From Fine-Tuning to Retrieval-Augmented Generation: Enhancing Document Question Answering with Large Language Models

Jay Agrawal
230913640
Paulo Oliva
MSc. Computer Science

Abstract—**This research paper aims to discuss the challenges and solutions associated with fine-tuning Large Language Models (LLMs) for question-answering and information retrieval tasks for lengthy documents. At first, the goal of the work was to enhance the performance of BERT-based models for the tasks specific to the legal field. However, due to token constraints of models such as BERT or RoBERTa, models with higher max input lengths like LLaMA were considered. Even though the fine-tuned models showed improvements to the context length, models had issues with hallucination, which pushed for the use of a Retrieval Augmented Generation (RAG) system. In this paper, all the problems experienced in this development process in fine-tuning LLMs for large legal documents, as well as the creation of an RAG-based document chat application, are discussed. The work advances the research by identifying the shortcomings of fine-tuning strategies and proving that RAG systems can reduce hallucinations while generating accurate and contextually appropriate language in legal contexts.**

Keywords— Fine-tuning, Large Language Model (LLM), question answering, tokenization, hallucinations, Natural Language Processing (NLP), Retrieval-Augmented Generation (RAG), Quantized Low-Rank Adaptation (QLoRA), Quantization, Parameter Efficient Fine Tuning (PEFT), Vectors, Embeddings, Chunking, Langchain.

I. INTRODUCTION

The development of NLP has grown, especially in recent years, creating new opportunities for automation and improvement of various activities in several domains. One such application is the creation of systems that could answer questions related to legal documents and contracts. This dissertation aims to explain the process of developing an efficient and accurate document-question-answering application for legal professionals, with details of difficulties observed and the measures taken.

Firstly, my work mainly involved improving large language models such as BERT and RoBERTa to answer questions in legal documents. These models have shown excellent performance across various NLP tasks. However, I soon realized certain constraints because of the input token size of these models (512 tokens), which is not suitable for long documents, especially legal ones. This eventually forced me to attempt to resolve this constraint by trying more sophisticated LLMs that have a higher input token limit, such as LLaMA.

Despite the improved token capacity, fine-tuning LLaMA on legal domain contracts revealed another significant challenge: the problem of the model hallucinating or producing wrong imaginary information. When presented with new queries, the model memorized the training dataset, resulting in irrelevant outputs. This problem highlighted the need for an alternative approach that could maintain accuracy while leveraging the power of large language models.

These issues prompted the use of a Retrieval-Augmented Generation system. RAG integrates the features of information retrieval with the generation capacities of large language models, thus being as more precise and context-oriented. The retrieval process in RAG also involves a mechanism that checks the relevance of the information retrieved with the input query or context. RAG enhances the relevance of the response given by the LLM by supplying it with contextually appropriate information. This contextual grounding is beneficial in minimizing the production of irrelevant responses [1].

The transition to a RAG system addresses two key challenges – Firstly, RAG can handle much longer contexts by retrieving relevant information from a large text, effectively bypassing the token limitations of the LLMs. Secondly, compared to the computer's generation of false or irrelevant information, RAG systems have the ability to ground their responses in the information they have retrieved.

In this discussion, we show the technical difficulties, the choice of methods, and the facts discovered that contributed to this shift. The results are helpful to build in building more effective and precise question-answering systems for documents where the context length is very large, and it is critical to be precise while answering the questions. In addition, we explain the details of our final document chat application using the RAG and show that our RAG-based model works effectively in addressing users' queries related to provided legal documents.

Therefore, addressing these issues and explaining how it has been solved, this study contributes to the literature in employing the advanced NLP techniques for the domains that have large textual content. These findings suggest that future work in the development of additional accurate question answering systems should consider two features- large context length and accuracy- outside of legal text analysis.

II. LITERATURE REVIEW

Many recent studies have explored the comparison between fine-tuning and retrieval-augmented generation (RAG) for enhancing large language models (LLMs), particularly in handling domain-specific or less popular knowledge. These studies offer crucial information for my paper's research on using RAG for the document chat application for legal contracts.

A. Fine-Tuning LLMs

The first paper, *"A Study of Optimizations for Fine-tuning Large Language Models"* by *Singh et al*. [2], gives a comprehensive overview of the optimizations for fine-tuning large language models. Although their main contribution is fine-tuning, it was beneficial to know how they monitor and measure memory usage and the time for execution during the fine-tuning phase for my project. The solutions for fine-tuning very large models and allowing for considerable context lengths are crucial issues I encounter in adapting LLaMA for legal contract analysis. In addition, their proposed optimization mixtures for fine-tuning under GPU resource constraints give me concrete suggestions on how I can implement cost-efficient fine-tuning methods in my research, possibly by utilizing techniques such as LoRA and quantization (QLoRA) to enhance the fine-tuning procedures for LLaMA models while concurrently handling the constraints of the GPU resources.

A similar study, *"LoRA: Low-Rank Adaptation of Large Language Models"* by *Edward et al.* [3], also focuses on fine-tuning large language models using Parameter-Efficient Fine-Tuning (PEFT) techniques, specifically Low-Rank Adaptation (LoRA). The authors showed that adopting this strategy could cut the amount of computational effort by a considerable margin, retaining excellent results across multiple NLP tasks. They described more precise performance of their models and decreased time for training compared to full fine-tuning. However, the paper also noted limitations in handling very long documents and occasional hallucinations in complex reasoning tasks. This research is valuable for my dissertation as it explores advanced fine-tuning techniques that I considered before moving to RAG. It gives me an understanding of the capabilities and downsides of the current fine-tuning methods for LLMs and thus helps with the reasoning behind searching for an alternative approach, such as RAG, to work with large legal documents and mitigate hallucinations.

