AI-POWERED AVATAR CHATBOT FOR DIGITAL INTERACTION

A PROJECT REPORT

Submitted by

UMAR AHMED KHAN A	**(710021104032)**
RAGAVAN K	**(710021104305)**
TAMILVANAN VM	**(710021104314)**
SRIMAN YAGHAV C	**(710021104319)**

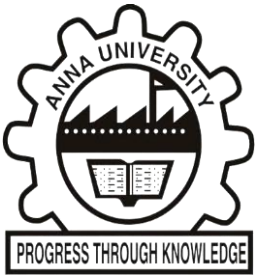
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2025

ANNA UNIVERSITY REGIONAL CAMPUS COIMBATORE

BONAFIDE CERTIFICATE

Certified that this project report " **AI-POWERED AVATAR CHATBOT FOR DIGITAL INTERACTION** " is the bonafide work of " **UMAR AHMED KHAN A (710021104032), RAGAVAN K (710021104305), TAMILVANAN VM (710021104314)**, **SRIMAN YAGHAV C (710021104319)** " who carried out the project work under my supervision.

SIGNATURE
Dr. S. V. MANISEKARAN M.E., Ph.D.
HEAD OF THE DEPARTMENT
Department of Computer Science and Engineering,
Anna University Regional Campus, Coimbatore.

SIGNATURE
Dr. N. FAREENA M.E., Ph.D.,
SUPERVISOR
Department of Computer Science and Engineering,
Anna University Regional Campus, Coimbatore

Submitted for the project viva voce examination held at ANNA UNIVERSITY REGIONAL CAMPUS, COIMBATORE on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGMENT

We would like to express our sincere gratitude and appreciation to all those who have contributed towards the successful completion of our project.

We would like to thank **Dr. M. SARAVANAKUMAR Ph.D**., Dean of Anna University Regional Campus, Coimbatore, for his kind patronage and support throughout the project.

Our heartfelt thanks to **Dr. S. V. MANISEKARAN, M.E., Ph.D**., Head of the Department, Computer Science and Engineering, who guided us with care with his constant encouragement and guidance towards the completion of this project.

We also like to extend our sincere thanks to our Project Coordinator and our class advisor who helped us with thoughtful advice, **Dr. P. MARIKKANNU, M.Tech., Ph.D**., Assistant Professor, Computer Science and Engineering, for his valuable guidance, supervision, throughout the project.

We would like to thank our project guide **Dr. N. FAREENA M.E., Ph.D.,** Assistant Professor, Computer Science and Engineering, for her constant support and guidance towards the completion of the project.

We would like to express our gratitude to all the teaching and non-teaching staff members of our department for their unwavering support and cooperation in completing the project.

Lastly, we thank our parents and friends for their endless encouragement and support, which has been invaluable to our success.

ABSTRACT

This project centralised on the design and implementation of an AI-powered avatar chatbot for digital interaction, aiming to simulate real-time, human-like communication through voice, animations, and expressive facial gestures. The core objective is to deliver a natural and responsive conversational experience by unifying multiple open-source technologies into a single web-based system. User inputs are met with contextual text responses, realistic synthesized voice output, and lip-synced avatar animations. Key integrations include the Google Gemini API for text generation, Coqui TTS for speech synthesis, and Rhubarb Lip Sync for phoneme-based mouth movement mapping. Emotion-driven animations such as laughing, crying, and dancing are dynamically triggered based on the conversation's emotional context, enhancing user engagement.

The system’s backend is built using Node.js and FastAPI, ensuring smooth coordination between AI responses and multimedia output. Coqui TTS runs locally to reduce dependency on commercial services like ElevenLabs or OpenAI’s voice tools. FFmpeg is used for audio format standardization, while Rhubarb generates lip-sync JSON data to animate the avatar's mouth and expressions. The frontend uses modern JavaScript frameworks to render the interactive 3D avatar, providing users with a rich and immersive interface. Unlike traditional chatbots, this project introduces a lifelike virtual entity that merges voice AI, intelligent conversation, and character animation into a unified, cost-effective framework. Its modular design allows for future upgrades, such as multi-language support, face tracking, or voice input through the browser positioning the system as a viable solution for education, assistance, or entertainment without reliance on proprietary platforms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

API	Application Programming Interface
DOM	Document Object Model
FBX	Filmbox
FFmpeg	Fast Forward Media Encoding
GLB	GL Transmission Format (Binary)
JSON	JavaScript Object Notation
LLM	Large Language Model
OBJ	Object File
TTS	Text-to-Speech
VITS	Variational Inference Text-to-Speech
WAV	Waveform Audio File Format

CHAPTER 1
INTRODUCTION

1.1 OBJECTIVE OF THE PROJECT

The primary objective of this project is to design and implement an AI-Powered Avatar Chatbot for Digital Interaction that closely mimics human-like interaction through the integration of speech synthesis, natural language processing, and facial animation. The system is developed with a focus on accessibility, responsiveness, and realism utilizing only open-source frameworks to avoid reliance on commercial or paid APIs.

By harnessing recent advancements in artificial intelligence and voice technology, the project aims to demonstrate a seamless fusion of text-based communication**,** real-time voice output, and lip-synced avatar animation, all within a web-based interface.

One of the key objectives is to employ Google Gemini API for generating intelligent textual responses. Gemini serves as the core conversational engine, allowing the assistant to respond in a coherent, helpful, and human-like manner.

These text responses are then converted into speech using Coqui TTS, a high-quality open-source text-to-speech library that supports multiple voices and languages. This audio output is further processed using FFmpeg to ensure compatibility, and lip synchronization is achieved using Rhubarb Lip Sync, which generates JSON-based phoneme timing data aligned with the spoken audio. This data drives the facial animations of a 3D avatar, making the assistant appear to "speak" in real-time.

Additionally, the virtual assistant is programmed to react not just with voice but also with gesture-based animations and facial expressions such as smiling, laughing, crying, or showing surprise. These animations are triggered dynamically based on the emotional tone or keywords in the response, creating a more immersive user experience.

The entire project is built using open-source technologies including FastAPI (Python), Node.js (JavaScript), and modern frontend tools to ensure that the system remains fully functional without incurring licensing costs. This makes it highly suitable for educational, assistive, and personal applications, offering a highly engaging and accessible form of human-computer interaction.

Ultimately, the goal is to push the boundaries of digital assistants by combining voice AI, natural language understanding, and animated avatars into a single cohesive system.

1.2 PROBLEM STATEMENT

Despite the advancements in AI-powered digital assistants, most existing systems rely heavily on cloud-based infrastructure and offer limited visual or expressive interaction. These systems often lack customization, transparency, and offline functionality, making them less adaptable for certain user scenarios such as educational tools or assistive technologies.

Additionally, current chatbot implementations rarely integrate real-time speech synthesis, accurate lip-syncing, and dynamic avatar animations into a unified platform.

This project addresses the need for a self-contained, AI-powered avatar chatbot that combines natural language understanding, expressive speech output, and lifelike avatar animation to create a more natural and engaging form of human-computer interaction.

1.3 SCOPE OF THE PROJECT

The scope of this project extends to the development and deployment of a real-time, AI-based virtual assistant chatbot capable of facilitating seamless human-computer interaction using advanced natural language generation, speech synthesis, and facial animation.

Designed to be platform-independent and cost-effective, the system operates entirely on open-source tools and frameworks, ensuring wide accessibility and ease of integration into various real-world scenarios. The chatbot can receive user input via text and respond in real-time through expressive voice output and synchronized avatar gestures, simulating human-like interaction.

This chatbot system has practical applications across multiple domains. In the education sector, it can serve as a personalized virtual tutor that explains technical topics, provides summaries, or assists with queries in natural language making learning more engaging and accessible. On websites or public kiosks, the assistant can act as a virtual guide or concierge, welcoming users and providing directions or information based on queries. Within software and technical support environments, it can function as a front-line helpdesk assistant, answering common technical questions, providing documentation links, or simulating guided troubleshooting. Additionally, the system can be integrated into personal productivity setups as a virtual assistant that helps manage tasks, remind users of schedules, or automate responses in various applications.

