# TEXT TO PODCAST: AN LLM BASED PODCAST SYNTHESIZER FOR EFFECTIVE CONTENT DEVELOPMENT

# ABSTRACT

Podcasting is a rapidly growing medium, but content creators often struggle with the time-consuming process of scripting and recording episodes. Existing solutions require multiple tools, making the workflow fragmented. This project addresses these challenges by providing an integrated AI-driven system that automates podcast production from script generation to voice synthesis.

The system follows a structured process involving four key stages: text extraction and preprocessing, podcast script generation, script refinement, and text-to-speech (TTS) conversion. First, text and images are extracted from a given PDF document. The extracted text is tokenized and segmented for efficient processing, while image descriptions are seamlessly incorporated into the script. Next, a Large Language Model (LLM) generates a structured podcast script in a two-speaker conversational format. Speaker 1 (the host) introduces topics and asks questions, while Speaker 2 (the guest) provides explanations. This enhances engagement and ensures a natural dialogue flow. Image descriptions are integrated into the conversation without disrupting coherence.

The script then undergoes refinement to improve clarity, coherence, and accuracy. This process ensures a smooth conversational structure and enhances readability. Finally, the refined script is converted into speech using a TTS model, assigning distinct voices to speakers for a realistic and engaging podcast experience. By automating the entire process, this system significantly reduces the time and effort required for podcast production. Content creators can efficiently transform static documents into structured, high-quality audio content with minimal manual intervention, making professional podcasting more accessible and scalable.

# INDEX

# LIST OF FIGURES

# CHAPTER-1

# INTRODUCTION

## 1.1 INTRODUCTION TO THE PROJECT

In today's fast-paced digital world, podcasts have surged in popularity, offering an accessible and engaging way to share knowledge, stories, and ideas. Yet, for content creators, producing a podcast remains a time-intensive and fragmented process—scripting compelling dialogues, refining them for clarity, and recording audio often require multiple tools and significant effort. This project tackles that challenge head-on by delivering a transformative solution designed to simplify and accelerate podcast production.

Introducing an innovative, AI-powered system that streamlines the entire process from start to finish. This platform takes a simple PDF document—be it an article, report, or research paper—and automates the journey from text to audio. By leveraging cutting-edge technologies like large language models (LLMs) and text-to-speech (TTS) synthesis, it extracts meaningful content, crafts structured podcast scripts, refines them for natural flow, and generates professional-grade audio—all within a single integrated solution.

The process unfolds in four seamless stages. First, the system extracts and preprocesses text and images from the PDF, breaking the content into manageable chunks for efficient handling. Next, it generates an initial podcast script, weaving in image descriptions to create a lively, conversational dialogue between a host and guest. Then, the script is refined to ensure accuracy, coherence, and engagement, aligning perfectly with the source material. Finally, advanced TTS technology brings the script to life, producing a polished audio episode ready for listeners.

## 1.2 INTRODUCTION TO GENERATIVE AI:

Generative AI, or GenAI, represents a groundbreaking frontier in artificial intelligence, where machines don't just analyze or process data—they create it. Unlike traditional AI systems that focus on recognizing patterns or making predictions, GenAI leverages

advanced algorithms, such as large language models (LLMs) and neural networks, to generate new content, from text and images to audio and beyond. Powered by vast datasets and sophisticated training techniques, these systems can produce human-like outputs, making them invaluable tools for innovation across industries.

The rise of GenAI has transformed how we approach creativity and production. In fields like content creation, education, and entertainment, it's breaking down barriers by automating complex tasks that once required extensive human effort. Tools like GPT-based models, diffusion models for images, and text-to-speech synthesizers exemplify GenAI's versatility, enabling everything from writing articles to composing music—all with remarkable speed and quality. Its ability to understand context, generate coherent narratives, and mimic natural communication has made it a cornerstone of modern technological advancements.

In the context of this project, Generative AI is the beating heart of an automated podcast production system. By harnessing LLMs to craft engaging scripts from raw PDF documents and integrating TTS technology to produce lifelike audio, GenAI streamlines the creative process for content developers. It doesn't just replicate—it enhances, refining raw text into structured dialogues and breathing life into static documents. This introduction to GenAI sets the stage for understanding how it empowers our solution, turning written ideas into captivating podcast episodes with unprecedented efficiency.

### 1.2.1 ADVANTAGES OF GENERATIVE AI:

- **Saves Time**: Automates content creation, turning hours of work into minutes.
- **Boosts Creativity**: Generates fresh ideas and enhances human output.
- **Scales Easily**: Produces one or many pieces of content with consistent quality.
- **Cuts Costs**: Reduces need for expensive tools, studios, or personnel.
- **Ensures Consistency**: Delivers uniform tone and accuracy across outputs

## 1.2.2 DISADVANTAGES OF GENERATIVE AI:

- **Uneven Quality**: Output may not always feel human-like.
- **Needs Good Data**: Poor input leads to poor results.
- **Overuse Risk**: Could weaken human creativity.

## 1.1.3 APPLICATIONS:

- **Content Creation**: GenAI excels at producing written and spoken content. In this project, it transforms PDFs into engaging podcast scripts, crafting dialogues between a host and guest—proof of its knack for turning static text into dynamic narratives for blogs, articles, or social media too.
- **Audio Production**: By pairing GenAI with text-to-speech (TTS) technology, it generates lifelike audio. This system converts refined scripts into polished podcast episodes, showcasing how it can also create audiobooks, voiceovers, or virtual assistants with minimal human effort.
- **Image and Video Generation**: Beyond text and sound, GenAI can produce visuals. While this project integrates image descriptions into scripts, tools like DALL·E or Stable Diffusion highlight its ability to craft artwork, animations, or even video storyboards for multimedia projects.
- **Education and Training**: GenAI personalizes learning by generating tailored materials. Imagine auto-converting textbooks into podcast-style lessons—as demonstrated here—or creating interactive quizzes and simulations for students and professionals.
- **Gaming and Entertainment**: It powers immersive experiences by generating storylines, dialogue, or music. This project's scriptwriting mirrors how GenAI could design game narratives or compose soundtracks, enhancing creative industries.

# CHAPTER – 2

# PROBLEM SPECIFICATION

## 2.1 EXISTING SYSTEM AND DISADVANTAGES:

Existing systems for podcast generation, including Wondercraft AI, Podcastle, Descript, Resemble AI, Adobe Podcast, and HyperWrite AI, are theoretically anchored in the fusion of artificial intelligence and audio production technologies to streamline or enhance podcast creation. Drawing on natural language processing (NLP), large language models (LLMs), and text-to-speech (TTS) synthesis, these platforms aim to automate labor-intensive tasks like scripting, voice generation, and audio editing, while leveraging signal processing to refine sound quality. Their shared hypothesis posits that AI can lower the barriers of technical expertise and time traditionally required, making podcasting more accessible and efficient. Whether through script generation (HyperWrite), synthetic voice creation (Resemble), or integrated editing (Descript), these systems seek to meet rising podcast demand by harnessing generative AI and audio engineering, theorizing that automation can either mimic human creativity or amplify production polish, depending on their focus.