B. Mitigating Hallucinations in LLMs

Addressing the hallucination challenge in LLM development, *Islam et al.* [4] present *"A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models"*, giving a broad overview of the available approaches for managing hallucinations in LLMs, which clearly relates to our project switching from fine-tuning to the RAG model for legal contract analysis. The authors discuss over 32 techniques, of which Retrieval Augmented Generation is among them and has been used in the design of our system. The paper offers good groundwork for handling hallucinations in LLMs by using the RAG system. This survey supports the value of our approach and provides suggestions for enhancing the factual content and accuracy of specific legal document analysis.

C. Comparative Study: Fine-Tuning vs. RAG

A comparative research, namely, *"Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge"* by *Soudani et al.* [5], presented a detailed analysis to compare fine-tuning with Retrieval Augmented Generation (RAG) in relation to the representational capability of less frequent knowledge in LLMs. In various experiments, the researchers have tested different kinds of LLMs and assessed the results in tasks concerning low-frequency entities. Analyzing the results, the authors stated that fine-tuning offered significant advantages that enhanced the model's performance when working with entities within the context of varying popularity. However, each time, RAG proved to yield higher overall performance in comparison with other methods, particularly in the treatment of domain-specific or low-frequency knowledge. These results are helpful to identify the strengths and weaknesses of each of these methods for handling domain-specific question answering.

Another study by *Lewis et al.* [6] proposed the idea of Retrieval-Augmented Generation and experimentally compared it with fine-tuning regarding several knowledge-related NLP tasks. This is the work that lays down the basis for our analysis, as the authors also proved that RAG models work better than fine-tuned models when it comes to accomplishing different tasks, such as open-domain question answering. More importantly, they explain how RAG can work with large volumes of information while not necessarily requiring frequent updates to models, which is essential to our application in the legal domain. As a result of this study, I now have a basis for moving forward in investigating the use of RAG instead of fine-tuning to increase accuracy and reduce hallucinations in contract analysis.

D. Implementing RAG

Showcasing a practical application of RAG, *Gupta and others* [7] used Hugging Face embeddings based on *LangChain* and *Sentence Transformers* known as Legal Magician, which is legal-oriented and accurately constructed, focusing on the Indian Constitution's texts. They used *ChromaDB* and *LangChainVectorStore* technologies for analysis, as the data had to be pre-processed and split into parts. Based on the results of the work, the presented approach can be further used to continue the analysis of legal language and can be considered one of the promising tools for effectively processing significant legal texts. This research is also essential for our project methodology because it describes how this kind of model and embedding is used for the legal domain.

These research papers firmly state that using an RAG system is more efficient than fine-tuning a model for question answering with large contexts. Hence, I decided to switch from fine-tuning LLMs to building a Retrieval Augmented Generation system to deal with large context documents and eliminate hallucination issues in the fine-tuned model.

III. METHODOLOGY

A. Problem Approach

The problem approach began with fine-tuning large language models on a good legal domain labelled dataset. I chose BERT (Bidirectional Encoder Representations from Transformers) base models as they are effective in natural language processing tasks like question answering and encoding context information bi-directionally. However, BERT base models like RoBERTa have only a limit of 512 tokens, which was insufficient for my project scope as the legal documents are much larger than those of 512 tokens. The results were good for shorter context length. However, the token limitation led me to explore LLMs that have larger input token limits and found the *LLaMA* model, which has a significantly large token limit of 4096 tokens. Despite having a large context window, the fine-tuned model showed

hallucinations when tested, prompting a shift towards implementing an RAG system.

B. Data Collection

For the fine-tuning phase of our research, we utilized the *Contract Understanding Atticus Dataset (CUAD)* dataset available on Hugging Face [9]. This dataset includes a variety of legal agreements, such as license agreements, distributor agreements, and many more. The choice of such a dataset was made based on the fact that it offers a wide variety of legal documents, which means there would be a profound range of both styles and structures used in legal contexts. This dataset had a total of 510 legal documents that have question and answer pairs with their respective contexts, which are the main categories to fine-tune a model for context-based question answering.

Now, to prepare the dataset for model fine-tuning, we implemented some pre-processing steps. After investigating the dataset, I found that there were many samples that had questions but did not have answers. This could add noise to our fine-tuning process, so I decided to drop the samples that had no answers. Also, the questions had unnecessary details that were not relevant to training the model, so I cleaned them out as well. Before pre-processing, the dataset had 22450 training samples and 4182 test samples. After pre-processing, we had 11180 training samples and 1244 test samples.

