Assignment 02
SE4010 – Current Trends In Software Engineering

IT Number	Name
IT21228612	De Silva S.J.W.

May 2025

BSc (Hons) in Information Technology Specialising in Software Engineering

Table of Contents

1 Introduction 2
2 Justification of LLM Choice 4
3 Justification of Development Approach 6
4 Challenges and Lessons Learned 9
4.1 Challenges Encountered 9
4.2 Lessons Learned 10
5 Use of GenAI Tools (Prompts and Outputs) 11
6 Video Demonstration Link 11
7 Conclusion 12

1 Introduction

Large Language Models (LLMs) are advanced AI systems trained on vast datasets to understand and generate human-like text. Despite their capabilities, LLMs are not inherently aware of private or domain-specific content, such as lecture materials. To overcome this limitation, the Retrieval-Augmented Generation (RAG) technique is used. This method augments the LLM with an external document retrieval mechanism, allowing it to respond with higher accuracy and contextual relevance.

In this assignment, the RAG technique is used to create a chatbot that can answer questions strictly based on the *Current Trends in Software Engineering (CTSE)* module lecture notes. The chatbot is implemented in a Jupyter Notebook, where each code cell corresponds to a key processing stage in the pipeline. This introduction walks through the actual steps taken in the notebook to build the chatbot.

1. **Lecture Note Extraction**: The process begins by reading a combined PDF of CTSE lecture slides and extracting the full text content from each page. The extracted text is saved as a .txt file, forming the raw dataset that powers the chatbot.

2. **Text Cleaning**: The raw extracted text is cleaned to remove unnecessary artifacts like page numbers or whitespace clutter. This ensures that the content is in a consistent format, suitable for downstream processing.

3. **Document Chunking and Semantic Indexing**: The cleaned text is loaded and split into smaller overlapping chunks to preserve context. Each chunk is then converted into a numerical embedding using a high-performance sentence embedding model (intfloat/e5-large-v2). These embeddings are stored and indexed using FAISS—a vector search library that enables rapid similarity-based retrieval.

4. **Retriever Initialization**: The precomputed FAISS index and associated document chunks are reloaded, and a custom retriever is defined. This retriever accepts a user's query, computes its semantic embedding, and finds the top-matching lecture chunks based on similarity.

5. **Language Model Setup**: A text-to-text generation model (Flan-T5) is initialized using the Hugging Face transformers library. It serves as the LLM component that will produce the final response by conditioning on both the user's question and the retrieved lecture content.

6. **Prompt Design and Chat Loop**: A structured prompt template is defined to guide the LLM in generating grounded answers strictly from the provided lecture material. A chat interface is then set up to allow users to ask questions interactively. The system maintains a short history of previous turns and ensures that responses are based only on lecture-relevant content. If the answer is not found in the notes, the bot is designed to respond with "I don't know based on the lecture notes."

This structured pipeline allows for an interpretable and accurate chatbot experience, ensuring that all answers are grounded in the actual CTSE lecture material. The full technology stack includes:

- **PyMuPDF** – For PDF text extraction.
- **LangChain** – For document loading, chunking, and prompt handling.
- **SentenceTransformers (intfloat/e5-large-v2)** – For semantic vectorization.
- **FAISS** – For efficient vector similarity search.
- **Transformers (Flan-T5)** – For natural language generation.
- **Jupyter Notebook** – As the interactive development and deployment environment.

Through these carefully defined steps, the chatbot provides helpful, focused answers, supporting students in revisiting and reinforcing lecture content with clarity and confidence.

2 Justification of LLM Choice

In this assignment, the choice of a suitable Large Language Model (LLM) is critical for ensuring that the chatbot can provide accurate and coherent responses based on the lecture materials. After considering several LLMs, the Flan-T5 model was selected for the following key reasons:

1. **Task-Specific Fine-Tuning** - Flan-T5 is a fine-tuned version of Google's T5 (Text-to-Text Transfer Transformer) model. Unlike standard pre-trained models, Flan-T5 has been specifically fine-tuned to follow instructions and respond appropriately to structured tasks, such as question answering. This makes it ideal for generating answers based on the specific context of the CTSE lecture notes.

1. **High Performance on Text Generation Tasks** - Flan-T5 is known for its strong performance on text generation tasks, which include summarization, translation, and question answering. It is capable of understanding the context within a conversation and generating coherent, contextually appropriate answers. This is crucial for educational chatbots where the responses must be both relevant and understandable to users.

