Masters Programmes: Assignment Cover Sheet

Student Number:	**5588187**
Module Code:	**IB9CW0**
Module Title:	**Text Analytics**
Submission Deadline:	**6-June-24, 12:00 PM**
Date Submitted:	**6-June-24**
Word Count:	**1500**
Number of Pages:	**9**
Question Attempted: *(question number/title, or description of assignment)*	**Individual Assignment**
Have you used Artificial Intelligence (AI) in any part of this assignment?	**Yes**

Table of Contents

1. Introduction

The ever-growing field of natural language processing (NLP) has seen the rise of innovative techniques like retrieval-augmented generation (RAG) systems. They combine the strengths of two key NLP components:

- Information retrieval, which refers to the process of identifying pertinent information from large volumes of textual data, and,
- Large Language Models (LLMs), that are used to generate human-like texts and summarise outputs.

RAGs are significant improvement over LLMs (Lewis et al., 2020), which struggle to generate contextual outcomes. They overcome this shortcoming by enriching user queries with contextual relevance, making them more reliable by increasing their accuracy (Dash, 2023). A RAG system leverages information retrieval to find specific pieces of text relevant to a user's query. It then feeds this retrieved information, along with the original query, to an LLM. The LLM then utilizes this combined information to generate a comprehensive and informative response. Its architecture can be seen in the given Fig. 1.

Diagram of functioning of RAG

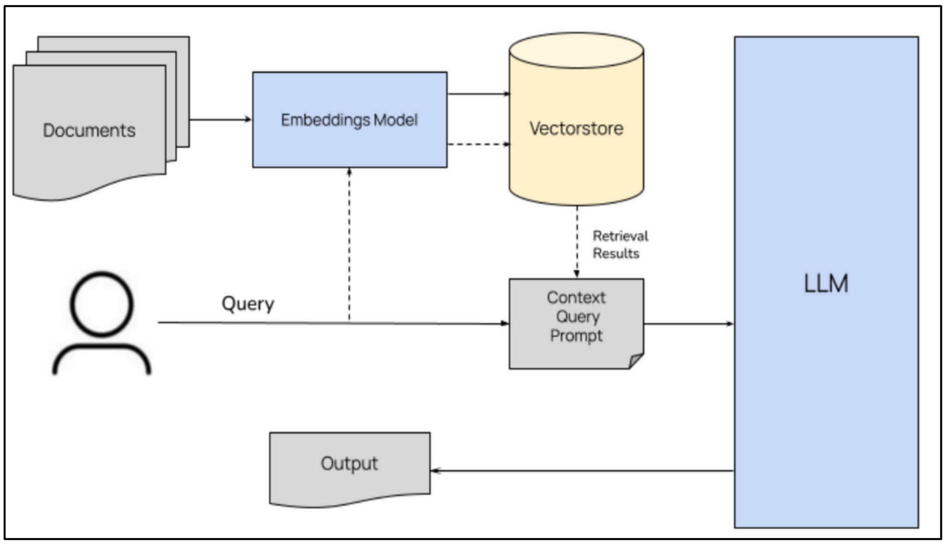


Fig.1 RAG Architecture (LeNormand, 2024)

2. Motivation

The objective of this project is to create a Retrieval-Augmented Generation (RAG) system specifically designed for "Harry Potter and the Sorcerer's Stone" (referred to as "Philosopher's

Stone" in certain regions). The system will utilise pre-processed text data from Kaggle (Moxxis, 2024) to allow users to delve into the world of Harry Potter by:

- Improved capability of providing comprehensive answers to specific inquiries regarding various aspects, such as the history of Harry Potter. These queries may not be effectively addressed by LLMs, since they could lack pre-training on this particular subject matter, and,
- Producing informative descriptions of particular chapters or character profiles, utilising contextual details, to assist users in reviewing key elements of story or exploring objectives of the character.

This RAG system, which integrates context-based information retrieval with the capabilities of LLMs (Finardi et al., 2024), will be a great tool for obtaining accurate and trustworthy information for admirers of the Harry Potter series.