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 11180
    })
    test: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 1244
    })
})
```

Fig. 1. Pre-processed dataset

To make proper estimations regarding our data and choose a proper model and necessary fine-tuning, we decided it would be helpful to analyze the context length distribution of documents after pre-processing. Fig. 2 below shows a histogram depicting the context length in words and the sample frequency. The average length is around 12272 words, which is approximately but not accurately 16000 tokens, which is very large even for LLaMA models with a 4096 tokens limit.

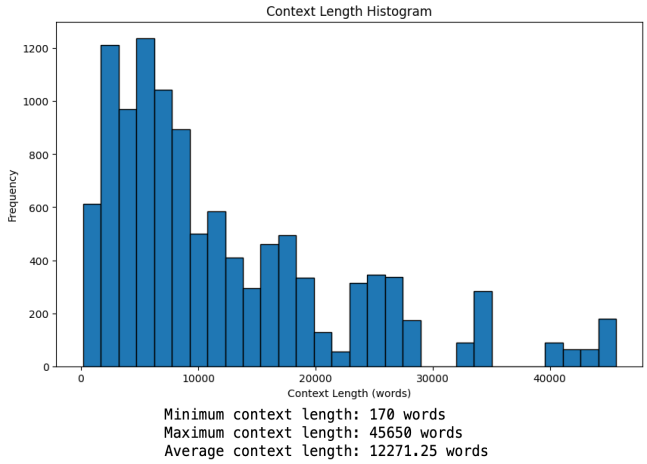


Fig. 2. Context Length Analysis

This analysis pointed out the necessity of adjusting models like BERT and LLaMA to work with longer contexts. Because of the limit on the context length, it is necessary to process most of the documents in our corpus without truncating text portions.

C. Fine-Tuning BERT model

For the first approach, BERT was selected because of its ability to demonstrate high contextual understanding, which the model excelled at when tested on various NLP tasks. However, its token limit of 512 tokens proved to be very limiting for our application as it mainly deals with large legal documents. I utilized the *"roberta-base-squad2"* variant of the BERT model, which is a fine-tuned model of the *"roberta-base"* model [11]. This model is fine-tuned for t downstream task of question answering using the *SQuAD* dataset. Fig. 3 below shows the fine-tuning flow diagram for BERT models.

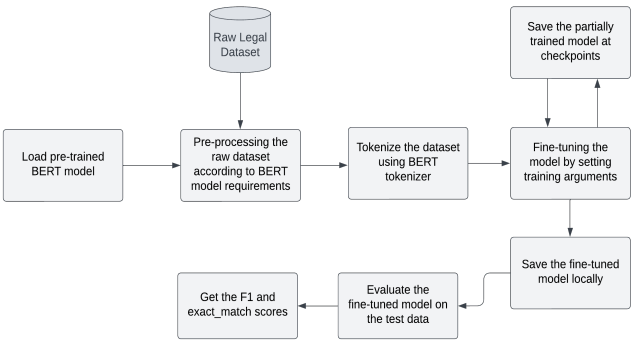


Fig. 3. BERT Fine-tuning process

The process starts with loading the *RoBERTa* model to the GPU available from *Apocrita's HPC* [8]. BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that has been pre-trained on a large corpus of text. This pre-training provides the model with a strong understanding of language, making it suitable for various downstream tasks such as text classification, question answering, and more.

After loading the model, the next step is to pre-process the training dataset before fine-tuning the model. BERT models have a token limit of only 512 tokens, and according to our context analysis, we have an average context length of 16000 tokens (excluding the user prompt input tokens). Hence, there were two ways to manage the context: the first was to drop the samples that were beyond the token limit, or the second option was to chunk the documents into individual samples. Choosing the first option would have resulted in a massive loss of crucial context in data samples, as most of the context was lengthy. Hence, I chose the second option of chunking large contexts into individual samples.

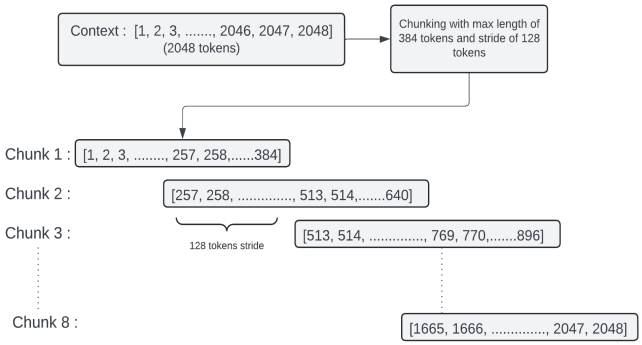


Fig. 4. Chunking context illustration

Fig. 4 shows an example of how a context of 2048 tokens gets chunked into eight individual chunks. Using this chunking method keeps the context by dividing it into multiple samples. However, there was a limitation to this chunking technique as it would add noise to our training dataset. For instance, if a document is chunked into eight parts and there is one question-answer pair. There would only be one chunk among these eight chunks where our answer would be present in the context, and the remaining chunks would count towards adding noise to our dataset as there are no answers present in these chunks. The dataset also had many samples with no answers, which were useless for training the model as it would add more noise to the dataset. Hence, such samples were dropped from the training dataset.

After the pre-processing step is completed, we tokenize our training data. Large language models are trained to process tokens, not raw text. Tokenization converts text into a format that these models can understand and process. It also handles padding and truncation and creates attention masks required for BERT's input. We tokenize the question, and the context features from the dataset as these are the inputs on which the model would be trained. Now we set up the BERT model for training using several training arguments. TABLE I below shows the list of training parameters that I chose according to the dataset and computational resources.

TABLE I. TRAINING PARAMETERS FOR BERT MODEL

Parameter	***Value***	***Description***
learning_rate	2e-5	A low learning rate to ensure gradual fine-tuning, preventing overfitting
max_length	384	Ensure that the tokenized input does not exceed 384 tokens
per_device_train_batch_size	32	Batch sizes optimized for the available computational resources
num_train_epochs	1	A single epoch to avoid overfitting on the limited dataset
weight_decay	0.01	Regularization to prevent overfitting
stride	128	Allows overlapping tokens between chunks, preserving context continuity
dataloader_num_workers	4	Utilize multiple CPU/GPU cores for data loading
save_steps	100	Save checkpoints at every 100 steps

After the fine-tuning, the model is saved locally to ensure it can be reloaded and used for inference or further training in the future. The fine-tuned model is evaluated using a separate test dataset to measure its performance. This step is crucial to ensure the model generalizes well to unseen data. The performance of the model is evaluated using evaluation metrics like the F1 score and Exact Match (EM) score. These metrics provide insight into the model's accuracy and completeness in predicting the correct outputs.

D. Fine-Tuning LLaMA model

For the second approach, the LLaMA model was selected because of its high input token limit of 4096 tokens. This model was chosen for its ability to handle large context lengths, which is essential for lengthy document analysis. I chose the *LLaMA-2-Chat* model [17] for fine-tuning, as the chat variant outperforms various other open-source chat models. This model is large in size as it has 7 billion parameters, so it requires a GPU with quite a large memory to load the model. Fig. 5. shows the process of fine-tuning LLaMA model step by step.

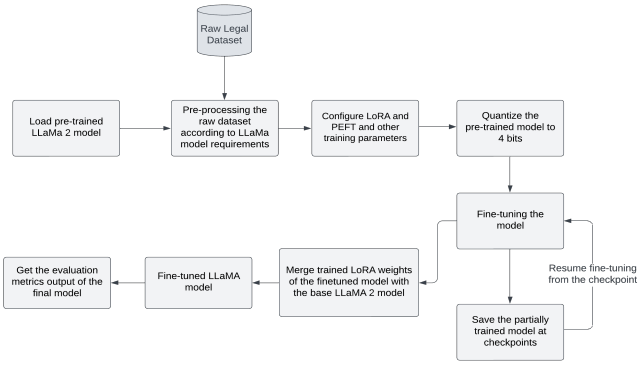


Fig. 5. LLaMA Fine-tuning process

Like BERT, the raw dataset msut be pre-processed according to the LLaMA model's requirements. Unlike the pre-processing step in BERT, where the context and question were handled separately, pre-processing is different for LLaMA. The training dataset is converted to having only a single column named "text", where the context, question and answer are merged into a prompt. Fig. 6 below shows the official prompt template by Meta to follow for LLaMA 2 chat models.

