Can we use Retrieval-Augmented Generation and Large Language Models to create a chatbot that enables self-management of mental health?

Thomas Lee Stoyles

2312447

A thesis submitted for the degree of Master of Science in Artificial Intelligence

Supervisor: Prof. Luca Citi

School of Computer Science and Electronic Engineering

University of Essex

August 2024

1. Abstract

This report explores the potential of using Retrieval-Augmented Generation (RAG) combined with Large Language Models (LLMs), specifically Llama models and other models from the Ollama library, to develop a chatbot capable of assisting in the self-management of mental health. The study investigates whether these models, augmented by retrieval systems, can provide specific and accurate responses to mental health queries. By analysing the performance of different models and comparing them, this report aims to determine if RAG can effectively enhance the responsiveness and accuracy of LLMs in a mental health context. The findings contribute to the growing field of AI in healthcare, highlighting both the possibilities and limitations of current AI technologies in addressing mental health challenges.

Table of Context

2. Introduction

Artificial Intelligence (AI) is rapidly evolving, with applications in healthcare ranging from rehabilitation assistance to automated surgical operations. Among these, mental health is a crucial area that requires attention due to the high prevalence of conditions like depression, Stress and anxiety, coupled with the social stigma that often prevents individuals from seeking necessary support. This report addresses the question: Can Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) be leveraged to create a chatbot that enables self-management of mental health? I believe they can.

In this thesis, we aim to develop a chatbot using the best open source LLM available to us that utilises information from text files to assist users experiencing mental distress. The report is structured as follows:

- **Literature Review**: This section identifies research gaps by reviewing existing literature on LLMs, NLP, and their applications in mental health.
- **Assumptions and Research Questions**: This part outlines the assumptions guiding the research and the key questions the study seeks to answer.
- **Methodology**: A detailed explanation of the research design, including the models used, data collection, and analysis methods.
- **Design and Implementation**: This section discusses the design of the notebook and RAG system, as well as the processes involved in text file reading and processing.
- **Results**: A presentation and analysis of the findings from the experiments conducted with all models.
- **Discussion**: An interpretation of the overall findings, exploring their implications for the future of AI in mental health.
- **Conclusion**: A summary of the study's outcomes, their significance, and suggestions for future research directions.

3. Literature review

The field of Natural Language Processing (NLP) has undergone significant transformations since its inception in the 1950s, evolving from basic rule-based systems to the sophisticated large language models (LLMs) of today [10]. Early NLP efforts relied heavily on manually crafted linguistic rules to analyse and generate text, which were often inadequate for handling the complexity and variability inherent in human language [5]. The late 20th century saw the advent of statistical NLP techniques, where probabilistic models, such as n-gram models, began to leverage large text corpora to learn linguistic patterns [10]. The real breakthrough, however, came in the early 2010s with the emergence of deep learning and neural network architectures, particularly the development of transformer models [5]. These advancements enabled LLMs to capture intricate linguistic patterns and contextual dependencies, leading to state-of-the-art performance in various NLP tasks [10]. Today, models like GPT, BERT, and XLNet represent milestones in LLM development, showcasing remarkable capabilities in natural language understanding and generation, and revolutionising applications across domains such as communication, content creation, and scientific research [11]. In short, LLM's are pre-trained on large datasets giving then pre-determined answers. This way when the user asks a question it will try and fit the question into an answer it already has been trained on. It does this by leveraging deep learning during the training phase with the main goal of generating text for the user whereas NPL, has the main goal of language analysis.

3.1 Transformer architecture

Transformers are a key part to any LLM, A transformer is a set of neural networks that are divided into an encoder and a decoder, both with a self-attention capability. [17].

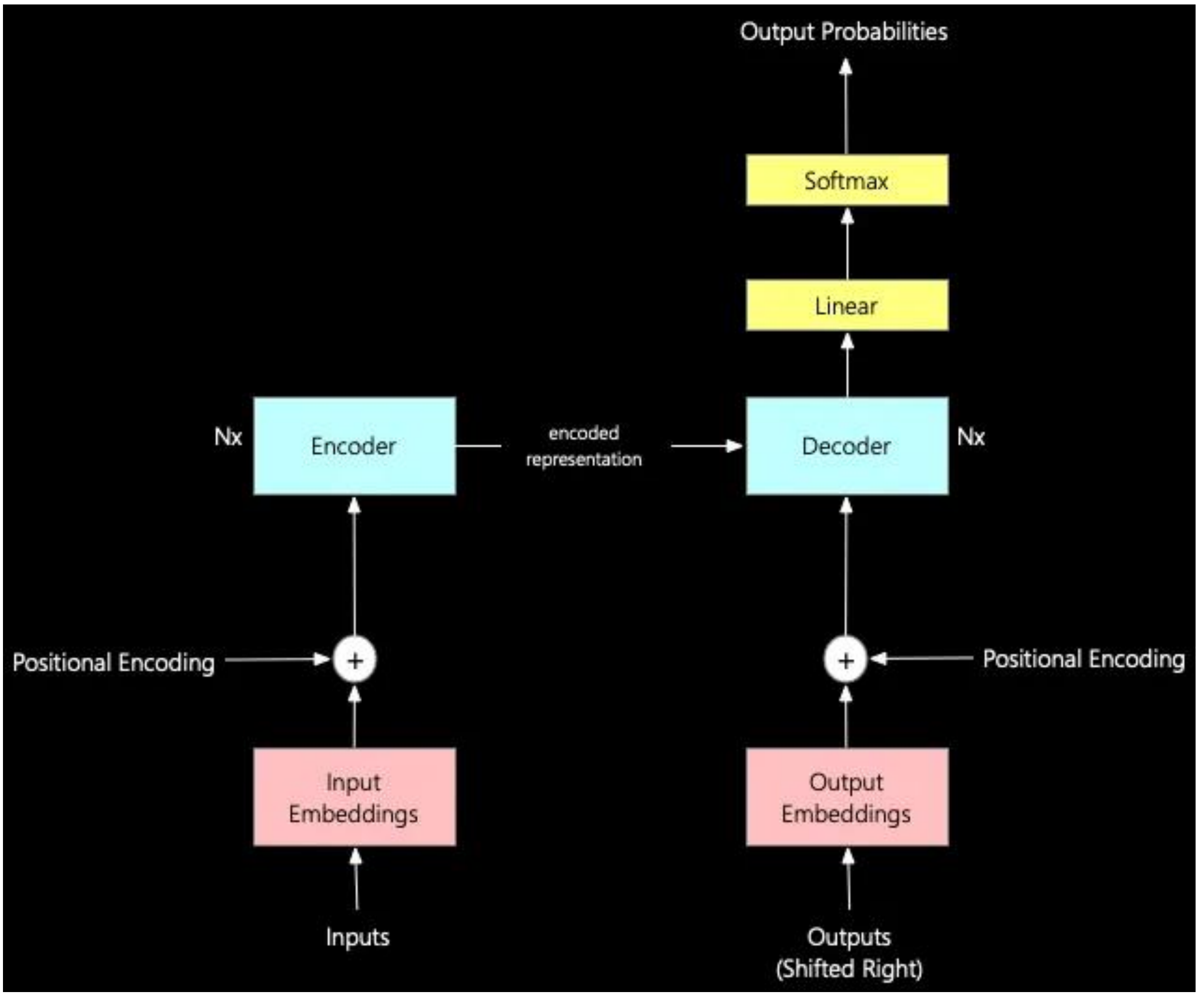

Figure 1: Gives us an insight into what a transformer architecture can look like. [17]

3.1.1 Encoders

The job of the encoder is to take the inputted data, such as a sentence, and turn this into a format that the model can better understand. To do this the encoder has different layers each with two parts.

The self-attention layer plays a crucial role in enabling Large Language Models (LLMs) to understand the relationships between words, also known as tokens, within a sequence. This mechanism is parallel to how humans comprehend meaning while reading. When reading a sentence, we consider both preceding and subsequent words to get the broader context. Similarly, the self-attention layer analyses the connections between each token in a sequence, allowing the LLM to construct a context-aware representation for each token. This representation captures both the individual word's meaning and its relative importance within the sequence structure. Allowing the LLM model to understand the questions presented to it better, subsequently leading to an accurate response.

The feed-forward layer is next, and this is where the main prediction of the model is done. A neural network is a mini brain, it mimics what our brains do every day by taking neurons and giving each response weights. If the answer is correct during training, nine out of ten times, then the weights are very strong. However, if the answer is incorrect nine out of ten times the weights need to be re-evaluated using backwards propagation, luckily, we aren't doing that here, as we will only be using decoder LLM's. This means we will not be updating the weights of any questions as the LLM will be using our text files to read the answer and will not rely on any prior training. This part of the encoder is much simpler the network here will take the data, created by the self-attention layer, and make a prediction based of this by looking at each position in the sequence independently.

Within the encoder there are some more elements that we won't get into however need to be mentioned. These are Positional encoding, which helps the model remember the order of words in sentence. Layer normalisation, which helps the model learn more effectively and Residual Connections, which help the information flow through the network better by providing shortcuts.

3.1.2 Decoders

Once the encoder has transferred the inputted data into a format the model is able to understand, the decoder starts its job of turning the data into a form that is useful for us, the users. The decoder turns the data into sentences or phrases depending on the task, which is formed from three parts which are used through several separate layers.

Similar to the encoder, the decoder starts with a self-attention layer, this layer allows the model to recognise other words in the sentence that has been provided. The only difference here is that it hides the tokens that have come before the current token that the model is looking at, not the ones that come after.

As the decoder is responsible for writing sentences to the user. It needs to make sure that the output will make sense to us, to do this the cross-attention layer is implemented. This layer allows the decoder to look back at the input sentence which has been provided by the encoder. It does this whilst generating the output for the user making responses more accurate to the user.

Same as the previous feed-forward neural network layer, this is a mini brain in the decoder that learns and predicts based on the data that has been provided to it. Expect this takes the output of both attention layers, whereas the encoder only uses the self-attention layer. This allows the decoder to make predictions on more complex patters, which is how the models view languages.
Since the LLM's that we will be using are only decoder, not encoded LLM's we will mainly be focusing on this. We are doing this because we are going to be using a RAG system for the experiments, meaning that we will not rely on the pre-trained weights of any LLM, as the answers will not come from previous training, they will come from the text files provided to the model. The weights will also not have to be updated because of this, as we will be getting all information from the text files provided. This will save us time as back propagation is a timely process and an expensive one. We will have to load the weights at least once when we ask the first question, which can lead to longer response times for that one question, however this will be considered, and a "warm-up" question will be asked prior to the experiments starting.

3.2 Tokens

Tokenization is a critical aspect when working with Large Language Models (LLMs). Tokens represent the smallest units of text that a model can process, which can be whole words, sub words, or even different characters. Tokenization breaks down text into small pieces, which allow the LLM to understand and generate human language more effectively. The number of tokens a model can process in a single input is typically capped, which directly impacts the model's efficiency and response speed. If this cap were not in place, the LLM would consume unnecessary computing power, leading to slower response times. Ensuring that text files stay within the token limit is essential for accurate and complete responses from the LLM. If the input exceeds the token limit, the model might abbreviate the input, resulting in incorrect outputs. Likewise, if the input is too short, the model may not have enough context to generate meaningful responses. Therefore, maintaining a consistent token count across tests can serve as a baseline for evaluating model performance.

3.3 Retrieval-Augmented Generation

With the use of LLM’s becoming more and more popular, people started to think about how to manipulate the answers that the LLM had been trained on. In the beginning they tried retraining models on different data, this allowed the users to choose what data the models were learning from however this came with its issues. It was very costly to retrain an entire LLM as well as timely [13]. This meant that retraining the entire model would not be worth the cost or time required. It also stopped the LLM from making sense in some instances [13], as the model would not be trained on enough tokens. Next people started to think about fine tuning the models however this posed the same issue, it can be very expensive and time consuming to finetune models. The other issue with fine tuning models is that you can risk ruining good models. For example, if you have a model that is good in law, and you retrain it to be good in housing law, something more specific. Then it may be more proficient in housing law however its English may become worse than what it was before you retrained it. Therefore, all that retraining you have just completed becomes obsolete as you can’t understand the responses from the LLM.