**Cost and Accessibility**: A primary drawback of these systems is their high cost and limited accessibility, which restrict their reach among diverse content developers. Wondercraft AI's subscription-based model, while offering quick text-to-podcast conversion, prices out small creators who can't afford recurring fees, undermining its accessibility goal. Descript locks advanced features like "Overdub" voice cloning behind expensive tiers, catering to well-funded users rather than the broader podcasting community. Adobe Podcast's professional-grade suite demands steep licensing costs, targeting established producers over independents, and Resemble AI's pricing escalates with custom voice usage, making large-scale production prohibitively expensive. These financial hurdles clash with podcasting's ethos as a low-barrier medium, leaving budget-conscious creators at a disadvantage compared to free or low-cost alternatives.

**Lack of Full Automation and Integration**: Another significant limitation is the incomplete automation and fragmented workflows these systems impose, hindering seamless podcast production. Wondercraft AI generates scripts and audio but lacks a refinement step, often requiring manual edits to ensure coherence, while Podcastle's mix of TTS and editing tools stops short of a full document-to-podcast pipeline, leaving input preparation to users. Descript's text-based editing excels but doesn't automate script creation from documents, and HyperWrite AI's text-only output necessitates external TTS integration, breaking the process into disjointed steps. Adobe Podcast focuses on post-processing without content generation, and Resemble AI's TTS specialization demands additional tools for scripting or editing, fragmenting the workflow further. This lack of integration contrasts with the need for an end-to-end solution, reducing efficiency for creators.

## 2.2 PROPOSED SYSTEM AND ADVANTAGES

The proposed system, "TEXT TO PODCAST: AN LLM BASED PODCAST SYNTHESIZER FOR EFFECTIVE CONTENT DEVELOPMENT" is theoretically rooted in a four-stage automation pipeline designed to transform PDF documents into podcast episodes efficiently, leveraging artificial intelligence (AI) to streamline content creation. The tokenization stage, utilizing natural language processing (NLP) and tools like tiktoken, preprocesses text and images extracted via pdfplumber and fitz into manageable chunks, hypothesizing that structured segmentation enables scalable handling of diverse inputs. This foundational step ensures accessibility by simplifying complex documents for subsequent stages, offering a time-saving advantage by reducing manual preparation to minutes, surpassing the fragmented workflows of systems like Descript or Wondercraft AI.

Building on this, the script generation and refinement stages employ the Google Gemini large language model (LLM) to craft and polish host-guest dialogues, theorizing that generative AI can emulate human creativity while iterative processing ensures quality. Script generation converts tokenized content into structured narratives, uniquely integrating image descriptions to enrich context—a novel application of multimodal AI—while refinement enhances coherence and alignment with the source, addressing common

AI output flaws like lack of nuance. This provides a creativity advantage, producing engaging, image-enhanced dialogues absent in text-only tools like HyperWrite AI or unrefined systems like Wondercraft AI. The system's consistency advantage shines here, delivering polished scripts that maintain fidelity to the original document, offering a reliability that fragmented or generic alternatives struggle to achieve.

The audio generation stage completes the pipeline, theorizing that text-to-speech (TTS) synthesis, initially with gTTS and supported by pydub for audio assembly, can translate refined scripts into listener-ready episodes with multi-speaker dynamics. This stage assumes synthetic voices, enhanced by signal processing, can replicate podcast intimacy, with potential upgrades to premium TTS like Google Cloud TTS for scalability. The system's cost-effectiveness advantage stems from using free tools, eliminating the high subscription costs of Adobe Podcast or Descript, while its scalability allows processing of large PDFs into multiple episodes—a flexibility surpassing Resemble AI's costly scaling. By offering an end-to-end, affordable solution, the system provides versatility, handling text and images in one platform, outstripping the narrow focus of existing tools and redefining podcast production for content developers.

## 2.3 SCOPE OF THE SYSTEM

The scope of the proposed system, "From PDF to Podcast – Automating Content Creation with AI," encompasses the development and implementation of an integrated, AI-driven platform that automates the transformation of PDF documents into fully realized podcast episodes, targeting content developers such as educators, marketers, writers, and researchers. Its primary objective is to streamline podcast production through a four-stage pipeline—tokenization, script generation, refinement, and audio generation—leveraging tools like Google Gemini LLM for script creation, gTTS for audio synthesis, and a Streamlit interface for user interaction. The system is designed to process textual and visual content from PDFs, generating structured host-guest dialogues that incorporate image descriptions, refining them for coherence, and outputting downloadable audio files and scripts. This scope includes providing a cost-effective, accessible solution that eliminates

the need for manual scripting, recording, or fragmented workflows, aiming to serve users who seek to repurpose static documents—such as academic papers, reports, or articles—into engaging audio formats efficiently.

Within its current scope, the system targets small to medium-scale content creation needs, supporting documents of varying lengths and complexity, from single-page articles to multi-page reports, with the capacity to produce multi-episode series. It is intended for individual creators, small businesses, or educational institutions lacking resources for traditional podcasting, offering versatility across genres like educational lectures, business insights, or narrative storytelling. However, its scope is constrained by the limitations of free TTS tools like gTTS, which impose rate limits and affect scalability for large-scale production (e.g., processing extensive PDFs or generating dozens of episodes), and the synthetic voices' limited emotional depth compared to human narration. The system's initial deployment focuses on English-language content, with the Streamlit interface providing a user-friendly experience for tech-savvy and novice users alike, though it assumes basic familiarity with file uploads and digital interfaces.

Looking forward, the scope of the system extends to potential enhancements that could broaden its applicability and impact. Integration of premium TTS solutions, such as Google Cloud TTS, could overcome rate limits and improve voice quality, enabling large-scale commercial use or real-time processing for live podcasting applications. The framework supports future expansion into multilingual support by adapting the LLM and TTS components, catering to global audiences, and could incorporate advanced features like user-customizable voice styles or automated episode summarization for enhanced listener engagement. Additionally, the system's scope includes potential adaptation for other input formats (e.g., Word documents, web articles) and output types (e.g., audiobooks, voiceovers), positioning it as a versatile tool beyond podcasting. While currently a proof-of-concept for automated content transformation, its modular design and AI foundation offer a scalable platform for future research and development, aiming to redefine content creation in the digital audio landscape.

# CHAPTER 3

# SYSTEM ANALYSIS

The proposed system requires several components to function effectively. A large language model (LLM) is necessary for script generation, while a text-to-speech (TTS) model with multi-speaker synthesis ensures realistic voice outputs. A topic extraction module is needed to structure content logically, and an AI-driven personalization module adapts content based on listener preferences. Additionally, cloud storage and processing capabilities are essential for efficient deployment and scalability.

The feasibility of the system is analyzed across multiple dimensions. From a technical standpoint, the system is viable using existing AI models such as GPT-4, Llasa-3B, and Gemini, which provide advanced language generation and speech synthesis capabilities. Economically, although initial development costs may be significant, automation reduces long-term operational expenses by minimizing manual intervention. Operationally, the system is designed to be user-friendly and adaptable to various podcast formats, making it an attractive option for content creators.

The system follows a modular design to ensure efficiency and scalability. The script generation module utilizes NLP models to create structured content, while the speech synthesis module converts text into expressive speech using multi-speaker TTS technology. A content structuring module implements topic modeling techniques to organize episodes cohesively, and a personalization engine adjusts the content dynamically based on audience engagement. Lastly, a post-processing module integrates multimedia elements such as image descriptions and sound effects to enhance the overall listening experience.

