we have left out scraping of tables as this was a cumbersome affair especially considering how the LLM would understand these when given in a simple JSON file without proper formatting. This does however result in the LLM having a harder time answering very short and precise on factual questions as tables often provide questions and answer contexts.

With all the sorting and filtering being done, we can now scrape the different amounts we find fitting for testing the LLM. This is done by adjusting the depth limit in the Scrapy settings, a depth limit of one, means that the spider can only crawl one link[2] of depth before returning. This was done for a depth limit of 1, 2, 3 and no limit, resulting in a scrape of the full website. The extracted data from each depth resulted in the following amount: 1 = 1.408KB, 2 = 5.768KB, 3 = 8.745KB and no limit, All = 15.425KB. The data is then set up in a JSON file as "Title" containing the title, "Content" containing all the body text and "URL" containing the URL. This helps the LLM clarify what pieces of content are connected to each other and is also later used for setting up the search engine.

6.2 RAG - JUPYTER

To support the RAG system's capability to provide domain-specific information, the first step involves loading data stored in JSON format from the web scraping process. This ensures that the LLM operates on accurate and relevant information tailored to the specific domain of interest. The implementation uses the JSONLoader from the langchain_community.document_loaders module to streamline the process of loading and parsing JSON files. The JSONLoader is initialized with the file path of the target JSON file and a schema definition to ensure all relevant data is extracted efficiently. Additionally, the loader provides flexibility in processing complex or nested JSON structures, making it highly adaptable to different data formats.

2 The scrapes were made on the 21/11/2024 from the https://lotr.fandom.com/wiki/Main_Page

To enable efficient retrieval of relevant data during user interactions, the RAG system employs vector embeddings. This step involves converting textual data into numerical representations that capture semantic meaning, allowing the system to match user queries with the most contextually relevant information. The implementation uses OllamaEmbeddings from the langchain_ollama module to generate embeddings. This model, specifically designed for embedding text, ensures high-quality vector representations. The text is first split into manageable chunks using the RecursiveCharacterTextSplitter from langchain_text_splitters. This splitter creates overlapping chunks of specified size to preserve context across splits while ensuring compatibility with the embedding model. The chunking process is critical for optimizing both the quality of embeddings and the performance of the vector database. For this implementation: A chunk size of 512 characters is used as a starting point, this can be adjusted at any point to the specific dataset, to balance context retention and processing efficiency. Overlap between chunks is set at 50 characters to maintain continuity between splits. Once the text is prepared, the embeddings are generated and stored in a vector database. The system uses Chroma from the langchain_community.vectorstores module as the database. This lightweight, open-source vector store is ideal for hosting embeddings locally, aligning with the project's emphasis on privacy and efficiency Before populating the database, any existing collection is cleared to avoid conflicts or redundancy. The embeddings are then added to the database under a collection named "local-rag," ensuring the system operates on the latest and most relevant data.

The retrieval phase ensures that the RAG system fetches the most relevant information from the vector database based on user queries. This step combines query reformulation techniques and precise context-based prompt engineering to secure the quality of responses generated by the LLM.

To handle query reformulation and response generation, the system uses a locally hosted LLM model (llama 3.2) via the ChatOllama module. This ensures all interactions are processed privately. The LLM is configured with a prompt template to guide its behavior. A custom prompt template is used to reformulate user queries. The prompt positions the LLM as a domain expert, capable of generating multiple precise variations of the original question. This reformulation maximizes the likelihood of retrieving the most relevant information from the vector database with the intent of improving coverage and specificity.

The system utilizes a MultiQueryRetriever to interact with the vector database. By leveraging the LLM, the retriever generates multiple

reformulations of the user's original question, broadening the search scope and improving specificity. This increases the likelihood of identifying the most relevant data. The MultiQueryRetriever interacts with the vector database, in this case, ChromaDB, to locate the most contextually appropriate chunks. It identifies and retrieves text segments that are semantically aligned with the reformulated queries. Once the relevant chunks are retrieved, they are combined with the original user question and passed to the LLM for further processing.

Once relevant documents are retrieved, a secondary prompt is designed to generate answers based exclusively on the provided context. This ensures the LLM remains grounded, relying strictly on the retrieved data to minimize hallucinations.

The retrieval system is implemented as a chain of three operations: To begin with, user questions will be passed through the MultiQueryRetriever to extract the relevant information to answer the question. Secondly, a predefined response template processes the retrieved data and the original template to make sure answers from the LLM are only based upon the retrieved chunks to avoid hallucinations. Lastly, the LLM generates an answer which is aligned with the retrieved context.