Retrieval-Augmented Generation (RAG) has recently been developed as a technique to enhance the capabilities of Large Language Models. It does this by supplementing the original prompts with appropriate passages or documentation retrieved through an Information Retrieval (IR) system. [12]. With the developments in RAG, we have been able to remove the issues that we originally had. We no longer must retrain models as we can use the RAG and IR system to bypass the weights of the LLM and get it to respond with up-to-date information. This saves us money and time the two main issues which we were facing previously. We also now don’t ruin good LLM’s as we are not changing the weights or data that they have been trained on, meaning there English should stay the same. With this development you can specialize some of the response that the LLM will reply with without, ruining the already good models or having to retrain them.

3.4 LLaMA-2

LLaMA-2 is an open source, large language model which can be combined with pre-trained transformers, such as GPT, to allow the study of complex text to high effectiveness. These open source LLM’s can be fine-tuned for custom specific tasks, such as financial analysis [8] or to catch online predators [7]. However, llama2 is not the only open-source large language model we also have PaLM-Bison, Falcon and Vicuna to name a few. So why choose llama2 over these other options?

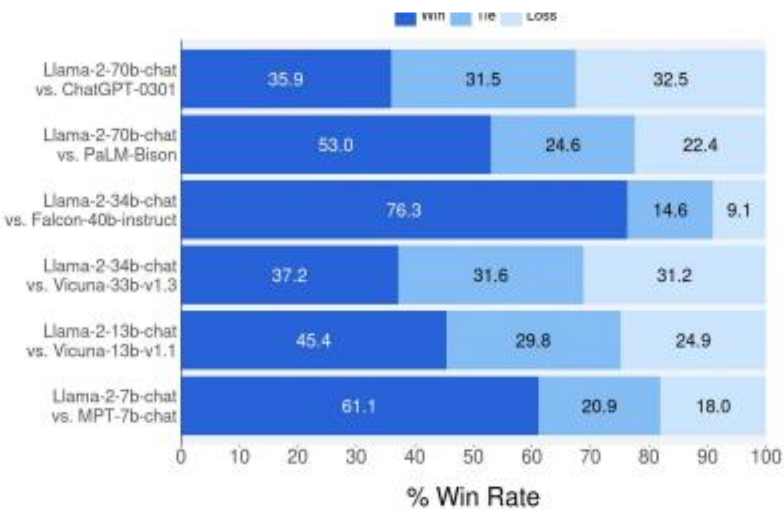

Figure 2: An experiment conducted by Meta showing the power of llama2 vs other open-source models [18]

To save us time I have taken an experiment conducted by Meta which is labelled as figure 2, which compares llama2 against other open source and closed source models. This measures each models' responses which was considered helpful. If the response was helpful then that model would "win" the test and the other would "loss". To decide if the model was helpful or not was decided by a group of human evaluators, 3 evaluators per response [18]. From this we have a good indication that llama2 is better than all other open-source models and just as good as the closed source models, when it comes to helpful responses. This means that when we are working with mental health, we can trust that the responses that we get will be some of the best responses possible.

As llama2 is readily available to the public quite a lot of work has already been completed on it such as using financial news analytics to fine tune llama2. This report shows the power of finetuning llama2 as the team managed to get llama2 to produce and summarize main points from the financial reports that they provided the model with. [9] By doing this the model followed the teams desired layout and answered the questions to a high degree of accuracy. We want to do something similar with our project but instead of financial reports it will be based on text files that contain mental health advice and support.

Within the paper, [4] LLaMA-2 has a series of large language models ranging from 7 to 70 billion parameters, available as both pretrained and fine-tuned versions. These models enhance the methodology found in LLaMA- 1 with improved data cleaning, an updated mix of pretraining data, more tokens and extended context length. Fine-tuning involves supervised training and reinforcement learning with human feedback, ensuring the models are both helpful and safe. Evaluations indicate that LLaMA-2 outperforms existing open-source models and competes well with closed-source alternatives, with a strong emphasis on safety through various testing and evaluation techniques.

In another study, [3] LLaMA-2 is adapted to Italian through a process called language adaptation. This involves fine-tuning the chosen model on a significant dataset of Italian text, this enhances its ability in understanding and generating the Italian language. This version allows LLaMA-2 to excel in a variety NLP tasks specific to the Italian language, such as text generation, sentiment analysis, and question answering. This showcases LLaMA-2's flexibility and effectiveness in handling diminishing languages, making it a valuable tool for verbal research and applications. I can take this study and be confident that LLaMA-2 will be flexible enough to complete any task that I ask of it.

LLaMA-2 models have shown considerable versatility, as seen in out last study, in various applications due to their robust architecture and training on large-scale datasets. One notable use is in the field of medical image registration, as explained in [2] The study integrates LLaMA-2 into a registration model, LLaMA-Reg, which controls the model's abilities to transform visual features into a language effective for encoding. This method utilizes adapters to link the visual and language domains. This allows for accurate multi-scale registration of medical images. The integration of LLaMA-2 enhances the model's ability to capture differences between image pairs, improving the performance of medical image registration tasks. This study proves that LLaMA-2 is already being used in the medical field to a high level of success, which I hope to replicate.

One piece of work within the medical field is the diagnosis of ophthalmic reports [1]. Within this project the team took llama2 and finetuned it to learn from large datasets of ophthalmic reports, enabling them to create "Ophtha-LLaMA2". This model can accurately predict and identify carious ophthalmic diseases and abnormalities [1]. This shows us that LLaMA-2 is a powerful tool that can help the healthcare industry integrate with AI in new and exciting ways. Now llama3 is out to public domain, we can take these sorts of projects further than we was previously able too.

3.5 Llama3

Since writing my proposal Meta have released a new open source LLM, Llama3. With little or no work completed on Llama3, it presents a unique opportunity to compare the results from our LLaMA-2 work and see if we can replicate or better these results on llama3. According to the Meta website [18] Llama 3 is a significant advancement over LLaMA-2, offering improved performance and new capabilities. Some of the key improvements included are…

- larger training dataset
- enhanced model architecture with a 128K token vocabulary
- advanced post-training techniques.

Llama3 also supports 8B and 70B parameter models, showing state-of-the-art performance in benchmarks and real-world applications like reasoning and code generation. It also includes new trust and safety tools for responsible AI use and a more efficient pretraining process.

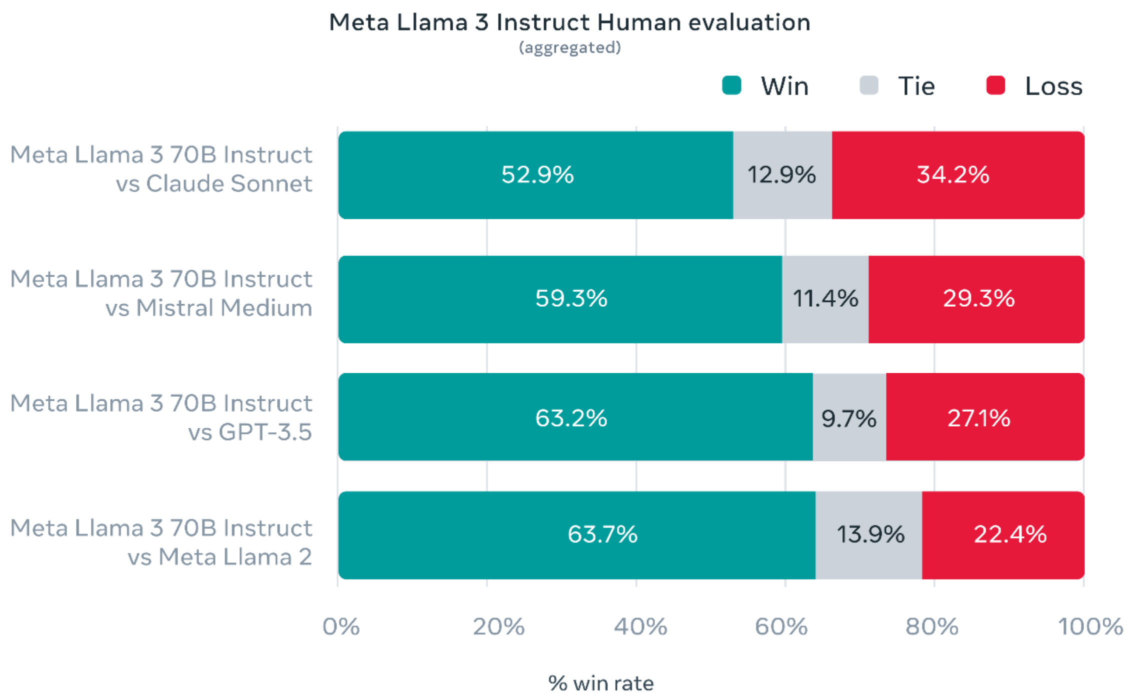

Figure 3: Shows Metas Llama3 model vs other top LLM's [18]

Meta also provided this figure (3), showing what Llama3 is like compared to other top line LLM's. As you can see from the human evaluation Llama3 came out on top the majority of the time, with its highest win rate over LLaMA-2. [18]

LLaMA-3 models are new and exciting. The primary use is for large-scale model editing, which is explored in [6] This research investigates the efficiency of sequential-batched model editing and compares it to simple batched edits. The findings suggest that sequential-batched editing, particularly with a batch size of 1024, yields optimal scaling performance. This is good to know as when I compare LLaMA-3 with LLaMA-2 in my experiment. I now have an ideal starting point for the batch amount to use. The study also establishes benchmarks and provides transparent procedures for model editing, highlighting LLaMA-3's capability to support complex, large-scale modifications while maintaining robustness and accuracy, which is what I will need.

4. Assumptions

Before conducting the experiment, I had some assumptions on what will happen during the experiment and my expected results. My first assumption for this experiment is the larger llama2 models will take longer to respond with little or no changes in the response that they provide. I believe this will happen as the larger llama2 models have more pre-trained data which means that they will have more possible words for the responses to the questions. By doing this the models will spend more time deciding on the output for a longer duration of time. I also believe that this will not make much difference in the overall response, although it may be written correctly, the smaller models can do just the same without needing to add more complex wording. Since the models are going to be using RAG as well the information provided to the user will be mostly written in the text files meaning that the model will not need to expand on the already written documents. They just need to put them into a more user-friendly context.

Another assumption I have is that the llama3 model will outperform the llama2 model in all aspects of performance. With llama3 being a new and improved version of llama2 I expect the response time and the detail in responses to be better than its counterpart, even if the information given to it is the same.

5. Research Questions

Some of the questions I hope to answer during this experiment are…

- Can we use RAG to develop a LLM that can answer questions about mental health with more specific responses?
- What is the best LLaMA-2 model?
- Are there better models than LLaMA-2 from the Ollama library?
- Can a Llama3 model using our RAG system provide a better response than the best Llama2 model edited with RAG?
- How far can we push llama3?

6. Methodology

6.1 Models

With LLaMA-2 being our choice of open source LLM, we have a plethora of models to choose from some small and requiring minimal computing power and others a lot larger, requiring a vast amount of computing power. Since the computer that we are working with has 32gb of RAM we will be testing multiple models both small and large. The models that we plan to test are…

- llama-2-7b-chat.ggmlv3.q4_K_S.bin
- llama-2-7b-chat.ggmlv3.q3_K_S.bin
- llama-2-7b-chat.ggmlv3.q5_K_S.bin
- llama-2-7b-chat.ggmlv3.q8_0.bin
  [22]

These are the models for LLaMA-2 that we will test to find the best LLaMA-2 model for our problem. The best model will then be compared to llama3 model and other Ollama models, so we can find any advancements that other models have made on LLaMA-2. These models also give us a large scope of both small and large models which should give us a variety of results, in term of quantization. With model llama-2-7b-chat.ggmlv3.q8_0.bin being the biggest and, model llama-2-7b-chat.ggmlv3.q3_K_S.bin being the smallest, meaning that it should generate responses quicker. If you look at the models, we are only using 7b models. Mainly because we won't need large models as they will take to long to respond and require more computing power.

