Talk With Your PDF

Project Submitted in Partial Fulfillment of the Requirements for the Digital Egypt Pioneers Initiative DEPI

Presented BY:
1.Mona Abdelazim Abdelgawad
2.Mohamed Bassel Ibrahim
3.Bola Magdy Sydhom
4.Hussien Hamdy Anwer
5.Mohamed Lotfy Alcafory
6.Ahmed Gamal

Supervised By:
ENG. Abdullah El-Shabsi

Abstract

Generative AI plays a crucial role in transforming how we interact with PDF files by enabling users to ask questions and receive intelligent, context-specific answers. Traditionally, extracting relevant information from large or complex documents required time-consuming searches and manual reading.

"Talk with Your PDF" is a project designed to enable interactive, AI-driven conversations with PDF documents. This tool leverages generative AI models to process and understand the content of a PDF file, allowing users to ask questions and receive contextually relevant, human-like responses. The system extracts and organizes key information from the PDF, enabling a seamless interaction where users can inquire about specific sections, topics, or details without manually searching the document. By using a fine-tuned of Llama3, the tool is capable of retrieving the correct answer, enhancing accessibility and efficiency for users working with large or technical documents. This tool is especially beneficial in fields like research, education, and professional services, where understanding detailed documents quickly and accurately is critical.

Table of Contents

1.Introduction

In today's digital landscape, vast amounts of information are often stored in PDF documents, ranging from research papers and technical manuals to legal contracts and educational materials. While these files serve as convenient containers for dense content, manually searching for specific information within them can be a daunting and time-consuming task, especially when dealing with lengthy or complex documents. Traditional methods, such as keyword searches, provide limited assistance, often lacking the ability to interpret context or understand the user's intent behind their queries.

The advent of generative AI offers a transformative solution to this problem. By allowing users to engage in natural language interactions with PDF files, generative AI models can extract, summarize, and present relevant information quickly and intuitively.

The "Talk with Your PDF" project aims to bridge the gap between static documents and dynamic, conversational information retrieval. This system empowers users to ask questions directly to a PDF, generating accurate and context-aware answers, thus simplifying the process of understanding complex documents. It not only improves accessibility but also enhances productivity across a range of industries where document comprehension is essential.

The project Consists of three main modules:

1. Fine tuning Fine-Tuning a Quantized LLaMA 3.1 8B Model on the PubMed Dataset.

2. Building RAG pieline for the Quantized Model
3. Building a Streamlit Application as a user Interface.
4. Implement MLFlow for model training monitoring.

Each of the previous modules is described in details in the next chapters.

2. Document Structure

Chapter 3:Fine tuning Fine-Tuning a Quantized LLaMA 3.1 8B Model on the PubMed Dataset.

Chapter 4: RAG-Based Pipeline for PDF Question-Answering

Chapter 5: Streamlit Application

3.Fine-Tuning a Quantized LLaMA 3.1 8B Model on the PubMed Dataset

This chapter discuss the details of fine-tuning a quantized version of the LLaMA 3.1 8B model on the PubMed dataset. The base model used was unsloth/Meta-Llama-3.1-8B, quantized to 4-bit precision (unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit). Low-Rank Adaptation (LoRA) was employed to optimize resource efficiency during fine-tuning, with LoRA rank and alpha both set to 16. The dataset consisted of 1000 records, and fine-tuning was conducted using the SFTTrainer from the trl and unsloth libraries. Additionally, the report discusses the model's tokenization process, which played a critical role in handling the complex biomedical text from PubMed. The next sections outlines the methods used, challenges faced, and results obtained from the fine-tuning process.

3.1. Introduction

In recent years, large language models (LLMs) have demonstrated exceptional performance across a wide range of natural language processing (NLP) tasks. However, fine-tuning such models is computationally expensive, particularly when working with large datasets and resource-intensive architectures. In this project, I aimed to overcome these challenges by fine-tuning a **quantized** version of the LLaMA 3.1 8B model, which reduces memory requirements and speeds up inference times.

The PubMed dataset was selected as the target corpus due to its relevance to scientific literature, especially in the medical field. By adapting the LLaMA model to the language and nuances of PubMed, the goal was to enhance its ability to

perform specific NLP tasks such as summarization, question-answering, and text classification in the biomedical domain.