Testing and maintenance are crucial to ensuring the system's reliability and performance. Unit testing is conducted on individual modules, including script generation, TTS, and topic modeling, to verify their functionality. Integration testing ensures seamless interaction between these modules, while user testing gathers feedback from podcast creators and listeners to refine the system. Maintenance involves regular updates, model

fine-tuning, bug fixes, and improvements to enhance accuracy, engagement, and usability over time.

## 3.1 UML DIAGRAMS

A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.

### 3.1.1 USE CASE DIAGRAM

- These are a set of use cases, actors, and their relationships.
- A use case represents a particular functionality of a system.
- These controllers are known as **actors**.
- The two main components of a use case diagram are use cases and actors.

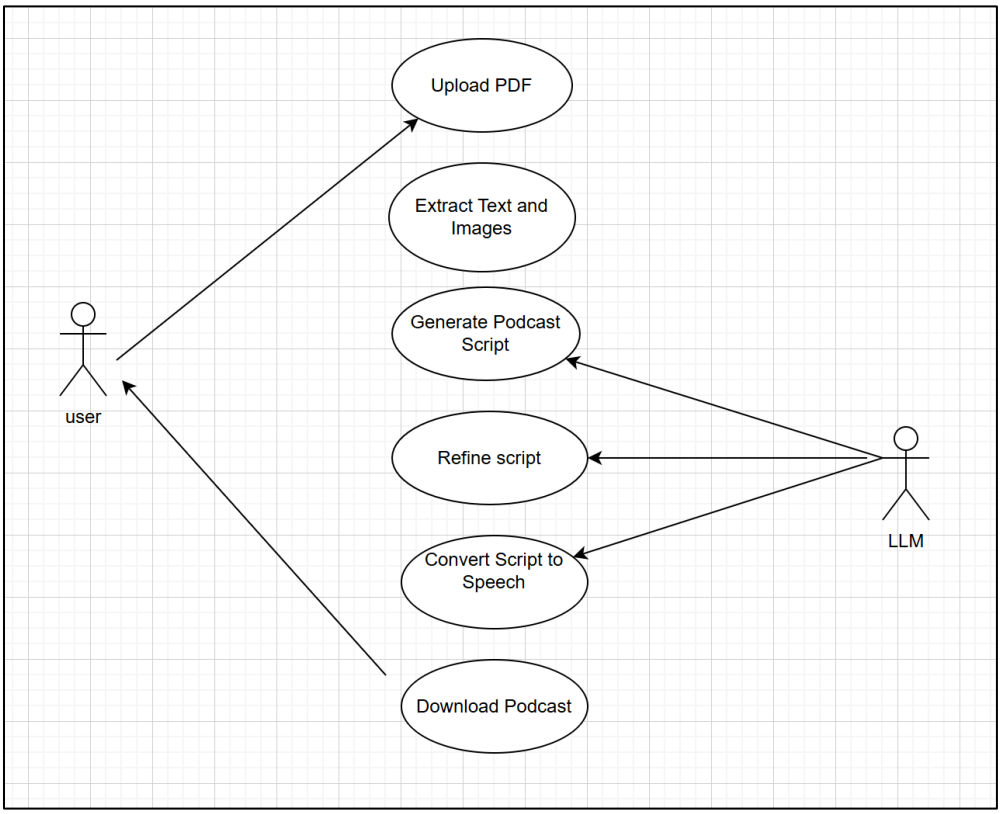


*Figure 1 Showing Use Case Diagram*

### 3.1.2 Class diagram

- Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented language

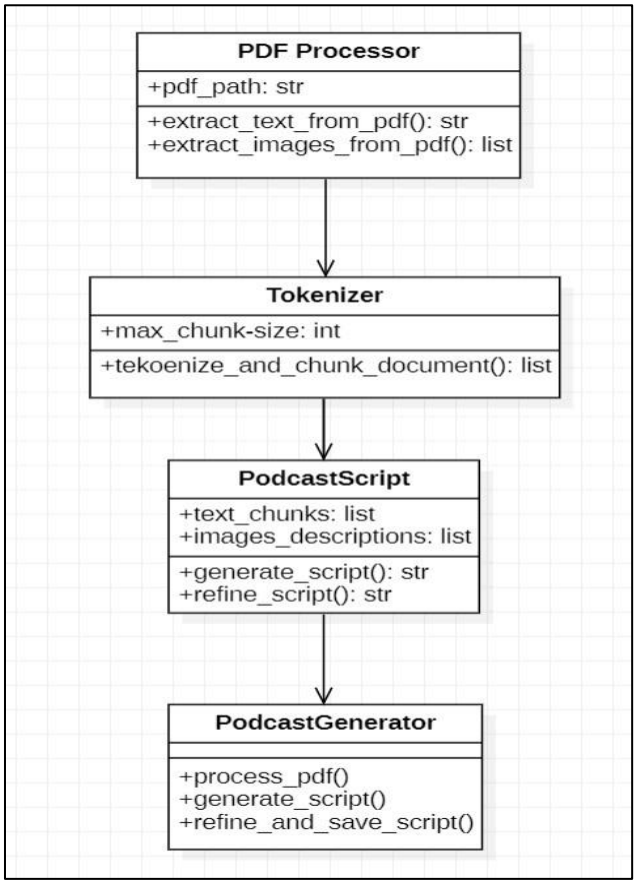


*Figure 2 Showing Class Diagram*

### 3.1.3 ACTIVITY DIAGRAM

- Activity diagram describes the flow control of a system.
- The flow can be sequential, concurrent, or branched.
- It consists of activities and links.
- Activities are nothing but the functions of a system.
- An activity is essentially a flow chart.
- This is prepared to have an idea of how the system will work when executed.