CHAPTER 2
LITERATURE SURVEY

2.1 OBJECTIVE

Advancements in AI have enabled chatbots to deliver more natural interactions through NLP, speech synthesis, and real-time facial animation. Recent literature explores the integration of open-source tools like Google Gemini for text generation, Coqui TTS for voice synthesis, and Rhubarb for phoneme-based lip sync, supported by audio processing with FFmpeg.

These technologies collectively aim to enhance the realism and responsiveness of digital assistants. Studies also emphasize the importance of modular, open-source systems that allow flexible development without relying on commercial APIs. This survey provides technical insights into building a unified, animated voice chatbot capable of real-time interaction within a browser-based environment.

2.2 LITERATURE SURVEY

Text-to-Speech (TTS) systems have significantly evolved from rule-based synthesis to advanced deep learning architectures capable of producing highly natural speech. Early open-source tools like Festival [1] laid the groundwork for speech synthesis research by providing a flexible platform based on linguistic and waveform synthesis rules. However, their audio quality was limited compared to modern deep learning models.

With the rise of deep generative models, WaveNet [2] revolutionized the TTS landscape. Introduced by DeepMind, WaveNet generates raw audio waveforms using autoregressive techniques and demonstrated substantial improvements in naturalness over concatenative and parametric methods. This led to the development of neural TTS pipelines such as Tacotron [3], which generates mel-spectrograms from text, and

Tacotron 2 [4], which combines Tacotron with WaveNet to improve expressiveness and fidelity.

In pursuit of real-time and robust solutions, FastSpeech [5] and FastSpeech 2 [6] introduced non-autoregressive models for faster and more stable synthesis while maintaining quality. These models enabled scalable TTS applications by decoupling duration modeling and spectrogram generation. Complementary to these, Diff-TTS [7] introduced a denoising diffusion probabilistic model to refine synthesis further, offering high-quality audio without adversarial training.

Conditional generative modeling also found its place in TTS research. Kim et al. [8] presented a Conditional Variational Autoencoder (CVAE) combined with adversarial learning for end-to-end TTS, integrating expressiveness and diversity into voice synthesis. Similarly, ESPnet-TTS [9] provided a unified, fully end-to-end speech synthesis framework based on PyTorch, supporting multiple backends and promoting reproducibility in academic and industrial settings.

Alongside acoustic modeling, data quality plays a vital role. The Noisy Speech Database by Valentini-Botinhao et al. [10] has become a benchmark for evaluating speech enhancement algorithms and robustness in TTS systems, aiding research in adverse acoustic conditions.

On the software and deployment side, Mozilla TTS [11] has contributed significantly to the open-source community by offering a powerful deep learning toolkit for TTS, supporting models like Tacotron and Glow-TTS, and making high-quality voice synthesis more accessible.

Parallel to the development of TTS systems, conversational agents and language models have advanced rapidly. Rasa [12], an open-source framework for natural language understanding and dialogue management, provides modular tools for intent classification, entity recognition, and dialogue flow. Meanwhile, Facebook's BlenderBot [13] focuses on open-domain conversational modeling using large-scale pretraining.

The integration of large language models like GPT-3 [14] has also influenced conversational TTS systems, enabling few-shot learning and more context-aware dialogue responses. These developments pave the way for emotionally intelligent and responsive virtual assistants.

2.3 CONCLUSION

The review supports the viability of building a responsive, expressive, and cost-effective chatbot system using open-source libraries. Unlike systems that depend heavily on cloud-based solutions, this project emphasizes offline/local hosting, which is ideal for students, researchers, or institutions aiming for data privacy and customization.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Most existing chatbot architectures are designed to support either text-based interaction or voice-based communication, but lack synchronized visual representation. Popular voice assistants embedded in mobile and smart devices often rely on centralized cloud-based infrastructure. These systems do not provide access to low-level components such as lip synchronization or avatar animation, and their closed-source nature limits extensibility, transparency, and adaptability for research purposes or offline deployment.

Several open-source conversational frameworks, including Rasa and Botpress, are effective for natural language understanding and dialogue management. However, they are primarily optimized for text-based communication and do not offer built-in support for real-time text-to-speech synthesis, voice output, or three-dimensional avatar integration.

Although tools like Mozilla TTS and Coqui TTS support high-quality voice synthesis in real-time, integration with animated avatars and lip-sync engines is rarely implemented in a modular or end-to-end fashion. In most cases, the speech output is generated in static batches and lacks synchronization with dynamic avatar rendering, which limits the effectiveness of real-time interaction.

Furthermore, systems that attempt to synchronize voice with avatar animation in real time often require high-performance computing resources such as dedicated graphical processing units or server-grade hardware. This dependence on advanced hardware restricts their use in local deployment scenarios, especially on consumer-grade devices or in environments with limited computational resources.

3.2 DISADVANTAGES OF EXISTING SYSTEM

Many existing systems are either heavily dependent on cloud infrastructure or rely on closed-source application programming interfaces, limiting their flexibility and increasing operational costs. This dependence introduces latency due to network communication and poses concerns regarding data privacy and control, especially in offline or sensitive environments.

Most commercial solutions offer limited customization, restricting developers from modifying response formats, adjusting animations, or integrating personalized voice models. This lack of adaptability hinders innovation and restricts the system's suitability for specific use cases.

Another major limitation is the absence of real-time lip synchronization in open-source platforms. In many cases, lip-sync functionality is either missing entirely or generated in offline batches, resulting in unnatural interactions. The integration between text-to-speech engines and avatar animation is often complex, poorly documented, and non-standardized.

As a result, developers are left with fragmented systems that are difficult to debug, scale, or optimize. Additionally, many platforms focus exclusively on text-based communication, lacking voice output or animated visual representation. This reduces the overall engagement, interactivity, and accessibility of the system for diverse users.

3.3 PROPOSED SYSTEM

The proposed system is a fully open-source, AI-powered animated chatbot capable of real-time response generation, voice output, and synchronized avatar animation. It uses Google's Gemini API for natural conversation generation and Coqui TTS for fast, high-quality speech synthesis. Rhubarb Lip Sync is integrated with FFmpeg to generate phoneme-timed animation data that synchronizes with the avatar's mouth movement.

The chatbot also includes facial expressions and gesture-based animations that reflect the emotional context of the message.

The backend is implemented using Node.js (for managing Gemini and frontend logic) and Python's FastAPI (for serving Coqui TTS). This modular design ensures parallelized processing of requests. Unlike commercial systems, this project runs entirely on a local server, offering offline access, customization, and full control over the chatbot behavior, voice model, and UI design.

3.4 ADVANTAGES OF PROPOSED SYSTEM

Open Source and Free from Licensing Fees

The proposed system leverages entirely open-source technologies, including Coqui TTS, Google's Gemini API, Rhubarb Lip Sync, Node.js, and FastAPI. By avoiding reliance on commercial or proprietary APIs such as ElevenLabs or OpenAI, the system remains free from licensing costs. This approach enhances accessibility and sustainability, making it suitable for developers and organizations seeking cost-effective AI-powered chatbot solutions.

Real-Time Multimodal Interaction

The assistant delivers responses not only through text but also via real-time speech synthesis and lip-synchronized avatar animations. This integration facilitates a realistic and immersive user experience, significantly improving engagement compared to systems limited to text or disjointed voice outputs. The synchronization of speech and visual cues ensures natural interaction, which is crucial for effective communication in conversational AI.

Modular and Scalable Design

The system's architecture distinctly separates text-to-speech processing, managed by FastAPI, from chatbot control logic, implemented in Node.js.

This modular design enhances maintainability and scalability, allowing independent development and optimization of components. Moreover, the design is adaptable for future enhancements, such as incorporating voice cloning or webcam-based emotion detection, without major architectural overhauls.