```
<s>[INST] <<SYS>>
{your_system_message}
<</SYS>>

{user_message_1} [/INST] {model_reply_1}</s>
```

Fig. 6. LLaMA 2 Chat prompt template

In the above template, the system message is generally an instruction given to the model on how to give a response. User message refers to the question with the context that the user will ask, and finally, the model reply is the response given by the model. I merged the question and context in the user message, and the answer in the model reply part of the template. This way, there was only a single column in the training dataset after pre-processing. Fig. 7 below shows how I created the prompt using the prompt template.

```
f'''<s>[INST] You are a helpful assistant.
Answer the given question from the context provided.

Context:
{context}

Question:
{question}

If you cannot answer the question from given context
then don't try to make up an answer.[/INST] {answer} </s>'''
```

Fig. 7. Prompt preparation

LLaMA is a huge model with 7 billion parameters, and loading the model required some techniques to help accelerate the fine-tuning process and the utilization of computational resources. Thus, the *Low-Rank Adaptation (LoRA)* method with *Parameter Efficient Fine-Tuning (PEFT)* was used to fine-tune the LLaMA model [3] effectively.. LoRA is the technique of fine-tuning several parameters of the pre-trained models to the new task with few parameters. The given method is an important part of the larger strategy known as

PEFT. Instead of modifying the whole model, LoRA adds low-rank matrices in each layer, and the rank matrices are learned during the fine-tuning phase. This makes the adaptation efficient in terms of computational complexity as well as memory space complexity. Here, it is important to note that in PEFT, we update only a small portion of the model's parameters, making the process much faster. This means that PEFT can obtain high performance with the low utilization of resources, and that is why it is effective in large models such as LLaMA.

After configuring the LoRA parameters, we quantize the LLaMA model down to 4 bits. Quantization means changing the parameters of the model to a lower-bit representation. This makes the model have a smaller size, and the time taken to make an inference is also reduced while the accuracy is comparatively close to the original model. After the quantized model is loaded, we proceed with the fine-tuning process with the help of the pre-processed data and certain train parameters. Below, TABLE II shows the list of training parameters chosen according to the dataset and computational resources.

TABLE II. TRAINING PARAMETERS FOR LLAMA MODEL

Parameter	***Value***	***Description***
lora_r	32	Determines the rank for low-rank factorization in LoRA
lora_alpha	64	Controls the trade-off between learning new tasks and preserving original knowledge
lora_dropout	0.1	Prevents overfitting by randomly setting some activations to zero during training
use_4bit	True	Enables 4-bit quantization. Reduces the model size and computational load while maintaining performance
bnb_4bit_compute_dtype	float16	Float16 is used to balance precision and computational efficiency
use_nested_quant	True	Enables nested quantization. Further reduces model size and computational requirements by applying additional quantization layers
num_train_epochs	1	Chosen for quick and efficient training
per_device_train_batch_size	2	A smaller batch size is chosen to fit within memory constraints while maintaining effective training
gradient_accumulation_steps	2	A smaller batch size is chosen to fit within memory constraints while maintaining effective training
learning_rate	2e-4	A moderate learning rate was chosen for stable and efficient training
weight_decay	0.001	Regularization to prevent overfitting
max_length	4096	Maximum length for input data
lr_scheduler_type	cosine	Reduces the learning rate following a cosine curve, which helps in gradually reducing the learning rate

After the fine-tuning, the trained LoRA weights are merged with the base LLaMA 2 Chat model. This step ensures that the updated or learned weights are integrated into the pre-trained model after fine-tuning. Finally, we test the fine-tuned model on the test data by creating a *huggingface* pipeline. The results were not very promising, as many of the model responses were repetitive and showed hallucinations, and so I explored RAG to mitigate this.

E. Retrieval Augmented Generation (RAG)

The RAG system was used to overcome some of the issues noted with fine-tuning large language models such as LLaMA. It is, therefore, more effective when the context size is large and also reduces the level of hallucinations as compared to the traditional retrieval-based and LLM generative systems. The RAG system architecture consists of several key components, namely document processing, embedding generation, vector storage, retrieval, re-ranking, prompt creation, and response generation steps.

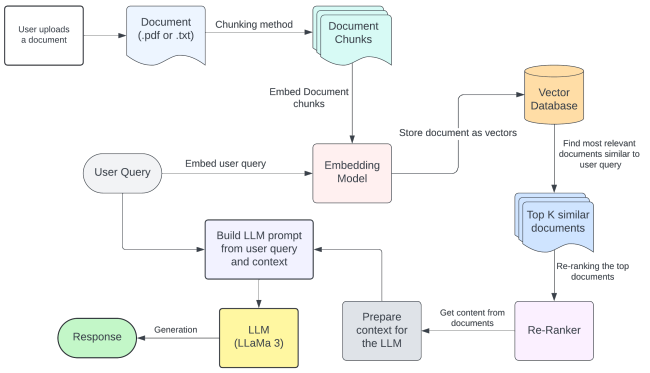


Fig. 8. RAG process

The flow of the system is depicted in the following Fig. 8. The whole process involves the following steps:

- Chunking Document: The first step is to use a chunking method to split the document into separate chunks of a specific size and overlap.
