LLM Text Summarization

Final Year Project

Session 2021-2024

A project submitted in partial fulfillment of the degree of

BS in Gaming & Multimedia

Department of Computer Science

Faculty of Computer Science & Information Technology

The Superior University, Lahore

Fall 2024

Type (Nature of project)	[ ] **D**evelopment [✔] **R**esearch [ ] **R&D**
Area of specialization	Text Summarization
FYP ID	FYP-BGMM-S24-002

Project Group Members

Sr.#	Reg. #	Student Name	Email ID	*Signature
(i)	Bgmm-s21-001	Ahmad Raza	Bgmm-s21-001@superior.edu.pk	
(ii)	Bgmm-s21-002	Shaheer Nouman	Bgmm-s21-002@superior.edu.pk	
(iii)	Bgmm-s21-006	Ali Hassan	Bgmm-s21-006@superior.edu.pk	

*The candidates confirm that the work submitted is their own and appropriate credit has been given where reference has been made to work of others

Plagiarism Free Certificate

This is to certify that I, Ahmad Raza S/D of Muhammad Mumtaz Ahmad, group leader of FYP under registration no Bgmm-s21-001 at Computer Science Department, The Superior University, Lahore. I declare that my FYP report is checked by my supervisor.

Date:

Name of Group Leader: Ahmad Raza Signature: _____

Name of Supervisor: Sir Muhammad Owais

Designation: Lecturer

Signature: _____

HoD: Dr. M. Azam

Signature: _____

Project Report

LLM Text Summarization

Change Record

Author(s)	Version	Date	Notes	Supervisor's Signature
Shaheer Nouman	1.0	20-01-24	Start Working on gathering Dataset	
Ahmad Raza	1.2	11-02-24	Doing Preprocessing of Data	
Ali Hassan	1.6	08-03-24	Diagrams	
Ahmad Raza	2.0	25-04-24	Prototyping	
Ahmad Raza	2.4	23-09-24	Research papers	
Ahmad Raza	2.6	10-10-24	Different Models, Fine-Tuned on Selected model	
Ahmad Raza	2.7	02-11-24	Testing, Making it on Roman Urdu	
Ahmad Raza	3.0	11-11-24	Creating API then do prediction with web steamlit UI	

APPROVAL

.

PROJECT SUPERVISOR

Comments: _____	
_____	
Name: _____	
Date: _____	Signature: _____

.

PROJECT MANAGER	
Comments: _____	
Date: _____	Signature: _____

HEAD OF THE DEPARTMENT	
Comments: _____	
Date: _____	Signature: _____

Dedication

We dedicate this Large Language model project to our supervisor, Sir Owais, whose unwavering guidance and expertise have been the cornerstones of our learning journey. Their good feedback, help, and fabulous suggestions have shaped this project and enhanced our skills in artificial intelligence. We are also sincere to my FYP manager, Sir Jawad Ahmad, for their helpful support and guidance in this project. His view encouraged navigating the challenges and complexities of our text summarization project. This project was a valuable lesson, and I gained good experience under your supervision. We are grateful that we got the opportunity to work under them and their strong guidance, and we will strengthen our skills for a solid foundation in this field in the future. Thank you for being our best mentor and inspiration on this journey

Acknowledgements

To get invaluable help from our supervisor, Sir Owais, who has wavered for support and encouragement throughout the entire project of our FYP, we present our supervisor, Sir Owais. Their specialization, expertise, good feedback, and kindness to our regular and professional development for the successful completion of this project and contribution to overall steps. Our heartfelt thanks to our department students, friends, and class fellows for their ideas, shared insights, and good spirit. Your support is like a source of motivation when we face the challenges in this project. Additionally, we would like to acknowledge the university for providing us with a platform to learn and strengthen our skills using their great environment. This project marks the good of our educational journey, and that would not have been possible without the guidance and help received during the war. Thank you, Sir Owais, for being our supervisor and guide who believed in us, our potential, and our growth

Executive Summary

This large language model research project aims to explore the fundamentals of preprocessing data and algorithms that will be applied and the impact of this Text Summarization on artificial intelligence (AI). So, the project is divided into different steps, using NLP, a model like Pegasus. The research also used the roadmap of AI, also called generative AI, such as present trends, challenges, and future ways. The project will be a comprehensive report and presentation summarizing the implementation of chatbots on AI and our society.

Table of Contents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1: Introduction

The introduction explores a large language model in the domain of text summarization using a deep learning algorithm called RNN, and its types like Lstm, GRU, and BRU. LLM has been remarkable in NLP tasks, and their application that we can used that could lead to significant automatic text summarization techniques to ease and our time. This report presents all the steps we took in practice experiments to find the best model then we apply in LLM text summarization to help us get good results.

1.1 Background

The fundamental method of this model relies on different rules, and their ability to learn human language is limited. With various kind of models like gpt3, gpt4, and Bert, there is a significant background towards a data-driven approach, the power of machine learning, and deep understanding that generates summaries that the data is relevant and accurate. This project builds upon this foundation to explore the potential of LLMs in text summarization and contribute to advancing specific fields

1.2 Motivations and Challenges

The motivation behind this project stems from the increasing need for efficient and accurate text summarization techniques used in our project, particularly in handling the ever-growing large amount or volumes of textual data available online. By leveraging LLMs, this research seeks to address the limitations of traditional summarization methods and improve the quality and effectiveness of text summaries of paragraph. However, integrating LLMs into text summarization poses several challenges, including the need for extensive training data on specific fields because the chatbots didn’t train on fields.

1.3 Goals and Objectives

This project aims to appraise the performance of the PEGASUS model in the circumstances of text summarization using the Samsum dataset. Also, the goal is well and good objective in a specific field. The objectives include Preprocessing the Samsum is dataset for training and evaluation. To fine tune the PEGASUS on the Samsum data for text summarization.

1.4 Literature Review/Existing Solutions

Text summarization is a field with a rich impact history of research, encompassing various methodologies to distill information from texts and paragraphs. Traditional methods, like extractive summarization, have limitations in coherence and relevance. There is an existing solution for the chatbot, but we taught in one and many domain that will help to solve other problems. Modern approaches, especially those leveraging models and transformers, offer better abstractive summarization techniques in a transformer-based model

1.5 Gap Analysis

Coherence and Consistency: While transformer models have improved the coherence of generated summaries, there is still a gap in ensuring that summaries are consistently coherent, especially with complex language structures.

Biases and Fairness: Addressing biases in generated summaries remains challenging, requires further research to development more effective bias detection and mitigation techniques.

Data Efficiency: Training large transformer models like PEGASUS requires vast amounts of data and not less amount, highlighting the need for more data-efficient approaches to text summarization.

Evaluation Metrics: Current evaluation metrics like ROUGE may not fully capture the quality of generated summaries of paragraph, indicating a need for more nuanced and comprehensive evaluation methods.

Real-time Summarization: Many existing models are computationally and intensive, hindering real-time summarization applications and other. Future research could focus on developing lightweight models for faster summarization.

1.6 Proposed Solution

The proposed solution of our project is that in some companies and other domains, factories help them to understand easily rather than reading many paragraphs, like a student can quickly clear his concept using a large language model of text summarization. That’s why, in this era, Generative AI benefits us. You need to take advantage of this latest technology.

1.7 Project Plan

Month 1-2:

- Background Research Study the fundamentals of AI, using ML, DL, and NLP. Explore history, development, and critical concepts of large language models.

Month 3-4:

- Algorithm Understanding: Dive deep into the algorithms used in machine learning, focusing on neural networks, attention mechanisms, and transformer architectures. Study the working principles of large language models.

Month 5-6:

- Fine Tune our model and transfer learning, including current trends, challenges, and future directions. Investigate the applications of large language models in various fields, such as healthcare, finance, and education, or any other specific domain you need to

Month 7-8:

- Deployed on Streamlit or cloud according to background money and Analysis. Synthesize the knowledge gained from the research into a comprehensive understanding of large language models. Analyze the impact of large language models on artificial intelligence and their potential for future advancements.