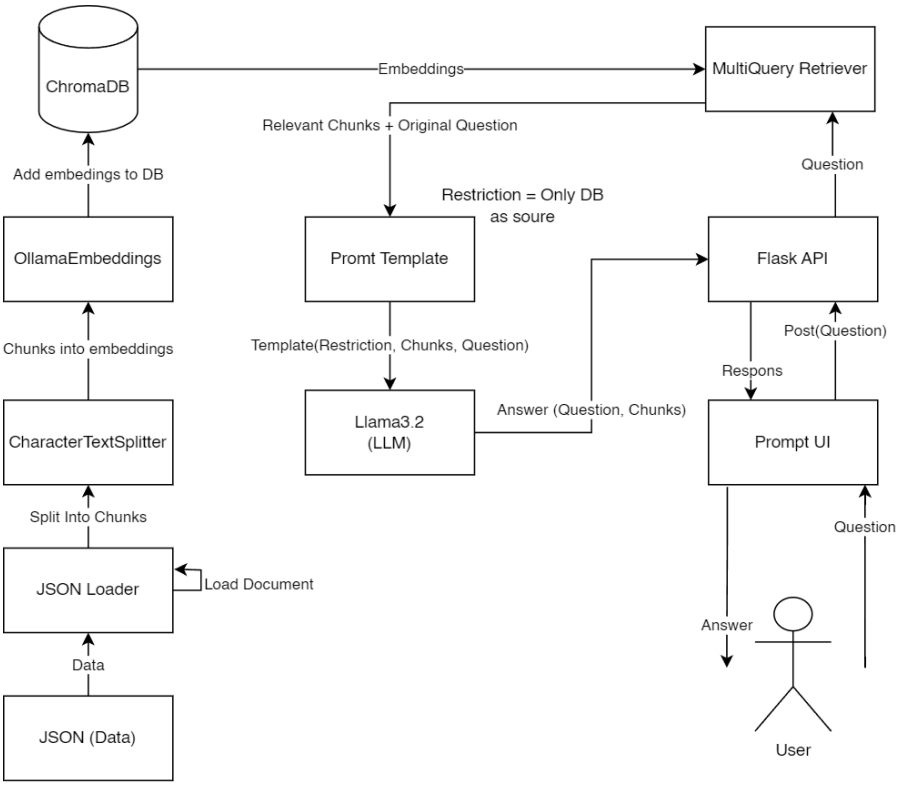

Figure 2: UML of implemented Local RAG LLM using Ollama

The system leverages Flask, a lightweight web framework for Python, to expose its functionality via a RESTful API. The architecture is designed to handle queries efficiently and provide seamless integration with external applications. Below is a breakdown of the core components and functionality.

A Flask app instance is created to serve as the foundation for the API. The CORS middleware is used to handle Cross-Origin Resource Sharing, allowing requests from external sources. This setup ensures that the API can be accessed from different platforms or services.

The chain_invoke function is designed to handle the core logic of processing user queries. This function calls the chain.invoke method to interact with the underlying model or service, providing the necessary output. The function is kept modular, allowing it to be easily modified or extended for different tasks.

The API defines two main endpoints:

- **Root Endpoint (/)**: A simple route that returns a message confirming that the Flask API is up and running. This serves as a health check for the service.
- **Invoke Endpoint (/invoke)**: This is the key endpoint for querying the system. It accepts POST requests with a JSON payload containing a question. The endpoint processes the request, invokes the chain.invoke method to generate a response, and returns the result as a JSON object. Error handling is included to ensure that missing or invalid data is appropriately flagged, providing clear feedback to the client.

The /invoke endpoint includes error handling. If the question parameter is missing from the request, a 400 status code is returned with an error message. If any other errors occur during the processing of the request, a 500 status code is returned.

The website serves as a straightforward front-end interface for interacting with the Flask API. It is built using standard web technologies, including HTML, CSS, and JavaScript, to create a lightweight and responsive design. The layout emphasizes simplicity and usability, providing users with a simple platform to submit queries and view results. At its core, the website features a form where users can input their questions. This form is the primary interaction point, consisting of a text input field and a submit button. When a user submits a question, the input is processed via JavaScript, which sends the data to the Flask API's /invoke endpoint using a POST request. Once the response is received, it is dynamically displayed on the webpage.

6.3 ELASTICSEARCH SEARCH ENGINE

To assess the performance, strengths, and weaknesses of our LLM, we implemented a search engine using Elasticsearch, tailored to the same domain-specific topic. This approach enables a direct comparison between the two systems and highlights their respective advantages and limitations.

The search engine leverages the same dataset as the LLM, specifically the LOTR wiki fandom, to ensure consistency in evaluation. Elasticsearch, a powerful and open-source search and analytics engine, was chosen for its fast, scalable, and relevant data retrieval capabilities. Its support for full-text searches and vector space querying makes it particularly suitable for comparing with the LLM, which also utilizes vector-based features for analysis.