3.2. Model and Dataset Overview

3.2.1 Base Model: unsloth/Meta-Llama-3.1-8B

The base model, unsloth/Meta-Llama-3.1-8B, is part of the Meta-LLaMA family and offers a versatile architecture for various NLP tasks. The 8B parameter model was chosen for its balance between size and performance. However, to address resource constraints, this model was quantized.

3.2.2 Quantization: unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit

Quantization is a technique used to reduce the precision of a model's weights, in this case, from 16-bit floating point (FP16) to 4-bit precision, effectively reducing memory usage while retaining a high level of performance. The bnb-4bit quantization method was employed, which ensures faster inference with a smaller model footprint.

3.2.3 Dataset: PubMed

The dataset used for fine-tuning was a subset of the PubMed dataset, consisting of **1000 records**. PubMed provides access to a vast array of medical and scientific literature, making it an ideal candidate for fine-tuning models in the biomedical domain. Each record consisted of abstracts and full-text articles containing complex medical terminology and syntax.

The PubMed dataset, consisting of sections like **context**, **background**, and **results**, provided depth and richness to the input data. This structured content helped enhance the model's ability to generate comprehensive and nuanced results, especially in the

biomedical domain, where such contextual information is critical for understanding scientific literature. But also, this required more data preparation and preprocessing.

3.3. Tokenization and Tokenizer Setup

3.3.1 Tokenizer Overview

The tokenizer plays a critical role in transforming the raw text from the PubMed dataset into tokens that the LLaMA model can understand. For this project, the **LLaMA tokenizer** was used, which is a subword-based tokenizer designed specifically for the LLaMA architecture. It handles tokenizing text into smaller subwords or characters to ensure the model can process even rare or complex words commonly found in the biomedical literature.

The tokenizer is pretrained with a vocabulary that captures a wide range of subwords and common patterns in the input text, allowing it to split biomedical terms like "eosinophilia" into manageable tokens. This reduces the likelihood of out-of-vocabulary (OOV) issues, which can negatively affect model performance, especially in niche fields like medicine.

3.3.2 Tokenization Process

The tokenization process used a **max sequence length of 2048 tokens**, allowing the model to handle longer inputs, which is especially important for processing the full-text articles and detailed abstracts found in the PubMed dataset. The tokenizer splits the text into manageable subwords, ensuring even complex biomedical terms are appropriately handled.

3.4. Fine-Tuning Process

3.4.1 Low-Rank Adaptation (LoRA)

To further optimize the fine-tuning process, I implemented **Low-Rank Adaptation (LoRA)**. LoRA is an efficient method that reduces the number of trainable parameters by projecting model updates into lower-rank matrices, which minimizes memory and computational load.

- **LoRA Rank**: 16
- **LoRA Alpha**: 16

These parameters allowed the model to adapt to the domain-specific language of the PubMed dataset without overfitting or consuming excessive resources.

3.4.2 Fine-Tuning Setup

The fine-tuning process was managed using the SFTTrainer from the trl and unsloth libraries.

Key Training Parameters

- **Batch Size**: 2 (per device).
- **Gradient Accumulation**: 4 steps.
- **Learning Rate**: 2e-4.
- **Number of Epochs**: 1 (to prevent overfitting on the relatively small dataset).
- **Optimizer**: adamw_8bit (optimized for memory efficiency).
- **Precision**: Mixed precision was enabled, using **FP16** or **BF16**, depending on hardware support.

To ensure reproducibility, a fixed random seed (3407) was set.

3.5. Results and Evaluation

3.5.1 Evaluation Metrics

Due to the fine-tuning process focusing on a limited subset of the dataset, traditional evaluation metrics like loss and perplexity were monitored. The **logging frequency** was set to log after each training step, providing insights into model performance at regular intervals. Key metrics include:

- **Training Loss**: The loss decreased steadily, indicating that the model was learning and adapting to the dataset.
- **Validation Performance**: Although not extensively validated, preliminary tests showed promising results for text generation tasks in the biomedical domain.

3.5.2 Impact of Tokenization

The tokenization process proved effective in handling the biomedical jargon present in PubMed records. By splitting rare and complex terms into subwords, the model was able to process these terms effectively without encountering out-of-vocabulary issues. The 2048-token sequence length ensured that both abstracts and full-text articles could be represented in a manageable size for training.

