A PROJECT REPORT ON

GenVox: A VOICE ACTIVATED MULTIMODEL GENERATIVE AI COMPANION

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, KAKINADA

For Partial Fulfilment of Award of the Degree of

BACHELOR OF TECHNOLOGY

Submitted By

J. Mahalakshmi	**21X41A4222**
P. Manikanta	**21X41A4239**
S. V. V Prasanna Kumar	**21X41A4251**
L. Ganesh Kumar	**22X45A4204**

Under the esteemed guidance of

Dr. D. Anusha

Associate Professor, Department of CSE – AI&ML

DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

S.R.K INSTITUTE OF TECHNOLOGY (AFFILIATED TO JNTU, KAKINADA)

Enikepadu, Vijayawada – 521108.

April 2025

S.R.K INSTITUTE OF TECHNOLOGY ENIKEPADU, VIJAYAWADA.
DEPARTMENT OF CSE- ARTIFICIAL INTELLIGENCE & MACHINELEARNING

CERTIFICATE

This is to certify that this project report entitled "GenVox: A VOICE ACTIVATED MULTI-MODEL GENERATIVE AI COMPANION" is the work of J . Mahalakshmi (21X41A4222), P. Manikanta (21X41A4239), S. V. V. Prasanna Kumar (21X41A4251), L. Ganesh Kumar (22X45A4204) in partial fulfillment of the requirements for the award of the graduate degree in BACHELOR OF TECHNOLOGY during the academic year 2024-2025. This Work has carried out under our supervision and guidance.

(Dr. D. Anusha)
Signature of the Guide

(Dr. D. Anusha)
Signature of the HOD

Signature of the External Examiner

DECLARATION

We J. Mahalakshmi, P. Manikanta, S. V. V. Prasanna Kumar, L. Ganesh Kumar hereby declare that the project report entitled "GenVox: A VOICE ACTIVATED MULTI-MODEL GENERATIVE AI COMPANION" is an original work done in the Department of CSE – AI & ML, SRK Institute of Technology, Enikepadu, Vijayawada, during the academic year 2024-2025, in partial fulfillment for the award of the Degree of Bachelor of Technology in CSE– AI&ML. We assure you that this project is not submitted to any other College or University.

Roll Number	Name of the Student	Signature
21X41A4222	J. Mahalakshmi	
21X41A4239	P. Manikanta	
21X41A4251	S. V. V. Prasanna Kumar	
22X45A4204	L. Ganesh Kumar	

ACKNOWLEDGEMENT

Firstly, we would like to convey our heart full thanks to the Almighty for the blessings on us to carry out this project work without any disruption.

We are extremely thankful to Dr. D. Anusha, our guide throughout the project. We also thank her for most independence and freedom throughout the given to us during various phases of the project.

We are also thankful for our project coordinator Dr. D. Anusha, for their valuable guidance which helped us to bring this project successfully.

We are very much grateful to Dr. D. Anusha. H.O.D of CSE-AI&ML Department, for his valuable guidance which helped us to bring out this project successfully. His wise approach made us learn the minute details of the subject. His mature and patient guidance paved away for completing our project with a sense of satisfaction and pleasure.

We are greatly thankful to our principal Dr. M. Ekambaram Naidu for his kind support and facilities provided at our campus which helped us to bring out this project successfully.

Finally, we would like to convey our heart full thanks to our Technical Staff, for their guidance and support in every step of this project. We convey our sincere thanks to all the faculty and friends who directly or indirectly helped us with the successful completion of this project.

PROJECT ASSOCIATES

J. Mahalakshmi (21X41A4222)

P. Manikanta (21X41A4239)

S. V. V. Prasanna Kumar (21X41A4251)

L. Ganesh Kumar (22X45A4204)

LIST OF CONTENTS

ABSTRACT

GenVox a voice-activated multimodal generative AI companion could combine the latest advancements in natural language processing, image generation, and audio synthesis to deliver a highly interactive and versatile experience and it is a virtual AI companion that interacts seamlessly with users through voice commands and produces outputs in multiple formats, such as text, images and audio. It could serve as a personal assistant, creative collaborator, or learning partner, leveraging generative AI to adapt to the user's needs.

The project includes core features like voice-controlled text generation like storytelling, content creation, Q&A, and summarization via voice commands. Image creation via voice prompts like generate art, illustrations, or designs based on spoken descriptions. Audio Synthesis content.

Interactive Cross-Modal Content Creation Combine models based on vocal input, such as describing a text, image and etc. By using the technologies like large language models (LLM's) for the content creation and gTTs enterprise for the text to speech conversion, pyttsx3 for the natural language processing, python 3.0 for the project development and python Kivy for android development, API keys of Together ai, hugging face for using of prebuilt generative models. This project would let users activate and interact with various generative AI models through natural voice commands, making content creation intuitive and accessible.

Keywords: LLM's, gTTs, Llama-3.0-turbo-free, Python3.0, Python Kivy, Together ai, hugging face API keys

CHAPTER 1

INTRODUCTION

1.1 Overview

The rapid advancement in artificial intelligence (AI) and natural language processing (NLP) has revolutionized human-computer interaction. Voice-activated AI systems, such as virtual assistants, have become ubiquitous, but most are limited to single-modal outputs like text or voice. GenVox emerges as a next-generation, multimodal AI companion that integrates voice commands with generative capabilities for text, images, audio, and video. This project leverages cutting-edge technologies like large language models (LLMs), speech recognition, and diffusion models to create a versatile, interactive AI system. Unlike conventional assistants (e.g., Alexa or Siri), GenVox focuses on creativity, personalization, and cross-modal content generation, making it a powerful tool for education, entertainment, and productivity.

1.2 About the Project

GenVox is a voice-activated AI companion designed to bridge the gap between human creativity and machine-generated content. It enables users to generate text (stories, code, summaries), images (art, designs), audio (music, speech), and animations through intuitive voice commands. Built using Python, Kivy (for Android), the system integrates APIs like OpenAI, Gemini, and Hugging Face to deliver seamless multimodal outputs. The project's modular architecture ensures scalability, allowing future integration of additional AI models or features. GenVox targets diverse users, including students, artists, developers, and professionals, by offering tailored responses and learning from user preferences.

1.3 Purpose

The primary purpose of GenVox is to democratize access to generative AI by eliminating technical barriers. Traditional AI tools require textual inputs or complex interfaces, but GenVox simplifies interaction through natural voice commands. Key objectives include:

- **Enhancing accessibility**: Voice-first design caters to users with limited literacy or technical skills.
- **Fostering creativity**: Cross-modal generation (e.g., converting spoken descriptions to images or music) empowers artists and educators.

- **Improving productivity**: Automating tasks like code generation, summarization, and translation saves time.
- **Ensuring privacy**: On-device processing for sensitive data and compliance with GDPR/CCPA standards.

1.4 Scope

GenVox supports the following functionalities:

- **Voice-to-Text**: Convert spoken queries into actionable commands (e.g., "Write a poem about nature").
- **Multimodal Outputs**: Generate text, images (via Stable Diffusion), audio (via gTTS), and animations.
- **Personalization**: Adapt responses based on user history (e.g., preferred art styles or coding languages).
- **Integration**: Compatible with calendars, messaging apps, and smart home devices.
- **Platforms**: Android app (Kivy), web interface (Gradio), and cloud backend (Together AI).

CHAPTER 2

LITERATURE SURVEY

1. **Voice Control Using AI-Based Voice Assistant**
    - **Authors:** S. Subhash, R. Kumar, P. Sharma, M. Verma
    - **Year:** 2020
    - **Abstract:** This study explores the implementation of AI-based voice assistants for automating tasks through voice commands. The research employed Google Text-to-Speech (GTTS) for speech synthesis and Python for building a personal voice assistant. The study demonstrated the potential of AI-driven voice assistants in simplifying daily tasks through hands-free automation, improving accessibility, and making human-computer interaction more seamless. AI-powered voice commands for task automation. Demonstrated the efficiency of AI voice commands for simplifying everyday tasks.
2. **Voice-Based Virtual Assistant**
    - **Authors:** Dilshad Ahmad, Kiran H, Girish Kumar, Hanumanta DH
    - **Year:** 2023
    - **Abstract:** This study developed a smart, hands-free, voice-activated assistant using machine learning and speech recognition. It employed Speech Recognition for speech-to-text conversion and pyttsx3 for text-to-speech synthesis. The research demonstrated how AI-powered virtual assistants enhance interaction, automation, and real-time information retrieval. The study also explored the future potential of voice assistants in IoT integration, particularly in smart home automation. Developing a hands-free, smart AI assistant for real-time task automation. Created a multilingual, context-aware AI assistant with task automation

3. **Research on the Development of Voice Assistants in the Era of Artificial Intelligence**
    - **Authors:** Yuqi Huang, L. Zhang, F. Wei, T. Chen
    - **Year:** 2023
    - **Abstract:** This research examined the increasing role of AI voice assistants in consumer electronics and human-computer interaction. A comparative analysis was conducted on AI voice assistants in different regions, highlighting how cultural and technological factors influence their development and adoption. The study emphasized the need for improved usability and adaptability in voice assistants to cater to diverse user preferences and needs.
4. **Intelligent Personal Assistants: The Future of AI-Powered Voice Assistants**
    - **Authors:** J. Williams, R. Patel, M. Anderson, K. Lee
    - **Year:** 2021
    - **Abstract:** This study analyzed the evolution of voice assistants like Siri, Alexa, and Google Assistant. The research demonstrated how deep learning models have significantly improved voice recognition and natural language processing capabilities. The study provided insights into how AI advancements enhance the accuracy and contextual awareness of voice assistants, paving the way for more personalized and intuitive interactions.
5. **Voice Assistant Applications in Healthcare**
    - **Authors:** R. Patel, M. Gupta, S. Khan, P. Roy
    - **Year:** 2022
    - **Abstract:** This study investigated AI voice assistants in telemedicine, patient monitoring, and medical record management. AI-powered speech-to-text features were found to enhance the efficiency of medical documentation and improve accessibility in healthcare. The research demonstrated how voice assistants can assist healthcare professionals by automating routine tasks and providing voice-enabled support for patients with disabilities.

6. **Voice-Based Home Automation Using AI**
    - **Authors:** M. Sharma, R. Mehta, A. Joshi, S. Desai
    - **Year:** 2023
    - **Abstract:** This research studied the integration of voice assistants with IoT devices for smart home automation. The study utilized cloud-based AI models for speech processing and command execution, demonstrating how voice-controlled smart homes enhance convenience and accessibility. The findings highlighted the potential of AI in creating fully automated, user-friendly smart environments. Using AI to control IoT-based smart home systems via voice. Integrated voice-controlled IoT devices, enhancing home automation efficiency.
7. **Enhancing Voice Assistants with Generative AI Language Models for Interactive Conversations**
    - **Authors:** Shivam Hajong
    - **Year:** 2024
    - **Abstract:** This study developed a voice assistant using HTML, CSS, JavaScript, Python, SQLite, and the Hugging Face API. The research proposed a comprehensive methodology covering data collection, model training, frontend and backend development, integration, testing, and deployment. The study highlighted how generative AI and large language models (LLMs) enhance voice assistants, making interactions more intelligent and dynamic. Using LLMs to create an interactive and intelligent voice assistant system. Developed a voice assistant application integrating HTML/CSS, JavaScript, Python, SQLite, and Hugging Face API.
8. **Smart AI Voice Assistant through Generative Text Transformer and NLP Implementation in Python**
    - **Authors:** Devesh Bajpai, Mallela Uday Kiran, Busi Hemanth Reddy, Suresh Kumar Natarajan
    - **Year:** 2024

- **Abstract:** This research employed Natural Language Processing (NLP), AI, Generative Text Transformers (GTT), and Speech Recognition. The study demonstrated the synergistic potential of these technologies in creating highly intelligent voice assistants. The research provided insights into how AI-driven assistants can transform user interactions, making them more intuitive and personalized. Improving AI decision-making and user interaction using AI, NLP, GTT, and speech recognition. Implemented Generative Text Transformer (GTT) and NLP for advanced language processing in AI assistants.

9. **Artificial Intelligence-Based Chatbot with Voice Assistance**

    - **Authors:** A. Balamurugan, D. Thiruppathi, S. P. Santhoshkumar, K. Susithra
    - **Year:** 2024
    - **Abstract:** This study developed an AI-based chatbot integrating voice assistance using GPT-3.5. The application converts voice input into text, processes it using a generative AI model, and returns an audio response. The study showcased the growing significance of AI-powered chatbots in providing interactive and human-like assistance across various domains. Merging AI-based chatbots with voice assistance to improve user engagement. Developed a web-based AI chatbot integrating GPT-3.5 and voice assistance for improved interaction.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Existing System

Voice-Based Virtual Assistant (VBVA) project aimed at creating a smart, hands-free, voice- activated assistant leveraging cutting-edge machine learning and speech recognition technologies. The assistant provides personalized responses, task automation (e.g., setting reminders, making calls, sending messages), and retrieves real-time information (e.g., weather, news).

It includes multilingual and context-aware functionalities to enhance usability and adaptability. Core technologies include Python libraries like Speech Recognition for speech-to-text and pyttsx3 for text-to-speech. Future directions involve integrating VBVA into compact hardware like Raspberry Pi and expanding smart home automation capabilities. This innovation targets improved accessibility, efficiency, and intuitive user interaction.

Conventional AI assistants such as Alexa and Google Assistant offer text and speech-based responses but lack multimodal content generation. These systems rely on predefined scripts and do not adapt dynamically to user inputs. Additionally, they have limited personalization capabilities, reducing user engagement.

3.1.1 Disadvantages of Existing System

1. Restricted Output Modalities: Lack of support for generating images, videos, or multimodal content.
2. Low Interactivity: Limited personalization and static responses.
3. Application Constraints: Tailored for specific use cases, with minimal adaptability for creative tasks.
4. Limited Generative Capabilities: Existing assistants cannot generate complex responses beyond basic speech output.
5. Dependency on Predefined Commands: Responses are often scripted and do not evolve based on user interactions.
6. Lack of Multimodal Interaction: No integration of text, image, and audio generation in a unified system.

3.2 Proposed System

GenVox, is a **Voice-Activated Multi-Modal Generative AI Companion** that integrates advanced AI models to generate multimodal content, including **text, speech, and images**. It aims to overcome the limitations of traditional voice assistants by leveraging **large language models (LLMs)**, **text-to-speech (TTS) synthesis**, and **AI-powered image generation**.

This proposed system **bridges the gap between traditional AI assistants and modern generative AI capabilities**, making human-computer interaction **more dynamic, creative, and engaging**.

3.2.1 Proposed System Methodology

1. Voice Recognition & NLP Processing

    Captures user input via speech-to-text (gTTS).

    Converts voice commands into structured text queries.

    Processes queries using LLMs like LLaMA-3.

2. Multimodal AI Interaction

    Processes text input for different AI models.

    Image generation via Stable Diffusion.

    Code generation via LLaMA-3.

    Audio generation via gTTS and Pyttsx3.

3. User Response and Output Handling

    Displays generated results in the mobile app.

    Converts responses into voice output using TTS.

    Stores user preferences and interaction history in MongoDB.