The main difference between these models is the quantization of each model. Quantization can affect different areas of the model, such as inference speed, memory and storage, energy efficiency and model accuracy. Each model will have different quantization, with the lower quantization expected to have a drop in accuracy and requiring more post-processing but saves on energy expenditure and storage. The higher quantization models however should be more accurate at the cost of energy and storage. With this said I expect the llama-2-7b-chat.ggmlv3.q8_0.bin model to provide the most accurate responses and llama-2-7b-chat.ggmlv3.q3_K_S.bin model to provide less accurate responses. llama-2-7b-chat.ggmlv3.q5_K_S.bin and llama-2-7b-chat.ggmlv3.q4_K_S.bin to be good balanced options that are good for most problems but, do not provide the same accuracy as llama-2-7b-chat.ggmlv3.q8_0.bin.

These models' whist being pre trained will be able to read the text files that we have loaded, using RAG, and answer the questions that we ask it too, based on these text files. By doing this we can give more specific answers to any questions based around mental health which will be beneficial as it will provide more information to the users rather than what it currently provides. Since each person speaks and words things differently, I will need to test multiple different ways of explaining symptoms or feelings to make sure that the model is providing the correct information still. This will be done once the best llama2 model has been chosen based on set questions.

When using the models there is some things that I will need to make sure that the model does correctly. This is…

- Using correct English which is relevant to the documentation provided.
- Says only the information provided to the model by the documentation.
- Can pull information from more than one document.
- By ensuring these bullet points are employed we can ensure the validity of the experiment and the information that we are providing.

Once I have established the best Q for llama2 I plan to then test this against other open-source models which are within the Ollama library. Using LangChain alongside Ollama libraries I believe that we can employ the same RAG system that I have used for the llama2 testing to, test other models within the Ollama library. The chosen models from the Ollama library are…

- Llama2(Baseline found in experiment 1)
- Llama3
- Solar
- Phi3
- Mistral

These models were selected as they are a perfect blend of small and large models, which have been trained in different ways. This will allow me to test if llama2 is the best open-source model or, if it excels in one particular area. Ollama also has llama3 as a model option which is the upgrade to llama2 which means I can compare both Meta models whilst comparing them to all the other open-source large language models.

6.2 Mental health text files

To create the text files that will be used instead, of the pre-trained data that the LLM is currently supplying, I have used trusted sources like the NHS website [14, 15, 16, 21, 23] and MindUK [22], alongside other websites or papers that maybe be useful and provide good, correct information [19, 20]. To make sure that the number of words in each file is matched with the number of tokens that the model can handle, I will be creating multiple text files for each mental illness, allowing us to spread the information evenly. This will also help me test if the model could respond to questions by taking information from more than one place and, to what extent it will take the information if it suggests similar information. I will be checking to make sure that the information that is provided to the LLM is up-to-date and relevant, so the model does not give any incorrect advice. I also want to test if the model can respond to a user looking out for another person and how well the model would respond to these types of questions.

6.3 Experiment

The experiment can be split into two different parts. The first part will be creating a notebook that can take inputs from text files and display these instead of displaying the pre trained information on mental health, using RAG. These will be text files that I have written from the research conducted above. Once the notebook has been written I will test the different llama2 models, which have been stated above. These models will provide me with responses so we can see how accurate the responses are at reading the information provided to them as well as, how quickly they respond to each question that is asked. Each model will have the same text file and will be asked the same question. This is to make sure that the experiment is fair and controlled.

Part 2 will begin once the best llama2 model has been discovered. We will take the notebook and adapt it for the Ollama library models. As llama3 has only recently released we are limited to the models that we have access to in that regards however, llama3 models should be more advanced and have more power than the llama2 models. [18] Even the finely tuned ones. Consequently, I will test the difference between llama2 and Ollama models to see if any improvements can be made on my “best” llama2 model. The Ollama models will be given the same text to allow for a fair testing environment. Once done I will take the best Ollama model, if I can improve it or break it to find any flaws within the system.

The questions that I plan to ask during experiment 2 will be separate to each illness. One aimed at depression, one at stress and one at anxiety. I will also add more questions which, will relate to multiple illnesses. I plan to do this to see if the models can still generate accurate responses with similar information applying to both text files. This is needed as mental health is not as straight forward as a sickness. There is no defining symptom for depression, stress, or anxiety. They all share symptoms which is why I feel that it will be imperative that the LLM is able to distinguish between all three and make sure it is not providing inaccurate results when faced with these types of questions.

Another thing to note is that all experiments will be judged by me, therefore I will know what models are answering the question which could lead to some bias. However, since I am reporting on response time there is no way for me to influence how quickly the models take to respond I can only report and comment on what is shown to me. Regarding the accuracy of the model this could be influenced by me as it is my opinion on the validity of the response. To ensure that there was no bias in the accuracy of the model I split this into two categories, “Good” and “Bad”, which was later expanded into “Good”, “Bad” and “Don’t not follow document” in experiment two. In experiment one only “Good” and “Bad” was used, as there was no instance of “Don’t not follow document” but what do this mean? If a model has a good response, it means that the documents are perfectly, the wording of the response allied with the document information, and nothing was added or changed from the document that was provided to it. The “bad” response means it did follow the documents, but the response was poorly put together and was hard to understand. Lastly “Did not follow documents” means exactly what it says, the model responded to the question however the response did not follow the documents, and the model used its pre-training to answer.

6.4 Data collection

The data in this experiment will be the information provided in the text files that we will use to answer the questions that are asked. For the experiment I will be using the NHS website and the Mind charity website and compare the information provided by both. Then I will take this information and combine it to create our text files as well as, any other website that has up-to-date information on these three illnesses.

6.5 Design research

6.5.1 Design of the Notebook

The notebook will be carefully designed to demonstrate the implementation of a Retrieval-Augmented Generation (RAG) system, which will synergize the ability of retrieval-based and generation-based models, which have been stated above. The idea of the notebook will be to enhance the accuracy and relevance of each generated response. This can be achieved by using essential libraries and modules, such as langchain and sentence-transformers, which will be vital for handling data preprocessing, natural language processing, and model integration. The process will begin with reading and preprocessing any text file that I create. This will be used to create a document store. This will allow me to load and encode the documents into embeddings using a pre-trained model from the sentence-transformers library. Subsequently, these embeddings will be indexed using FAISS (Facebook AI Similarity Search), which will allow effective document retrieval. The core components of the notebooks RAG system will include, configuring a language model and setting up a retriever that fetches the most relevant documents based on the query(question) presented by the user. A PromptTemplate will be used to format the query and retrieve the appropriate context before passing it to the LLM.

6.5.2 Design of RAG System

The RAG system in the notebook will be a key part. I plan to create a sophisticated approach to improving response quality through the integration of retrieval, augmentation and generation. The planned process will start with retrieval, the conversion of documents into embeddings via a pre-trained model from the sentence-transformers library, specifically the all-MiniLM-L6-v2 model. FAISS (Facebook AI Similarity Search) will be used to index these embeddings, allowing for rapid and efficient similarity search. The retriever component utilises the FAISS index to identify the top documents that are most like a given query embedding, with x=1 indicating the retrieval of the single most relevant document. I assume if I want to use more than one document I can increase x to the number of documents, however this will be tested during the creation. Subsequently, Augmentation is close to the retrieval stage where the documents are passed to the model rather than the model using its pre-trained knowledge. A PromptTemplate is employed to structure the query and the retrieved document into a prompt. This structured prompt is then fed into the language model, which will generate a response. Generation will then also be complete using the retrieved documents as context. This most likely will happen in a separate function which is called upon or a chain. The integration of these mechanisms should ensure that the generated responses are not only relevant but also grounded to the retrieved documents, enhancing their accuracy and reliability.

6.5.3 Text File Reading and Processing

In the notebook, the reading and processing of text files are fundamental. To ensure this I plan to install a robust and searchable document store. Firstly, the text files will be read from a specified directory using Python's file handling capabilities, and the content of each file is put into a data structure, most likely a list. Each document's content is then transformed into embeddings using a pre-trained transformer model from the sentence-transformers library. This conversion will translate the textual data into numerical representations. These embeddings are subsequently indexed using FAISS, a library which is known for its efficiency in handling large-scale similarity searches. FAISS enables the quick and precise retrieval of documents that are most relevant to a given query.

6.6 Jira

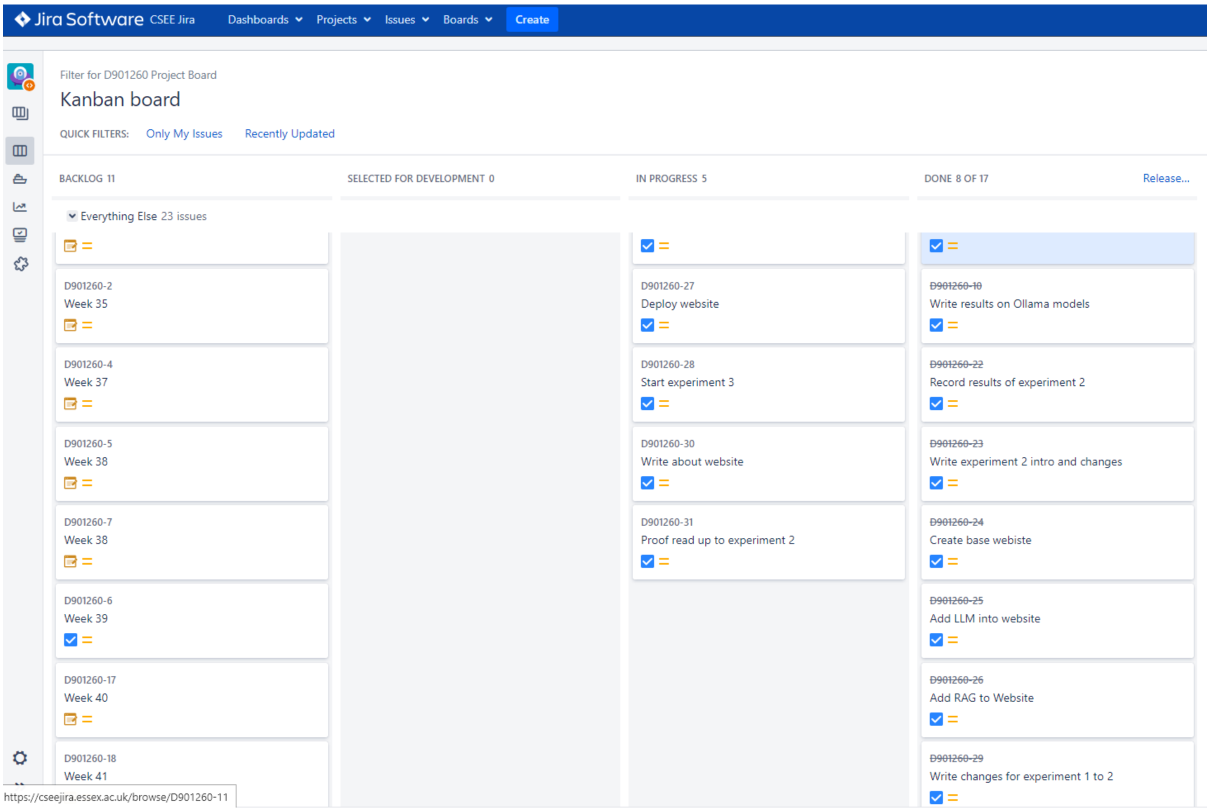

Figure 4: The Jira board used throughout the project

I have a Jira board where I plan to update with everything that is completed each week, this way I know what is next to be done and when jobs will be completed and in which sprint. This also gives me a list of everything that needs to be completed and when by so I can see how much work I have left to do. It also will allow me to break each task into sprints, if I want to use Agile as my design methodology. Each sprint will be 2 weeks long and I can add anything I wish to complete in that sprint onto the board.

7. Results

7.1 Experiment 1

To begin the experiment, I started by created the notebook that could run the LLaMA-2 model and the RAG system simultaneously. I used different python libraries to help aid this, these are LangChain, sentence-transformers, FAISS-CPU and CTransfromers. Firstly, I prepared the documents and stored them in a local database so that the LLM model could easily reach them when called for. To do this I defined the location of the text files that I wanted to store, then used MiniLM, which is found in the sentence-transformer package to interpret information from the text files proved. Lastly, I stored the documents into a local database using FAISS (Facebook AI Similarity Search). The database is a vector database which has been optimized for looking across large and high dimensional datasets.