3. Steps involved

a. Preparing the Environment

To start, we install the required packages for dealing with the LLM, chunking, and embedding the text. Phi 3 (Bilenko, 2024), that refers to the LLM has been installed using Ollama. The Llama Index (Llamaindex, n.d.) was utilised to establish a connection between data sources and the LLM (Phi 3). The FAISS GPU library (Meta AI, 2024) was installed to store embeddings, while the HuggingFace embedding model was installed to turn the chunks into vectors. This is a crucial step for setting up our environment to handle the task of efficient information retrieval.

b. Loading the Text Data and chunking it

We retrieve the textual information from the Harry Potter book and divide it into small chunks using sentence splitter chunking technique. Multiple chunking methods are available (Schwaber-Cohen, 2023), including sentence splitting, semantic chunking, specialised chunking, recursive chunking, and others, to meet the specific requirements of the task. Sentence splitter is our preferred choice as it divides the long text into individual sentences and generating meaningful segments that are necessary for embedding. It is crucial to perform this step to verify that the text is in an appropriate format required for processing and embedding.

c. Configuring LlamaIndex to store embedding after integrating it FAISS

Next, we must utilise an existing embedding model to embed the text segments and then save these embeddings in a vector database (FAISS). This enables the effective identification of comparable texts and subsequent retrieval of relevant documents based on user queries. We utilise the pre-trained model "BAAI/bge-small-en-v1.5" (Hugging Face, n.d.) from Hugging Face's LlamaIndex configuration to produce numerical representations (embeddings) of the text input. The integration of Faiss into LlamaIndex allows for the creation of a searchable data structure. This process involves including the text data and enabling the ability to perform semantic searching.

d. Preparing Question for Searchable Index

Afterwards we transform the question into the vector space using the same pre-trained model described above and compare it to the stored text data in the embedding space. Subsequently, the model selects and retrieves the most relevant sentences that answer the question based on their similarity in the embedding space. Different types of similarity models, such as cosine similarity, Euclidean similarity and others (Majumder et al., 2016), can be used to perform semantic search. Here, cosine similarity was used to search similar texts from our vector database.

e. Build the RAG System

Then our RAG was constructed by installing Phi 3 using the Ollama. We then proceed to set up the model and configure it within the Colab environment, using Linux/BASH instructions in the background. The pre-trained LLM (Phi 3) is utilised to generate text specifically concerning the Harry Potter series. This generated text will then be used to compare and assess the performance of our RAG model in comparison to the LLM. The environment is set up for running the LlamaIndex alongside Ollama, taking advantage of the powerful capabilities of LLMs in integrating text indexing and search features. Additionally, a timeout mechanism, a hyperparameter used to detect errors when the execution time exceeds a certain threshold, is also configured to effectively manage any errors that occur during the run-time.

f. Creating Prompt Template and for usage

Text-based question answering prompt template is the penultimate step. Importing ‘ChatMessage’ and ‘ChatpromptTemplate’ from LlamaIndex creates organised Q&A prompts for our RAG system. We create a Q&A prompt string with placeholders for context and query. It provides contextual information and enables the model to answer a question based on it. We next generate a ChatMessage list. The first message is SYSTEM and says, "Always answer the question, even if the context isn't helpful." So the model always tries to answer. The second message, USER, builds user input using the previous template. Finally, we create a ChatPromptTemplate from these chat messages. This architecture helps the model efficiently process context and queries, ensuring relevant and structured responses.

g. Integrating and executing the Query Engine to demonstrate the outcome

To implement the Question-Answer (Q&A) system, we create a temporary query engine using the ‘as_query_engine’ method. Here, previously defined Q&A prompt template was utilised, ensuring that user inputs and context are structured effectively for processing. We can then set the response modes, tree summarize in our case, to determine how the response is generated, providing a summarized answer. We can also use different types of response modes, such as ‘Bullet Points’, ‘concise response’, and others to structure our responses. Finally, the query ("Who is Harry Potter?") method executes the query, and displays the outcome. This integration ensures that the system can provide structured and relevant responses to user queries, demonstrating the practical application of the setup.

