Technical Documentation for AcademiAI - AI-Powered Academic Assistant

Student: Amro Gamar ALdwlah

• Table of Content.

- **Overview:** AcademiAI is an AI-powered academic assistant designed to simplify, enhance, and revolutionize the academic and research experience for students, professors, and researchers. The platform offers a suite of tools that help users save time, streamline their work, and focus on learning, teaching, and discovery. The project is built using Streamlit for the frontend and integrates various AI models, including OpenAI's GPT-4, for backend processing.

- **Project Structure**
  The project is organized into several modules, each corresponding to a specific functionality:

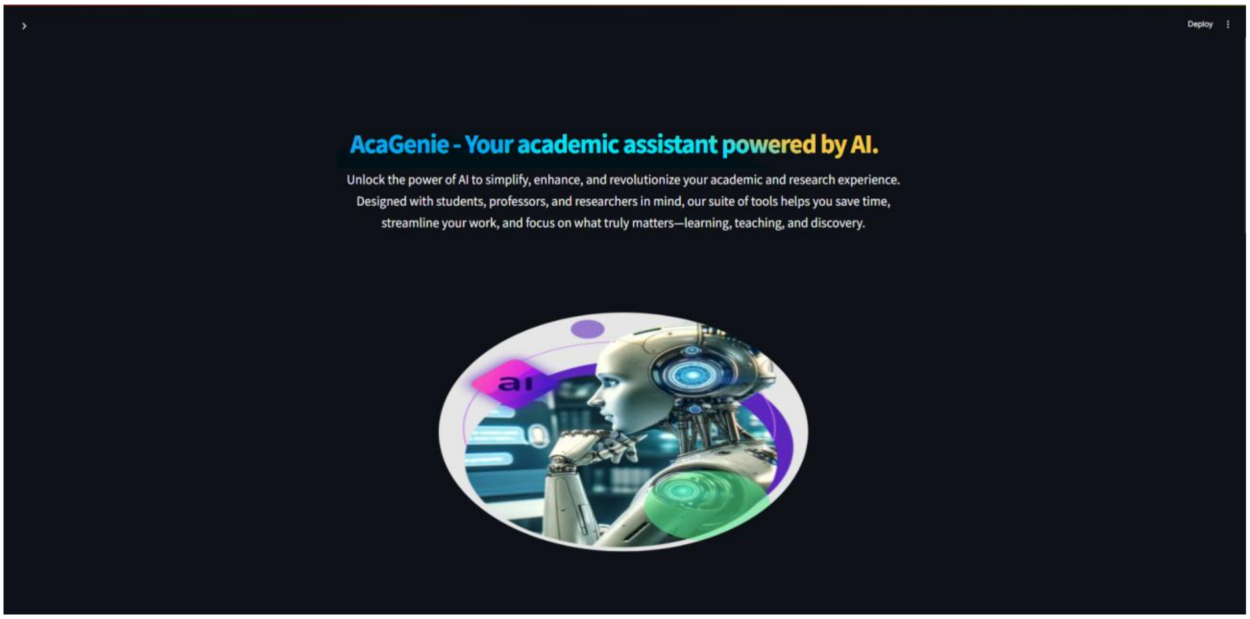
  1. **Home_Page.py**: The main landing page of the application, providing an overview of the features and functionalities.
  2. **HtmlTemplates.py**: Contains HTML and CSS templates for styling the chat interface.
  3. **AudioBook_Generator .py**: Converts PDF documents into audiobooks with custom cover art.
  4. **Chat_With_PDF .py**: Allows users to interact with multiple PDFs using AI to extract insights and answer questions.
  5. **Presentation_Generator .py**: Generates PowerPoint presentations from text inputs.
  6. **Quiz_Generator .py**: Creates quizzes and flashcards from uploaded documents.
  7. **Summary_Generator.py**: Summarizes PDFs, blogs, and YouTube videos based on user prompts.

1. Home Page

The home page provides an overview of the AcademiAI platform, highlighting its features and benefits.

- **Technologies Used**: Streamlit, custom CSS for styling.
- **Functionality**:
    - Displays a hero section with a dynamic title and subtitle.
    - Features an animated AI assistant image.
    - Provides a detailed description of the platform's offerings.

- **Custom CSS Styling**
- The page uses custom CSS to style the layout, animations, and overall appearance. The CSS is embedded directly into the Streamlit app using the **st.markdown()**

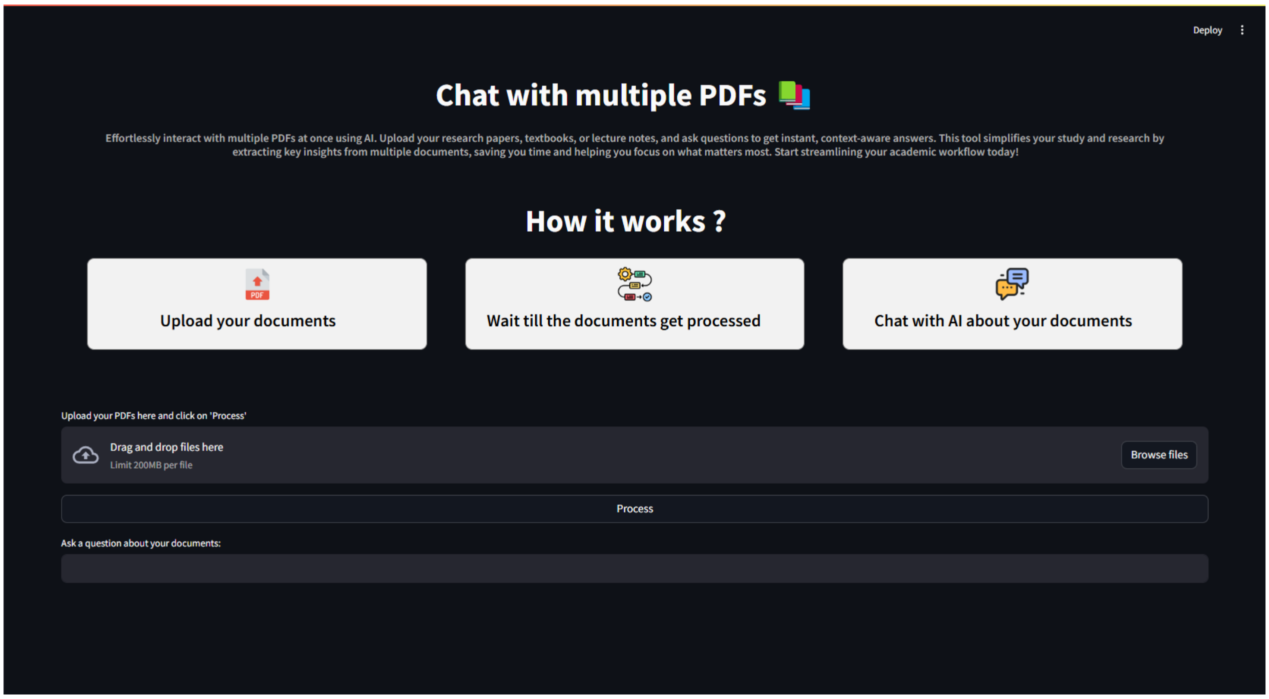


AcaGenie - Your academic assistant powered by AI.

Unlock the power of AI to simplify, enhance, and revolutionize your academic and research experience. Designed with students, professors, and researchers in mind, our suite of tools helps you save time, streamline your work, and focus on what truly matters—learning, teaching, and discovery.

Chat with PDF Feature

The **Chat with PDF** feature allows users to interact with multiple PDF documents using AI. Users can upload PDFs, and the system processes the text to enable a conversational interface where users can ask questions and receive context-aware answers. Built with **Streamlit**, **LangChain**, **OpenAI**, and **FAISS**, this feature simplifies document interaction for academic and research workflows.



