Sri Lanka Institute of Information Technology

Faculty of Computing

Department of Software Engineering

Year 4 Semester 2

SE4010 – Current Trends in Software Engineering

IT21252754 - Madhumalka K. C. S.

Assignment 02 - AI/ML

LLM Development Toolkit: Building a CTSE Lecture Notes Chatbot

2025 May

TABLE OF CONTENTS

1 Introduction

In the last couple of years, the integration of Artificial Intelligence (AI) and Natural Language Processing (NLP) in educational systems has transformed how students interact with study materials. This project focuses on developing a simple chatbot that leverages Large Language Model (LLM) capability to assist students by answering questions based on lecture notes of the "Current Trends in Software Engineering (CTSE)" module. The primary objective is to create an intelligent assistant capable of retrieving and presenting relevant information from course materials in response to user queries and thereby enhance accessibility and understanding of the content.

The whole system is developed in a Jupyter Notebook, making it open and reproducible for educational use. It uses open-source components, including Hugging Face embeddings, the LangChain framework, and FAISS for semantic search, to avoid relying on commercial APIs like OpenAI. The chatbot uses the google/flan-t5-large model from Hugging Face for answer generation, selected for its balance between performance and computational requirements. The chatbot is designed to ingest PDF lecture notes, split them into smaller chunks, convert them into vector representations, and extract the most relevant chunks of information when a user queries it with a question. This approach maintains responses grounded in the original content while demonstrating the potential of AI-powered retrieval-augmented generation (RAG) techniques for educational settings.

2 Justification of LLM Choice

In the development of the CTSE lecture notes chatbot, the selection of an appropriate Large Language Model (LLM) was an important decision that directly impacted on the accessibility, affordability, performance, and alignment of the system with the learning outcomes of the assignment. Rather than opting for commercial LLMs such as OpenAI's GPT-3.5 or GPT-4, which require API keys, internet connectivity, and are charged per usage the project utilized fully open-source models hosted on Hugging Face. This choice allows for a more reproducible, transparent, and adaptable development process, all of which are much appreciated in experimental and academic settings.

For the embedding model, the project uses sentence-transformers/all-MiniLM-L6-v2, a Sentence Transformers library model that is widely used. This model is especially applicable for creating dense vector representations of text for semantic search and information retrieval. MiniLM has a decent trade-off between precision and speed when compared to other larger transformer models. It enjoys high semantic similarity performance at significantly lower computational expense, which is ideally suited for tasks such as matching user queries with portions of lecture notes. Being downloadable through Hugging Face without any API or internet necessity further increases its utility in offline, local, or privacy-limited environments.

As an optional text generator, the model google/ flan-t5 models-large was chosen. flan-t5-large was selected instead of flan-t5-base to provide stronger generation quality while remaining runnable on available hardware. It belongs to Google's FLAN (Fine-tuned Language Net) family and has been fine-tuned to follow instructions across a wide variety of natural language tasks. The model can do more than just text retrieval with it, as it gives coherent, human-like output text that is generated from the content retrieved. FLAN-T5 models are flexible and efficient compared to many larger transformer-based models, while flan-t5-large offers a balance between performance and resource requirements suitable for this project's hardware.

The use of Hugging Face models falls under the spirit of open science and reproducible AI research. By avoiding the use of closed-source APIs and commercial services, students and educators can learn from, modify, and extend the chatbot without cost limitations. In addition, the use of local or hosted open-source models prevents any possible risks of API quotas, latency, or service unavailability making the chatbot readily available for learning and demonstration at all times.

Briefly, Hugging Face was selected as the LLM provider due to its openness, flexibility, affordability, and suitability to embedding-based retrieval and lightweight generative answering. These models are powerful enough to facilitate the intended educational use case while being efficient, free, and easy to integrate in a LangChain-based pipeline.

3 Justification of Development Approach and System Design

To develop the CTSE lecture notes chatbot, a retrieval-based and modular design was utilized with open-source technology such as LangChain, Hugging Face Transformers, and FAISS.

This approach ensures the chatbot can efficiently understand and respond to student queries using lecture materials, all while remaining cost-effective, easy to replicate, and adaptable for academic settings. The whole system is contained within a Jupyter Notebook for convenience and demonstration.

Development Approach

The architecture is built using the Retrieval-Augmented Generation (RAG) architecture, where documents are embedded and stored in a vector database. When a user inputs a query, the system retrieves the most semantically similar chunks of the stored content and runs them through a lightweight language model optionally to generate a fluent response.