After this I created an AI that was aware of the local files content and could show the LLM these instead of the LLM using its pre-training. To do this I started by creating a template variable so that the model knows what the print of the answer should look like. For this template I took inspiration from others that have created templates for similar tasks, I noticed that the format is mostly context, question asked, followed by answer to the question, which is also what I decided to follow. Then I loaded the LLM model that I planned to use; this is where I would change the models that I was testing. To do this I used CTransformers with the variables of the models .bin file, model type which was llama and the max tokens that the model could use which started at 256.

Once this was working, I used MiniLM again to load the information which was previously interpreted and recalled the FAISS database to retrieve the already pre-loaded model. Lastly before asking the question, I had to prepare a version of the LLM model which was loaded previously and load it with the local content which was stored in the FAISS database. To do this I used a retriever to get the local text files and the turn them into local content for the LLM model to read and select what best answers the question asked, which was then displayed following the template which we previously stated. Once this was debugged the file was able to answer questions on the testing information that was in the text files.

7.1.1 Issues

After this I started to load the text files with information on depression only. I did depression only as for this part of the experiment I was testing to find the best Q version of the selected LLaMA-2 models.

Whilst doing this I came across my first major issue within the project. Tokens. Tokens, as previously said represent the smallest units of text that a model can process because of this any text file that exceeded a token count of 256 would break the script and not allow the model to answer the question that had been asked. To fix this I knew that I had to find out how many tokens was in each file, so I could figure out if the files was too big or if it was something more. To do this I created a function that went into the text files folder and counted each documents' tokens and displayed this to the console. By doing this I could see that the tokens in the documents was too large so I made multiple documents which was all below 256 however, this did not fix the issue.

So my next step was to experiment with the max_tokens value which I set to 2048 which I knew was more than enough to cover the text files token count individually and combined however this also didn’t fix the issue. So, I looked online to see if anyone else had had this issue and they did. It turns out that LLaMA-2 models are capped at 512 tokens which I was unaware of. I then changed the max_tokens in each file to be under 512 tokens and changed the max_token variable to it as well. This led to some interesting results, making some questions work and other not. After doing research I discovered that setting the max_token variable to 2048 doesn’t set the token count for llama2 to 2048 it is still hard capped at 512. So, to change this I had to change the context_length variable which changes the amount of context, in this case tokens, the model can read. Once I changed this to 2048 the models worked perfectly with the files having more tokens that 512.

```python
# Set the path to where you have the text files
directory_path = r"Z:\Uni-CE901\22-24_CE901-CE911-CF981-SU_stoyles_thomas\Data"  # Change this to your directory

# Initialize the tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2") # Change if you wish to test other tokenizers

#counting tokens in files and directory
def count_tokens_in_file(file_path, tokenizer):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()
    tokens = tokenizer.encode(text)
    return len(tokens)

def check_tokens_in_directory(directory_path, tokenizer, max_tokens=2048):
    for file_name in os.listdir(directory_path):
        if file_name.endswith('.txt'):
            file_path = os.path.join(directory_path, file_name)
            num_tokens = count_tokens_in_file(file_path, tokenizer)
            print(f"File: {file_name}, Tokens: {num_tokens}")
            if num_tokens > max_tokens:
                print(f"WARNING: {file_name} exceeds the maximum token limit of {max_tokens}")

# Check token counts in the directory
check_tokens_in_directory(directory_path, tokenizer, max_tokens=512)  # Adjust max_tokens to your max token count
```
Python

Figure 5: Displays code used to debug token issue

```python
def truncate_tokens(text, max_tokens, tokenizer):
    tokens = tokenizer.encode(text, truncation=True, max_length=max_tokens)
    return tokenizer.decode(tokens[:max_tokens])

# Ensure context length does not exceed model limit
max_context_length = 2048  # change if you change token values
```

Figure 6: Displays code used to solve token issues

7.1.2 Results of experiment 1

Next, I moved onto testing each of the models to do this I wrote 4 questions all about Depression this is because, at the time, I only had the text files of depression created. Therefore, the questions I asked was only on Depression with was across 4 different text files. This allowed me to test 2 things. The response generated from the LLaMa-2 models and if the model will read more than one text file. On the graphs that display the results below, the questions are as followed, with the side values being time in seconds:

- Question 1: How to treat Depression?
- Question 2: When should I see a doctor about Depression?
- Question 3: How to tell if you have Depression?
- Question 4: What can I do if I’m living with Depression?

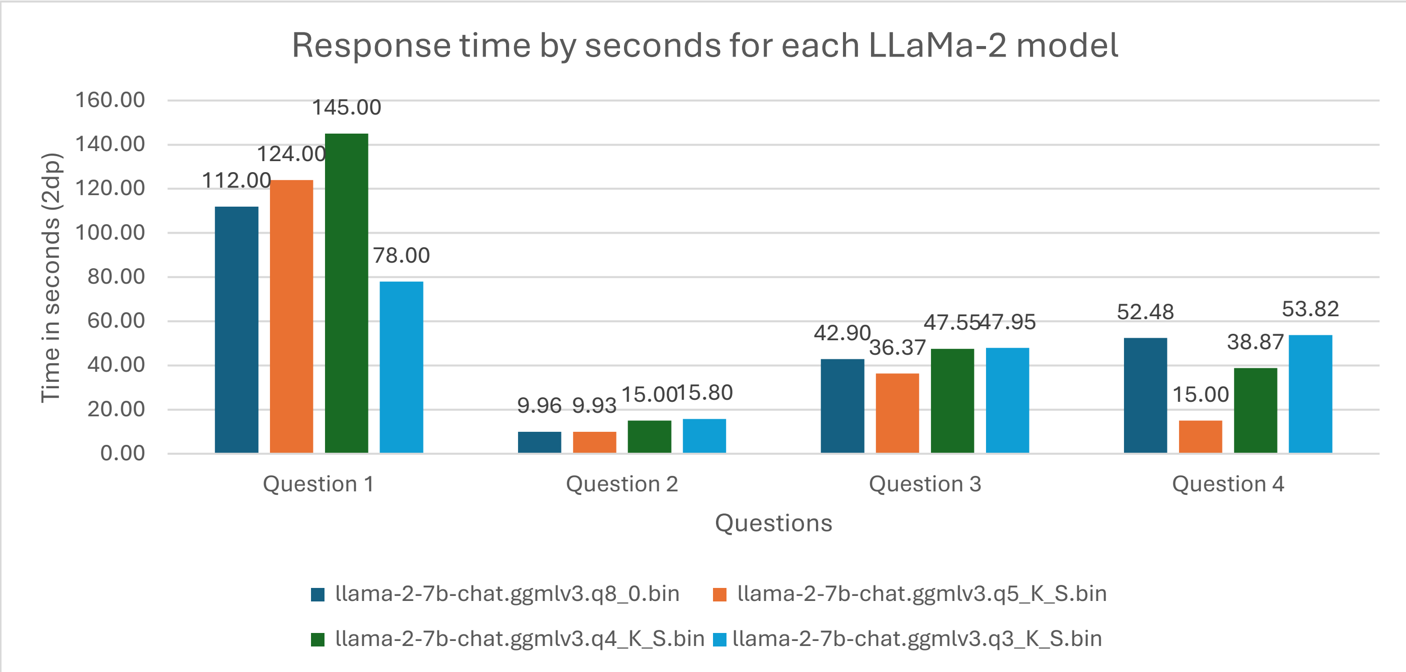

Figure 7: Shows results of response time of experiment one, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM

Figure 7 shows our results for each LLaMa-2 model's response time when asked one of the 4 questions. Question 1 had some very high response times which I originally thought was due to it being the first question, and the model not reading all the text files beforehand and not having the weights loaded. To clarify this, I made sure that during all experiments there was a "warm up question" of *"What is Depression?"* asked. This question was not asked again during any of the experiments to stop any advantages of seeing the question once before. By doing this I ensured that the model had loaded any weights needed to answer the questions. However, after testing the models' multiple tests it turns out that the models did take this long to respond to the question *"How to treat Depression?"* with the longest being llama-2-7b-chat.ggmlv3.q4_K_S.bin taking 145 seconds to generate a response which is over two minutes. This was not good enough especially for a potential chat bot, where people expect quick responses. Other noticeable results: Question 2 *"When should I see a doctor about Depression?"* had the quickest response time being below 10 seconds which was the minimum I considered a success for the chatbot. Hopefully with changes to the notebook I can get this even lower. Question 3 and 4 had similar results being under a minute however far to long for a chatbot response. One noticeable thing is the model llama-2-7b-chat.ggmlv3.q5_K_S.bin responded to question 4 much quicker than all the other models, with a response time of 15 seconds, that is over half the time of the next quickest, being 38.87 seconds. This could mean that the llama-2-7b-chat.ggmlv3.q5_K_S.bin model could be the best model, however before I make that assumption I wanted to check the accuracy of each model as well. to make sure the information they are providing is correct.

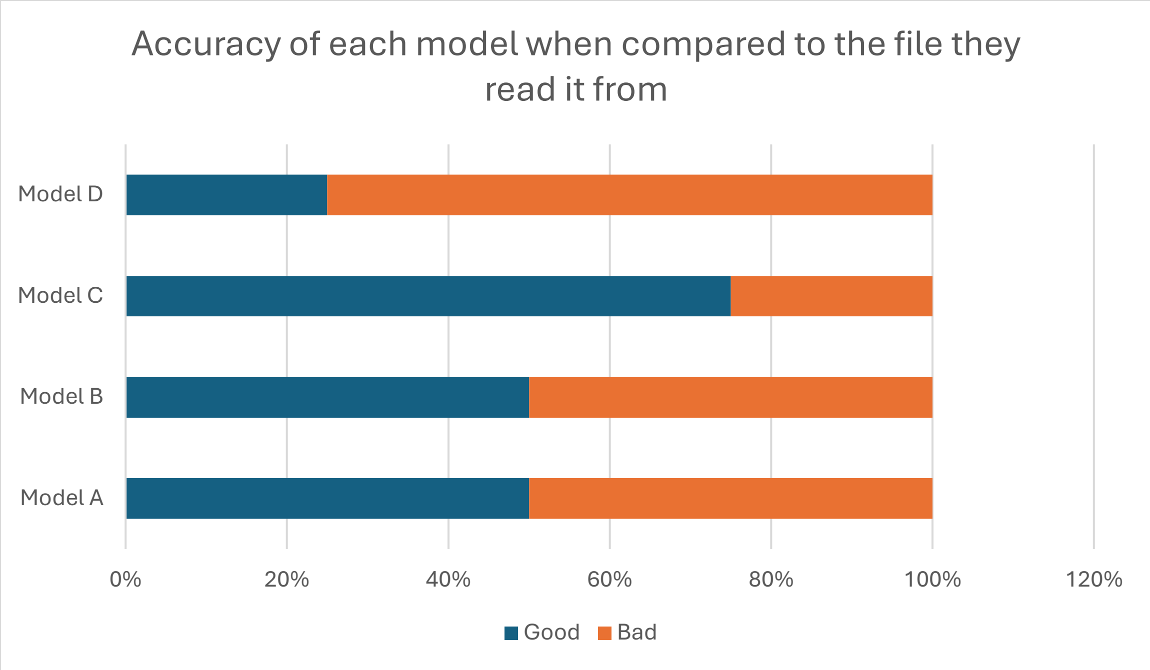

Figure 8: Shows results of the response accuracy of experiment one, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM

When I started to investigate the accuracy of each model my assumption was that if each model is taking this long to respond then they must be 100% accurate. When looking at the data no model achieved 100% accuracy which is disappointing considering some questions took over 2 minutes to answer. Seeing this I knew that something was wrong with the way the RAG system was being set up and the way text files were being saved. This is when I started to fine tune the model in hopes of getting a quicker response time and more accurate responses.