2. **Flexibility and Customization** - Flan-T5 supports prompt-based customization, allowing us to design a prompt template that strictly limits the LLM's responses to the provided lecture material. This capability is essential in ensuring that the model does not "hallucinate" information or generate off-topic responses, which could happen if the model were left to freely generate based on its pre-existing knowledge base.

3. **Efficiency** - Flan-T5 is relatively efficient in terms of computational resources, especially when compared to larger models like GPT-3. Its relatively smaller size (compared to other large models) allows it to run efficiently in the context of an educational chatbot while still providing high-quality responses.

4. **Support for Integration** - Flan-T5, being part of the Hugging Face Transformers library, provides smooth integration with other tools used in the assignment, including the LangChain framework for prompt handling and FAISS for document retrieval. This integration allows for an efficient workflow where the LLM can generate answers based on dynamically retrieved content.

5. **Free to Use** - One of the key advantages of using Flan-T5 for this assignment is that it is freely available through the Hugging Face platform. Hugging Face offers an accessible API for the model, which enables free usage within certain limits. This makes Flan-T5 an excellent choice for educational and research-based assignments that have limited computational resources or budgets.

6. **Attempt to Use Stronger LLMs** - In an effort to improve the chatbot's performance, I explored stronger LLMs with larger parameter sizes, such as:
    - **google/flan-t5-large**: A model with 3 billion parameters, which requires approximately 12–16 GB of VRAM. It is known for its excellent performance on sequence-to-sequence tasks, including answering questions based on structured input.
    - **google/flan-t5-xl**: A variant of the above model, also with 3 billion parameters but distributed across two shards, requiring 24–30 GB of VRAM, making it more difficult to load and utilize efficiently.
    - **meta-llama/Llama-2-7b-chat-hf**: A 7-billion parameter model optimized for chat-based applications, requiring 14–20 GB of VRAM (using FP16 precision). This model is designed for conversational fluency and would have significantly improved the quality of the chatbot's responses.

    However, due to the hardware limitations of my laptop, which lacks the required VRAM to run these larger models, I was unable to use them in the assignment. Despite this, Flan-T5 provided a good balance of performance and resource efficiency, making it a suitable choice for the task.

3 Justification of Development Approach

The design of the CTSE Lecture-Notes Chatbot was carefully considered to ensure the system could generate fluent and contextually accurate responses based solely on the provided lecture content. Each decision in the development process was made to optimize performance, accuracy, and resource efficiency, while prioritizing free tools and resources wherever feasible. Below, I outline why specific approaches and tools were chosen over other alternatives.

1. Text Extraction and Cleaning:

Choice of PyMuPDF (fitz) for PDF Parsing:

- I opted for PyMuPDF to extract text from the provided PDF because it provides a reliable and flexible method for handling PDFs of various complexities, including those with images, tables, and complex layouts. PyMuPDF is known for its accuracy in extracting clean text, which is crucial for the next steps in the pipeline. It is also open-source and free to use, making it an ideal choice for cost-conscious projects.
- Other libraries like pdfminer or PyPDF2 were also considered. However, PyMuPDF was selected because it is faster and offers better text structure retention (e.g., correct extraction of paragraphs and headings). This is important when dealing with lecture notes where formatting and content clarity are essential for accurate text cleaning.
- While pdfminer and PyPDF2 are also free, PyMuPDF provided better overall performance and ease of integration, making it the preferred choice.

Cleaning Approach:

- The cleaning step focused on removing irrelevant artifacts such as page numbers, headers, and footers. This was necessary to ensure the extracted text was clean and useful for downstream processing. The cleaning was done using regular expressions (regex) and basic Python string manipulation, both of which are built-in and completely free tools. Manual or paid solutions were avoided in favour of this efficient and scalable approach.

2. Text Chunking and Embedding Generation:

Choice of Sentence Transformers (e5-large-v2):

- For generating the embeddings, I selected the intfloat/e5-large-v2 model, available on Hugging Face, because it is free to use and produces high-quality vector representations of text. The decision to use this model was based on its balance of accuracy and computational efficiency. Hugging Face offers numerous free models, including this one, and it provides free access to the model via their API, making it a viable and cost-effective solution.

- While other embedding models, such as BERT or DistilBERT, could have been used, the e5-large-v2 model was preferred because it is optimized for generating dense, semantically rich embeddings suitable for similarity-based search tasks. This made it particularly well-suited for use with FAISS, ensuring that the retrieved content accurately reflected the meaning of the user's query.
- Other models like BERT-based variants would have required more fine-tuning to produce embeddings that were as effective for this task, making the e5-large-v2 model a more out-of-the-box solution.

