MINI PROJECT REPORT

ON

TEXT SUMMARIZATION USING LLM

Submitted to

Department of Computer Science & Engineering

KESHAV MEMORIAL ENGINEERING COLLEGE

(Affiliated to Osmania University)

Submitted by

MANCHALA SAI VENKATA **245521733303**

KRISHNA BHARGAV

Under the Guidance of

Dr. D.V.S.S SUBRAHMANYAM

HoD , Dept. of CSE

KESHAV MEMORIAL ENGINEERING COLLEGE

(Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad)

D.No. 10 TC-111, Kachavanisingaram (V), Ghatkesar (M), Medchal-Malkajgiri, Telangana – 500088

(2023-2024)

KESHAV MEMORIAL ENGINEERING COLLEGE

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project report titled **TEXT SUMMARIZATION USING LLM** that is submitted by Mr. **MANCHALA SAI VENKATA KRISHNA BHARGAV (245521733303)** under the guidance of **Dr. D.V.S.S SUBRAHMANYAM, HoD CSE Department** as per the requirements of the course curriculum in VI Semester of **B.E (CSE) of Osmania University** is a bonafide work carried out by the above said student under my guidance and supervision..

Dr. D.V.S.S SUBRAHMANYAM
HoD,
Internal Guide

Dr. D.V.S.S SUBRAHMANYAM
HoD

EXTERNAL EXAMINER

Submitted for Viva voce Examination held on _____

DECLARATION

This is to certify that the mini project titled **TEXT SUMMARIZATION USING LLM** is a bonafide work done by us in fulfillment of the requirements for the award of the degree **Bachelor of Engineering** in Department of Computer Science and Engineering, and submitted to the **Department of CSE, Keshav Memorial Engineering College, Hyderabad.**

I also declare that this project is a result of my own effort and has not been copied or intimated from any source. Citations from any websites are mentioned in the bibliography. This work was not submitted earlier at any other university for the award of any degree.

MANCHALA SAI VENKATA KRISHNA BHARGAV (245521733303)

ACKNOWLEDGEMENT

This is to place on our record my appreciation and deep gratitude to the persons without whose support this project would never been this successful.

We are grateful to **Mr. Neil Gogte,** Founder Director, for facilitating all the amenities required for carrying out this project.

It is with immense pleasure that we would like to express our indebted gratitude to the respected **Prof. P.V.N Prasad, Principal, Keshav Memorial Engineering College,** for providing great support and for giving us the opportunity of doing the project.

We express our sincere gratitude to **Mrs. Deepa Ganu,** Director Academics, for providing an excellent environment in the college.

We would like to take this opportunity to specially thank to **Dr. D.V.S.S.SUBRAHMANYAM, HoD, Department of CSE, Keshav Memorial Engineering College,** for inspiring us all the way and for arranging all the facilities and resources needed for our project.

We would like to take this opportunity to thank our internal guide Dr. D.V.S.S SUBRAHMANYAM**, HoD, Department of CSE, Keshav Memorial Engineering College,** who has guided us a lot and encouraged us in every step of the project work. His valuable moral support and guidance throughout the project helped us to a greater extent.

We would like to take this opportunity to specially thank our Project Coordinator, Dr. D.V.S.S SUBRAHMANYAM**, HoD, Department of CSE, Keshav Memorial Engineering College,** who guided us in successful completion of our project.

Finally, we express our sincere gratitude to all the members of the faculty of Department of CSE, our friends and our families who contributed their valuable advice and helped us to complete the project successfully.

MANCHALA SAI VENKATA KRISHNA BHARGAV (245521733303)

CONTENTS

PROBLEM STATEMENT

Develop a comprehensive application that utilizes open-source large language models (LLMs) to process and analyze various types of data inputs, including PDF files, images containing text, and plain text files. The application will provide two main functionalities:

1. Text Summarization
2. Question Answering (QA) Chatbot

ABSTRACT

Text summarization is a critical task in natural language processing (NLP) that seeks to condense large volumes of text into concise, informative summaries, preserving essential information and key insights. This project, "Text Summarization and Question Answering Using Open-Source Large Language Models (LLMs)," leverages advanced machine learning techniques and state-of-the-art LLMs to develop a powerful, versatile tool for both text summarization and interactive question answering (QA).

The application is designed to process various types of text input, including PDFs, images containing text, and plain text files, and provide summaries in English. It utilizes open-source LLMs, ensuring efficiency, accuracy, and flexibility in handling text summarization tasks. The tool also features an interactive QA chatbot that allows users to ask questions related to the uploaded content and receive precise answers, enhancing user engagement and providing quick access to specific information.
Key Features include PDF and Image Summarization, Text Summarization and Interactive Q&A.

This project aims to simplify the extraction of meaningful summaries from extensive documents and images, providing valuable tools for researchers, students, and professionals to manage information overload and enhance productivity.

TEXT SUMMARIZATION USING LLM

1. INTRODUCTION

1.1 Introduction about the Concept

This project aims to streamline text processing by developing an application that integrates advanced Natural Language Processing (NLP) techniques for text summarization, Optical Character Recognition (OCR) for extracting text from images, and grammar correction tools. The core functionality allows users to upload various file types, including PDFs and images, to extract, summarize, and correct the content in a seamless manner. By leveraging cutting-edge AI models, the application ensures that the text is not only condensed but also grammatically accurate.

A comprehensive tool designed to meet the diverse summarization and text processing needs of non-internet-connected networks. In today's fast-paced world, access to relevant information is crucial, even in environments where internet connectivity is limited or unavailable. Our application addresses this challenge by providing advanced AI/ML-driven text summarization capabilities, all within a self-contained offline environment.

1.2 Existing System and Disadvantages

Current text processing tools typically focus on a specific task, such as OCR, text summarization, or grammar correction, without offering a unified platform. Users often face inefficiencies, such as the manual transfer of text between different applications, limited input format support, and lack of integration between these functionalities. For example:

1. Tools may support only PDF summarization without OCR capabilities.
2. Existing systems may require users to manually correct grammar after summarization, leading to fragmented workflows.
3. Text extraction from images is often not supported alongside other text processing tasks, limiting the flexibility of the tools.

1.3 Literature Review

The development of this application builds upon several advancements in NLP and computer vision:

1. T5 Transformer Models for Summarization: T5 (Text-To-Text Transfer Transformer) is a versatile NLP model known for its effectiveness in text summarization, enabling concise and informative summaries.

2. Optical Character Recognition (OCR) Technologies: Tools like Tesseract have been extensively studied for their ability to accurately extract text from images, making them critical in applications requiring image-to-text conversion.

1.4 Proposed System and Advantages

The proposed system offers a comprehensive solution that combines multiple text processing tasks into a single, user-friendly platform:

1. Integrated workflow: The application provides a unified interface that supports OCR, summarization, and grammar correction, eliminating the need to switch between multiple tools.
2. Efficiency: By automating the process from text extraction to summarization and grammar correction, the system significantly reduces the time and effort required to process documents.
3. Accuracy: Utilizing state-of-the-art NLP models and OCR technologies ensures high accuracy in both summarization and text extraction, while the grammar correction tool enhances the clarity and correctness of the output.
4. Flexibility: The application supports various file formats, including PDFs and images, allowing users to handle diverse text sources in one platform.

This all-in-one solution is designed to simplify text processing tasks for researchers, students, and professionals, providing an efficient and accurate tool to manage and summarize large volumes of text.

2. SYSTEM ANALYSIS

2.1 Feasibility Study

The feasibility study confirms that the integration of OCR, PDF text extraction, and NLP summarization is both technically and economically viable. The use of robust open-source tools and models ensures that the solution can be effectively implemented and offers significant benefits for users needing comprehensive text processing capabilities.

2.2 System Requirements

2.2.1. Hardware Requirements

The application is designed to run efficiently on standard computing hardware with the following minimum requirements:

1. Processor: Intel i5 or equivalent; minimum 2 GHz, Quad-Core.

2. RAM: 8 GB or higher.

3. Storage: 20 GB of free disk space.

4. GPU (Optional): NVIDIA GPU with CUDA support for faster model inference (recommended for improved performance).

2.2.2. Software Requirements

The project utilizes a range of software tools and libraries to implement the text processing functionalities:

1. **Operating System:** Windows 10 or higher, or any modern Linux distribution.

2. Programming Language: Python 3.8 or higher.

3. Libraries and Frameworks:

-Streamlit: For building the interactive web application.

-PyTorch: For machine learning and model inference.

-Transformers: For utilizing pre-trained NLP models.

-Tesseract OCR: For extracting text from images.

-PyPDF2: For handling PDF files.

2.2.3. Functional Requirements

The system must support the following core functionalities:

1. File Handling:

- Ability to upload and process PDF, image, and text files.
- Extract and summarize text from uploaded files.

2. Text Processing:

- Perform OCR on images to extract text accurately.
- Summarize text data with configurable summary lengths.

2.2.4. Non functional Requirements

The application should meet the following non-functional requirements:

1. Performance:

- Provide summaries and process text within a reasonable time (under 30 seconds for a standard document).
- Ensure high accuracy in text extraction and summarization.

2. Usability:

- Offer a fast and responsive user interface.
- Provide a user-friendly experience with minimal training required.