7.1.3 Parameter tuning

To begin tuning I started to investigate new way to help the model read the information quicker an get a quicker response time. Originally, I planned to investigate a new form of database to store the text files. I came across Pinecone another vector-based database which can be used in python. Pinecone is described as “Lighting Fast” and “Exceptional scalability”]. The scalability side does not interest me as the experiment is not going to be bigger that the files that I give it. The speed of Pinecone was interesting however this could solve my response time issues. After doing more research I was able to implement a Pinecone database into my script however it did not fix my response times and was more complicated that the FAISS database I had previously installed therefore I decided against continuing with Pinecone and retuned to FAISS.

After the failure with the databases I tried to investigate how the text files were be read into the database and if there was anything that could be changed within the script.

```python
# Reading txt files

# define what documents to load
class UTF8TextLoader(TextLoader):
    def __init__(self, file_path: str, encoding: str = 'utf-8'):
        super().__init__(file_path, encoding)

loader = DirectoryLoader("./", glob="*.txt", loader_cls=UTF8TextLoader)

# interpret information in the documents
documents = loader.load()
splitter = RecursiveCharacterTextSplitter()
texts = splitter.split_documents(documents)
embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'})

# create and save the local database
db = FAISS.from_documents(texts, embeddings)
db.save_local("faiss")
```

Figure 9: Displays script before any editing

While looking through the script, shown in figure 9, I thought about changing the UTF text loader however after some quick research this was not going to be the best option and, I already investigated changing the database from FAISS which was also not a viable option. This led me to looking into the splitter and texts variable. I noticed that I had uses a recursive character text splitter which searches for characters which are repeated but, this didn’t make sense since the information in each text file was not repeated it was only stated once. So, I decided to change this to just character text splitter which instead splits the documents by characters. To do this I needed to add a chunk length to the document, the chunk length is the number of characters in each chunk. This makes it easier for the model to find specific parts that it needs to answer any questions, as they are in chunks rather than a full document. I also added an overlap feature however that was set to 0 as I did not want any duplicate chunks of data to affect the model’s decision making.

```python
# define what documents to load
class UTF8TextLoader(TextLoader):
    def __init__(self, file_path: str, encoding: str = 'utf-8'):
        super().__init__(file_path, encoding)

loader = DirectoryLoader("./", glob="*.txt", loader_cls=UTF8TextLoader)

# interpret information in the documents
documents = loader.load()

splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
texts = splitter.split_documents(documents)

embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'})

# create and save the local database
db = FAISS.from_documents(texts, embeddings)
db.save_local("faiss")
```

Figure 10: The script after parameter tuning the splitter variable for better response time

After this was completed, I decided to test the models again using the same questions and text files to see if there was any difference in response time or the accuracy of the model.

7.1.4 Post Parameter Tuning Results

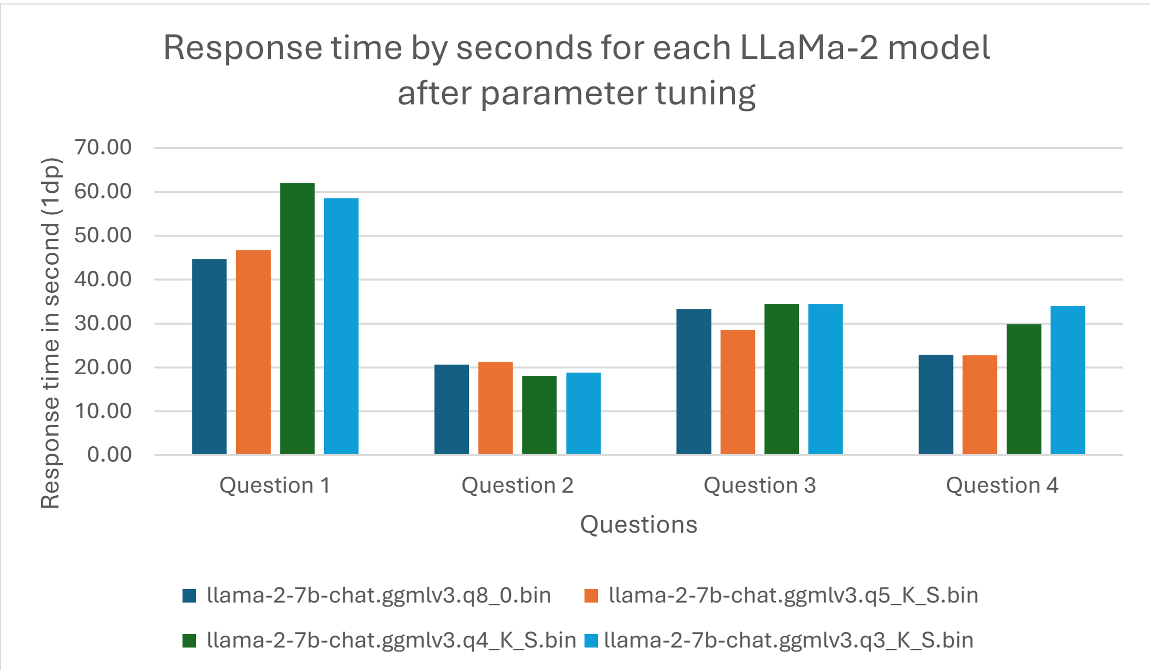

Figure 11: Shows the results of the same models with the improvements made, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM

As you can see in figure 11 above, by changing the splitter variable to a CharacterTextSplitter with a chunk size of 1000 and an overlap of 0, it has drastically improved the response time of each model, apart from question 2 *"When should I see a doctor about Depression?"*. Nevertheless, Question 1 has gone from over a minute and for some models over 2 minutes to, under a minute for all models is a significant improvement to the response time with over a 50% decrease across 3 models. Questions 3 and 4 also see a decrease in response time although not as big of an improvement as question 1, still an improvement. One thing I noticed was Question 2 previously being under 10 seconds for some models however when the changes have been implemented the response time has increased rather than decreasing. This was a surprise considering the other 3 questions decreased in response time, the reason for this is still unknown however I did test the changing the information to other text files and it still did not decrease the response time. Another noticeable change was the model llama-2-7b-chat.ggmlv3.q5_K_S.bin got worse on question 4 which again was unexpected as it originally was the best performing model out of all the ones I was testing. Overall, I feel this is a great improvement however, it is not where I would like it. With most chatbots able to respond in seconds that should be my goal.

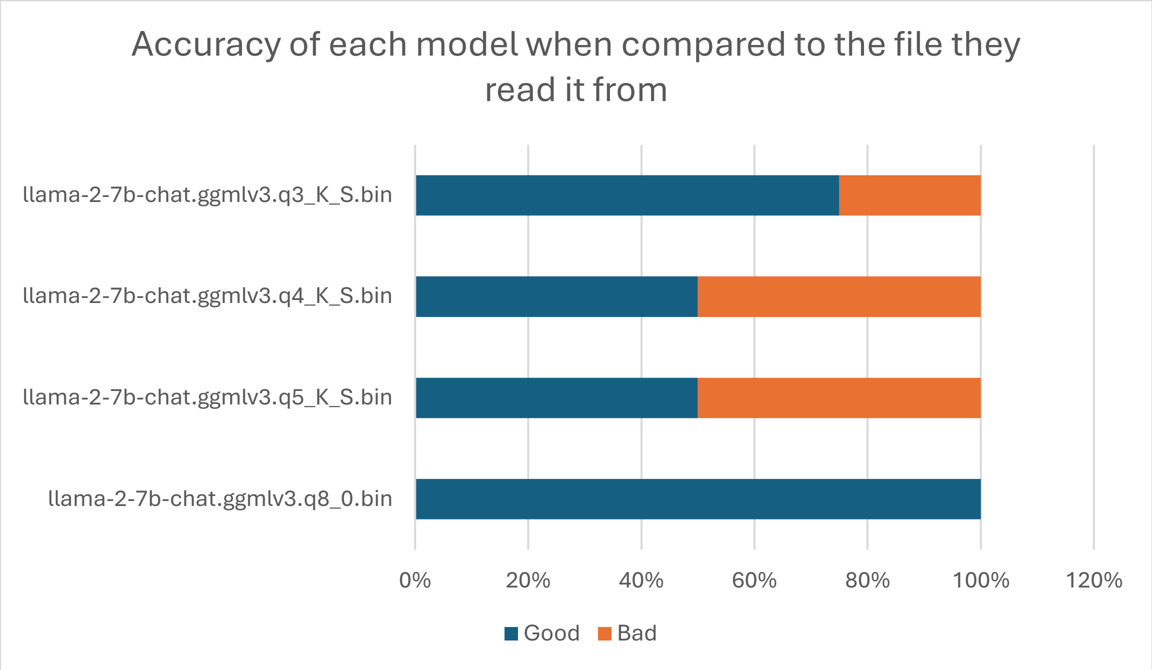

Figure 12: Shows the accuracy of each model after the changes, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM

We also looked at the accuracy of each model with the changes implemented and although the changes made don't improve some models such as llama-2-7b-chat.ggmlv3.q5_K_S.bin and llama-2-7b-chat.ggmlv3.q4_K_S.bin. It improved llama-2-7b-chat.ggmlv3.q8_0.bin and llama-2-7b-chat.ggmlv3.q5_K_S.bin to have better responses and even gave llama-2-7b-chat.ggmlv3.q8_0.bin, a perfect response score, which is exactly what we wanted to achieve from the changes that we made.

7.1.5 Best LLaMa-2 model

From this experiment we can now say what model will be our baseline. The model that we are going to compare to all other models too. llama-2-7b-chat.ggmlv3.q8_0.bin was the best model out of the 4 that has been tested. With a 100% accuracy to the text files and good response times across all questions, once the changes were made. We will be using this as our baseline model and seeing if we are able to find other models which are better than it.

7.2 Experiment 2

After finding my baseline from the LLaMA-2 experiment, I moved onto editing the notebook so that I could compare my baseline to other LLM's. This will allow me to see which LLM will be best suited for the chatbot. With the new models there are some things that I would like the new models to improve upon. These are the response time and the way that the questions are answered. Ideally, I want the response time to be instant which I would consider to be under 5 seconds for each response. I would also like the new models to respond with the correct information from each text file. Since for this experiment I have increased the amount of text files and added more questions for each model to answer, I will also be running the baseline model again so we have data on how it will answer the new questions added and how it will handle the new text files that have been added.

The models that I plan to test are from the Ollama package, these models are all found here [44] in the site it explains what the features of each model are. The following models are the ones that I am testing…

- Mistral
- Solar
- Phi3
- Llama3
- Baseline (Llama2 model)

These models will give me a wide range of big and small models, which I am expecting to affect the performance of each model in response time and answer quality.

As I am using different models which require different packages, I had to change the notebook to incorporate these new models. The main change that was done was how I called each model. The old way was too complicated since I downloaded each LLama2 model's bin file and ran that to generate the response. To make this an easier approach I installed the Ollama package and used this instead to download and run each model. This meant that I did not need each model's raw bin file and instead could pull them from the Ollama library. Other than this I didn't change anything as I wanted the experiment to be staged in the same way that the baseline experiment was, by doing this it allows for the results to not be affected by different factors such as, the way that the RAG system had been set up or, the way the text files are being read. This means that the results we have will be fair and equal and we can compare them without discrepancies. For this experiment I plan to first test the 4 models listed above to find the best model out of them then use this model to compare against the baseline model to see any differences between the two.

To have a comparison between experiment 1 and 2 I used the same 4 questions as experiment 1 however I also added questions about Anxiety and Stress to see how the models will handle more text files and more mental illnesses. The questions used in the experiment are…

- Question 1: How to treat Depression?
- Question 2: When should I see a doctor about Depression?
- Question 3: How to tell if you have Depression?
- Question 4: What can I do if I'm living with Depression?
- Question 5: How can I tell if I have anxiety?

- Question 6: How can I tell if I'm stressed?
- Question 7: How can I treat my stress?
- Question 8: What is Anxiety?

These questions cover the old questions which we used during experiment 1 and some new questions to allow us to test more text files and to see if this will change the way that the model responds and how quickly.