Privacy-Friendly and Offline Capable

All processing occurs locally on the user's device, ensuring that sensitive data is not transmitted to third-party servers. This local deployment model directly addresses privacy concerns, particularly relevant in domains such as education and healthcare where data security is paramount. Offline capability also enables the system to function without internet connectivity, expanding its usability in resource-constrained or secure environments**.**

Customizable Animation Pipeline

The system provides extensive flexibility in customizing avatar animations and voice synthesis. Developers can modify facial expressions, gestures, and voice models without restrictions, facilitating tailored implementations that meet domain-specific requirements. This adaptability is valuable for diverse applications ranging from educational tools and personal assistants to kiosk interfaces and virtual receptionists, allowing each deployment to reflect its unique user experience goals.

3.5 SUMMARY

The proposed AI-powered avatar chatbot integrates free, open-source tools to deliver real-time, multimodal interaction. By combining Gemini's contextual responses, Coqui's speech synthesis, and Rhubarb's lip sync, it offers a flexible, cost-free, and visually engaging alternative to traditional chatbots ideal for developers, researchers, and educators seeking a transparent and customizable AI solution.

CHAPTER 4
SYSTEM SPECIFICATION

4.1 SYSTEM REQUIREMENTS

Establishing accurate system requirements is critical for the development of an AI-powered avatar chatbot that integrates voice synthesis, lip sync, and animation. These requirements ensure optimal performance, fast response time, and compatibility with diverse user systems. The system is designed to function offline, without the need for paid APIs or external services, making efficiency and accessibility vital.

Natural Language Processing and Generation

The system uses Google's Gemini API for generating context-aware chatbot responses. The requirement for this module includes a stable internet connection and an API key, as Gemini is cloud-based. The system sends structured prompts and expects structured JSON replies for downstream rendering.

Real-Time Voice Synthesis

Coqui TTS is employed to convert textual responses into natural speech in real time. As this component runs locally using PyTorch models, sufficient memory and processor power are necessary to meet timing expectations and maintain clarity in audio responses.

Avatar Animation & Lip Sync

Rhubarb Lip Sync converts WAV audio into phoneme-based JSON files used for animating a 2D or 3D avatar. FFmpeg is required for preprocessing audio files into standard formats. The front-end consumes both audio and lip sync data to animate a virtual assistant that mimics human speech gestures.

Frontend Experience

The chatbot's interface is built with modern web technologies, providing intuitive UX with features like text input, animated avatar, and audio playback. Requirements include a modern browser (Chrome, Edge, Firefox) that supports WebGL and audio streaming.

Privacy and Offline Access

A key system objective is complete control and transparency. All voice and chat data are processed locally, ensuring user privacy and independence from cloud-based TTS or animation services.

4.2 HARDWARE REQUIREMENTS

To run the avatar chatbot smoothly, the following hardware specifications are recommended:

- **Processor:** Intel Core i5 or higher (2.0 GHz quad-core minimum)
- **Memory:** 8 GB DDR4 RAM (16 GB recommended for faster processing)
- **Graphics:** Integrated or dedicated GPU (optional but helps for 3D rendering and faster TTS inference)
- **Storage:** Minimum 2 GB free space for model weights, cache, and assets Operating System: Windows 10/11 (64-bit), Fedora Linux (or Ubuntu 20.04+)
- **Audio:** Speaker or headphone output for synthesized voice playback

Although GPU support is optional, having it enables acceleration of TTS processing (for users with compatible CUDA hardware).

4.3 SOFTWARE REQUIREMENTS

4.3.1 Platforms

The project is compatible with both Windows and Linux environments. Windows is primarily used for development and frontend deployment, while Linux (Fedora or Ubuntu) is suited for backend model serving using FastAPI and Python-based TTS frameworks.

4.3.2 Development Tools

Visual Studio Code: Used for editing frontend and backend code (JavaScript, Python, HTML/CSS).

Postman / Thunder Client: Utilized for testing REST APIs during development and debugging of request-response cycles.

Google Chrome: For running the browser-based chatbot interface and inspecting animation behavior.

4.3.3 Programming Languages

JavaScript (Node.js): Used for backend chatbot logic, API requests to Gemini, and animation event handling.

Python: Used for TTS processing and Rhubarb lip sync generation.

HTML, CSS, and Three.js: Used in the frontend for avatar rendering and UI/UX design.

4.3.4 Libraries and dependencies

The following packages are required for full functionality of the chatbot pipeline:

Python Libraries

Coqui TTS (TTS): Open-source text-to-speech engine that converts text into WAV files.

FastAPI: Lightweight Python web framework used to serve the TTS API.

FFmpeg: Used to convert and process audio formats for compatibility with lipsync tools.

Rhubarb Lip Sync: External binary used to generate JSON-based phoneme timelines from WAV input.

Node.js Modules

express: HTTP server to manage chatbot requests and structure conversation flow.

cors: Enables cross-origin access from frontend to backend services.

node-fetch: Fetches Gemini responses and handles HTTP requests.

child_process: Executes FFmpeg and Rhubarb binaries via shell commands.

Frontend Libraries

Three.js: For rendering 3D avatars and animating facial features and gestures.

Howler.js or HTML5 Audio: For playing the generated voice response in sync with lip sync data.

4.4 SUMMARY

The system specification outlines all hardware and software requirements to build, deploy, and run the AI-powered avatar chatbot. It focuses on integrating cutting-edge yet open technologies such as Gemini, Coqui TTS, and Rhubarb Lip Sync into a real-time voice-enabled interface. Using FastAPI and Node.js in combination ensures performance and modularity, while the front-end delivers a seamless visual experience.

CHAPTER 5
SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The architecture of the AI-Powered Talking Avatar Chatbot is designed to enable seamless real-time interaction between users and a virtual assistant capable of understanding input, generating contextual responses, and presenting them via speech and synchronized avatar animation. The system integrates several technologies, each responsible for a key layer of the user experience from natural language understanding and speech synthesis to animation rendering and user interface delivery.

The workflow begins with a user sending a text message through the frontend chat interface. This message is sent to the backend via an API, where the Node.js server processes it and forwards it to the Google Gemini API for contextual and intelligent response generation. The Gemini model returns a structured JSON array, which includes the response text, desired facial expression, and associated animation for the avatar.

The response text is passed to a FastAPI-based Python service that uses Coqui TTS to convert it into speech. The resulting WAV audio file is post-processed using FFmpeg to ensure it is compatible with lip-sync analysis tools. Rhubarb Lip Sync, a phoneme-based tool, then generates a JSON-based phonetic timeline from the audio, which is used by the frontend to animate the avatar's mouth in sync with the generated voice.

This architecture not only ensures modularity (by isolating chat logic, TTS processing, and animation) but also maximizes response speed and realism without reliance on paid services like ElevenLabs or OpenAI. All processing happens locally or through free cloud-based APIs, making the system highly scalable and cost-efficient.