3. Security:

- Secure handling of uploaded files to prevent data leakage and ensure privacy.

4. Scalability:

- Handle multiple concurrent users effectively without performance degradation.

3. SYSTEM DESIGN

3.1 Introduction

The system is designed as a modular Streamlit application, integrating distinct functionalities for file handling, text extraction, summarization, and grammar correction. This modular architecture ensures the system is scalable, maintainable, and efficient.

3.2. Modules and Description

The system comprises the following modules:

1. File Handling Module:

- Purpose: Manages file uploads and validates supported file types (PDFs, images, text files).
- Description: Allows users to upload various file formats and initiates further processing based on the file type.

2. OCR Module:

- Purpose: Extracts text from images.
- Description: Utilizes Tesseract OCR to convert text in image files into machine-readable text.

3. PDF Processing Module:

- Purpose: Extracts text from PDF files.
- Description: Uses PyPDF2 to handle PDF documents and extract text for further processing.

4. Summarization Module:

- Purpose: Generates concise summaries from text.
- Description: Employs the T5 Transformer model to produce summaries from the extracted or provided text.

5. Grammar Correction Module:

- Purpose: Checks and corrects grammar and vocabulary.
- Description: Integrates LanguageTool to analyze and correct grammatical errors and improve text quality.

6. UI Module:

- Purpose: Provides user interaction.
- Description: Built using Streamlit, this module facilitates a user-friendly interface for uploading files, viewing results, and interacting with the application.

3.3 Block Diagram

The block diagram outlines the flow of data through the system:

1. User Input:

- Users upload files (PDFs, images, text).

2. File Handling:

- Files are received and validated.

3. Text Extraction:

- For images: Text is extracted using the OCR module.
- For PDFs: Text is extracted using the PDF Processing module.

4. Text Processing:

- Summarization: Extracted or provided text is summarized using the Summarization module.
- Grammar Checking: Text is checked and corrected using the Grammar Correction module.

5. Output Display:

- Summarized text or corrected text is displayed to the user through the UI module.

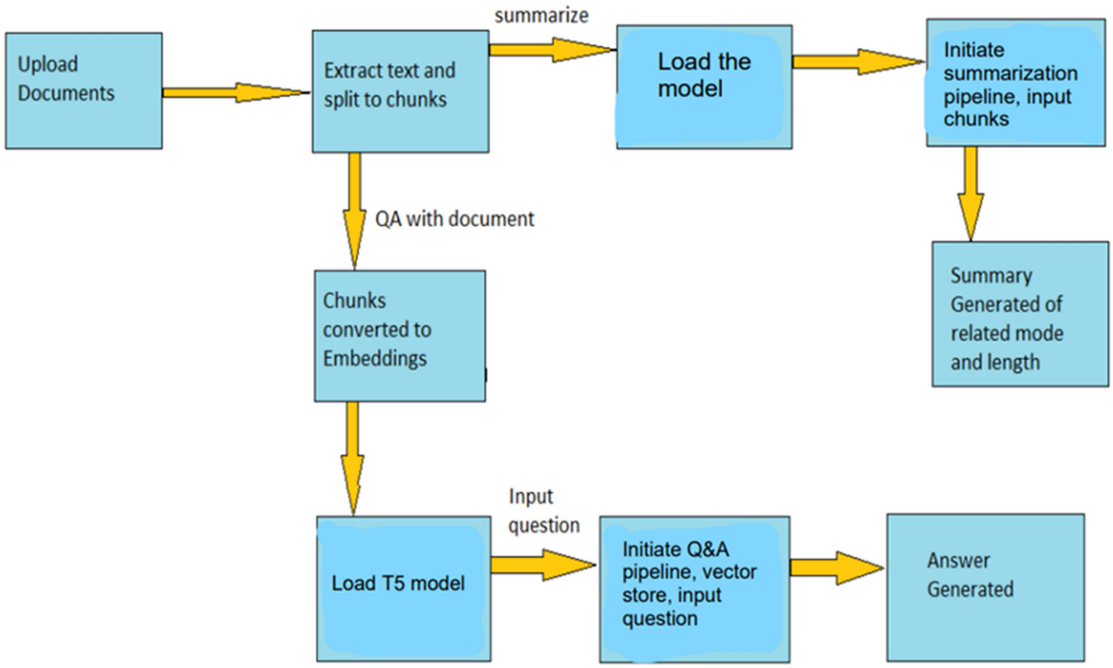

Figure 3.1 – Implementation / Workflow

3.4. UML Diagrams

UML (Unified Modeling Language) is a structured modelling language that consists of an interconnected collection of diagrams that was created to assist system and software developers in defining, visualizing, designing, and recording software system objects, as well as business modelling and other non-software systems. The UML is a critical component of object-oriented software creation and the software development process. A UML diagram is a diagram based on the UML (Unified Modeling Language) that is used to visually represent a system, including its key actors, functions, behavior, objects, or classes, in order to better understand, modify, retain, or record system information

3.4.1. Class Diagram

This diagram outlines the structure of the application by detailing its main classes and their methods. The `Main` class handles key functionalities such as OCR, file preprocessing, word counting, text summarization, and grammar checking. It interacts with other classes like `PdfReader`, `Speller`, and `LanguageTool` to perform specific tasks like text extraction, spell-checking, and language processing.

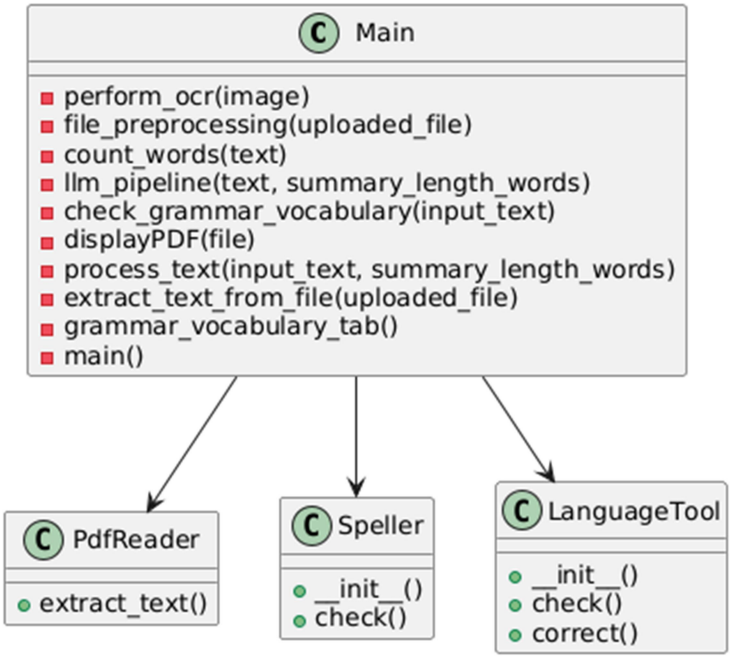

Figure 3.2 – Class diagram

3.4.2. Usecase Diagram

This diagram highlights the various use cases that the application supports, such as uploading files, extracting text, summarizing content, and checking grammar. It shows the relationship between the user and these use cases, emphasizing how different functionalities are interconnected.

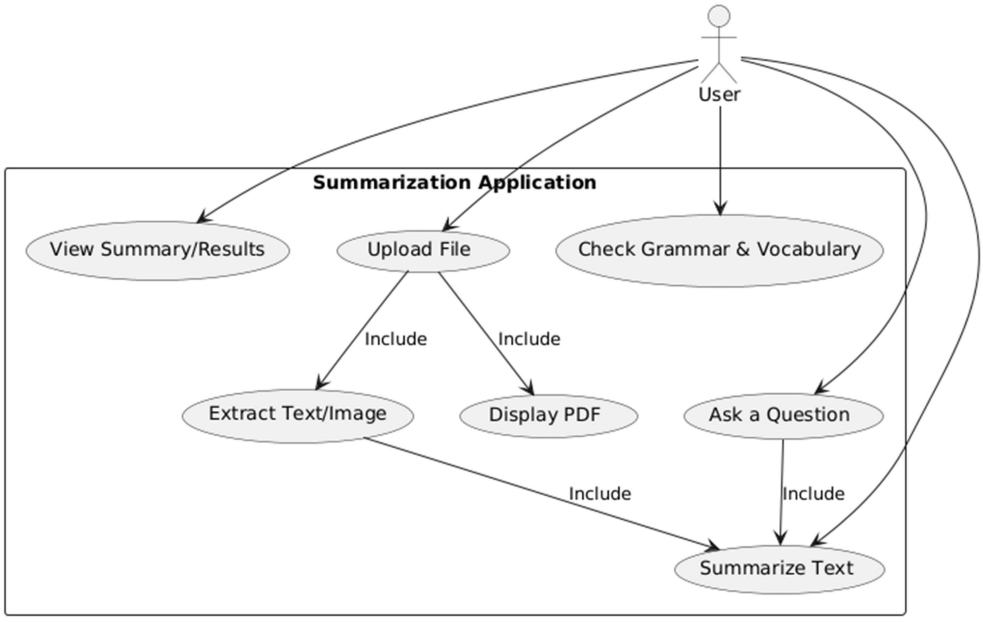

Figure 3.3 – Usecase diagram

3.4.3. Data Flow Diagram

The DFD provides a visual representation of how data moves within the application. It shows how user inputs, such as text or uploaded files, are processed through steps like text extraction, OCR, grammar checking, and summarization, leading to the generation of outputs like summaries, corrected text, or answers.