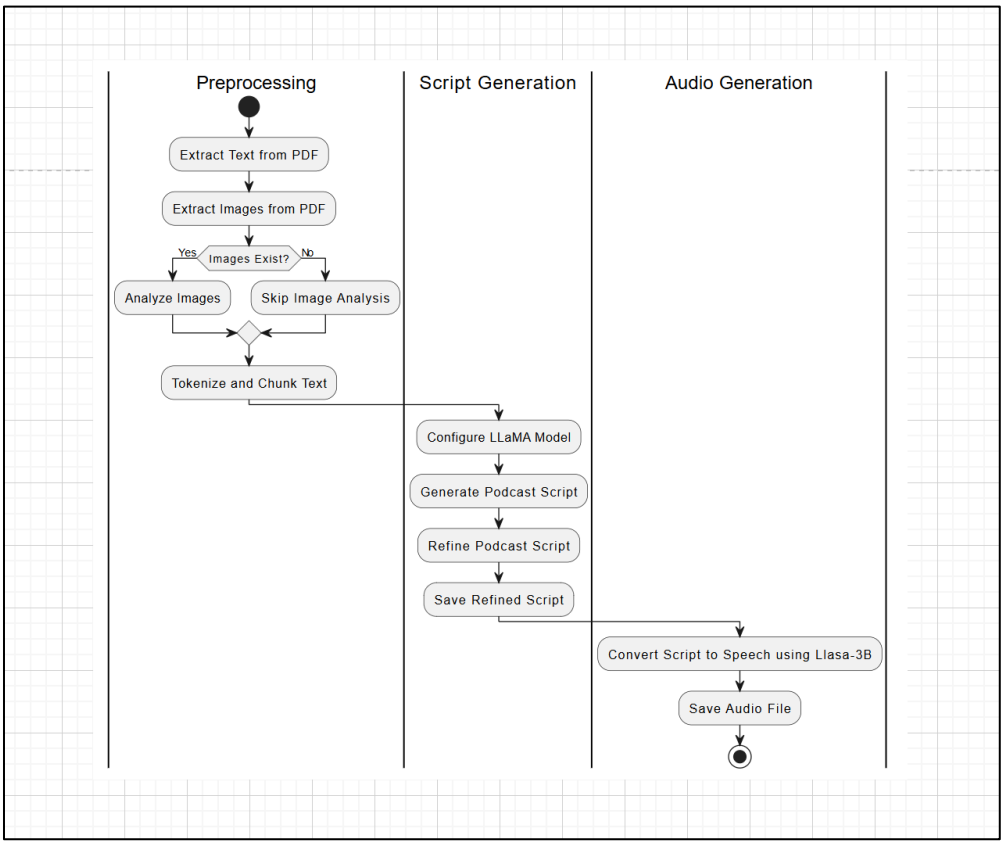


*Figure 3 Showing Activity Diagram*

### 3.1.4 SEQUENCE DIAGRAM

- Shows the order of processes in the system.
- Represents interactions between components like text extraction, script generation, and audio synthesis.
- Depicts how data flows between different modules.
- Helps developers understand the execution sequence of tasks.

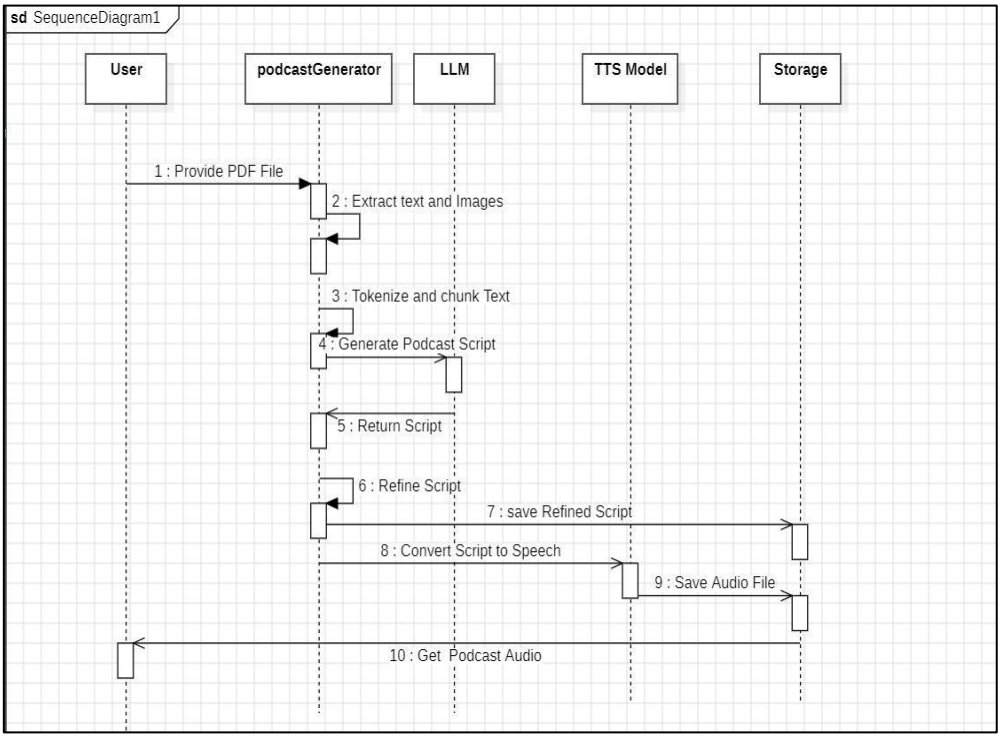


*Figure 4 Showing Sequence Diagram*

# CHAPTER 4

## REQUIREMENT SPECIFICATION

## 4.1 FUNCTIONAL REQUIREMENTS

- **Document Input:** The system must allow users to upload or input written documents from which podcast scripts will be generated.
- **Tokenization:** It should tokenize the input documents to extract meaningful information relevant to podcast content.
- **Script Generation:** The system must automatically generate an initial podcast script based on the tokenized content.
- **Script Refinement:** Users should be able to refine the generated script for clarity, coherence, and flow.
- **Voice Synthesis:** The system must integrate voice synthesis capabilities to convert the refined script into audio format.

## 4.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements (NFRs) are the qualities that the system must possess in order to fulfill its functional requirements. For podcast synthesizer , some of the important non-functional requirements are:

- **Usability:** The interface should be user-friendly, allowing users to input PDFs easily and receive output in both text and audio formats.
- **Consistency**: The system should ensure that content generated from a document is consistent throughout. There should be no discrepancy in the voice or tone of the podcast, and no content should be missing or altered unexpectedly.

## 4.3. HARDWARE REQUIREMENTS:

The Hardware consists of the physical components of the computer that input storage processing control, output devices. Most hardware only has operating system requirements or compatibility. For example, a printer may be compatible with Windows XP but not compatible with newer versions of Windows like Windows 10, Linux, or the Apple macOS.

The kind of hardware used in the project is

- Processor: Intel Core i3 or higher
- RAM: 8GB or higher
- Hard Disk: 256GB or higher

## 4.4 SOFTWARE REQUIREMENTS

**Software requirements** refer to the specifications and conditions that a software system must meet to function correctly and efficiently. These requirements define the expected behavior, performance, and constraints of the system, serving as a guide for developers, testers, and stakeholders throughout the software development lifecycle.

- Operating system: Windows 10/11
- Development tools: Google colab or jupyter notebook
- Programming language: Python
- Libraries and packages:
    - google.generativeai – Interacts with Google's Generative AI (Gemini) to generate structured podcast scripts based on extracted text.
    - pdfplumber – Extracts text from PDF documents for processing into podcast content.
    - fitz (PyMuPDF) – Extracts images from PDFs, enabling the system to describe visuals within the podcast script.
    - tiktoken – Efficiently tokenizes text for processing by AI models, reducing computational overhead.

- nltk.tokenize.word_tokenize – Splits text into words for easier processing and analysis.
- nltk.corpus.stopwords – Filters out common stopwords to improve text quality and topic modeling accuracy.
- gensim.models.LdaModel – Implements topic modeling using Latent Dirichlet Allocation (LDA) to identify key themes in podcast content.
- gtts (Google Text-to-Speech) – Converts AI-generated podcast scripts into realistic speech audio.