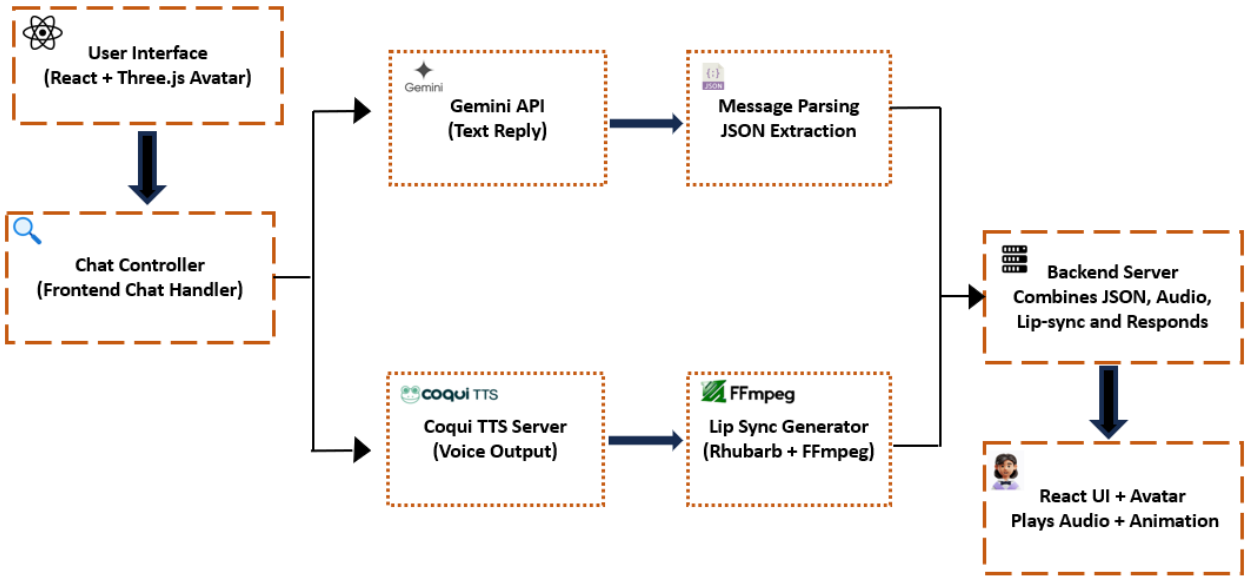


Fig. 5.1 Architecture Diagram

The Figure. 5.1 illustrates the complete architecture of the AI-powered chatbot avatar system, detailing the flow of data from user input to final animated output.

User Interaction

Users interact with the chatbot through a web interface built using HTML, CSS, JavaScript, and Three.js. Input is typed into a chat box and submitted via REST API to the Node.js server.

Text Response Generation

The text input is sent to the Gemini API via a secure HTTP POST request. The Gemini model processes the conversation context and user message, generating a response in a structured JSON format. Each message includes not only textual output but also emotional context and animation cues.

Voice Generation

The response text is forwarded to the FastAPI Python backend. Coqui TTS is used to synthesize the text into high-quality audio. The output is saved as a .wav file.

Audio post-processing

The audio is cleaned and standardized using FFmpeg to ensure it meets the requirements for further phoneme extraction. This step also adjusts sampling rate and audio channels.

Lip Sync Generation

The cleaned audio is passed to Rhubarb Lip Sync, which analyzes it and produces a phoneme-to-time JSON transcript. This data ensures the avatar's mouth movement aligns with the speech.

Frontend Synchronization

The audio and lip sync JSON, along with animation commands from the Gemini API, are sent back to the frontend. The avatar renders animations, facial expressions, and lip movements in sync with the spoken response, offering a human-like interaction.

5.2 COMPONENTS

User Interface (UI)

- Web-based chat interface built using HTML, CSS, and JavaScript
- Chat input box, avatar display area, and message log

3D/2D Avatar Renderer

- Visual representation of the avatar
- Uses Three.js for animations, facial expressions, and lip-syncing
- Responds to backend cues in real time

Chatbox Component

- Text input field for user queries
- Submit button and chat history display

Audio Player Component

- Handles playback of synthesized voice
- Syncs with avatar animation for a lifelike experience

Animation Controller

- Controls the avatar's gestures, facial expressions, and movements
- Receives animation and emotion cues from Gemini API

Status/Loading Indicator

- Shows when the bot is processing a response or generating speech

Settings Panel (Optional)

- Lets users change avatar appearance, voice type, or language
- Can be implemented with libraries like Leva GUI

5.3 ADVANTAGES OF SYSTEM DESIGN

Modular Architecture: Divides chatbot logic, voice processing, and animation layers for maintainability and scalability.

Open-Source & Cost-Free: All components are built using free and open-source libraries, avoiding licensing fees.

Real-Time Interaction: With optimized TTS models and lip sync, the system delivers responsive and engaging feedback.

Customizable Avatars & Voice Pipelines: Users can change animation sets, expression mappings, or TTS models as needed.

Data Privacy: No audio/text data is shared with third-party voice APIs, making it suitable for educational and private deployments.

5.4 MODULE DESCRIPTION

5.4.1 User Interface Module

The User Interface (UI) module is the primary point of interaction between the user and the AI-powered chatbot. It is designed to be web-based, intuitive, and engaging allowing users to type queries, receive real-time animated responses, and listen to AI-generated voice output. The interface includes a visually animated avatar that reacts with gestures, facial expressions, and lip-synced speech.

Interactivity: The chat interface is developed using modern web technologies such as HTML5, CSS3, JavaScript, and Three.js. It provides features like message input fields, animated avatar canvas, and playback of synchronized audio responses. The avatar reflects emotions like smiling, crying, or looking surprised based on the AI-generated output.

Customization: The UI supports theming and visual customization. Users can change the avatar appearance, background color, or even select different animation styles to suit different moods or use-cases.

Accessibility: Accessibility is incorporated through screen reader compatibility, keyboard navigation, and clear visual indicators. The chatbot can also be extended to include voice input in future versions, making it even more inclusive.

Cross-Platform Compatibility: The frontend is built to be responsive, supporting desktops, tablets, and mobile devices. It uses lightweight assets and adaptive rendering to ensure performance across devices with varying screen sizes and resolutions.

5.4.2 Response Generation & Animation Module

This module handles the core processing of user input, generating intelligent responses and animating them via audio, facial expressions, and avatar movement. It combines three critical processes: language generation (using Gemini), speech synthesis (using Coqui TTS), and lip sync animation (using Rhubarb).

Language Understanding and Reply Generation: When a user submits a message, it is sent to the backend Node.js server, which constructs a contextual conversation log and sends it to the Google Gemini API. Gemini generates a structured JSON array containing not only the AI's response text but also metadata for facial expression and animation.

Speech Synthesis: The response text is forwarded to a Python FastAPI service that runs the Coqui TTS engine. The engine synthesizes human-like speech in real time and returns the audio as a .wav file.

Lip Sync & Animation: Using FFmpeg, the audio is cleaned and standardized, after which Rhubarb Lip Sync processes the audio to generate a JSON timeline of phonemes. This timeline is sent back to the frontend, where the avatar's mouth moves in sync with the spoken words.

Integration and Flow: This module ensures seamless flow across all steps. From generating responses to voice playback and real-time animation, each output is synchronized before being sent back to the client. This results in a smooth, believable interaction that mimics real human communication.

5.4.3 Backend Module

The backend architecture of the AI-powered chatbot plays a crucial role in orchestrating the interaction between the user interface, AI language model, text-to-speech engine, and lip-sync generator. The system utilizes a hybrid backend framework composed of Node.js and Python's FastAPI, providing both responsiveness and scalability.

The Node.js server handles the primary routing of user messages, API integration with the Gemini language model, and the coordination of text, animation, and voice synthesis. It prepares structured prompts, parses AI responses, and passes the results to the TTS and lipsync modules. Node.js is chosen for its event-driven, non-blocking I/O model, ensuring real-time performance.

The FastAPI backend, running in Python, serves as a lightweight RESTful service that exposes endpoints for Coqui TTS. This modular backend receives text input, generates speech using the selected TTS model, and returns high-quality .wav audio to the Node.js layer. FastAPI is selected for its asynchronous capabilities and compatibility with Python-based ML libraries like PyTorch.

Together, this dual-backend setup allows the chatbot to deliver natural voice output, dynamic animations, and intelligent conversation flow, while keeping the architecture modular and maintainable.

5.5 WORKING MECHANISM

The working mechanism of the chatbot is designed to simulate human-like conversation with expressive, synchronized animations and real-time voice synthesis. The process is divided into multiple stages:

5.5.1 User Message Input

The user initiates a conversation through a text-based input on the web interface. The message is sent to the backend server (Node.js) via an API call.

5.5.2 Contextual Prompting and AI Response Generation

The Node.js backend formats the current conversation history and sends it to the Gemini API. Gemini generates a JSON-structured response including the AI's reply text, associated facial expression (e.g., smile, angry, sad), and animation type (e.g., Talking, Laughing, Rumba).