The first step in creating a domain-specific search engine is configuring Elasticsearch on our system. We have opted for a local setup to keep all data internal. Next, we preprocess the scraped data to ensure consistent formatting. This involves structuring the content with fields like content, summary, title, and URL to present the data effectively in the search results. The prepared data is then stored in an Elasticsearch index, which will be utilized for vector space search. Each piece of data is inserted into the Elasticsearch index as a document, uniquely identified by its URL.

To handle searches, a multi-match query is employed to search across multiple fields, such as name, summary, and content, with different weightings. For example, the name field is assigned a higher weight since it often plays a critical role in providing accurate context. Additionally, vector similarity search is utilized to ensure the most relevant content appears at the top of the results, even if the words are not an exact match. This approach captures semantic similarities, such as recognizing that terms like "wizard" and "magician" belong to the same conceptual space. This is all integrated with a front-end web application using Flask to interact with users. Where pagination has been implemented for the user to browse through multiple pages if the first five pages does not show the appropriate answer.

All this combined should give us a fitting system to compare our search-engine against the LLM when trying to identify strengths and weaknesses of the LLM.

7 EVALUATION

In this chapter we will describe the evaluations and how they were carried out. First, an initial pilot test to assess whether the RAG LLM outputs correct and satisfying answers. Then, a quantitative evaluation, testing specific parameters for optimizing the performance of the RAG LLM. Lastly, how well the RAG LLM answers questions compared to Elasticsearch.

7.1 INITIAL PILOT TEST

Initially, when we started testing our system, we generated 10 questions where each question was answered 10 times. Then we would go through each question manually to identify whether the RAG system would answer both correctly and fulfilling. The test indicated that the local LLM had potential to become expert in domain specific data, however, this method quickly proved to be very inefficient and inconsistent due to our lack of expertise in linguistics and limited experience in evaluating semantic accuracy. Furthermore, for each of these question and answer sets, the BERTScore was calculated. We found the following to be the best parameters based solely on BERTScore, which we will later use as a baseline to compare with other parameters.

Baseline: The LLM is using the Llama3.2 model with an Impersonation prompt, a chunk size of 512 and a depth of all data.

7.2 RAG EVALUATION

To make sure that our RAG LLM was optimized to generate relevant answers within a domain specific area containing vast amounts of data, before comparing it with a domain specific search engine, we decided to evaluate the RAG LLMs performance.

As mentioned earlier in the design of evaluation, we planned to test our RAG LLM by evaluating with the Ragas benchmark framework as well as BERTScore. Due to limitations regarding Ragas and inaccessibility of larger models, we have been forced to downscale some parameters to be able to run even small parts of an evaluation with Ragas.

Therefore, we decided to test what was possible with Ragas, although it does not align with the way we intended to evaluate. Furthermore, we decided to add another test utilizing ChatGPT to imitate some of the metrics measured by the Ragas framework.

To emulate the evaluation setup used by Ragas, we decided to leverage ChatGPT as it offered the simplest approach. Our goal was to replicate the Ragas evaluation methodology by using the same techniques outlined in the framework. However, instead of relying on the LLM within Ragas, we used ChatGPT as the evaluating LLM to assess each parameter that required testing.

As an example when evaluating the answer relevance of our RAG LLM, we created a prompt template for ChatGPT. We would give ChatGPT the answer generated by our RAG LLM and then have it generate candidate questions that could lead to the answer. If the generated questions are closely aligned with the original question, we would manually give it a score of 1, 0.5 or 0.

7.2.1 *Answer Relevancy Prompt Template - ChatGPT*

Prompt Template Answer Relevancy
Answer: [Insert answer...]
Using the provided answer, reverse-engineer three candidate questions. Present these in a comparison table with the following columns:
Generated Question: A question based on the answer.
Comparison: A brief explanation of how the generated question aligns with the answer.

Prompt templates were created for the metrics, answer relevancy and faithfulness, which could be executed for each of the questions and its necessary content.

The evaluation ended up consisting of testing of the following parameters: prompt, model and depth. The parameters were chosen based on analysis of related work that has researched which parameters are generally most relevant when trying to optimize a RAG based LLM [7]. Chunk size was left out because of limitations of the LLM used with Ragas and will be discussed later. After having generated the answer datasets with our RAG LLM, we randomly picked three questions and their respective answer, ground truth and context pairs to evaluate with Ragas, ChatGPT and BERTScore.

7.2.2 *Prompt*

Before running the tests we created thirteen different prompt templates based on the prompt brittleness research done by Kaddour et al.[7]. These were used with our RAG LLM to create thirteen datasets that we could evaluate upon. After generating the datasets, we analysed the results and found only four of those thirteen to have appropriate results for further evaluation. These were Impersonation, Chain-of-

Thought (CoT), Tree-of-Thought (ToT) and Scratchpad.