# CHAPTER 5

# SYSTEM DESIGN

## 5.1 INTRODUCTION

System design defines the architecture, components, and interactions within the podcast generation system. It ensures that the system functions efficiently, integrating text processing, AI-driven script generation, and text-to-speech conversion into a seamless workflow. The design includes data flow, component interactions, and processing logic to maintain coherence, scalability, and high-quality output

## 5.2 SYSTEM ARCHITECTURE

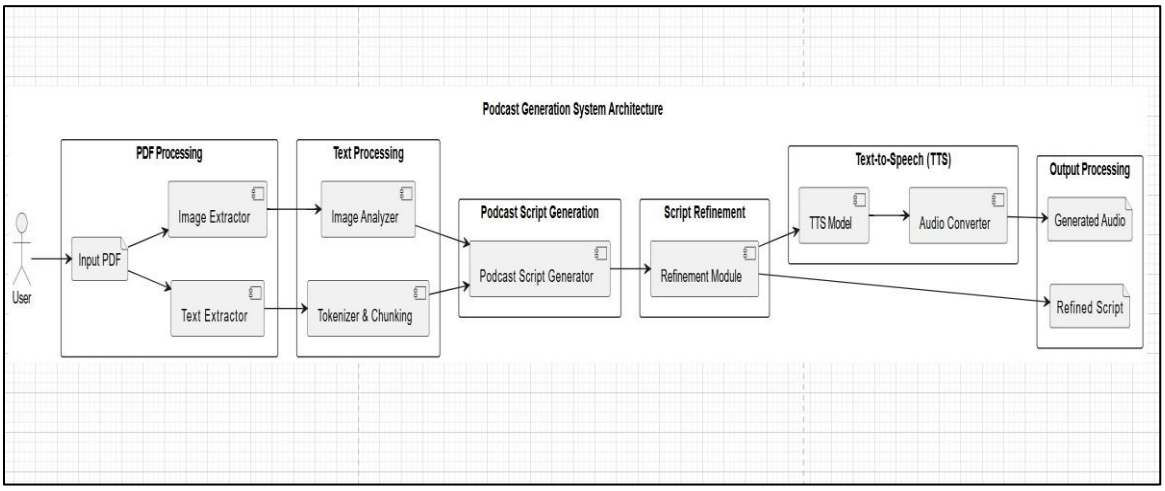


*Figure 5 System Architecture*

### 5.2.1 ARCHITECTURE DESCRIPTION

- **Overall Structure**: Represents the complete workflow of the podcast generation system, showing how different components interact.
- **PDF Processing**: Extracts images and text from the provided PDF document.
- **Text Processing**: Tokenizes and chunks the extracted text for structured analysis and AI processing.

- **Script Generation**: AI generates a podcast script by incorporating chunked text and image descriptions to create an engaging narrative.
- **Refinement**: Ensures consistency, coherence, and natural flow in the generated script by refining grammar and structure.
- **Audio Generation**: Converts the final refined script into an audio podcast using a text-to-speech (TTS) engine, ensuring a clear and engaging listening experience.

### 5.2.2 Key Stages in the Podcast Synthesizer

Podcast generation using AI automates the process of converting text-based documents into engaging audio content. This system involves multiple stages, including tokenization, script generation, refinement, and audio synthesis to create a seamless listening experience.

**Introduction**

Podcast generation using AI automates the process of converting text-based documents into engaging audio content. This system involves multiple stages, including tokenization, script generation, refinement, and audio synthesis to create a seamless listening experience.

**1. Tokenization**

Tokenization is the first stage where the extracted text from the PDF is broken down into smaller units, such as words or sentences. This process ensures that the text is structured and manageable for further processing. Chunking is then applied to divide the text into meaningful segments, which are later used for script generation. This step helps maintain context and ensures smooth AI processing.

**2. Script Generation**

In this stage, the AI model processes the tokenized and chunked text to generate a structured podcast script. The script is designed to be engaging and informative, incorporating both the extracted text and any available image descriptions. The AI ensures a conversational tone, making the content suitable for audio presentation. The generated

script follows a defined structure, including an introduction, main content, and a conclusion.

### 3. Script Refinement

Once the script is generated, it undergoes a refinement process to ensure coherence, accuracy, and a natural flow. The refinement step corrects any inconsistencies, improves grammar, and enhances clarity. The goal is to make the script more engaging and structured while maintaining the original context of the extracted document. This step ensures that the podcast script is ready for narration without awkward phrasing or disjointed segments.

### 4. Audio Generation

The final stage involves converting the refined script into an audio podcast using a text-to-speech (TTS) engine. The TTS model assigns distinct voices to different speakers, ensuring a dynamic and natural listening experience. The generated audio is then processed further, incorporating pauses and emphasis where necessary to improve clarity and engagement. The final output is a high-quality podcast episode that accurately conveys the original content in an auditory format.

# CHAPTER-6

## IMPLEMENTATION

## 6.1 INTRODUCTION

The implementation of the podcast generation system is designed to efficiently transform textual content into engaging audio podcasts using AI-driven techniques. The system leverages natural language processing, machine learning, and text-to-speech conversion to automate script generation and voice synthesis. By integrating PDF processing, topic modeling, and AI-based script refinement, the system ensures high-quality podcast episodes with minimal manual intervention. The implementation is structured into distinct stages, each focusing on a crucial aspect of the workflow, from document analysis to final audio output.

## 6.2 IMPLEMENTATION:

The implementation of the podcast generation system follows a structured approach to ensure efficiency, accuracy, and automation. The system is developed using Python, integrating various AI models and libraries for text processing, natural language generation, and text-to-speech conversion. The entire process is modular, allowing for seamless execution from document processing to podcast generation.

1. **PDF Processing:**

The first stage involves extracting content from PDFs using pdfplumber for text extraction and PyMuPDF (fitz) for extracting images. The extracted text serves as the input for further processing, while the images are analyzed for descriptions that can be incorporated into the podcast script. This ensures that both textual and visual content from the document are accurately represented in the final podcast.

2. **Text Processing:**

Once the text is extracted, it is tokenized and chunked to ensure structured and meaningful script generation. Tiktoken is used for efficient tokenization, breaking the text into smaller segments that can be easily processed by the AI model. The chunked text ensures smooth script generation, maintaining logical coherence and readability.

3. **Script Generation:**

The processed text is then used as input for the Google Gemini AI model, which generates a structured podcast script. The script incorporates both text and image descriptions, ensuring an engaging and informative dialogue. The AI model generates content in a conversational format, with distinct speaker roles, maintaining clarity and listener engagement. The system enforces a structured format with Introduction, Main Content, and Conclusion to ensure consistency across episodes.

4. **Script Refinement:**

To ensure high-quality content, the generated script undergoes a refinement step that corrects grammatical errors, improves sentence structure, and enhances logical coherence. The AI refines the script to maintain a natural flow and ensure alignment with the original document. This step guarantees that the script is engaging, well-structured, and suitable for audio presentation.

5. **Audio Generation:**

The final script is then converted into an audio podcast using Google Text-to-Speech (gTTS). Different voices are assigned to each speaker, ensuring a dynamic and engaging listening experience. The generated audio is further processed using pydub, which allows for the addition of pauses, emphasis, and enhancements to improve clarity and engagement. The output is a high-quality podcast file ready for distribution.

6. **Question Generation Using Topic Modeling:**

The system incorporates an automated question generation process using topic modeling techniques. The extracted text is analyzed using Gensim's LDA (Latent Dirichlet Allocation) to identify key topics. Based on these topics, the system generates 5-7 relevant questions that align with the podcast script content. These questions can be used for

audience engagement, knowledge assessment, or as discussion points in follow-up episodes. The AI model ensures that the generated questions are meaningful and contextually relevant to the extracted document.