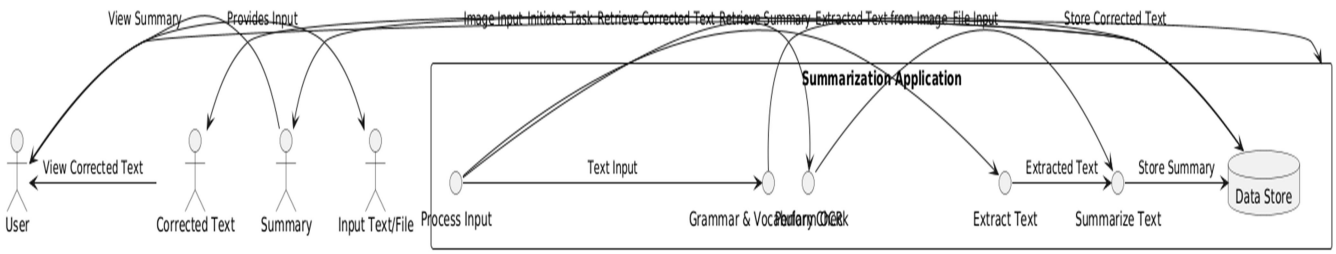

Figure 3.4 – Data flow diagram

3.4.4. Sequence diagram

This diagram illustrates the interaction between the user, the Streamlit UI, and backend components during the execution of tasks. It follows the sequence of events, such as file upload, text extraction, and summarization, showing how data flows through the system, from user input to final output.

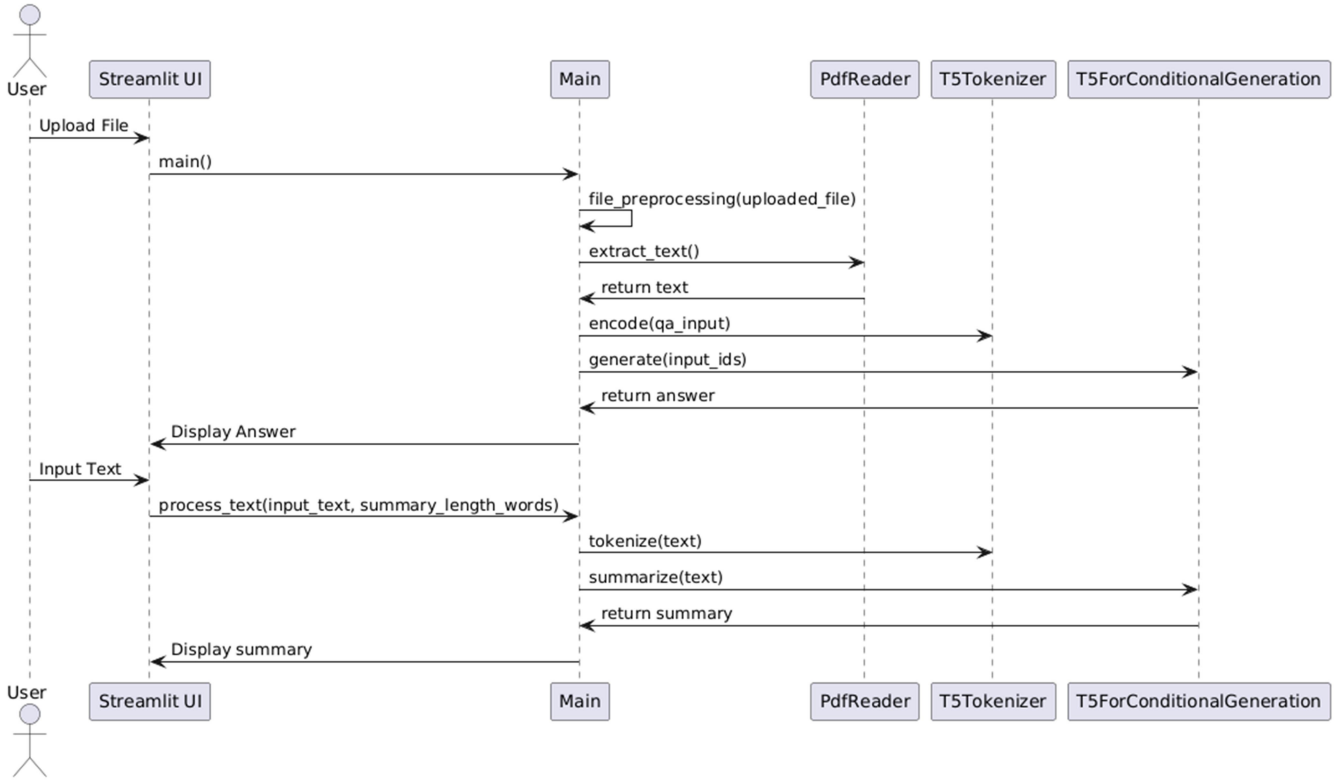

Figure 3.5 – Sequence diagram

3.4.5. Activity Diagram

This diagram represents the workflow of the application based on user-selected tasks. It starts with task selection, followed by either text summarization or grammar/vocabulary checking. Depending on the task, the diagram shows steps such as file upload, text extraction, summarization, and displaying results, capturing the decision-making process within the application.

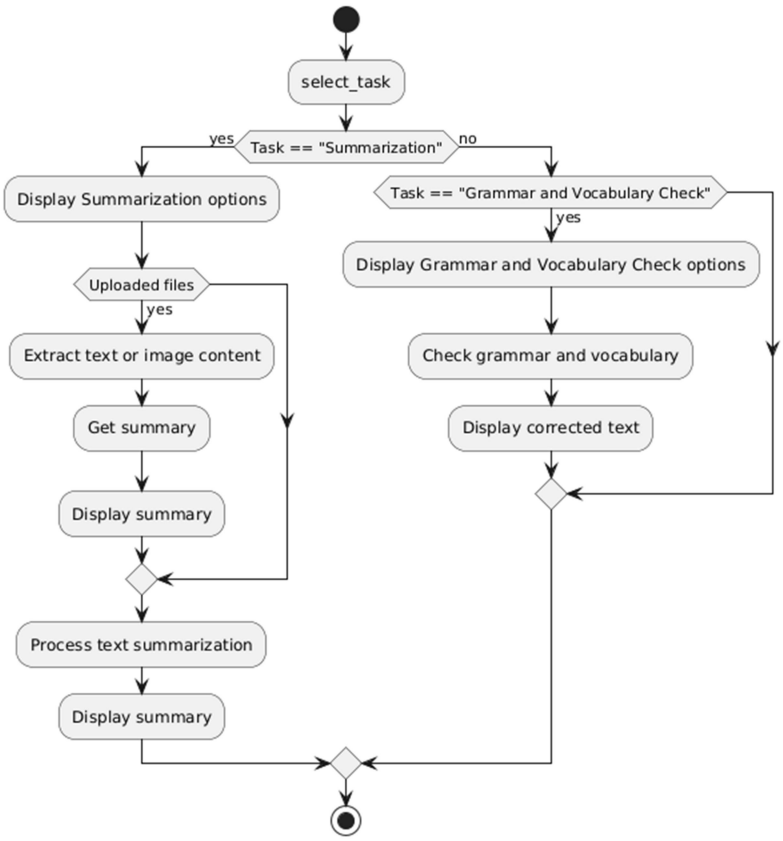

Figure 3.6 – Activity diagram

3.4.6. State Chart diagram

The state chart diagram captures the different states the application goes through based on user interactions. Starting from an idle state, it transitions through states like processing summarization, grammar checking, and displaying results, depending on the tasks selected by the user.

