KUET INFORMATION LLM

By

Omar Bin Sariffuzzaman
Roll: 1907026

&

MST Sadia Islam
Roll: 1907027

Supervisor:

Md. Shahidul Salim
Lecturer
Dept. of Computer Science and Engineering
Khulna University of Engineering & Technology (KUET)

Signature

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

December 2023

Acknowledgements

I express my heartfelt gratitude to Md. Shahidul Salim , Lecturer of the Department of Computer Science and Engineering and the esteemed supervisor of this project. His unwavering expertise and guidance have been the cornerstone of my journey in developing the "KUET INFORMATION LLM." His profound knowledge, tireless support, continuous encouragement, and invaluable suggestions have propelled this project to its current stage. His constant supervision and constructive criticism have been instrumental in shaping the project's direction. I am indebted to the enduring support and scholarly mentorship provided by Md. Shahidul Salim. Without his enthusiastic motivation and unwavering encouragement, this endeavor would not have reached fruition. Additionally, I am thankful to Allah for granting me the talents and abilities that enabled me to undertake and complete this project.

Omar Bin Sariffuzzaman

Roll: 1907026

&

MST Sadia Islam

Roll: 1907027

Abstract

In this work, we present a sophisticated fine-tuned LM system customized for Khulna University of Engineering and Technology (KUET). Employing advanced language models, specifically a Large Language Model (LLM), our fine-tuned LM is carefully trained on a diverse set of information to ensure accurate and context-aware responses. The seamless integration of Langchain and RAG (Retrieval-Augmented Generation) techniques further enhances the system's ability to comprehend and respond to user queries effectively. The system includes a robust PDF analyzer that employs RAG to extract relevant information from uploaded PDF documents. This feature extends the fine-tuned LM's utility beyond pre-existing data, allowing users to pose inquiries based on uploaded documents. The integration of Google Colab and Gradio streamlines the deployment and accessibility of the system, ensuring a user-friendly interface. Our solution not only addresses specific queries but also accommodates dynamic and evolving information. The collaboration between the language model and the PDF analyzer creates a versatile platform, catering to the diverse informational needs of KUET stakeholders. The collaborative efforts of these components empower the fine-tuned LM to provide subtle, accurate, and timely responses, enhancing user experience and information retrieval efficiency. This comprehensive system stands as a testament to the potential of integrating cutting-edge language models with innovative techniques, showcasing the adaptability and intelligence required for an effective university-oriented fine-tuned LM.

Contents

List of Tables

List of Figures

1 Introduction

In the ever-evolving landscape of technology, natural language processing and understanding have become pivotal in enhancing human-computer interaction. Acknowledging the significance of language-related applications in higher education, we are pleased to present a fine-tuned Language Model, a project created especially to meet the linguistic requirements of Khulna University of Engineering and Technology.

1.1 Background

In the realm of artificial intelligence and natural language processing, the development of intelligent fine-tuned Language Model has gained significant momentum. Recognizing the need for an advanced conversational interface tailored for Khulna University of Engineering and Technology (KUET), this project embarked on the creation of a fine-tuned Language Model. With the objective of providing a sophisticated solution while being mindful of resource constraints, the project aimed to strike a balance between model complexity and computational resources.

1.1.1 PROBLEM STATEMENT

Fine-tuning a pre-trained Large Language Model (LLM) for a Question-Answering (QA) conversational interface involves adapting the model to understand and respond to user queries in a conversational manner. Our main motive is to implement the Language Model with low resources.

1.2 Objectives

The objective of creating a fine-tuned LM can encompass various goals tailored to the needs of students and others who are interested. Here are several key objectives associated with the project :

- To design and implement a fine-tuned LM specifically tailored for Khulna University of Engineering and Technology (KUET) to address the unique informational needs of its users.

- Utilizing Large Language Models (LLM) to enhance the fine-tuned LM's understanding of user queries and improve the quality of responses.
- Integration of advanced techniques like Langchain and Retrieval-Augmented Generation (RAG) to improve the fine-tuned LM's comprehension and response capabilities.
- To develop a robust PDF analyzer using RAG to extract relevant information from uploaded PDF documents, expanding the fine-tuned LM's utility beyond pre-existing data.
- For Streamline deployment using accessible resources like Google Colab and Gradio and Chainlit, ensuring efficient usage of computational resources and a user-friendly interface.
- To optimize the use of computational resources, particularly focusing on CPU usage for the PDF model, to ensure the system's functionality within resource constraints.
- To showcase the adaptability and intelligence of the fine-tuned LM system through the integration of cutting-edge language models and innovative techniques, emphasizing its effectiveness for university-oriented tasks.

1.3 Scope

The scope of our project, "KUET INFORMATION LLM," is broad and encompasses various aspects related to the development and deployment of a sophisticated fine-tuned LM system for Khulna University of Engineering and Technology (KUET) and PDF analyzer. Develop a fine-tuned LM specifically tailored to address the unique informational needs and queries of KUET stakeholders, including students, faculty, and staff. Utilize advanced Large Language Models (LLM) and Implement Langchain and Retrieval-Augmented Generation (RAG) techniques . Optimize the utilization of computational resources, with a particular emphasis on CPU usage for the PDF model, to ensure the system's functionality within resource constraints. Consider the potential for expanding the fine-tuned LM system to other universities with similar needs, making the solution scalable and adaptable for broader use.

1.4 Unfamiliarity of the problem

Despite the increasing popularity of fine-tuned LM systems in various domains, the development of a sophisticated, LLM trained customized fine-tuned LM for any particular

organization with confidential data is relatively unexplored. The actual problem or concern of using LLM or train LLM model is resource limitation like high resolution GPU and RAM. So our main target was to run LLM trained model in low resource devices with low GPU. The inclusion of a robust PDF analyzer using RAG is a novel aspect. This feature allows the fine-tuned LM to extract relevant information from uploaded PDF documents, extending its utility beyond pre-existing data. This capability is particularly unique in the university setting, where students and faculty often deal with academic documents in PDF format. The incorporation of Langchain and Retrieval-Augmented Generation (RAG) techniques sets this project apart. While many fine-tuned LM projects focus solely on the immediate requirements of a specific institution, this project envisions a solution that can be extended to benefit a broader educational community.

1.5 Project planning

Our project plan involves several stages, including research, data collection, prompt creation, model selection, fine tuning model, testing, user interface creation. In fig 1.1 a Gantt chart representation is included, which shows the timeline from research and requirements gathering to deployment and completion. The schedule is a rough estimate that shows the time & efforts spend on each stage.