5.5.3 Text-to-Speech Generation

The generated response text is passed to the Coqui TTS service (hosted with FastAPI in Python), which converts the text into a natural-sounding audio file (.wav). This voice is aligned with the avatar's speaking behavior.

5.5.4 Lip Sync Generation

The output .wav file is processed using Rhubarb Lip Sync, which analyzes the phonemes in the speech and returns a JSON file with a frame-by-frame breakdown of mouth movements. This data is sent back to the frontend for synchronized animation playback.

5.5.5 Avatar Rendering and Playback

The frontend combines the response text, base64-encoded audio, lipsync JSON, and avatar animation into a single rendered scene. The 3D avatar performs the corresponding gestures while speaking the response, giving users a life-like interaction.

5.5.6 Continuous Interaction and Logging

All interactions are logged in the session’s context memory, ensuring that the AI can reference previous messages in future responses. This improves coherence and realism in multi-turn conversations.

5.5.7 Efficiency and Optimization

To reduce delay, operations such as TTS generation and lipsync processing are optimized with pre-processed models, lightweight WAV formatting using FFmpeg, and minimal network overhead between Node.js and FastAPI. The result is a fast, immersive, and responsive virtual assistant experience.

5.6 DESCRIPTION ABOUT THE DATASETS

This project integrates three core technologies: Google Gemini API for natural language generation, Coqui TTS for real-time text-to-speech synthesis, and Rhubarb Lip Sync for phoneme-based lip synchronization. Each of these components either utilizes or is based on pre-existing datasets that enable accurate and human-like behavior

5.6.1 Coqui TTS (your_tts Model)

The Coqui your_tts model is a multilingual and multispeaker text-to-speech model trained on a variety of open-source datasets. These datasets provide high-quality audio recordings alongside their corresponding transcriptions in different languages.

Notable datasets include:

VCTK Corpus: A dataset consisting of recordings from 109 English speakers with different accents, commonly used to train speaker-aware and accent-tolerant TTS systems.

LJSpeech: Contains over 13,000 English audio clips recorded by a single female speaker, ideal for general-purpose English synthesis.

CSS10 Dataset: A collection of single-speaker datasets in 10 different languages, providing a foundation for multilingual speech synthesis.

LibriTTS: A large corpus of English read speech derived from audiobooks, featuring multiple speakers and natural prosody.

Common Voice by Mozilla: A multilingual, crowdsourced dataset containing voice samples from thousands of volunteers across the globe.

These datasets enable your_tts to generate highly natural and expressive speech in various voices and languages without the need for retraining. The model can synthesize new voices using speaker embeddings or samples.

5.6.2 Rhubarb Lip Sync – Audio Phoneme Extraction

Rhubarb Lip Sync does not need training datasets because it is a rule-based phoneme extractor. It uses a built-in phoneme prediction engine based on:

- CMU Pronouncing Dictionary (for English phoneme mapping),
- Audio signal processing for pitch and envelope analysis,
- Heuristic rules for phoneme segmentation from the waveform.

CHAPTER 6
IMPLEMENTATION AND RESULT

6.1 OVERALL PROJECT STRUCTURE

The implementation of the AI-powered virtual assistant was divided into three interconnected modules Chat Processing, Voice Synthesis, and Lip Sync Animation, each running on separate servers but working in real-time to create a seamless user interaction.

The Chat Processing Module is implemented using Node.js and Express. It handles incoming chat requests, sends the prompt to Google Gemini API, and receives a JSON response that contains the text reply, facial expression, and avatar animation instructions. It then passes the text to the TTS module for audio generation and triggers lip sync generation.

The Voice Synthesis Module is developed in Python using FastAPI and Coqui TTS. When a message text is received from the Node.js backend, this server synthesizes the audio using the multilingual your_tts model and returns the audio file back to the Node.js layer.

The Lip Sync Animation Module uses Rhubarb Lip Sync, invoked via command line in the backend. It processes the WAV file generated by Coqui TTS and converts it into JSON viseme data, which can be used to animate a 3D or 2D avatar’s mouth movements in real time.

```
AI-Powered Avatar Chatbot for Digital Interaction/
|
├── frontend/                        #Frontend files - HTML/JS/Three.js
|   ├── index.html
|   └── src
|   |   └── assets
|   |   └── components
|   |   └── hooks
|   └── Public
|       └── animation
|       └── models
|
├── backend/                        # Node.js backend
|   ├── index.js                    # Main chatbot orchestration logic
|   ├── audios/                     # TTS and Rhubarb output files
|   |   ├── message_0.wav
|   |   ├── message_0.json
|   |   └── ...
|   ├── bin/                        # Rhubarb Lip Sync executable
|   |   └── rhubarb.exe
|   ├── .env                        # Contains GEMINI_API_KEY
|   ├── package.json
|   └── README.md
|
├── COQUI_TTS_SERVER/               # Python FastAPI + Coqui TTS server
    ├── tts_server.py
    └── output.wav
```

Fig. 6.1 Overall project structure

The figure 6.1 presents the structured organization of the AI-powered avatar chatbot project. At the root level, the project is divided into distinct directories representing the backend logic, text-to-speech server, frontend interface, and auxiliary components.

- The **backend/** folder contains the index.js file, which is responsible for managing chatbot logic using Node.js. This includes handling user messages, making requests to the Gemini API for language generation, and invoking Rhubarb Lip Sync for animation data generation.

- The **tts_server/** directory hosts tts_server.py, which uses FastAPI to expose an HTTP endpoint that processes text input via Coqui TTS and returns synthesized speech as WAV output.
- The **frontend/** directory includes React components such as UI.jsx, WelcomeLoader.jsx, and App.jsx that build and manage the interactive user interface, render the 3D avatar using Three.js, and handle voice playback.
- The **audios/** folder stores dynamically generated audio responses (message_0.wav) along with associated phoneme data (.json) used for real-time lip synchronization.
- Finally, the **bin/** directory contains the executable binary rhubarb.exe, which is used to process WAV files into JSON-based lip-sync timelines.

6.2 TERMINAL LOGS

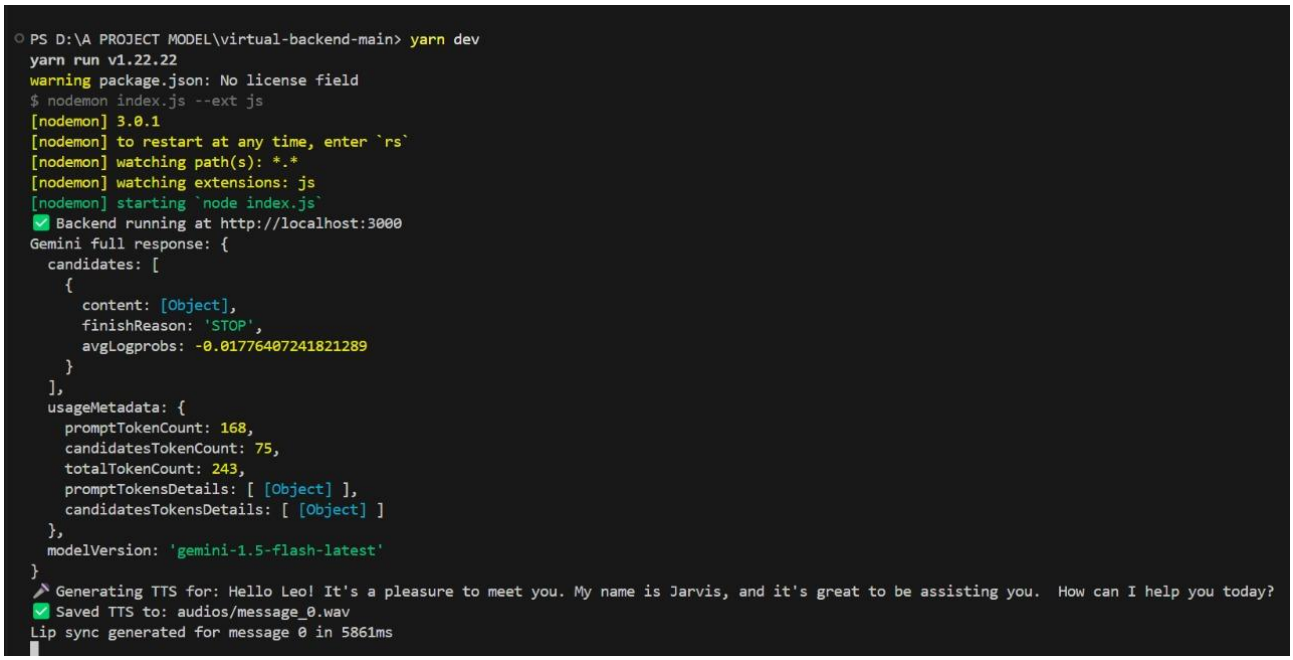


Fig. 6.2 Backend Server Running (Node.js Terminal)