Effortlessly interact with multiple PDFs at once using AI. Upload your research papers, textbooks, or lecture notes, and ask questions to get instant, context-aware answers. This tool simplifies your study and research by extracting key insights from multiple documents, saving you time and helping you focus on what matters most. Start streamlining your academic workflow today!

Upload your PDFs here and click on 'Process'

Ask a question about your documents:

1. Overflow Explanation

1. Input: Question or Query

- The process starts when a user asks a question, such as *"What is a neural network?"*.
- The question is converted into an **embedding** (a numerical representation) using a **Language Model (LLM)** to capture its semantic meaning.

2. Semantic Search

- The system performs a **semantic search** to find relevant information from a **knowledge base**, which consists of text chunks (e.g., sections of PDF documents) that have been converted into embeddings.
- The question's embedding is compared with the embeddings of the text chunks to find the most relevant information.

3. Vector Store (FAISS)

- The embeddings of the text chunks are stored in a **vector store**, such as **FAISS** (Facebook AI Similarity Search).
- FAISS is optimized for **fast and efficient similarity searches**, allowing the system to quickly retrieve the most relevant text chunks based on the question's embedding.

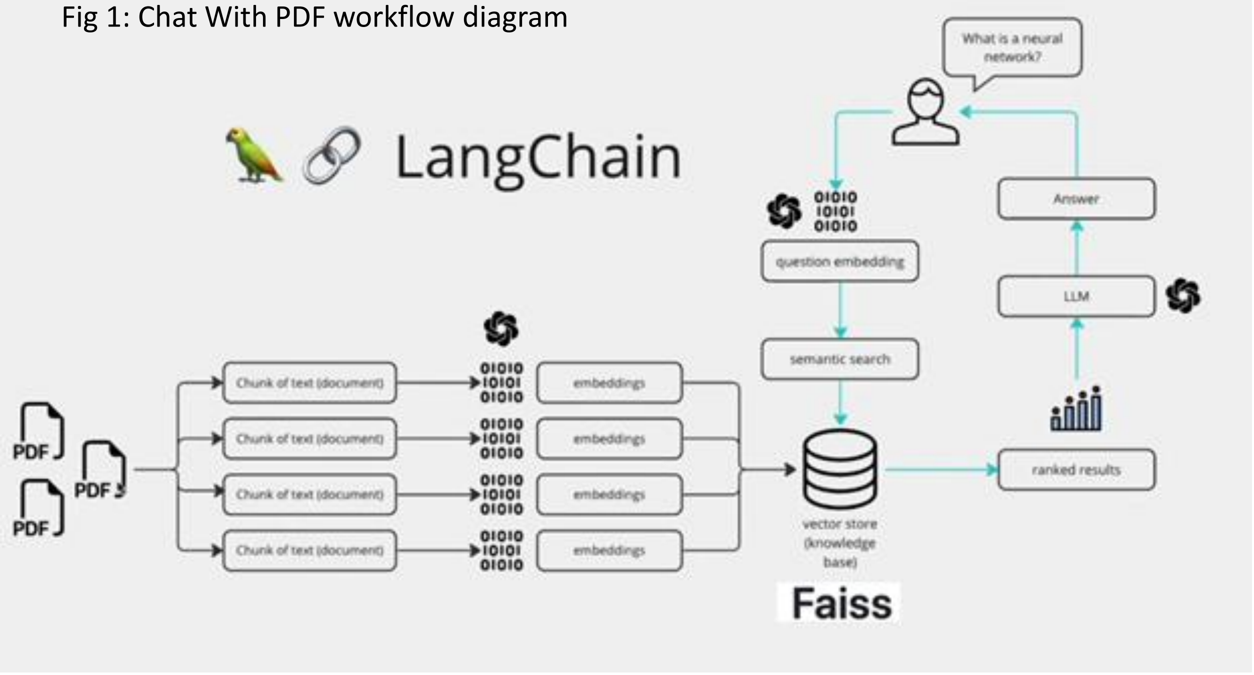
4. Ranked Results

- The retrieved text chunks are **ranked** based on their similarity to the question.
- Only the **most relevant chunks** are passed to the LLM for generating an answer, ensuring the input stays within the model's context limit.

5. Output: Answer

- The **LLM** processes the ranked text chunks and generates a **coherent answer** to the user's question.
- The answer is then returned to the user.

Fig 1: Chat With PDF workflow diagram



2. Overflow Handling

1. Text Chunking

- **Issue**: Large PDFs can produce excessive text, leading to **memory overflow**.
- **Solution**: The text is split into **smaller, manageable chunks** (e.g., 1000 characters each), preventing memory overflow.

2. Embedding Storage

- **Issue**: Storing embeddings for large datasets can consume **significant memory**.
- **Solution**: **FAISS** is used to store and retrieve embeddings efficiently, minimizing memory usage while enabling **fast searches**.

3. Semantic Search

- **Issue**: Searching through a large number of text chunks can be **computationally expensive**.
- **Solution**: FAISS performs **approximate nearest neighbor (ANN) searches**, which are faster and more efficient than exact searches.

4. Context Limitation

- **Issue**: Language models like GPT have a **limited context window** (e.g., 4096 tokens for GPT-3).
- **Solution**: Only the **most relevant text chunks** are passed to the LLM, ensuring the input stays within the model's context limit.

3. Key Components

1. Text Extraction

- **Function**: `get_pdf_text(user_pdfs)`
- **Purpose**: Extracts text from uploaded PDFs using `PyPDF2.PdfReader`
- **Output**: A single string containing all text from the PDFs.

2. Text Chunking

- **Function**: `get_text_chunks(text)`
- **Purpose**: Splits text into smaller chunks (1000 characters) with 200-character overlap for context preservation.
- **Output**: A list of text chunks.

3. Vectorization

- **Function**: `get_vectorstore(text_chunks)`
- **Purpose**: Converts text chunks into vector embeddings using `OpenAIEmbeddings` and stores them in a **FAISS** vector store for fast retrieval.
- **Output**: A FAISS vector store.

• 4. Conversation Chain

- **Function**: `get_conversation_chain(vectorstore)`
- **Purpose**: Creates a conversational AI model using `ChatOpenAI` and `ConversationBufferMemory` to maintain chat history.
- **Output**: A conversational AI model ready to answer questions.

5. Handling User Questions

- **Function**: `handle_user_question(user_question)`
- **Purpose**: Processes user questions, retrieves relevant information from the vector store, and generates responses using the AI model.
- **Output**: Displays the AI's response in a chat-like interface using custom HTML templates.

4. Technical Implementation

1. Streamlit Frontend

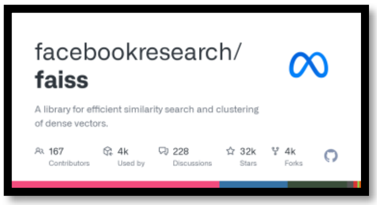
- The app uses `st.file_uploader()` to allow users to upload multiple PDF files.
- A "Process" button triggers the text extraction, chunking, and vectorization processes.
- Users ask questions via `st.text_input()`, and the responses are displayed in a chat-like interface.

2. LangChain and OpenAI

- **LangChain** is used to manage the conversational AI pipeline, including text splitting, embeddings, and memory management.
- **OpenAI's GPT models** generate responses to user questions.
- The `ConversationalRetrievalChain` combines the language model with the vector store to provide context-aware answers.

3. FAISS Vector Store

- **FAISS** (Facebook AI Similarity Search) is used to store and retrieve vector embeddings of the text chunks.
- FAISS enables fast and efficient similarity searches, allowing the AI to quickly find relevant information from the PDFs.


facebookresearch/
faiss

A library for efficient similarity search and clustering of dense vectors.