Key components of the approach:

i. Document Loading

CTSE lecture notes in PDF format are loaded using LangChain's PyPDFLoader. This avoids the need for text to be manually extracted and supports course materials to be integrated directly.

ii. Text Chunking

The whole text is divided into overlapping chunks by RecursiveCharacterTextSplitter. This helps ensure lengthy papers are being divided into manageable chunks that preserve context, improving semantic matching and retrieval performance.

iii. Semantic Embedding

Each chunk is mapped to a vector via Hugging Face's all-MiniLM-L6-v2 embedding model by means of HuggingFaceEmbeddings. The model offers a satisfactory performance-speed ratio, which is well-suited for running on CPUs or low-end GPU environments.

iv. Vector Storage and Retrieval

The vectors are stored in a FAISS index, which allows for fast similarity search. When a user enters a query, it is embedded and compared against the stored vectors to identify the most relevant chunks of lecture content.

v. Optional Answer Generation

A light, instruction-fine-tuned model (flan-t5-large) is used to transform retrieved information into a natural language answer. This step is powered by Hugging Face's pipeline and integrated into LangChain's RetrievalQA chain using HuggingFacePipeline.

This strategy maintains the system comprehensible, adaptable, and easy to debug or enlarge a set of essential characteristics for educational prototyping and demonstration.

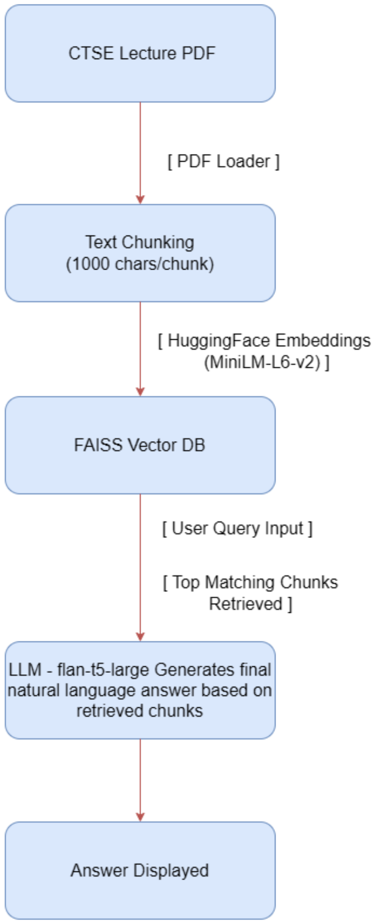

Figure 1. System diagram

4 Challenges and Lessons Learned

During the development of the CTSE lecture notes chatbot using Hugging Face models and LangChain, several challenges were encountered. These challenges provided valuable learning opportunities and helped shape the final solution to be more robust, efficient, and educationally aligned.

- Handling Outdated Libraries and Importing Issues

One of the earliest problems that were faced was that of deprecation warning and wrong use of incompatible modules, specifically of OpenAIEmbeddings. LangChain was recently updated, reorganizing itself with the organization changing from a lot of its older integrations being in common code to a suite of separate packages such as langchain-openai. Having deprecated imports created runtime errors. Changing Hugging Face embeddings fixed the issue along with updating the import paths accordingly and highlights the importance of having the latest libraries along with referencing official resources.

- Understanding AI Tooling Ecosystems

one of the first issues was figuring out how various tools LangChain, Hugging Face, FAISS, and Transformers cooperated with one another. Documentation for these tools is generally written for experienced developers, so initially, it was difficult to determine how things such as vector stores, embedding models, and generative pipelines connected. Getting over this required close reading, tutorial watching, and experimentation coding.

- Designing a Clean and Functional Pipeline

First, the reasoning of the chatbot was coded in poorly structured blocks of code with no clear organization, so it was harder to debug and maintain. Following instruction and exploration, it was clear that partitioning code into nicely structured steps (loading, chunking, embedding, retrieval, answering) not only improved capability but made the system easier to comprehend and document.

- Managing Computational Requirements of flan-t5-large

One of the challenges faced during development was managing the memory and computational demands of the flan-t5-large model. Running this model locally or in environments like Jupyter notebook sometimes led to out-of-memory errors or slower inference times, especially when generating longer responses. This issue required careful adjustment of generation parameters, such as lowering max_new_tokens and testing different batch sizes, to ensure the model could generate answers without exceeding available resources. Through iterative testing and parameter tuning, the chatbot achieved a balance between output quality and computational feasibility, enabling it to run reliably within the constraints of the development environment.

5 Use of GenAI Tools (Prompts and Outputs)

During the development of the CTSE lecture notes chatbot, generative AI tools (mainly ChatGPT) were used for planning, coding, debugging, comparing models, and selecting tools. Below are examples of prompts, actual outputs from the AI, and explanations of their role in the project.

i. **Prompt:** How can I build a chatbot that answers questions based on PDF lecture notes using LangChain and Hugging Face?

Output: You can follow these steps:

Load the PDF using PyPDFLoader.

Split the text into chunks with RecursiveCharacterTextSplitter.