The figure. 6.2 displays the terminal interface of the Node.js server actively running the backend of the chatbot system. This backend handles routing, manages communication with the Gemini API for generating conversational responses, and coordinates processes such as triggering FFmpeg and Rhubarb Lip Sync for audio and

animation synchronization. Successful startup messages and real-time logs indicate the server is actively listening for incoming requests, confirming proper backend deployment and operational readiness of the chatbot application.

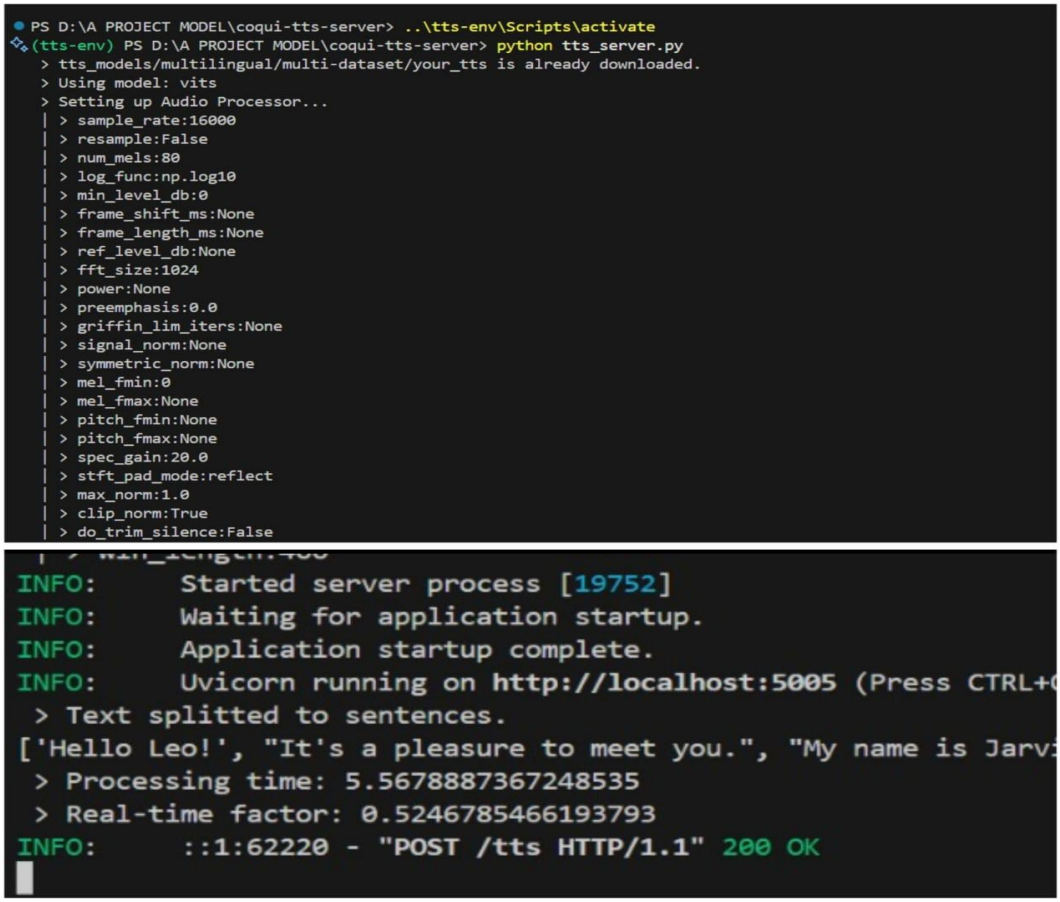


Fig. 6.3 FastAPI TTS Server Terminal (tts_server.py)

The figure. 6.3 shows the terminal output of the FastAPI-based text-to-speech (TTS) server, which is responsible for converting chatbot responses into synthesized speech. The tts_server.py script initializes the Coqui TTS engine and exposes it through a FastAPI endpoint. Upon successful launch, the terminal displays the server URL, confirming that the TTS service is accessible and ready to receive input text. Real-time logs are shown as the server processes TTS requests, enabling seamless audio generation for the avatar's voice output.

6.3 3D AVATAR DESIGN AND ANIMATION SETUP IN BLENDER

Fig. 6.4 Leva GUI Control Panel for Avatar Customization and Interaction

The figure. 6.4 shows Leva GUI control panel is integrated into the avatar dashboard interface to provide real-time customization and debugging features for developers. It offers a user-friendly set of sliders, toggles, and selectors that allow dynamic manipulation of avatar parameters without altering the code. Through this interface, developers can test voice playback behavior, trigger facial animations, adjust lighting and rendering settings, and monitor phoneme synchronization generated by Rhubarb. Additionally, the panel supports toggling debug modes, visualizing mouth shapes, and modifying environmental elements, making it a powerful tool for fine-tuning the 3D avatar's interaction and appearance during development.

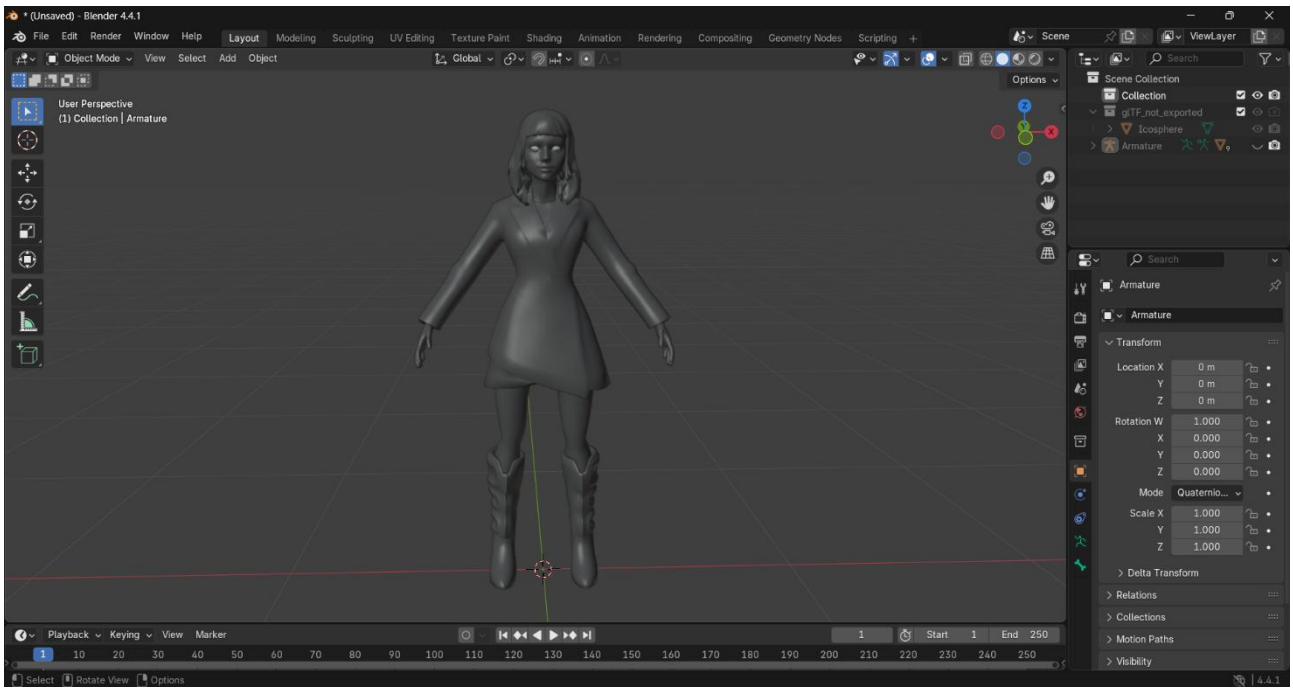


Fig. 6.5 Skeleton Rigging of 3D Avatar in Blender

The figure. 6.5 illustrates the skeleton rigging process of the 3D avatar in Blender. The armature is carefully structured with bones aligned to key joints of the model, enabling realistic animation and movement. Rigging ensures that facial and body gestures can be dynamically animated based on lip-sync and response data f rom the backend. This skeletal framework forms the foundation for integrating real-time voice-driven expressions in the chatbot interface.