7. **Topic Extraction:**

The first step in question generation involves text preprocessing, where the extracted text is cleaned by removing stopwords, punctuation, and irrelevant characters. The preprocessed text is then tokenized and converted into a bag-of-words representation. The Gensim library's LDA model is applied to analyze the document and extract the most relevant topics. Each topic consists of a set of keywords that represent its core idea.

8. **Question Formation:**

Once the topics are identified, the system uses AI-based language models to generate thought-provoking and informative questions. The extracted topics serve as the basis for formulating questions that align with the main themes of the document. The questions are structured to cover different aspects of the topic, ensuring a comprehensive understanding of the subject matter. AI ensures that the generated questions maintain grammatical accuracy and contextual relevance.

9. **Deployment and Execution**

The entire system is deployed using Streamlit, a web-based interactive framework that allows users to upload PDFs and generate podcasts with a user-friendly interface. Users can configure settings such as chunk size, episode length, and API key input within the Streamlit app. The system processes the input document in the background and returns downloadable podcast scripts and audio files.

The final output is a fully automated podcast, which can be accessed and downloaded directly from the Streamlit interface, making AI-driven podcast generation more accessible and efficient.

## 6.3 SAMPLE CODE:

### 6.3.1 EXTRACTING TEXT AND IMAGES FROM PDF:

```
def extract_text_from_pdf(pdf_path):
    text = ""
    try:
        with pdfplumber.open(pdf_path) as pdf:
            for page in pdf.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n"
        return text.strip()
    except Exception as e:
        print(f"Error extracting text from {pdf_path}: {e}")
        raise


def extract_images_from_pdf(pdf_path, output_folder="extracted_images"):
    os.makedirs(output_folder, exist_ok=True)
    images = []
    try:
        pdf_document = fitz.open(pdf_path)
        for page_number in range(len(pdf_document)):
            page = pdf_document[page_number]
            image_list = page.get_images(full=True)
            for img_index, img in enumerate(image_list):
                xref = img[0]
                base_image = pdf_document.extract_image(xref)
                image_bytes = base_image["image"]
                image_ext = base_image["ext"]
                image_filename = os.path.join(output_folder,
f"page{page_number+1}_img{img_index+1}.{image_ext}")
```

```
                with open(image_filename, "wb") as img_file:
                    img_file.write(image_bytes)
                images.append(image_filename)
        return images
    except Exception as e:
        print(f"Error extracting images from {pdf_path}: {e}")
        return []
```

### 6.3.2 TOKENISATION AND CHUNKING

```
def tokenize_and_chunk_document(document, max_chunk_size=1000):
    tokenizer = tiktoken.get_encoding("cl100k_base")
    tokens = tokenizer.encode(document)
    chunks = [tokenizer.decode(tokens[i:i + max_chunk_size]) for i in range(0,
len(tokens), max_chunk_size)]
    return chunks
```

### 6.3.3 SCRIPT GENERATION

```
def generate_podcast_script(model, text_chunks, image_descriptions, episode_number,
total_episodes, previous_summary=""):
    chat = model.start_chat(history=[])
    system_prompt = f"""You are a world-class podcast writer, having ghostwritten for
famous podcasters like Joe Rogan, Lex Fridman, Ben Shapiro, and Tim Ferris.

Your job is to craft an engaging **audio** podcast dialogue for Episode
{episode_number} of a {total_episodes}-episode series based on the provided PDF
content chunks. The podcast follows a structured format with three mandatory sections:
**Introduction, Main Content, and Conclusion.** The conversation flows naturally
between Speaker 1 (Host) and Speaker 2 (Guest), incorporating all textual content from
the chunks and image descriptions seamlessly into a single cohesive script. Ensure
continuity with previous episodes and a clear transition to the next.
```

**Speaker Roles**:

- **Speaker 1 (Host):** Engages the Guest with insightful, engaging, and occasionally humorous questions. They are unfamiliar with the document and ask for clarifications, examples, or follow-ups to ensure clarity.
- **Speaker 2 (Guest):** An expert on the document, providing structured, detailed, and engaging explanations. They use examples, metaphors, and anecdotes to enhance understanding.

---

**Podcast Structure (MANDATORY - STRICTLY ENFORCED)**:

1 Introduction

- Speaker 1 (Host) introduces Episode {episode_number} with an engaging summary of this portion of the document's scope, covering all chunks provided.
- If this isn't Episode 1, briefly recap the previous episode using this summary: "{previous_summary}".
- Clearly state this is part of a {total_episodes}-episode series and set up the topic for this episode.
- Capture the listener's attention with a hook relevant to this episode's content.

2 Main Content

- A lively, continuous conversation unfolds between Speaker 1 and Speaker 2, integrating all provided chunks into a single narrative.
- Speaker 1 asks thoughtful and natural questions, while Speaker 2 provides clear, engaging explanations tied to this episode's content across all chunks.
- **Weave image descriptions naturally into the dialogue** without explicit references (e.g., no "as you can see here"), ensuring they fit the flow of the conversation.
- Ensure the content feels distinct from other episodes while building on the overall narrative.

3 Conclusion

- Speaker 1 summarizes the key points from this episode, reflecting all chunks.

- If this isn’t the final episode, tease the next episode (e.g., "Next time in Episode {episode_number + 1}, we’ll explore...").
- End with a **single, natural closing remark** that reinforces the series’ continuity (e.g., "Stay tuned for more!").
---

Every line MUST start with **"Speaker 1:"** or **"Speaker 2:"**. Combine all provided chunks into one script with a single Introduction, Main Content, and Conclusion for this episode.
"""
    combined_text = "\n\n".join([f"Chunk {i+1}:\n{chunk}" for i, chunk in enumerate(text_chunks)])
    combined_images = "\n\n".join([f"Image {i+1}: {desc}" for i, desc in enumerate(image_descriptions)]) if image_descriptions else "No additional images in this section."
    input_message = f"{system_prompt}\n\nText Content for Episode {episode_number}:\n{combined_text}\n\nImage Insights for Episode {episode_number}:\n{combined_images}"

    try:
        response = chat.send_message(input_message, stream=False)
        return response.text
    except Exception as e:
        print(f"Error generating script for Episode {episode_number}: {e}")
        return (
            f"Speaker 1: Welcome to Episode {episode_number} of our {total_episodes}-episode series. "
            f"{'Last time, ' + previous_summary if previous_summary else 'Today, we’re starting fresh.'}\n"
            f"Speaker 2: Unfortunately, we hit a snag with the content, but let’s talk about what we can.\n"

```
        f"Speaker 1: Fair enough! What's one takeaway we can share?\n"
        f"Speaker 2: Even when tech fails, resilience keeps us going.\n"
        f"Speaker 1: That's Episode {episode_number}! "
        f"{'Next time in Episode ' + str(episode_number + 1) + ', we'll dive deeper.' if
episode_number < total_episodes else 'Thanks for joining us!'} "
        f"Stay tuned for more!"
    )
```