4. Advantages of RAG systems over LLMs

Using RAG systems over has many LLMs benefits, in context of the book ‘Harry Potter and The Sorcerer’s Stone’ it offers many advantages. Some of them are listed below –

- RAG improves accuracy by retrieving responses from specific passages of the book, ensuring that answers are precise and accurate. It can be seen in the results, where outcome of RAG system (see Fig.3 and Appendix 2) is more accurate and contextually relevant than the outcome of the LLM (see Fig.2 and Appendix 1)

- Additionally, they also reduce the occurrence of hallucinations (Martineau, 2021) as the system obtains precise information from the text-based sources instead of potentially incorrect details of the LLMs.
- In addition, RAG systems are very scalable and can efficiently handle large datasets, such as several Harry Potter volumes.
- They reduce the computing cost of the LLM, making them less expensive in operation
- Furthermore, they enable continuous knowledge updation, allowing the system constantly take in new data and keep answers up to date without requiring expensive retraining.

It has several benefits that make it an effective solution for applications that need precise, contextually related, and scalable replies.

5. Limitations and challenges

While RAG systems have numerous benefits and serve as powerful instruments for improving NLP, they have certain limitations and shortcomings. They are –

- Due to their high demand for computational power, storage capacity, and data processing capabilities, they are cost and resource-intensive in nature
- As the size of the dataset grow, it becomes challenging to find a balance between improving accuracy and reducing response time (Thakur, 2023).
- They require fine-tuning several hyperparameters, such as the timeout value and query response style. This increases the complexity of the system.
- Utilising complex evaluation metrics to analyse the success of the RAG systems presents significant challenges. Terms like precision, accuracy, and contextual relevance are very subjective and rely on the quality and veracity of the external data provided.
- Technical challenges emerge when integrating these elements with the existing setup due to their complex designs and architecture. This often lead to difficulties in creating a system that is integrated with the current configuration.

These constraints emphasise the importance of careful planning and execution while building a RAG system and subsequently integrating it into present framework.

6. Conclusion

This report highlights the benefits of constructing a RAG system over the existing LLM, through an example of the development for the novel 'Harry Potter and the Sorcerer's Stone'. By leveraging the two important components of NLP - information retrieval and LLMs, it has enhanced accuracy, reduced hallucinations, and improved contextual relevance of the responses when compared with independent LLM. Whilst the report provides an educational overview, details of the steps in the formation of the RAG system and its advantages, it also discusses about the limitations due to computational constraints. Additional research is required to enhance the technology and overcome the limitations for harnessing the strength of NLP through the development of a precise and scalable RAG system.

7. References

Bilenko, M. (2024). New models added to the Phi-3 family, available on Microsoft Azure. [online] Microsoft Azure Blog. Available at: https://azure.microsoft.com/en-us/blog/new-models-added-to-the-phi-3-family-available-on-microsoft-azure/. [Accessed 2 Jun. 2024]

Dash, S. (2023). What is Retrieval-Augmented Generation (RAG) in AI? [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2023/09/retrieval-augmented-generation-rag-in-ai/. [Accessed 2 Jun. 2024]

Finardi, P., Avila, L., Castaldoni, R., Gengo, P., Larcher, C., Piau, M., Costa, P. and Caridá, V., 2024. The Chronicles of RAG: The Retriever, the Chunk and the Generator. arXiv preprint arXiv:2401.07883.

Hugging Face (n.d.). BAAI/bge-small-en · Hugging Face. [online] Available at: https://huggingface.co/BAAI/bge-small-en [Accessed 2 Jun. 2024]

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T. and Riedel, S., 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, pp.9459-9474.

LeNormand, G. (2024). Mastering Retrieval-Augmented Generation (RAG) Architecture: Unleash the Power of Large Language Models. *Medium*. Available at: https://medium.com/@glennlenormand/mastering-retrieval-augmented-generation-rag-architecture-unleash-the-power-of-large-language-a1d2be5f348c [Accessed: 3 June 2024]

LlamaIndex. (n.d.). Local Embeddings with HuggingFace - LlamaIndex. [online] Available at: https://docs.llamaindex.ai/en/stable/examples/embeddings/huggingface/ [Accessed 1 Jun. 2024]

Majumder, G., Pakray, P., Gelbukh, A. and Pinto, D., 2016. Semantic textual similarity methods, tools, and applications: A survey. Computación y Sistemas, 20(4), pp.647-665.