Faithfulness and Context Recall were left out for the prompts as Ragas failed to get the results due to too large contexts.

PROMPT	ANSWER RELEVANCY	CONTEXT PRECISION	TOTAL
CoT	0.67	0.37	0.46
ToT	0.80	0.30	0.55
ScratchPad	0.75	0.38	0.50
Impersonation	0.68	0.56	0.52

Table 1: Prompt results for answer relevancy and context precision from Ragas.

As seen in table 1, running the Ragas evaluation resulted in the following Answer Relevancy and Context Precision scores.

The table shows Tree of thoughts (ToT) to be the most superior prompting type when evaluating the prompt parameter.

PROMPT	ANSWER RELEVANCY	FAITHFULNESS	TOTAL
CoT	0.78	0.16	0.47
ToT	0.89	0.40	0.65
ScratchPad	0.56	0.00	0.28
Impersination	0.72	0.70	0.71

Table 2: Prompt results for answer relevancy and context precision.

In contrast, as seen in table 2 when imitating the Ragas framework using ChatGPT, the results showed Impersonation to be the superior prompt parameter.

7.2.3 *Model*

In search for the best performing model four models were found for testing provided by Ollama these were Llama3.2, Mistral, Gemma2 and Phi3. After the datasets for each model was created, Phi3 was left out as it generated irrelevant results for later evaluation.

Evaluating the models with Ragas we were able to generate all metrics except Faithfulness and Context Recall for Question 2 in the evaluation of Llama3.2. This resulted in the following Answer Relevancy, Faithfulness, Context Recall and Context Precision scores and a Total Score as seen in table 3.

MODELS	ANSWER RELEVANCY	FAITHFULNESS	CONTEXT RECALL	CONTEXT PRECISION	TOTAL SCORE
Llama3.2	0.68	0.40	0.58	0.37	0.51
Gemma2	0.56	0.79	0.60	0.72	0.67
Mistral	0.84	0.71	0.58	0.94	0.77

Table 3: Models results calculated by Ragas.

Table 3 shows Mistral to be the superior model when performing the evaluation using the Ragas framework compared to Gemma2 and Llama3.2.

MODELS	ANSWER RELEVANCY	FAITHFULNESS	TOTAL SCORE
Llama3.2	0.72	0.70	0.71
Gemma2	0.72	1.00	0.86
Mistral	0.67	0.96	0.82

Table 4: Models results calculated manually.

In table 4 where the results are based on the imitation of the Ragas framework using ChatGPT, Gemma2 seems to be the most accurate model across answer relevancy and faithfulness.

When evaluating models, time has to be taken into account as long as computing power is limited. Therefore, we measured the average answering time for each model in seconds, which can be seen in table 5.

MODEL	AVERAGE TIME (S)
Llama3.2	12.1
Gemma2	83.7
Mistral	25.3

Table 5: The average time take to generate a response for one question for each model.

For reference, this was done on a Nvidia RTX 2060 graphics card. Time is relative, with greater computational power, time decreases and therefore becomes less relevant. However, in this case time is essential for the usability of the system and therefore Llama3.2 were chosen as the best model for further testing.

7.2.4 *Depth (Dataset size)*

The parameter depth is really the amount of data that is fed to the LLM. We call it depth, as we use a depth limit for our scrape of data, to get different amounts from our scrape. This is done by limiting how many links it can scrape through before returning. We intended to

evaluate the dataset size, to understand if there is a certain amount of data needed, to generate relevant answers and if there is an upper threshold for data, where too much noise is generated. Is more data always better, or is there a certain sweet spot where the amount of data is optimized.

As mentioned earlier in the section about Scraping in chapter 6 Architecture & Implementation we have collected four different sized datasets. We call these depth1, depth2, depth3 and depthAll. Running through the Ragas evaluation, we get the following scores for Answer Relevancy, Faithfulness, Context Recall, Context Precision and Total Score as seen in table 6.

DEPTH	ANSWER RELEVANCY	FAITHFULNESS	CONTEXT RECALL	CONTEXT PRECISION	TOTAL SCORE
Depth1	0.57	0.33	0.67	0.35	0.48
Depth2	0.65	0.37	0.62	0.76	0.60
Depth3	0.73	0.67	0.58	0.56	0.64
Depth All	0.68	0.40	0.58	0.37	0.51

Table 6: Depth results calculated by Ragas.

The results from table 6 indicate that a scraping depth of 3 achieves the highest total score when evaluating using the Ragas framework.

DEPTH	ANSWER RELEVANCY	FAITHFULNESS	TOTAL SCORE
Depth1	0.50	0.77	0.64
Depth2	0.78	0.85	0.82
Depth3	0.50	1.00	0.75
Depth All	0.72	0.70	0.71

Table 7: Depth results calculated by ChatGPT.