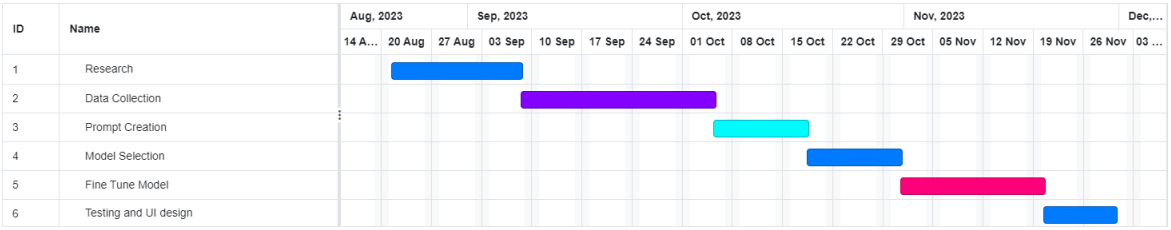

Fig 1.1 - Gantt Chart for project planning timeline

2 Related Works

2.1 Existing Solutions

Some relevant work details and contribution are added in the given table .

Table 2.1 - Existing Solutions

Project Title	Links
Llama2-Medical-Chatmodel	https://github.com/AIAnytime/Llama2-Medical-Chatmodel
CSVMODEL	https://github.com/wg-dev35/csvmodel
CRCIKET-AMA	https://github.com/Ankush-Chander/cricket-ama

These ideas are implemented on medical information, about cricket and one is from csv file. In our system we have fine-tuned the model by data based on a university. So it is a university based fine-tuned Language Model.

3 System Design

System design involves creating a comprehensive blueprint for the system. It encompasses defining the system architecture, specifying components , modules , interfaces , data of the system .

3.1 Analysis of the system

KUET Information Fine-tuned LM is a model to answer queries based on the provided information in this case information about KUET. A user can ask queries about KUET and chat with the LM. In PDF analyzer fine-tuned LM, user can upload any pdf and ask queries from that pdf to get intelligent answers.

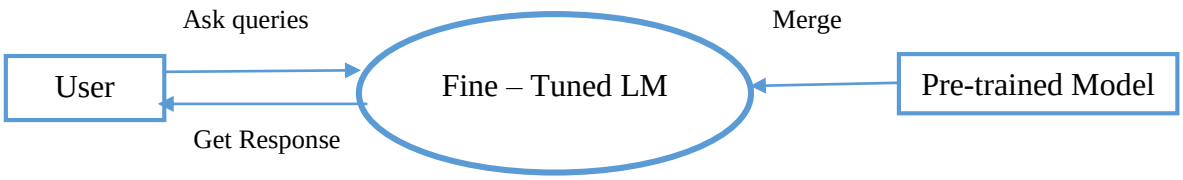

Fig 3.1.1 - Abstract Flow Diagram

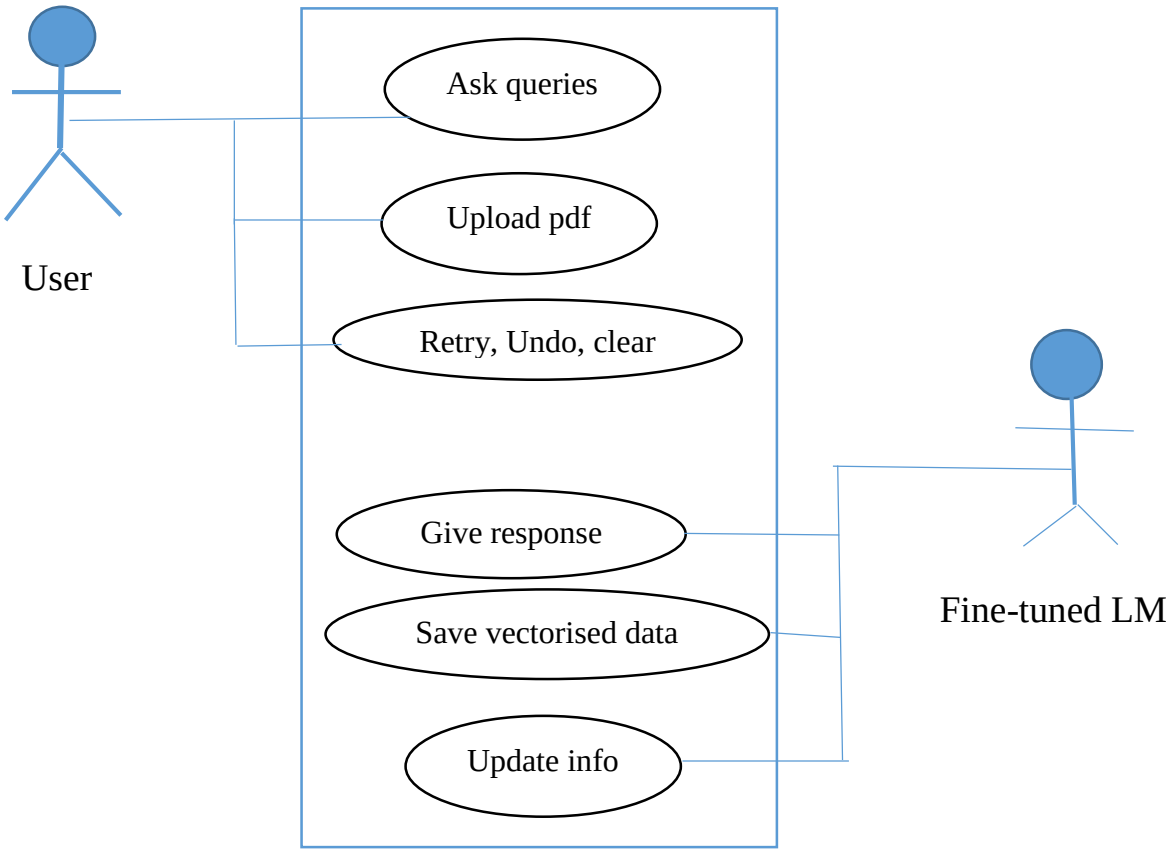

Fig 3.1.2- USE-CASE Diagram of fine-tuned LM

3.2 System architecture

System architecture is the high level structure that defines how a system is organized and how it's components interact.

3.2.1 Fine-tuned LM architecture: This system architecture is based on some layers such as Data Collection, Prompt creation, Tuning the selected model with collected data, training the model , testing , UI interface design through gradio and final implementation.