3.5.3 Impact of Quantization and LoRA

The quantization technique significantly reduced the model's memory consumption, allowing it to be fine-tuned and run on less powerful hardware. Additionally, the use of LoRA ensured that the fine-tuning process was efficient and resource-friendly, while still achieving good performance on the limited dataset.

3.6. Challenges and Solutions

3.6.1 Random Initialization Issue

During the implementation, an issue arose due to the random initialization of the quantized weights, which required additional adjustments. This was mitigated by adjusting the learning rate and employing LoRA to handle the complexity of the fine-tuning process.

3.6.2 Mixed Precision Training

One of the challenges faced was the hardware support for mixed precision training (FP16 vs BF16). This was addressed by detecting the hardware capabilities and adapting the fine-tuning process accordingly.

3.7. Conclusion

Fine-tuning the LLaMA 3.1 8B model on the PubMed dataset was successful, with quantization and LoRA techniques proving invaluable in managing the resource constraints. The tokenization process was essential for handling complex biomedical language, and the quantized model showed good performance despite the challenges of working with a relatively small dataset.

The fine-tuned model can now be applied to various downstream tasks in the biomedical field, including question-answering, summarization, and classification of medical literature.

4.RAG-Based Pipeline for PDF Question-Answering

This chapter discuss the details of the Retrieval-Augmented Generation (RAG) pipeline implemented to enable answering user questions based on content extracted from a PDF document. The pipeline includes processing a PDF file, extracting its content, tokenizing documents, building vector store for the documents, retrieving relevant documents, preparing the prompt and then generates a response using the Llama quantized model. Below is a detailed explanation of the pipeline components:

4.1 RAG_Chain Function :

The RAG_Chain function performs the main task of PDF-based question-answering. It takes the Pdf_file uploaded by the user and the question entered by the user then returns the answer to be displayed in the UI.

Here's how it works step by step:

A) PDF Processing:

The function uses PyPDF2 to read the contents of the PDF file, extracting all the text and concatenating it into a single document string.

B)Document Splitting :

To handle large documents, the **RecursiveCharacterTextSplitter** is used to split the document into smaller chunks (default size of 1000 characters with an overlap of 200 characters between chunks).

C)Embedding and Vector Store Creation :

After splitting the text, **HuggingFaceEmbeddings** is employed to create embeddings for each chunk. These embeddings are stored using Chroma , which

allows for fast retrieval of relevant chunks based on semantic similarity to the question.

D)Retrieving Relevant Documents :

The Chroma vector store is then queried with the user's question to retrieve the top 4 most relevant document chunks. These are concatenated to form the context for the language model to generate an answer.

E)Prompt Preparation :

The retrieved chunks are combined into a formatted prompt that includes the question and the relevant context.

F)Response Generation :

The script utilizes a Llama model from the local system to generate an answer based on the prepared context and question. The model is initialized with custom parameters for context length, temperature, and CPU threads.

The function returns the generated answer from the Llama model based on the retrieved document chunks.

4.2 Libraries and Models :

The script begins by importing several key libraries for document processing, text embedding, and model inference:

1. psutil : System monitoring (not used in the main function but could be useful for performance tracking).
2. PyPDF2 : Used for extracting text from PDF files.
3. transformers : Hugging Face library for NLP, used for tokenization and model loading.

4. Llama Model (llama_cpp) : Used for loading and utilizing a pre-trained Llama model.
5. LangChain Components :
6. `RecursiveCharacterTextSplitter`: Splits large documents into manageable chunks.
7. `Chroma` and `HuggingFaceEmbeddings`: Used to create a vector store for efficient retrieval of relevant document sections.

4.3. Key Components in the Function :

4.3.1 Model Selection :

The script uses a Llama model in GGUF format , indicating that a highly compressed and optimized version of the model is used. This is suitable for large-scale, resource-efficient inference.

4.3.2 Efficient Retrieval :

By employing the Chroma vector store and HuggingFaceEmbeddings , the system is able to retrieve the most relevant portions of the document efficiently, which boosts both accuracy and performance of the overall question-answering system.

4.3.3 Context-Aware Generation :

Using a robust model like Llama , which is known for handling long-form content, enhances the accuracy of the responses generated based on the given context.

