**Research Project**

**On**

**Ai Based Multi Audio Dubbing System**

**By**

**Tanmay Khodankar**

**Mohit Mohadikar**

**Pranav Nagmote**

**Sapana Jadhao**

**Under the Guidance of**

**Prof. Dr. Prakash Kene**

**Master In Computer Application**

**P.E.S’s Modern College Of Engineering,**

**PUNE – 411005.**

**(An Autonomous Institute Affiliated to Savitribai Phule Pune University)**

**2025-26**

A logo with text and fish

AI-generated content may be incorrect.Progressive Education Society's

**Modern College of Engineering**

(An Autonomous Institute Affiliated to Savitribai Phule Pune University)

**MCA Department**

**CERTIFICATE**

This is to certify that **Pranav Nagmote, Sapana Jadhao, Tanmay Khodankar, Mohit Mohadikar**

of Master in Computer Application have successfully completed the Research Project work titled ‘**Ai Based Multi Audio Dubbing System**’ during the academic year 2025-26. This report is submitted as partial fulfilment of the requirement of degree in MCA Engineering of Modern College of Engineering.

**External Examiner:**

**Dr. Mrs. K. R. Joshi Dr. Shivani A. Budhkar Dr. Prakash Kene**

**Principal Head of Department Project Guide**

**Acknowledgement**

We take this opportunity to express our sincere gratitude to all those who have guided and supported us throughout the development of our project, *“AI BASED MULTI AUDIO DUBBING SYSYTEM”*

We are deeply thankful to our guide - Prof. Dr. Prakash Kene, Department of Computer Applications, PES Modern College of Engineering, for their constant guidance, encouragement, and valuable insights that helped us shape this work.

We also extend our heartfelt thanks to the Head of Department, Dr. Shivani Budhkar, and all faculty members of the Department of Computer Applications for their continuous support and for providing us with the necessary facilities to carry out this project successfully.

Lastly, we express our deep gratitude to our families and friends for their patience, motivation, and unwavering support throughout this journey.

**Sign of Student/s**

**Pranav Nagmote[52146]**

**Tanmay Khodankar[52135]**

**Sapana Jadhao [52126]**

**Mohit Mohadikar [52144]**

**Abstract**

The *Auto Transcriber* is an intelligent web-based application designed to automatically convert spoken language into accurate and well-formatted text transcripts. This project aims to simplify the process of transcription for meetings, lectures, interviews, and media content by integrating advanced speech recognition and natural language processing (NLP) technologies.

Developed using modern frameworks and APIs for speech-to-text conversion, the system provides real-time audio analysis, punctuation restoration, and noise handling to ensure high transcription accuracy. It supports multiple audio formats and offers features such as text editing, timestamp insertion, and export options in standard file formats.

Built with a modular architecture, the Auto Transcriber demonstrates seamless integration of AI-driven models with an interactive and user-friendly interface. This project highlights key concepts in machine learning, speech processing, and web application development, providing an efficient and automated solution to replace manual transcription efforts.

**Keywords:** Speech Recognition, Audio Transcription, Natural Language Processing, AI, Automation, Web Application

**Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Title** | **Page No.** |
| 1. | Introduction and Aims / Motivation and Objectives | 1–2 |
| 2. | Literature Survey | 3–8 |
| 3. | Problem Statement / Definition | 9–11 |
| 4. | Software Requirement Specification (SRS) | 12–18 |
| 5. | Flowchart | 19–21 |
| 6. | Project Requirement Specification | 22–25 |
| 7. | Proposed System Architecture | 26–28 |
| 8. | High-Level Design of the Project | 29–31 |
| 9. | System Implementation | 32–36 |
| 10. | Test Cases | 37–40 |
| 11. | Proposed GUI (Graphical User Interface) | 41–44 |
| 12. | Project Plan | 45–46 |
| 13. | Conclusion | 47–49 |
| 14. | Bibliography / References | 50 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Figure Title** | **Page No.** |
| 1. | Flowchart of the Auto Transcriber System | 19 |
| 2. | Proposed System Architecture | 26 |
| 3. | Entity–Relationship (ER) Diagram | 27 |
| 4. | Data Flow Diagram (Level 0) | 28 |
| 5. | Data Flow Diagram (Level 1) | 29 |
| 6. | Class Diagram | 30 |
| 7. | Use Case Diagram | 31 |

**1. Introduction**

In today’s digital era, audio and video recordings have become integral to education, business meetings, interviews, and media production. However, manually converting these recordings into written text is a time-consuming and error-prone task. The *Auto Transcriber: Intelligent Audio-to-Text Conversion System* aims to solve this challenge by automating the transcription process using advanced speech recognition and natural language processing (NLP) technologies.