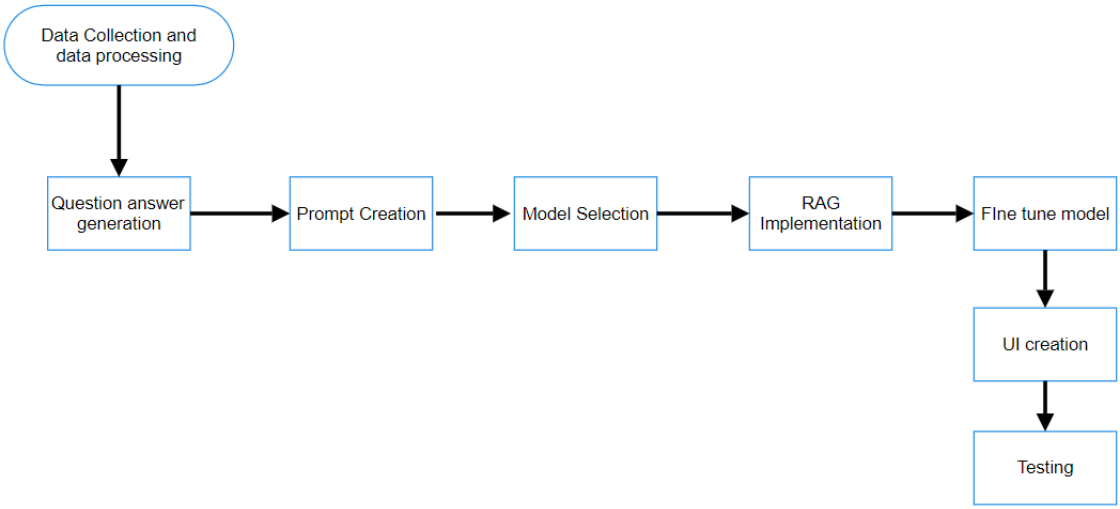

Fig 3.2.1- System Architecture

3.2.2 PDF Analyzer Model Architecture: Here we used a llama-2-7b -chat-GGML model to develop the system. Langchain preprocesses the extracted data, splits the data into chunk, embedding model, store vectorized data and finally the LLM creates the response.

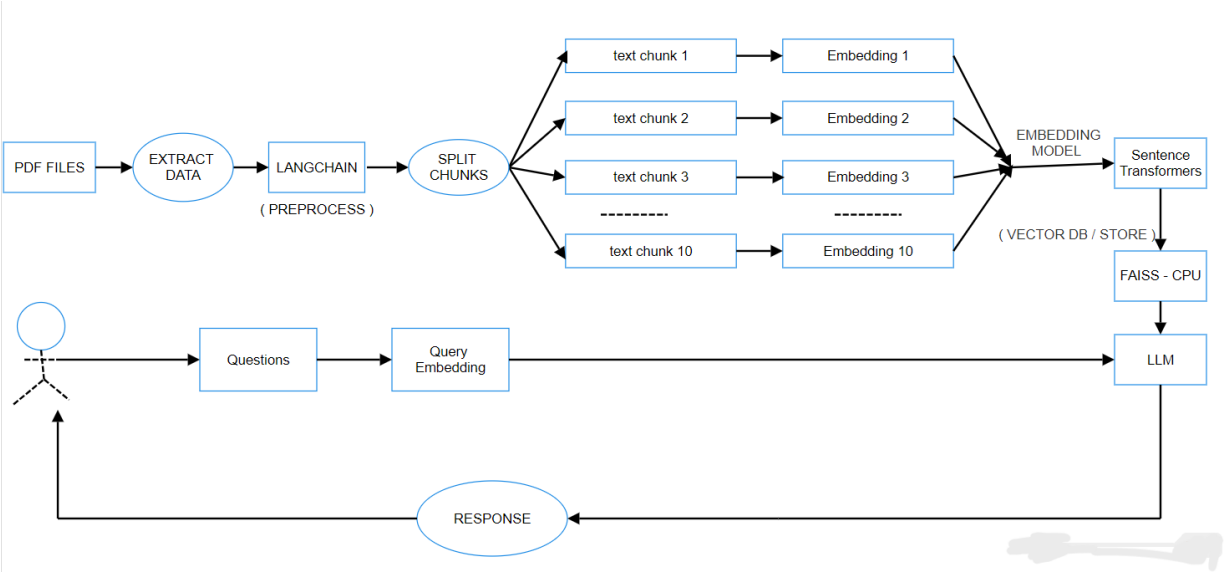

Fig 3.2.2- PDF Analyzer Model Architecture

3.2.3 Transformer Model Architecture: In order to develop pre-trained LLM based fine-tuned LM we used a llama-2-7b-chast-hf model which is a type of transformers model. Key components of Transformer models are:

1. Self-Attention Mechanism:

- The core innovation of the Transformer is the self-attention mechanism.
- Self-attention allows the model to weigh different words in a sequence differently, capturing dependencies regardless of their distance.
- It calculates attention scores by considering all words in the input sequence simultaneously.

2. Encoder-Decoder Structure:

- The Transformer architecture consists of an encoder and a decoder, modelh built using layers of self-attention and feedforward neural networks.
- The encoder processes the input sequence, while the decoder generates the output sequence.

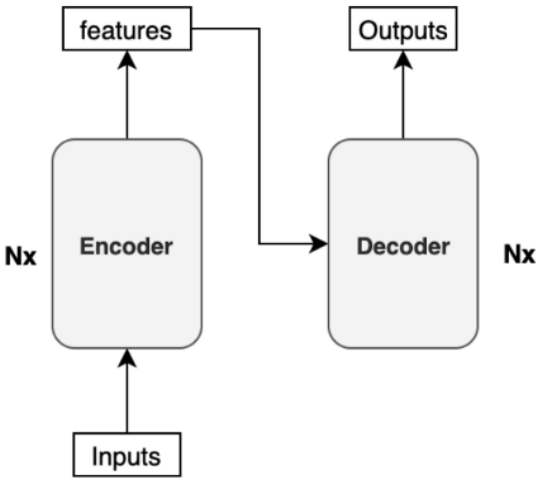

Fig 3.2.3- Encoder Decoder Techniques

3. Positional Encoding:

- Since self-attention doesn't inherently consider the order of elements in a sequence, positional encoding is added to provide information about the positions of words.

- Positional encodings are added to the input embeddings to give the model information about the relative or absolute position of each token.

4. Multi-Head Attention:

- To enhance the expressive power of the model, self-attention is applied in parallel multiple times, each with different learned linear projections.
- The outputs from multiple attention heads are concatenated and linearly transformed.

5. Feedforward Neural Networks:

- After self-attention, each position in the sequence is passed through a feedforward neural network independently.
- The feedforward network consists of fully connected layers and a non-linear activation function (e.g., ReLU).