When calculating the total score of depth based on the ChatGPT evaluation we end up with Depth 2 as the most optimal depth as seen in table 7.

Hence the results from the ChatGPT evaluation shows the ideal LLM while using a RTX 2060 graphics card would be Llama3.2 with an impersonation prompt and a depth of 2. While the results from Ragas given unlimited computing power shows Mistral as the preferred model with an Impersonation prompt and a depth of 3.

7.2.5 *BERTScore*

Alongside the two other evaluations we have calculated the BERTScore for all parameters as well to compare as this score gives relevant

insight into the similarity of the answers and ground truths. We get the following F1 scores:

MODEL	BERTSCORE F1
Llama3.2	0.811
Gemma2	0.802
Mistral	0.797

Table 8: Generated F1 BERTScore for models.

PROMPTS	BERTSCORE F1
CoT	0.781
ToT	0.780
ScratchPad	0.781
Impersination	0.811

Table 9: Generated F1 BERTScore for prompts.

DEPTH	BERTSCORE F1
Depth 1	0.793
Depth 2	0.809
Depth 3	0.799
Depth All	0.811

Table 10: Generated F1 BERTScore for depth.

For BERTScore we get the best combined result with Impersonation as prompt (table 9), Llama3.2 as model (table 8) and Depth All as depth (table 10).

Evaluating the RAG LLM was generally found to be the most troublesome challenge. The Ragas evaluation framework seemed perfect for the task, but difficulties in integration meant that we could not perform the intended scale of evaluation. We had to lower the chunk size to a maximum of 128 to even generate results, therefore testing of various chunk sizes unfortunately had to be removed. Additionally, this meant that we only evaluated upon three question and answer sets where multiple still failed to generate a complete set of metrics. For example question 1 test of Depth 1 were missing all metrics but answer relevancy making it difficult to compare with other tests that generated all metrics.

To add a comparison to validate this evaluation, we tried to imitate the Ragas evaluation with the use of ChatGPT, this led to results only consisting of faithfulness and answer relevancy due to calculation complexities for the context metrics. Answer relevancy were generally

not too far off the Ragas results, but faithfulness scores especially for the prompt examples, were inconsistent and unreliable.

Lastly, we incorporated BERTScore into the evaluation of our RAG LLM to assess whether these evaluation tools are aligned and can complement or validate each other.

Generally, we can neither compare the evaluation methods of Ragas, ChatGPT and BERTScore directly, nor conclude that we have found the most optimized parameters for a RAG LLM. Therefore, based on the scale of the evaluation, we cannot confidently say that the results align with the expected outcome. The evaluation is too deficient to conclude that these results are generally valid when optimizing parameters for a RAG LLM. More testing will be needed to conclude the sweet spot of the parameters. For this reason, as well as the limitation of chunk size, the initial baseline model was used for further user evaluation.

7.3 USER TESTING LOCAL CHATBOT VS ELASTICSEARCH

To evaluate the performance of the RAG system as a chatbot versus Elasticsearch, a cognitive walkthrough was conducted with two computer science students who were unfamiliar with The Lord of the Rings (LOTR). Each participant was asked to answer six questions using both systems, which were designed to cover a range of question types, including two factual, two complex, and two open-ended questions, some examples can be seen in table 11.

The study was structured to ensure fairness and reliability. Each participant began by using one of the systems, either chatbot or Elasticsearch, to answer the six questions. Afterwards, they rated their experience on a Likert scale from 1 to 5 and provided additional feedback. They then repeated the same process with the other system, with the order of use alternating between the participants to minimize potential biases.

Once both systems had been tested, the participants took part in a semi-structured interview. This final step allowed them to express their experiences, share their preferences, and offer suggestions for improving the systems.

QUESTIONTYPE	QUESTION
Factual Question	Who created the one ring?
Complex Question	Why are the hobbits the ones to carry the ring?
Open-ended Question	Who is the greatest fighter of Gimli and Legolas?

Table 11: Examples of questions asked during the user testing

7.3.1 *Chatbot Results*

Participants generally found the chatbot simple to use, describing the chatbot as user-friendly. The system's ability to provide quick and contextually relevant answers stood out as one of its strongest attributes, contributing to a positive overall experience. The participants appreciated how the chatbot's conversational style made interactions feel natural and seamless, allowing them to focus on understanding the content rather than figuring out how to operate the system. Even though they pointed out the chatbot had a tendency to provide too much information for simple questions.

7.3.2 *Elasticsearch Results*

Using Elasticsearch proved to be significantly more challenging for the participants, requiring greater mental effort to navigate and interpret the results. Unlike the conversational ease offered by the chatbot, Elasticsearch presented its answers in the form of a list of search results, which often lacked clear prioritization or organization. Participants often struggled to identify the correct answer within these results, leading to moments of uncertainty and, at times, frustration.