Martineau, K. (2021). What is retrieval-augmented generation? [online] IBM Research Blog. Available at: https://research.ibm.com/blog/retrieval-augmented-generation-RAG. [Accessed: 4 June 2024]

Meta AI (2024). FAISS. [online] Available at: https://ai.meta.com/tools/faiss/. [Accessed 2 Jun. 2024]

Moxxis (2024). Harry Potter all books(pre-processed). Kaggle. Available at: https://www.kaggle.com/datasets/moxxis/harry-potter-lstm. [Accessed: 4 June 2024]

Schwaber-Cohen, R. (2023). Chunking Strategies for LLM Applications. *Pinecone* [online]. Available at: https://www.pinecone.io/learn/chunking-strategies/. [Accessed 1 June 2024]

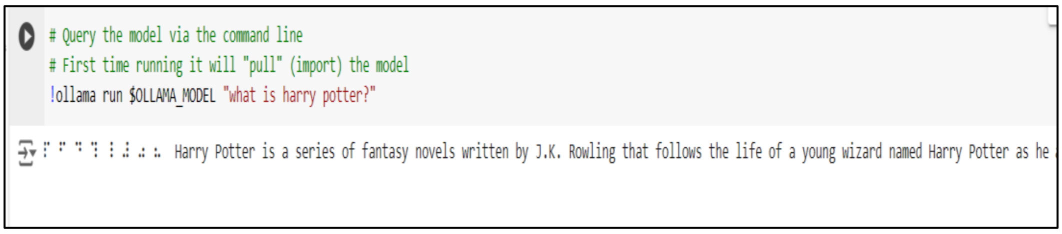
Thakur, P. (2023). A Gentle Introduction to Retrieval Augmented Generation (RAG). [online] W&B. Available at: https://wandb.ai/cosmo3769/RAG/reports/A-Gentle-Introduction-to-Retrieval-Augmented-Generation-RAG---Vmlldzo1MjM4Mjk1#challenges-and-future-directions-of-rag [Accessed 3 Jun. 2024]

8. Appendices

i. Appendix 1 – Outcome of LLM

```
⠋ ⠙ ⠹ ⠸ ⠼ ⠴ ⠦ ⠧ Harry Potter is a series of fantasy novels written
by J.K. Rowling that follows the life of a young wizard named Harry
Potter as he attends Hogwarts School of Witchcraft and Wizardry,
battles dark wizards, and tries to save the wizarding world from
evil forces.
```

Screenshot of the LLM outcome



```
# Query the model via the command line
# First time running it will "pull" (import) the model
!ollama run $OLLAMA_MODEL "what is harry potter?"
```

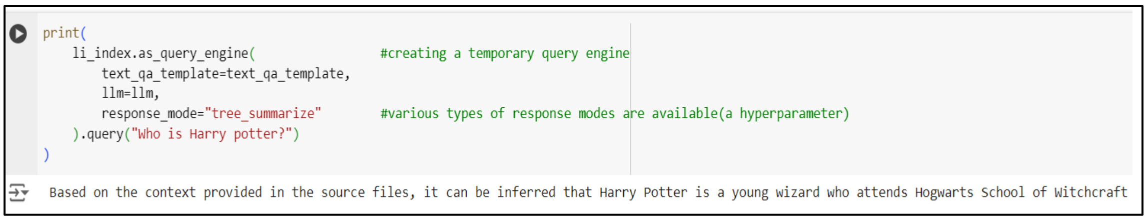
```
⠋ ⠙ ⠹ ⠸ ⠼ ⠴ ⠦ ⠧  Harry Potter is a series of fantasy novels written by J.K. Rowling that follows the life of a young wizard named Harry Potter as he
```