1.8 Roles & Responsibility Matrix

Table 1 Role & Responsibility Matrix

Task/Stakeholders	Duration	Responsible Team Member(s)
Project Planning	February	All
Text Preprocessing	March	Ahmad
Model Selection	April	Ahmad Raza
Fine Tuning	August	Ali Hassan
Testing and Validation	October	Shaheer Nouman
Deployment	November	Ahmad Raza

1.9 Gantt Chart

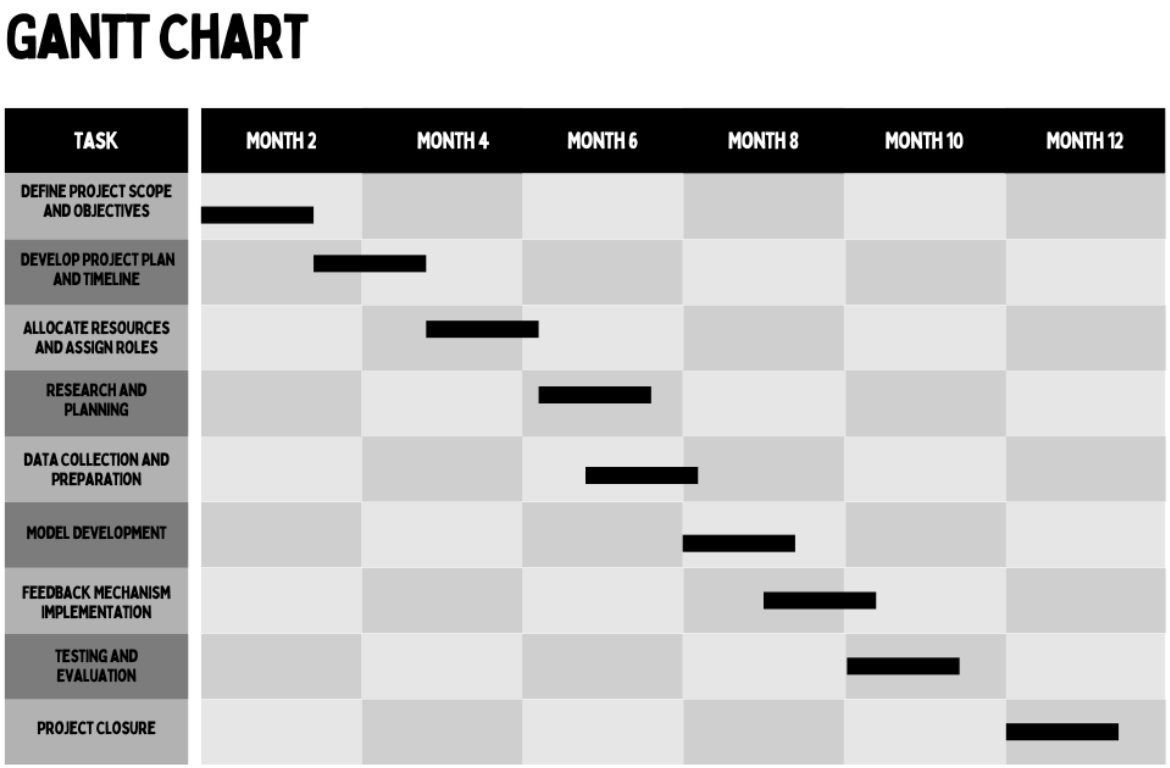


Figure 1gantt chat

1.10 Empathy Map

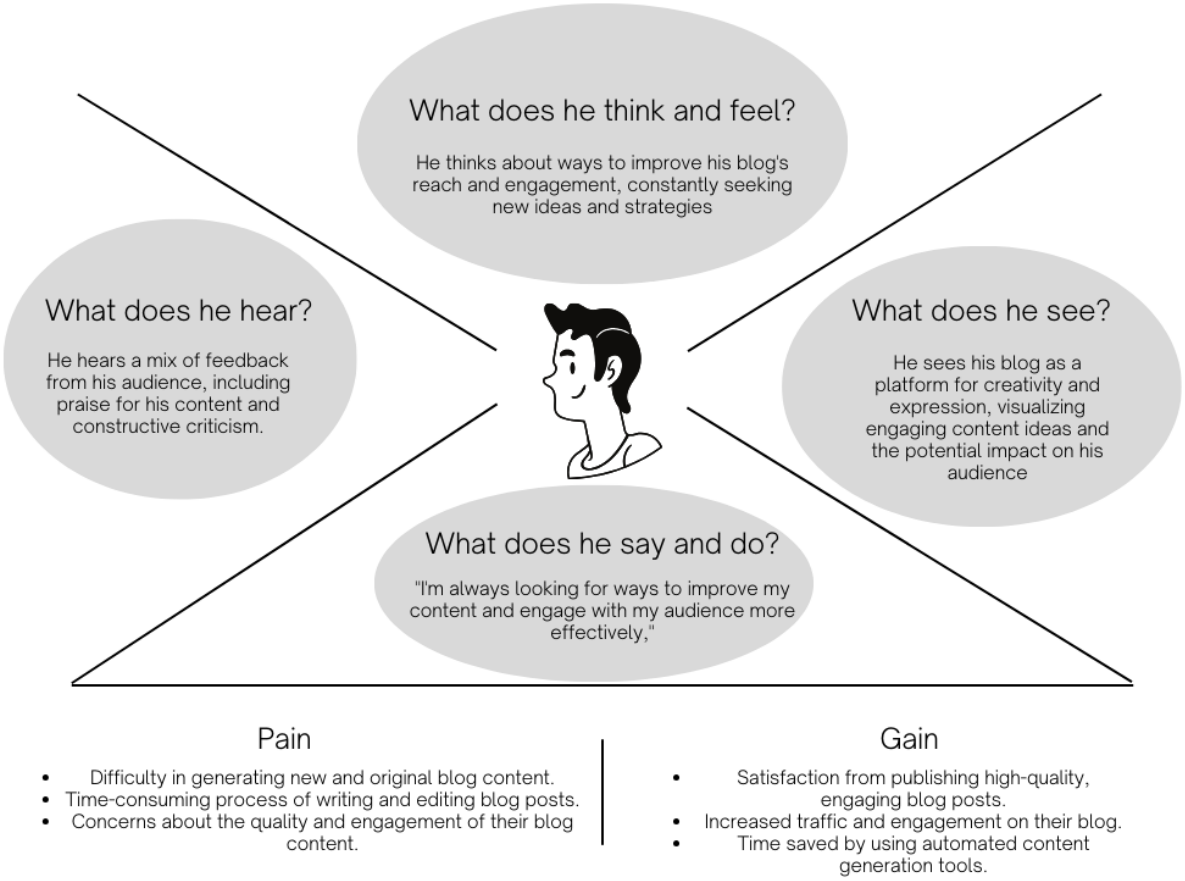


Figure 2 empathy map

Chapter 2

Software Requirement Specifications

Chapter 2: Software Requirement Specifications

Introduction

2.1.1 Purpose

The purpose is to take extensive text data as input that can auto-generate abstractive summaries. This scope covers entire text summarization, including preprocessing given data in that domain after fine-tuning our Pegasus model and giving summaries. It also integrates other components of the system. Like user interface, storage, as well as an external operation that is required

2.1.2 Document Conventions

This project follows the steps of the document to justify consistency in present needs. The heading and subheading are in bold text and have a caliber of 12pt and a spacing of 1.5. At the same time, the statement is simple text. Other priority levels with bullets is standard requirements. Key terms or concepts may be highlighted using italicized text for emphasis

2.1.3 Intended Audience and Reading Suggestions

The project is intended for various audiences involved in the system's development, management, and documentation. Developers will use this document to understand the requirements and implement them during the software development process. Project managers will gain insights into the scope according to the market and its requirements, objectives, and constraints of the project, enabling them to plan and manage resources effectively.

2.1.4 Product Scope

One of the extraordinary scopes of this product that saves time is the powerful ability to effectively summarize large amounts of text into conversational text and less effort for our users who do not need to read an amount of data to understand all the processes that have been given to him according to his task or other. The primary key of this system to create a robust and best environment text summarization using the model’s name Pegasus that will be to our dataset names called same sum to optimize to get excellent performance and

evaluate using its metrics like such as rough just like we used in machine learning algorithm are TP, TN, FN, and FP.

2.2 Overall Description

2.2.2 Product Perspective

The project outlines the functional context of the overall description and distinguishes the software system. This tells the system summary. Its human computer interacts with the user and the technology and dependencies. Additionally, all descriptions used the system features, like user characteristics and hardware requirement limitations. The set stage for in-depth requirements and UI specification

2.2.3 Product Functions

A software application that is used to generate short text using long text. Its user-friendly UI allows them to input or give data and get output. The system does not need any other outside system or database for its work, like cores that like access to the internet for additional updates or any need for other resources. Its LLM text summarization is the part, or you say that system of eco of natural language processing and technologies to advancement of ai text driven processing

2.2.4 User Classes and Characteristics

The system used various user classes and characteristics, each with different and unique roles. Also have requirements. Researchers well-known in machine learning, deep learning, and natural language processing depend on the system to use algorithms, datasets, and their metrics to create research and do practical. The developer depends on implementing and editing codes to maintain their environment system; they also need in-depth knowledge of architecture, code language, and integration. Managers overview and oversee the entire development system and planning deployment and how to manage the project. Depending on the results, the tester validates the functional and non-functional users who seek a user-friendly interface UI. Each user class contributes uniquely to all these systems, and in the end, they highlight the essential key points that approach to meet their specific needs

2.2.5 Operating Environment

The environment is designed like every user with operating systems like Windows, Linux, and MacBook can operate to use this. It requires a minimum of 16GB of RAM and a multi-core processor for optimal performance. The system relies on the Python programming language and libraries such as TensorFlow and PyTorch for machine learning, deep learning, and natural language processing tasks. An internet connection is required if you are using the Google Colab engine, but in our project, we used