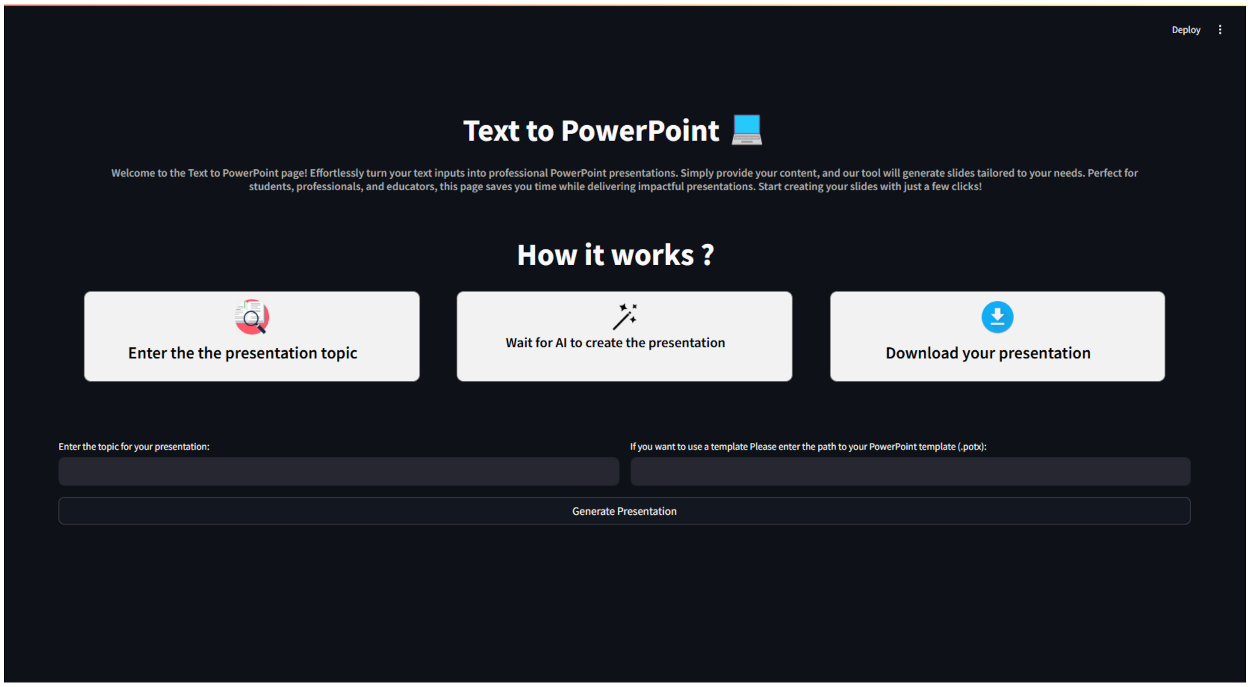
167 Contributors · 4k Used by · 228 Discussions · 32k Stars · 4k Forks

4. Custom CSS and HTML Templates

- The chat interface is styled using custom CSS to create a visually appealing and user-friendly experience.
- HTML templates (`user_template` and `bot_template`) are used to format the chat messages, with different styles for user and bot messages.

Text to PowerPoint Feature

The **Text to PowerPoint** feature allows users to generate professional PowerPoint presentations from a given topic using AI. The feature leverages **OpenAI's GPT-4** for generating slide titles and content, **Unsplash API** for fetching relevant images, and **python-pptx** for creating the PowerPoint presentation. The entire process is integrated into a **Streamlit** web application, providing a user-friendly interface.



Welcome to the Text to PowerPoint page! Effortlessly turn your text inputs into professional PowerPoint presentations. Simply provide your content, and our tool will generate slides tailored to your needs. Perfect for students, professionals, and educators, this page saves you time while delivering impactful presentations. Start creating your slides with just a few clicks!

How it works ?
Enter the the presentation topic
Wait for AI to create the presentation
Download your presentation
Enter the topic for your presentation:
If you want to use a template Please enter the path to your PowerPoint template (.potx):
Generate Presentation

1. Workflow Explanation

1. User Input

- The user enters a topic for the presentation and optionally provides a PowerPoint template path.

2. Slide Title Generation

- The `generate_slide_titles(topic)` function sends a prompt to GPT-4 to generate 10 slide titles for the given topic.
- The titles are filtered to remove any empty or invalid entries.

3. Slide Content Generation

- For each slide title, the `generate_slide_content(slide_title)` function sends a prompt to GPT-4 to generate content.
- The content is stored in a list for later use.

4. Image Fetching

- The `fetch_image(topic)` function queries the Unsplash API to fetch a relevant image for the presentation.

5. PowerPoint Creation

- The `create_presentation_with_template(template_path, topic, slide_titles, slide_contents)` function creates the PowerPoint presentation:
- A title slide is added with the topic and a subtitle.
- For each slide title and content, a new slide is created with the title, content, and an image (if available).
- The presentation is saved to the `generated_ppt` `directory`

6. Download Presentation

- The generated PowerPoint file is made available for download via a **Streamlit download button**.

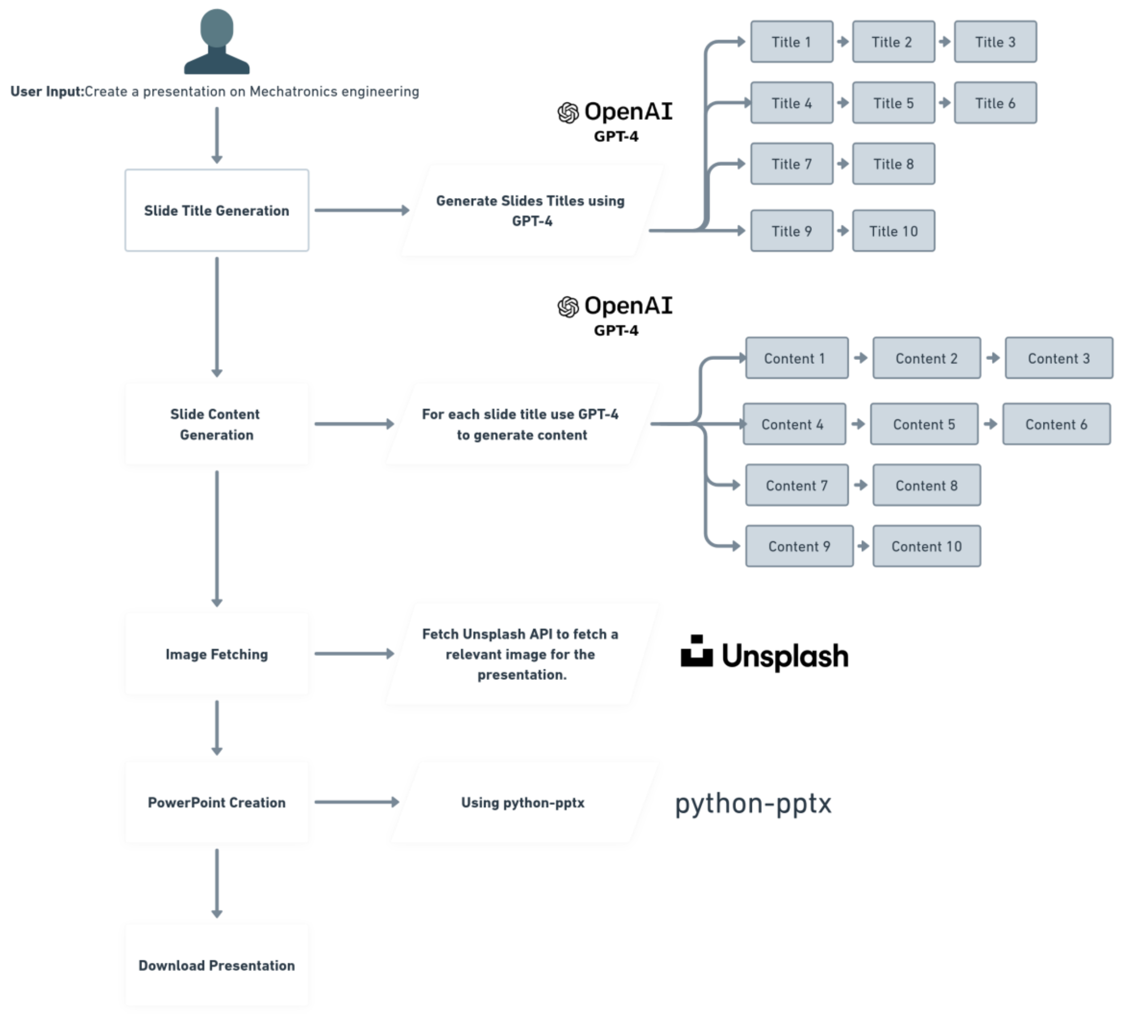


Fig 2: Presentation generator workflow diagram

2. Overflow Handling

To address potential overflow issues, the following strategies are implemented:

1. Input Topic Limitation

- Limits slide titles to **10** to avoid excessive generation.