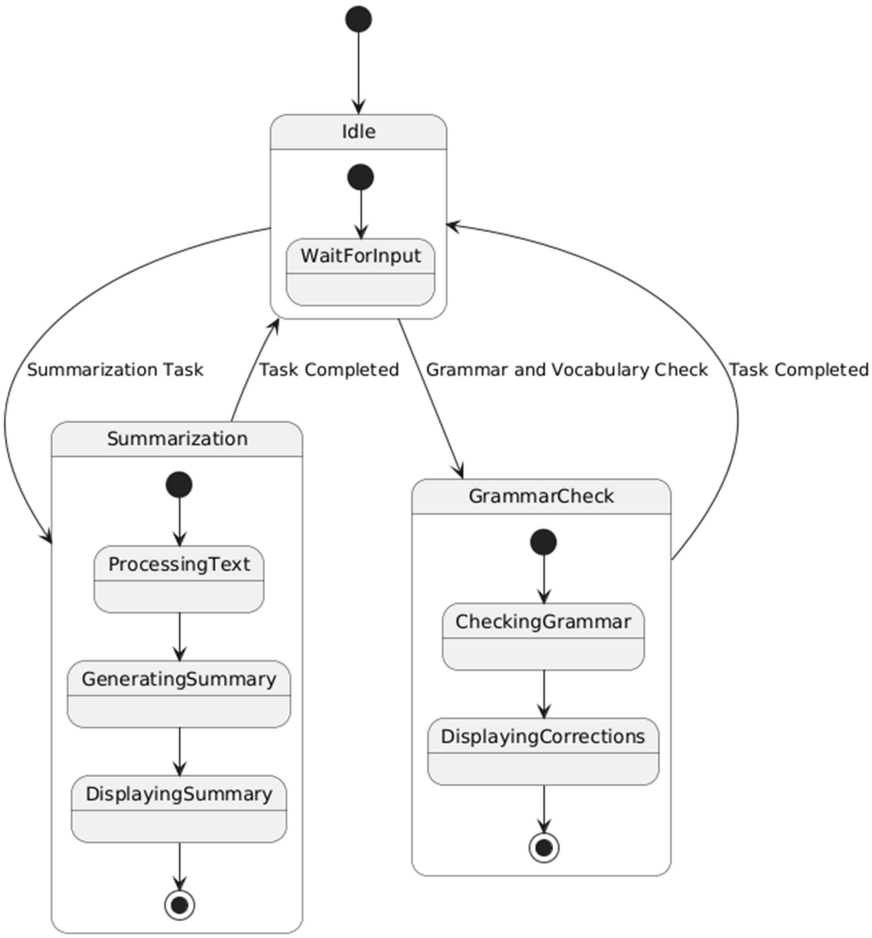

Figure 3.7 – State chart diagram

3.4.7 Component diagram

The Component Diagram showcases the modular architecture of the application, where the Streamlit Application is the central component responsible for key tasks like OCR, file preprocessing, text summarization, and grammar checking.

- It integrates with several external libraries:
  - PyPDF2: For text extraction from PDFs.
  - PIL: For image processing in OCR.
  - pytesseract: For extracting text from images via OCR.
  - transformers: For tokenization and text summarization.
  - Speller: For spell-checking.
  - language_tool_python: For grammar and vocabulary correction.

- These components work together to enable comprehensive text processing and analysis within the application.

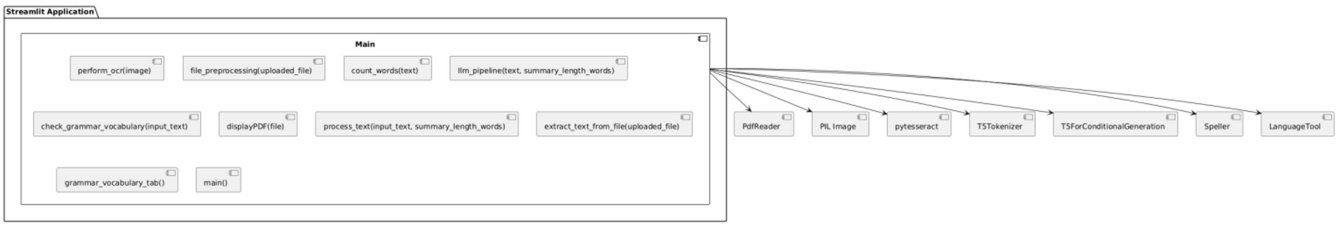

Figure 3.8 – Component diagram

3.5. Database Tables

There are no database tables in this system, as it focuses primarily on real-time processing of text and image files. The application handles data in-memory during processing and does not require persistent storage of user data or results.

4. SYSTEM IMPLEMENTATION

4.1 Description of Platform, Database, Technologies, Methods, Applications using and involving in development

1. Platform:

- The application is built on Streamlit, an open-source framework designed to simplify the creation of interactive web applications directly from Python scripts. Streamlit's straightforward approach enables rapid development and deployment, making it particularly suitable for applications involving machine learning models and data science workflows. Its reactive programming model allows the application to dynamically update based on user input, providing a seamless user experience.

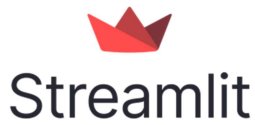

Figure 4.1 – Streamlit

2. Database:

- The system leverages in-memory storage via Streamlit's session state, effectively managing user interactions and temporary data during runtime. This method is well-suited for real-time processing tasks such as text summarization, OCR, and grammar checking, where persistent storage is unnecessary.

3. Technologies:

- **Python:** The core programming language used for developing the application. Python's extensive ecosystem of libraries and frameworks makes it the ideal choice for integrating machine learning and NLP tools.

Figure 4.2 – Python

- **Hugging Face Transformers:** The application utilizes the T5 Transformer model from Hugging Face, a leading provider of pre-trained language models. The T5 model is employed for text summarization, transforming input text into concise summaries. T5, with its 220 million parameters in the T5-Base configuration, is capable of handling diverse NLP tasks, including translation, summarization, and question answering.

Figure 4.3 – Hugging Face

Figure 4.4 – LaMini-LM Model

- Model Description: LaMini-Flan-T5-248M

LaMini-Flan-T5-248M is a fine-tuned version of the Flan-T5 model, optimized for summarization tasks. It has 248M parameters and was developed as part of the LaMini-LM model series. The model is fine-tuned with a dataset of 2.58M samples for instruction-based tasks, ensuring high-quality output while being more resource-efficient compared to larger models like GPT-3.5-turbo.

Key Features:

- Instruction Fine-Tuning: The model has been fine-tuned for instruction-based tasks, making it particularly effective in generating summaries.
- Model Distillation: Derived from larger models to retain generative capabilities in a smaller, more efficient model.

Parameters for Inference:

- Learning Rate: 0.0005
- Train Batch Size: 128
- Eval Batch Size: 64
- Seed: 42
- Gradient Accumulation Steps: 4
- Total Train Batch Size: 512
- Optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- LR Scheduler Type: Linear
- Number of Epochs: 5

- **Langchain:** A framework that enhances the multilingual capabilities of the application, allowing it to process and generate text across multiple languages. Langchain's integration ensures that the application remains versatile and accessible to a global audience.

Figure 4.5 – LangChain

-**Transformers:** A state-of-the-art NLP model architecture that represents the evolution of the encoder-decoder framework. Transformers excel at tasks involving sequence transduction, such as machine translation and text generation. They rely heavily on the **self-attention mechanism**, allowing them to focus on different parts of the input sequence simultaneously and to process tokens in parallel, leading to significant performance improvements over traditional RNN models like LSTM.

A transformer is a type of artificial intelligence model that learns to understand and generate human-like text by analyzing patterns in large amounts of text data.

- **PyTorch:** A deep learning library used to implement the T5 model within the application. PyTorch's dynamic computation graph and ease of integration with other Python libraries make it a popular choice for deploying machine learning models.

Figure 4.6 – Pytorch

- **Tesseract OCR:** An Optical Character Recognition tool used to extract text from images. Tesseract is integrated into the application to process image-based text, enabling further analysis and summarization.

- **LanguageTool:** A grammar and vocabulary checking tool that enhances the quality of text output by providing real-time feedback on grammatical errors. LanguageTool supports multiple languages, ensuring that the processed text is accurate and polished.

- **PyPDF2:** A Python library used to extract text from PDF documents, enabling the application to handle various file formats and provide consistent text processing capabilities.

4. Methods:

- OCR (Optical Character Recognition): Implemented using Tesseract OCR, this method converts text from images into machine-readable formats, making the application versatile in processing various input types, including scanned documents and images.

- Text Summarization: Performed using the T5 Transformer model, which generates concise summaries from longer text inputs. This method is essential for users needing to quickly understand large volumes of text.

- Grammar Correction: Conducted using LanguageTool, which checks and corrects grammatical errors in the input text, ensuring that the final output is both accurate and high-quality.

5. Transformer Architecture:

- **The Transformer Model:** First introduced in the paper "Attention is All You Need," the Transformer model revolutionized NLP by solving sequence transduction tasks, which involve transforming an input sequence to an output sequence. Unlike RNNs, which process input sequentially, Transformers handle input data in parallel, significantly improving processing speed and performance.

- **Encoder-Decoder Structure:** The Transformer model is composed of two main components:

- Encoder: Processes the input sequence and generates a contextualized representation. For example, in translating the English sentence "How are you?" the encoder creates a representation that captures the meaning of each word in context.

- - Decoder: Uses the encoded representation to generate the output sequence. In our example, the translated sentence "¿Cómo estás?"

- **Self-Attention Mechanism:** The core innovation of the Transformer model, self-attention allows the model to weigh the importance of different elements in a sequence relative to one another. This mechanism enables the model to focus on relevant parts of the input, such as understanding that "cat" in the sentence "The cat sat on the mat" is related to "The" and "mat."