2.2.6 Design and Implementation Constraints

Firstly, the computational resources available may limit the system's performance, including processing power and memory. We need to train many parameters, including called weights and other models, such as chatbot trained on billions of parameters. The complexity of the algorithms used, particularly for text summarization and natural language processing, may affect the system's ability to generate accurate summaries. Integration with external systems or services for data retrieval, processing, or storage introduces constraints related to compatibility and security. Compliance with relevant regulations and standards, such as those related to data privacy and security, is also a constrain

2.2.7 User Documentation

The design of the system application is easy to use and interacts with entities through various functionalities. Complete their task using all functional requirements from the task they are given to their university and college. It is intuitive and has varying levels of technical expertise. User can input data, and the system provides output as the data according to their functional requirements that need to be fulfil

2.2.8 Assumptions and Dependencies

To support its operations effectively, the LLM text summarization system operates under certain assumptions and dependencies that assume access to adequate computational resources, including the CPU's processing power, GPU, and cores or memory. Additionally, the system relies on the availability of high-quality training data, like we train a model to our Samsum dataset, to train and optimize its algorithms for text summarization. A stable

internet connection is also assumed for accessing online resources, updates, and additional datasets that several epochs run to, depending on batch size, if needed, to calculate loss and accuracy. The main problem is that the vanishing gradient problem does not occur.

2.3 External Interface Requirements

2.3.2 User Interfaces

LLM provides two offers: variety, preference and need. The web interface allows access to the viewpoint and initiative platform through standard web browsers like Chrome, Edge, and Firefox. It is as simple as a layout with an area of input that provides data according to processing to summarize. On the other hand, you can interact with a screen of the desired results to copy and paste. A client called the command line interface will be shown

2.3.3 Hardware Interfaces

Hardware interfaces are compatible and run computers, and laptops. However, hardware components require a high level of 16gb RAM, generation, and i7 12^{th} generation CPU and 10gb GPU to train our model and fine-tune or transfer learning to get our desired result and accuracy. However, it is also optimized for more giant screens for a better user experience. Overall, with its smooth operation, users easily interact across a wide range. For offline user can use this text summarization to get his summary with 4gb ram and i5 6^{th} generation CPU

2.3.4 Software Interfaces

The LLM Text summarization software components are the best for functionality and good performance. Its capability is designed for various operating systems, including Windows, Linux, and MacBook, for executing memory and process by process. Software used the nltk library of natural language, providing for tokenization and parsing data to data its processes to analyze it

2.4 System Features

The LLM offers features to facilitate our user text data. Its central core lies in giving informative summaries using the Pegasus model, a powerful specific domain-trained model

for abstract summarization. Users can interact with it and get their desired results as output, allowing them to view, download, or copy according to their needs. More features will be added in the future as the development will come

2.5 Other Nonfunctional Requirements

2.5.2 Performance Requirements

The LLM text summarization system is designed so that its performance should not consume higher usage of your central processing unit and memory resources because if it happens, your system will slow. A large language model app will work well in your high score operating system. To maintain this, we should evaluate the ROUGH-1 score. These performances are essential to secure the system and meet user expectations and regulatory standards.

2.5.3 Reliability Requirements

The reliability requirement of the LLM text summarization system focuses on the consistency of CRUD operation in software engineering, like dependable operation. A system called peer-to-peer connection should be available occasionally. So, there is minimal downtime to ensure that users can access it when needed. Data integrity should measure that data loss or corruption is in place to prevent. On the other hand, regular backups are implemented so the user should not lose their data. They can save them.

2.5.4 Maintainability/Supportability Requirements

The maintenance of an extensive language model system focuses on the fact that it can be easily maintained and supported 24/7, called 24-hour and seven-day active online user supportability. Version control system GitHub that manages code changes and future updates to facilitate the performance of our chatbot. The system should have sign-in or monitoring capabilities to track their app communication performance, which can detect issues with our application, so we get timely feedback on maintenance.

2.6 Other Requirements

The portability of our LLM text summarization system aim is that the system can easily be deployed and used across various platforms and environments. The platform should be system-dependent, running on other platforms efficiently without any lag or slow like on Windows, Linux, and MacBook. It should widely support the project's compatibility. Easy to install and good handy portability

Chapter 3

Use Case Analysis

Chapter 3: System Analysis

Use case analysis is the critical aspect of this project development that helps us understand our system functionality from the user feedback. This chapter focuses on using a Case diagram and how we create it to visualize the human-computer interaction between the system and the user. A use case description will give you a concept of how the users interact with our software to achieve their goals, providing details and mapping how each case will be used and implemented. It explores the relationship between interactions and dependencies. Prioritization of use cases will be based on their importance and impact on the system's functionality. Finally, a summary of the use case model will be provided to give an overview of the system's behavior and functionality.

3.1. Use Case Diagram

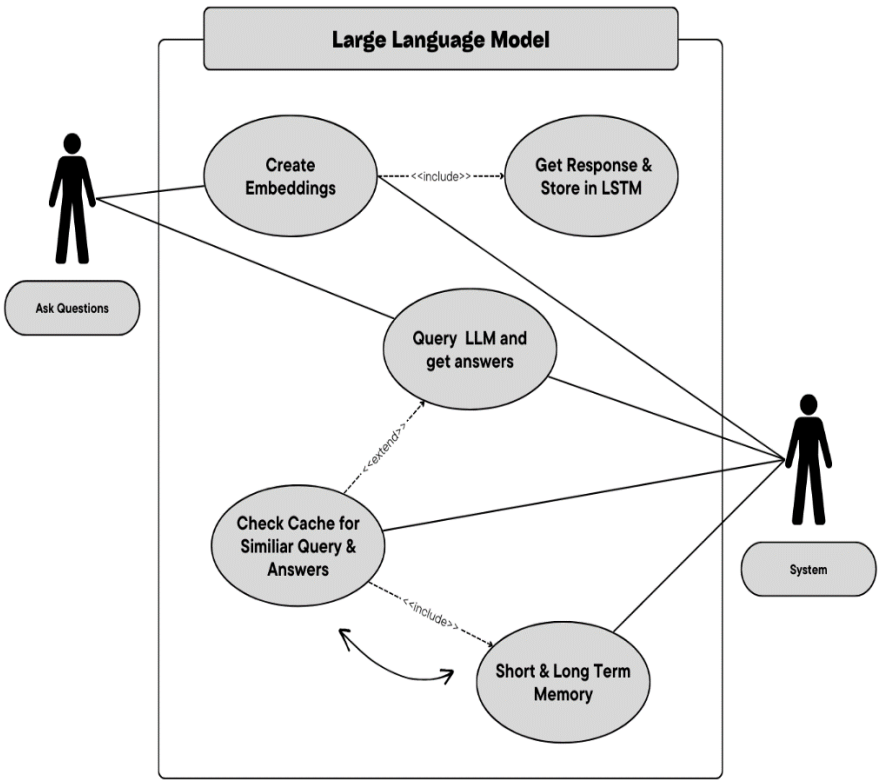


Figure 3 Use Case Diagram

3.2. Use Case Descriptions

Use Case: Input Dialogue

- **Actors**: User
- **Description**: The user inputs a dialogue text into the Streamlit UI, which will be processed by the system to generate a summary.
- **Preconditions**: Streamlit UI is accessible, and the user has a dialogue to summarize.
- **Postconditions**: The dialogue is accepted and stored temporarily for processing.

Use Case: Display Summary

- **Actors**: User
- **Description**: The system generates a summary based on the provided dialogue and displays it in the Streamlit UI for the user.
- **Preconditions**: The dialogue has been processed, and a summary has been generated by the system.
- **Postconditions**: The user views the generated summary.

Use Case: Configure Training Parameters

- **Actors**: Data Scientist
- **Description**: The data scientist configures the model training parameters in Google Colab, setting values such as batch size, learning rate, and number of epochs.
- **Preconditions**: The Pegasus model and tokenizer are accessible in Google Colab.
- **Postconditions**: The model is set up with the configured parameters and is ready for training.

Use Case: Load and Preprocess Data

- **Actors**: System
- **Description**: The system loads the dataset in Google Colab and preprocesses it, including tokenizing dialogues and preparing inputs for training.
- **Preconditions**: The dataset is available and accessible in Colab.
- **Postconditions**: The data is tokenized and prepared for model training.

Use Case: Train Model

- **Actors**: System
- **Description**: The system trains the Pegasus model on the preprocessed dataset in Google Colab, adjusting the model weights based on input and target outputs.
- **Preconditions**: The model is configured with training parameters, and preprocessed data is available.
- **Postconditions**: The model has been trained on the dataset and is ready to be saved.

Use Case: Save Model and Tokenizer

- **Actors**: System
- **Description**: The system saves the trained model and tokenizer in a directory to be accessed later for predictions in the Streamlit app.
- **Preconditions**: The model has been trained successfully.
- **Postconditions**: The model and tokenizer files are saved in a directory accessible to the inference engine.