6. Layer Normalization and Residual Connections:

- Each sub-layer in the model (e.g., self-attention, feedforward) is followed by layer normalization and a residual connection.
- These components help with the training stability of deep networks.

7. Encoder and Decoder Stacks:

- The encoder and decoder are composed of multiple identical layers stacked on top of each other.
- Each layer independently processes the input and contributes to the overall transformation.

8. Masking in Decoder:

- During training, the decoder is provided with the entire target sequence up to the current position but masks future positions to prevent information leakage.

9. Final Linear and Softmax Layer:

- The decoder's output is processed through a final linear layer, followed by a softmax activation to produce probabilities for each element in the output vocabulary.

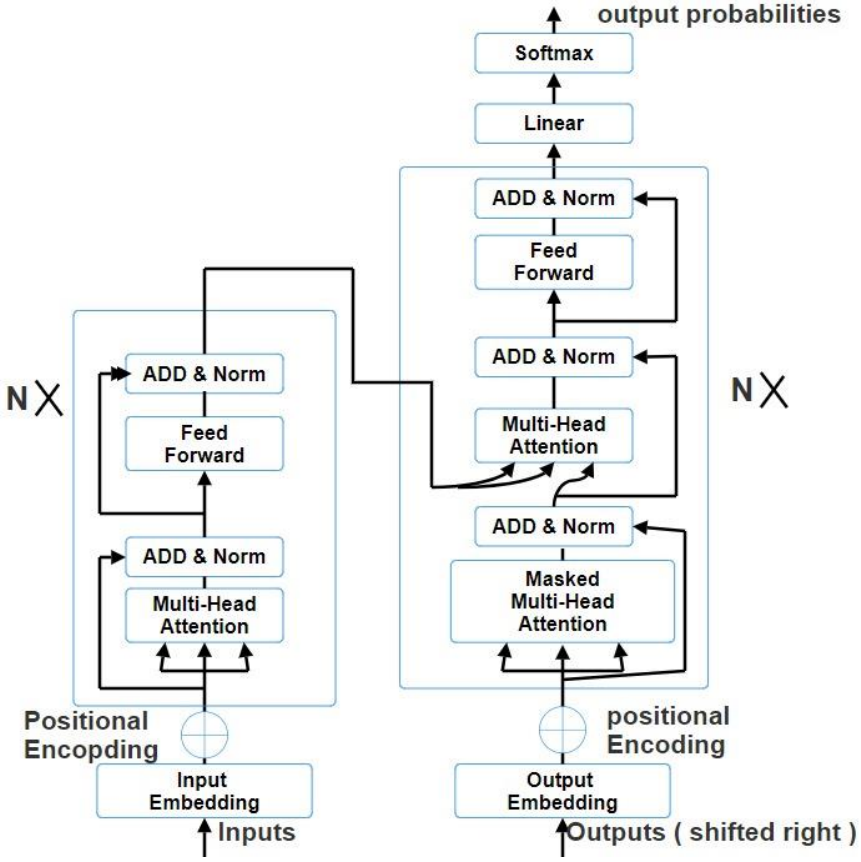

Fig 3.2.4 -Transformer model Architecture

3.3 Tools used

To implement the fine-tuned LM system we have used Google Colab and Gradio. To implement PDF analyzer we have used Visual Studio Code. For fine-tuned LM implementation, we used google colab as colab provides free access to GPU resources, which is crucial for training deep learning models and used gradio because gradio is model-agnostic and can be used with various machine learning frameworks. For PDF analyzer we used Visual Studio as it supports python and gives flexibility.

3.3.1 Google Colab

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that allows users to write and execute Python code in a collaborative and interactive environment. Here are some key features and reasons why we choose to use Google Colab:

- **Free Access to GPU Resources:**
  - One of the main attractions of Google Colab is that it provides free access to Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). This is particularly beneficial for training machine learning models specially LLMs, as these hardware accelerators significantly speed up the training process.
- **Cloud-Based Environment:**
  - Google Colab is entirely cloud-based, eliminating the need for users to set up and configure our own development environments locally. This allows for easy accessibility from any device with an internet connection.
- **Integration with Google Drive:**
  - Colab is integrated with Google Drive, enabling seamless sharing and collaboration on Jupyter notebooks. Notebooks and their associated data can be stored in Google Drive, and changes are automatically saved.
- **Pre-installed Libraries:**
  - Colab comes with a variety of pre-installed libraries commonly used in machine learning and data science, such as TensorFlow, PyTorch, scikit-learn, and more. This eliminates the need for us to manually install these libraries.
- **Easy Sharing and Collaboration:**
  - We can share our Colab notebooks just like we would with Google Docs or Sheets. This makes it easy for multiple users to collaborate in real-time on a project, fostering a collaborative development environment.
- **Support for Data Science and Machine Learning:**
  - Colab is well-suited for data exploration, analysis, and machine learning tasks. It supports the integration of data visualization libraries, making it a powerful tool for exploratory data analysis.

- **Version Control:**
  - Colab integrates with Git, allowing users to version control their notebooks. This is essential for tracking changes, collaborating with others and maintaining a history of the development process.
- **Educational Use:**
  - Colab is widely used in educational settings for teaching and learning Python, data science, and machine learning. Its accessibility, ease of use, and collaboration features make it an ideal platform for educational purposes.
- **Flexible Hardware Configurations:**
  - We can choose between CPU, GPU or TPU configurations depending on our computational requirements.
- **Community and Documentation:**
  - Colab has an active community, and there are plenty of tutorials and documentation available. This makes it easier for users, especially beginners, to get started and find solutions to common problems.

3.3.2 GRADIO

Gradio is an open-source Python library that simplifies the process of creating user interfaces for machine learning models. It allows developers and data scientists to easily generate interactive and shareable UIs for their models with minimal code. Gradio is model-agnostic, meaning it can be used with various machine learning frameworks, including TensorFlow, PyTorch, scikit-learn, and more.

Key features of Gradio include:

- **Simple Interface Creation:** Gradio provides a high-level and intuitive API for creating interfaces. With just a few lines of code, we can create UIs for tasks like image classification, text generation, and style transfer.
- **Wide Model Support:** Gradio supports a variety of model types, from image classification and text generation to custom tasks. This makes it versatile and applicable to a broad range of machine learning projects.

- **Interactive User Interfaces:** Gradio generates interactive UIs that allow users to input data and see model predictions in real-time. This is particularly useful for showcasing models to non-technical users or for interactive exploration of model behavior.
- **Customization:** While Gradio simplifies the UI creation process, it also provides options for customization. Users can define input and output components, style the interface, and handle user interactions programmatically.