This increased cognitive load occasionally caused the participants to resort to guesswork, attempting to piece together information from scattered results rather than finding a definitive answer. One participant suggested that the experience could have been improved if the search engine had provided better sorting or ranking of its answers, allowing faster identification of relevant information. In general, the process felt more labor intensive and less intuitive, ultimately affecting the satisfaction of the participants with the system.

7.3.3 *Comparison*

When comparing the two systems, both participants expressed a preference for the chatbot over Elasticsearch, particularly when it came to answering general queries. They appreciated the chatbot's simplicity, ease of use, and its ability to provide direct and somewhat relevant answers without requiring extensive effort to sift through results. The conversational interface of the chatbot also contributed to a more enjoyable and less mentally taxing experience, making it their preferred choice for quick, straightforward question and answer sessions.

However, participants recognized that Elasticsearch had potential advantages in certain scenarios. They noted that while it was less intuitive for general use, its ability to retrieve a wide range of information could make it better suited for in-depth exploration within a specific domain. In cases where users might need to dive deeper into a

topic or cross-reference multiple sources, Elasticsearch's broader data access could be an asset, provided that the search results were better organized or ranked. Despite this acknowledgment, the participants overall found the chatbot to be more practical and user-friendly for the tasks they were given during the study. The average score of each question of likert scale can be seen in table 12

METRIC	CHATBOT	SEARCH ENGINE
Happiness with answers	3.0	2.0
Overall Experience	4.5	2.5
Mental Demand	3.0	5.0
Effort to find answers	1.0	4.0
Irritation during use	1.0	3.5

Table 12: Average Likert Scale ratings rated from 1 to 5, with 5 being the best

8 DISCUSSION

8.1 DATA COLLECTION/SCRAPING

When scraping data the tables from the wiki was excluded due to the scraping tool we used was unable to process them correctly. This might have given us a problem in the evaluation when the participants were using Elasticsearch there were no tables to give a quick overview but they had to read dense blocks of text to find the answer they were looking for. Scraping the tables could also have made the RAG model able to answer questions in a shorter format instead of the long winded answers it currently gives. This might have slightly influenced the evaluation outcome, as participants felt that the RAG responses were generally too lengthy and found using Elasticsearch somewhat cumbersome due to the text being presented as a single block without tables.

8.2 HARDWARE LIMITATIONS

One significant challenge encountered during the development of this project was hardware restrictions, which affected the performance of the system. Specifically, the CPU (AMD Ryzen 5 5600H) processing times were noticeably long, with query response times reaching up to five minutes. On the other hand utilizing a GPU significantly improved performance of running the local LLM, reducing the response time to approximately 11 seconds when using an NVIDIA RTX 2060 graphics card. However, even with the GPU, initializing the vector database required a substantial amount of time taking around 10 minutes. These hardware limitations not only impacted the development process but also highlighted the resource-intensive nature of LLMs and their associated workflows.

8.3 LIBRARY AND DEPENDENCY CHALLENGES

Another challenge rose from the rapid evolution of libraries and dependencies used in this process. Many of the libraries are actively maintained and frequently updated, inconsistencies and compatibility issues became recurring issues. For example, the import path for ChatOllama functionalities changed from langchain_community.chat_models import ChatOllama to langchain_ollama import ChatOllama, necessitating updates to the codebase. The evolving nature of these libraries highlights the difficulties of working with cutting-edge technologies,

where maintaining stability and backward compatibility is not always prioritized.

8.4 RAGAS AS EVALUATION TOOL WITH SMALLER MODELS

The use of evaluation tools adds further complexity to the system. For example, in our case, Ragas requires data in specific formats. Questions must be presented as a list of strings, while the context is organized as a list of elements, with each element being a list of all contexts as strings associated with a specific question. Furthermore, version compatibility issues arise with each new version, requiring in depth search to backtrack to a version of the langchain community library that integrates with Ragas. Additionally, when running the Ragas evaluation we encountered a failure from the Ragas Output Parser, it seems there is a certain limit threshold on the output size of our LLM, Llama3.2, which gives the Ragas Output Parser nothing to evaluate for that specific entry and therefore fails the full evaluation. The problem seems to be with most available models which we have tested (Llama3.2, Mistral, Gemma2, Phi3). This unfortunately leads to a required smaller chunk size to shrink the output which potentially leads to less context and less meaningful answers.

8.5 STRENGTHS AND WEAKNESSES

RAG: Through the evaluation we learned that RAG is a nice and easy way to find an answer to a given question, but when users has to dive deeper into a subject RAG is not always the preferred tool to use. Another drawback of using RAG is that it does not always provide a straightforward answer, even for simple yes/no questions, this can confuse users by overwhelming them with unnecessary information they did not ask for. Furthermore, there is always a chance for it to hallucinate and give out wrong information, which the user likely will find as truthful. Most of the time it only provides one answer, which potentially can have big consequences depending on what subject the user is asking about e.g. medical or legal advice. This is also why chatbots like GPT4 or Gemini avoid these questions and ask the user to seek out real advisors like doctors or lawyers.