The system is designed to automatically process recorded or live audio input, convert speech into structured text, and present it in a readable format with punctuation, timestamps, and speaker identification (optional). By utilizing modern AI-driven models and APIs for speech-to-text processing, the tool ensures high accuracy even in noisy environments.

Developed as a web-based application, Auto Transcriber integrates an interactive and user-friendly interface that allows users to upload audio files, preview transcription results, make edits, and export the final transcript in multiple formats such as TXT, DOCX, or PDF. The system significantly reduces manual effort, boosts productivity, and ensures accessibility across devices.

This project demonstrates the application of artificial intelligence and web technologies to create a real-world solution that enhances automation, efficiency, and usability in digital transcription.

**Aim**

The aim of this project is to develop an intelligent and automated system capable of converting speech from audio recordings into accurate, structured, and readable text. The system seeks to provide a reliable transcription tool that minimizes human intervention while ensuring high accuracy and efficiency in various use cases such as lectures, interviews, and meetings.

**Motivation**

Manual transcription of audio content is labour-intensive, slow, and prone to human errors. With the rise of AI and speech recognition technologies, it has become possible to automate this process effectively. The motivation behind this project is to leverage machine learning and NLP advancements to build a tool that simplifies transcription, saves time, and improves accessibility for students, researchers, journalists, and professionals.

This project aims to provide an affordable, browser-based solution that eliminates dependency on expensive transcription software and offers a seamless experience with minimal technical requirements.

**1.2 Objectives And Scope**

**Primary Objectives**

1. Automated Speech Recognition: Convert audio input into text using AI-based speech recognition models.
2. Real-Time Transcription: Support live and recorded audio transcription with accurate time synchronization.
3. Text Formatting and Punctuation: Implement automatic sentence structuring and punctuation correction for readable transcripts.
4. User Interface: Develop a clean, responsive web interface for uploading audio, viewing transcripts, and exporting results.
5. Multi-Format Export: Enable users to download transcripts in multiple formats such as TXT, DOCX, and PDF.

**Secondary Objectives**

1. Language Support: Extend the system to handle multiple languages and accents.
2. Noise Handling: Improve accuracy in noisy environments using filtering and preprocessing.
3. Timestamp Insertion: Provide optional timestamps for paragraphs or sentences.
4. Speaker Differentiation: Identify and label different speakers within the audio file.
5. Cloud Integration: Allow users to save or sync their transcription history.

**Project Scope**

The project focuses on developing a web-based AI transcription platform that performs the following tasks:

* Accepts audio files of various formats (e.g., MP3, WAV).
* Converts audio speech into accurate text using speech recognition APIs or AI models.
* Provides text formatting with punctuation and timestamps.
* Offers an interactive editor for users to review and refine transcripts.
* Allows export of the final text file in multiple formats.
* Ensures cross-platform compatibility and responsive design for usability on both desktop and mobile devices.

**Out of Scope**

* Real-time voice-to-text integration in external applications.
* Offline transcription without internet or cloud APIs.
* Complex multi-user collaboration or role-based access control.
* Advanced analytics such as sentiment analysis or keyword extraction.
* Direct integration with third-party meeting software (e.g., Zoom, Teams).

**Literature Survey**

**2.1 Database Design Principles**

Database design is the process of producing a detailed data model of a database containing logical and physical design choices and physical storage parameters. According to Elmasri and Navathe (2015), the database design process consists of three main phases:

**2.1.1 Conceptual Design**

The conceptual design phase involves creating a high-level description of the database structure without considering implementation details. Entity-Relationship (ER) modelling, introduced by Peter Chen in 1976, has become the standard approach for conceptual database design.

The ER model provides three main constructs:

- Entities: Represent real-world objects or concepts

- Attributes: Describe properties of entities

- Relationships: Define associations between entities

**2.1.2 Logical Design**

During logical design, the conceptual model is transformed into a logical schema. This involves:

- Converting entities to relations/tables

- Converting attributes to columns

- Handling relationships through primary and foreign keys

- Normalization to eliminate data redundancy

**2.1.3 Physical Design**

Physical design determines how the logical schema will be implemented in a specific database management system. This includes:

- Data type selection

- Index creation

- Storage parameter optimization

- Performance tuning

**2.2 Existing Erd Tools**

Various tools and methodologies have been developed to support ERD creation and database design:

**2.2.1 Commercial Tools**

Microsoft Visio

- Widely used for creating ERDs and other diagrams

- Integrates well with Microsoft Office suite

- Supports various database notations (Crow's foot, UML, etc.)