Use Case: Load Saved Model

- **Actors**: System
- **Description**: The system (Streamlit app) loads the saved model and tokenizer from the directory for inference.
- **Preconditions**: The trained model and tokenizer files are saved and accessible.
- **Postconditions**: The model and tokenizer are loaded into memory, ready for generating summaries.

Use Case: Generate Summary

- **Actors**: System
- **Description**: The system uses the loaded model and tokenizer to generate a summary from the user-provided dialogue input.
- **Preconditions**: The model and tokenizer are successfully loaded, and the user has provided input dialogue.
- **Postconditions**: A summary is generated based on the dialogue and is ready to be displayed

Chapter 4

System Design

Chapter 4: System Design

The system design phase is a critical step in the software development lifecycle, where the high-level system architecture and components are designed to meet the specified requirements. This chapter has provided a detailed overview of the system design, covering various aspects such as architectural design, database design, user interface design, component design, deployment design, security design, integration design, and design patterns.

4.1. Architecture Diagram

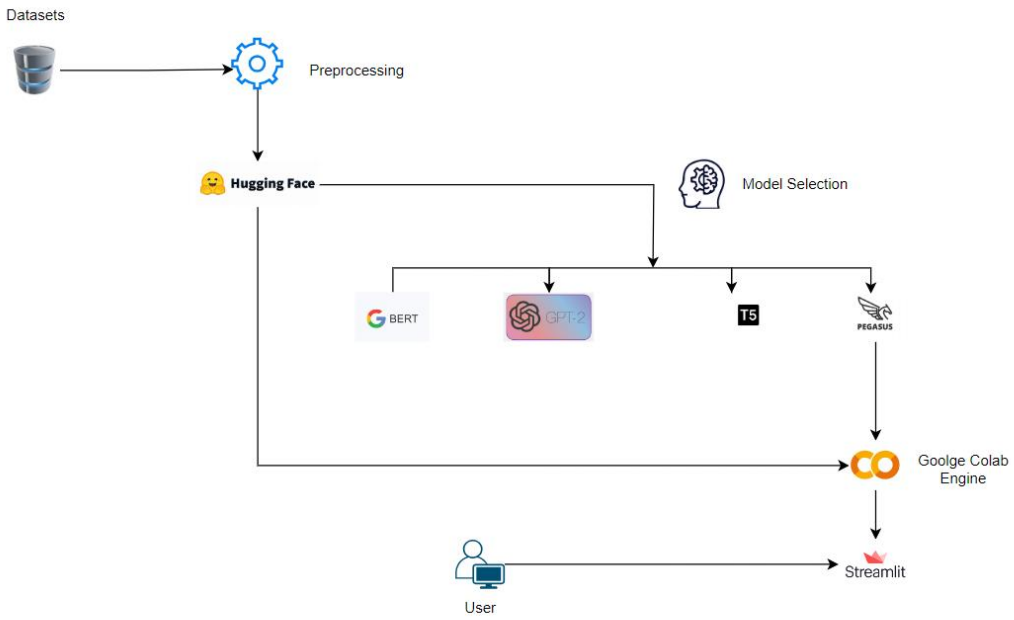


Figure 4 Architecture Diagram

4.2. Domain Model

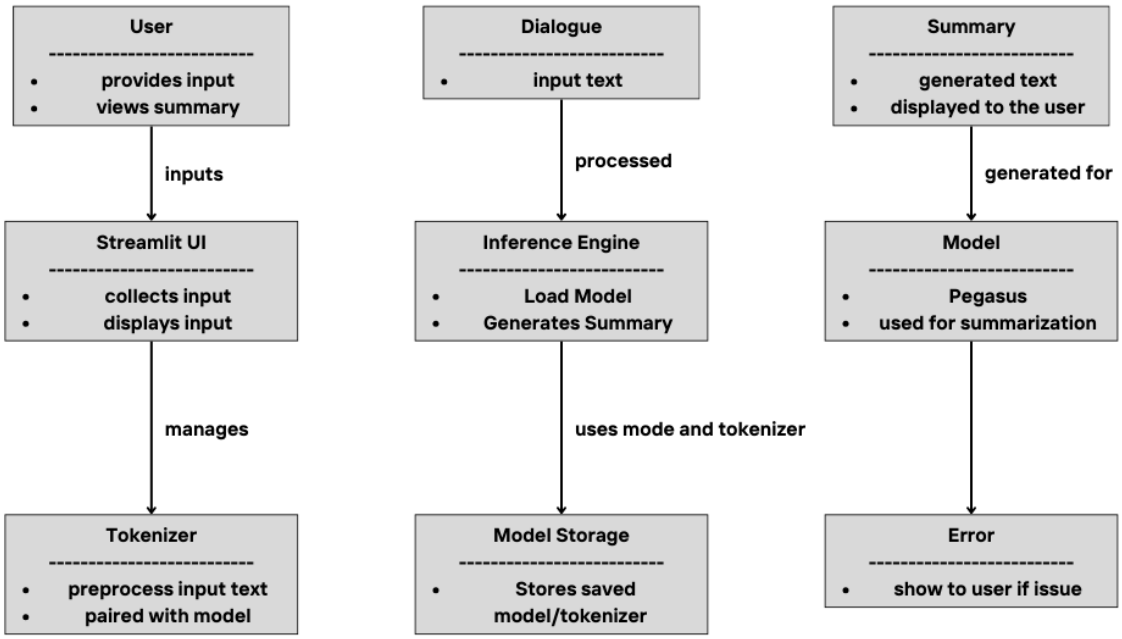


Figure 5 Domain Model

4.3. Sequence / Collaboration Diagram

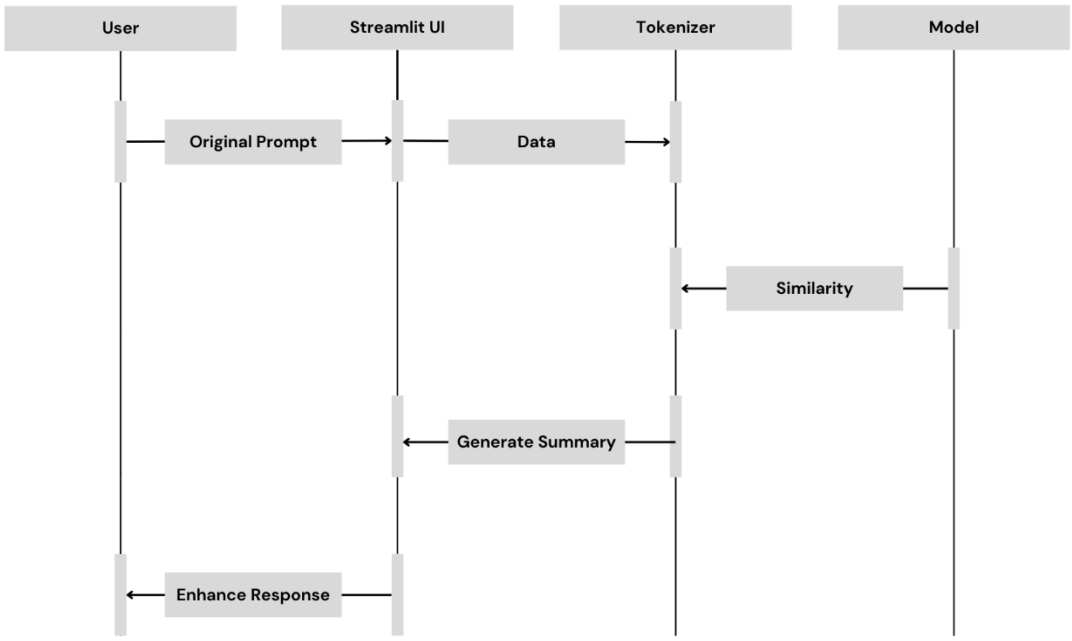


Figure 6 Sequence / Collaboration Diagram

4.4. Operation contracts

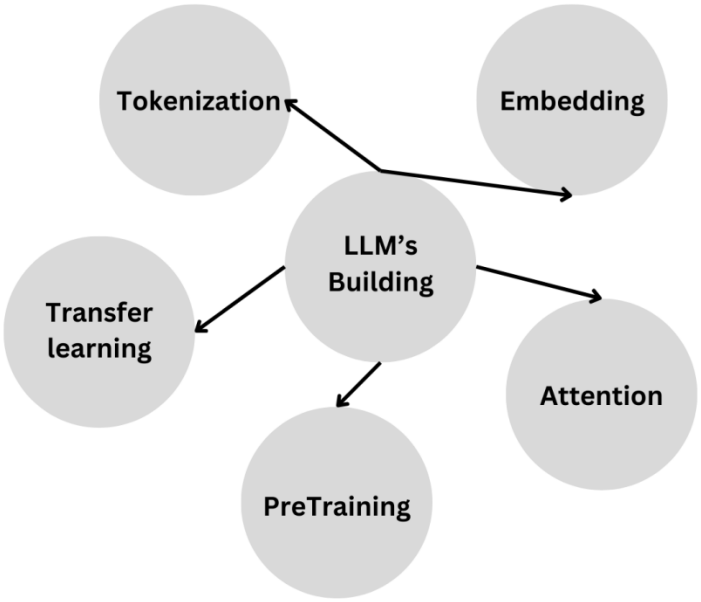


Figure 7 Operation Contracts

Accept Dialogue Input

- **Operation**: acceptDialogueInput()
- **Preconditions**: Streamlit UI is running, and the user provides a dialogue input.
- **Postconditions**: The input dialogue is accepted and stored for processing.