4. Deployment & UI Integration

    Frontend: Kivy-based Android app for a seamless voice and chatbot interface.

    Backend: API integration for AI model processing.

    Database: MongoDB for storing user preferences.

Cloud Hosting: Together AI API for AI model requests.

Security: User authentication and data management.

3.2.2 Advantages of Proposed System

GenVox aims to overcome these challenges with the following innovations from the general Existing system like Alexa, google assistant:

1. Voice-Activated Interaction: Real-time, natural conversation using advanced speech recognition.
2. Multimodal Generative AI: Outputs in text, audio, image based on context and user needs.
3. Personalization: Learns from user preferences to deliver customized experiences.
4. Scalable Framework: Modular architecture allows integration with various third-party APIs and services.

3.2.3 Algorithm

A) Text-to-Speech Processing (Google gTTs)

- Input: User's voice.
- Preprocessing: Noise filtering & speech segmentation.
- Model Processing: The gTTS (Google Text-to-Speech) library is a Python-based tool that converts text into natural-sounding speech. It utilizes Google's TTS API, supporting multiple languages and accents to enhance accessibility and communication. The model processes input text, synthesizes it into speech, and saves it as an audio file or plays it directly. It uses deep learning-based speech synthesis, ensuring clarity and proper pronunciation. With its lightweight and easy-to-use nature, gTTS is widely used in AI assistants, voice-enabled applications, and accessibility tools.
- Output: reading the user input as loud

B) Text & Code Generation (LLaMA-3)

- Input: User's text prompt (converted from speech if needed).
- Processing: The **Meta-Llama 3.3-70B Instruct-Turbo** model generates code using an **autoregressive transformer** approach, predicting the next token based on prior context. It has been **fine-tuned on diverse programming languages** to understand

syntax, structure, and best practices. The model leverages **self-attention mechanisms** to maintain logical consistency across complex code blocks. Using **context-aware token prediction**, it ensures accurate function definitions, variable usage, and formatting. Finally, its **probability-based decoding** helps generate efficient, readable, and optimized code for various programming tasks.

- Output: AI-generated text or code snippet.

C) Image Generation (Stable Diffusion)

- Input: User prompt describing an image.
- Processing: The **Stable Diffusion XL Base 1.0** model generates images using a process called **latent diffusion**. Instead of working directly in pixel space, it compresses images into a lower-dimensional latent space for more efficient processing. A **CLIP-based text encoder** converts input prompts into numerical embeddings, which guide the image generation. The model starts with random noise and progressively refines the image through a **U-Net-based denoising process**, following a **noise scheduling** technique. Finally, a **decoder reconstructs the image** from the latent space, often with optional upscaling for higher-quality outputs.
- Output: AI-generated image.

3.2.4 System Architecture

Architecture Components:

- **Frontend:** Kivy-based Mobile App (Android UI for Voice & Chatbot interface).
- **Backend:** Kivy for AI model integration.
- **Database:** MongoDB for user data storage.
- **AI Model Processing:** Together for LLaMA-3, and Hugging Face models.
- **Voice & NLP:** gTTS for Speech-to-Text and pysxtt3 for speech recognition.

Workflow:

- 1. User interacts via voice input.
- 2. The app processes the command and sends it to the backend.
- 3. Backend calls the appropriate AI model (text, image, code, or audio generation).
- 4. The AI-generated response is sent back to the app.
- 5. The result is displayed in the app (or converted to speech for voice response).

Architecture:

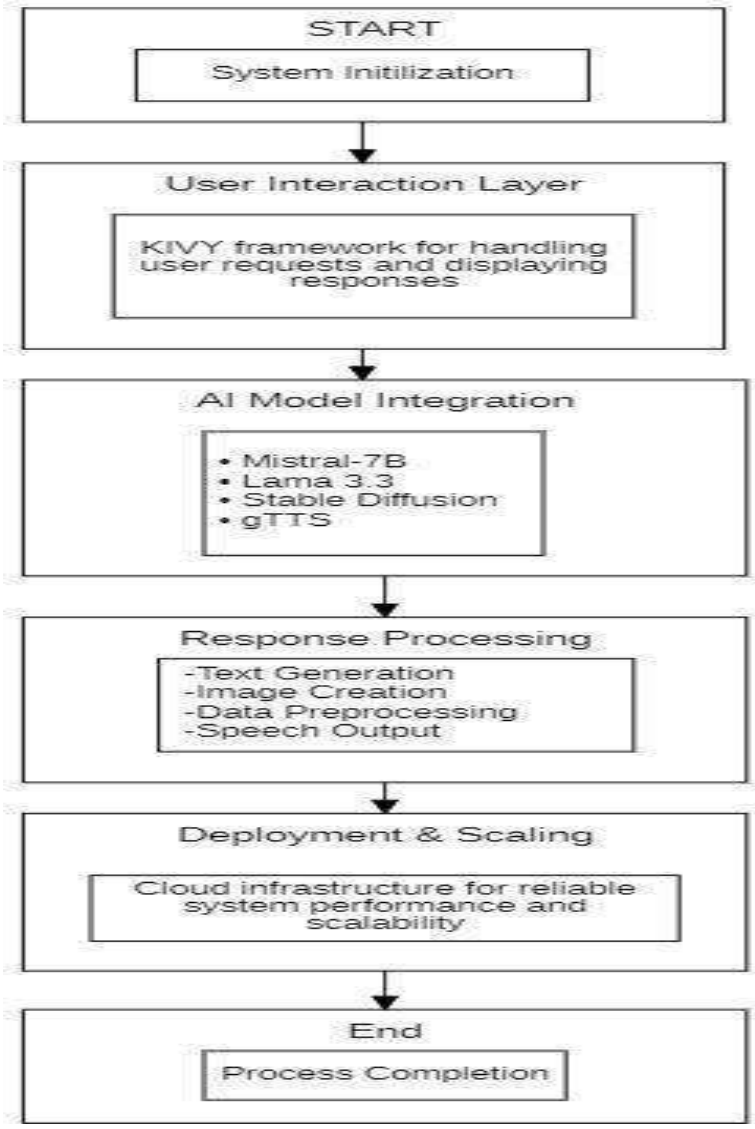


Fig:[3.2.4.1] System Architecture

3.2.5 Modules

1. User Module
    - Purpose: Handles user interactions and personalization.
    - Features:
        - Speech-to-text conversion for voice commands.
        - Multilingual support for diverse users.

- Personalization based on user preferences and interaction history.
- Task automation like setting reminders, sending messages, and retrieving information.

2. Administration Module
    - Purpose: Manages the system's backend and monitoring.
    - Features:
        - Monitoring user details and system performance.
        - Configuring AI model parameters and updates.
        - Implementing and auditing privacy and security measures.
3. Database Module
    - Purpose: Stores and retrieves user data, configurations, and interaction logs.
    - Features:
        - Secure storage of user profiles, preferences, and interaction history.
        - Logs of voice commands and system responses.
        - APIs to access and manage stored data for personalization and analytics.
        - Compliance with data protection regulations (e.g., GDPR, CCPA)
4. System Module

    1. Task Management Module
        - Purpose: Executes user-requested tasks.
        - Features:
            - Integration with calendar, messaging, and smart home systems.
            - Real-time retrieval of information (e.g., weather, news).
            - Handling multiple tasks simultaneously with a queue system.

    2. Generative AI Module
        - Purpose: Processes user queries and generates intelligent responses.

- Features:
  - Natural Language Processing for intent recognition.
  - AI-based content generation using GPT or similar models.
  - Context awareness to maintain conversation flow.

3. Security and Compliance Module

- Purpose: Ensures the system's privacy, security, and compliance.
- Features:
  - Data encryption and secure API communication.
  - User consent management and data deletion options.

CHAPTER 4

SYSTEM SPECIFICATIONS

4.1 Hardware Requirements

- Processor: Intel i5 or above (for development)
- RAM: Minimum 8 GB
- Microphone and speaker setup for voice input/output
- Optional: GPU-enabled device for model training/testing

4.2 Software Requirements

- Development Tools:
  - Python3.0
  - Python Kivy for mobile app development
  - Google Colab
- APIs and Libraries:
  - Google gTTs speech recognition
  - Stability Ai for text and image generation
  - Text-to-Speech (TTS) libraries
  - Hugging face LLM API's
  - Together Ai API's

4.3 Network Requirements

- Internet connection for API usage and cloud interactions

CHAPTER 5

SYSTEM DESIGN

5.1 UML Diagram

Unified Modeling Language (UML) is a standardized visual modeling language used to design, specify, visualize, and document software systems. It plays a crucial role in object-oriented software engineering, enabling developers to model the structure and behavior of a system before implementation. UML is managed by the Object Management Group (OMG) and consists of various diagram types to represent different aspects of the system.

Goals

The primary objectives of UML are:

Standardized Visual Modeling – Provides an expressive and structured way to develop and share system designs. Extensibility and Scalability – Allows easy integration of additional AI models in the future. Independence from Specific Programming Languages – Enables flexibility in using different technologies.

Formal Understanding of the System – Ensures clear documentation of how various components interact.

Encouraging Growth in AI-powered Assistants – Contributes to advancements in multimodal AI assistants.

Incorporation of Best Practices – Integrates industry best practices in AI, NLP, and voice-based interaction.

These UML diagrams provide a **clear representation of system architecture** and help in designing an efficient, scalable AI-based voice assistant.

UML diagrams help to **illustrate the system's architecture, interaction flow, and module behavior**. The following UML diagrams are used in this project:

5.1.1 Use Case Diagram

A Use-Case Diagram represents the interactions between the user and the system. It defines actors (users or external systems) and their interactions with various system functionalities.

This diagram helps **visualize user interactions** with the system and ensures that all functionalities are well-defined.

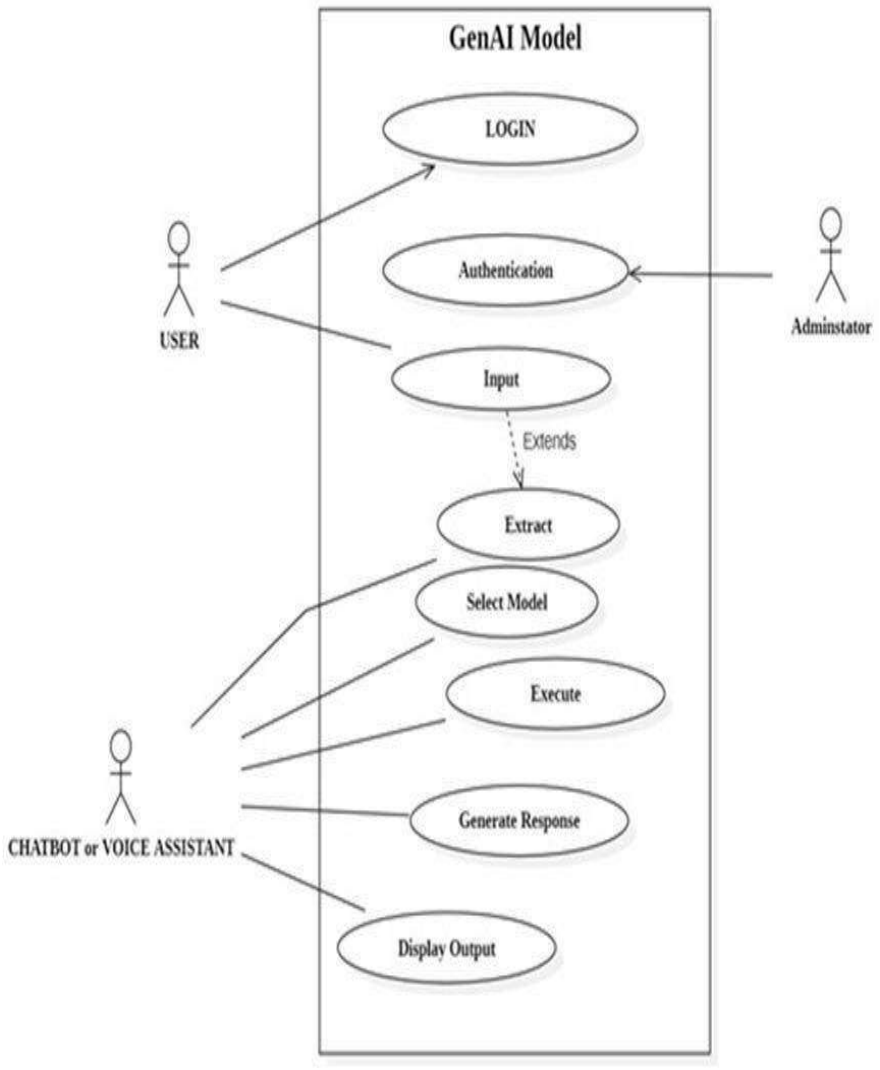


Fig: [5.1.1] Use case Diagram

5.1.2 Class Diagram

A Class Diagram depicts the static structure of the system by showing classes, attributes, methods, and relationships.

Defines the **relationships between different system components** and serves as a **blueprint for implementation**.

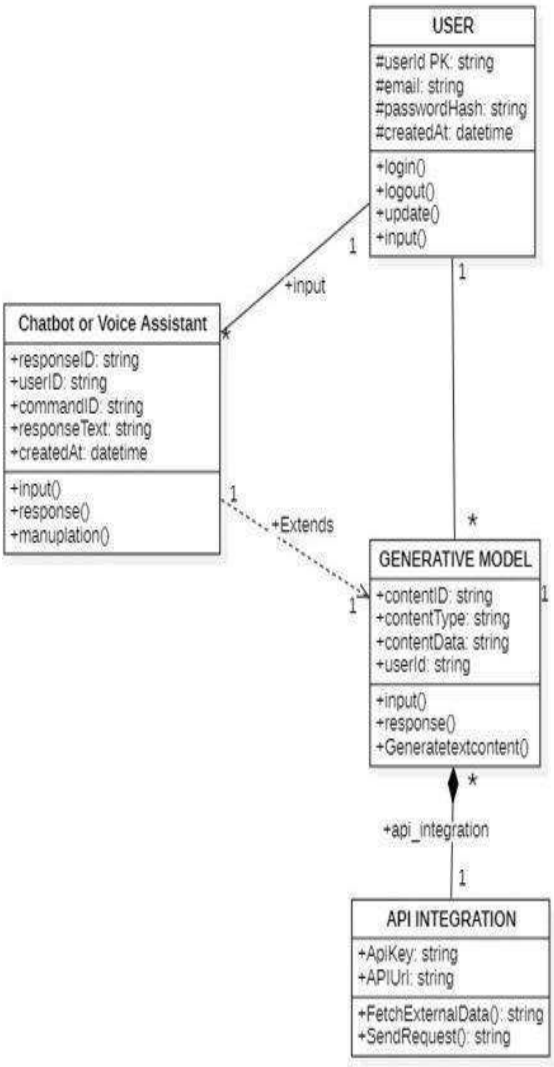


Fig: [5.1.2] Class Diagram

5.1.3 Sequence Diagram

Outlines the step-by-step process of voice input to AI response generation.

A **Sequence Diagram** illustrates the **order of interactions between components** in a system. It shows how messages are passed over time.

Helps in understanding the **exact execution flow and message passing** between system components.