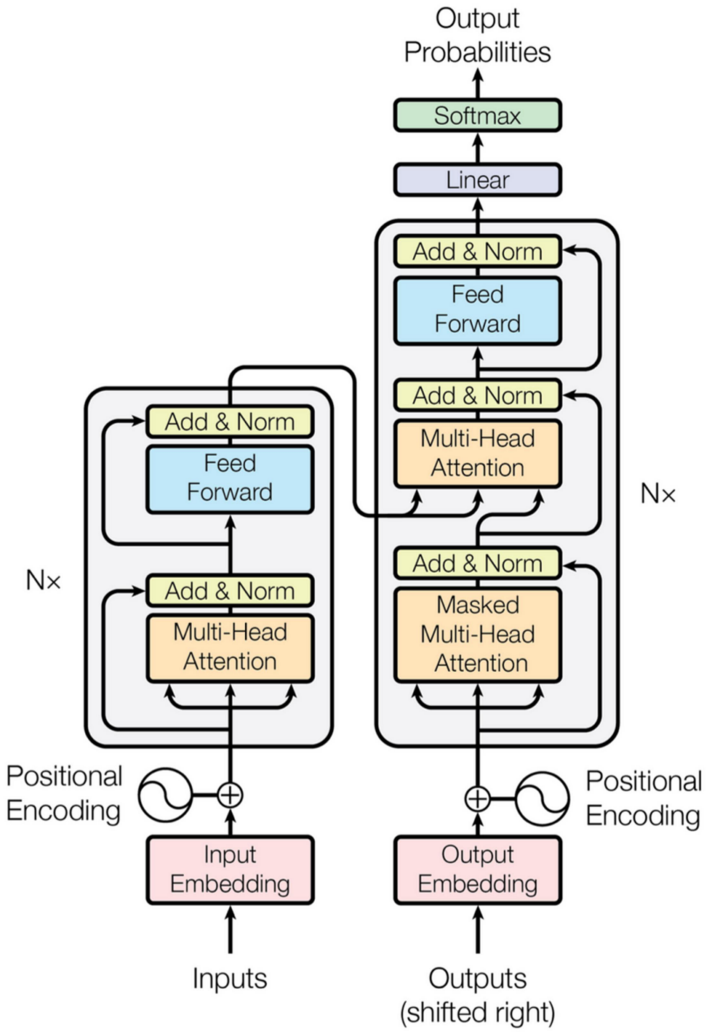

Figure 4.7 – Transformer Architecture

- **Multi-Head Attention:** The Transformer model uses multi-head attention, where the self-attention mechanism is applied multiple times with different learned projection matrices. This allows the model to capture various aspects of the input simultaneously, such as focusing on the subject, verb, and object in a sentence.

- **Positional Encoding:** Since Transformers lack an inherent understanding of the order of elements in a sequence, positional encodings are added to the input embeddings to convey the positional information. This enables the model to understand the sequence order, which is crucial for tasks like translation.

- **Masked Self-Attention:** Used in the decoder to prevent future tokens from influencing the generation of the current token. This is essential for tasks like language modeling, where the model should only consider past and current tokens.

This combination of platforms, technologies, methods, and applications, along with the advanced Transformer architecture, enables the system to provide comprehensive text processing capabilities. The system effectively handles tasks such as text summarization, OCR, grammar checking, and more, all while leveraging state-of-the-art NLP models to deliver accurate and efficient results.

- **Matrix Multiplication (MatMul):** The attention mechanism involves matrix multiplication between query and key vectors to generate attention scores, which determine how much focus each word in the sequence should have on the others.

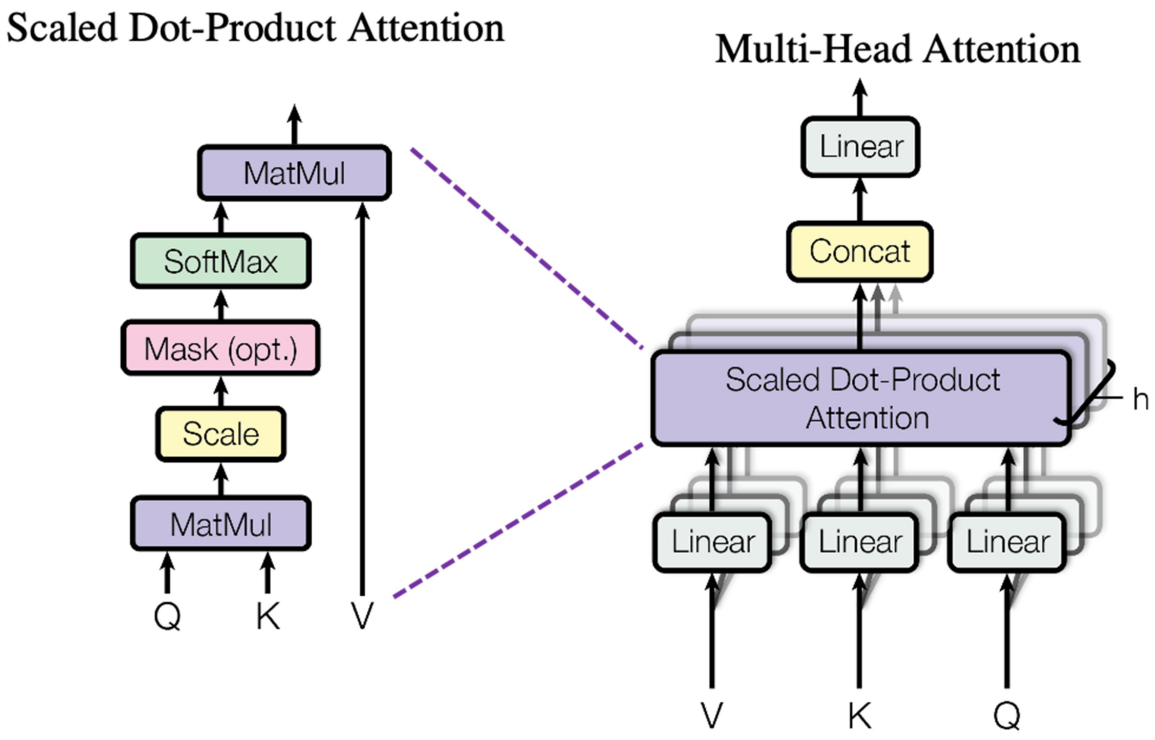

Figure 4.8 – Matrix multiplication and Multi head attention

5. SYSTEM TESTING

5.1. Test Plan

A comprehensive test plan will be executed to validate the system's functionality and performance. The plan includes:

- **Unit Testing**: Each module (e.g., file upload, text extraction, summarization, grammar checking) will be tested individually to ensure correct functionality.
- **Integration Testing:** Tests will be conducted to ensure that all modules work together seamlessly.
- **UI Testing:** The Streamlit interface will be tested for usability and correctness.

5.2. Scenarios

Test scenarios include:

1. File Upload and Format Validation:
   - Ensure the system correctly handles and validates various file formats (PDF, image, text).
2. Text Extraction Accuracy:
   - From PDF: Test the accuracy of text extraction from PDF documents.
   - From Image: Verify the precision of text extraction from images using OCR.
3. Summarization Accuracy:
   - Evaluate the quality and accuracy of the text summaries at various configurable lengths.
4. Grammar and Vocabulary Correction:
   - Validate the effectiveness of grammar and vocabulary correction features by comparing corrected text against standard grammar rules.

5.3. Output Screens

Screenshots and examples will be provided to demonstrate:

- File Upload Interface: Display the UI for uploading files.
- Summary Display: Show the interface where summarized text is presented.
- Corrected Text Display: Illustrate how corrected text appears after grammar checking.