4.4. Potential Enhancements :

4.4.1 Error Handling :

The script already includes a check for empty documents; however, further exception handling could be implemented for potential errors in model loading, embedding creation, or retrieval failures.

4.4.2 System Resource Management :

Using psutil to monitor CPU and memory usage during processing would provide better insights into performance and potential bottlenecks, especially when handling large PDFs or deploying on resource-constrained systems.

4.4.3 GPU Acceleration :

Firstly, the pipeline used the CPU for model inference. After that, multiple GPU layers were specified to speed up the inference.

4.5Model monitoring using MLFlow

MLflow, at its core, provides a suite of tools aimed at simplifying the ML workflow. It is tailored to assist ML practitioners throughout the various stages of ML development and deployment.

This project implemented MLFlow code to monitor system resource usage and log key performance metrics when processing a question using a retrieval-augmented generation (RAG) model. Here's a breakdown:

1)monitor_resources(): Uses the psutil library to monitor CPU and memory usage. Measures CPU usage over a 1-second interval and memory usage in megabytes (MB). Returns both CPU and memory usage values.

2)get_important_facts(question): The function accepts a question input and tracks the resource usage while processing it. It measures CPU and memory usage

both before and after executing a model (rag_chain) to get the answer. It uses MLflow to log various metrics:

- The input question.
- Time taken to generate the answer (response_time).
- Differences in CPU (cpu_diff) and memory (mem_diff) usage before and after running the model.

The resource usage and response time are printed for review, and the final answer is returned.

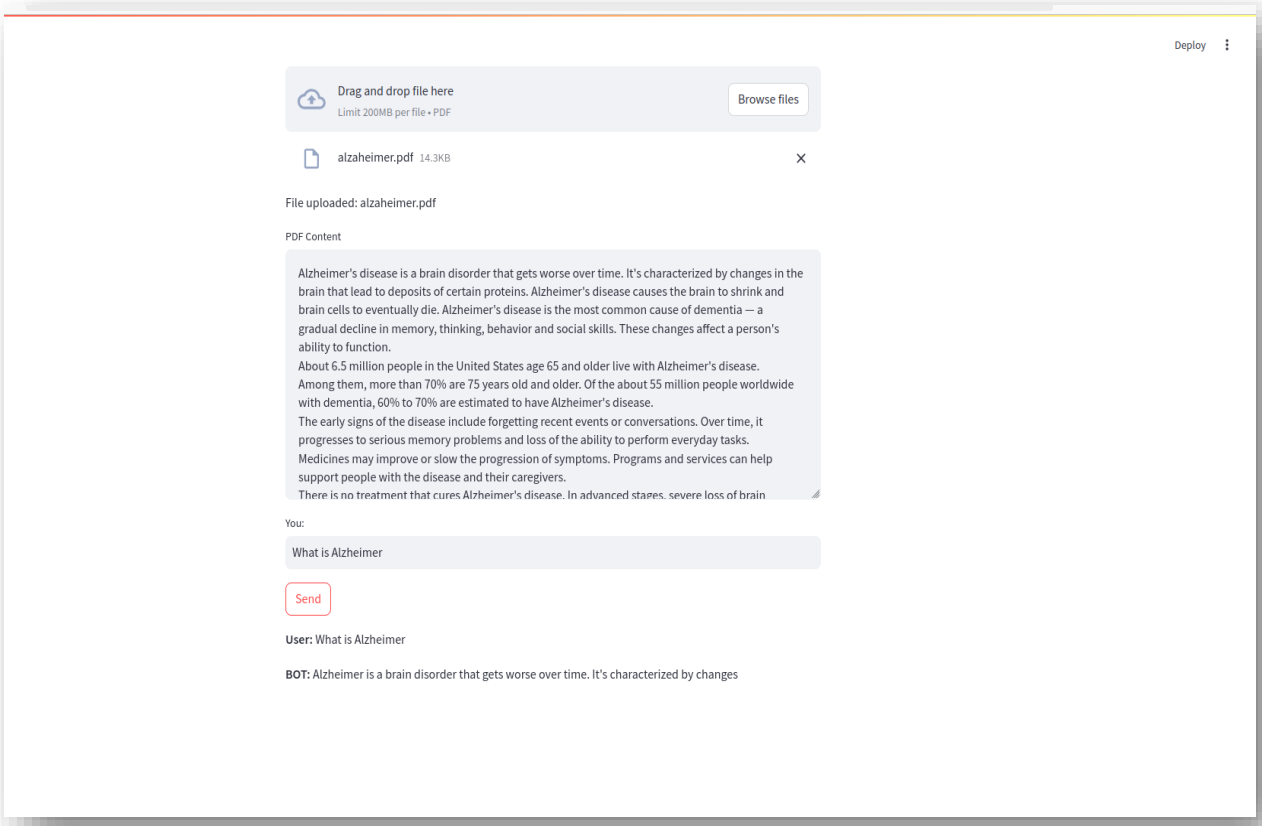
This function is designed for performance monitoring and optimization during AI model execution, especially useful in research or production environments where resource consumption matters.