Load Model & Tokenizer

- **Operation**: loadModelAndTokenizer()
- **Preconditions**: Model and tokenizer files are saved in the specified directory, and the directory is accessible.
- **Postconditions**: The model and tokenizer are loaded into memory for inference.

Exceptions:

If files are missing or inaccessible, the system raises an error and logs it.

- **Check if Model Loaded**
- **Operation**: checkModelLoaded()
- **Preconditions**: loadModelAndTokenizer() has been called.
- **Postconditions**:
    - **Success**: Proceeds to generate a summary.
    - **Failure**: Displays an error message indicating that the model couldn't be loaded.

Generate Summary

- **Operation**: generateSummary(dialogue)
- **Preconditions**: Model and tokenizer are successfully loaded, and dialogue input is available.
- **Postconditions**: A summary is generated based on the input dialogue.
- **Exceptions**: If the generation process fails (e.g., due to model errors), an error message is displayed.

Display Summary

- **Operation**: displaySummary(summaryText)
- **Preconditions**: A summary has been successfully generated by generateSummary().
- **Postconditions**: The generated summary is displayed to the user in the Streamlit UI.

Display Error Message

- **Operation**: displayErrorMessage(message)
- **Preconditions**: An error has occurred during model loading or summary generation.
- **Postconditions**: An error message is shown to the user in the Streamlit UI, informing them of the failure.