Fig.2 – Outcome of LLM

ii. Appendix 2 – Outcome of RAG

```
Based on the context provided in the source files, it can be
inferred that Harry Potter is a young wizard who attends Hogwarts
School of Witchcraft and Wizardry. He has been receiving letters
from the school and has learned all the course books by heart.
However, there seems to be a scandal surrounding his identity as
he claims to be Harry Potter but other kids in their world are also
aware of him. The only person who is aware of this is Hagrid, who
keeps Harry hidden and informs Dumbledore about it.
```

Screenshot of the RAG outcome



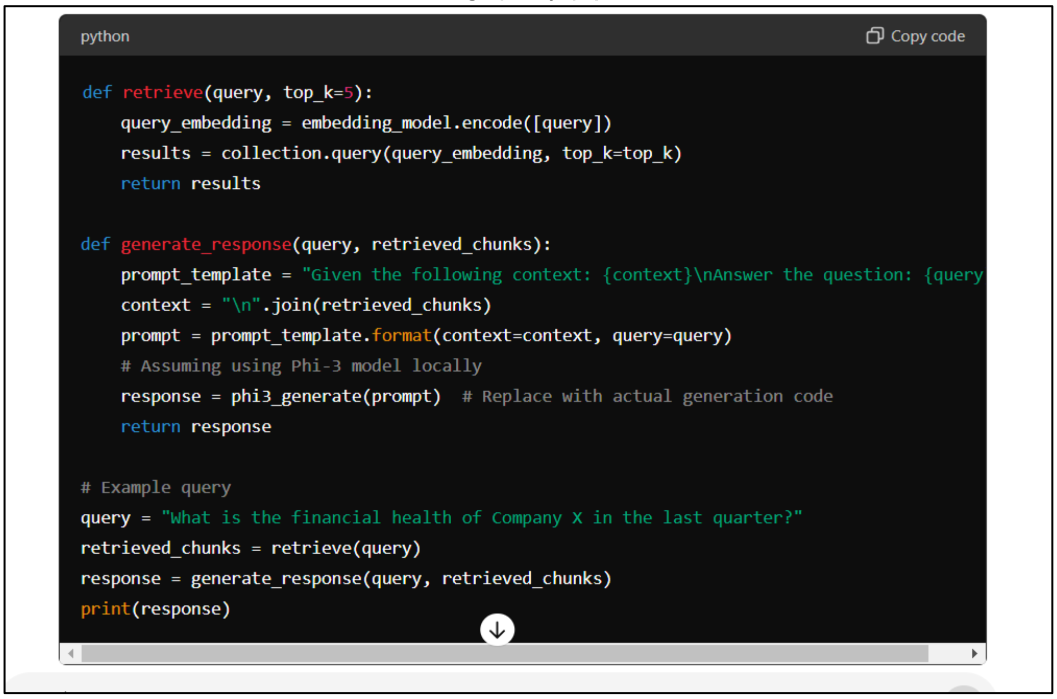
```
print(
    li_index.as_query_engine(               #creating a temporary query engine
        text_qa_template=text_qa_template,
        llm=llm,
        response_mode="tree_summarize"      #various types of response modes are available(a hyperparameter)
    ).query("Who is Harry potter?")
)
```

```
Based on the context provided in the source files, it can be inferred that Harry Potter is a young wizard who attends Hogwarts School of Witchcraft
```

Fig. 3 – Outcome of RAG

iii. Appendix 3 – Use of AI

For creating query pipeline



```python
def retrieve(query, top_k=5):
    query_embedding = embedding_model.encode([query])
    results = collection.query(query_embedding, top_k=top_k)
    return results

def generate_response(query, retrieved_chunks):
    prompt_template = "Given the following context: {context}\nAnswer the question: {query
    context = "\n".join(retrieved_chunks)
    prompt = prompt_template.format(context=context, query=query)
    # Assuming using Phi-3 model locally
    response = phi3_generate(prompt)  # Replace with actual generation code
    return response

# Example query
query = "What is the financial health of Company X in the last quarter?"
retrieved_chunks = retrieve(query)
response = generate_response(query, retrieved_chunks)
print(response)
```

Fig.4 – Use of AI in creating query pipeline