7.2.1 Results of experiment 2

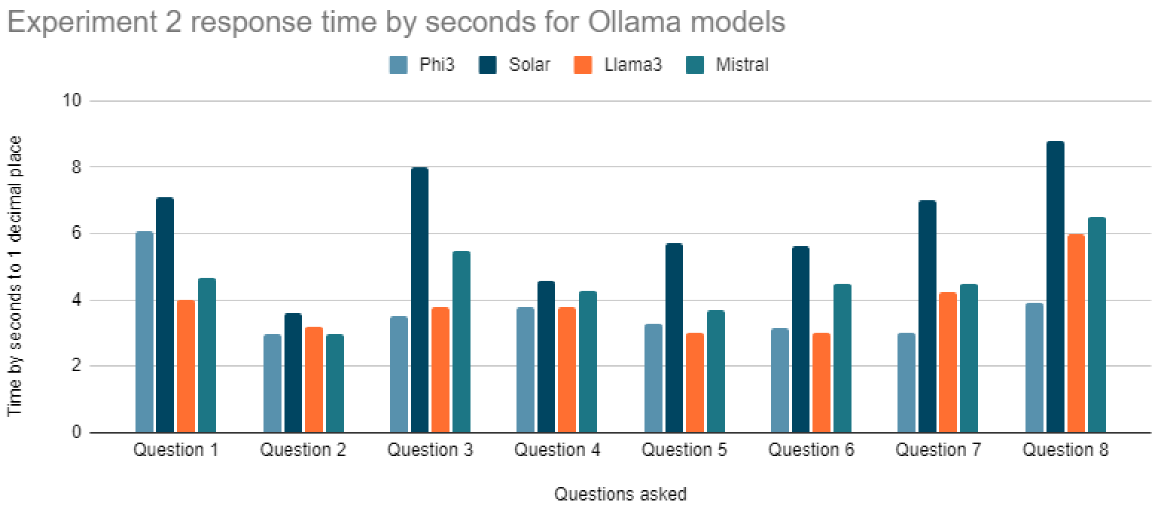

Figure 13: Shows results of response time in experiment 2, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM.

The results from experiment 2 were not what I expected, firstly let's look at the response times. As you can see from figure 13 all the responses are below 10 seconds which is completely unexpected, this is the baseline that I had set out in the beginning as a minimum requirement response time and some models are consistently below 5 seconds which is astonishing and an ideal response time. Now let's look at the models, from the results we can see some clear conclusions. Solar is the worst model out of all 4 in terms of response time, even though it constantly is below 10 seconds, which is better than our baseline. It has the highest response time out of all 4 models across all 8 questions. Next was Mistral, significantly better than Solar however consistently worse than both Llama3 and Phi3, except for Questions 1 and 2 where it was second quickest. Now Llama3 and Phi3 clearly are the best at response time, both constantly first and second quickest with the exceptions mentioned above. Both models are under 5 seconds for all but one question each. (Question 1 for Phi3 and Question 8 for Llama3).

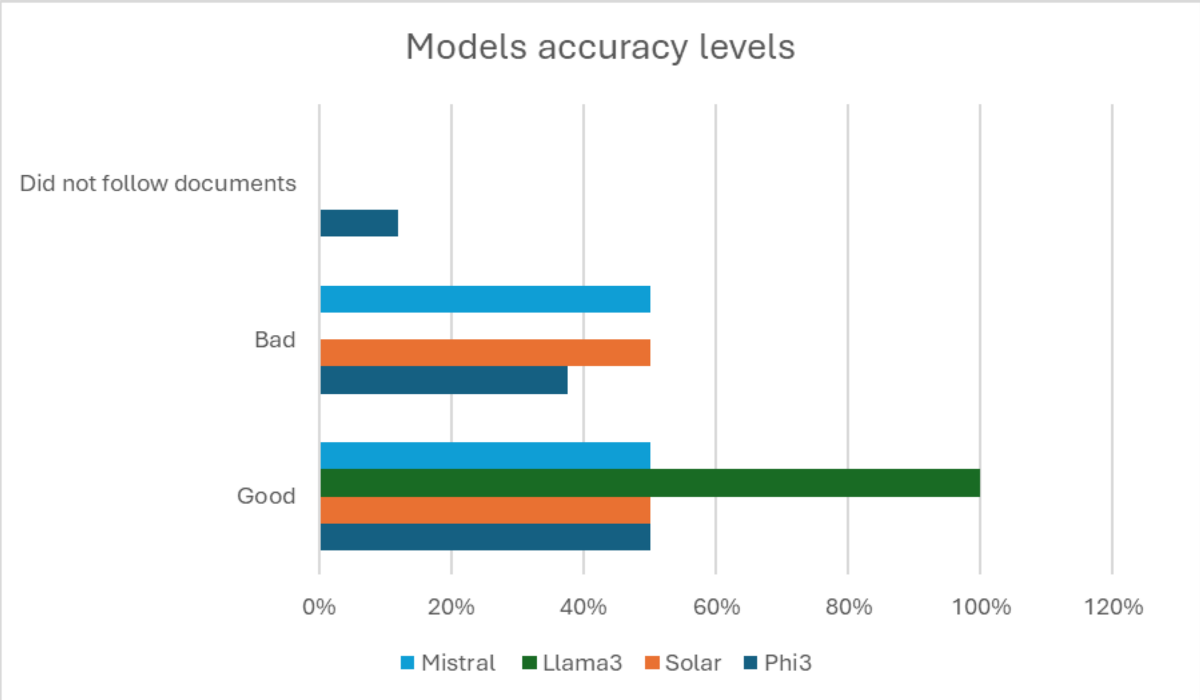

Figure 14: Shows models accuracy levels of experiment 2, conducted on a desktop pc with a 3.7GHz, 6 core CPU and 32gb of RAM

Before looking into the accuracy of each model I will explain what “Good”, “Bad” and “Did not follow documents” mean as they are a bit vague. If a model has a good response, it means that the documents are perfectly, the wording of the response allied with the document information, and nothing was added or changed from the document that was provided to it. The “bad” response means it did follow the documents, but the response was poorly put together and was hard to understand. Lastly “Did not follow documents” means exactly what it says, the model responded to the question however the response did not follow the documents, and the model used its pre-training to answer.

Now if we investigate the model’s accuracy numbers, we have a clear best model with that being Llama3 which has a 100% accuracy on responses. The responses from llama3 all followed the documents correctly and in some cases word for word. Solar and Mistral have very similar results, both with a 50/50 split on good and bad responses. Phi3 was the surprise with quick responses however was unable to match Llama3’s accuracy. Phi3 had the worst accuracy with a 50% good, matching Solar and Mistral however it is the only model that had a “Did not follow documents" response which is alarming and cannot be happening when dealing with people that may have a mental illness, as we cannot give them incorrect information.

7.3 Comparison

With the experiment being concluded I think that LLama3 was the best model to come from experiment 2, with 100% accuracy and a response time below 5 seconds for all but one question. If we compare this to our LLaMA-2 baseline model, it is evident that Llama3 is a better model in comparison to our baseline. If we compare them side by side both models have an accuracy of 100% which is perfect and what we want however, if we look at the response time our baseline has an average of 30.25 second estimated and Llama3 has an average response time of 4 seconds estimated. That is a decrease of 86% estimated which is a huge difference in the time it takes the model to respond. Therefore, I believe that Llama3 will be the best model for a chatbot that utilises RAG systems.

7.4 Experiment 3

Now I have found the best model. I want to test its limitations and see if I can fix any limitations that I find within. To do this I plan to take the Llama3 model and see if I can break it, understand how I have broken it and why the model acts the way it does, in the broken state. Then see if I can find a solution to the broken state and potentially unlock more possibilities within the model.

One of the first things I wanted to test was cross documentation reading. By this I mean what if more than 2 documents have the same information? Which document should the model follow? Can the model read more than one document and provide a response which combines both documents? These are some of the questions I planned to answer and if the model doesn't work then attempt to find a fix. To do this I edited the notebook to display all the source documents that it was reading rather than one, which is what it was doing previously. I then started asking the model questions that would require more than one document to answer. These questions asked the model for comparisons between illnesses, such as *"What symptoms do Anxiety and Stress share?"* By adding these types of questions, I was able to test if the model successfully read and retrieved all the required information from the text files and did not revert to its previous training as shown in figure 15.

```
Source Document 1:

Symptoms of anxiety
Anxiety can cause many different symptoms. It might affect how you feel physically, mentally and how you behave.
It's not always easy to recognise when anxiety is the reason you're feeling or acting differently.
Physical symptoms
faster, irregular or more noticeable heartbeat
feeling lightheaded and dizzy
headaches
chest pains
loss of appetite
sweating
breathlessness
feeling hot
shaking
Mental symptoms
feeling tense or nervous
being unable to relax
worrying about the past or future
feeling tearful
not being able to sleep
difficulty concentrating
fear of the worst happening
intrusive traumatic memories
obsessive thoughts
Changes in behaviour
not being able to enjoy your leisure time
difficulty looking after yourself
struggling to form or maintain relationships
worried about trying new things
avoiding places and situations that create anxiety
compulsive behaviour, such as constantly checking things

Causes of anxiety, fear and panic
There are many different causes of anxiety, fear or panic and it's different for everyone.
When you're feeling anxious or scared, your body releases stress hormones, such as adrenaline and cortisol.
This can be helpful in some situations, but it might also cause physical symptoms such as an increased heart rate ar
Regular anxiety, fear or panic can also be the main symptom of several health conditions. Do not self-diagnose - spe
Identifying the cause
If you know what's causing anxiety, fear or panic, it might be easier to find ways to manage it.
Some examples of possible causes include:
work - feeling pressure at work, unemployment or retirement
family - relationship difficulties, divorce or caring for someone
financial problems - unexpected bills or borrowing money
health - illness, injury or losing someone (bereavement)
difficult past experiences - bullying, abuse or neglect
Even significant life events such as buying a house, having a baby or planning a wedding could lead to feelings of s
You might find it hard to explain to people why you feel this way, but talking to someone could help you find a solu

Source Document 2:

Symptoms of stress
Stress can ause many different symptoms. It might affect how you feel physically, mentally and also how you behave.

It's not always easy to recognise when stress is the reason you're feeling or acting differently.
Physical symptoms
headaches or dizziness
```

Figure 15: Displays that the model reads more than one document without reverting to previous training

Upon further testing I was able to validate that the model does read the necessary documents and only provides information that is within the text files to the same standard as before. It also can read multiple documents and still perform to the same standard as before.

Another issue I had to fix earlier in the experiments was the token count of the model with the llama2 model being capped at 512, which I was able to resolve by changing the context_length variable. Due to this, I wanted to see how different token lengths would affect the model. Would it make it slower? Is it going to be as accurate? To test this, I edited the notebook to have a list of token amounts ranging from 1 token to 4024 tokens. This list was then looped through asking the model the same question multiple times with different token values to see if the token amount affected the response time or accuracy. Each token amount was allowed to choose from multiple documents, with the question being *"What symptoms does Stress have?"*.

The response times of all token counts was below 5 seconds which is more than acceptable however, the responses came with some noteworthy results. Token values 1, 124, 256, 512 came back with the most comprehensible results, all the information was from the text files provided and was readable in very good English. However, token amounts 1024, 2048, 4024 even though they could read the full documents was not as clear as the other amounts. Using * symbols to split up the symptoms, even though that symbol is never mentioned in any of the documents chosen by the model to read, as seen in figure 16.

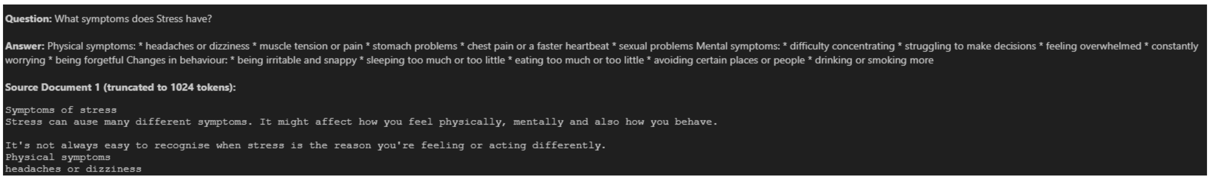

*Figure 16: Displays model using * instead of bullet points at higher token amounts*

After test different ways to fix this, I was unable to figure out why the token count affected the way that the document data was presented. My theory is that the model is trying to use bullet points to list the information in the document rather than display it in its raw form. So, without adding bullet points into the text files, which could ruin other results from the experiment, I don't think the model is advanced enough in displaying the correct grammar to list these options in the way that it wants too. This means that this could be a future piece of work to be conducted.