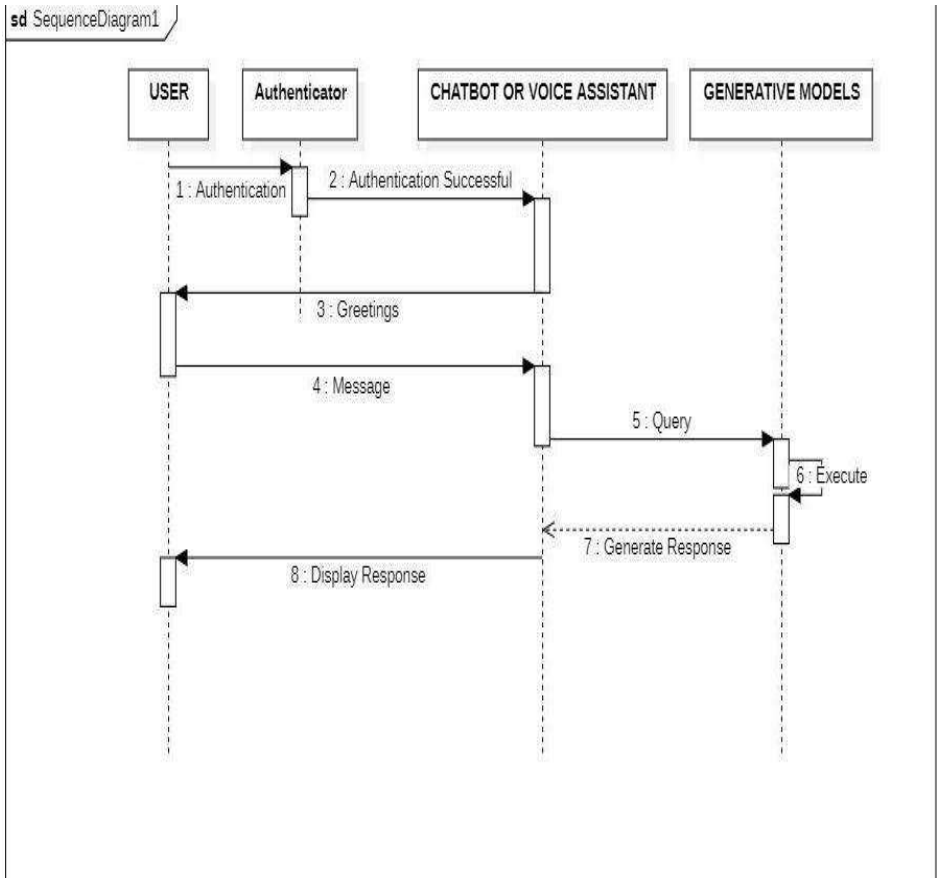


Fig: [5.1.3] Sequence Diagram

5.1.4 Communication Diagram

A **Communication Diagram** (Collaboration Diagram) shows the **interactions between objects**, focusing on **relationships and message flow**. Helps **optimize system communication and interaction** between various modules.

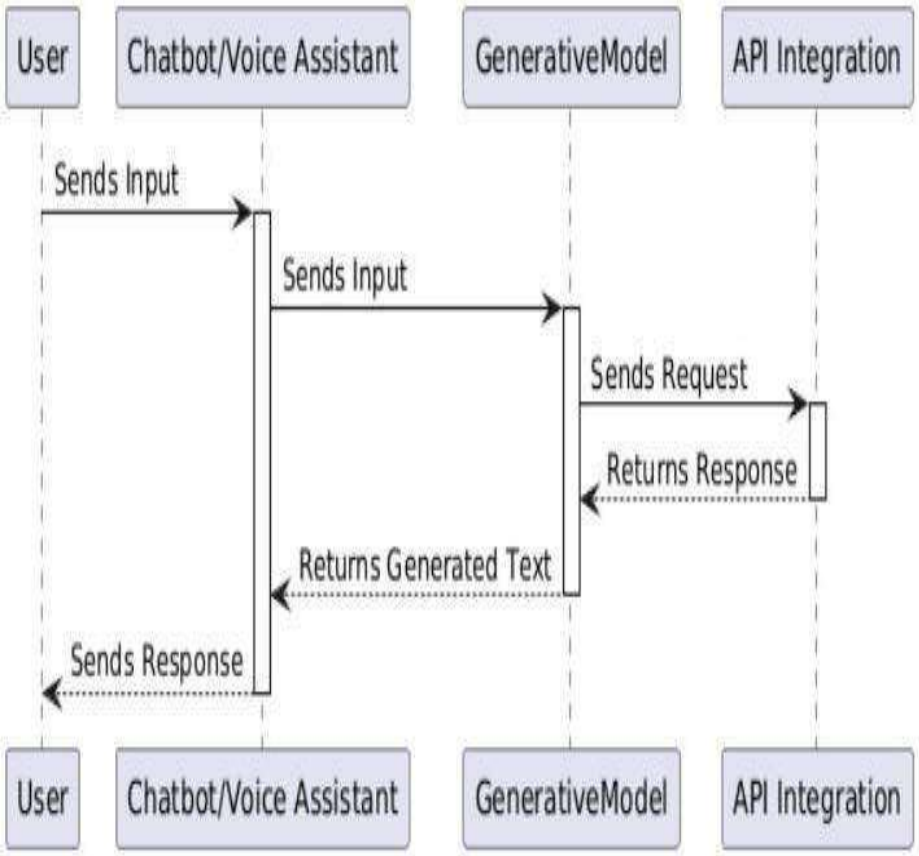


Fig: [5.1.4] Communication Diagram

5.1.5 Activity Diagram

An Activity Diagram represents the workflow of the system, showing the sequence of activities and decision-making points.

Helps in identifying the **flow of actions and decision points** within the system.

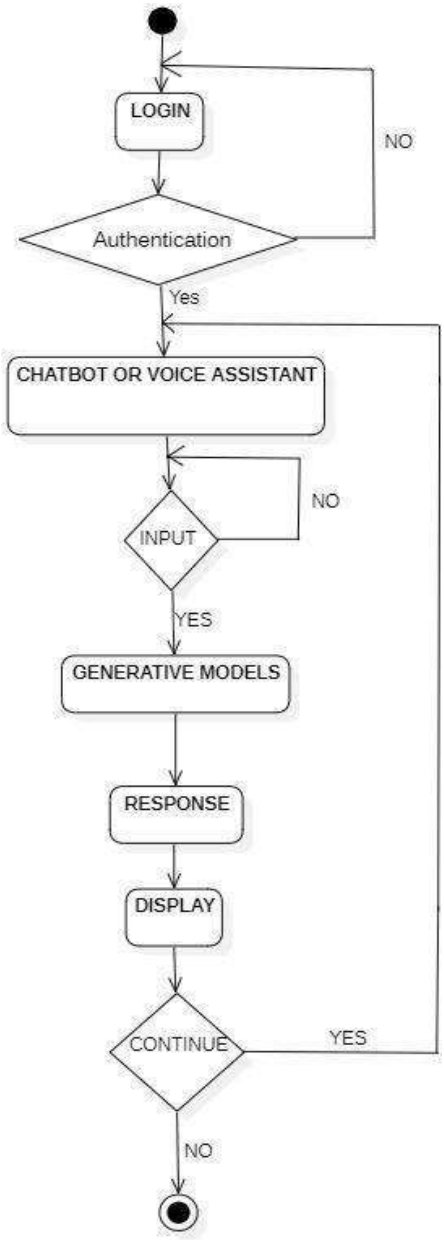


Fig: [5.1.5] Activity Diagram

5.1.6 Component Diagram

A Component Diagram represents the physical components of the system and their dependencies.

Defines the **modular structure** of the system and ensures efficient interaction between components.

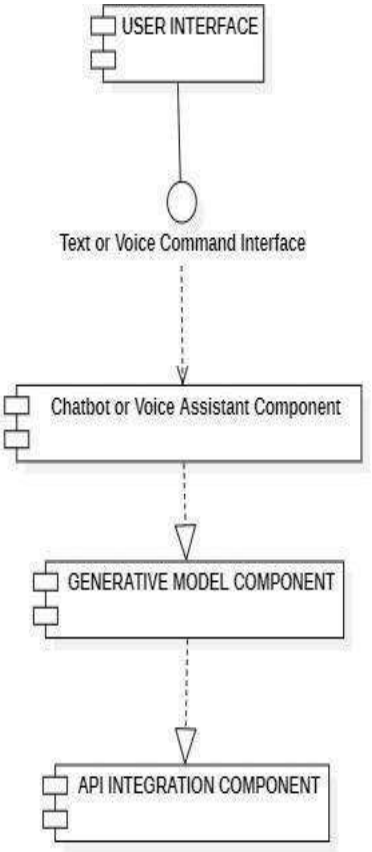


Fig: [5.1.6] Component Diagram

5.1.7 State Machine Diagram

A State Machine Diagram represents the various states the system can be in and how it transitions between them.

Helps in understanding how the system **transitions between different states** based on user interactions.

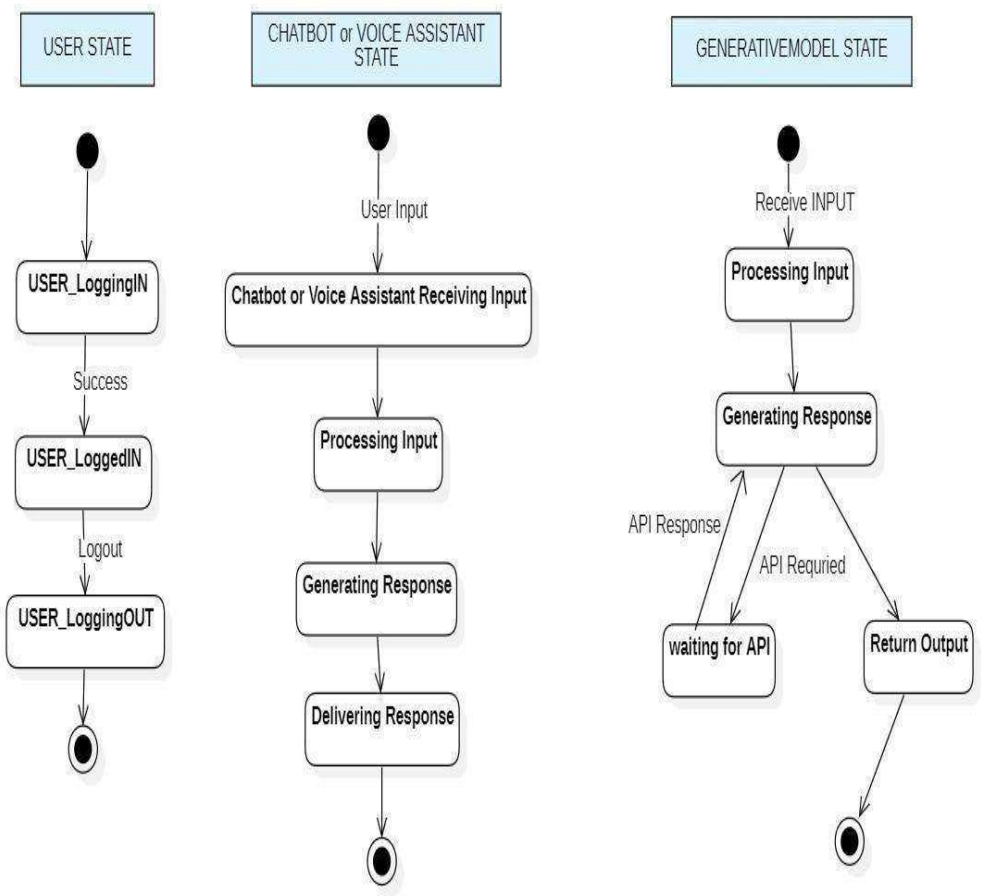


Fig: [5.1.7] State Machine Diagram

5.1.8 Deployment Diagram

A Deployment Diagram illustrates how the software components are distributed across hardware components in the system.

Helps in understanding the **hardware and software distribution** to ensure **scalability and performance optimization**.

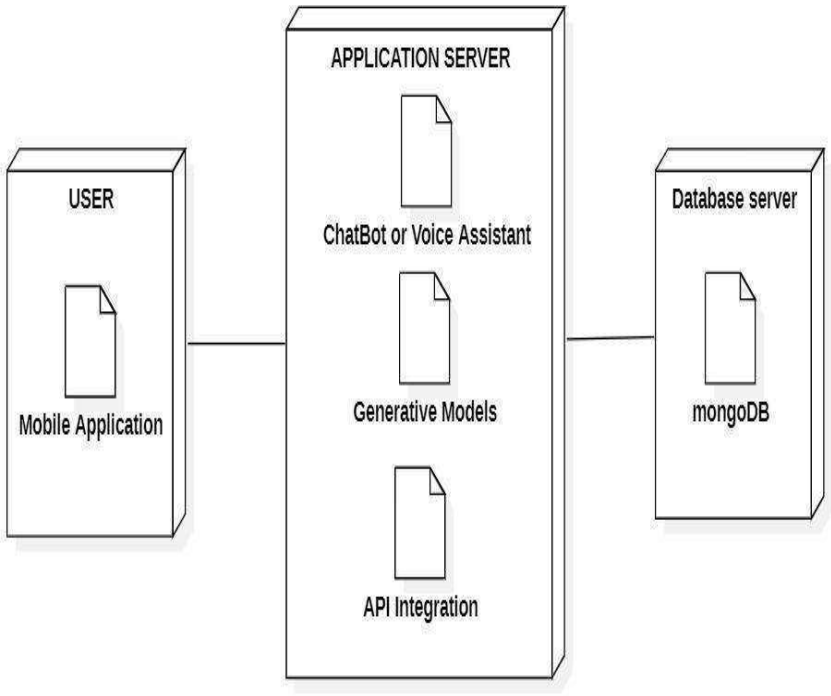


Fig: [5.1.8] Deployment Diagram

Chapter 6

SYSTEM IMPLEMENTATON

6.1 Libraries

1. gTTS – Google Text-to-Speech for converting text to speech.
2. pyttsx3 – Text-to-speech conversion for natural language processing.
3. SpeechRecognition – Converts speech to text.
4. Kivy – Used for Android development.
5. Together AI API – For AI model requests.
6. Hugging Face API – For using prebuilt generative models.
7. Stable Diffusion XL Base 1.0 – For image generation.
8. Meta-LLaMA 3.3-70B Instruct-Turbo – For text and code generation.
9. MongoDB – Database storage.
10. kivy.app (App) – To create and manage the Kivy application.
11. kivy.uix.floatlayout (FloatLayout) – A layout to position widgets using relative coordinates.
12. kivy.uix.boxlayout (BoxLayout) – A layout that arranges widgets in a vertical or horizontal box.
13. kivy.uix.gridlayout (GridLayout) – A layout to arrange widgets in a grid.
14. kivy.uix.label (Label) – For displaying text.
15. kivy.uix.textinput (TextInput) – For user input fields.
16. kivy.uix.button (Button) – For clickable buttons.
17. kivy.uix.screenmanager (ScreenManager, Screen) – To manage multiple screens in the app.
18. kivy.graphics (Color, Rectangle, Ellipse, RoundedRectangle, Line) – For drawing and UI customization.
19. kivy.core.window (Window) – To access and modify the application window

properties.