- Embedding Generation: The query text and chunked documents are transformed into dense vectors using the Sentence Transformer embeddings.
- Initial Retrieval: The semantic search is done using the FAISS vector store that returns a list of top K relevant documents that are related to the user query.
- Re-Ranking: This model estimates the relevance of each document-query pair and gives a score to it.
- Selection: The documents which are ranked high are used as context in the generation phase.
- Response Generation: The selected chunks are passed to the LLaMA model, which generates a response based on the provided context and query.

A critical step to an efficient RAG system is a document chunking strategy. I used the *"RecursiveCharacterTextSplitter"* method from the *"langchain"* library to chunk documents [20]. The text splitter divides each document into 1024 characters, with 256 characters overlapping between consecutive chunks. The decision to use a chunk size of 1024 characters is to ensure enough context is retrieved while simultaneously avoiding the retrieval of large and unnecessary data. This size is also feasible for the vector store and co-occurs with the model's processing capacity at this size. By having 256 characters that overlap between the chunks, it becomes possible to reduce the loss of context due to chunk boundaries where vital information is likely to be cut off and can be retrieved.

For embeddings, the model called *"all-MiniLM-L12-v2"* from Sentence Transformers was chosen. This model is optimized for both computational complexity and the quality of embedding needed for large-scale document retrieval applications. Embeddings are used to convert text into high-dimensional vectors that are easy to index and compare. We will use the embedding model to process the user query as well as the chunked documents. The chunked documents are then stored in a vector database, namely, *FAISS (Facebook AI Similarity Search)* [19].

The generative component of the RAG system uses the *"Sanctum AI/Meta-Llama-3-8B-Instruct-GGUF"* model based on the LLaMA architecture specifically designed for instructive tasks. The model is downloaded from the Hugging Face Hub using *"hf_hub_download"* to ensure access to the latest version. Key parameters include *n_ctx=4096* for context length and *temperature=0.0001* to minimize randomness in responses, promoting consistency and factuality. The model is loaded with the help of the *LlamaCpp* library, which allows for fast inference on both CPU and GPU.

Another key step for RAG system to give accurate response is prompt creation. The structure of the prompt template aims at helping the LLM providing accurate and context-related responses. They directly prohibit the model from making up information, which is a problem with hallucination that was seen in previous attempts at fine-tuning. In our prompt creation implementation, the template expects a context and a question and underlines that the answer should be derived exclusively from the context. This design ensures that the model remains anchored to factual information.

One of the improvements in our RAG system is the addition of a re-ranking step involving a Cross-Encoder model namely, *"cross-encoder/ms-marco-MiniLM-L-12-v2"*. The re-ranker then filters the initial retriever's data set through a score function that measures the document's relevance to the query. This step helps prioritize the most appropriate documents to improve the outcomes of the generated responses. Below, TABLE III Shows the configuration and parameters used for an efficient RAG model implementation.

TABLE III. RAG SYSTEM CONFIGURATION

Component	***Value***	***Description***
LLM	Meta-Llama-3-8B-Instruct	Generates response to the user query.
Embeddings	all-MiniLM-L12-v2	Converts text to high dimensional vectors
Re-Ranker	cross-encoder/ms-marco-MiniLM-L-12-v2	Scores the top relevant document chunks based on user query
Vector DB	FAISS	Used to retrieve top K similar documents based on user query
Chunk Size	1024	Size of each chunked document
Chunk Overlap	256	Overlap between chunks to maintain context continuity

The above-proposed RAG system effectively handles large legal documents, which was not a strength in the fine-tuning approaches. Combining the retrieval and generation components of the RAG system helps minimize hallucinatory answers and make legal document analysis more accurate.

F. RAG System Design and Integration

The design and integration of the document chat application play a central role in delivering a positive user experience and good performance. This section focuses on the general structure of the whole system, database management, and API integration, giving the reader a better understanding of how the different aspects of the system work together to achieve the intended purpose of the application. It consists of both backend and frontend components, each playing a crucial role in the system's operation. Following are the components used in both frontend and backend development:

- React.js: The frontend is developed using React.js, a popular JavaScript library for building user interfaces. It facilitates a responsive and dynamic user experience.
- Flask: The backend is built using Flask, a lightweight framework for Python. It handles user authentication, document processing and query handling.