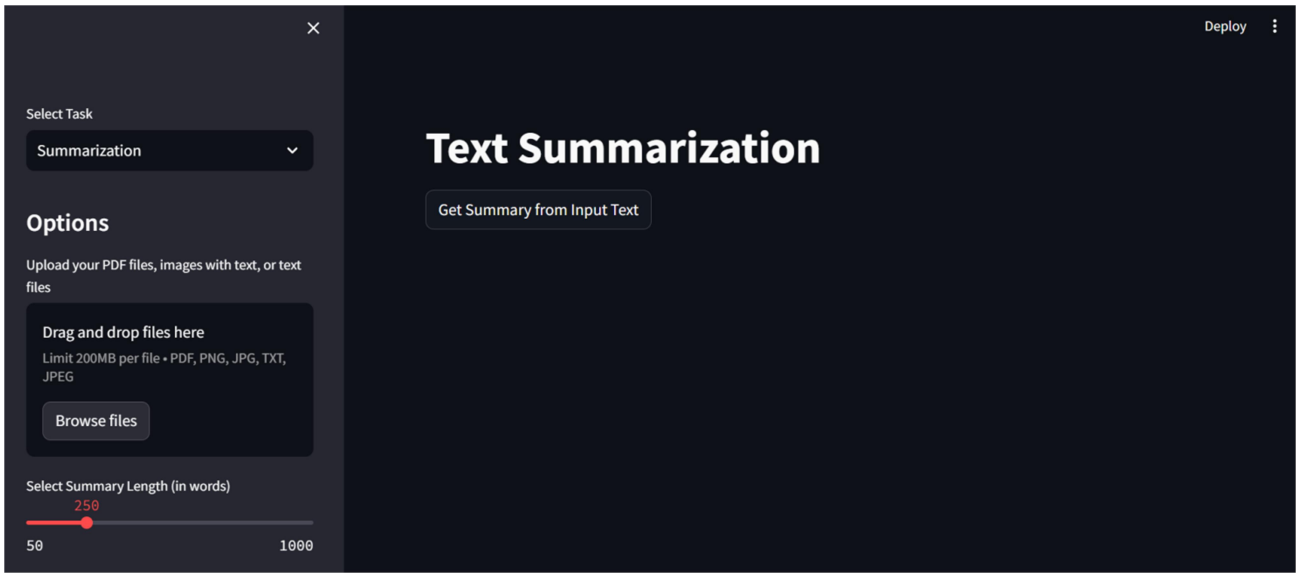

Figure 5.1 – Application Home page user interface

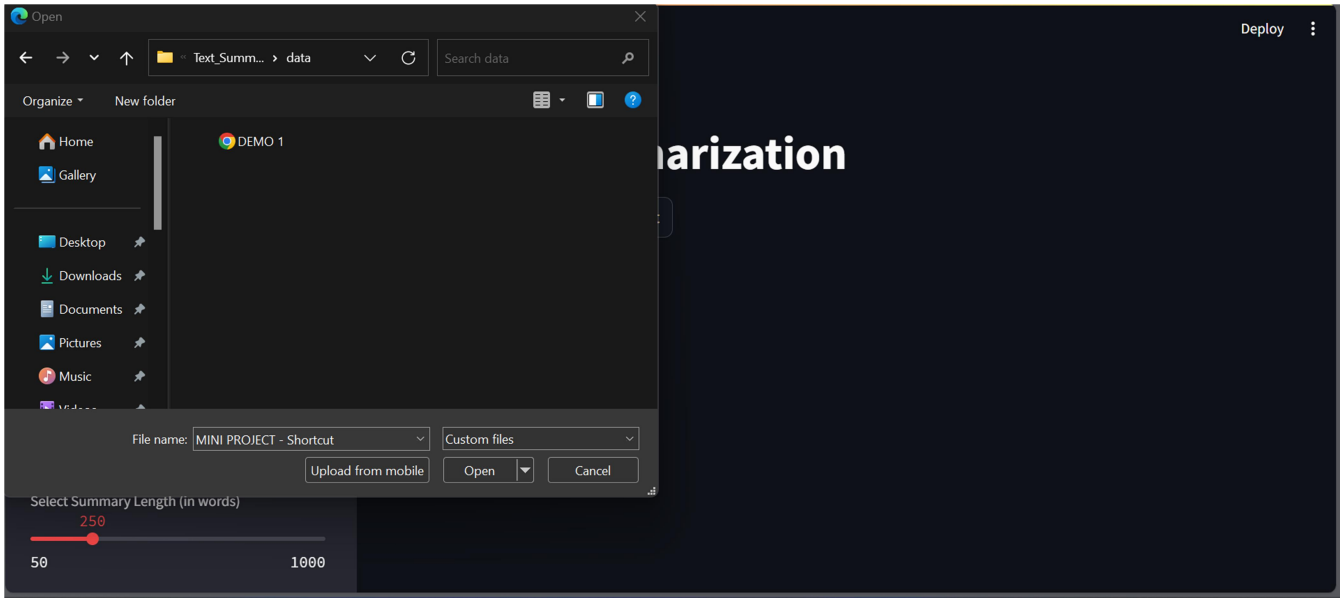

Figure 5.2 – File uploader interface

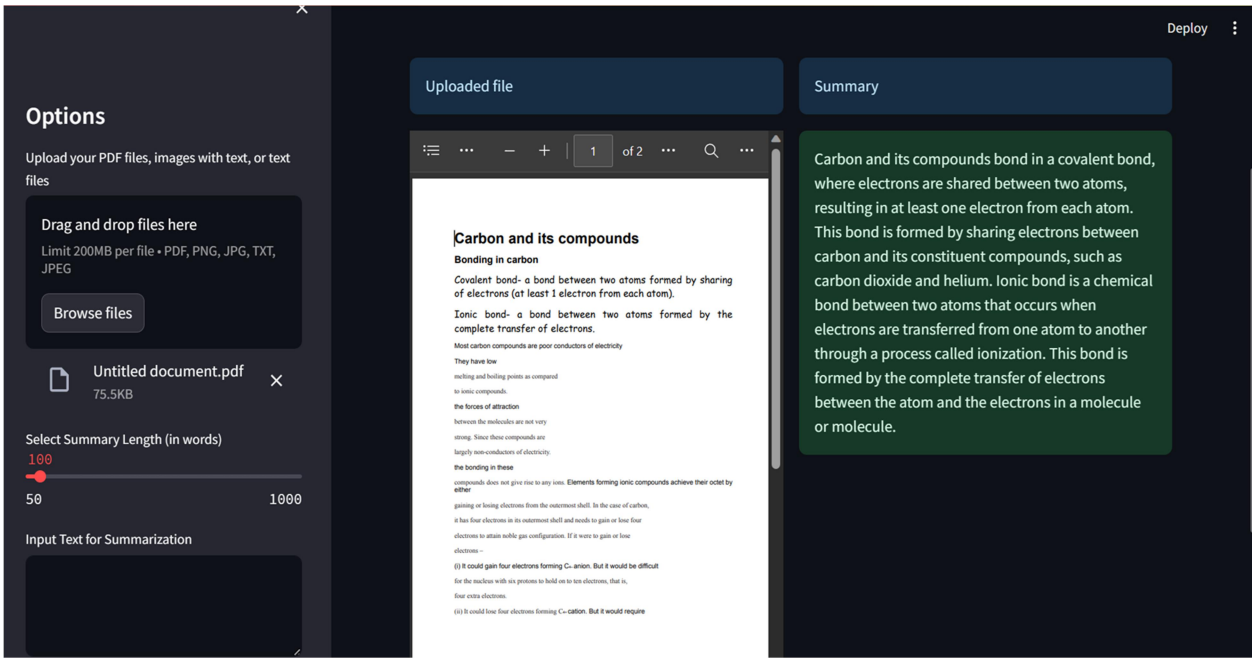

Figure 5.3 – Summary display

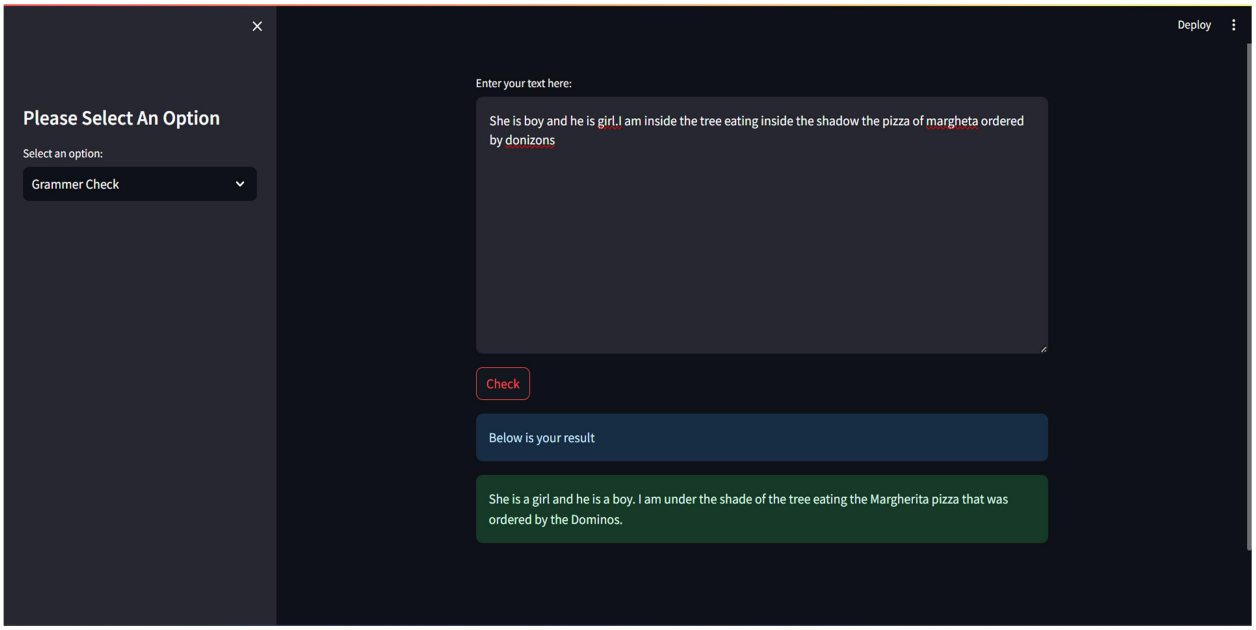

Figure 5.4 – Grammar check

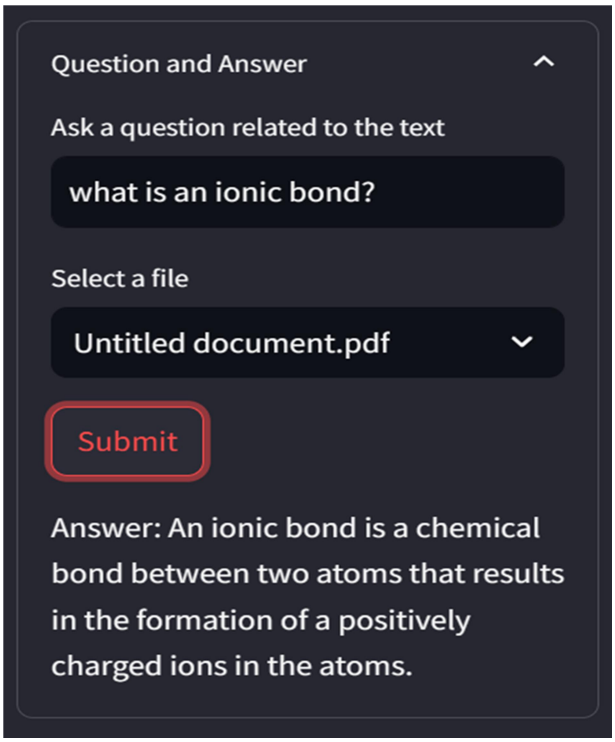

Figure 5.5 – QA Chatbot

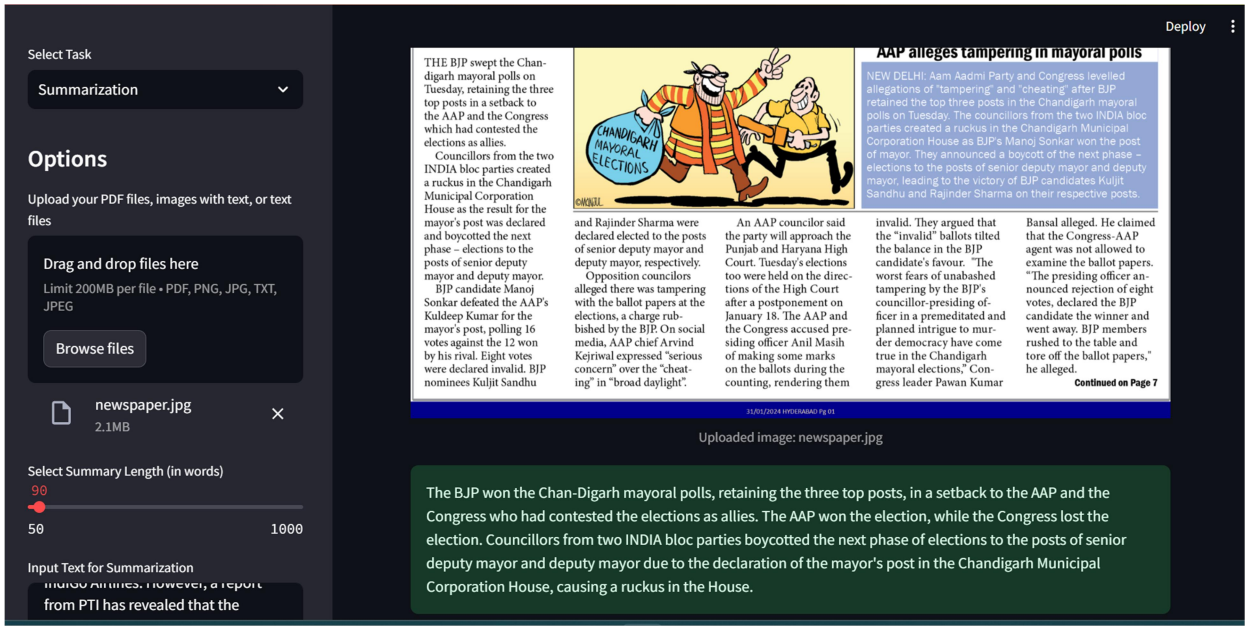

Figure 5.6 – QA Chatbot output

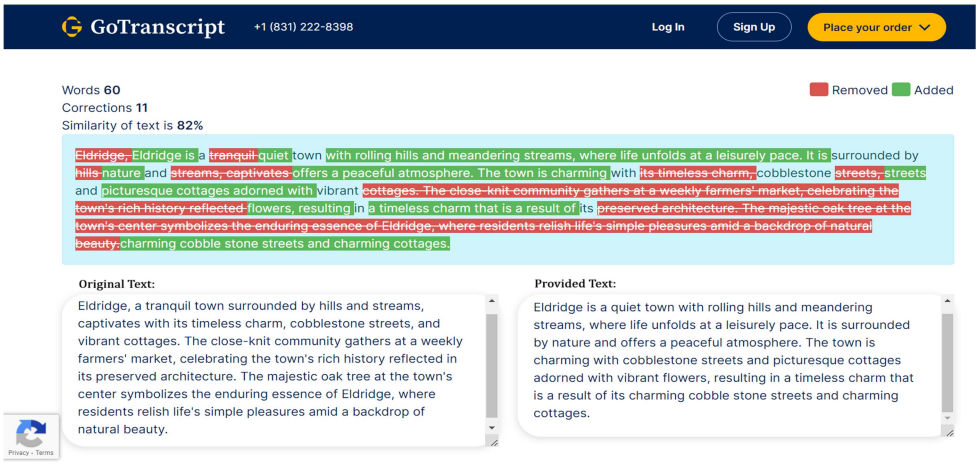

Figure 5.7 – Corrected check display after grammar check-1

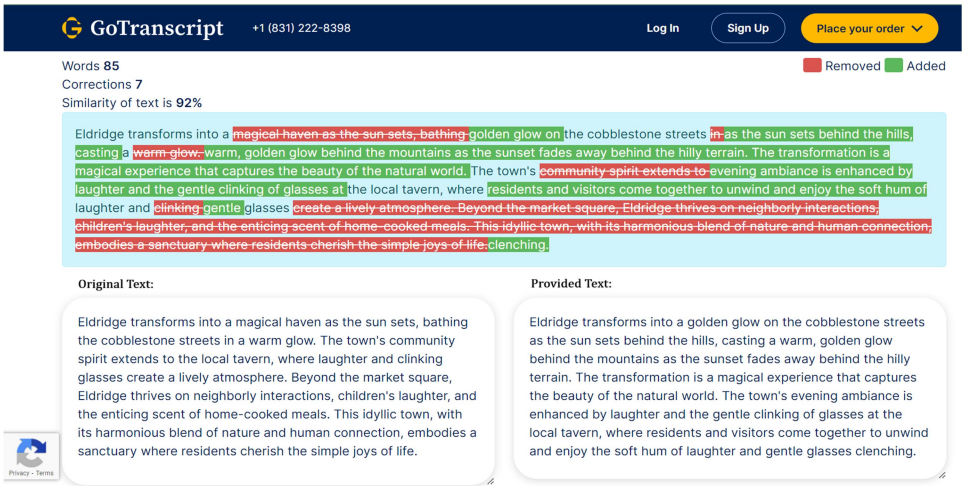

Figure 5.8 – Corrected check display after grammar check-2

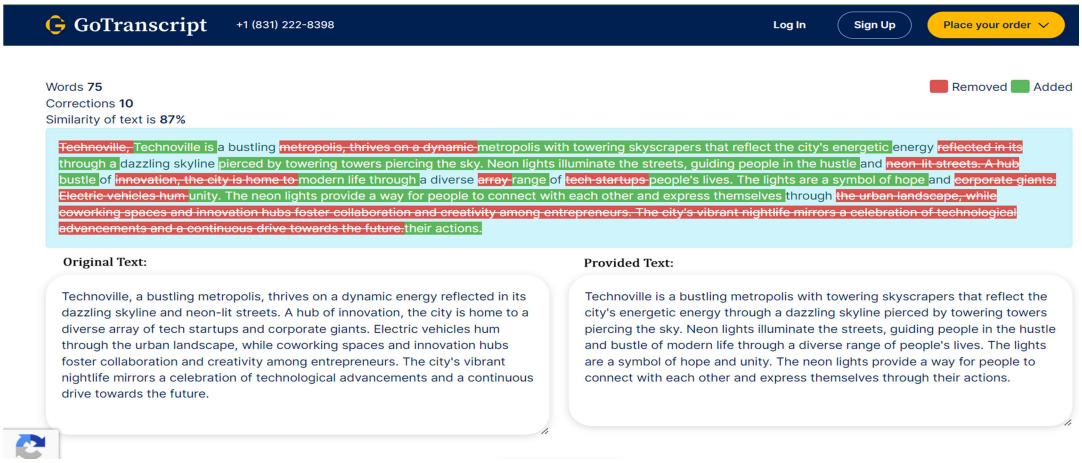

Figure 5.9 – Corrected check display after grammar check-3

6. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

The application successfully integrates text extraction, summarization, and grammar correction into a single platform, improving accessibility and usability for users handling diverse text inputs.

6.2 Future Scope

Future enhancements could include:

- Adding support for additional file formats (e.g., DOCX).
- Improving model accuracy with larger data models.
- Expanding the system to support multilingual text processing.
- Summarizing YouTube videos using audio-to-text models (e.g., Whisper).
- Summarizing from video transcripts.
- Addressing real-time problems that can be solved using text summarization.

7. REFERENCES

7.1. Bibliography

1. LaMini-LM documentation

LaMini-LM: A Diverse Herd of Distilled Models from Large-Scale Instructions

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji

CoRR, Vol. abs/2304.14402, 2023.

Available at: [https://arxiv.org/abs/2304.14402] (https://arxiv.org/abs/2304.14402)

2. Google-T5/ T5-base Model documentation

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu

Journal of Machine Learning Research, Vol. 21, No. 140, pp. 1-67, 2020.