20. kivy.animation (Animation) – To create animations for UI elements.

21. kivy.clock (Clock) – For scheduling events and updating UI elements over time.

22. kivy.properties (NumericProperty, BooleanProperty, ObjectProperty) – For defining properties in Kivy widgets.

23. random – For generating random numbers.

24. math – Provides mathematical functions.

25. database (authenticate_user, get_user, update_user) – A user-defined module for handling authentication and user data

6.2 Sample code

Main.py:

```
import kivy
from kivy.app import App
from kivy.uix.screenmanager import Screen, ScreenManager
from index_page import IndexScreen
from signin_page import SignInScreen
from login_page import LoginScreen
from home import MainScreen
from chatbot import ChatBotUI
from voice_interface import VoiceAssistantUI
from profile_page import ProfileScreen
from menu_page import MenuScreen
from imagegenereation import ImageGeneratorScreen
from textgeneration import TextGeneratorScreen
from codegeneration import CodeGeneratorScreen
from poetrygeneration import PoetryGeneratorScreen
from problemsolver import ProblemSolverScreen
from Storyteller import StoryGeneratorScreen
from translator import TranslatorScreen
from text_to_speech import SpeechScreen
from textsummarization import SummarizationScreen
from about_page import HomePage
from admin_page import AdminLoginScreen
from admindashboard import AdminDashboardScreen

# Define the minimum Kivy version required
kivy.require('2.0.0')

# Wrapping ChatBotUI & VoiceAssistantUI inside Screen classes
class ChatBotScreen(Screen):
    def _init_(self, **kwargs):
        super(ChatBotScreen, self)._init_(**kwargs)
        self.add_widget(ChatBotUI())

class VoiceAssistantScreen(Screen):
    def _init_(self, **kwargs):
        super(VoiceAssistantScreen, self)._init_(**kwargs)
        self.add_widget(VoiceAssistantUI())

class MainApp(App):
    user_email = None  # Store logged-in user email
    def build(self):
```

```
        sm = ScreenManager()

        # Adding all screens to the screen manager
        sm.add_widget(IndexScreen(name="index"))
        sm.add_widget(SignInScreen(name="signin"))
        sm.add_widget(LoginScreen(name="login"))
        sm.add_widget(MainScreen(name="home"))
        sm.add_widget(ChatBotScreen(name="chatbot"))

        sm.add_widget(VoiceAssistantScreen(name="voice"))
        sm.add_widget(ProfileScreen(name="profile"))
        sm.add_widget(MenuScreen(name="menu"))
        sm.add_widget(ImageGeneratorScreen(name="image"))
        sm.add_widget(TextGeneratorScreen(name="textbot"))
        sm.add_widget(CodeGeneratorScreen(name="code"))
        sm.add_widget(PoetryGeneratorScreen(name="poem"))
        sm.add_widget(ProblemSolverScreen(name="solve"))
        sm.add_widget(StoryGeneratorScreen(name="Story"))
        sm.add_widget(TranslatorScreen(name="trans"))
        sm.add_widget(SpeechScreen(name="speech"))
        sm.add_widget(SummarizationScreen(name="Summarize"))
        sm.add_widget(HomePage(name="about"))
        sm.add_widget(AdminLoginScreen(name="admin"))
        sm.add_widget(AdminDashboardScreen(name="admim_dashboard"))
        sm.current = "index"  # Start from the Index Page
        return sm

if _name___== "_main_":
    MainApp().run()
```

Chatbot.py:

```
import os
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput
from kivy.core.window import Window
from kivy.uix.scrollview import ScrollView
from kivy.uix.gridlayout import GridLayout
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.relativelayout import RelativeLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, RoundedRectangle, Ellipse, Rectangle, Line
from kivy.utils import get_color_from_hex
from kivy.animation import Animation
from kivy.metrics import dp, sp
from kivy.clock import Clock
from kivy.properties import NumericProperty, BooleanProperty, ObjectProperty
```

```
from kivy.uix.image import AsyncImage
from together import Together
import threading
import random
import  math
TOGETHER_API_KEY="6fcd9484266e33a4e349a09c463bd9b5b0c1f6b66277631ef78e634
c65d8ac2c"
client = Together(api_key=TOGETHER_API_KEY)




    def process_message(self, message):
        try:
            # Check if we should generate an image (if message contains image request keywords)
                is_image_request = any(keyword in message.lower() for keyword in ["picture",
"image", "draw", "photo", "generate image"])

            if is_image_request:
                # Call generate_image directly - no need for another thread
                self.generate_image(message)
            else:
                # For text-only requests, get text response from the LLM
                response = client.chat.completions.create(
                            model="meta-llama/Llama-3.3-70B-Instruct-Turbo-Free",  # or another
supported model
                    messages=[
                            {"role": "system", "content": "You are GenVox, a helpful and friendly AI
assistant."},
                        {"role": "user", "content": message}
                    ],
                    max_tokens=1024
                )

                # Get the assistant's response
                assistant_response = response.choices[0].message.content

                # Remove typing indicator and add the text response using add_message method
                Clock.schedule_once(lambda dt: self.remove_typing_indicator(), 1)
                    Clock.schedule_once(lambda dt: self.add_message(message=assistant_response,
bot=True), 1.2)

        except Exception as e:
            # Handle errors
            error_message = f"Sorry, I encountered an error: {str(e)}"
            Clock.schedule_once(lambda dt: self.remove_typing_indicator(), 1)
                    Clock.schedule_once(lambda  dt:  self.add_message(message=error_message,
bot=True), 1.2)

    def add_bot_response(self, response):
```

```
        # Remove typing indicator first
        self.remove_typing_indicator()
        # Add the bot's response
        self.add_message(message=response, bot=True)

    def generate_image(self, prompt):
        try:
            # Remove typing indicator since we'll be showing an image
            Clock.schedule_once(lambda dt: self.remove_typing_indicator(), 1)

            # Call image generation API
            response = client.images.generate(
                model="stabilityai/stable-diffusion-xl-base-1.0",  # Use the full model name
                prompt=prompt,
                n=1,


  size="512x512"
            )

            # Get image URL from response
            image_url = response.data[0].url

            # Display the image in the UI
            Clock.schedule_once(lambda dt: self.add_image_message(image_url), 1.5)

        except Exception as e:
            # Handle image generation errors
            error_message = f"I couldn't generate that image: {str(e)}"
            Clock.schedule_once(lambda dt: self.remove_typing_indicator(), 1)
                    Clock.schedule_once(lambda  dt:  self.add_message(message=error_message,
bot=True), 1.2)
class GenVoxApp(App):
    def build(self):
        return ChatBotUI()

if _name___== "_main_":
    GenVoxApp().run()
```

Voice.py:

```
class VoiceManager:
    def __init__(self):
        # Initialize speech recognizer
        self.recognizer = sr.Recognizer()
        self.recognizer.dynamic_energy_threshold = True
        self.recognizer.energy_threshold = 300  # Adjust sensitivity

        # Initialize text-to-speech engine
```

```
        self.engine = pyttsx3.init()
        voices = self.engine.getProperty('voices')
        # Set female voice if available
        for voice in voices:
            if 'female' in voice.name.lower():
                self.engine.setProperty('voice', voice.id)
                break

        # Set speech rate (default is 200)
        self.engine.setProperty('rate',  180)

        # Queue for speech output
        self.speech_queue = queue.Queue()

        # Thread for speech output
        self.speech_thread = threading.Thread(target=self._speech_worker, daemon=True)
        self.speech_thread.start()



 # Variables for recording and speech state
        self.is_listening = False
        self.listening_thread = None
        self.is_speaking = False

    def start_listening(self, callback):
        """Start listening for voice input"""
        with self.thread_lock:
            if self.is_listening:
                return

            self.is_listening = True
              self.listening_thread = threading.Thread(target=self._listen_worker, args=(callback,),
daemon=True)
            self.listening_thread.start()

    def stop_listening(self):
        """Stop listening for voice input"""
        self.is_listening = False
        if self.listening_thread:
            self.listening_thread.join(timeout=1)
            self.listening_thread = None

    def _listen_worker(self, callback):
        """Background worker for voice recognition"""
        with sr.Microphone() as source:
            self.recognizer.adjust_for_ambient_noise(source,  duration=0.5)

def _speech_worker(self):
        """Background worker for text-to-speech"""
```

```
        while True:
            try:
                text = self.speech_queue.get()
                if text:
                    self.is_speaking = True

                    # Split text into smaller chunks for more natural speech
                    chunks = self._split_text_into_chunks(text)
                    for chunk in chunks:
                        self.engine.say(chunk)
                        self.engine.runAndWait()

                    self.is_speaking = False

                self.speech_queue.task_done()
            except Exception as e:
                print(f"Error in speech synthesis: {e}")
                self.is_speaking = False


class ChatBubble(BoxLayout):
    def __init__(self, message="", bot=False, **kwargs):
        super()._init_(orientation="horizontal", size_hint=(None, None), padding=(dp(10),
dp(5)), spacing=dp(5), **kwargs)

        # Use BoxLayout as container for label to ensure proper alignment
        self.label_container = BoxLayout(orientation='vertical', size_hint=(1, None))

        # Create the message label with dynamic font size and proper text wrapping
        self.msg_label = Label(
            text=message,
            font_size=sp(16),
            size_hint=(1, None),
            color=TEXT_COLOR,
            halign=align,


            valign="middle",
            text_size=(self.bubble_width - dp(20), None),
            padding=(dp(10), dp(10)),
            markup=True,
            shorten=False,
            line_height=1.2
        )

        self.msg_label.bind(texture_size=self._adjust_label_size)
        self.label_container.add_widget(self.msg_label)
        self.label_container.bind(minimum_height=self.label_container.setter('height'))

        self.add_widget(self.label_container)
```

```
        # Bind to window size changes for responsiveness
        Window.bind(on_resize=self._on_window_resize)



class ImageChatBubble(BoxLayout):
    def __init__(self, image_url="", bot=True, **kwargs):
            super()._init_(orientation="vertical", size_hint=(None, None), padding=(dp(10),
dp(5)), spacing=dp(5), **kwargs)
        self.bot = bot
        self.bubble_width = min(dp(300), Window.width * 0.75)
        self.width = self.bubble_width
        self.height = dp(300)  # Starting height for image



        # Background color for bot messages
        bg_color = (40/255, 80/255, 170/255, 0.7)  # Translucent blue for bot

        with self.canvas.before:
            Color(*bg_color)
            self.bg_rect = RoundedRectangle(size=self.size, pos=self.pos, radius=[dp(20)])

        self.bind(
            size=self._update_rect,
            pos=self._update_rect
        )



def process_user_input(self, text):
        # Show typing indicator
        typing_bubble = self.add_message("Thinking...", bot=True)

        # Queue the API request
        self.request_queue.put((text, typing_bubble))

    def _process_api_requests(self):
        """Worker thread to process API requests"""
        while True:
            try:
                user_text, typing_bubble = self.request_queue.get()

                try:
                    # Make API call to Together API
                    response = client.chat.completions.create(
                        model="mistralai/Mistral-7B-Instruct-v0.1",
                        messages=[
                                {"role": "system", "content": "You are GenVox, a helpful, concise, and
friendly voice assistant. Provide clear and direct answers."},
                            {"role": "user", "content": user_text}
```

```
                ],
                temperature=0.7,
                max_tokens=500
            )

            # Get assistant's response
            assistant_response = response.choices[0].message.content

def add_message(self, message, bot=True):
    # Create a chat bubble for the message
    bubble = ChatBubble(message=message, bot=bot)

    # Add the bubble to the chat layout
    self.chat_layout.add_widget(bubble)
    self.chat_bubbles.append(bubble)

    # Scroll to the bottom to show the latest message
    Clock.schedule_once(self._scroll_to_bottom, 0.1)

    return bubble


class VoiceAssistantApp(App):
  def build(self):
    self.title = 'GenVox Voice Assistant'
    # Set window size - adjust for your testing
    Window.size = (400, 700)
    return VoiceAssistantUI()

if __name__ == '__main__':
  VoiceAssistantApp().run()
```

Image_generation.py:

```
def generate_image(self):
    prompt = self.ids.prompt_input.text.strip()
    if prompt:
        # Show loading message
        self.ids.feedback_label.text = "Generating image..."

        # Clear previous image
        self.ids.image_display.source = ""

        # Run the API call in a background thread to avoid freezing the UI
        threading.Thread(target=self._execute_api_call, args=(prompt,)).start()
    else:
        self.ids.feedback_label.text = "Please enter a description for the image."

  def _execute_api_call(self, prompt):
    try:
```

```
            response = client.images.generate(
                prompt=prompt,
                model="stabilityai/stable-diffusion-xl-base-1.0",
                width=1024,
                height=1024,
                num_images=1
            )

            if response and hasattr(response, "data") and response.data:
                image_url = response.data[0].url
                # Update UI in main thread
                Clock.schedule_once(lambda dt: self._update_image(image_url), 0)
            else:
                # Handle empty response
                  Clock.schedule_once(lambda dt: self.update_feedback("Failed to get image from
API!"), 0)

        except Exception as e:
            # Create a local variable to avoid lambda scope issues
            error_message = f"Error: {str(e)}"
            Clock.schedule_once(lambda dt: self.update_feedback(error_message), 0)

    def _update_image(self, image_url):
        self.ids.image_display.source = image_url
        self.ids.feedback_label.text = "Image generated successfully!"

    def clear_inputs(self):
        self.ids.prompt_input.text = ''
        self.ids.image_display.source = ''
        self.ids.feedback_label.text = ''

class ImageGenApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(ImageGeneratorScreen(name="image"))
        # Add a menu screen (placeholder)
        return sm

if __name__ == '__main__':
    ImageGenApp().run()
```