2. Content Truncation

- Sets `max_tokens` to **100** for concise slide content.

3. Image Optimization

- Fetches only **one image per presentation** and resizes it to 4 inches width.

4. File Size Management

- Limits slides to **10** and uses compressed images to reduce file size.

3. Key Components

1. OpenAI Integration

- **Purpose**: Generates slide titles and content using GPT-4.
- **Functions**:
  - `generate_slide_titles(topic)` : Generates 10 slide titles for the given topic.
  - `generate_slide_content(slide_title)` : Generates content for each slide title.
- **Process**:
  - The system sends a prompt to GPT-4 to generate slide titles or content.
  - The response is parsed and returned as a list of titles or a string of content.

OpenAI
GPT-4

2. Unsplash API Integration

- **Purpose**: Fetches relevant images for the presentation based on the topic.
- **Function**: `fetch_image(topic)`
- **Process**:
  - The function queries the Unsplash API with the topic as the search term.
  - It retrieves a random image URL related to the topic.
  - The image is downloaded and embedded into the PowerPoint slide.

Unsplash

3. PowerPoint Creation

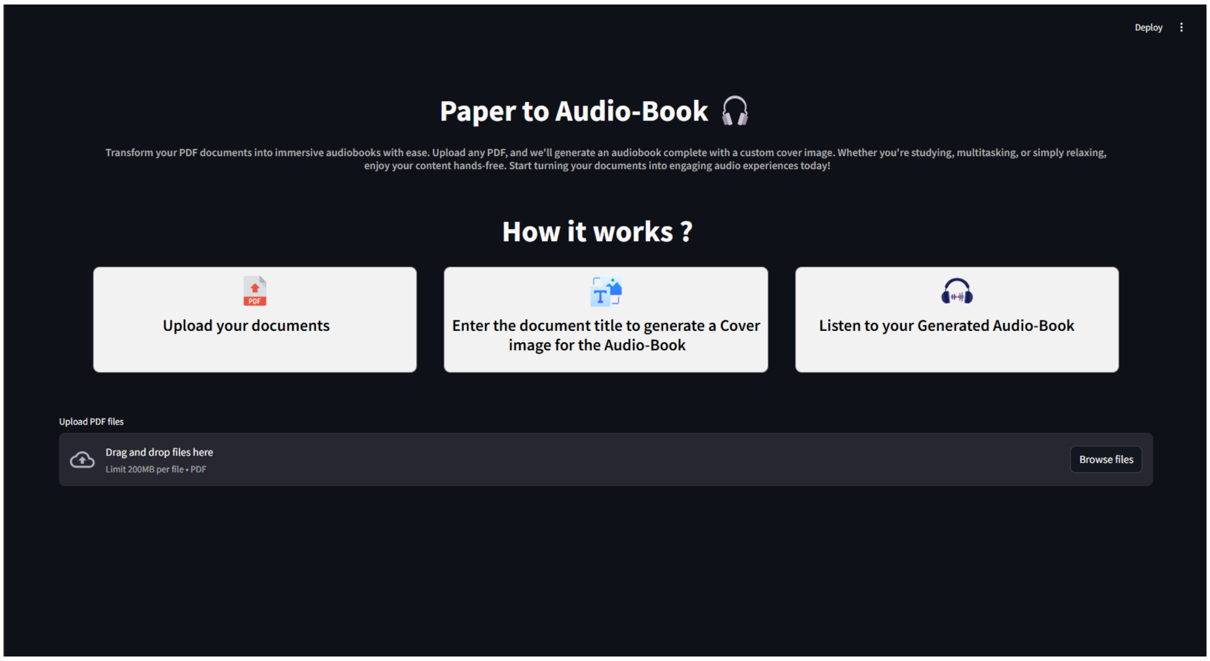
- **Purpose:** Creates a PowerPoint presentation using the generated content and images.
- **Function**: `create_presentation_with_template(template_path, topic, slide_titles, slide_contents)`
- **Process**:
    - A PowerPoint presentation is created using a template (if provided) or a default layout.
    - The title slide is customized with the topic and a subtitle.
    - For each slide title and content, a new slide is added with:
        - A title (styled with a larger font size).
        - Content text (styled with a smaller font size).
        - An image fetched from Unsplash (if available).
    - The presentation is saved in the `generated_ppt directory`

4. Streamlit Frontend

- **Purpose**: Provides a user-friendly interface for generating presentations.
- **Process**:
    - Users enter a topic for the presentation and optionally provide a PowerPoint template path.
    - A "Generate Presentation" button triggers the generation process.
    - Once the presentation

Paper to Audio-Book Feature

The **Paper to Audio-Book** feature allows users to convert PDF documents into immersive audiobooks. Users can upload PDFs, generate a custom cover image using **OpenAI's DALL·E**, and convert the text into speech using **gTTS** (Google Text-to-Speech). The feature is integrated into a **Streamlit** web application, providing a user-friendly interface for generating and downloading audiobooks.



Paper to Audio-Book 🎧

Transform your PDF documents into immersive audiobooks with ease. Upload any PDF, and we'll generate an audiobook complete with a custom cover image. Whether you're studying, multitasking, or simply relaxing, enjoy your content hands-free. Start turning your documents into engaging audio experiences today!

How it works ?

Upload your documents

Enter the document title to generate a Cover image for the Audio-Book

Listen to your Generated Audio-Book

Upload PDF files

Drag and drop files here
Limit 200MB per file • PDF

1. Workflow Explanation

The application follows a structured workflow to convert PDF documents into audiobooks:

1. **User Uploads PDF Files**:
   - The user uploads one or more PDF files containing the content they want to convert into an audiobook.
   - The application extracts text from the uploaded PDFs using the `pdfplumber` library

2. **Generate Cover Art**:
   - The user provides a title for the audiobook.
   - The application uses OpenAI's DALL·E model to generate a custom cover art image based on the provided title.

3. **Convert Text to Speech**:
   - The extracted text (or generated dialogue) is converted into speech using the `gTTS` library.
   - The generated audio is saved as an MP3 file.

4. **Provide Downloadable Output**:
   - The application displays the generated cover art and provides a downloadable link for the MP3 file.