4.5. Activity Diagram

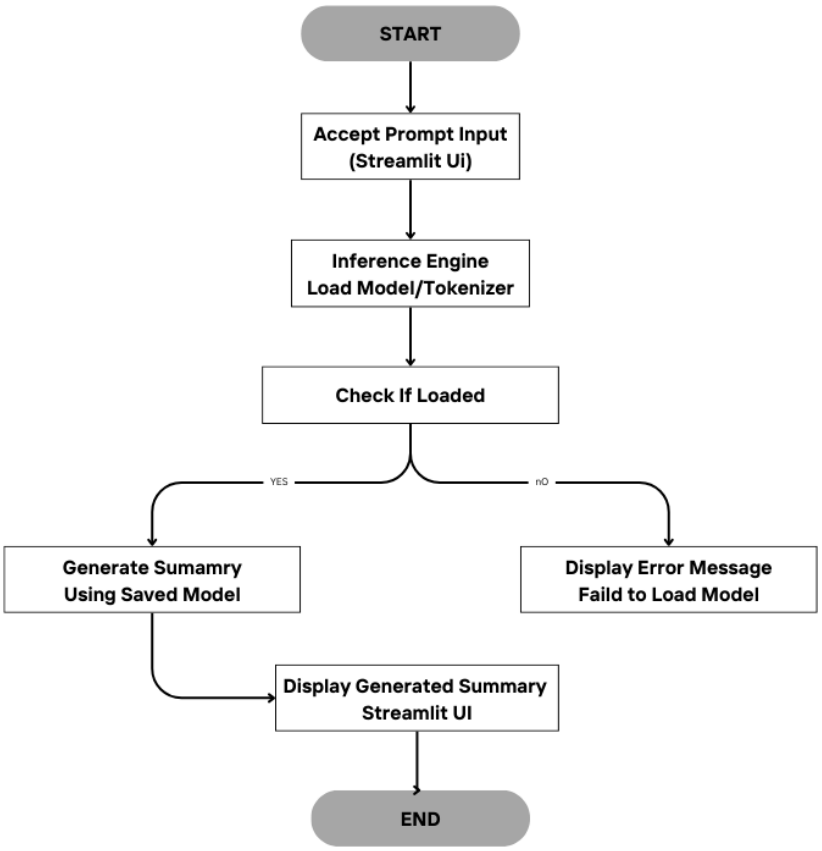


Figure 8 Activity Diagram

4.6. State Transition Diagram

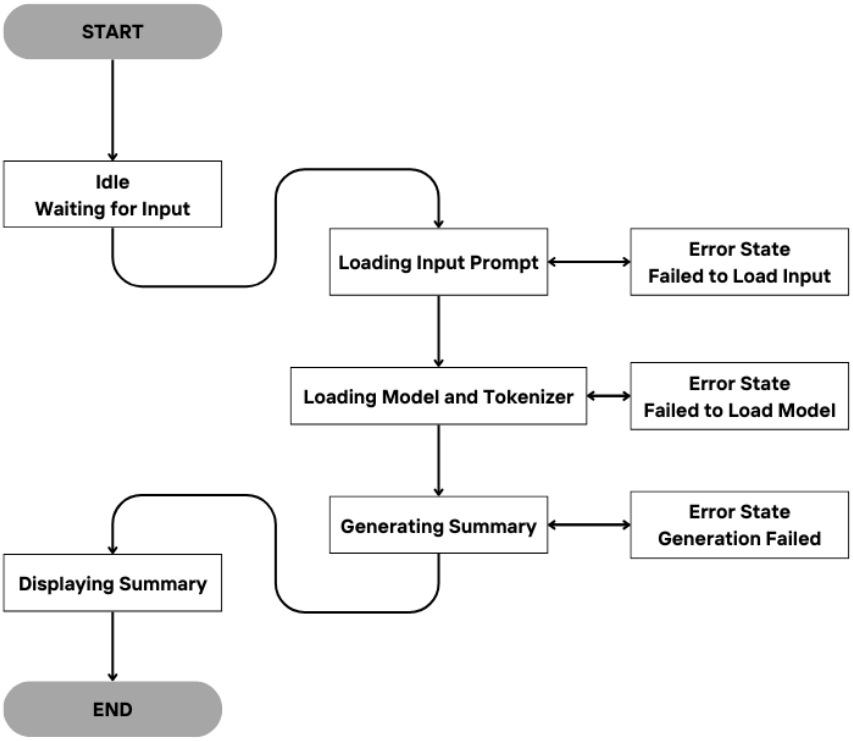


Figure 9 State Transition Diagram

4.7. Component Diagram

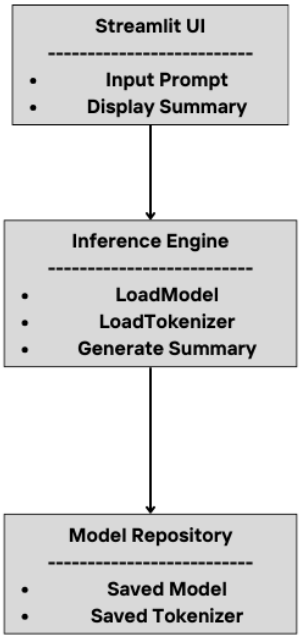


Figure 10 Component Diagram

4.8. Deployment Diagram

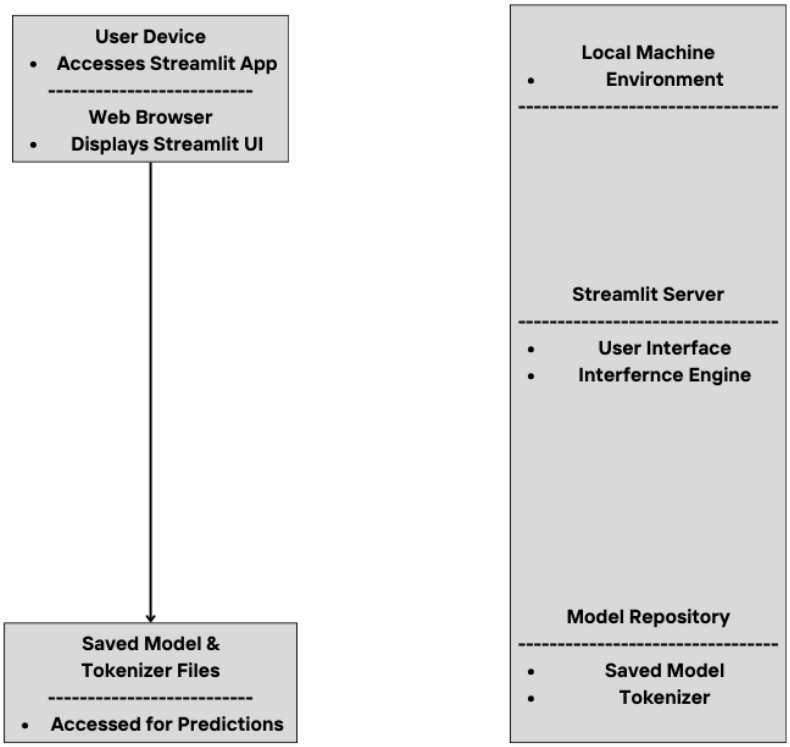


Figure 11 Deployement Diagram

4.9. Data Flow diagram

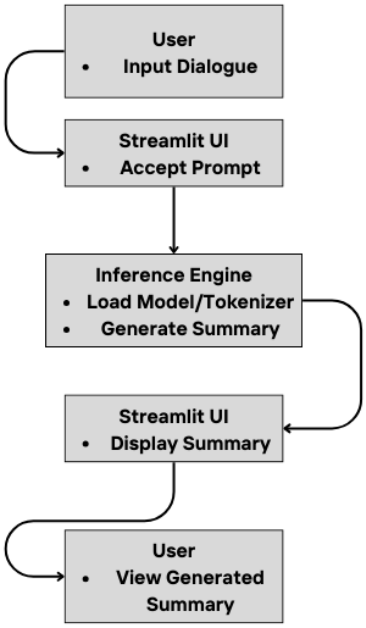


Figure 12 Data Flow Diagram

Chapter 5
Implementation

Chapter 5: Implementation

This process of implementation we used on Google Colab Engine that gives us T4 GPU and RAM scores to process fast as soon or as much as possible includes developing the core functionality of the system, such as integrating the PEGASUS model for abstractive summarization and implementing the necessary preprocessing steps for text analysis. Additionally, the implementation includes creating a user-friendly interface for users to interact with the system.

5.1. Important Flow Control/Pseudo codes

```
BEGIN Text Preprocessing
  LOAD IMDB Dataset

  // Convert reviews to lowercase
  FOR each review IN dataset:
    review = review.lower()

  // Remove HTML tags
  FOR each review IN dataset:
    review = remove_html_tags(review)

  // Remove URLs from the reviews
  FOR each review IN dataset:
    review = remove_url(review)

  // Remove punctuation from the reviews
  FOR each review IN dataset:
    review = remove_punctuation(review)

  // Replace chat words with full forms
  FOR each review IN dataset:
```

```
        review = chat_conversation(review)

    // Correct misspelled words
    FOR each review IN dataset:
        review = correct_text(review)

    // Remove stopwords from the reviews
    FOR each review IN dataset:
        review = remove_stopwords(review)

    // Remove emojis from the reviews
    FOR each review IN dataset:
        review = remove_emojis(review)

    // Tokenize the reviews
    FOR each review IN dataset:
        words = tokenize_words(review)

    // Apply stemming
    FOR each review IN dataset:
        review = stem_words(review)

    // Apply lemmatization
    FOR each review IN dataset:
        review = lemmatize_words(review)

END Text Preprocessing

BEGIN Model Setup and Summarization
    // Load the dataset
    dataset = load_dataset("cnn_dailymail", "3.0.0")
```

```
    // Load pre-trained models and tokenizers
    models = {
        "gpt2-medium": load_model("gpt2-medium"),
        "t5-base": load_model("t5-base"),
        "bart-large-cnn": load_model("facebook/bart-large-cnn"),
        "pegasus-cnn_dailymail": load_model("google/pegasus-cnn_dailymail")
    }

    // Summarize the input text using each model
    FOR each model IN models:
        summary = model.summarize(dataset['train'][1]['article'][:2000])
        store summary in summaries dictionary

END Model Setup and Summarization



BEGIN Model Evaluation
    // Load ROUGE metric
    rouge_metric = load("rouge")

    // For each model, compute the ROUGE scores
    FOR each model IN summaries:
        rouge_metric.add(prediction=summaries[model],
reference=dataset['train'][1]['highlights'])
        score = rouge_metric.compute()
        store score in results

    // Display ROUGE scores in a DataFrame
    create_dataframe_from_results(results)
END Model Evaluation
```

```
BEGIN Model Training
   // Initialize the model and tokenizer
   model_ckpt = "google/pegasus-cnn_dailymail"
   tokenizer = load_tokenizer(model_ckpt)
   model = load_model(model_ckpt)

   // Prepare dataset for training
   dataset_samsum = load_dataset("samsum")
   prepared_dataset = prepare_dataset_for_training(dataset_samsum)

   // Configure training parameters
   training_args = configure_training_args()

   // Initialize Trainer
   trainer = Trainer(model=model, args=training_args, data_collator=seq2seq_data_collator,
train_dataset=prepared_dataset["train"], eval_dataset=prepared_dataset["validation"])

   // Start training
   trainer.train()

END Model Training

BEGIN Model Evaluation on Test Dataset
   // Split dataset into smaller batches
   batch_size = 16
   article_batches, target_batches = split_dataset_into_batches(dataset['test'], batch_size)

   // For each batch, generate summary using model
   FOR each batch IN article_batches:
      summaries = model.generate(input_ids=inputs, attention_mask=attention_mask)
      store decoded_summaries in results
```

```
    // Compute ROUGE scores
    score = calculate_rouge_score(results)
    print_rouge_scores(score)

END Model Evaluation on Test Dataset



BEGIN Save Model and Tokenizer
    // Save the trained model and tokenizer to a directory
    save_model(model, "saved_model_directory")
    save_tokenizer(tokenizer, "saved_model_directory")
END Save Model and Tokenizer

BEGIN Prediction
    // Load saved model and tokenizer
    model = load_model("saved_model_directory")
    tokenizer = load_tokenizer("saved_model_directory")

    // Take user input via Streamlit
    user_input = get_user_input("Type your Dialogue here:")

    IF user presses "Submit":
        // Tokenize user input
        inputs = tokenizer.encode("summarize: " + user_input)

        // Generate summary using the model BEGIN Streamlit Interface
    SET title to "LLM Text Summarization"

    // Input text area for user
    DISPLAY text_area("Type your Dialogue here:")

    // When button is clicked
```

```
   IF "Submit" button pressed:
      SHOW spinner("Generating response...")
      GENERATE response using model
      DISPLAY the response
END Streamlit Interface       summary = model.generate(inputs)

      // Decode and display the summary
      display_summary(summary)
END Prediction
```

5.2. Components, Libraries, Web Services and stubs

Implementing the LLM text summarization system will rely on various components and libraries like NumPy, pandas, matplotlib, seaborn, sklearn, and TensorFlow. This project is based on libraries such as transformers that can access models and tokenization, also known as natural language processing. And for the web, we used streamlit

5.3. Deployment Environment

In the deployment of our project, we used a Python library called Streamlit as a deployment environment, a prevalent framework for making machine learning, deep learning, and other web applications easy to use web applications rather than creating a completed front end using different programming languages like HTML, CSS, and JavaScript. Streamlit is a dynamic rendering for components based on input updates

5.4. Tools and Techniques

This project leverages various platform tools and techniques for better or successful development and deployment. Python will be used as a programming language with the help of its libraries, such as natural language processing, transformers, sci-kit learn, TensorFlow, and Streamlit, and it will be used for building web applications and friendly user interfaces. On the other hand, the git command and GitHub platform facilitate the version control system.

5.5. Best Practices / Coding Standards

Implementing significant language text summarization will be the best practice; you can use Google Colab. On the other hand, you can use the Kaggle platform for best practices and coding standards for their models to work to and contribute to your coding competitions for the best efficiency, maintainability, and scalability. Following the Python environment will guide you in ensuring a consistent code style for your model design that promotes your code.

5.6. Version Control

We used version control for our project text summarization, which will be managed by the system using git commands, a widely used version control system distributed. Git gives a robust platform for changing our codes and collaborating with our team members who participate in our project and can manage different versions of it. The repository on this platform, such as GitHub, can be accessed using the link provided, which offers features like tracking, some request, pull and push, or other tools. This will enable the developers to work occasionally and update track changes. Git branching, such as git-flow, manages the development and manageability of the extensive language model text summarization system to its lifecycle.

Chapter 6

Testing and Evaluation

Chapter 6: Testing and Evaluation.

This chapter of testing and evaluation methods are used in our development of text summarization project. Various testing is applied to ensure that the performance we need is fulfilled and good for user inputs as friendly.by covering many testing approaches. Like case testing and boundary analysis, this chapter gives a comprehensive evaluation of robustness. Additionally, we used extreme usage conditions.