Code_generator.py:

```
class CodeGeneratorScreen(Screen):
    def __init__(self, **kwargs):
        super(CodeGeneratorScreen, self).__init__(**kwargs)
        self.recording = False
        self.recognizer = sr.Recognizer()
    def listen_for_speech(self):
        try:
```

```
            with sr.Microphone() as source:
                # Adjust for ambient noise
                self.recognizer.adjust_for_ambient_noise(source, duration=0.5)
                audio = self.recognizer.listen(source, timeout=10.0)

            # Try to recognize the speech
            try:
                text = self.recognizer.recognize_google(audio)
                # Update the UI in the main thread and auto-generate response
                Clock.schedule_once(lambda dt: self.update_text_and_generate(text), 0)
            except sr.UnknownValueError:
                # Use a function that doesn't depend on accessing the exception variable
                    Clock.schedule_once(lambda dt: self.update_feedback("Could not understand
audio"), 0)
            except sr.RequestError:
                # Pass the error message directly to avoid lambda scope issues
                error_msg = "Error connecting to Google Speech Recognition service"
                Clock.schedule_once(lambda dt: self.update_feedback(error_msg), 0)

        except Exception as e:
            # Create a local variable with the error message to avoid lambda scope issues
            error_message = f"Error: {str(e)}"
            Clock.schedule_once(lambda dt: self.update_feedback(error_message), 0)

        # Reset recording state in the main thread
        Clock.schedule_once(lambda dt: self.reset_recording_state(), 0)

    def update_feedback(self, message):
        self.ids.feedback_label.text = message

    def reset_recording_state(self):
        self.recording = False
        self.ids.voice_button.text = '
self.ids.voice_button.background_color = (64/255, 89/255, 140/255, 1)

    def ask_question(self):
        user_query = self.ids.user_input.text.strip()
        if user_query:
            # Show loading message
            self.ids.feedback_label.text = "Generating response..."

            # Run the API call in a background thread to avoid freezing the UI
            threading.Thread(target=self._execute_api_call, args=(user_query,)).start()
        else:
            self.ids.feedback_label.text = "Please enter a valid question."

    def _execute_api_call(self):
        user_query = self.ids.user_input.text.strip()
        try:
            response = client.chat.completions.create(
```

```
            model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
            messages=[{"role": "user", "content": user_query}],
        )
        response_text = response.choices[0].message.content.strip()

        # Update UI in main thread
        Clock.schedule_once(lambda dt: self._update_response(response_text), 0)
    except Exception as e:
        # Create a local variable to avoid lambda scope issues
        error_message = f"Error: {str(e)}"
        Clock.schedule_once(lambda dt: self.update_feedback(error_message), 0)
def clear_inputs(self):
    self.ids.user_input.text = ''
    self.ids.response_output.text = ''
    self.ids.feedback_label.text = ''
class CodeGenApp(App):
def build(self):
    sm = ScreenManager()
    sm.add_widget(CodeGeneratorScreen(name="code"))
    # Add a menu screen (placeholder)
    return sm

if __name__ == '__main__':
CodeGenApp().run()
```

Poetry_Generator.py:

```
class PoetryGeneratorScreen(Screen):
def __init__(self, **kwargs):
    super(PoetryGeneratorScreen, self).__init__(**kwargs)

def update_text(self, text):
    # First update the text input
    current_text = self.ids.theme_input.text
    if current_text and not current_text.endswith(' '):
        self.ids.theme_input.text = f"{current_text} {text}"
    else:
        self.ids.theme_input.text = f"{current_text}{text}"

    self.ids.feedback_label.text = "Voice input received! Generating poem..."

    Clock.schedule_once(lambda dt: self.generate_poem(), 0.5)

def generate_poem(self):
    theme = self.ids.theme_input.text.strip()
    if theme:
        # Show loading message
        self.ids.feedback_label.text = "Generating poem..."

        # Run the API call in a background thread to avoid freezing the UI
        threading.Thread(target=self._execute_api_call, args=(theme,)).start()
```

```
        else:
            self.ids.feedback_label.text = "Please enter a theme for your poem."


    def _format_poem_for_speech(self, poem):
        """Format the poem to make it sound better when read aloud
        - Add pauses at the end of lines
        - Emphasize certain words
        - Add proper inflection
        """
        # Simple implementation - add commas at the end of lines for natural pauses
        lines = poem.split('\n')
        formatted_lines = []

        for line in lines:
            if line.strip():  # Skip empty lines
                # If the line doesn't end with punctuation, add a comma for a pause
                if not line.strip()[-1] in ['.', ',', '!', '?', ':', ';']:
                    line += ','
                formatted_lines.append(line)
            else:
                # Add a longer pause for stanza breaks (empty lines)
                formatted_lines.append(".")  # Period causes a longer pause

        return '\n'.join(formatted_lines)
    def clear_inputs(self):
        self.ids.theme_input.text = ''


        self.ids.poem_output.text = ''
        self.ids.feedback_label.text = ''

        # Disable speak button when there's no poem
        try:
            self.ids.speak_button.disabled = True
        except:
            pass
class PoetryApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(PoetryGeneratorScreen(name="poetry"))
        # Add a menu screen (placeholder)
        return sm

if __name__ == '__main__':
    PoetryApp().run()
```

Problem_solver.py:

```
class ProblemSolverScreen(Screen):
```

```
    def _init_(self, **kwargs):
        super(ProblemSolverScreen, self)._init_(**kwargs)

    def solve_problem(self):
        problem = self.ids.problem_input.text.strip()
        if problem:
            # Show loading message
            self.ids.feedback_label.text = "Analyzing problem..."

            # Provide haptic feedback if on Android
            if platform == 'android' and vibrator and hasattr(vibrator, 'vibrate'):
                vibrator.vibrate(0.1)  # Short vibration

            # Run the API call in a background thread to avoid freezing the UI
            threading.Thread(target=self._generate_solution, args=(problem,)).start()
        else:
            self.ids.feedback_label.text = "Please describe your problem first."
def _generate_solution(self, problem):
        try:
            # Using Together API for the problem solving
            headers = {
                "Authorization": f"Bearer {TOGETHER_API_KEY}",
                "Content-Type": "application/json"
            }

            # Prepare the request payload for Together API
            # Adjust model and parameters as needed


payload = {
                "model":  "mistralai/Mixtral-8x7B-Instruct-v0.1",


"prompt": f"I need to solve the following problem step by step:\n\n{problem}\n\n",
                "max_tokens": 1024,
                "temperature": 0.7,
                "top_p": 0.9,
                "frequency_penalty": 0,
                "presence_penalty": 0
            }

            # Make the API request
            response = session.post(TOGETHER_API_URL, headers=headers, json=payload)

            # Parse the response
            if response.status_code == 200:
                result = response.json()
                # Extract solution from the response
                # Note: Adjust this based on the actual structure of Together API response
                if 'choices' in result and len(result['choices']) > 0:
                    solution_text = result['choices'][0]['text'].strip()
```

```
                # Update UI in main thread
                Clock.schedule_once(lambda dt: self._update_solution(solution_text), 0)
            else:
                # Handle unexpected response format
                Clock.schedule_once(
                    lambda dt: self.update_feedback("Error: Unexpected response format from
API"),
                    0
                )

    def _format_text_for_speech(self, text):
        """Format the text to make it sound better when read aloud"""
        # Add pauses at appropriate points
        # Replace certain symbols with their spoken equivalents
        text = text.replace('=', 'equals')
        text = text.replace('+', 'plus')
        text = text.replace('-', 'minus')
        text = text.replace('*', 'times')
        text = text.replace('/', 'divided by')
        text = text.replace('√', 'square root of')
        text = text.replace('^', 'to the power of')

        # Add pauses after sentences
        text = text.replace('. ', '. <break> ')

        return text

    def stop_speaking(self):
        """Stop the TTS engine if it's speaking"""
        if self.tts_engine and self.speaking:
            self.tts_engine.stop()



 self.reset_speaking_state()

    def reset_speaking_state(self):
        """Reset the speaking button state"""


        self.speaking = False
        self.ids.speak_button.text = '}
        self.ids.speak_button.background_color = (64/255, 89/255, 140/255, 1)
        self.ids.feedback_label.text = "Solution ready"
class ProblemSolverApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(ProblemSolverScreen(name="problemsolver"))
        # To add a menu screen:
        # sm.add_widget(MenuScreen(name="menu"))
        return sm
```

```
if _name___== '_main_':
    ProblemSolverApp().run()
```

Stroyteller.py:

```
class StoryGeneratorScreen(Screen):
    def _init_(self, **kwargs):
        super(StoryGeneratorScreen, self)._init_(**kwargs)
        self.client =
Together(api_key="a77f068040638cbe4879024a9d83a58f7b6775708fad8483d0802c468ff79
566")
        def generate_story(self):
        story_idea = self.ids.story_input.text.strip()
        if story_idea:
            # Show loading message
            self.ids.feedback_label.text = "Generating story..."

            # Run the API call in a background thread to avoid freezing the UI
            threading.Thread(target=self._execute_api_call, args=(story_idea,)).start()
        else:
            self.ids.feedback_label.text = "Please enter a story idea."

    def _execute_api_call(self, story_idea):
        prompt = f"Write a short creative story based on this idea: {story_idea}"
        try:
            response = self.client.chat.completions.create(
                model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
                messages=[{"role": "user", "content": prompt}],
            )
            response_text = response.choices[0].message.content.strip()

            # Update UI in main thread
            Clock.schedule_once(lambda dt: self._update_response(response_text), 0)

except Exception as e:
            error_message = f"Error: {str(e)}"
            Clock.schedule_once(lambda dt: self.update_feedback(error_message), 0)

    def _update_response(self, response_text):
        self.ids.story_output.text = response_text
        self.ids.feedback_label.text = "Story generated successfully!"


class StoryGeneratorApp(App):
    def build(self):
        return StoryGeneratorScreen()

if _name___== '_main_':
    StoryGeneratorApp().run()
```

Text_to_speech.py:

```
class SpeechScreen(Screen):
    def _init_(self, **kwargs):
        super(SpeechScreen, self)._init_(**kwargs)
        self.audio_path = "temp_speech.mp3"
        self.sound = None
        self.paused_position = 0
        self.volume = 1.0
        Clock.schedule_once(self.init_ui, 0.1)

    def generate_speech(self):
        # Get the text input
        text_input = self.ids.text_input.text.strip()
        if not text_input:
            self.show_loading_label("Please enter some text!")
            return

        # Show loading label
        self.show_loading_label("Generating speech...")

        try:
            tts = gTTS(text=text_input, lang="en")
            tts.save(self.audio_path)
            self.sound = SoundLoader.load(self.audio_path)
            if self.sound:
                self.sound.volume = self.volume
                self.ids.audio_status.text = f"00 / {int(self.sound.length)}"
                self.ids.progress_bar.max = self.sound.length
            self.show_loading_label("Speech generated!")
            Clock.schedule_once(lambda dt: self.hide_loading_label(), 2)
        except Exception as e:
            self.show_loading_label(f"Error: {str(e)}")

    def play_audio(self):
        if self.sound:
            try:
                if self.sound.state == "play":


                self.paused_position = self.sound.get_pos()
                    self.sound.stop()
                    self.ids.play_button.text = "play"
                else:
                    self.sound.seek(self.paused_position)
                    self.sound.play()
                    self.ids.play_button.text = " ⏸ "
                    Clock.schedule_interval(self.update_audio_status, 1)

    except Exception as e:
                print(f"Error playing audio: {str(e)}")
```

```
    def download_audio_android(self):
        try:
            storage_path = primary_external_storage()
            downloads_dir = os.path.join(storage_path, "Download")
            if not os.path.exists(downloads_dir):
                os.makedirs(downloads_dir)
            save_path = os.path.join(downloads_dir, "speech_output.mp3")
            import shutil
            shutil.copy2(self.audio_path, save_path)
            self.show_loading_label("Saved to Downloads folder")
            Clock.schedule_once(lambda dt: self.hide_loading_label(), 2)
        except Exception as e:
            self.show_loading_label(f"Error saving: {str(e)}")

    def upload_file(self):
        if platform == "android":
            self.upload_file_android()
        else:
            self.upload_file_desktop()
    def upload_file_android(self):
        try:
            file_path = filechooser.open_file(filters=[("*.txt")])[0]
            if file_path:
                with open(file_path, "r", encoding="utf-8") as file:
                    text_content = file.read()
                    self.ids.text_input.text = text_content
        except Exception as e:
            self.show_loading_label(f"Error loading file: {str(e)}")
```

Text_generation.py:

```
class TextGeneratorScreen(Screen):
    def _init_(self, **kwargs):
        super(TextGeneratorScreen, self)._init_(**kwargs)
        self.client =
Together(api_key="a77f068040638cbe4879024a9d83a58f7b6775708fad8483d0802c468ff79
566"
    def generate_text(self):
        user_query = self.ids.prompt_input.text.strip()
        if user_query:
            # Show loading message
            self.ids.feedback_label.text = "Generating text..."

            # Run the API call in a background thread to avoid freezing the UI
            threading.Thread(target=self._execute_api_call, args=(user_query,)).start()
        else:
            self.ids.feedback_label.text = "Please enter a valid prompt."

    def _execute_api_call(self, user_query):
```

```
        try:
            response = self.client.chat.completions.create(
                model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
                messages=[{"role": "user", "content": user_query}],
            )
            response_text = response.choices[0].message.content.strip()

            # Update UI in main thread
            Clock.schedule_once(lambda dt: self._update_response(response_text), 0)
        except Exception as e:
            # Create a local variable to avoid lambda scope issues
            error_message = f"Error: {str(e)}"
            Clock.schedule_once(lambda dt: self.update_feedback(error_message), 0)

    def _update_response(self, response_text):
        self.ids.response_output.text = response_text
        self.ids.feedback_label.text = "Text generated successfully!"

    def clear_inputs(self):
        self.ids.prompt_input.text = ''
        self.ids.response_output.text = ''
        self.ids.feedback_label.text = ''

class TextGeneratorApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(TextGeneratorScreen(name="text_generator"))
        # You would add a menu screen here in a full app
        sm.current = "text_generator"
        return sm

if __name__ == '__main__':
    TextGeneratorApp().run()
```

Text_summarization.py:

```
def summarize_text(self, instance):
        user_input = self.input_text.text.strip()
        if user_input:
            try:
                self.feedback_label.text = "Generating summary, please wait..."
                response = client.chat.completions.create(
                    model="meta-llama/Llama-3.3-70B-Instruct-Turbo",
                    messages=[
                        {"role": "user", "content": f"Summarize the following text clearly and
concisely:\n\n{user_input}\n\nSummary:"}
                    ],
                    max_tokens=100,

 temperature=0.2,
                )
```

```
            self.output_text.text = response.choices[0].message.content.strip()
            self.feedback_label.text = "Summary generated successfully!"


 except Exception as e:
            self.feedback_label.text = f"Error: {str(e)}"
        else:
            self.feedback_label.text = "Please enter valid text."

# For integration with main.py
class SummarizationApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(SummarizationScreen(name='summarize'))
        return sm

# For standalone testing
if _name___== '_main_':
    SummarizationApp().run()
```