4.5 Conclusion :

This script provides a powerful and efficient method for answering questions based on the content of PDF documents. By leveraging HuggingFace's embedding models , Chroma vector storage , and the Llama language model, it combines retrieval-augmented generation with modern NLP tools. This is ideal for applications where users need detailed and contextually accurate answers derived from long or complex documents.

5.Streamlit Application

The chapter describes the details of (app.py) which provides an interactive web application that allows users to upload a PDF file, ask questions about its contents, and receive contextually relevant answers. This is achieved using Streamlit for the user interface and a Retrieval-Augmented Generation (RAG) Chain for extracting and processing PDF data.



The project implements a web-based interface for users to upload a PDF document and interact with it by asking questions. The core functionalities include document uploading, content display, and a question-answering mechanism based on the RAG Chain using the previously defined function (RAG_Chain).

5.1. Libraries and Dependencies :

The script imports several essential libraries and modules for different tasks:

1. Streamlit : The primary library for creating the web interface.
2. PyPDF2 : Used to read and extract text from PDF files.
3. RAG_Chain from RAG_GGUF : A custom function (previously described) to handle the question-answering task based on PDF content.

5.2. Application Structure :

5.2.1 PDF Upload and Display :

A) File Upload :

Users can upload a PDF file through Streamlit's `file_uploader` widget. The uploaded file is then passed to the `read_pdf()` function, which uses PyPDF2 to extract text from each page of the document.

B) Content Display :

The extracted PDF content is displayed in a `text_area` widget, which provides a scrollable window to view the document content. If no file is uploaded, the text area displays a message prompting the user to upload a file.

5.2.2 User Input and Interaction :

A) Text Input :

Users can type their questions into a `text_input` field labeled "You:". This acts as the user's message, which will be processed to generate a response.

B) Chat History :

A session state variable `st.session_state.chat_history` is used to maintain a persistent chat history throughout the session. This allows users to view the

conversation as it unfolds, retaining both the user's questions and the chatbot's responses.

5.2.3 Handling User Queries :

A) RAG Chain Integration :

When the user clicks the Send button, the script invokes the `RAG_Chain()` function, which processes the uploaded PDF and the user's question to generate a relevant response. The generated response is then added to the chat history.

B) Chat Display :

The chat history is displayed as markdown text, with distinct formatting for the user's input and the bot's responses.

5.3. User Interaction Workflow :

1. Uploading a PDF :The user is prompted to upload a PDF file. Once uploaded, the file is read, and its content is displayed in the text area for reference.

2. Asking Questions :The user enters a question in the text input box labeled "You:".

3. Receiving Responses :Upon clicking the Send button, the RAG chain processes the PDF content to retrieve relevant information and provides a response. The user's input and the generated response are both appended to the chat history and displayed in the application.

4. Chat History :The conversation history is persistent, allowing the user to see previous exchanges during the session.

5. Key Features :

1. Interactive PDF Chat : The app offers an intuitive way for users to upload a PDF and interact with it via questions, receiving accurate answers in real time.

2. Persistent Chat History : The session-based history ensures users can view the entire conversation during the interaction.
3. Streamlit Integration : The script uses Streamlit's widgets and state management for a seamless user experience.

5.4 Conclusion :

This chapter discuss the details of a user-friendly application for interacting with PDFs through a question-answering interface, leveraging a Retrieval-Augmented Generation Chain . The use of Streamlit ensures ease of deployment, making it a practical tool for users who need to extract specific information from lengthy documents without manually searching through the text.