3. FAISS Indexing for Efficient Retrieval:

Choice of FAISS (Facebook AI Similarity Search):

- FAISS was chosen as the indexing and search library due to its high-performance capabilities in similarity search. FAISS is optimized for handling large-scale data and is extremely efficient when working with high-dimensional vector spaces, making it an ideal choice for our embedding-based retrieval system. It is open-source and free to use.
- Alternatives such as Elasticsearch or traditional relational databases were considered, but FAISS was selected because it is purpose-built for similarity search, allowing for much faster retrieval times. FAISS also supports GPU acceleration, which improves performance, especially when dealing with large datasets.
- While other retrieval methods, like BM25 or vector databases like Pinecone, could have been used, FAISS was preferred for its ease of integration with the sentence embeddings and its ability to scale efficiently as the lecture material grows, all while remaining free.

4. Language Model for Answer Generation:

Choice of Flan-T5 (Base Model):

- For generating responses to user queries, the Flan-T5 model was chosen due to its strong performance on instruction-based tasks, such as answering questions. The model has been fine-tuned for tasks like question answering, making it particularly well-suited for this project. The Flan-T5 base model is available for free through Hugging Face, offering excellent performance without the need for expensive API services.
- While other models like GPT-3 or GPT-4 could have been considered, they would have required paid access, making them less feasible for this project. Flan-T5 was selected because it strikes a good balance between computational efficiency and performance. It is less resource-intensive than models like GPT-3 or GPT-4, making it an ideal solution for keeping the chatbot responsive and cost-effective.

- Unlike GPT models, which are more conversational and prone to generating text that might deviate from the lecture content, Flan-T5 is trained for sequence-to-sequence tasks and is therefore more reliable for the focused objective of answering based solely on lecture notes.

5. Retriever System for Contextual Answering:

Choice of Custom Retriever with FAISS:

- The retriever system was built using FAISS as the core engine for similarity search. This design was chosen because it enables the chatbot to efficiently pull relevant sections of lecture material based on a query. By combining FAISS with sentence embeddings from e5-large-v2, the system ensures that the most relevant chunks of lecture notes are retrieved based on their semantic similarity to the user's query.
- Alternative approaches like rule-based retrieval or keyword search were considered, but they were discarded because they lack the flexibility and semantic understanding necessary to effectively handle the diverse range of possible user queries. FAISS, when used with embedding-based retrieval, ensures that the system is capable of understanding and processing the underlying meaning of a query, rather than just relying on exact keyword matches.

6. Prompt Design for Controlled Responses:

Choice of Strict Prompt Template:

- The prompt template was designed to strictly constrain the model to only answer based on the provided lecture notes. This intentional design decision ensures the model remains grounded in the provided content, preventing it from generating answers that may be incorrect or fabricated. The strict template was chosen to maintain accuracy and reliability.
- More flexible prompt templates that allow for free-form answers could have been used, but these would have led to answers that may not be strictly based on the lecture content, which would undermine the chatbot's educational purpose. The strict template ensured the model's responses were consistently tied to the lecture material, which is crucial for the task at hand.

4 Challenges and Lessons Learned

4.1 Challenges Encountered

1. Handling PDF Extraction Noise : Extracting lecture notes using PyMuPDF (fitz) often included unwanted elements like page numbers and headers. A dedicated text cleaning step was necessary to produce semantically clean input for chunking and embedding.

2. Vector Store Compatibility Issues (Chroma → FAISS) : Initially, Chroma was used as the vector database. However, it caused persistent version conflicts and runtime errors within the Jupyter environment. Despite efforts to resolve the issues, Chroma remained unstable. The decision was made to switch to FAISS, which offered better reliability and compatibility with NumPy-based embeddings and provided consistent performance across sessions.

3. Embedding Time and Computational Overhead : Using the intfloat/e5-large-v2 model to embed lecture content took 30–40 minutes, depending on system performance. Initially, this embedding logic was placed inside a single cell along with downstream code, causing the entire embedding process to re-run if anything failed. This inefficiency was resolved by separating the embedding process into its own cell.

4. Long-running Tasks Without Feedback (Solved with TQDM and Logs) : When embedding and indexing large documents, there was no way to visually determine if the code was stuck or progressing. To solve this, the tqdm library was used to add progress bars for looped operations, particularly during chunk embedding. Additionally, status print messages like "FAISS index built." and "Index and docs saved." were added after key steps. This provided immediate feedback and greatly helped in monitoring long-running operations.