### 6.3.4 SCRIPT REFINEMENT:

```
def refine_script_against_documen
t(model, document, generated_script, episode_number, total_episodes):
    chat = model.start_chat(history=[])
    system_prompt = f"""
    You are an expert podcast script editor refining the script for Episode
{episode_number} of a {total_episodes}-episode series. Ensure:
    - It accurately represents the key content from the provided document chunk,
covering all provided chunks in a single cohesive narrative.
    - Corrects any deviations or inaccuracies compared to the document.
    - Maintains the structure: **Introduction, Main Content, and Conclusion**.
    - Improves clarity, grammar, and flow while keeping it engaging and episode-
specific.
    - Integrates image descriptions naturally without breaking continuity.
    - Ensures the script feels like part of a cohesive series with clear narrative
progression.
    **ONLY RETURN the refined script** without commentary.
    Document content for this episode:
    {document}
    Generated script:
    {generated_script}
    """
    try:
```

```
            response = chat.send_message(system_prompt, stream=False)
            return response.text if hasattr(response, "text") else generated_script
        except Exception as e:
            print(f"Error refining Episode {episode_number}: {e}")
            return generated_script
```

### 6.3.5 AUDIO GENERATION :

```
def generate_episode_audio(script, episode_number,
output_folder="podcast_audio"):
    os.makedirs(output_folder, exist_ok=True)
    if not script or not script.strip():
        print(f"Error: Script for Episode {episode_number} is empty or
whitespace.")
        return None
    script = "\n".join(line.strip() for line in script.splitlines() if line.strip())
    lines = script.split("\n")
    audio_segments = []
    styles = {"Host": ('com.au', False), "Guest": ('co.in', False)}
    for line in lines:
        line = line.strip()
        if not line:
            continue
        if line.startswith("**Speaker 1:**") or line.startswith("Speaker 1:"):
            text = line.replace("**Speaker 1:**", "").replace("Speaker 1:", "").strip()
            if not text:
                continue
            tts = gTTS(text=text, lang='en', tld=styles["Host"][0],
slow=styles["Host"][1])
            mp3_fp = io.BytesIO()
            tts.write_to_fp(mp3_fp)
            mp3_fp.seek(0)
```

```
            audio_segments.append(AudioSegment.from_file(mp3_fp,
        format="mp3"))
        elif line.startswith("**Speaker 2:**") or line.startswith("Speaker 2:"):
            text = line.replace("**Speaker 2:**", "").replace("Speaker 2:", "").strip()
            if not text:
                continue
            tts = gTTS(text=text, lang='en', tld=styles["Guest"][0],
        slow=styles["Guest"][1])
            mp3_fp = io.BytesIO()
            tts.write_to_fp(mp3_fp)
            mp3_fp.seek(0)
            audio_segments.append(AudioSegment.from_file(mp3_fp,
        format="mp3"))

    if audio_segments:
        combined = AudioSegment.empty()
        for segment in audio_segments:
            combined += segment + AudioSegment.silent(duration=500)
        final_file = os.path.join(output_folder,
        f"podcast_episode_{episode_number}.mp3")
        combined.export(final_file, format="mp3")
        return final_file
    else:
        print(f"Warning: No audio segments generated for Episode
        {episode_number}.")
        return None
```

### 6.3.6 TOPIC EXTRACTION:

```
def extract_topics(text, num_words=5, num_topics=3):
    chunks = tokenize_and_chunk_document(text, max_chunk_size=1000)
    all_tokens = [preprocess_text(chunk) for chunk in chunks if chunk.strip()]
```

```
    if not any(all_tokens):
        return ["No significant topics found."]

    dictionary = corpora.Dictionary(all_tokens)
    if len(dictionary) < num_words:
        return ["Insufficient unique terms for topic modeling."]

    corpus = [dictionary.doc2bow(tokens) for tokens in all_tokens if tokens]
    try:
        lda_model = LdaModel(corpus, num_topics=num_topics, id2word=dictionary,
passes=15, iterations=100, random_state=42)
        topics = [" ".join([word for word, _ in topic]) for _, topic in
lda_model.show_topics(num_words=num_words, formatted=False)]
        return topics
    except Exception as e:
        print(f"Error in LDA topic modeling: {e}")
        return ["Topic extraction failed."]
```

### 6.3.7 QUESTIONS GENERATION:

```
def generate_questions_from_text(model, text):
    chat = model.start_chat(history=[])
    topics = extract_topics(text)
    if "No significant topics found." in topics:
        return ["Unable to generate meaningful questions due to insufficient content."]
    prompt = (
        f"Generate 5 to 7 thoughtful and simple questions based on the following topics
extracted from a document:\n\n"
        f"{topics}\n\n"
        f"### Output Format:\n"
        f"1. [First Question]\n"
        f"2. [Second Question]\n"
```

```
    )
    try:
        response = chat.send_message(prompt, stream=False)
        questions_text = response.text if hasattr(response, "text") else "No questions generated."
        questions = [line.strip() for line in questions_text.split("\n") if line.strip() and re.match(r"^[1-7]\.", line.strip())]
        return [q.split(". ", 1)[1].strip("[]") for q in questions][:7]
    except Exception as e:
        print(f"Error generating questions: {e}")
        return ["Unable to generate questions due to an error."
```