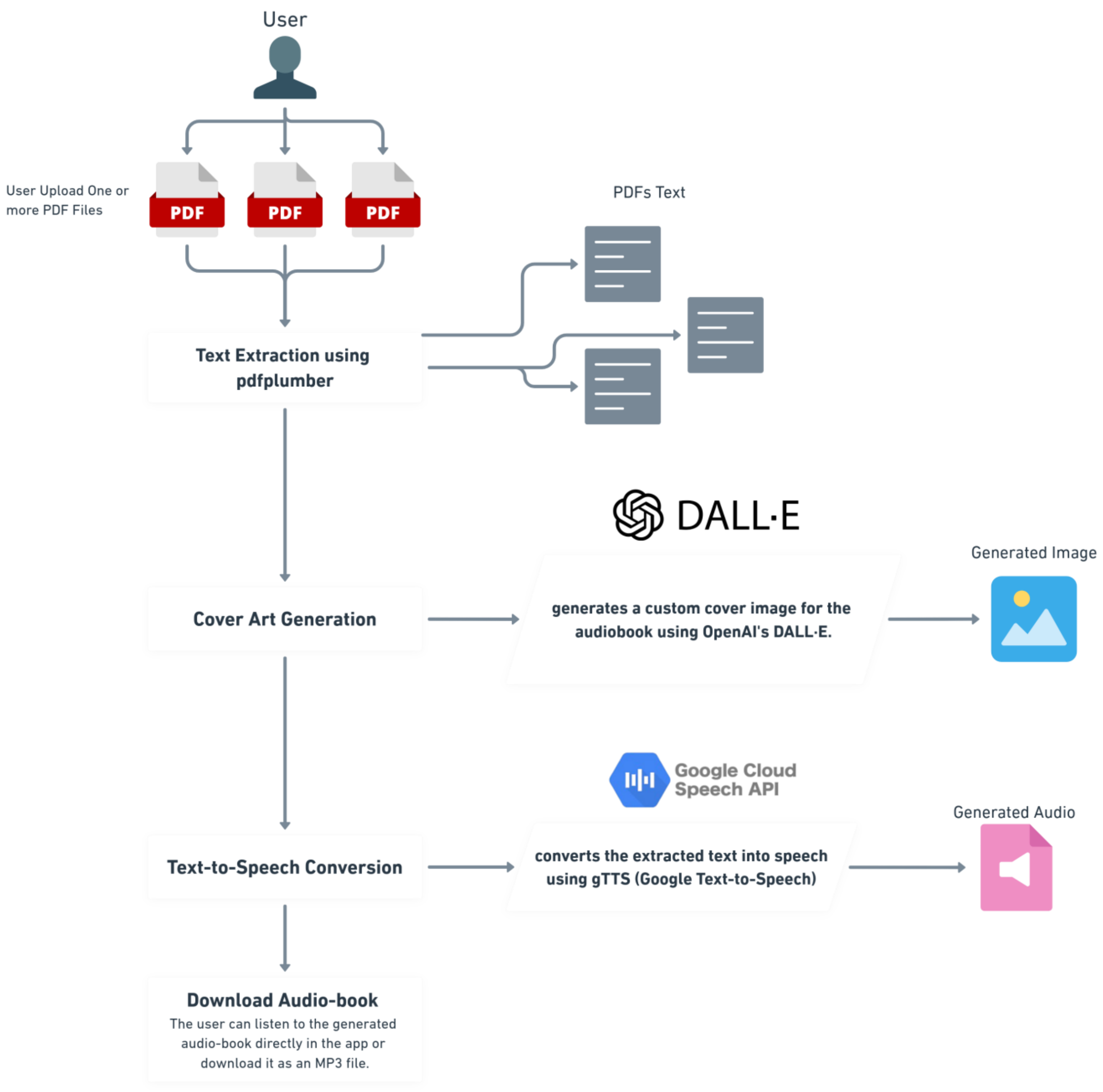

User
User Upload One or more PDF Files
PDF
PDF
PDF
PDFs Text
Text Extraction using pdfplumber
DALL·E
Generated Image
Cover Art Generation
generates a custom cover image for the audiobook using OpenAI's DALL·E.
Google Cloud Speech API
Generated Audio
Text-to-Speech Conversion
converts the extracted text into speech using gTTS (Google Text-to-Speech)
Download Audio-book
The user can listen to the generated audio-book directly in the app or download it as an MP3 file.

Fig 3: Audio-Book generator workflow diagram

2. Overflow Handling

The application is designed to handle various edge cases and overflows:

1. File Upload Validation:

- The application ensures that the uploaded files are in PDF format. If non-PDF files are uploaded, they are rejected.

2. Text Extraction Handling:

- If a PDF file contains non-text elements (e.g., images, scanned documents), the application gracefully handles the extraction process by skipping non-text content.

3. API Rate Limits:

- The application includes a `time.sleep(10)` delay before generating speech to avoid hitting API rate limits for the `gTTS` service.

4. Error Handling:

- If the OpenAI API key is missing or invalid, the application displays an error message and halts further execution.
- If the DALL·E model fails to generate an image, the application provides a fallback message and continues with the audio generation.

5. Memory Management:

- The application uses `BytesIO` to handle file uploads in memory, ensuring efficient memory usage for large PDF files.

3. Key Components

- The application consists of the following key components:

1. Text Extraction:

1. The `extract_text_from_pdf(uploaded_file)` function uses the pdfplumber library to extract text from uploaded PDF files.

2. Cover Art Generation:

The `generate_cover_art(cover_art_title)` function uses OpenAI's DALL·E model to generate a custom cover art image based on the user-provided title.

3. Text-to-Speech Conversion:

- The `text_to_speech(text, output_path)` function uses the gtts library to convert the extracted text or generated dialogue into an MP3 audio file.

4. Streamlit UI:

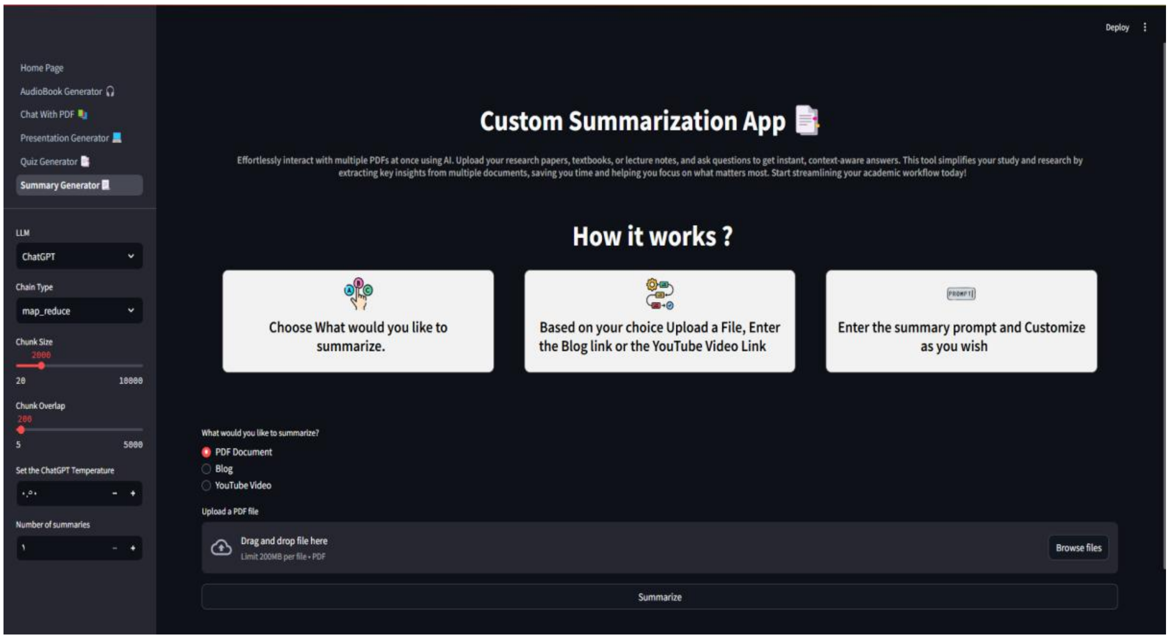
- The application uses Streamlit to create a user-friendly interface for uploading files, generating cover art, and downloading the audiobook.

5. CSS Styling:

- Custom CSS is applied to enhance the visual appeal of the Streamlit interface, including styled buttons, headers, and layout elements.

Custom Summarization Feature

the **Custom Summarization App**, allows users to summarize content from PDF documents, blogs, or YouTube videos using AI-powered tools. The application leverages OpenAI's GPT models, LangChain for text processing, and Streamlit for the user interface. Below is a detailed breakdown of the workflow, overflow handling, key components, and technical implementation.