- MongoDB: A NoSQL database used to store user data, documents and chat history.
- RAG system: The whole RAG model is implemented in the backend using *Langchain* and *FAISS* vector database as discussed in the previous section.

These components are developed coherently to ensure they all work together in a manner that is smooth for the end user. The frontend and backend are connected through RESTful APIs, where the frontend is responsible for receiving inputs from the users while the backend is responsible for processing information and interacting with the LLM. Below TABLE IV shows all the API endpoints used in the app:

TABLE IV. API ENDPOINTS

Endpoint	***Method***	***Description***
/auth/signup	POST	Creates a new user
/auth/login	POST	Authenticates user and return access and refresh tokens
/auth/refresh	POST	Refreshes the access token after it has been expired
/document/upload	POST	Uploads and processes the document sent by the user
/chat/list	GET	Get list of all chats of logged user
/chat/:chat_id	GET	Get chat data of a specific chat
/chat/delete	DELETE	Delete a specific chat
/chat/query	POST	Processes a query and returns generated response

The system architecture flow is depicted in Fig. 9, which explains the whole process from frontend to backend. Users interact with the frontend, built using React.js, to sign up, log in, upload documents, and ask questions. The frontend sends API requests to the backend, implemented with Flask, for user authentication, document upload, and querying. Upon document upload, the backend processes the document by converting pdf to text using *PyPDF* library, then chunks it using *RecursiveCharacterTextSplitter*, and stores the chunks in MongoDB. Now, when the user submits a query, the backend retrieves relevant document chunks from MongoDB using the FAISS vector store and re-ranks them with a *CrossEncoder*. The RAG system then generates a response

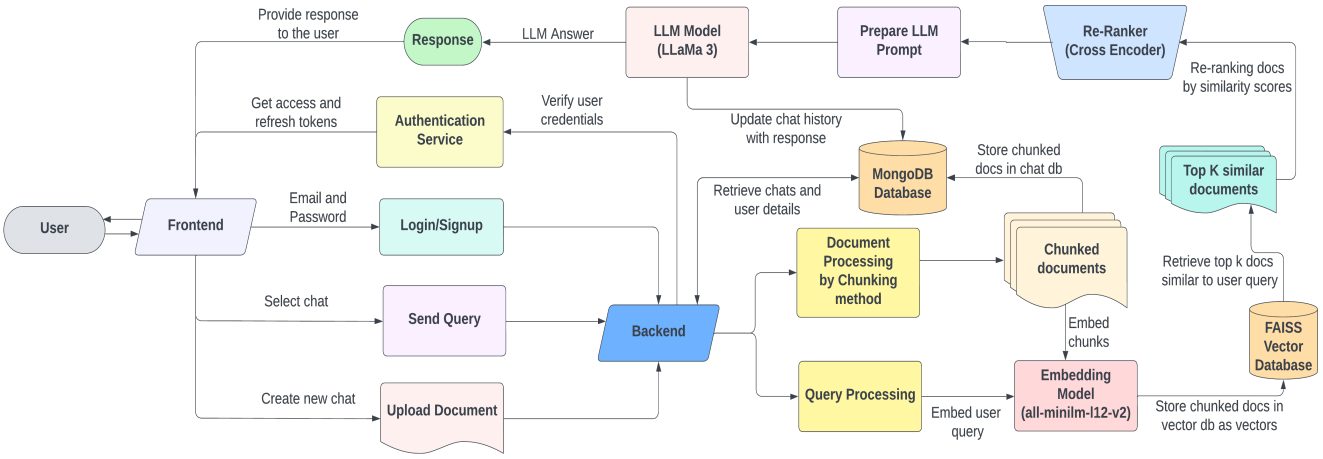


Fig. 9. System Architecture Flow

based on the retrieved chunks and returns it to the frontend, where it is displayed to the user. By integrating these components, the system ensures efficient document processing, accurate query handling, and a good user experience. The detailed design and integration of the backend and frontend components, supported by a robust database and API layer, form the backbone of our document chat application.

IV. RESULTS AND TESTING

In this section, we detail the comprehensive evaluation and testing methodologies employed to assess the performance and robustness of our Retrieval-Augmented Generation (RAG) system. The evaluation spans from fine-tuning language models to testing the integrated RAG system, as well as ensuring the reliability of the backend and frontend components by unit testing.

A. Fine-Tuned Models Evaluation

BERT and Llama, were fine-tuned and evaluated using the SQuAD (Stanford Question Answering Dataset) metrics [16], which include F1 scores and Exact Match (EM) scores. SQuAD metrics are more suitable for measuring question-answering models, which is why the proposed models are evaluated using it as well. The F1 score gives a balance between the precision and the recall rate, through using the harmonic mean. The EM score indicates the percentage of predictions that exactly match the ground truth answers, providing insight into the model's accuracy.

The BERT model achieved an F1 score of *56.54* and an EM score of *45.25*. These outcomes show that BERT can capture context and give accurate answers due to its architecture that allows it to recognize complex dependencies in text. However, these results were achieved with context limit of only 512 tokens. Due to pre-processing the training dataset and input context limit of only 512 tokens, the results obtained are moderate.

The results for LLaMA model were not what I expected compared to the BERT model. The Llama model, recorded an F1 score of *20.23* and an EM score of *0.0*. The results obtained are very less compared to BERT model. Although, LLaMA model's input limit was kept at 4096 tokens but the results obtained shows that the model was hallucinating on the test dataset. The response of the LLaMA model was repetitive and irrelevant for some of the queries. In some test cases the model after giving the answer in its response asked another question to itself and giving the answer to it again. This showed that the model was memorizing the training dataset and giving irrelevant responses. I shifted my focus from BERT to LlaMA due to token limitation, but as the results of LLaMA model were not good, I tried to mitigate the hallucinations by building an RAG model.

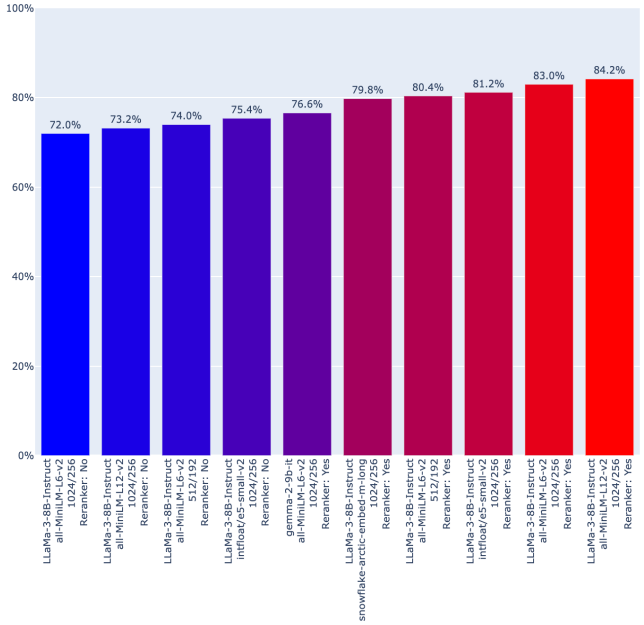
B. RAG Evaluation

As there is no available metrics to calculate the RAG model's performance, we leverage the use of LLMs to evaluate the RAG system. A study by Zheng et al. [22], explains how strong LLMs can be used as judges to evaluate the performance of models. A definitive cookbook available on Huggingface [21], showed the process of evaluating the RAG system performance. The RAG system was evaluated using a sophisticated approach involving synthetic dataset generation and a *Llama-3-70B* model as a judge. This model was chosen due to the high count of parameters (70 billion) that improve the capability for the evaluation of the consistency and fluency of the generated replies.