Language_translator.py:

```
  class TranslatorScreen(Screen):
        def __init__(self, **kwargs):
            super().__init__(**kwargs)
        layout = BoxLayout(orientation='vertical', padding=20, spacing=10)
        self.add_widget(layout)

        # Background
        with self.canvas.before:
            Color(76/255, 94/255, 132/255, 1)
            self.bg = Rectangle(size=self.size, pos=self.pos)
        self.bind(size=self.update_bg, pos=self.update_bg)

        # Header with Back Button
        header_layout = BoxLayout(orientation='horizontal', size_hint=(1, 0.1))
        back_button = Button(text='⬅ Back', size_hint=(None, None), size=(80, 40))
        back_button.bind(on_press=self.go_back)
        header_layout.add_widget(back_button)
        header_layout.add_widget(Label(text='Language Translator', font_size='28sp',
bold=True))
        layout.add_widget(header_layout)

        # Language Selection
        self.language_spinner = Spinner(
            text='Select Language',
            values=list(HF_API_URLS.keys()),
            size_hint=(1, 0.1)
```

```
        )

    layout.add_widget(self.language_spinner)

        # User Input
        self.user_input = TextInput(hint_text='Enter text to translate...', multiline=False,
                                    background_color=(190/255, 208/255, 244/255, 1),
                                    foreground_color=(0, 0, 0, 1),

                                     font_size='16sp', size_hint=(1, 0.1))
        layout.add_widget(self.user_input)

        # Response Output
        self.response_output = TextInput(readonly=True,
                                         background_color=(190/255, 208/255, 244/255, 1),
                                         foreground_color=(0, 0, 0, 1),
                                         hint_text='Translated text will appear here...',
                                         size_hint=(1, 0.6))
        layout.add_widget(self.response_output)

        # Buttons
        button_layout = BoxLayout(size_hint=(1, 0.1), spacing=10)
        translate_button = Button(text='Translate', background_color=(0, 0, 0, 1))
        clear_button = Button(text='Clear', background_color=(0, 0, 0, 1))
        translate_button.bind(on_press=self.translate_text)
        clear_button.bind(on_press=self.clear_inputs)
        button_layout.add_widget(translate_button)
        button_layout.add_widget(clear_button)
        layout.add_widget(button_layout)

        # Feedback Label
        self.feedback_label = Label(text='', color=(0, 0, 0, 1), size_hint=(1, 0.05))
        layout.add_widget(self.feedback_label)

    def update_bg(self, *args):
        self.bg.size = self.size
        self.bg.pos = self.pos

    def go_back(self, instance):
        self.manager.current = "menu"

    def translate_text(self, instance):
        user_text = self.user_input.text.strip()
        selected_language = self.language_spinner.text
        api_url = HF_API_URLS.get(selected_language)

        if user_text and api_url:
            try:
                payload = {"inputs": user_text}
                response = session.post(api_url, headers=HEADERS, json=payload)
                translated_text = response.json()[0]['translation_text'].strip()
```

```
                self.response_output.text = translated_text
                self.feedback_label.text = "Translation successful!"
            except Exception as e:
                self.feedback_label.text = f"Error: {str(e)}"


    else:
            self.feedback_label.text = "Please enter text and select a language."

    def clear_inputs(self, instance):
        self.user_input.text = ''
        self.response_output.text = ''
        self.feedback_label.text = ''


class TranslatorApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(MenuScreen(name="menu"))
        sm.add_widget(TranslatorScreen(name="translator"))
        return sm

if __name__ == '__main__':
    TranslatorApp().run()
```