Custom Summarization App

Effortlessly interact with multiple PDFs at once using AI. Upload your research papers, textbooks, or lecture notes, and ask questions to get instant, context-aware answers. This tool simplifies your study and research by extracting key insights from multiple documents, saving you time and helping you focus on what matters most. Start streamlining your academic workflow today!

How it works ?

Choose What would you like to summarize.

Based on your choice Upload a File, Enter the Blog link or the YouTube Video Link

Enter the summary prompt and Customize as you wish

LLM: ChatGPT

Chain Type: map_reduce

Chunk Size: 2000 (20 – 10000)

Chunk Overlap: 200 (5 – 5000)

Set the ChatGPT Temperature: ٠.٥٠

Number of summaries: ١

What would you like to summarize?

- [✓ PDF Document]
- Blog
- YouTube Video

Upload a PDF file

Drag and drop file here
Limit 200MB per file • PDF

1. Workflow Explanation

The application follows a structured workflow to summarize content from various sources:

1. User Selects Summarization Type:

1. The user chooses the type of content they want to summarize: **PDF Document**, **Blog**, or **YouTube Video**.

2. Content Input:

1. **PDF Document**: The user uploads a PDF file, and the application extracts its text.
2. **Blog**: The user provides a blog URL or manually pastes the blog content.
3. **YouTube Video**: The user provides a YouTube video URL, and the application fetches the transcript.

3. Text Processing:

- The extracted text is split into smaller chunks using the `RecursiveCharacterTextSplitter` to handle large documents efficiently.

4. Custom Prompt Input:

1. The user provides a custom prompt to guide the summarization process.

5. Summarization:

1. The application uses OpenAI's GPT models (ChatGPT or GPT-4) to generate summaries based on the custom prompt.
2. The summarization process can be customized using different chain types (map_reduce, stuff, refine).

6. Output:

1. The generated summaries are displayed to the user.
2. For YouTube videos, the application also embeds the video for reference.

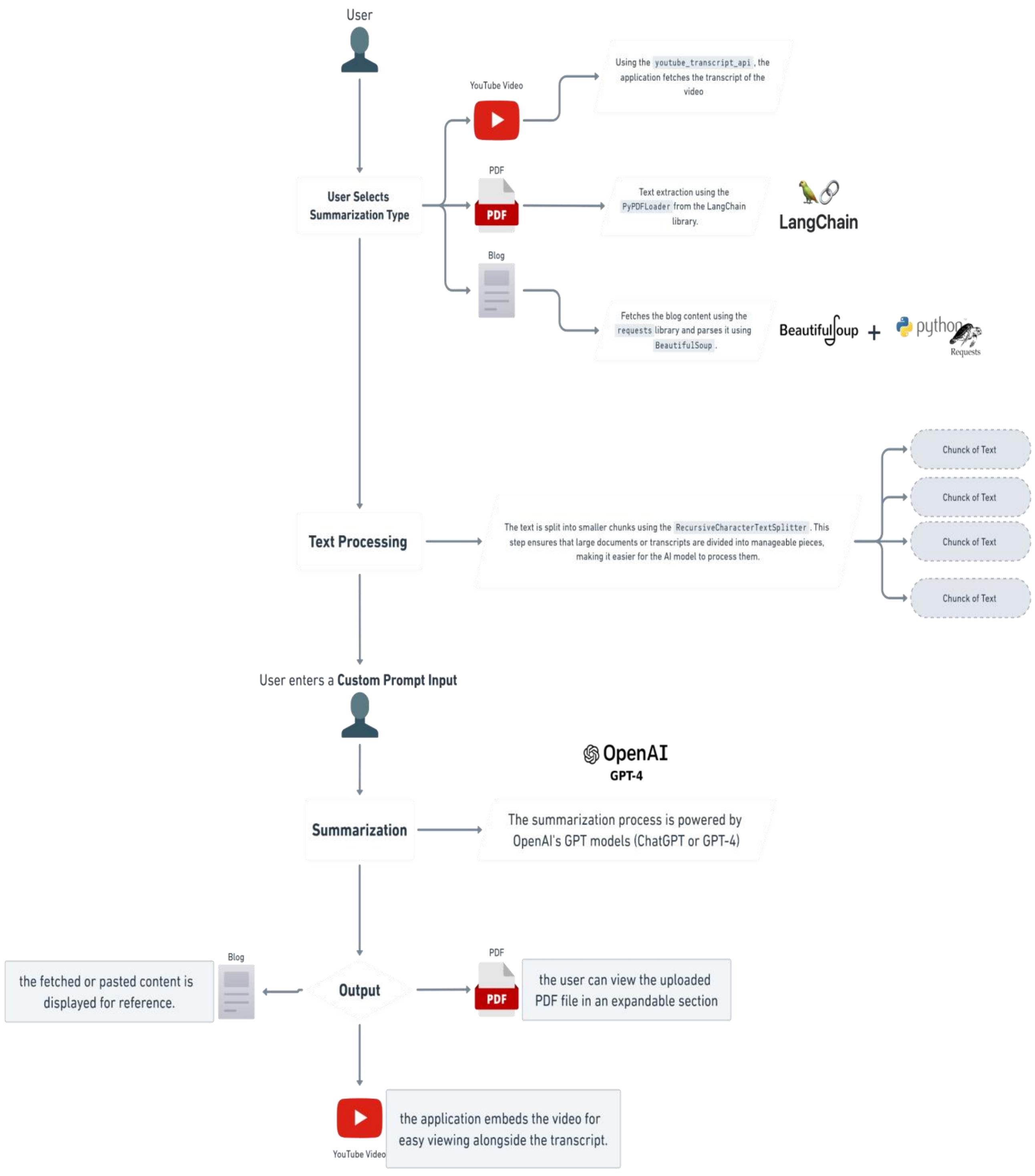

User
YouTube Video
Using the youtube_transcript_api, the application fetches the transcript of the video
User Selects Summarization Type
PDF
Text extraction using the PyPDFLoader from the LangChain library.
LangChain
Blog
Fetches the blog content using the requests library and parses it using BeautifulSoup.
BeautifulSoup + python Requests
Text Processing
The text is split into smaller chunks using the RecursiveCharacterTextSplitter. This step ensures that large documents or transcripts are divided into manageable pieces, making it easier for the AI model to process them.
Chunck of Text
Chunck of Text
Chunck of Text
Chunck of Text
User enters a Custom Prompt Input
OpenAI GPT-4
Summarization
The summarization process is powered by OpenAI's GPT models (ChatGPT or GPT-4)
Output
Blog
the fetched or pasted content is displayed for reference.
PDF
the user can view the uploaded PDF file in an expandable section
YouTube Video
the application embeds the video for easy viewing alongside the transcript.

Fig 4: Summary generator workflow diagram

2. Overflow Handling

The application is designed to handle various edge cases and overflows:

1. **File Upload Validation**:
    1. The application ensures that the uploaded files are in PDF format. If non-PDF files are uploaded, they are rejected.

2. **Content Fetching Errors**:
    1. If the blog URL is invalid or the content cannot be fetched, the application displays an error message and allows the user to manually paste the content.
    2. If the YouTube video URL is invalid or the transcript cannot be fetched, the application displays an error message.

3. **Text Chunking**:
    - The `RecursiveCharacterTextSplitter` ensures that large documents are split into manageable chunks, preventing memory overflow. **API Rate Limits**:

4. API Rate Limits:

The application uses `st.cache_data` to cache results and avoid redundant API calls, reducing the risk of hitting rate limits.

5. Error Handling:

1. If the OpenAI API key is missing or invalid, the application displays an error message and halts further execution.
2. If the summarization process fails, the application provides a fallback message and allows the user to retry.