Elasticsearch: On the other hand, having to use Elasticsearch, the user has to read and understand more of the search results, since it does not just come with the answers for them, but the user has to go find it themselves in the texts. This of course takes longer and demands more from the user but it also lets the user understand the subject they search for better since they will have to come to a conclusion to the asked question themselves. This is backed by what the participants of the evaluation told us, they would use a search

engine if they had to deep-dive into a subject but if they just had a simple question, they would use a different tool for the task e.g. a chatbot.

9 STEPS TOWARDS BUILDING A LOCAL RAG LLM

This chapter presents some of the key aspects of how to build your own RAG LLM and what challenges this bring.

9.1 EASIER STEPS

The first sections describe some of the easier steps in creating your own RAG LLM.

9.1.1 *Web Scraper / Gather data*

An essential step is to extract relevant information which is focused upon the domain of interest. An efficient tool for this task would be to utilize a Web Scraper e.g. Scrapy, which is capable of handling this step. If you already have the documents on hand but just needs an LLM to know about these, langChain already have libraries to handle tasks for documents like PDF.

9.1.2 *Download Model to local device*

Select a suitable local Large Language Model (LLM), such as those available through Ollama e.g. Llama3.2.

9.1.3 *Vector Embeddings Retrieval*

When data has been collected, the data needs to be turned into vector embeddings to make sure the RAG system can identify and understand the semantics of the data.

Divide the embeddings into manageable chunks of a user-defined size with a specified overlap to preserve the context and meaning across splits. This prevents the LLM from losing crucial information if a document is split at an inappropriate point. Use LangChain to store the embeddings in a vector database such as ChromaDB.

When having to retrieve data and ask questions to the downloaded LLM, we have to create a Prompt Template which dictates how the LLM should behave.

This is where we instruct the LLM only to answer the questions on the provided context, the data which we give it. Configure a retriever to fetch the most relevant chunks of data. This retriever will pair the

context with user queries and deliver them to the LLM for accurate responses.

9.1.4 *User Interface*

Develop a simple user interface to interact with your RAG system. A straightforward method is to create a REST API using Flask, allowing websites or applications to communicate with the local LLM.

9.2 HARDER STEPS

The next sections highlights some of the more challenging parts in creating the RAG LLM.

9.2.1 *WebScrape with Tables*

While extracting some relevant information from a website can be easy, extracting all relevant information and correctly placing it in vector spaces for the LLM to understand, is difficult. Information from tables, which is often very factual and essential data, can be difficult to fetch and feed the LLM in a fitting format.

9.2.2 *Optimize the RAG LLM*

When trying to optimize the RAG, challenges arise. Optimizing specific parameters are crucial to achieve factuality and comprehensiveness. To do this, proper evaluation tools are needed, to validate the optimization. Ragas is a great tool for this, but since Ragas is using other LLMs to evaluate, difficulties with integration can be encountered, as with all new technologies there are constant changes. Libraries and dependencies change, documentation is limited or not updated and therefore, it can be troublesome to properly test the RAG LLM when trying to optimize.

9.2.3 *Computational Power*

Lastly, limitation of computational power can greatly affect the capabilities and usefulness of the LLM. Time is relative and depends on the use scenario but generally, the more computational power the merrier.

10 CONCLUSION

This project explored the creation and assessment of a local RAG LLM tailored for domain-specific applications. Using the Lord of the Rings as a case study, the research involved tasks such as data collection, system design and development, qualitative and quantitative evaluations to analyze the model's effectiveness in a specific domain. The comparison between the RAG LLM and Elasticsearch highlighted different strengths and weaknesses. The RAG LLM provided answers that were easy to understand and conversational in tone, which made it user-friendly for general queries. However, it occasionally produced overly detailed responses, which could be overwhelming for simpler questions. Elasticsearch, on the other hand, was more suited for retrieving detailed and complex information but required users to invest more effort in searching for results. This difference illustrates the trade-offs between conversational AI and traditional search engines in terms of usability and efficiency. The project encountered several challenges. Hardware limitations impacted the performance and usability of the system, particularly in response times and the initialization of vector databases. Additionally, keeping up with rapidly evolving libraries and dependencies posed difficulties, requiring frequent updates to maintain compatibility and functionality. These issues underline the practical complexities involved in deploying local LLMs. The evaluation process employed metrics such as BERTScore and the Ragas framework to measure performance. While these tools offered useful insights into optimizing parameters like prompt design and dataset size, the evaluations were limited in scope due to system incompatibilities. This limitation suggests a need for further testing and refinement to gain a clearer understanding of the system's capabilities and potential improvements. In conclusion, the project demonstrated the possibilities and challenges of implementing a local RAG-based LLM. While the system showed potential in delivering relevant and contextually appropriate responses, areas like computational efficiency, evaluation scalability, and user experience require further exploration.