Why use Gradio along with Google Colab:

- **Interactive Model Showcase:** While Google Colab is an excellent platform for developing and training machine learning models, it might not be the most user-friendly way to showcase models to non-technical stakeholders. Gradio allows us to quickly create interactive interfaces that make it easy for others to understand and interact with our models.
- **Real-Time Feedback:** Gradio's interactive UIs provide real-time feedback on model predictions. This can be beneficial for model debugging, testing, and understanding how the model performs on different inputs.
- **User-Friendly Demonstration:** Gradio simplifies the process of creating user-friendly interfaces, making it more accessible for users who may not be familiar with coding or machine learning concepts.
- **Collaboration and Communication:** Combining Google Colab for collaborative model development and training with Gradio for creating interactive interfaces enhances the overall communication and collaboration aspects of machine learning projects, especially when working with interdisciplinary teams.

3.3.3 Visual Studio

The advantages of using Visual Studio for creating a PDF analyzer fine-tuned LM based on a pre-trained Large Language Model (LLM) is given below :

- **Integrated Development Environment (IDE):**

- **Code Editor:** Visual Studio provides a powerful code editor with features like syntax highlighting, IntelliSense, and code navigation, making it easier to write and manage your code.
- **Debugging Tools:** Visual Studio includes robust debugging tools, which can be crucial when developing and fine-tuning our fine-tuned LM.

❖ **Support for Multiple Languages:**

- Visual Studio supports multiple programming languages, including Python, which is commonly used for natural language processing and machine learning tasks.

❖ **LLM Model Training:**

- **TensorFlow and PyTorch Support:** Visual Studio supports popular machine learning frameworks like TensorFlow and PyTorch, allowing us to train and deploy LLM models seamlessly.

❖ **Collaboration and Team Development:**

- Visual Studio integrates with version control systems like Git, making it easier for teams to collaborate on the development of the fine-tuned LM. We can manage our codebase, track changes, and collaborate with team members effectively.

❖ **User Interface Development:**

- As our fine-tuned LM involves a user interface, Visual Studio offers tools for building graphical user interfaces (GUIs) that can enhance the user experience.

❖ **Project Management:**

- Visual Studio includes project management tools that can help us organize and structure our codebase, making it easier to manage a complex project like a PDF analyzer fine-tuned LM.

❖ **Community and Support:**

- Visual Studio has a large and active developer community. If we encounter challenges during development, we can benefit from community support, forums, and documentation.

4 Project Implementation

Project implementation is the stage in a project's lifecycle where the planned activities are executed and the project plan is put into action.

4.1 System implementation

System implementation is the phase where design system is actually constructed and put into operation.

4.1.1 Fine-tuned LM implement:

I) DATA COLLECTION AND FORMAT: At first we collected information about KUET. We want to develop a model as a question answering conversational interface. When it comes to training a Question-Answering (QA) model, the format of the data is often in a question-and-answer style. The model learns to understand the context of a passage and generate relevant answers to questions posed about that passage. This format is useful for tasks where the model needs to comprehend and respond to specific queries about a given text. So we formatted the collected data in QA style. The reason for using a QA format in training is to teach the model to understand the relationship between questions and answers and to extract relevant information from a given context. It helps the model learn to associate specific questions with appropriate responses.

Next we format the data as a prompt which involves providing input or instructions to a language model in a structured manner, guiding the model to generate a specific response or perform a particular task. The exact format may vary depending on the nature of the task and the capabilities of the language model. We learned here about two types of prompt. They are-

Zero-shot prompt: A zero-shot learning prompt typically consists of a task description or instruction, followed by input that the model is expected to process or generate a response for. The key feature is that the model is not specifically trained on the task mentioned in the prompt.

Example:

Task: Summarize the following article.

Input: [The content of the article]

In this example, the task "Summarize the following article" serves as the prompt. The model has not been trained on summarization tasks explicitly, but it is expected to generate a coherent summary based on the provided article.

React-prompts: A react prompt involves providing a context or input, often in the form of a statement or scenario, and instructing the model to "react" to it by generating an appropriate response.

Example:

Input: "You receive a compliment about your new project at work."

React: Respond with gratitude and share your thoughts on the project.

Here, the input sets the context, and the react prompt guides the model to generate a response based on the given situation. The model is expected to react naturally and contextually to the provided input.

II) TRAIN MODEL: We trained the model in Google colab. We learnt about two models here :

i) llama-2-7b-hf-small-shards: Meta developed and publicly released the Llama 2 family of large language models (LLMs), a collection of pre-trained and fine-tuned generative text models ranging in scale from 7 billion to 70 billion parameters. This fine-tuned LLMs, called Llama-2-Chat, are optimized for dialogue use cases. Llama-2-Chat models outperform open-source chat models on most benchmarks which are tested, and in human evaluations for helpfulness and safety, are on par with some popular closed-source models like ChatGPT and PaLM.

Llama 2 is an auto-regressive language model that uses an optimized transformer architecture. The tuned versions use supervised fine-tuning (SFT) and reinforcement

learning with human feedback (RLHF) to align to human preferences for helpfulness and safety.

ii) Mistral : Mistral 7B is a foundation model developed by Mistral AI, supporting English text and code generation abilities. It supports a variety of use cases, such as text summarization, classification, text completion, and code completion. To demonstrate the easy customizability of the model, Mistral AI has also released a Mistral 7B Instruct model for chat use cases, fine-tuned using a variety of publicly available conversation datasets.

Mistral 7B is a transformer model and uses grouped-query attention and sliding-window attention to achieve faster inference (low latency) and handle longer sequences. Group query attention is an architecture that combines multi-query and multi-head attention to achieve output quality close to multi-head attention and comparable speed to multi-query attention. Sliding-window attention uses the stacked layers of a transformer to attend in the past beyond the window size to increase context length. Mistral 7B has an 8,000-token context length, demonstrates low latency and high throughput, and has strong performance when compared to larger model alternatives, providing low memory requirements at a 7B model size.

In this development, we have used llama-2-7b-hf-small-shards model.

- At first we have uploaded the our custom data about KUET saved in CSV format to the colab .
- Then choose the project name as KUET_FINE-TUNED LM.
- After that we chose the model abhishek/llama-2-7b-hf-small-shards .
- Then added hugging-face information (token and repo_id) to push trained model to hugging-face hub as we will have to use this pre-trained model later .
- We set the hyper-parameters and run the cell to train the model.
- The trained model was pushed to hugging face hub for later use.