3. Key Components

The application consists of the following key components:

1. Text Extraction:

- **PDF Documents**: The `PyPDFLoader` from LangChain is used to extract text from uploaded PDF files.
- **Blogs**: The `fetch_blog_content(url)` function uses `requests and BeautifulSoup` fetch and parse blog content.
- **YouTube Videos**: The `fetch_youtube_transcript(video_url)` function uses the `YouTubeTranscriptApi` to fetch video transcripts.

2. Text Splitting:

- The `RecursiveCharacterTextSplitter` splits the extracted text into smaller chunks for efficient processing.

3. Summarization:

- The `custom_summary` function uses OpenAI's GPT models to generate summaries based on the custom prompt.
- The summarization process can be customized using different chain types (map_reduce, stuff, refine).

4. Streamlit UI:

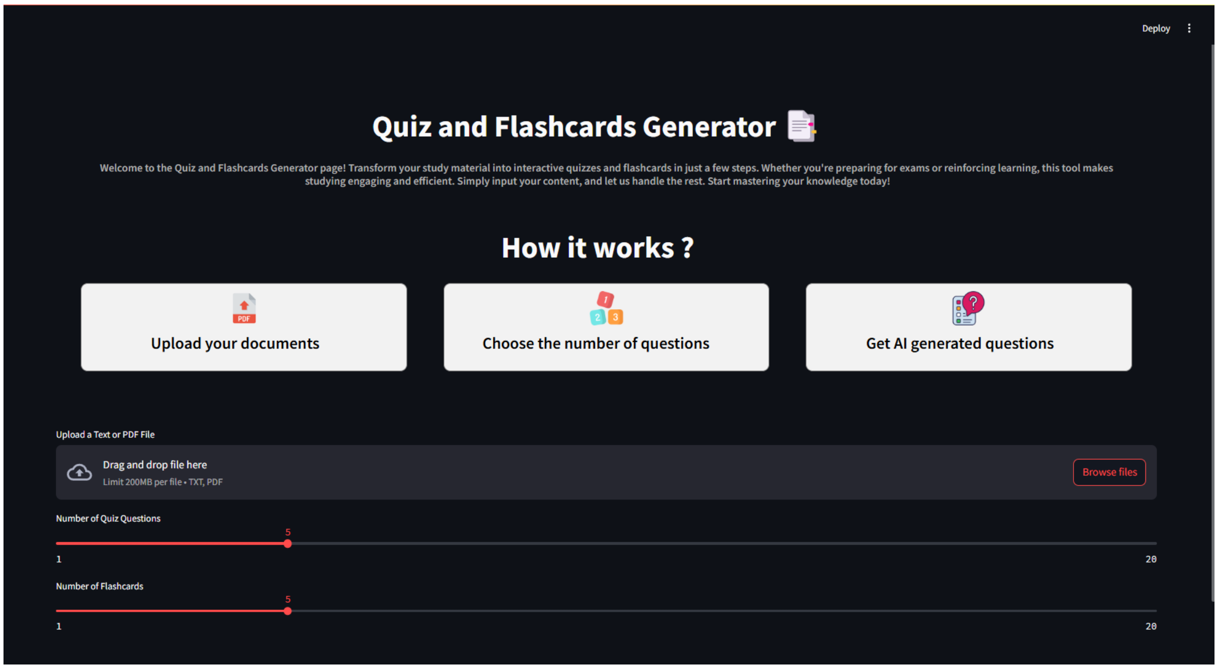
- The application uses Streamlit to create a user-friendly interface for uploading files, entering URLs, and viewing summaries.

5. CSS Styling:

- Custom CSS is applied to enhance the visual appeal of the Streamlit interface, including styled buttons, headers, and layout elements.

Quiz and Flashcards Generator

The **Quiz and Flashcards Generator** application, which allows users to generate quizzes and flashcards from uploaded text or PDF files using OpenAI's GPT-4 model. Below is a detailed breakdown of the workflow, overflow handling, key components, and technical implementation.



Quiz and Flashcards Generator

Welcome to the Quiz and Flashcards Generator page! Transform your study material into interactive quizzes and flashcards in just a few steps. Whether you're preparing for exams or reinforcing learning, this tool makes studying engaging and efficient. Simply input your content, and let us handle the rest. Start mastering your knowledge today!

How it works ?

1. Workflow Explanation

The application follows a structured workflow to generate quizzes and flashcards from user-uploaded content:

1.User Uploads a File:

- The user uploads a text or PDF file containing the content they want to use for generating quizzes and flashcards.
- The application supports .**txt** and .**pdf** file formats.

2. Text Extraction:

- For PDF files, the application uses the PyPDF2 library to extract text from each page.
- For **text files**, the application reads the content directly.
- The extracted text is displayed in a preview section for user verification.

3. User Customization:

- The user selects the number of quiz questions and flashcards they want to generate using sliders.

4. Quiz and Flashcards Generation:

- The application sends the extracted text to OpenAI's GPT-4 model with a prompt to generate:
  1. **Quiz**: Multiple-choice questions with four options and one correct answer.
  2. **Flashcards**: Questions and answers in a clear Q&A format.
- The generated content is displayed in the app interface.

5. Output:

- The quiz and flashcards are displayed in a user-friendly format.
- The user can view and interact with the generated content directly in the app.