After completing all the experiments Meta released a new version of Llama3 called Llama3.1 This version is meant to be better than Llama3 in every way with the added version of 405B! Where Llama3 has 8B and 70B versions respectfully. Rather than doing the experiment again I decided to take a different approach to this to remove bias completely from the experiment. Instead, I created a notebook that uses dictionaries to store the two model's responses and displays these to the user, to choose the best response however, due to bias the user would always pick the newer model. So, I hid which model created which response until the user has chosen the best response. Then the model that created that response is revealed. This is done by changing the query function, firstly initialised the empty dictionaries which the is iterated over each model in the model list. Then it generates a random key assigned to the model names, responses and other important information once it receives the question. We then set up our two models to compare in this case Llama3 and Llama3.1. These models are added to the model list. The models are then asked the question and passed to the new query function. The loop extracts the model's name and path and generates a random key to store the model's response in a way that the identity is hidden. I then record the time taken to respond and the response to the question or *'query'* and this is then stored. I then randomise the order of the responses so it's not always Llama3 first and Llama3.1 second before returning the dictionaries. With this done I was able to instantiate the models and pass the question through each model before its run through the query function. By doing this the models print the responses to the terminal displaying which documents were used to answer the query, without revealing which is which until the user has decided.

```
Please evaluate the following responses and pick the best one:
Response 967244651: According to the text, the symptoms of depression include:

**Psychological Symptoms:**

1. Continuous low mood or sadness
2. Feeling hopeless and helpless
3. Having low self-esteem
4. Feeling tearful
```

Figure 17: Displays first models' response

```
Note that these symptoms can vary from person to person, and not everyone will experience all of them.
--------------------------------------------------------------------------------
Response 3752103566: The psychological symptoms of depression include:

1. Continuous low mood or sadness
2. Feeling hopeless and helpless
3. Having low self-esteem
4. Feeling tearful
5. Feeling guilt-ridden
6. Feeling irritable and intolerant of others
7. Having no motivation or interest in things
8. Finding it difficult to make decisions
9. Not getting any enjoyment out of life
10. Feeling anxious or worried
11. Having suicidal thoughts or thoughts of harming yourself

The physical symptoms of depression include:
```

Figure 18: Displays second models' response

```
------------------------------------
The chosen model is: llama3
```

Figure 19: Displays what model the user has chosen once you enter the response number

8. Website

With the experiments concluded I wanted to turn these experiments into something practical therefore I decided to make a working Chatbot, that uses the same model and RAG system that was implemented in the experiments above, notably from experiment 2 using the llama3 model from Ollama. The website is fully functional locally however has not been posted to the web due to publishing costs, however, could be published later.

8.1 File structure

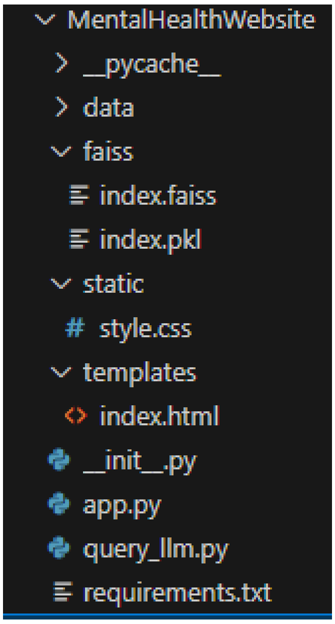

Figure 20: Shows the websites file structure

When creating the website, I started by looking into the file structure I wanted to use for the website. I settled on having a basic file structure, which you can see in figure 20. In this file structure you can see that all the text files are in a folder called data, this is where all the text files will be read from for the RAG system. Next, we have FAISS which is our database storage locally. Next template is where all HTML files will go, mainly our template for the home screen. We also have a static folder where I have put the style.css file which has the background colour and text colour stored in it, the styling of the website. Then outside the folders we have our requirements.txt which has the requirements to run the website and our app.py which is how we run the website, this also connects the HTML files to the query_llm file to allow the LLM to display its responses. Lastly, we have query_llm which is where the RAG system and LLM is put.

8.2 HTML Design

The design of the website is quite basic. I started by creating a very basic HTML script, which took a user input and displayed the input back to the user. I did this so I could easily change the features when the RAG system and LLM was implemented. Once the implementation of the RAG system and LLM was completed I changed the HTML to take user input, and then run this request through the LLM and display back the response to the user. I also created a history option when the previous questions will still be displayed when the user types in a new question, so you have chat history. However, after using the website for a while the history became confusing, and it was hard to read the newest question. Future work on the website could incorporate a history tab where you can go back and look at previous answers.

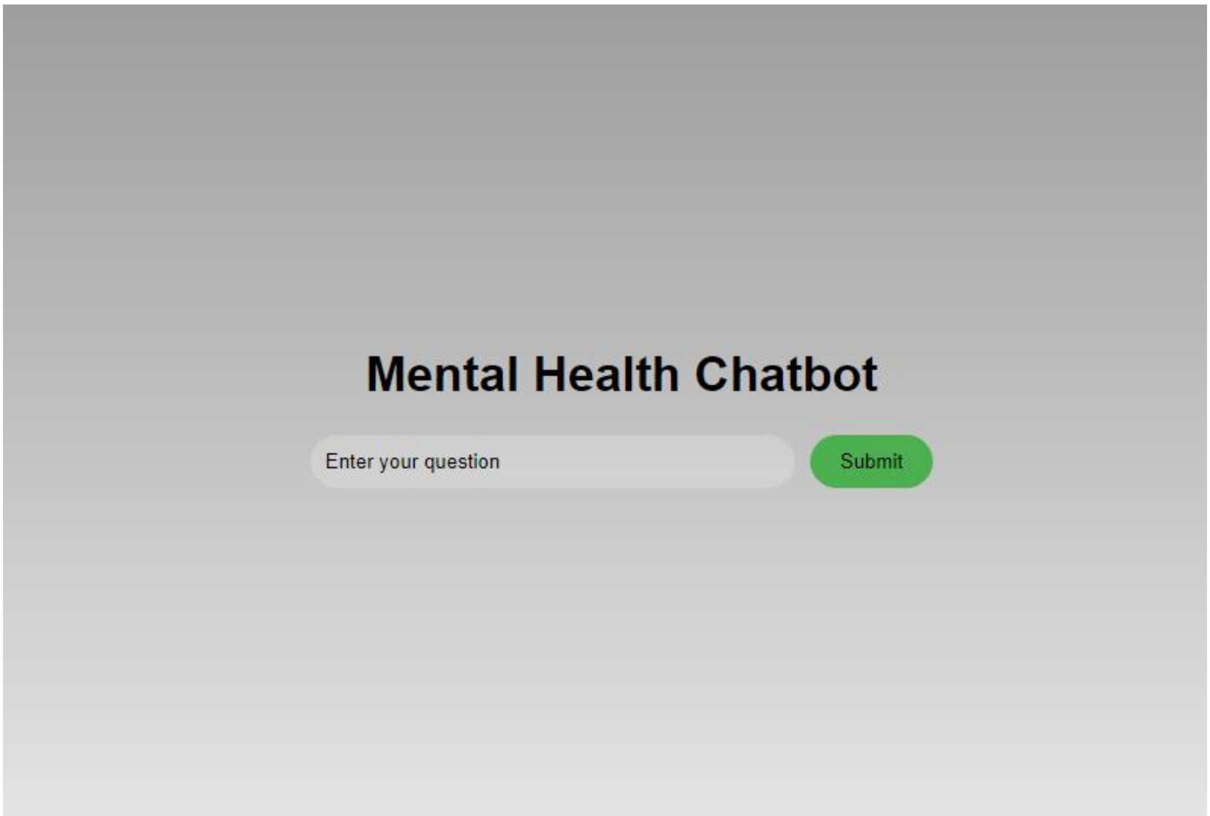

Figure 21: Shows the front page of the chatbot website

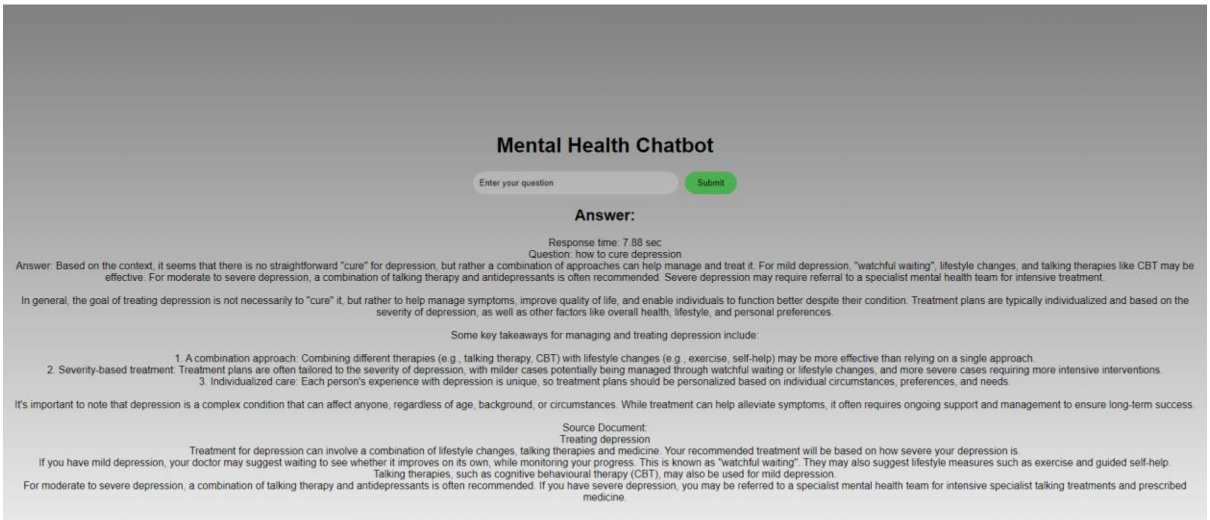

Figure 22: Shows the website after the user has entered a question and it's been answered

8.3 RAG Implementation

When the basic HTML was working, I then started to turn the Jupyter notebook into one python script that still followed the same RAG, text reading and model selection as the experiments. This was so the website would produce the response and times that I previously had. Since Llama3 was the best model from the experiment it was used as the website's model. I started by implementing the RAG system which was easy as it followed the same as the notebook like the model itself. Following some small errors, which were easily fixed, the debugging was completed, and the system worked within the terminal. Once working within the terminal, I added the required code to get the answers to be displayed within the HTML script so it would appear on the page. This did not take long and soon the chatbot was able to take the user input and run this through the model and display back the response to the user, with similar speeds and accuracy as our experiments.

9. Overall findings

This study has thoroughly evaluated various Large Language Models (LLMs), including LLaMA-2 and Llama 3, as well as Ollama models (Solar, Phi3, and Mistral). The primary objective was to assess these models' capabilities in generating accurate, relevant, and contextually appropriate responses to mental health-related queries. The evaluation focused on key metrics: response accuracy, relevance, and response time

- **LLaMA-2 Model Versions (Experiment 1)**: llama-2-7b-chat.ggmlv3.q8_0.bin emerged as the top performer within the LLaMA-2 family, delivering more accurate and relevant responses compared to the other Llama2 models. The llama-2-7b-chat.ggmlv3.q8_0.bin higher Q values contributed significantly to its superior performance, particularly when complex and nuanced mental health queries were involved. Nevertheless, these models were pale in comparison to the Ollama models which were tested afterwards.

- **Solar**: The Solar model from Ollama demonstrated competitive performance, especially in handling specialised queries that required specific and detailed knowledge. Although it performed well overall compared to the llama2 models, it was slower than all the other Ollama models.