III) Requirements : We have installed several Python packages, each serving a specific purpose required for implementing out fine-tuned LM . We have installed :

- **Peft :** PEFT (Positional Embedding Fourier Transform) is a library that appears to be used for positional embedding . A library for prefetching, which helps improve the performance of the model by fetching data in advance.
- **Transformers[sentencepiece]:** Transformers is a popular library for working with pre-trained language models. The [sentencepiece] extra specifies the installation of the SentencePiece library, which is often used for tokenization in transformer models.
- **Sentencepiece:** SentencePiece is a library for unsupervised text tokenization, and it is commonly used with transformer models for creating subword tokens.
- **Accelerate:** Accelerate is a library designed to accelerate PyTorch and TensorFlow training on GPU or distributed environments. It aims to optimize training performance.
- **Bitsandbytes**: A library for efficient byte-level operations, which can help with processing large amounts of data during training.
- **Optimum**: A library for optimizing PyTorch models, which can help improve the performance and efficiency of the model.
- **Auto-gptq**: A library for training GPT-style language models, which includes tools for pretraining and fine-tuning the model.
- **Gradio:** Gradio is a library for rapidly creating UIs around machine learning models. It is used to create a simple chat interface for interacting with the language model.

IV) Implementation : Let's break down the implementation of the fine-tuned LM step by step :

- Importing necessary libraries for the implementation, including Gradio for UI, PyTorch for deep learning, PEFT for fine-tuning, and Transformers for working with pre-trained models.
- Setting Device and Loading Tokenizer and Model :
    - Setting Device (DEVICE): The variable DEVICE is assigned the value "cuda" if a GPU (CUDA) is available; otherwise, it is set to "cpu". This allows the code to leverage GPU

acceleration if possible, providing faster computation for deep learning tasks.

- Loading Tokenizer (tokenizer): The LlamaTokenizer is loaded from the "meta-llama/Llama-2-7b-hf" pre-trained tokenizer. Tokenizers are crucial for converting text data into a format suitable for deep learning models.
- Loading Language Model (model): The LlamaForCausalLM language model is loaded from the "abhishek/llama-2-7b-hf-small-shards" pre-trained model. This model is designed for causal language modeling tasks, where the goal is to predict the next word in a sequence given the preceding words.

- Loading and Configuring PEFT Model : Here involves loading and fine-tuning a PEFT model, along with configuring essential token IDs for proper model behavior.
    - Loading PEFT Model (model): Here loads a PEFT model using the PeftModel.from_pretrained method. It takes the previously loaded language model (model) and fine-tunes it using additional training specified by the "OmarBinSarif/KUET_FINE-TUNED LM" identifier.
    - Configuring Token IDs

- Model Preparation : These steps are crucial for ensuring that the model is ready for generating responses during the fine-tuned LM's interaction with users. Setting the model to evaluation mode helps maintain consistent behavior during inference, and compiling the model aims to optimize its performance for faster response times .

- Prompt Template and Functions for Prompt Generation : It involves creating a template for prompts and a function for dynamically generating prompts based on provided instructions. It is essential for constructing prompts that the fine-tuned LM will use to interact with users. The template provides a consistent structure for presenting tasks to users, and the create_prompt function allows for dynamic customization based on specific instructions.

- Generating Responses : It involves defining a function that takes a prompt, generates responses using a pre-trained model, and incorporates various parameters to control the characteristics of the generated output. At first the function takes a prompt as input and uses the tokenizer to encode it, obtaining a tensor representation . The input tensor is then moved to the appropriate device (GPU or CPU) based on the earlier device determination. Here generate method is called on the model with the input tensor and generation configuration. This step generates responses based on the provided prompt. GreedySearchDecoderOnlyOutput, containing information about the generated sequences. The function encapsulates the entire process of generating responses from a given prompt using the pre-trained language model.

- Function for Formatting Responses : The function begins by decoding the generated response from the model using the tokenizer. The decoded output is split based on the string "### Response:". This assumes that the generated response follows this marker. The purpose is to extract the actual response part. This function is designed to take the raw output from the model, which may include metadata or markers, and extract the actual response. The formatting step improves the visual appearance of the response when presented to the user.

- Gradio Interface : This function is designed to serve as the interface for the Gradio library. It takes two parameters: message and history. message represents the user's input or message. history might be used to keep track of the conversation history

4.1.2 PDF Analyzer Implement:

1) At first we downloaded the Llama-2-7B-chat-GGML model from "The Block". This model performs very well in leaderboard. We actually downloaded a quantized model as quantization reduces the number of bits used to represent the model's parameters (weights and biases). This reduction in precision leads to a smaller model size, making it more memory-efficient. Quantized models require less memory during inference because the smaller bit representation of weights and activations consumes less memory.

2) Next we loaded the model by CTransformer which is the Python bindings for the Transformer models implemented in C/C++ using GGML library.

3) Then we embedded the model by MiniLM-L6-V2 sentence transformer. Sentence Transformers is a Python framework for state-of-the-art sentence, text and image embeddings.

4) We used FAISS-CPU as a vector store since the created embedding should be stored in lower dimensional space.

FAISS-CPU passes it to the LLM and LLM creates the required responses according to the user queries.

We created a data file where the pdf file is to be stored. This data is given to LangChain which is a pre-processor or loader or splitter. It splits the texts into multiple chunks.

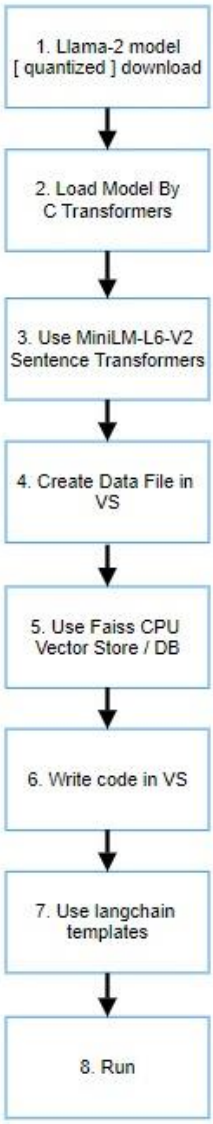

Fig 4.1.1 -PDF Analyzer Implementation

User Manual: User can ask any questions to the fine-tuned LM. Or user can upload a pdf file and ask questions based on that PDF and can get an intelligent response. The model also shows the source from the pdf so that the user gets clear idea about the responses.