A synthetic dataset was created using a text-generation pipeline with the Llama-3-70B model. This dataset comprised question-answer pairs generated from a large sample document, simulating real-world queries and responses. The RAG system was tested with various configurations, including different chunk sizes, chunk overlaps, re-ranker, embedding models, and LLMs. Each configuration was evaluated by comparing the generated answers against the synthetic dataset using a scoring rubric. The rubric assessed the accuracy, correctness, and factuality of the responses, with scores ranging from 1 (completely incorrect) to 5 (completely correct).

The choice of chunk size and chunk overlap was a critical factor in the RAG system's performance. Larger chunk sizes (1024 characters) generally provided more context for the LLM, enhancing its ability to generate accurate responses. Smaller chunk sizes (512 characters) can provide very less context to the LLM to generate an accurate response. In Fig. 10, we can see that configurations with 1024 chunk size strategy have good accuracy scores compared to smaller chunk size. The use of a re-ranker significantly improved the quality of the generated responses. By re-ranking the retrieved document chunks, the system was able to prioritize the most relevant information, leading to more accurate answers. Configurations utilizing the *cross-encoder/ms-marco-MiniLM-L-12-v2* re-ranker consistently outperformed those without, demonstrating the re-ranker's effectiveness in

refining the context for the LLM. Fig. 10 below shows the accuracy scores of different configurations used in the RAG model. It can be seen that using a re-ranker improved the accuracy scores in every configuration. Most of the configurations had *LLaMA-3-8B-Instruct* model to generate responses. I also tried using *gemma-2-9b-it* model [23], and the results were decent but not that great as compared to the LLaMA-3 model. In the below plot, the fifth column shows that gemma model had 76.6% accuracy whereas with the same configuration the LLaMA model in ninth column had 83% accuracy. Hence, I decided to test LLaMA-3 model with other configurations.



Fig, 10. RAG Evaluation with different configurations

The final decision of which embedding model to choose was vital to the system's performance in the documents chunks retrieval. Some of the embedding models provided better semantics and context than the others. The *all-MiniLM-L12-v2* embedding model was the top performing model compared to other models like *intfloat/e5-small-v2* and *all-MiniLM-L6-v2*. The *all-MiniLM-L12-v2* model is slightly better than The *all-MiniLM-L6-v2* model as it has 12 layers (L12) which is more than the latter with 6 layers (L6). This model when used with 1024 chunks and re-ranker showed the highest accuracy of 84.2%. Hence, I decided to use this configuration in my final RAG application.

Overall, the evaluation process provided valuable insights into the optimal configuration for the RAG system, highlighting the importance of carefully selecting chunking strategies, LLMs, re-rankers, and embedding models to enhance response quality and accuracy.

C. Application Testing

In addition to model evaluation, comprehensive unit testing was conducted on both the backend and frontend components to ensure robustness and reliability. This phase was a crucial component of the development process, ensuring that both the backend and frontend components of the Retrieval-Augmented Generation (RAG) application functioned correctly and efficiently.

The backend was tested using *PyTest*, a powerful testing framework for Python. PyTest was chosen for its simplicity, scalability, and ability to handle complex testing scenarios efficiently. Unit tests were implemented to verify the functionality of individual components, such as authentication, document processing, and response generation. These tests ensured that each module performed as expected and interacted seamlessly with other components. The frontend was tested using *Jest* and *React Testing Library* to simulate user interactions and verify the UI's responsiveness and accuracy. Unit testing and Integration testing were used to test the functionality of the frontend application. Tests covered various scenarios, including user authentication, chat management, and query submission, ensuring that the application provided a smooth and intuitive user experience.

Overall, the evaluation and testing processes were integral to the development of the document chat application, providing insights into the fine-tuned model performance, RAG system optimization, and application reliability. The methodologies employed ensured that the system was not only effective in generating high-quality responses but also robust and user-friendly.

V. Conclusion and Future Work

This dissertation presented the development and evaluation of a sophisticated Retrieval-Augmented Generation (RAG) system, designed to enhance the accuracy and relevance of responses generated by language models. Through comprehensive evaluation, of model fine-tuning and our RAG model, the RAG system has proven effective in generating accurate and contextually relevant responses. The application testing phase further ensured the robustness and reliability of the system, providing a seamless user experience.