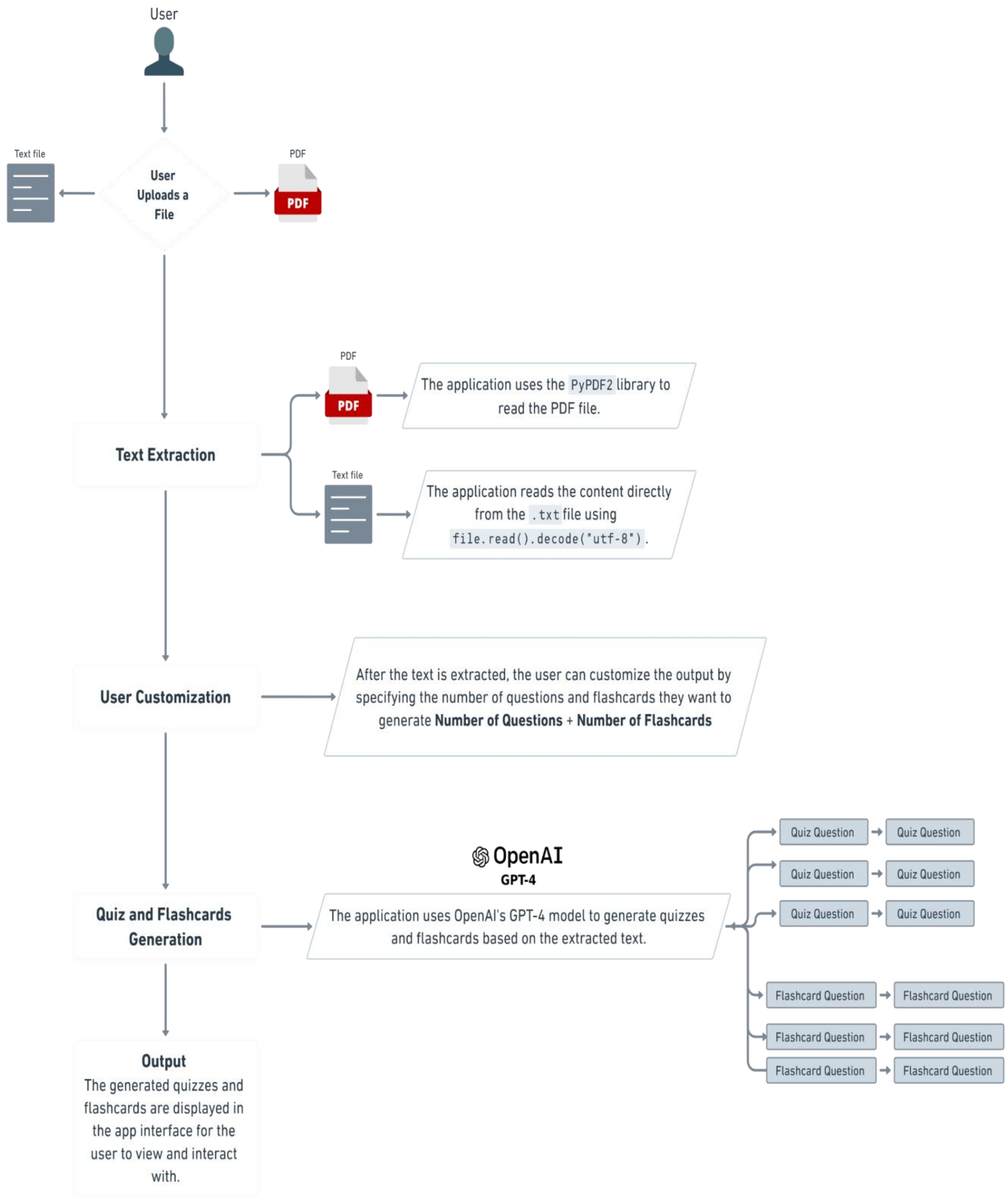

User
Text file
User Uploads a File
PDF
Text Extraction
PDF
The application uses the PyPDF2 library to read the PDF file.
Text file
The application reads the content directly from the .txt file using file.read().decode("utf-8").
User Customization
After the text is extracted, the user can customize the output by specifying the number of questions and flashcards they want to generate Number of Questions + Number of Flashcards
OpenAI
GPT-4
Quiz and Flashcards Generation
The application uses OpenAI's GPT-4 model to generate quizzes and flashcards based on the extracted text.
Quiz Question
Flashcard Question
Output
The generated quizzes and flashcards are displayed in the app interface for the user to view and interact with.

Fig 5: Quiz generator workflow diagram

2. Overflow Handling

The application is designed to handle various edge cases and overflows:

1.File Upload Validation:

- The application ensures that the uploaded files are in .txt or .pdf format. If unsupported files are uploaded, they are rejected.

2.Text Extraction Errors:

- If the PDF file contains non-text elements (e.g., scanned images), the application gracefully handles the extraction process by skipping non-text content.
- If the text file is empty or corrupted, the application displays an error message.

3.API Rate Limits:

- The application uses OpenAI's GPT-4 model, which has rate limits. If the limit is exceeded, the application displays an error message and prompts the user to try again later.

4.Large Text Handling:

- The application limits the preview of extracted text to the first 1000 characters to avoid overwhelming the UI. Users can expand the preview to view the full text.

5.Error Handling:

- If the OpenAI API key is missing or invalid, the application displays an error message and halts further execution.

1. If the GPT-4 model fails to generate content, the application provides a fallback message and allows the user to retry.

3. Key Components

The application consists of the following key components:

1. Text Extraction:

- PDF Files: The PyPDF2 library is used to extract text from uploaded PDF files.
- Text Files: The application reads the content directly from .txt files.

2. Quiz Generation:

- The `generate_quiz(text, num_questions=5)` function sends the extracted text to GPT-4 with a prompt to generate multiple-choice questions.
- The questions are formatted with four options and one correct answer marked with [Correct Answer].

3. Flashcards Generation:

- The `generate_flashcards(text, num_flashcards=5)` function sends the extracted text to GPT-4 with a prompt to generate flashcards.
- Each flashcard contains a question and its corresponding answer in a clear Q&A format.

4. Streamlit UI:

- The application uses Streamlit to create a user-friendly interface for uploading files, customizing the number of questions/flashcards, and viewing the generated content.
- Custom CSS is applied to enhance the visual appeal of the interface.

5. OpenAI Integration:

- The application integrates with OpenAI's GPT-4 model to generate quizzes and flashcards based on the provided text.

Dependencies and Requirements

- To ensure the **AcademiAI** platform functions seamlessly, the project relies on a variety of Python libraries and external APIs. Below is a categorized and organized list of all the dependencies used in the project, along with their purposes.

- **1. Core Libraries**
- These libraries are essential for basic functionality, file handling, and environment management.

Library	Purpose
os	Provides functions for interacting with the operating system.
dotenv	Loads environment variables from a .env file.
re	Handles regular expressions for text processing.
time	Provides time-related functions for delays and timing.
base64	Encodes and decodes data for file handling and display.
io	Provides tools for handling streams and file-like objects.
requests	Sends HTTP requests to interact with APIs and fetch web content.

3. OpenAI and AI Models

- These libraries enable integration with OpenAI's GPT models and other AI-driven functionalities.

Library	Purpose
openai	Integrates with OpenAI's GPT models for text generation and analysis.
langchain	Provides tools for working with language models, prompts, and chains.
ChatOpenAI	Enables interaction with OpenAI's chat-based models (e.g., GPT-4).
huggingface_hub	Integrates with Hugging Face models for additional AI capabilities.

4. Document and Text Processing

- These libraries handle file uploads, text extraction, and processing.

Library	Purpose
PyPDF2	Extracts text from PDF files.
pdfplumber	Provides advanced PDF text extraction and manipulation.
pptx	Handles PowerPoint files for presentation-related tasks.
BeautifulSoup	Parses HTML and XML content, useful for web scraping.
youtube_transcript_api	Fetches transcripts from YouTube videos.
TextFormatter	Formats YouTube transcripts into readable text.

5. Text Splitting and Embeddings

- These libraries are used for splitting text into chunks and generating embeddings for AI processing.

Library	Purpose
CharacterTextSplitter	Splits text into smaller chunks for processing.
RecursiveCharacterTextSplitter	Splits text recursively for better handling of large documents.
OpenAIEmbeddings	Generates embeddings using OpenAI's models.
HuggingFaceInstructEmbeddings	Generates embeddings using Hugging Face models.

6. Vector Stores, Memory and Conversational AI

- These libraries are used for splitting text into chunks and generating embeddings for AI processing + enable conversational interactions with the AI models.

Library	Purpose
FAISS	Stores and retrieves vector embeddings for efficient similarity search.
ConversationBufferMemory	Maintains memory for conversational AI interactions.
ConversationalRetrievalChain	Enables conversational retrieval of information using AI models.

8. Text-to-Speech (TTS)

- These libraries convert text into speech for audio-based functionalities.

Library	Purpose
gTTS	Converts text into speech using Google Text-to-Speech.

9. Image and File Handling

- These libraries handle image processing and file uploads.

Library	Purpose
PIL (Pillow)	Processes and manipulates images.
BytesIO	Handles in-memory binary streams for file uploads and processing.

- **Requirements File**

To install all the dependencies, you can use the following requirements.txt file:

```
streamlit==١,٢٦,٠
openai==٠,٢٨,٠
python-dotenv==١,٠,٠
PyPDF٣,٠,١==٢
pdfplumber==٠,٩,٠
python-pptx==٠,٦,٢١
beautifulsoup٤,١٢,٢==٤
youtube-transcript-api==٠,٦,١
langchain==٠,٠,٢٨٦
faiss-cpu==١,٧,٤
gTTS==٢,٣,٢
Pillow==١٠,٠,٠
requests==٢,٣١,٠
replicate==٠,١١,٠
huggingface-hub==٠,١٦,٤
```

Installation Instructions

1. **Clone the AcademiAI repository.**

2. **Navigate to the project directory.**

3. **Create a virtual environment:**

```
python -m venv venv
```

4. **Activate the virtual environment:**
    1. On Windows:

    ```
    venv\Scripts\activate
    ```

    2. On macOS/Linux:

    ```
    source venv/bin/activate
    ```

5. **Install the dependencies:**

```
pip install -r requirements.txt
```

6. **Create a (.env) file in the project root and add your OpenAI API key:**

```
OPENAI_API_KEY=your_openai_api_key_here
```

7. **Run the Streamlit app:**

```
streamlit run app.py
```