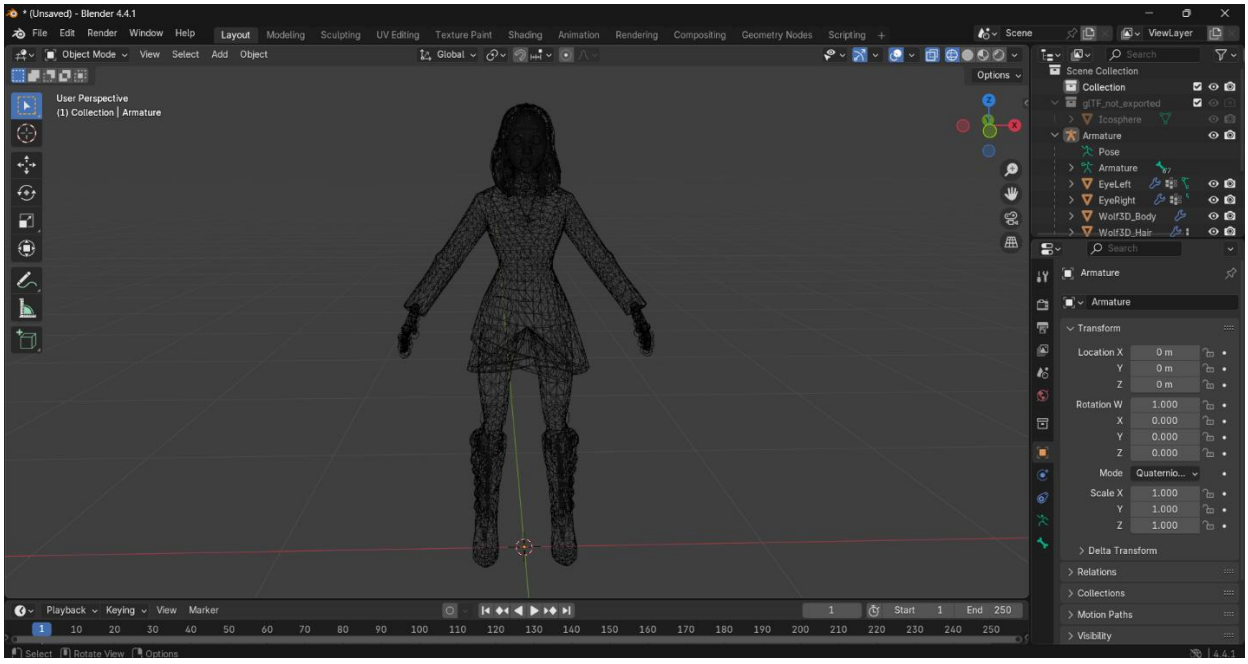


Fig. 6.6 Wireframe Screenshot of 3D Avatar in Blender

The figure. 6.6 shows the wireframe view of the 3D avatar model within the Blender interface. The wireframe representation reveals the mesh topology used to define the avatar's facial structure and overall geometry. This view is essential during modeling and rigging stages to ensure clean topology for smooth deformations, especially in lip sync and animation. The avatar's face is structured to support dynamic expression changes, allowing seamless synchronization with phoneme data generated during real-time conversations. This Blender setup forms the foundation for rendering and exporting the avatar into the web-based interface.

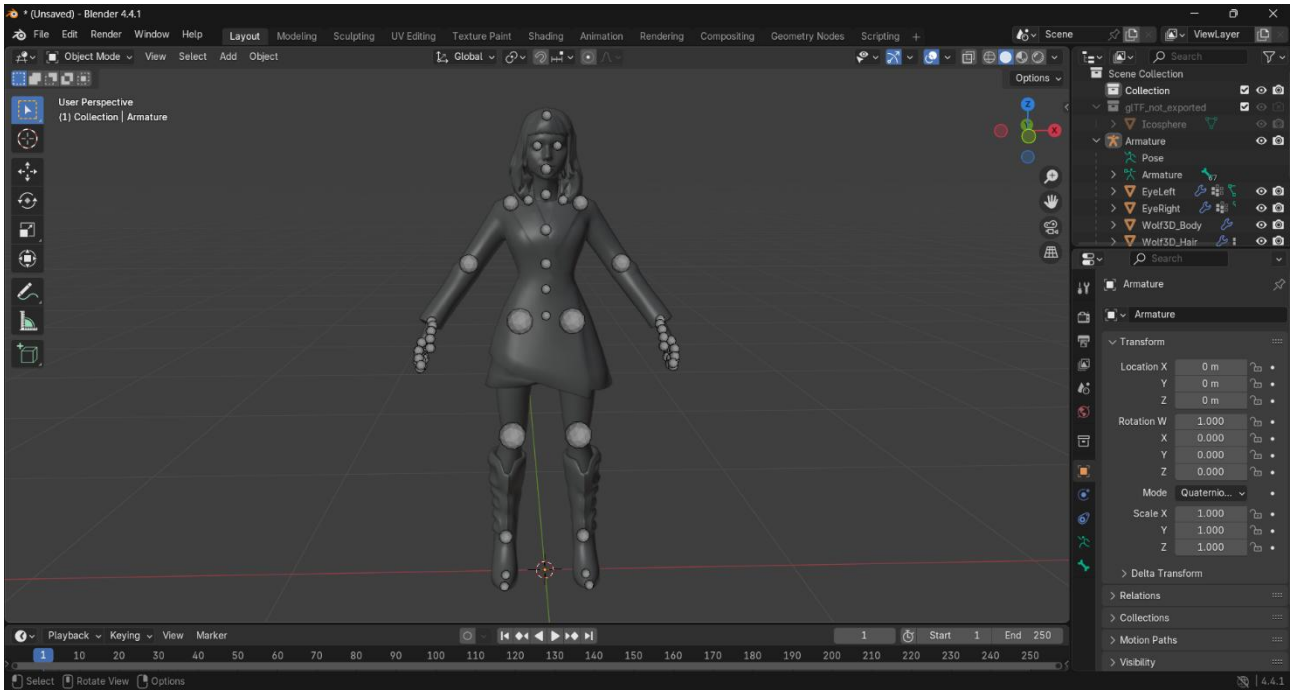


Fig. 6.7 Skeleton Setup for Avatar Movement in Blender

The figure. 6.7 presents the bone-based armature setup in Blender used to enable full-body movement for the AI avatar. Each major joint including the neck, shoulders, elbows, hips, knees, and ankles is rigged with corresponding bones to facilitate natural motion. The rig also includes control points for the hands and feet, allowing detailed pose manipulation. This rigging enables real-time animations synchronized with speech and responses, creating a more lifelike and responsive virtual assistant experience.