Available at: http://jmlr.org/papers/v21/20-074.html

7.2 Web References

1. Python Documentation

Available at: https://docs.python.org/3/

2. Streamlit Documentation

Available at: https://docs.streamlit.io

3. Hugging Face Transformers Documentation

Available at: https://huggingface.co/docs/transformers

4. LangChain Documentation

Available at: https://docs.langchain.com

5. Tesseract-OCR Official Documentation

Available at: https://github.com/tesseract-ocr/tesseract

6. PyTorch Documentation

Available at: https://pytorch.org/docs/

7. GitHub - AIAnytime

Available at: https://github.com/AIAnytime

8. GitHub - Jalammar

Available at: https://jalammar.github.io

9. Zakhtar Medium

Available at: https://medium.com/@zakhtar2020

10. DataCamp

Available at: https://www.datacamp.com

8. APPENDIX

Annexure-1: Sample Coding

```
import streamlit as st
from PIL import Image
import pytesseract
from transformers import T5Tokenizer, T5ForConditionalGeneration, pipeline
import torch
import base64
import PyPDF2
from PyPDF2 import PdfReader
from autocorrect import Speller
import language_tool_python
import io

# MODEL AND TOKENIZER
checkpoint = "D:\MINI PROJECT\Text_Summarization\La-Mini-Flan-T5-248M"
tokenizer = T5Tokenizer.from_pretrained(checkpoint)
base_model = T5ForConditionalGeneration.from_pretrained(checkpoint,
device_map='auto', torch_dtype=torch.float32,
token="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")
spell = Speller()

def perform_ocr(image):
    text = pytesseract.image_to_string(image)
    return text
```

```
import tempfile
import logging

def file_preprocessing(uploaded_file):
    if uploaded_file is None:
        st.error("No file uploaded!")
        return None

    if not uploaded_file:
        return None

    # Calculate file size from content
    file_content = uploaded_file.read()
    file_size = len(file_content)

    # Log file size for debugging
    logging.info(f"Uploaded file size: {file_size}")

    try:
        # Check for empty file
        if len(file_content) == 0:
            st.error("Uploaded file is empty!")
            return None

        # Check MIME type
        if uploaded_file.type not in ['application/pdf']:
            st.error("Invalid file format. Please upload a PDF file.")
            return None
```

```
            # Process the file using PyPDF2
            pdf_reader = PyPDF2.PdfReader(io.BytesIO(file_content))
            text = ""
            for page_num in range(len(pdf_reader.pages)):
                page = pdf_reader.getPage(page_num)
                text += page.extract_text()
            return text

        except PyPDF2.errors.PdfReadError as e:
            # Handle potential PDF format errors
            st.error(f"Error processing PDF: {e}. The file might be corrupted.")
            logging.error(f"Error processing PDF: {e}")
            return None

        except Exception as e:  # Catch other unexpected errors
            st.error(f"Error processing file: {e}")
            logging.error(f"Error processing file: {e}")
            return None



def count_words(text):
    words = text.split()
    return len(words)

def llm_pipeline(text, summary_length_words):
    pipe_sum = pipeline('summarization', model=base_model, tokenizer=tokenizer)
    summary = ""
    total_words = 0
```

```
    if text is not None:  # Check for None before splitting
        sentences = text.split(".")
        for sentence in sentences:
            result = pipe_sum(sentence, max_length=summary_length_words,
min_length=int(summary_length_words * 0.6), do_sample=False)
            sentence_summary = result[0]['summary_text']
            words_in_sentence = count_words(sentence_summary)
            if total_words + words_in_sentence <= summary_length_words:
                summary += sentence_summary + " "
                total_words += words_in_sentence
            if total_words >= summary_length_words:
                break
    return summary

def check_grammar_vocabulary(input_text):
    tool = language_tool_python.LanguageTool('en-US')
    matches = tool.check(input_text)
    corrected_text = tool.correct(input_text)
    return matches, corrected_text

@st.cache_data
def displayPDF(file):
    base64_pdf = base64.b64encode(file.read()).decode('utf-8')
    pdf_display = f'<iframe src="data:application/pdf;base64,{base64_pdf}"
width="100%" height="600" type="application/pdf"></iframe>'
    st.markdown(pdf_display, unsafe_allow_html=True)
```

```
def process_text(input_text, summary_length_words):
    pipe_sum = pipeline('summarization', model=base_model, tokenizer=tokenizer)
    summary = ""
    total_words = 0
    sentences = input_text.split(".")
    for sentence in sentences:
        result = pipe_sum(sentence, max_length=summary_length_words,
min_length=int(summary_length_words * 0.6), do_sample=False)
        sentence_summary = result[0]['summary_text']
        words_in_sentence = count_words(sentence_summary)
        if total_words + words_in_sentence <= summary_length_words:
            summary += sentence_summary + " "
            total_words += words_in_sentence
        if total_words >= summary_length_words:
            break
    return summary

# Streamlit code
st.set_page_config(layout='wide', page_title="Summarization")

def extract_text_from_file(uploaded_file):
    if uploaded_file is None:
        st.error("No file uploaded!")
        return None

    try:
        reader = PdfReader(uploaded_file)
        text = ""
```

```
        for page in reader.pages:
            text += page.extract_text()
        return text
    except Exception as e:
        st.error(f"Error extracting text from PDF: {e}")
        return None

def grammar_vocabulary_tab():
    input_text = st.text_area("Input Text for Grammar and Vocabulary Check")
    if st.button("Check Grammar and Vocabulary"):
        matches, corrected_text = check_grammar_vocabulary(input_text)
        if matches:
            for match in matches:
                try:
                    line_info = f" (Line {match.fromy}, Column {match.fromx})"
                except AttributeError:
                    line_info = ""
            st.subheader("Corrected Text:")
            st.markdown(f"<pre>{corrected_text}</pre>", unsafe_allow_html=True)

def main():
    selected_task = st.sidebar.selectbox("Select Task", ["Summarization", "Grammar and Vocabulary Check"])
    st.sidebar.title('Options')
    uploaded_files = st.sidebar.file_uploader("Upload your PDF files, images with text, or text files", type=['pdf', 'png', 'jpg', 'txt'], accept_multiple_files=True)
    summary_length_words = st.sidebar.slider("Select Summary Length (in words)", min_value=50, max_value=1000, value=250, step=10)
```

```
    if selected_task == "Summarization":
        st.title('Text Summarization')
        input_text = st.sidebar.text_area("Input Text for Summarization")
        get_summary_button = st.button("Get Summary from Input Text")
        if get_summary_button:
            summary = process_text(input_text, summary_length_words)
            st.success(summary)

        text_from_file = ""
        with st.sidebar.expander("Question and Answer"):
            question = st.text_input("Ask a question related to the text")
            corrected_question = spell(question)

            if uploaded_files:
                file_names = [uploaded_file.name for uploaded_file in uploaded_files]
            else:
                file_names = [input_text]

            selected_file = st.selectbox("Select a file", file_names)

            if selected_file:
                selected_uploaded_file = next((file for file in uploaded_files if file.name ==
selected_file), None)
                if selected_uploaded_file:
                    text_from_file = extract_text_from_file(selected_uploaded_file)

                if st.button("Submit"):
```

```
                if text_from_file and corrected_question:
                    qa_input = f"question: {corrected_question} context: {text_from_file}"
                    answer = base_model.generate(input_ids=tokenizer.encode(qa_input,
return_tensors="pt"), max_length=100)
                    decoded_answer = tokenizer.decode(answer[0],
skip_special_tokens=True)
                    st.write("Answer:", decoded_answer)
                else:
                    st.warning("Please select a file")
            else:
                st.warning("Please upload a file")

    elif selected_task == "Grammar and Vocabulary Check":
        st.title('Grammar and Vocabulary Check')
        grammar_vocabulary_tab()

    if uploaded_files is not None:
        for uploaded_file in uploaded_files:
            if uploaded_file.type == 'application/pdf':
                if st.button(f"Get Summary of {uploaded_file.name}"):
                    col1, col2 = st.columns(2)
                    with col1:
                        st.info("Uploaded file")
                        displayPDF(uploaded_file)
                    with col2:
                        st.info("Summary")
                        summary =
llm_pipeline(file_preprocessing(io.BytesIO(uploaded_file.read())),
```

```
summary_length_words)
                st.success(summary)

        elif uploaded_file.type.startswith('image/'):
            if st.button(f"Get Summary of {uploaded_file.name}"):
                image = Image.open(uploaded_file)
                st.image(image, caption=f"Uploaded image: {uploaded_file.name}",
use_column_width=True)
                text_from_image = perform_ocr(image)
                summary = process_text(text_from_image, summary_length_words)
                st.success(summary)

        elif uploaded_file.type == 'text/plain':
            if st.button(f"Get Summary of {uploaded_file.name}"):
                text = uploaded_file.getvalue().decode("utf-8")
                if text.strip():
                    summary = process_text(text, summary_length_words)
                    st.success(summary)
                else:
                    st.warning("The uploaded text file is empty. Please upload a text file with
content.")

if __name__ == '__main__':
    main()
```

Annexure-2: List of Figures

Annexure-3: List of Output Screens

Annexure-4: List of Tables

N/A

Annexure-5: Base Paper

1. ***Attention is All You Need***

- Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin
- Journal: Advances in Neural Information Processing Systems (NeurIPS)
- Year: 2017
- URL: https://arxiv.org/abs/1706.03762

2. ***LaMini-FLAN-T5: A Diverse Herd of Distilled Models from Large-Scale Instructions***

- Authors: Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, Alham Fikri Aji
- Journal: CoRR
- Year: 2023
- Volume: abs/2304.14402
- URL: https://arxiv.org/abs/2304.14402

Annexure-6: Published Paper

N/A