6.1. Use Case Testing

Use case testing that define system functions correctly based predefined user ideas.Each use case,such as giving prompt or get a summary, is esure to testing the system behavior as we expected.The test cases are gives directly from user needs,simulation and modeling in real-world interaction with streamlit app.This testing validates both typical and alternative workflows to confirmed that our product is user friendly and error free

6.2. Equivalence partitioning

This is using to divided our data into different things called categories that should create similar output data.For the summarization LLM,our model getting different types of prompt (e.g., short, medium, and long summary) are used as prompt input to confirm that all prompt variations is accurate and easily readable.By testing these inputs from each categories that helps to identify any error and issues without need to exhausting every need input

6.3. Boundary value analysis

Boundary value analysis focuses on testing the system at the edge limits of equivalence classes. This is especially relevant for input size limitations, such as character or word count in dialogue input. Testing boundary values, such as the minimum and maximum allowable input lengths, ensures that the system can handle inputs at the edges of acceptable ranges without errors or crashes. This technique helps prevent unexpected behavior when users provide inputs at or near the defined limits

6.4. Data flow testing

Data flow testing checks the system's handling of data as it moves through various operations, including loading, processing, and displaying text summaries. For this project, data flow testing ensures that dialogue input is correctly tokenized, summarized by the model, and displayed accurately. This testing type verifies that each stage of data processing, from input to output, functions smoothly and without data corruption or loss.

6.5. Unit testing

Unit testing involves testing individual components of the system to verify their correct functionality in isolation. Key functions, such as loading the model, tokenizing text, and generating summaries, are tested independently to ensure they work as expected. Unit tests confirm that each component performs its task accurately, providing a stable foundation for integrating components into the overall system

Table 2 Models Accuracy

Models	Rouge1	Rouge2	Rouge3	RougeLsum
gpt2-medium-380M	0.212389	0.018018	0.159292	0.194690
t5-base-223M	0.392157	0.140000	0.254902	0.313725
bart-large-cnn-400M	0.475248	0.222222	0.316832	0.396040
pegasus-cnn-568M	0.476832	0.232020	0.347228	0.407832

6.6. Integration testing

Integration testing examines the interaction between multiple components, such as the Streamlit UI, inference engine, and saved model. This testing ensures that all parts of the system, when combined, work together seamlessly. For instance, integration tests verify that input dialogue flows from the UI to the inference engine, resulting in accurate summary generation. Integration testing validates the complete workflow from user input to output display

6.7. Performance testing

Performance testing evaluates the system's efficiency in processing and generating summaries for various input sizes. This testing focuses on response time, memory usage, and overall resource consumption during operation. The goal is to verify that the system maintains acceptable performance levels and can generate summaries quickly and accurately under normal usage conditions. Performance testing identifies any bottlenecks that may affect user experience.

6.8. Stress Testing

Stress testing involves pushing the system to its limits with exceptionally large inputs or rapid sequences of requests. This type of testing assesses the system's robustness and stability under high-load conditions. For the text summarization system, stress testing simulates a large number of concurrent requests to evaluate the model's ability to handle peak loads without failure. Stress testing helps to identify potential weaknesses that could impact reliability in high-demand situations.

Chapter 7

Summary, Conclusion and Future Enhancements

Chapter 7: Summary, Conclusion & Future Enhancements

7.1. Project Summary

This project aimed to develop a robust text summarization system that leverages machine learning to generate concise summaries from dialogue inputs. The project combined Google Colab for model training, Streamlit for the user interface, and VS Code for local deployment, resulting in a comprehensive framework for summarization. Key features include efficient dialogue processing, model inference using a pre-trained Pegasus model, and a user-friendly interface where summaries are generated and displayed in real-time. By addressing user requirements for accurate and readable summaries, this project offers a practical application in areas such as customer service, content summarization, and educational tools.

7.2. Achievements and Improvements

The project successfully achieved several milestones, including the integration of the Pegasus model for high-quality text summarization, a stable data flow pipeline, and a streamlined interface for user interaction. Performance improvements were made in terms of response time and memory usage by optimizing the model loading and inference processes. Additionally, the system was tested for various input lengths and complexity levels, ensuring accurate and readable outputs across a wide range of use cases. Each component was fine-tuned and validated, resulting in a robust and reliable text summarization system.

7.3. Critical Review

Despite its achievements, the project faced some challenges. Training a large model like Pegasus required significant computational resources, which sometimes limited the depth of experimentation with model parameters. Additionally, managing different processing environments (Colab for training and VS Code for deployment) introduced occasional compatibility issues. Another challenge was optimizing the model for faster real-time

inference, especially for longer dialogues. While solutions were implemented to mitigate these issues, they highlight areas for potential improvement in future iterations of the project.

7.4. Lessons Learnt

Throughout this project, valuable lessons were learned in areas of machine learning model integration, data preprocessing, and system optimization. A key takeaway was the importance of efficient resource management during training and inference, especially when working with large models. The project also underscored the significance of testing for real-world conditions, as well as the need for modular design to facilitate testing and maintenance. Finally, the project reinforced the value of thorough planning in multi-environment workflows, helping to ensure smooth transitions between model training, deployment, and user interaction.

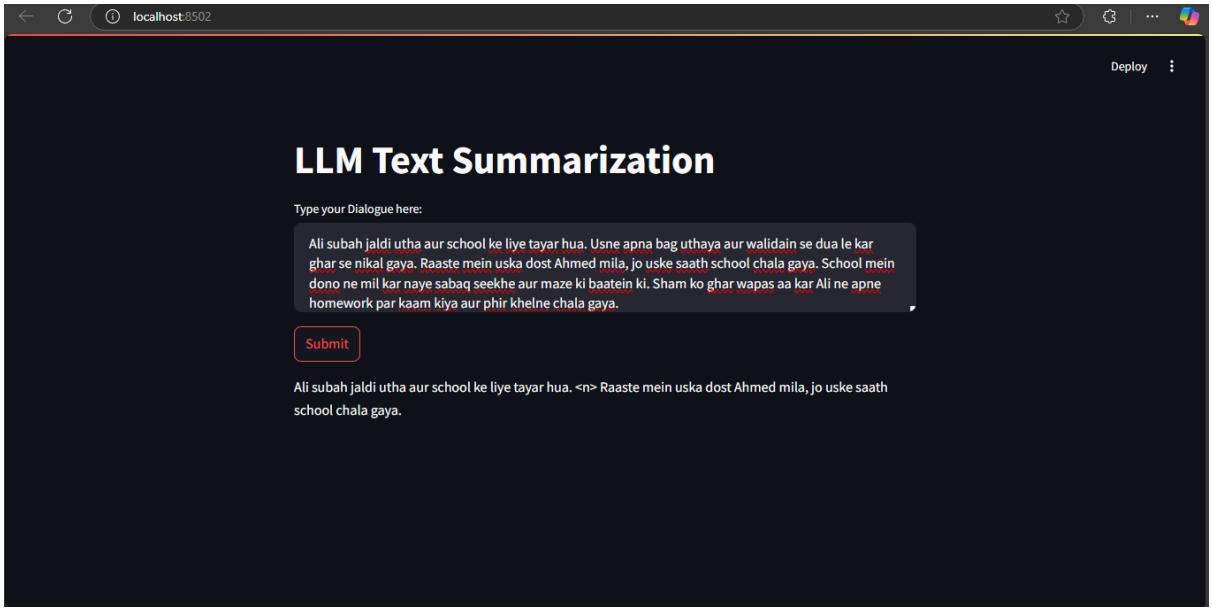
7.5. Future Enhancements/Recommendations

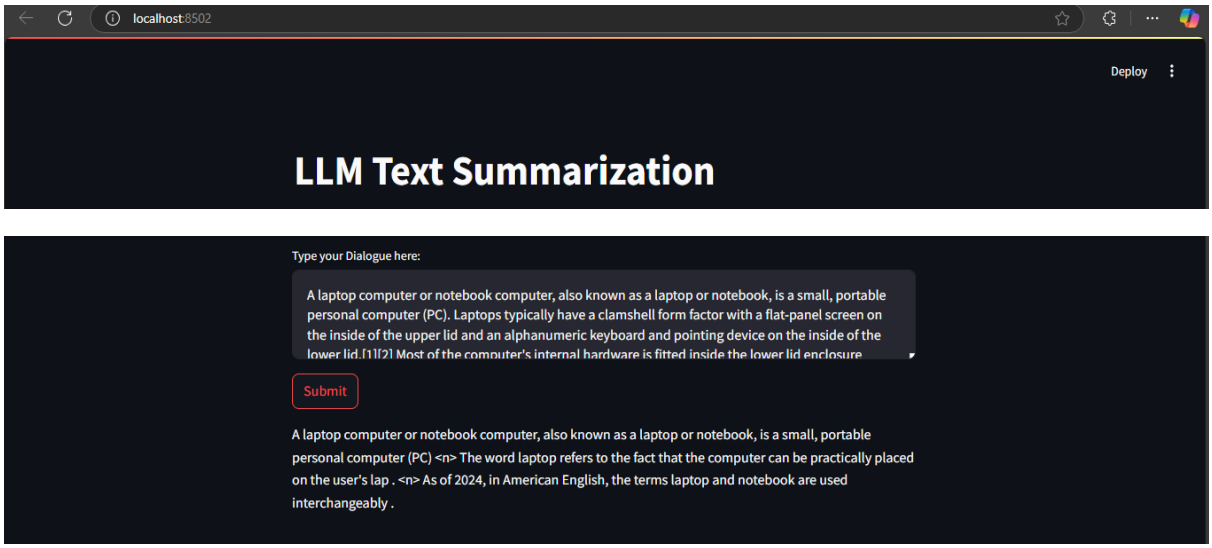
To enhance the project's functionality and efficiency, several future improvements are recommended. First, using a more advanced deployment platform with GPU support would enable faster real-time inference for larger models. Incorporating a model fine-tuning interface in Streamlit could allow for ongoing model refinement based on user feedback, improving accuracy. Additional support for multi-language summarization would also broaden the system's applicability. Finally, implementing further optimization techniques, such as model pruning or quantization, could reduce memory usage and increase response time, making the system more suitable for real-time applications