6.4 FINAL DASHBOARD

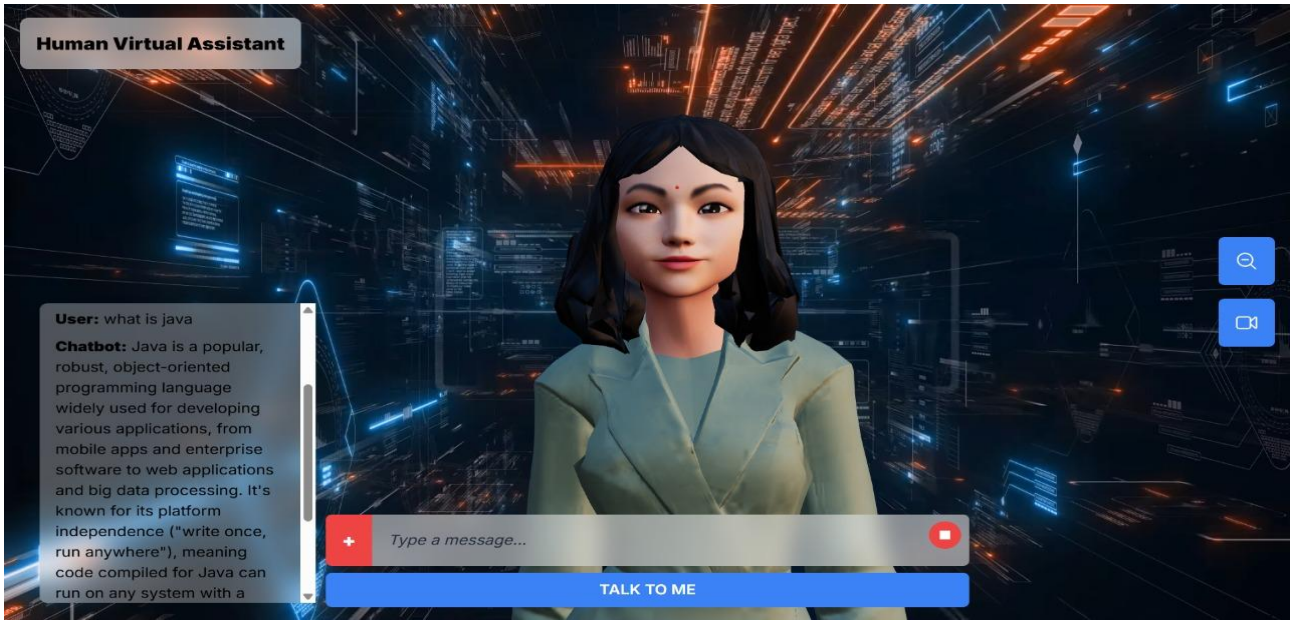


Fig. 6.8 Final Chatbot Output on UI

The figure. 6.8 presents the final integrated output of the AI-powered avatar chatbot on the browser-based user interface. It showcases the system's full functionality, where the chatbot generates a textual response using Google's Gemini API, synthesizes speech via Coqui TTS, and animates the avatar's facial expressions and lip movements using Three.js and Rhubarb Lip Sync. The avatar responds in real-time with synchronized audio and animation, delivering a multimodal and immersive interaction experience. This unified frontend demonstrates the effectiveness of the system’s pipeline from natural language understanding and speech synthesis to visual animation rendering.

CHAPTER 7
CONCLUSION

7.1 CONCLUSION

This project presents a fully functional, real-time, AI-powered virtual assistant chatbot that combines the power of Google Gemini API, Coqui TTS, and Rhubarb Lip Sync to simulate natural human interaction. The system is capable of generating intelligent text-based responses, converting them into natural-sounding voice output, and animating a 3D avatar with corresponding facial expressions and lip movements in real time. The backend, developed using Node.js and FastAPI, orchestrates the seamless integration between user input, AI response, speech synthesis, and avatar synchronization.

The successful deployment of this chatbot on a local system using only open-source tools demonstrates that high-quality virtual assistants can be created without reliance on commercial or paid APIs like OpenAI or ElevenLabs. The avatar reacts in sync with speech, expresses emotions (e.g., laugh, cry, angry), and executes animations (e.g., dance, idle), thereby enhancing engagement and realism. This system stands out in its capability to blend multi-modal interaction text, voice, and animation into a single pipeline while remaining cost-effective and customizable.

The developed assistant lays a strong foundation for further innovation in virtual communication, assistive AI, and educational tools. It serves not only as a technical demonstration but also as a functional prototype of how conversational AI can be personalized and deployed in real-world applications.

7.2 FUTURE ENHANCEMENT

In future iterations, several features can be added to improve the usability, scalability, and intelligence of the chatbot:

Voice Swapping / Cloning: Real-time voice customization could be implemented using speaker embedding models like your_tts or vits, allowing users to upload their voice and hear the assistant speak like them, enhancing personalization.

Contextual Memory: Integration of user context history and identity memory (e.g., remembering previous interactions or preferences) will improve long-term conversation quality and personalization.

WebSocket-Based Streaming TTS: To minimize latency and increase responsiveness, Coqui TTS can be upgraded to support real-time audio streaming using WebSockets instead of full request-response cycles.

Multi-language Support: Although English is the default, future versions could include dynamic language detection and translation using open-source NLP libraries and multilingual TTS pipelines.

Mobile App Integration: The web application can be extended to Android/iOS platforms to provide users with mobile access to the chatbot anytime, anywhere.

Educational & Assistive Deployment: This chatbot could be deployed as a digital tutor, museum guide, customer service assistant, or mental wellness coach in specific domains, each with tailored training data and expressions.

Emotion Detection from User Input: Integrating sentiment analysis from the user's message could trigger more emotionally aligned responses from the avatar (e.g., comforting tone when the user is sad).

REFERENCES

1. A. W. Black, P. Taylor, and R. Caley, The Festival Speech Synthesis System: System Documentation (1.4), University of Edinburgh, Centre for Speech Technology Research (CSTR), 1999.

2. A. van den Oord, S. Dieleman, H. Zen, et al., “WaveNet: A Generative Model for Raw Audio,” DeepMind, 2016.

3. Y. Wang, R. Skerry-Ryan, D. Stanton, et al., “Tacotron: Towards End-to-End Speech Synthesis,” arXiv preprint arXiv:1703.10135, 2017.

4. J. Shen, R. Pang, R. J. Weiss, et al., “Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions,” in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018.

5. Y. Ren, Y. Ruan, X. Tan, et al., “FastSpeech: Fast, Robust and Controllable Text to Speech,” in Advances in Neural Information Processing Systems (NeurIPS), 2019.

6. Y. Ren, C. Hu, X. Tan, et al., “FastSpeech 2: Fast and High-Quality End-to-End Text to Speech,” in Proceedings of the International Conference on Learning Representations (ICLR), 2021.

7. K. Li, Y. Li, D. Zhao, et al., “Diff-TTS: A Denoising Diffusion Model for Text-to-Speech,” arXiv preprint arXiv:2104.01409, 2021.

8. J. Kim, J. Kong, and J. Son, “Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech,” arXiv preprint arXiv:2106.06103, 2021.

9. T. Hayashi, R. Yamamoto, T. Koriyama, et al., “ESPnet-TTS: Fully End-to-End Text-to-Speech System Based on ESPnet,” in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020.

10. R. Valentini-Botinhao, C. Veaux, J. Yamagishi, and S. King, “Noisy Speech Database for Training Speech Enhancement Algorithms and TTS Models,” University of Edinburgh, 2017.

11. Mozilla Contributors, “Mozilla TTS: An Open-Source Deep Learning Toolkit for Text to Speech,” 2025.

12. T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, “Rasa: Open Source Language Understanding and Dialogue Management,” arXiv preprint arXiv:1712.05181, 2017.

13. Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Shuster, K., Smith, E. M., Boureau, Y.-L., & Weston, J. (2020). Recipes for building an open-domain chatbot. arXiv preprint arXiv:2004.13637.

14. T. Brown, B. Mann, N. Ryder, et al., “Language Models are Few-Shot Learners,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.