Looking ahead, several avenues for future work have been identified to further enhance the RAG system's functionality and user experience. Incorporating user feedback mechanisms and personalization features can significantly improve the system's adaptability to individual user needs. By allowing users to rate responses and provide feedback, the system can learn from interactions and tailor its outputs accordingly.

Additionally, addressing ethical considerations and bias mitigation is crucial as language models become more integrated into decision-making processes. Implementing strategies to detect and reduce biases, alongside training on diverse datasets, will ensure the system provides fair and unbiased information.

Further improvements can be made by enhancing the retrieval component to support dynamic retrieval from a broader range of sources, such as external databases and knowledge graphs, thereby improving the system's ability to provide up-to-date and relevant information.

Expanding support for various document types beyond PDF and TXT, such as DOCX and HTML, will increase the system's applicability across different domains. By addressing these areas, the RAG system can continue to evolve, delivering more accurate, relevant, and personalized responses while maintaining ethical standards and expanding its applicability across diverse contexts.

REFERENCES

[1] Google Cloud. (n.d.). *What Is Retrieval Augmented Generation (RAG)?* [online] Available at: https://cloud.google.com/use-cases/retrieval-augmented-generation?hl=en

[2] Singh, A., Pandey, N., Shirgaonkar, A., Manoj, P. and Aski, V. (2024). A Study of Optimizations for Fine-tuning Large Language Models. [online] arXiv.org. Available at: https://doi.org/10.48550/arXiv.2406.02290

[3] Hu, E. J. et al. (2021) 'LoRA: Low-Rank Adaptation of Large Language Models', arXiv [cs.CL]. Available at: http://arxiv.org/abs/2106.09685.

[4] Tonmoy, S.M.T.I., Zaman, S.M.M., Jain, V., Rani, A., Rawte, V., Chadha, A. and Das, A. (2024). *A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models*. [online] arXiv.org. Available at: https://arxiv.org/abs/2401.01313

[5] Soudani, H., Kanoulas, E. and Hasibi, F. (2024). *Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge*. [online] arXiv.org. Available at: https://doi.org/10.48550/arXiv.2403.01432

[6] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. [online] arXiv.org. Available at: https://doi.org/10.48550/arXiv.2005.11401

[7] Gupta, V. and P, S.R. (2024). Leveraging open-source models for legal language modeling and analysis: a case study on the Indian constitution. [online] arXiv.org. Available at: https://doi.org/10.48550/arXiv.2404.06751

[8] King, T., Butcher, S. and Zalewski, L. (2017). Apocrita - High Performance Computing Cluster for Queen Mary University of London. Zenodo. [online] Available at: https://doi.org/10.5281/zenodo.438045

[9] Dan Hendrycks, Collin Burns, Anya Chen, & Spencer Ball (2021). CUAD: An Expert-Annotated NLP Dataset for Legal Contract Review. arXiv preprint arXiv:2103.06268.

[10] Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. 2022. MTEB: Massive Text Embedding Benchmark. arXiv preprint arXiv:2210.07316.

[11] Liu, Y. et al. (2019) 'RoBERTa: A Robustly Optimized BERT Pretraining Approach', CoRR, abs/1907.11692. Available at: http://arxiv.org/abs/1907.11692.

[12] AI@Meta (2024) 'Llama 3 Model Card'. Available at: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[13] Reimers, N. and Gurevych, I. (11 2019) 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks', in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics. Available at: https://arxiv.org/abs/1908.10084.

[14] Devlin, J. et al. (2018) 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', CoRR, abs/1810.04805. Available at: http://arxiv.org/abs/1810.04805.

[15] Wang, Y. et al. (2024) 'Evaluating Quality of Answers for Retrieval-Augmented Generation: A Strong LLM Is All You Need', arXiv [cs.CL]. Available at: http://arxiv.org/abs/2406.18064.

[16] Rajpurkar, P. et al. (2016) 'SQuAD: 100, 000+ Questions for Machine Comprehension of Text', in EMNLP.

[17] Touvron, H. et al. (2023) 'Llama 2: Open Foundation and Fine-Tuned Chat Models', arXiv [cs.CL]. Available at: http://arxiv.org/abs/2307.09288.

[18] Vardhan, H. 2024. Fine-Tuning Llama 2 Using LoRA and QLoRA: A Comprehensive Guide. Medium. 19 June. Available at: https://medium.com/@harsh.vardhan7695/fine-tuning-llama-2-using-lora-and-qlora-a-comprehensive-guide-fd2260f0aa5f.

[19] Johnson, J., Douze, M. and Jégou, H. (2019) 'Billion-scale similarity search with GPUs', IEEE Transactions on Big Data. IEEE, 7(3), pp. 535–547.

[20] Chase, H. (2022) LangChain. Available at: https://github.com/langchain-ai/langchain.

[21] *RAG Evaluation - Hugging Face Open-Source AI Cookbook*. Available at: https://huggingface.co/learn/cookbook/en/rag_evaluation

[22] Zheng, L. et al. (2023) 'Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena', arXiv [cs.CL]. Available at: http://arxiv.org/abs/2306.05685.

[23] Team, G. (2024) 'Gemma'. Kaggle. doi: 10.34740/KAGGLE/M/3301.