Chapter 7

RESULTS AND SCREEN SHOTS

Fig:[7.1] Access Mode page

Fig:[7.2] Sign in page

Fig:[7.3] Sign up page

Fig:[7.4] Model Selection page

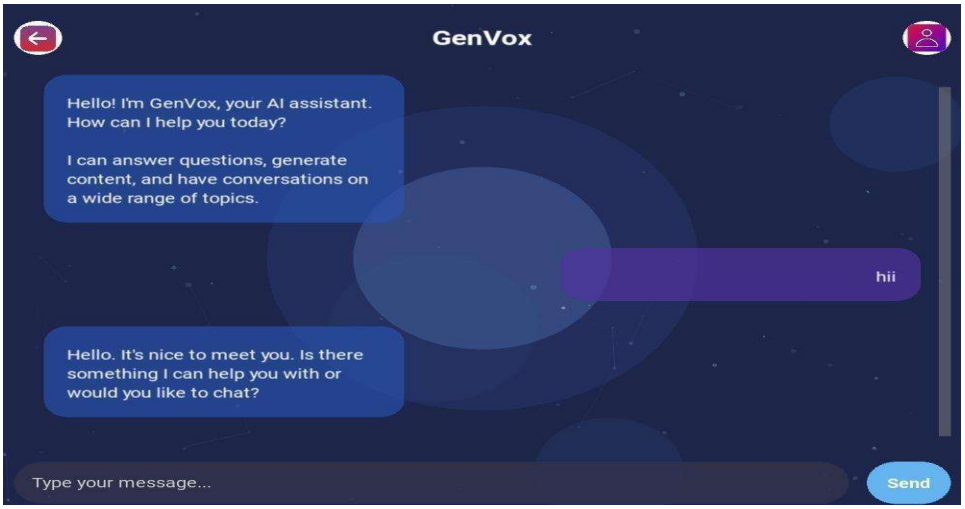


Fig:[7.5] Chatbot page

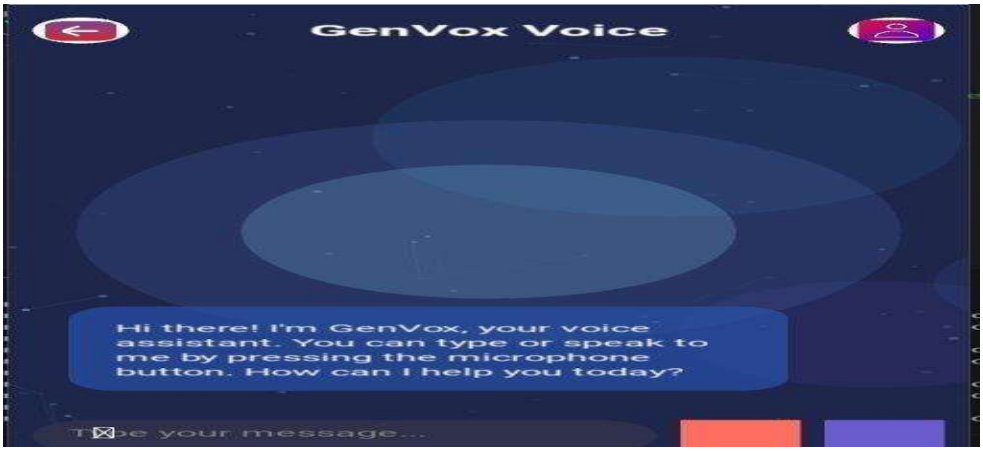


Fig: [7.6] Voice assistant page

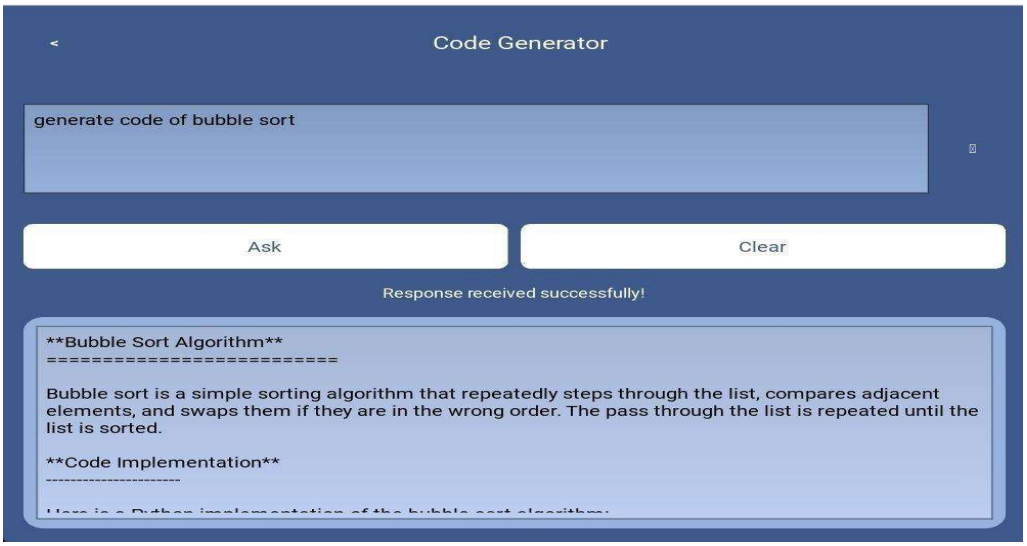


Fig:[7.7] Code generator page

Fig:[7.8] Image generator page

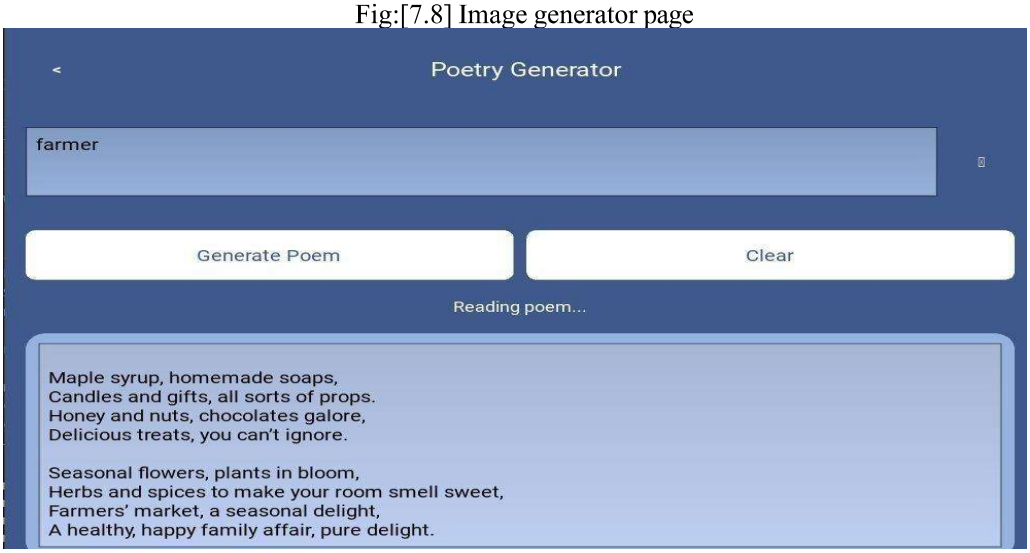


Fig:[7.9] Poetry generation page

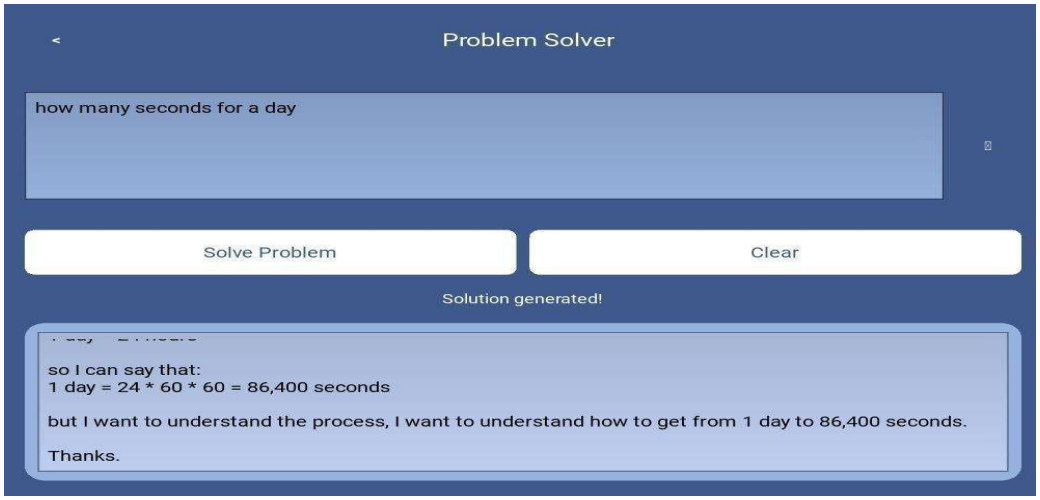


Fig:[7.10] Problem Solver page

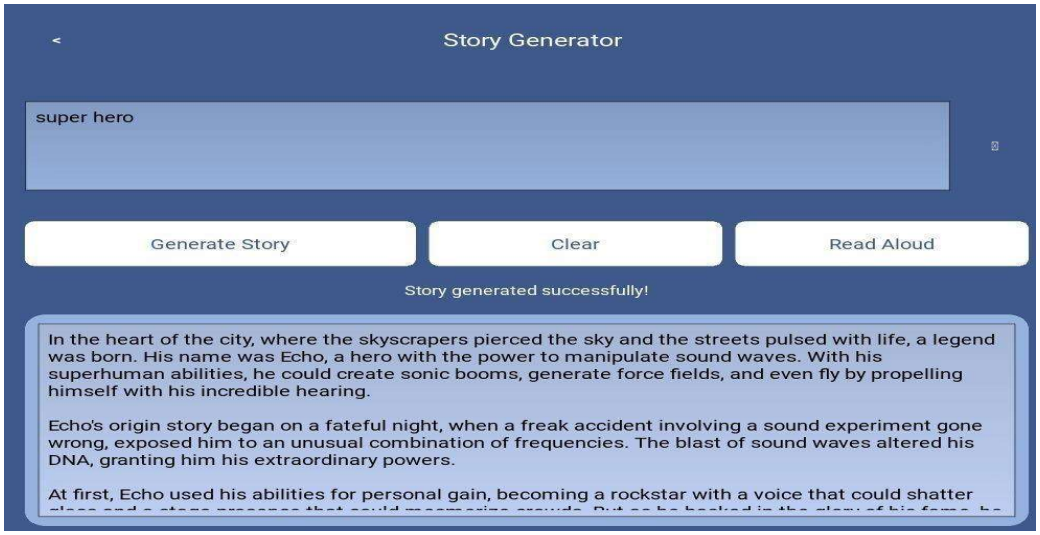


Fig:[7.11] Story Generator page

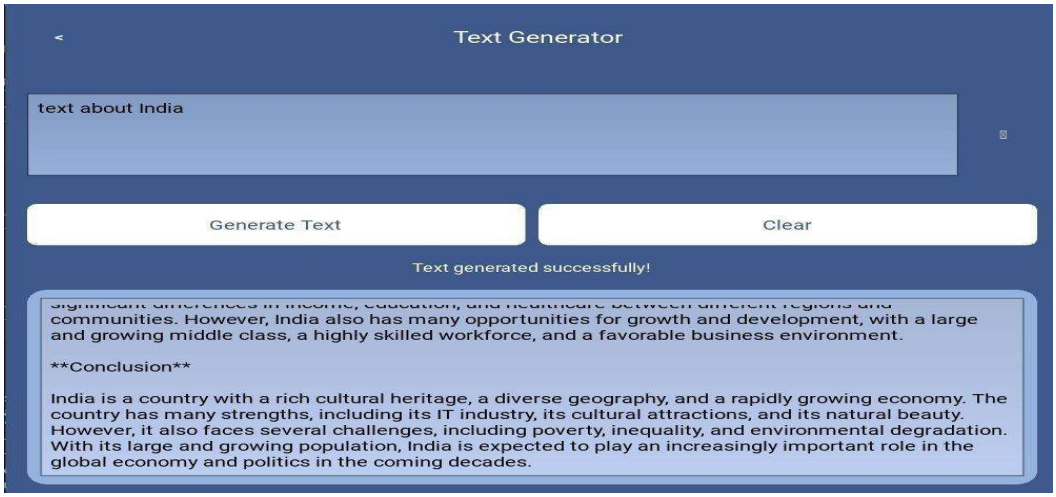


Fig: [7.12] Text Generator page

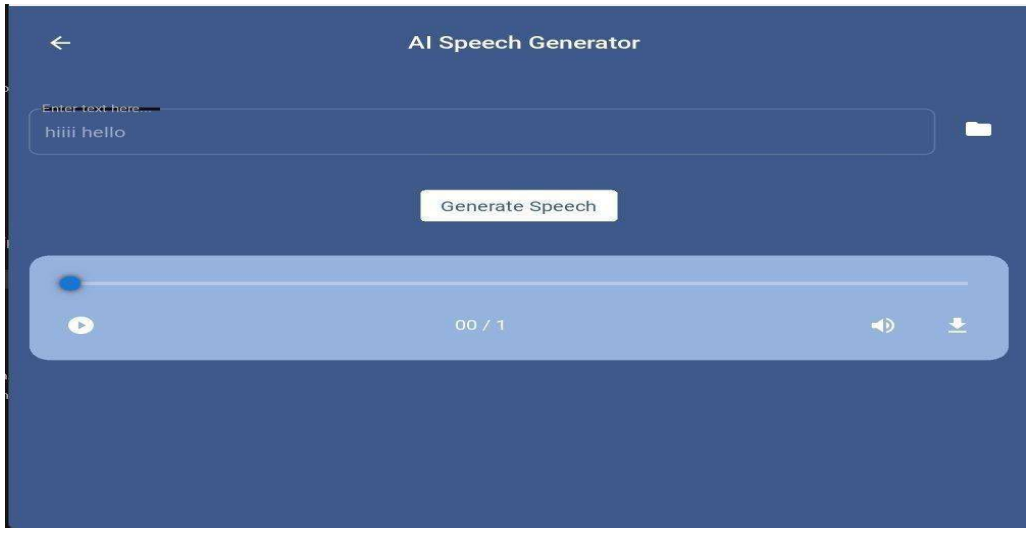


Fig: [7.13] Text-to-Speech page

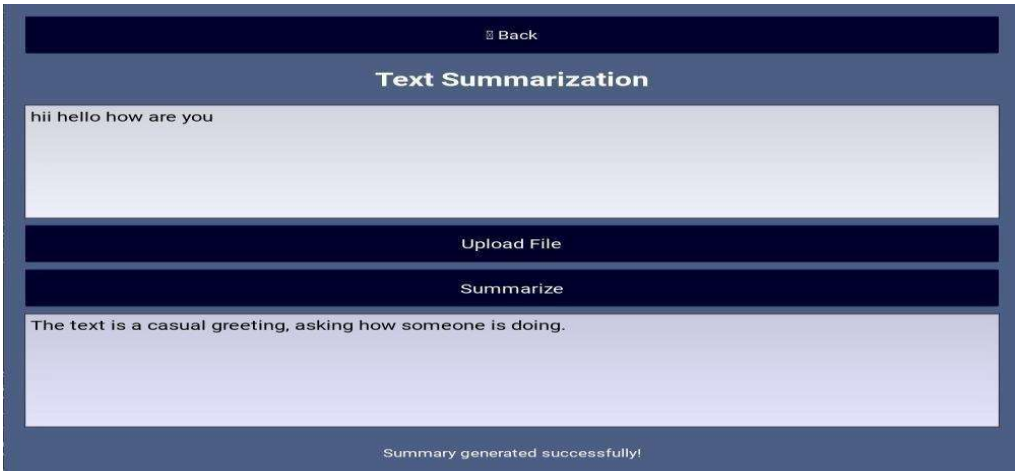


Fig: [7.14] Text Summarization page

Fig: [7.15] Language Translation page

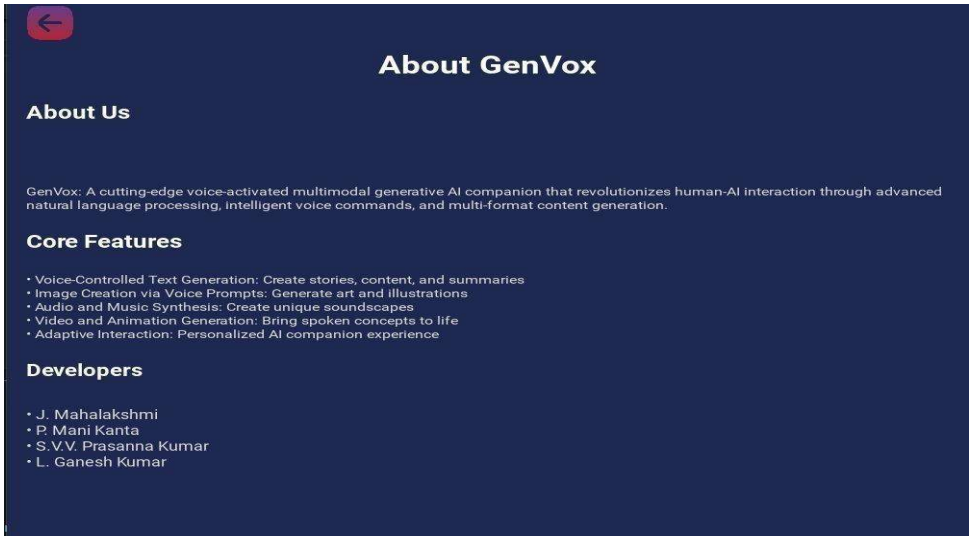


Fig: [7.16] About page

Chapter 8

8.1 FUTURE SCOPE

These applications use different forms of AI-driven automation and integrations to create better seamlessness, user experience, access and efficiency in their respective use cases.

The Voice Assistant application uses new speech to text processing capability to provide the user with hands free control to improve access and efficiency in the present use case. The Worker-Connect app reduces the burden of managing workers, with base capabilities such as worker registration, live location tracking and real-life service discovery, in order to improve productivity.

The AI Capable Chatbot will have the capability to perform intelligent dynamic conversations using free LLMs, while using its context awareness, and thus also provide instant response capability for the system in real-time. The Code Gen Tool will provide code generation from natural language prompts, which improves development speed, including an API Key Automation for a seamless integration experience.

Future work includes AI Optimization, Mult-lingual Capability, Security Improvements and UI/UX to make applications more robust and streamlined. Overall, these applications each use modern technology to reimagine automation, accessibility and intelligent real-time conversation for better smarter more adaptable AI enablement for all users

8.2 CONCLUSION

GenVox employs multimodal generative AI for a next level re-imagined user experience around speech recognition, content generation, and task automation.

The next generation of GenVox will include the following capabilities: - *Improved performance and AI*, including faster response times. - *Improved access for international audiences* with multilingual use. - *Improved security* with better encryption of data. - *Improved UI/UX*, providing greater engagement from the user.

GenVox move forward as an overall AI-centered automation experience with seamless integration and represents the advancement of next generation voice enabled AI applications.

REFERENCE

[1] Vaswani,A.,Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30, 5998-6008(Transformer model, fundamental for LLMs like LLaMA-3 used in your project)

[2] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems, 33, 1877-1901.

[3] (GPT-based models, relevant to your chatbot and text generation module)

[4] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10684-10695.

[5] (Stable Diffusion, related to your image generation module)

[6] Zhang, Y., & Chen, Q. (2023). *Multimodal Context Retention in Conversational AI*. Journal of Artificial Intelligence Research, 76, 45-78.

[7] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. International Conference on Machine Learning, 8748-8763.

[8] (CLIP-based text-to-image models used in Stable Diffusion)

[9] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". *Journal of Machine Learning Research*, 21(140), 1-67.

[10] Yu, J., Xu, Y., Koh, J. Y., Luong, T., Baid, G., Wang, Z., ... & Wu, Y. (2022). "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation". *arXiv preprint arXiv:2206.10789*.

[11] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). "Evaluating Large Language Models Trained on Code". *arXiv preprint arXiv:2107.03374*.

[12] Shazeer, N., & Stern, M. (2018). "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". *International Conference on Machine Learning*, 80, 4596-4604.

[13] Fedus, W., Zoph, B., & Shazeer, N. (2022). "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". *Journal of Machine Learning Research*, 23(120), 1-39.

[14] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). "Scaling Laws for Neural Language Models". *arXiv preprint arXiv:2001.08361*.

[15] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). "Training Compute-Optimal Large Language Models". *arXiv preprint arXiv:2203.15556*.

[16] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). "LLaMA: Open and Efficient Foundation Language Models". *arXiv preprint arXiv:2302.13971*.

[17] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2022). "PaLM: Scaling to 540 Billion Parameters". *arXiv preprint arXiv:2204.02311*.

PUBLISHED PAPER AND CERTIFICATES

GUIDE CERTIFICATE :

International Journal of Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

DR. D. ANUSHA

In recognition of the publication of the paper entitled

GENVOX: A VOICE-ACTIVATED MULTI-MODAL GENERATIVE AI COMPANION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 11, April 2025

Registration ID 175182 Research paper weblink:https://ijirt.org/Article?manuscript=175182

EDITOR EDITOR IN CHIEF

TEAM LEADER CERTIFICATE:

International Journal of Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

SIDDAMSETTI VEERA VENKATA PRASANNA KUMAR

In recognition of the publication of the paper entitled

GENVOX: A VOICE-ACTIVATED MULTI-MODAL GENERATIVE AI COMPANION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 11, April 2025

Registration ID 175182 Research paper weblink:https://ijirt.org/Article?manuscript=175182

EDITOR

EDITOR IN CHIEF

TEAM MEMBERS CERTIFICATES:

International Journal of Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

LOPINTI GANESH KUMAR

In recognition of the publication of the paper entitled

GENVOX: A VOICE-ACTIVATED MULTI-MODAL GENERATIVE AI COMPANION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 11, April 2025

Registration ID 175182 Research paper weblink:https://ijirt.org/Article?manuscript=175182

EDITOR

EDITOR IN CHIEF

International Journal of
Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

PANCHALA MANIKANTA

In recognition of the publication of the paper entitled

GENVOX: A VOICE-ACTIVATED MULTI-MODAL GENERATIVE AI COMPANION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 11, April 2025

Registration ID 175182 Research paper weblink:https://ijirt.org/Article?manuscript=175182

EDITOR

EDITOR IN CHIEF

International Journal of
Innovative Research in Technology

An International Open Access Journal Peer-reviewed, Refereed Journal
www.ijirt.org | editor@ijirt.org An International Scholarly Indexed Journal

Certificate of Publication

The Board of International Journal of Innovative Research in Technology (ISSN 2349-6002) is hereby awarding this certificate to

JAMPANA MAHALAKSHMI

In recognition of the publication of the paper entitled

GENVOX: A VOICE-ACTIVATED MULTI-MODAL GENERATIVE AI COMPANION

Published in IJIRT (www.ijirt.org) ISSN UGC Approved (Journal No: 47859) & 7.37 Impact Factor

Published in Volume 11 Issue 11, April 2025

Registration ID 175182 Research paper weblink:https://ijirt.org/Article?manuscript=175182

EDITOR

EDITOR IN CHIEF

ISSN 2349-6002

GenVox: A Voice-Activated Multi-Modal Generative AI Companion

[1]S.V.V. Prasanna Kumar, [2]J. Mahalakshmi ,[3] P. Manikanta, [4]L.Ganesh Kumar, [5]Dr.D. Anusha
[1,2,3,4] *B. Tech Students, Department of CSE-AI&ML, SRK Institute Technology, Vijayawada, A.P, India.*
[5] *Head of Department, Department of CSE-AI&ML, SRK Institute of Technology, Vijayawada, A.P, India*

Abstract: This paper describes GenVox is a multimodal voice-driven generative AI assistant that embeds the new innovation in natural language processing (NLP), image generation, and sound generation. GenVox enables users to interact in voice commands and receive feedback in various forms like text, images, and sound. Developed as an individual assistant, a creative companion, and a learning friend, GenVox employs generative AI models to respond accordingly in order to address the requirements of the users. Major features include voice-guided text generation, AI-based story creation, content generation, question and answer, summarization, voice-activated image generation, and interactive cross-modal content generation. The project also employs technologies such as large language models (LLMs), Google gTTS for text-to-speech synthesis, Pyttsx3 for natural language processing, Python 3.0 for coding, Kivy for Android app construction, and APIs from Together AI and Hugging Face for retrieval of pre-existing generative models.

INTRODUCTION

Artificial Intelligence (AI) has impacted how humans engage with computers differently, but specifically with voice-initiated formats. Current voice assistants do not support fully interactive multimodal communications. GenVox breaks this barrier by integrating text, speech, and image generation to provide seamless interactivity, and depth of experience, while working in tandem with the user. GenVox aims to support productivity, identify efficiencies in creative work, and support accessibility, with a friendly voice interfacing experience. GenVox provides a user a way generate and change text, audio, and visual output all through one AI enabled multimodal experience, making GenVox an adaptable, and deployable tool. GenVox is at the forefront in a cultural shift in how we interface with AI, whether that be content creation, enhancing brainstorming and idea generation forms of creative work, and supporting those with disabilities communicate. GenVox aims to break down the traditional silos of productivity and efficiency and supports not only the advantage that comes with rapid iterative creation of educational resources, but also the amount of creative and application potential that comes with productivity and input as an AI.

LITERATURE SURVEY

New research on AI voice assistants showcases their development in automation, interactivity, and convergence with new technologies. The studies from **2020 to 2024** investigate different uses cases, including tasks automation, nearby information retrieval, smart home automation, the health sector, and chatbots. In the foundational work of **S. Subhash, R. Kumar, P. Sharma, and M. Verma (2020)**, **speech synthesis and recognition** through AI like Google Text-to-Speech and assistants based on Python were tested. From there, projects including **Dilshad Ahmad, Kiran H, Girish Kumar, and Hanumanta DH (2023)** focused on the convergence of **machine learning, IoT, and cloud AI models** to support more real-time interactions and accessibility in smart environments. Further developing this field, through **generative AI, large language models (LLMs), and Natural Language Processing (NLP)**, voice assistants have been converted to more **context-aware and interactive systems**. As evidence of this research, the work of **Shivam Hajong (2024), Devesh Bajpai, Mallela Uday Kiran, Busi Hemanth Reddy, and Suresh Kumar Natarajan (2024), and A. Balamurugan, D. Thiruppathi, S. P. Santhoshkumar, and K. Susithra (2024)** demonstrate that models such as GPT, LLaMA, and Stable Diffusion contribute to

multimodal AI assistants that increase each user's personalization and engagement. All of these ideas show the resulting power of AI-enabled voice assistants to be even more effective for use in the **health sector, increased automation, or through interactive chatbots**, which opens-up the potential for **intelligent, multimodal assistants like GenVox** to facilitate the evolution of human-computer interaction.

EXISTING SYSTEM

At present, AI voice assistants (e.g., Alexa, Google Assistant) utilize solely text and voice input / output and have no '**multimodal capability**. Three notable examples of limitations include - o '**Limited Output Modality:**' Generative AI is not used to output images, video, or rich multimodal content; o '**Limited Interacting with User:**' Static responses to user interaction and limited personalization of user engagement limit user interaction quality; and o '**Limited Functional Utilization:**' AI assistants with voice commands can only be used to function for specific user tasks and cannot easily generalize for more creative applications such as content generation services or task automation. Building a more, '**interactive, adaptive, multimodal AI assistant**' like GenVox is important in addressing the limitations above.