Appendices

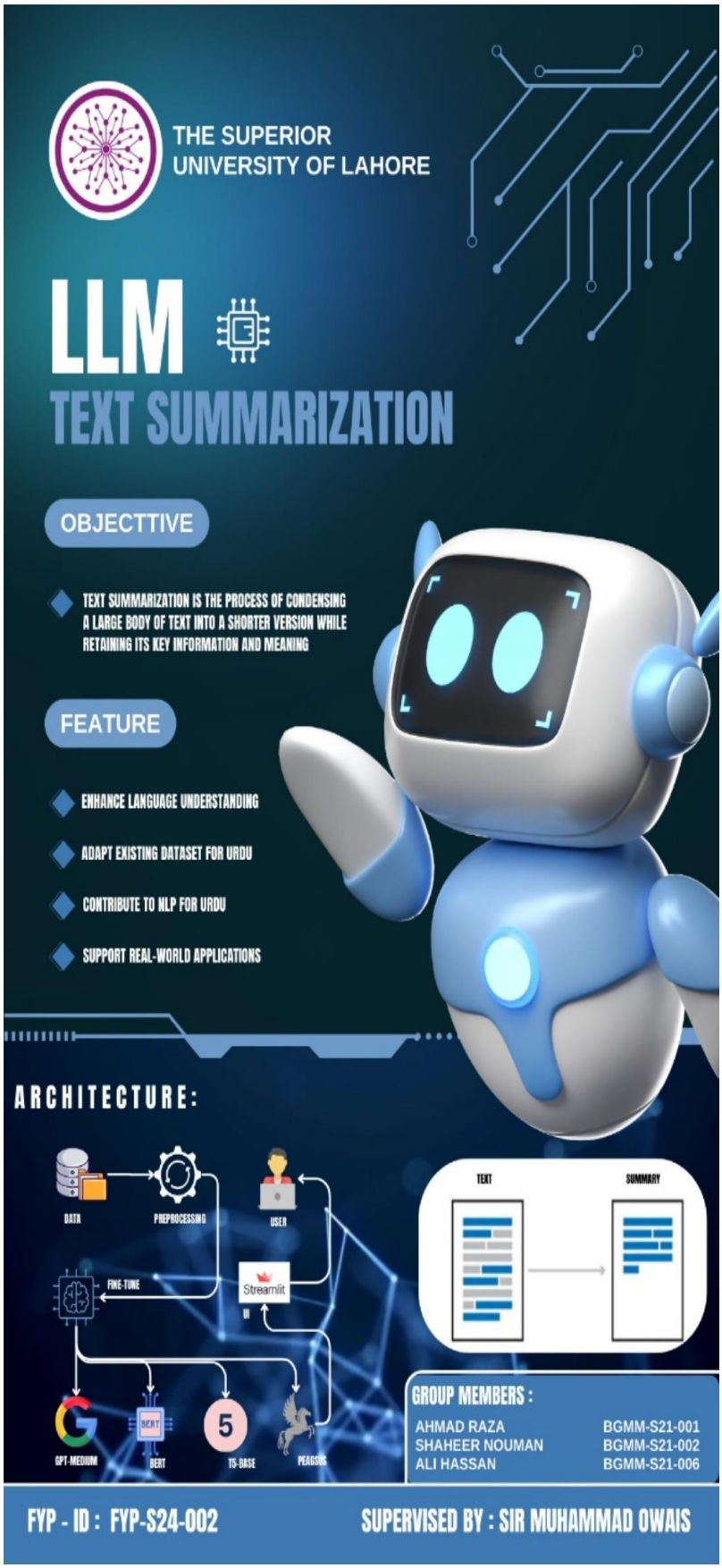
Appendix A:

A.1. UI





A.2. Standee


THE SUPERIOR UNIVERSITY OF LAHORE
LLM
TEXT SUMMARIZATION
OBJECTTIVE
TEXT SUMMARIZATION IS THE PROCESS OF CONDENSING A LARGE BODY OF TEXT INTO A SHORTER VERSION WHILE RETAINING ITS KEY INFORMATION AND MEANING
FEATURE
ENHANCE LANGUAGE UNDERSTANDING
ADAPT EXISTING DATASET FOR URDU
CONTRIBUTE TO NLP FOR URDU
SUPPORT REAL-WORLD APPLICATIONS
ARCHITECTURE:
DATA
PREPROCESSING
USER
FINE-TUNE
Streamlit
UI
GPT-MEDIUM
BERT
T5-BASE
PEAGSUS
TEXT
SUMMARY
GROUP MEMBERS :
AHMAD RAZA BGMM-S21-001
SHAHEER NOUMAN BGMM-S21-002
ALI HASSAN BGMM-S21-006
FYP - ID : FYP-S24-002
SUPERVISED BY : SIR MUHAMMAD OWAIS

Reference and Bibliography

Reference and Bibliography

1] Song, Shengli, Haitao Huang, and Tongxiao Ruan."Abstractive text summarization using LSTM-CNN based deep learning." Multimedia Tools and Applications 78 (2019): 857-875.

[2] Hanunggul, Puruso Muhammad, and Suyanto Suyanto."The impact of local attention in lstm for abstractive text summarization." 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI). IEEE, 2019.

[3] Zolotareva, Ekaterina, Tsegaye Misikir Tashu, and Tomás Horváth. "Abstractive Text Summarization using Transfer Learning." ITAT. 2020.

[4] Ranganathan, Jaishree, and Gloria Abuka. "Text summarization using a transformer model." 2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS). IEEE, 2022

[5] Zhang, Jingqing, et al. "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization." International Conference on Machine Learning. PMLR, 2020.

[6] Lalitha, Evani, et al. "Text Summarization of Medical Documents using Abstractive Techniques." 2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC). IEEE, 2023.

[7] Bohra, Mrinmoi, Pankaj Dadure, and Partha Pakray. "Comparative analysis of T5 model for abstractive text summarization on different datasets." (2022).

[8] Gupta, Anushka, et al. "Automated news summarization using transformers." Sustainable Advanced Computing: Select Proceedings of ICSAC 2021. Singapore: Springer Singapore, 2022. 249-259.

[9] Yadav, Hemant, Nehal Patel, and Dishank Jani. "Fine-Tuning BART for Abstractive Reviews Summarization." Computational Intelligence: Select Proceedings of InCITe 2022. Singapore: Springer Nature

Singapore, 2023. 375-385.

[10] Rehman, T., Das, S., Sanyal, D. K., & Chattopadhyay, S. (2022, August). An Analysis of Abstractive Text Summarization Using Pre-trained Models. In Proceedings of International Conference on Computational Intelligence, Data Science and Cloud Computing: IEM-ICDC 2021 (pp. 253-264).

[11] Narayan, Shashi, Shay B. Cohen, and Mirella Lapata. "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization." arXiv preprint arXiv:1808.08745 (2018).

[12] Hermann, Karl Moritz, et al. "Teaching machines to read and comprehend." Advances in neural information processing systems 28 (2015).

[13] McAuley, Julian John, and Jure Leskovec. "From amateurs to connoisseurs: modeling th evolution of user expertise through online reviews." Proceedings of the 22nd international conference on World Wide Web. 2013.

[14] Xue, Hao, Du Q. Huynh, and Mark Reynolds. "SS- LSTM: A hierarchical LSTM model for pedestrian trajectory prediction." 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018.

[15] Lewis, Mike, et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation,translation, and comprehension." arXiv preprint arXiv:1910.13461 (2019).

[16] Ahmed, Rafi, et al. "An overview of Pegasus." Proceedings RIDE-IMS93: Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems. IEEE, 1993.

[17] Carmo, Diedre, et al. "Ptt5: Pretraining and validating the t5 model on Brazilian Portuguese data." arXiv preprint arXiv:2008.09144 (2020).

[18] Brown, T. B., et al. "Language Models are Few-Shot Learners." arXiv preprint arXiv:2005.14165 (2020).

[19] Vaswani, A. et al. "Attention is All You Need." Advances in Neural Information Processing Systems. 2017.

[20] Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media, Inc.

[21] Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing (3rd ed.). Pearson.

[22] Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research, 57, 345-420.

[23] Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press.

[24] Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. arXiv preprint arXiv:1706.05098.

[25] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1798-1828.

[26] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[27] Chollet, F. (2017). Deep Learning with Python. Manning Publications.

Index

Index