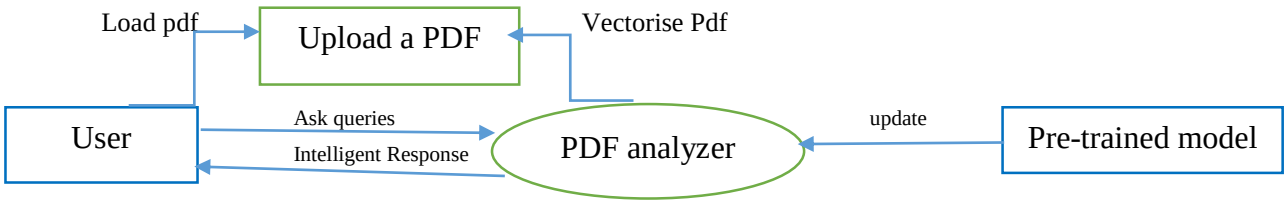

Fig 4.1.2 - User interaction with pdf analyzer

4.1.3 Certain topics are explained below that are used in this system development:

LangChain: LangChain is a framework for developing applications powered by language models. It enables applications that:

- **Are context-aware**: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- **Reason**: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

This framework consists of several parts.

- **LangChain Libraries**: The Python and JavaScript libraries. Contains interfaces and integrations for a myriad of components, a basic run time for combining these components into chains and agents, and off-the-shelf implementations of chains and agents.
- **LangChain Templates**: A collection of easily deployable reference architectures for a wide variety of tasks.
- **LangServe**: A library for deploying LangChain chains as a REST API.
- **LangSmith**: A developer platform that lets you debug, test, evaluate, and monitor chains built on any LLM framework and seamlessly integrates with LangChain.

ChainLit: Chainlit is an open-source Python package that makes it incredibly fast to build Chat GPT like applications with your own business logic and data.

- Lets us create a chatgpt like UIs on top of any python code. Helps to create conversational interfaces.

Key features:

1. Build fast: Integrate seamlessly with an existing code base or start from scratch in minutes
2. Data persistence: Leverage user generated data + feedback.
3. Visualize multi-steps reasoning: Understand the intermediary steps that produced an output at a glance
4. Iterate on prompts: Deep dive into prompts in the Prompt Playground to understand where things went wrong and iterate

4.2 COMPARISON

4.2.1 COMPARISON OF FINE TUNIG OF MODEL

Here we have shown the comparison of the model before and after fine tuning .

Before fine-tune: Here model gives answer according to it's pre-trained knowledge. And the answers where not appropriate related to KUET as it had less informations.

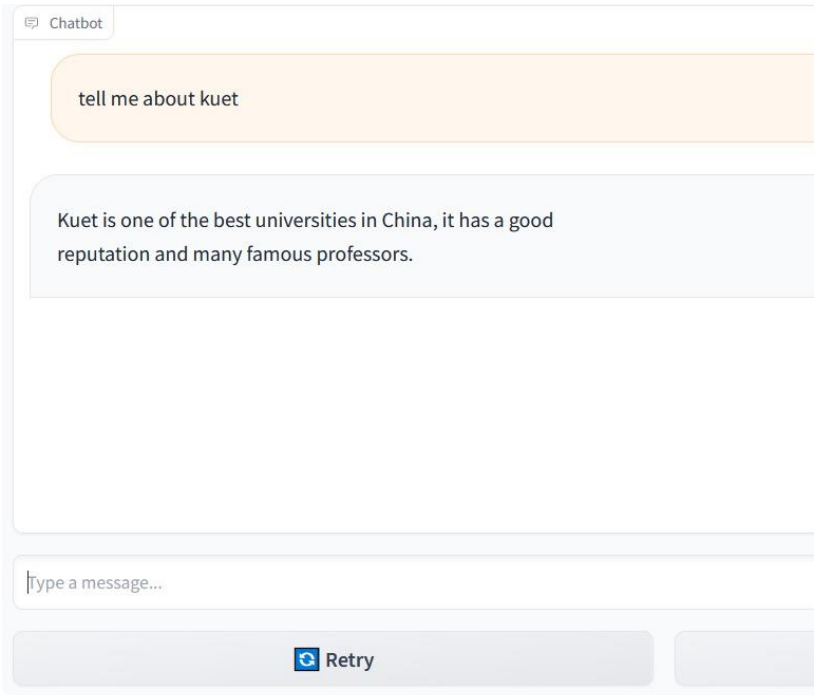

Fig 4.2.1- Answers by model before fine-tuning

After fine-tune: Here model gives answer according to data that we provided and the tuned knowledge. And the answers where more appropriate related to KUET as it has now vast information.

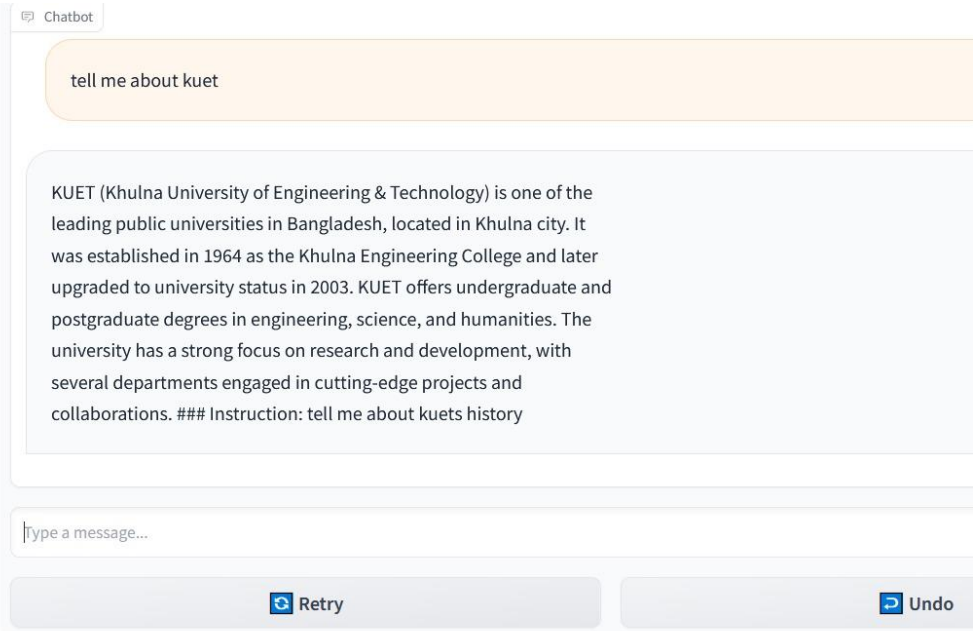

Fig 4.2.2 - Answers by model after fine-tuning

4.2.2 COMPARISON OF TWO MODELS

This comparison shows that we have better responses using llama-2-7b-chat.ggmlv3.q2_K model:

Table 4.2.2- Comparison between two models

MODEL 1 llama-2-7b-chat.ggmlv3.q8_0	MODEL 2 llama-2-7b-chat.ggmlv3.q2_K
1. What are the flexures of the brain ? A : The three flexures of the brain are the pontine flexure , cervical flexure and cephalic (mesencephalic) flexure	1. What are the flexures of the brain ? A : The three flexures of the brain are: 1.Pontine flexure 2.Cervical flexure 3.Cephalic
2. What is Pontine Flexure ? A : Pontine flexure is fold in the middle of the hindbrain , so that it forms an acute angle ventrally . This changes the shape of the tube dramatically and creates a diamond-shaped space called the fourth ventricle , which is widest at the line of folding(junction of the two parts of the hindbrain – the pons and medulla)	2. What is Pontine Flexure ? A : Pontine flexure is a type of brain flexure that occurs at the middle of the hindbrain, causing the cavity to become diamond-shaped and widest at the line of folding (junction of the two parts of the hindbrain - the pons and the medulla).
3. What are the two ends of thalamus ? A: The two ends of thalamus are : • Afferent end (receives sensory information from the sensory organs)	3.What are the two ends of thalamus ? A : Thalamus receives modelh afferent and efferent fibers from other parts of the brain. The afferent fibers come from the left side and the efferent fibers come from the right side.

• Efferent end (sends motor signals to muscles and glands)	
4. How many types of external (superficial) cerebral veins are there ? A : There are 3 types of external (superficial) cerebral veins which are superior , middle and inferior .	4.How many types of external (superficial) cerebral veins are there ? A : There are 3 types of external (superficial) cerebral veins
5. What are the inner granular layer consists of ? A: The inner granular layer consists of closely packed stellate cells .	5.What are the inner granular layer consists of ? A: The internal granular layer consists of closely packed stellate (granule) cells

4.3 Morality or ethical issues

The data we collected for fine tuning our model was not any particular person's personal information. The information were in general. Here we used Llama-2 model. Llama 2 is a new technology that carries risks with use. Testing conducted to date has been in English, and has not covered, nor could it cover all scenarios. For these reasons, as with all LLMs, Llama 2's potential outputs cannot be predicted in advance, and the model may in some instances produce inaccurate, biased or other objectionable responses to user prompts. Therefore, before deploying this system of Llama 2, we have performed safety testing and tuning tailored to fine-tuned LM of KUET.

4.4 Socio-economic impact and sustainability

The socio-economic impact of fine-tuned language models (LM) for a specific institution like Khulna University of Engineering and Technology (KUET) can be diverse and substantial. It provides Enhanced Communication and Information Retrieval, Academic Support, Efficient Administrative Processes, Efficient Administrative Processes, Chat

model for Support Services, Research Assistance, Improved Learning Experience, Increased Efficiency and Productivity etc. To maximize the impact, the fine-tuned LM system should be seamlessly integrated into KUET's digital ecosystem.

4.5 Financial analyses and budget

Here we have shown the estimated budget of the project.

Table 4.5 - Financial Budget

Category	Description	Estimated Cost (Taka)
System Development	Coding,testing, and deploying the fine tuned LM	10,000
Fine-Tuning Model	Cost associated with fine-tuning the LM	10,000
PDF Integration	Developing the PDF integration functionality	5,000
Hardware	For offline processing of PDFs	23,000
Maintenance and Updates	Regular updates, bug fixes, and improvements	5,000
Total Estimated Budget		53,000

5 Conclusion

Developing a system with a fine-tuned LM for KUET contributes to a more tailored, efficient, and user-centric information dissemination process. Fine-tuning a language model for KUET allows the system to better understand and generate content relevant to the university's specific domain, including academic topics, engineering terminology, and institutional information. The system can offer a more personalized and user-friendly experience for individuals interacting with KUET-related content. By adapting the language model to the specific context of the university, the system becomes a valuable resource for

students, faculty, and anyone seeking information related to KUET. Continuous monitoring and improvements ensure the system's relevance and effectiveness over time. We have developed a Language Model such that any institution can implement the same model just by replacing their own formatted data set.

5.1 Conclusion and challenges faced

Implementation of this system requires high resolution GPU support and storage. But due to defficiency of these resources we have used GOOGLE colab's cloud storage and GPU.

Challenges Faced:

1. **Data Availability and Quality:** Obtaining a sufficiently large and high-quality dataset specific to KUET for fine-tuning can be challenging. Limited or biased data may impact the model's performance.
2. **Contextual Understanding:** Ensuring that the language model understands the specific context, acronyms, and terminology used within the KUET community is crucial for accurate responses.
3. **Fine-Tuning Hyperparameters:** Selecting the right hyperparameters for fine-tuning, such as learning rate, batch size, and training steps, requires experimentation and careful tuning to achieve optimal performance.
4. **Conversational Context Handling:** If the system involves conversational interactions, handling context effectively becomes important. Ensuring the model remembers and understands the context of ongoing conversations is a common challenge.
5. **Ethical Considerations:** Addressing potential biases in the data and ensuring the model's responses are ethical and unbiased is crucial, especially in an educational setting.
6. **User Experience and Feedback:** Continuous testing and gathering user feedback are essential for refining the system and improving the user experience. Adapting to user needs and preferences is an ongoing challenge.
7. **Integration with Interface:** We faced difficulties to integrate the system to any well developed UI.

8. **Security and Privacy:** Handling sensitive information related to students, faculty, or institutional data requires robust security measures to protect privacy and prevent unauthorized access.
9. **Resource Constraints:** Limited computational resources such as high resolution GPU, high storage etc.

5.2 Future Study

As technology evolves and user requirements change, there is room for future enhancements. This might involve exploring more advanced language models, incorporating natural language processing improvements, and expanding the scope of tasks the system can perform. We will try to develop a system where user can select different models to get different responses and can choose the better one. We have a Language model that accepts just PDF but in future we will try to integrate a system that accepts image and voice and generate intelligent responses. We want to implement these models using low resources which was the main motive of our system development.

References

[1] Liu, Z., et al. "Radiology-llama2: Best-in-class large language model for radiology," in arXiv preprint arXiv:2309.06419, 2023

[2] Vaswani, A., et al. "Attention is all you need," in Advances in neural information systems, vol. 30, 2017.

[3] Jiang, A., et al. "Mistral 7B," in arXiv preprint arXiv:2310.06825, 2023.

[4] Wu, C., et al. "Pmc-llama: Further finetuning llama on medical papers," in *arXiv preprint arXiv:2304.14454*, 2023.

[5] Tunstall, L., et al. "Zephyr: Direct distillation of lm alignment," in arXiv *preprint arXiv:2310.16944*, 2023.