# CHAPTER-7

# OUTPUT SCREENS

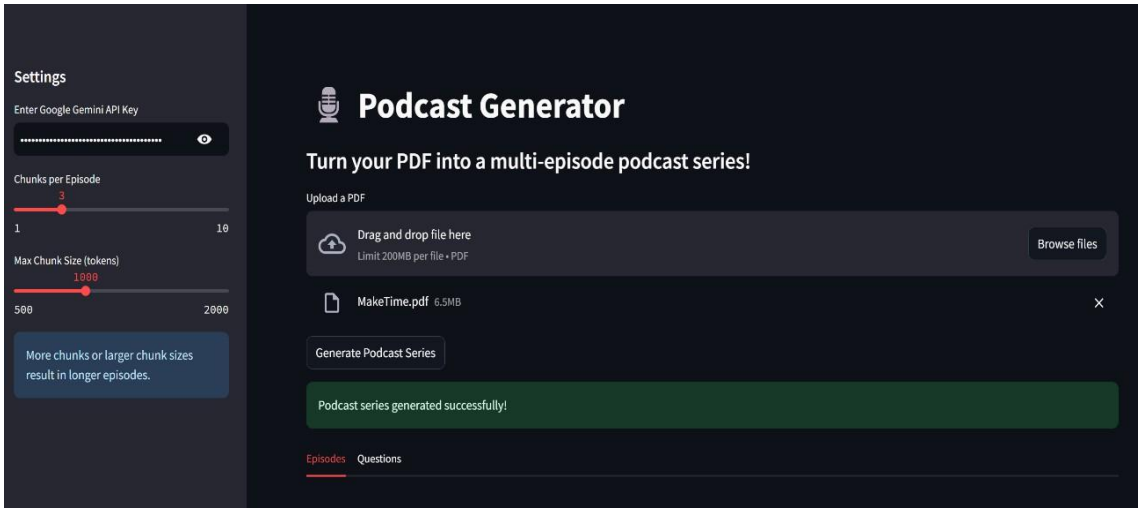


*Figure 6 Showing Podcast Generator Interface*

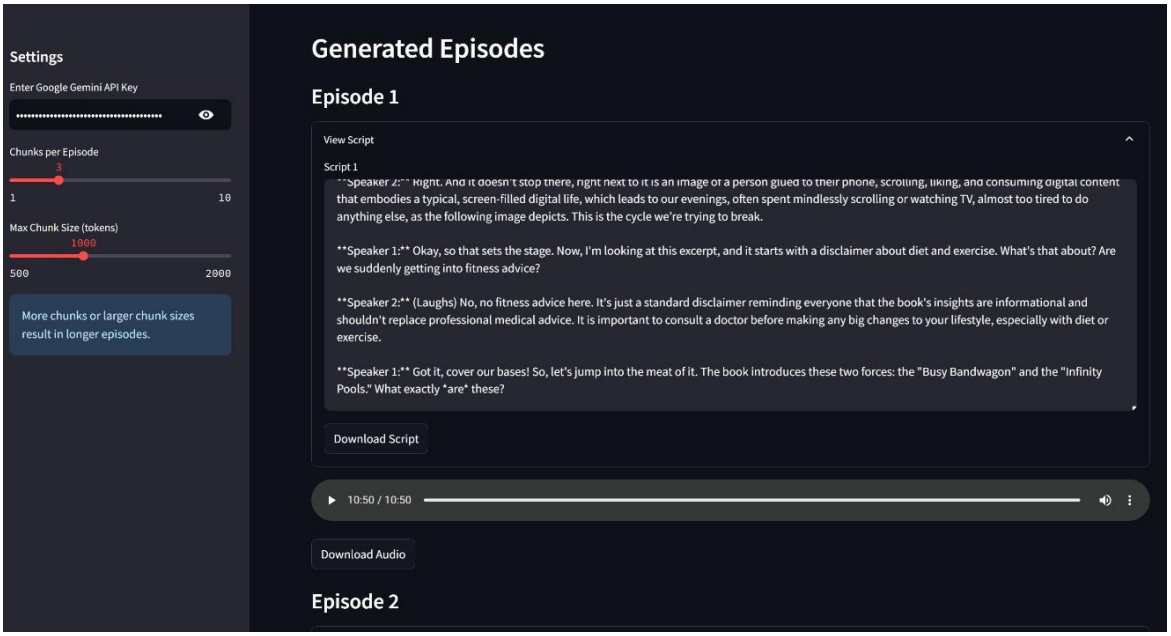


*Figure 7 Showing Podcast Episode Generation*

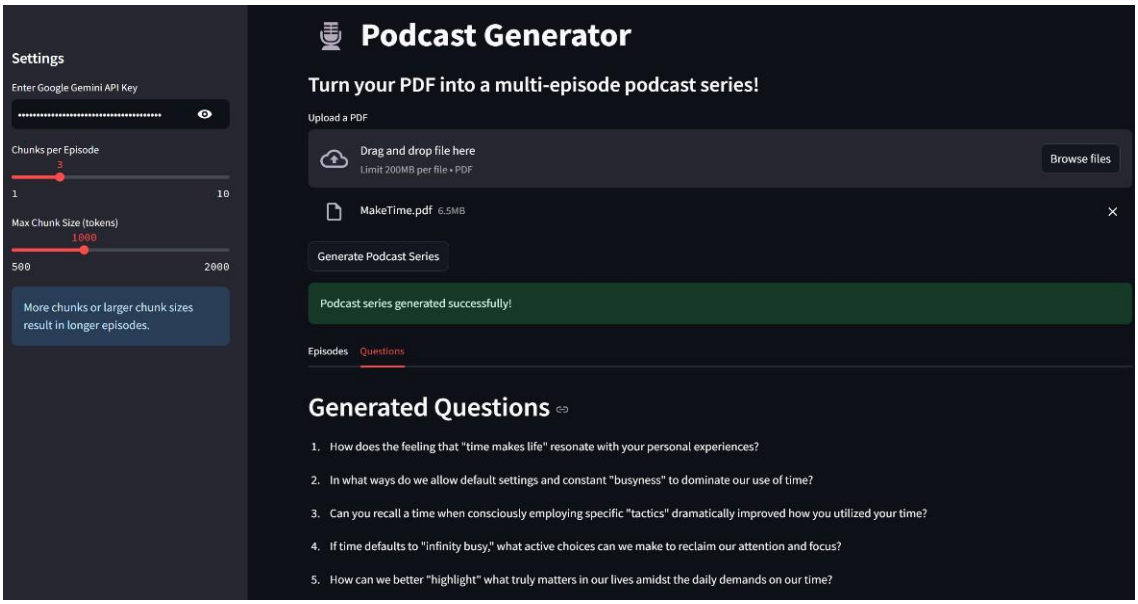


*Figure 8 Showing Questions Generation*

# CHAPTER-8

## CONCLUSION AND FUTURE ENHANCEMENT

## CONCLUSION

This project successfully converts PDF documents into engaging podcasts, enhancing accessibility and comprehension through AI-driven audio narration. By integrating AI-powered text generation and text-to-speech (TTS) synthesis, the system extracts text and images from PDFs, processes them into structured podcast episodes, and converts them into high-quality audio. Additionally, the system leverages topic modeling techniques to generate relevant questions based on extracted content, enabling deeper engagement and interactive discussions. This approach ensures that complex documents are transformed into easily digestible episodes, maintaining a conversational and listener-friendly format that improves understanding and engagement.

## FUTURE ENHANCEMENTS

The podcast generation system has the potential for several enhancements to make it more versatile and engaging for different audiences. One key future improvement is the ability to customize podcast content based on different age groups. This enhancement would allow the system to adapt the tone, style, and content presentation according to the target audience, ensuring a more personalized and engaging experience.

- **Audience-Specific Podcast Customization:** The system will generate content tailored for different age groups. For children, podcasts will be interactive and engaging, incorporating storytelling elements, playful tones, and expressive voices. For students, content will be educational yet engaging, integrating a sense of humor and conversational storytelling while adjusting complexity based on academic level. For professionals and general audiences, the system will ensure a structured, formal, and informative podcast experience

- **Multilingual Support:** Another future enhancement is the integration of **multilingual capabilities**, allowing podcasts to be generated in different languages, expanding accessibility to a global audience.
- **Voice Customization:** Advanced AI-driven voice modulation can be implemented, enabling users to select different voice styles and tones based on the intended audience.
- **Podcast Duration Modification:** A new feature will allow users to specify the desired duration of the podcast. If a user wants to complete the entire input document within a specific timeframe, such as 10 minutes, the system will automatically adjust the script length, speaking speed, and content summarization to meet the required duration.

By implementing these enhancements, the system will provide a more inclusive and adaptable podcast experience, catering to the needs of various listener demographics.

# CHAPTER-9

# REFERENCES


1. Google Generative AI (Gemini) [ https://ai.google/]

2. **Vaswani, A. et al. (2017).** Attention Is All You Need. NeurIPS. -

[https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf]

3. **Hsu, W. et al. (2020).** Text-to-Speech Synthesis with Generative Adversarial Networks. -[https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10431766]

4. ***Radford, A. et al. (2019).*** Language Models Are Few-Shot Learners.

-[https://arxiv.org/abs/2005.14165].

5. Gensim: Topic Modeling with LDA - [ https://radimrehurek.com/gensim/models/ldamodel]

6. Streamlit: Interactive Web Apps for Data Science - [ https://streamlit.io/]4


## GITHUB LINK

https://github.com/BhaskaraSaraswathi/podcast-synthasizer