- **Phi3 Model**: The Phi3 model showed promising results to begin with, particularly in terms of response time, being one of the fastest models competing with llama3. It performed significantly better than the Llama2 baseline. However, it did not quite reach the accuracy levels of the other Ollama models, especially in handling more sophisticated queries. Nonetheless, Phi3's performance indicates its potential as a viable alternative in scenarios where computational resources are more constrained and you are not looking for specialised responses.

- **Mistral Model**: The Mistral model, performed solidly, offering accuracy and relevance levels that were competitive with Llama3 and Solar, respectively. Notable for its strong performance in delivering relevant responses to complex queries, though it was still behind Llama 3 in the response time. The Mistral model is noteworthy for a balance between computational efficiency and output quality, making them attractive options for scenarios where resources are limited but high performance is still required.

- **Llama 3**: This model provided the most accurate and relevant responses across all the models evaluated. The combination of advanced architecture and understanding of the RAG system allowed Llama3 to blend and integrate external information effectively, making it particularly adept at handling complex mental health scenarios.

Best Performing Model: Llama3

The experiments identified Llama3 as the best-performing model overall. It delivers the highest accuracy, relevance, and ability to integrate external information, which is critical for complex and nuanced queries in mental health context.

10. Future Plans

The paper above explains multiple experiments that test different models to find the best one to use in a RAG system for a mental health chatbot; however, there is further work that can be conducted.

Firstly, the mental health illnesses throughout this paper have been focused on Depression, Anxiety and Stress however, there are a lot more mental health illnesses in the world such as, Bipolar Disorder, Post-Traumatic Stress Disorder (PTSD), Schizophrenia, just to name a few. The experiments here can serve as a template for further development by adding more text files on more illnesses, expanding the capabilities of the models RAG system providing the model is able to handle that much information. We could also change the format of the information provided, as it is not exclusive to text files. These files could be PDF or docx files, this works as the RAG system takes the text files as a string input, which many different file types can be. By doing this we can provide information from an academic piece written by doctors or, create a database with all the documents in and call the model to read from the database instead. Although the models will need to be tested again to make sure it works with these file types, theoretically there should be minimal difference.

With this paper being written in 2024 large language models are only at the beginning of their life cycle. As we have seen above new models are coming out every day, such as Llama 3.1, and old models are getting updated constantly. The results now are true and accurate however in the future there will be more advanced models with capabilities that we cannot fathom now. With each new update and advancement to the models we can install these into the system above and improve it further. We may have some new technologies to work with making the system even better than it already is. We may be able to handle more data than we can, allowing for even more accurate responses from the model or more accurate information on the Illnesses allowing for better advice. This isn't exclusive to the models, however. We may also get some future developments on RAG systems that should also be explored. If the RAG systems are better than the one implemented then we should investigate the benefits of these systems and implement them instead, so the experiment is constantly up to date.

As mentioned previously, more work could be done on working with higher token counts. I would suggest there should be an experiment to fix the way that models display information at higher token counts and, see if we are able to get models to display information grammatically correct and enhances readability. This would me the model use bullet points, numbers and other grammatical terms correctly.

Lastly, I was unable to test if the model would be able to pick out key information from documents that may have irrelevant information within them. I would suggest that experiments should be conducted on testing if models can read through the files and only pick out the key information within each file. This could be set up to have different illnesses in the same file and see if the model only displays information on the illness the user has spoken about or, put irrelevant information in the file that has the correct information within it and see if the model can pick out the correct information.

11. Conclusion

This report demonstrates that Retrieval-Augmented Generation (RAG) combined with Large Language Models (LLMs) like LLaMA-2 and Llama3 has significant potential in the development of chatbots for mental health self-management, with a website being developed in the report proving this. The experiments conducted revealed that while Llama 3 is the best model, until more models have been developed, in terms of response time and detail, however all the models tested benefit substantially from the integration of RAG, which allows them to provide more contextually relevant and accurate responses. Nevertheless, challenges remain, particularly concerning the balance between model power and the quality of output. Future work should focus on further refining these models, exploring additional data sources, and evaluating the system's effectiveness in real-world mental health scenarios. The findings underscore the growing role of AI in healthcare, offering promising avenues for enhancing mental health support through technology.

12. References

Journal and Conference Papers

1. H. Zhao, Q. Ling, Y. Pan, T. Zhong, J. Hu, and J. Yao, "Ophtha-LLaMA2: A Large Language Model for Ophthalmology," pp. 1-19.

2. Y. Yang, "LLaMA-Reg: Using LLaMA-2 for Unsupervised Medical Image Registration," pp. 1-34.

3. M. Polignano, B. Aldo, L. Siciliani, and B. Aldo, "LLaMAntino: LLaMA-2 Models for Effective Text Generation in Italian Language," vol. 1, no. 1.

4. H. Touvron, L. Martin, and K. Stone, "LLaMA-2: Open Foundation and Fine-Tuned Chat Models."

5. V. Sorin and E. Klang, "European Journal of Radiology Open," *European Journal of Radiology Open*, vol. 10, April 2023, Art. no. 100494.

6. U. C. Berkeley, "Is Bigger Edit Batch Size Always Better? - An Empirical Study on Model Editing with Llama-3," 2024.

7. T. T. Nguyen, C. Wilson, and J. Dalins, "Fine-Tuning LLaMA-2 Large Language Models for Detecting Online Sexual Predatory Chats and Abusive Texts," pp. 1-8.

8. B. M. Pavlyshenko, "Financial News Analytics Using Fine-Tuned LLaMA-2 GPT Model," pp. 1-14, 2023.

9. C. Study and D. Queries, "Enhancing LLM Factual Accuracy with RAG to Counter Hallucinations: A Case Study on Domain-Specific Queries in Private Knowledge-Bases."

10. S. Jana, R. Biswas, K. Pal, S. Biswas, and K. Roy, "The Evolution and Impact of Large Language Model Systems: A Comprehensive Analysis," vol. 13, pp. 65-77, March 2024.

11. M. U. Haque, I. Dharmadasa, Z. T. Sworna, R. N. Rajapakse, and H. Ahmad, "The Evolution and Impact of Large Language Model Chatbots in Social Media: A Comprehensive Review of Past, Present, and Future Applications," *Veri Bilimi*, vol. 6, no. 2, pp. 67-76, 2023.

12. F. Cuconasu et al., "The Power of Noise: Redefining Retrieval for RAG Systems," pp. 719-729, 2024.

13. Fan, W., Ding, Y., Ning, L., Wang, S., Li, H., Yin, D., Chua, T.-S., & Li, Q. (2024). *A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models*. http://arxiv.org/abs/2405.06211

Online Sources

14. "Signs of an Anxiety Disorder," NHS. Available: https://www.nhs.uk/mental-health/conditions/anxiety-disorders/.

15. "Depression in Adults," NHS. Available: https://www.nhs.uk/mental-health/conditions/depression-in-adults/.

16. "Get Help with Stress," NHS. Available: https://www.nhs.uk/mental-health/self-help/guides-tools-and-activities/tips-to-reduce-stress/.

17. R. Menon, "Introduction to Large Language Models and the Transformer Architecture," Medium. Available: https://rpradeepmenon.medium.com/introduction-to-large-language-models-and-the-transformer-architecture-534408ed7e61.

18. "How Transformers and Large Language Models (LLMs) Work," GopenAI Blog. Available: https://blog.gopenai.com/how-transformers-and-large-language-models-llms-work-3f20bb41c1ff.

19. "10 Things to Say and 10 Not to Say to Someone with Depression," Health. Available: https://www.health.com/condition/depression/10-things-to-say-and-10-not-to-say-to-someone-with-depression.

20. "Types of Depression," Mental Health UK. Available: https://mentalhealth-uk.org/help-and-information/conditions/depression/types-of-depression/.

21. "Depression Causes," NHS. Available: https://www.nhs.uk/mental-health/conditions/depression-in-adults/causes/.

22. "Causes of Stress," Mind. Available: https://www.mind.org.uk/information-support/types-of-mental-health-problems/stress/causes-of-stress/.

23. "Tips to Reduce Stress," NHS. Available: https://www.nhs.uk/mental-health/self-help/guides-tools-and-activities/tips-to-reduce-stress/.

24. “Llama-2 7B Chat”, Available: https://huggingface.co/spaces/huggingface-projects/llama-2-7b-chat

12.1 Additional Reading

Journal and Conference Papers

1. W. Du, T. Luo, Z. Qiu, Z. Huang, and Y. Shen, "Stacking Your Transformers: A Closer Look at Model Growth for Efficient LLM Pre-Training," pp. 1-40.
2. B. Peng, "On Limitations of the Transformer Architecture," pp. 1-14, 2024.
3. K. I. Roumeliotis, N. D. Tselikas, D. K. Nasiopoulos, and G. Scholar, "LLaMA-2: Early Adopters' Utilization of Meta's New Open-Source Pretrained Model," 2023.
4. L. G. May, "Strategies," pp. 1-6.
5. E. Wu and J. Zou, "Between RAG and LLMs' Internal Prior," pp. 1-11, 2023.
6. N. D. Q. Bui, H. Le, Y. Wang, J. Li, A. D. Gotmare, and S. C. H. Hoi, "S TATE - OF - THE - ART C ODE LLM," pp. 1-15.
7. O. Ridge and O. Ridge, "Comparing Llama-2 and GPT-3 LLMs for HPC Kernels Generation," pp. 1-13.
8. G. P. Transformer, A. Is, A. You, B. Transformers, T. Transformer, T. Transformer, L. Language, M. Meta, and M. Ai, "A Review of Transformer Models," vol. 5, pp. 1-12, 2023.
9. W. Fan et al., "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models," 2024.
10. M. Masalkhi et al., "A Side-by-Side Evaluation of LLaMA-2 by Meta with ChatGPT and Its Application in Ophthalmology," *Eye (Basingstoke)*, vol. 38, no. 10, pp. 1789-1792, 2024.
11. S. Sandmann, S. Riepenhausen, L. Plagwitz, and J. Varghese, "Systematic Analysis of ChatGPT, Google Search and LLaMA-2 for Clinical Decision Support Tasks," *Nature Communications*, vol. 15, no. 1, pp. 1-8, 2024.

Online Sources

12. "Llama-2-7B-Chat-GGML," TheBloke, Hugging Face. Available: https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGML#provided-files.
13. "Llama 3 Library," Ollama. Available: https://ollama.com/library/llama3.
14. "Meta Llama 3," Meta, Hugging Face. Available: https://huggingface.co/meta-llama/Meta-Llama-3-8B/discussions.
15. S. Vaniukov, "NLP vs LLM: A Comprehensive Guide to Understanding Key Differences," Medium. Available: https://medium.com/@vaniukov.s/nlp-vs-llm-a-comprehensive-guide-to-understanding-key-differences-0358f6571910.
16. "What Is a Large Language Model?," AWS. Available: https://aws.amazon.com/what-is/large-language-model/.
17. "Meta Llama 3 Blog," Meta AI. Available: https://ai.meta.com/blog/meta-llama-3/.

18. M. Gonzalez, "Breaking the Token Limit: How to Work with Large Amounts of Text in ChatGPT," Medium. Available: https://marco-gonzalez.medium.com/breaking-the-token-limit-how-to-work-with-large-amounts-of-text-in-chatgpt-da18c798d882.

19. "Character Text Splitter," LangChain Docs, Python LangChain. Available: https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/character_text_splitter/.

20. “Ollama models”, Available: https://ollama.com/library

21. “Using Llama 2 to answer questions about local documents”, Available: https://swharden.com/blog/2023-07-30-ai-document-qa/

22. “Using Llama 2 to create a document Q&A”, Available: https://medium.com/@srinandh28/using-llama-2-to-create-a-document-q-a-4b3a0a7898fe

23. “Ollama and LangChain: Run LLMs locally”, Available: https://medium.com/@abonia/ollama-and-langchain-run-llms-locally-900931914a46

24. “Ollama functions”, Available: https://python.langchain.com/v0.1/docs/integrations/chat/ollama_functions/

25. “Ollama”, Available: https://github.com/ollama/ollama?tab=readme-ov-file

26. “llama3”, Available: https://github.com/meta-llama/llama3/blob/main/README.md

27. “Getting started with Llama”, Available: https://llama.meta.com/docs/get-started/