DRAWBACKS TO EXISTING SYSTEM

At present, GAIA (AI voice assistants such Alexa and Google Assistant) show a number of limitations that restrict their effectiveness within **generative AI and multimodal interaction** - **Limited Generative Capability**: These systems do not respond outside of a verbal response - they are incapable of generating response that are complex, evolving and/or not coded into a response. - **Reliance on Pre-Generated Responses**: Their responses are largely dictated by scripted or programmed content, thereby preventing them from evolving, learning and responding in an interactive and dynamic manner - in matches to the speed of input or response by the user. - **No Use in Multi-modal Interaction**: Current GAIA systems do not use text, image, and audio generation in a unified system - therefore limiting their applicability for creative and interactive use. - **Limited Customization Options**: Users also have little to no ability customize anything these systems are capable of, therefore customization may be impossible when responding to a specific automation or creative task. Until these issues, GAIA systems, can be used to create more **intelligent, flexible, and multimodal AIs** like GenVox, to create **personalized, context-sensitive and interactive responses in various forms.**

PROPOSED SYSTEM

GenVox voice assistant is an AI-based multimodal agent for engaging in conversation and dynamic interaction with your device

.1. **Voice/Audio Recognition and Natural Language Processing**: The assistant listens to the user's speech and audio input (provided through the **gTTS** service), converting the speech into text-encoded structured queries. The assistant builds out capability for natural language processing and output responses building from the **LLaMA-3** lib.

2. **Processing Multimodal AI Interactions**: Next, the assistant converts the text input from voice recognition to text-based input allowing for requests to be made to many different AI models for processing. For example, **Stabble Diffusion** model is used for image generation, **LLaMA-3** for code-generation, and **gTTS/Pyttsx3** is used to convert the models messages to audio responses.

3. **Generating Output**: Finally, the response output is printed directly into the app UI, built with the **Kivy framework**, generates a corresponding voice output (audio response from the model) and the user history of the conversation history is recorded and stored to file by using **MongoDB** which can help support the user experience.

PROPOSED SYSTEM ARCHITECTURE

1. *Engagement Process:* The user uses the application to command, verbally.

2. *Send Command:* The application sends the command to the back-end processing it.

3. *Requests to AI Model:* The application asks the back-end to send the correct AI model (text, image, code, audio).

4. *Receive Reply:* When the AI model creates the reply it processes the reply and sends back to application.

5. Input for Reply, Voice for Reply: Application displays content reply to the AI or sends reply via AI generated text to speech.

Figure 1 shows the complete system flow, demonstrating efficient resource sharing among components.

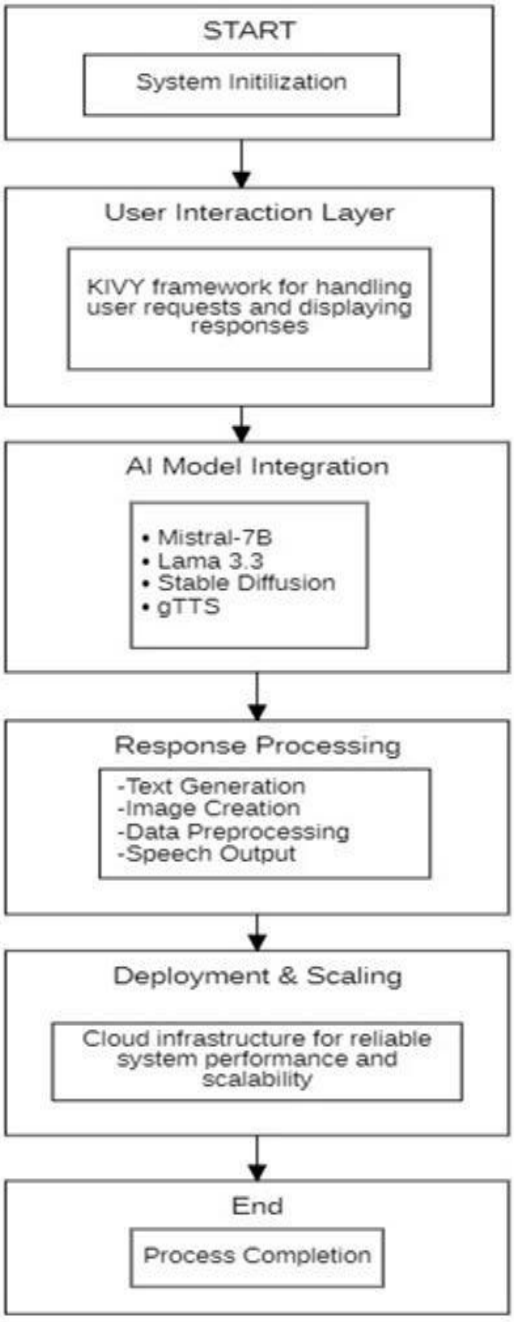


Fig:[1] Architecture of Proposed System

PROPOSED ALGORITHMS EXPLANATION

A. Text-to-Speech Processing (Google gTTS) - *Input:* The user initiates a vocal command; *Processing:* The Google Text-to-speech (TTS) application programming interface (API), analyzes the text output, and synthesizes a natural-sounding audio response; *Output:* The audio output that is produced is read back to the user aloud.

B. Text and code generation (LLaMA-3) – *Input:* The user provides input text or uses the voice query option; *Processing:* The user prompt serves as the input in generating the code "using autoregressive transformers;" *Output:* The output of the model for codex or text generation was provided by the AI.

C. Image Generation (Stable Diffusion) – *Input:* A user prompt that provides how they would like the image represented; *Processing:* The user prompt served as input to, in this case, the Stable Diffusion XL Base 1.0 model "using latent diffusion" to render an image; *Output:* The output was the AI-generated image responding to the user input prompt.

ADVANTAGES OF THE PROPOSED SYSTEM

GenVox enhances the overall end-user experience with multimodal interactions—promoting more dynamic and interactive engagement with the AI. It also automates content generation through helping users (everything from text, to images to code), thus simplifying the burden of creative and technical work. It also increases accessibility around the use of a voice which can be helpful to differently-abled users. It adds more learning, productivity, and automation of tasks through the real-time AI powered responses. It's also scalable so it's easy to add new AI models or even increase functionality, which brings flexibility and continuous improvement of AI based support.

RESULTS

Results from our experimental evaluation show that GenVox has been successfully designed as an **Android application** that combines **voice recognition, natural language processing (NLP), and generative AI models** to provide **multimodal responses**. The application efficiently translates

user voice commands into **text, images, and/or audio**, promoting engagement and accessibility. GenVox also provided **good engagement and good usability** from **large-sample user testing** across different forms of **personal assistance, creative writing, and task automation**. With **large language models (LLMs), Stable Diffusion for image generation, and text-to-speech (TTS) engines** further promotes GenVox's versatility. The application also features a **scalable architecture** that allows for **future integration of AI-generated models** for continued enhancement. GenVox not only provides ease of use in everyday tasks, but fosters **creativity and automates tasks** and thus, positions itself as a **versatile, intelligent AI assistant**. The development of GenVox presents the possibility through **multimodal AI-supported systems** can transform the **human-computer interaction** withmore complex and contextualized responses.

Fig:[2] Access mode page

Fig:[3] Sign up page

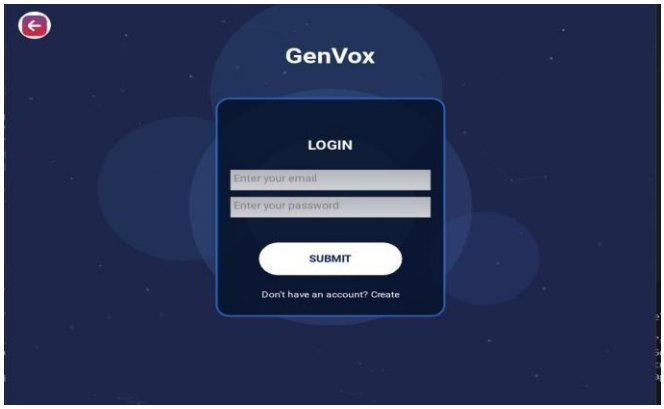


Fig:[4] Sign in page

Fig:[5] Model selection page

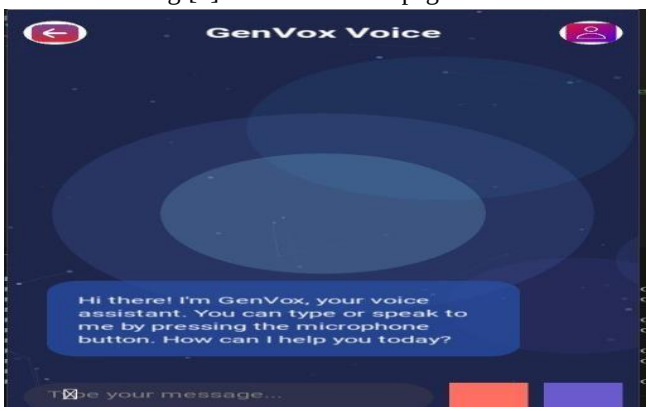


Fig:[6] Voice assistant Page

Fig:[7] Chatbot page

Fig:[8] Menu page

Fig:[9] Image Generation page

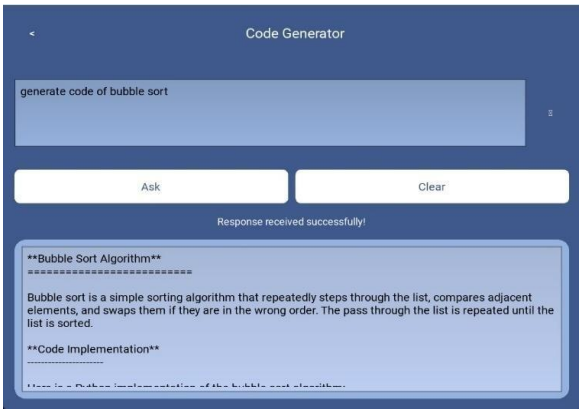
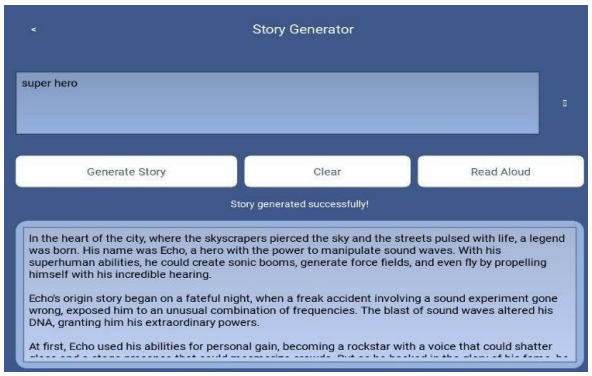


Fig:[10] Code Generation Page

Fig:[11] Poetry Generation Page



Fig[:12] Story Generation Page

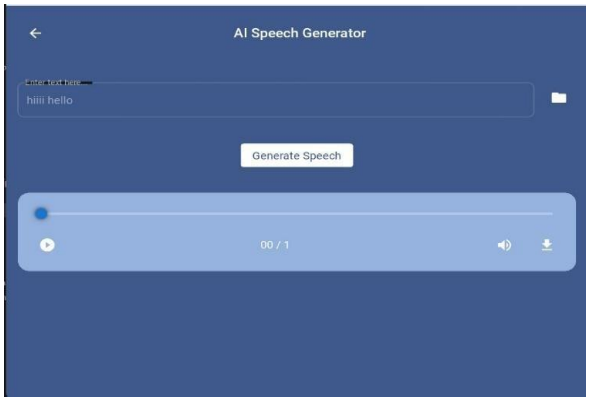


Fig:[13] Text-to-Speech Generation Page

Fig:[14] Language Translator Page

Fig:[15] Text Summarization Page

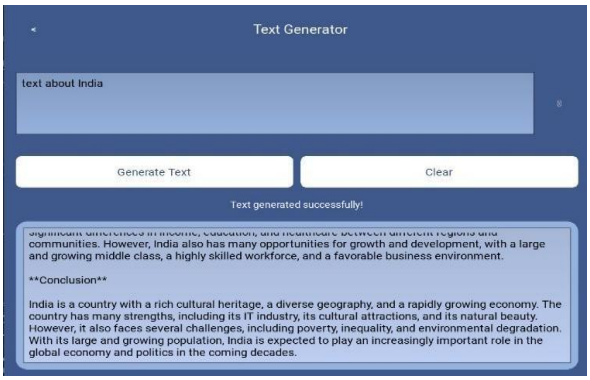


Fig:[16] Text Generation Page

Fig:[17] Problem solver page

CONCLUSION

GenVox employs multimodal generative AI for a next level re-imagined user experience around speech recognition, content generation, and task automation. The next generation of GenVox will include the following capabilities: - *Improved performance and AI*, including faster response times. - *Improved access for international audiences* with multilingual use. - *Improved security* with better encryption of data. - *Improved UI/UX*, providing greater engagement from the user. GenVox move forward as an overall AI-centered automation experience with seamless integration and represents the advancement of next generation voice enabled AI applications.

FUTURE SCOPE

These applications use different forms of AI-driven automation and integrations to create better seamlessness, user experience, access and efficiency in their respective use cases. The Voice Assistant application uses new speech to text processing capability to provide the user with hands free control to improve access and efficiency in the present use case. The Worker-Connect app reduces the burden of managing workers, with base capabilities such as worker registration, live location tracking and real-life service discovery, in order to improve productivity. The AI Capable Chatbot will have the capability to perform intelligent dynamic conversations using free LLMs, while using its context awareness, and thus also provide instant response capability for the system in real-time. The Code Gen Tool will provide code generation from natural language prompts, which improves development speed, including an API Key Automation for a seamless integration experience. Future work includes AI Optimization, Mult-lingual Capability, Security Improvements and UI/UX to make applications more robust and streamlined. Overall, these applications each use modern technology to reimagine automation, accessibility and intelligent real-time conversation for better smarter more adaptable AI enablement for all users.

REFERENCE

[1] Vaswani,A.,Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30, 5998-6008.

[2] (Transformer model, fundamental for LLMs like LLaMA-3 used in your project)

[3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems, 33, 1877-1901.

[4] (GPT-based models, relevant to your chatbot and text generation module)

[5] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10684-10695.

[6] (Stable Diffusion, related to your image generation module)

[7] Zhang, Y., & Chen, Q. (2023). *Multimodal Context Retention in Conversational AI*. Journal of Artificial Intelligence Research, 76, 45-78.

[8] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. International Conference on Machine Learning, 8748-8763.

[9] (CLIP-based text-to-image models used in Stable Diffusion)

[10] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". *Journal of Machine Learning Research*, 21(140), 1-67.

[11] Yu, J., Xu, Y., Koh, J. Y., Luong, T., Baid, G., Wang, Z., ... & Wu, Y. (2022). "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation". *arXiv preprint arXiv:2206.10789*.

[12] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). "Evaluating Large Language Models Trained on Code". *arXiv preprint arXiv:2107.03374*.

[13] Shazeer, N., & Stern, M. (2018). "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". *International Conference on Machine Learning*, 80, 4596-4604.

[14] Fedus, W., Zoph, B., & Shazeer, N. (2022). "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". *Journal of Machine Learning Research*, 23(120), 1-39.

[15] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). "Scaling Laws for Neural Language Models". *arXiv preprint arXiv:2001.08361*.

[16] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). "Training Compute-Optimal Large Language Models". *arXiv preprint arXiv:2203.15556*.

[17] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). "LLaMA: Open and Efficient Foundation Language Models". *arXiv preprint arXiv:2302.13971*.

[18] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2022). "PaLM: Scaling to 540 Billion Parameters". *arXiv preprint arXiv:2204.02311*.