AN APPENDIX

Links:

GitLab project:
https://gitlab.au.dk/bouvinkandidatgroup/rag-llm-versus-search-engine

Project video:
https://youtu.be/IGUzTIOGpNk

Evaluation test data:
https://docs.google.com/spreadsheets/d/1EzDcWeJHJvQldjCurP9dy_BOz8uctAtNnKYTPh2sPe8/edit?usp=drive_link

BIBLIOGRAPHY

[1] Kevin Matthe Caramancion. *Large Language Models vs. Search Engines: Evaluating User Preferences Across Varied Information Retrieval Scenarios*. 2024. arXiv: 2401.05761 [cs.IR]. URL: https://arxiv.org/abs/2401.05761.

[2] Yupeng Chang et al. "A Survey on Evaluation of Large Language Models." In: *ACM Trans. Intell. Syst. Technol.* 15.3 (Mar. 2024). ISSN: 2157-6904. DOI: 10.1145/3641289. URL: https://doi.org/10.1145/3641289.

[3] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. *Benchmarking Large Language Models in Retrieval-Augmented Generation*. 2023. arXiv: 2309.01431 [cs.CL]. URL: https://arxiv.org/abs/2309.01431.

[4] Rahul R. Divekar, Sophia Guerra, Lisette Gonzalez, and Natasha Boos. *Choosing Between an LLM versus Search for Learning: A HigherEd Student Perspective*. 2024. arXiv: 2409.13051 [cs.HC]. URL: https://arxiv.org/abs/2409.13051.

[5] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. 2023. arXiv: 2309.15217 [cs.CL]. URL: https://arxiv.org/abs/2309.15217.

[6] Yunfan Gao et al. "Retrieval-Augmented Generation for Large Language Models: A Survey." In: *CoRR* abs/2312.10997 (2023). DOI: 10.48550/ARXIV.2312.10997. arXiv: 2312.10997. URL: https://doi.org/10.48550/arXiv.2312.10997.

[7] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. "Challenges and Applications of Large Language Models." In: *CoRR* abs/2307.10169 (2023). DOI: 10.48550/ARXIV.2307.10169. arXiv: 2307.10169. URL: https://doi.org/10.48550/arXiv.2307.10169.

[8] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries." In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013.

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "Bleu: a Method for Automatic Evaluation of Machine Translation." In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[10] Sujoy Roychowdhury, Sumit Soman, H G Ranjani, Neeraj Gunda, Vansh Chhabra, and Sai Krishna Bala. *Evaluation of RAG Metrics for Question Answering in the Telecom Domain*. 2024. arXiv: 2407.12873 [cs.CL]. URL: https://arxiv.org/abs/2407.12873.

[11] Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. *ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems*. 2024. arXiv: 2311.09476 [cs.CL]. URL: https://arxiv.org/abs/2311.09476.

[12] Elena Vulpescu and Mihai Beldean. *Optimized Fine-Tuning of Large Language Model for Better Topic Categorization with Limited Data*. 2024. DOI: 10.21203/rs.3.rs-4697966/v1. URL: https://doi.org/10.21203/rs.3.rs-4697966/v1.

[13] Cunxiang Wang et al. "Survey on Factuality in Large Language Models: Knowledge, Retrieval and Domain-Specificity." In: *CoRR* abs/2310.07521 (2023). DOI: 10.48550/ARXIV.2310.07521. arXiv: 2310.07521. URL: https://doi.org/10.48550/arXiv.2310.07521.

[14] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. "BERTScore: Evaluating Text Generation with BERT." In: *CoRR* abs/1904.09675 (2019). arXiv: 1904.09675. URL: http://arxiv.org/abs/1904.09675.

[15] Wenxuan Zhang, Hongzhi Liu, Zhijin Dong, Yingpeng Du, Chen Zhu, Yang Song, Hengshu Zhu, and Zhonghai Wu. "Bridging the Information Gap Between Domain-Specific Model and General LLM for Personalized Recommendation." In: *Web and Big Data*. Ed. by Wenjie Zhang, Anthony Tung, Zhonglong Zheng, Zhengyi Yang, Xiaoyang Wang, and Hongjie Guo. Singapore: Springer Nature Singapore, 2024, pp. 280–294. ISBN: 978-981-97-7232-2.

[16] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2024. arXiv: 2303.18223 [cs.CL]. URL: https://arxiv.org/abs/2303.18223.