Create embeddings using a Hugging Face model like 'all-MiniLM-L6-v2'.

Store embeddings in a FAISS index.

Use a retriever to fetch relevant chunks.

Pass them into a language model (e.g., flan-t5) to generate an answer.

Implement an interactive loop to receive user questions and return answers.

ii. **Prompt:** Why should I use Hugging Face models instead of OpenAI API for building a RetrievalQA chatbot?

Output: Hugging Face models let you run locally or offline, avoid API usage fees, and avoid rate limits. They're open-source and transparent, making them great for academic projects where reproducibility and cost are concerns. OpenAI APIs are powerful but require internet, API keys, and paid access.

iii. **Prompt:** Why am I getting 'allow_dangerous_deserialization=True' error when loading a FAISS index in LangChain?

Output: This happens because LangChain uses pickle serialization for FAISS indexes. To safely load a saved index, you need to pass allow_dangerous_deserialization=True to FAISS.load_local(). Make sure you only use this flag if the file comes from a trusted source.

iv. **Prompt:** Why do I get ValueError: need at least one array to concatenate in FAISS index?

Output: This happens if you pass an empty list of documents to FAISS.from_documents. Check that your loader and splitter actually returned documents by printing their lengths. If empty, inspect the PDF content or parsing step.

v. **Prompt:** How can I get a complete answer for this / better grounded answers. No need to be exact answers in PDF. I need the best accurate answer.

Output: What you're seeing is a common challenge when using retriever-augmented generation (RAG) with small LLMs like flan-t5-base. The short, paraphrased response you're getting is due to model size limitations and lack of fine-tuning on academic or technical QA.

To generate better grounded and more complete answers, follow these strategies:

Use chain_type="stuff" with RetrievalQA.from_chain_type

By default, LangChain may use a basic prompt format. Switching to "stuff" tells it to stuff all retrieved documents into the prompt before asking the question, which gives the LLM better context. Use a Larger or Better QA Model from Hugging Face flan-t5-base is small and optimized for speed, not depth. For better quality, try a more powerful model. Use flan-t5-large.

vi. **Prompt:** Compare flan-t5-base, flan-t5-large, and flan-t5-xl for use in a RetrievalQA chatbot.

Output: flan-t5-base is light and runs quickly but gives less fluent answers; flan-t5-large improves generation quality with higher memory use; and flan-t5-xl provides the best output but demands high-end GPUs. It noted that flan-t5-large often provides a good balance for projects needing quality answers without needing very high compute resources.

vii. **Prompt:** What is the error and how can I fixed this error.

File D:\opencv\Lib\site-packages\requests\models.py:1024, in Response.raise_for_status(self) 1023 if http_error_msg: -> 1024 raise HTTPError(http_error_msg, response=self) HTTPError: 401 Client Error: Unauthorized for url: https://huggingface.co/google/flan-t5-large/resolve/main/config.json

Output: This means the model google/flan-t5-large you're trying to load requires authentication, and you're not providing a Hugging Face token.

6 Conclusion

This project was successfully able to demonstrate the development of a smart chatbot that is able to reply to questions drawn from the Current Trends in Software Engineering (CTSE) lecture notes using an end-to-end fully open-source and local development stack. Through integration of Hugging Face models with LangChain framework and FAISS vector database, the chatbot is able to retrieve semantically relevant data from course work and optionally return human-readable outputs.

The choice of light models such as all-MiniLM-L6-v2 for embeddings and flan-t5-large for generative answering allowed the system to run on typical hardware without access to commercial APIs. The development process improved some of the most critical software engineering practices such as design of modular architecture, selection of models according to performance requirements, and utilization of retrieval-augmented generation (RAG) techniques.

Through this hands-on learning, the project emphasized the possibility and practicality of building LLM-powered applications for learning. The project also demonstrated the importance of tool explainability, model evaluation, and responsible deployment of GenAI. In the future, the system can be extended with features such as intuitive interfaces, multi-document functionality, or language assistance to provide improved accessibility and effectiveness in learning situations.

Overall, the chatbot provides a useful illustration of how current ML and AI technologies can be used to enable learning and access to information in an organized, comprehensible, and reproducible way.

7 Appendix

- Video Demonstration Link - https://youtu.be/H0QzL2NBW5I
- GitHub Link - https://github.com/Chamaththa123/CTSE-Assignment-02

8 References

[1] Hugging Face, “sentence-transformers/all-MiniLM-L6-v2,” [Online]. Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[2] Google, “FLAN-T5: Instruction-tuned Models,” [Online]. Available: https://huggingface.co/google/flan-t5-large

[3] LangChain, “LangChain Documentation,” [Online]. Available: https://python.langchain.com/

[4] Hugging Face, “Transformers Documentation,” [Online]. Available: https://huggingface.co/docs/transformers