5. Optimal Chunking Strategy : Choosing the right chunk_size and chunk_overlap in RecursiveCharacterTextSplitter was crucial. Chunks that were too small missed context; chunks that were too large either exceeded embedding model limits or introduced redundancy. A 400-character chunk size with 100-character overlap was chosen after several trials.

6. Saving & Reloading FAISS Index : To support repeated use of the chatbot without re-embedding, the FAISS index and document metadata were saved to disk (faiss_index/index.faiss and docs.pkl). Proper serialization and path handling ensured they could be reloaded efficiently in subsequent sessions.

7. Prompt Length and Truncation Management : Since too much contextual input could exceed token limits or confuse the model, a hard limit of 3000 characters was set for combined retrieved content. When exceeded, content was truncated with a warning like ...[truncated] to maintain model performance and avoid token overflow errors.

8. LLM Pipeline Resource Management : Running google/flan-t5-base in a local Hugging Face pipeline led to occasional memory pressure. Parameters like max_new_tokens=150 were fine-tuned to prevent timeouts or incomplete responses during extended sessions.

4.2 Lessons Learned

1. Transparent Progress Logging : Adding progress bars (via tqdm) and status messages for embedding and indexing was essential. Without them, it was nearly impossible to tell if the system was functioning or stuck. This small addition significantly improved debugging, user experience, and trust in the pipeline.

2. Swapping Tools Is Sometimes Necessary : Chroma's integration problems demonstrated that newer tools, while promising, may not yet be stable across all environments. FAISS, being battle-tested and straightforward, turned out to be a more practical choice.

3. Modular Cells Save Time : Separating major stages like embedding, indexing, and interaction into independent notebook cells minimized rework and allowed for faster iteration—especially during debugging or configuration changes.

4. Good Chunking Enables Better Retrieval : The effectiveness of RAG systems is heavily influenced by how the source documents are chunked. Overlapping and sizing chunks thoughtfully preserved context and improved retrieval quality.

5. Prompt Engineering Is Crucial in RAG : A well-crafted prompt, including explicit instructions to limit answers to the provided content, helped mitigate hallucinations and encouraged the model to defer with "I don't know based on the lecture notes" when uncertain.

6. Persistence Mechanisms Prevent Re-computation : Storing the FAISS index and metadata on disk not only saved time but also reduced compute costs when rerunning the chatbot across multiple notebook sessions.

5 Use of GenAI Tools (Prompts and Outputs)

Below is the link to my ChatGPT conversation, which I used to troubleshoot issues, refine the system architecture, and plan the assignment. To ensure clarity and traceability without cluttering the report, I've shared the full chat instead of including screenshots.

- https://chatgpt.com/share/6821ba46-7224-8011-a7b4-1fa6f288f84f

6 Video Demonstration Link

https://mysliit-my.sharepoint.com/:v:/g/personal/it21228612_my_sliit_lk/EVDY6QwvLfpCiBZaifBszRYBEwKb0O37jPal6dq0vAi8gw?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=8ydjnU

7 Conclusion

The project successfully implemented a chatbot system leveraging Retrieval-Augmented Generation (RAG) techniques to enable intelligent querying over lecture materials. By combining tools such as SentenceTransformers for embedding generation, FAISS for vector indexing and retrieval, and the Flan-T5 language model for response generation, the system demonstrated the ability to provide accurate and contextually relevant answers grounded in the content of the course lectures.

Initially, Chroma was considered for vector storage and retrieval; however, due to version conflicts and instability, it was replaced with FAISS, which offered improved compatibility and performance. To enhance transparency during embedding and indexing operations, progress bars and descriptive status messages were integrated. These additions played a crucial role in monitoring execution progress, particularly during lengthy or resource-intensive tasks.

The chatbot was carefully designed with a strict prompt template to ensure that all responses remained confined to the lecture content. A lightweight retriever and a custom chat loop enabled dynamic interaction, while mechanisms such as chunking and context truncation optimized input length and model efficiency.

Overall, this implementation demonstrates the practical use of large language models and vector databases in academic settings, providing a foundation for scalable and domain-specific question-answering systems. The approach reflects a modern and modular pipeline for building AI-assisted educational tools, combining text processing, embedding, retrieval, and generation in a coherent and effective workflow.