- Limitations: Expensive licensing, general-purpose tool not specialized for databases

**ERwin Data Modeler**

- Comprehensive data modelling solution

- Advanced features for enterprise databases

- Supports reverse engineering from existing databases

- High cost makes it inaccessible for individual developers

**Lucidchart**

- Cloud-based diagramming tool

- Collaboration features

- Templates for various diagram types

- Subscription-based pricing model

**2.2.2 Open Source Alternatives**

**Draw.io (diagrams.net)**

- Free, web-based diagramming tool

- Large library of shapes and templates

- Supports real-time collaboration

- No specific ERD features or SQL generation

**MySQL Workbench**

- Official tool for MySQL database design

- ERD creation with forward engineering

- Direct database connectivity

- Limited to MySQL ecosystem

**2.2.3 Academic and Research Tools**

Several academic projects have explored automated ERD generation:

- Research by Silva et al. (2018): Automatic generation of ERDs from source code analysis

- NLP-based approaches: Using natural language processing to extract entities from requirements documents

- Ontology-driven modeling: Using semantic web technologies for conceptual modeling

**2.2.4 Web-based Solutions**

Recent developments have focused on web-based ERD tools:

- DBDiagram.io: Web-based database diagramming with SQL export

- QuickDBD: Simple web-based ERD creation tool

- SQLDBM: Cloud-based database design and modeling

These tools provide varying levels of automation but lack comprehensive features like automatic junction table creation and real-time SQL visualization.

**2.3 Web Technologies For Diagram Visualization**

Web technologies have evolved significantly, enabling complex interactive applications:

**2.3.1 HTML5 and Canvas API**

HTML5 introduced the Canvas API for dynamic graphics rendering. The Canvas API provides:

- Low-level graphics rendering capabilities

- Direct pixel manipulation

- Performance optimizations for animations

**2.3.2 SVG (Scalable Vector Graphics)**

SVG provides vector-based graphics with several advantages:

- Scalability without loss of quality

- DOM integration for interactivity

- Accessibility features

- Small file sizes for simple graphics

**2.3.3 WebGL and 3D Graphics**

WebGL enables hardware-accelerated 3D graphics in browsers, though it's often overkill for 2D diagramming applications.

**2.3.4 Javascript Libraries For Diagramming**

**ReactFlow**

- React-based library for creating node-based diagrams

- Declarative API matching React patterns

- Extensible through custom nodes and edges

- Active community and regular updates

**D3.js (Data-Driven Documents)**

- Powerful visualization library for web browsers

- Data binding and DOM manipulation

- Large ecosystem of visualization components

- Steeper learning curve

**Cytoscape.js**

- Specialized for complex network visualization

- Good performance with large graphs

- Layout algorithms for automatic graph arrangement

**2.4 React And Modern Web Frameworks**

**2.4.1 React Evolution**

React, developed by Facebook in 2013, has become the dominant library for web user interfaces:

Key Features:

- Component-based architecture

- Virtual DOM for performance optimization

- Declarative programming model

- Unidirectional data flow

2.4.2 Next.js Framework

Next.js provides additional capabilities on top of React:

- Server-side rendering (SSR)

- Static site generation (SSG)

- API routes for backend functionality

- File-based routing system

**2.4.3 Component Libraries**

**Radix UI**

- Accessible, customizable components

- Focus on functionality over styling

- Works well with Tailwind CSS

**Tailwind CSS**

- Utility-first CSS framework

- Responsive design utilities

- Minimal bundle size with purging

**Research Gap**

Despite significant progress in automatic speech recognition (ASR) and transcription technologies, several challenges persist that limit the efficiency and accuracy of real-time audio-to-text systems. Current transcription tools and APIs (such as Google Speech-to-Text, Whisper, and AWS Transcribe) have demonstrated impressive accuracy under controlled conditions but often fail to deliver consistent performance in real-world environments.

The following key **research gaps** have been identified:

**Noise and Accent Sensitivity**

Most existing transcription systems struggle with diverse accents, dialects, and noisy environments. The accuracy of transcription significantly drops in multi-speaker or background-noise conditions, highlighting the need for adaptive noise filtering and accent-aware models.

1. **Real-Time Performance Optimization**

Many open-source ASR models prioritize accuracy over speed. However, real-time transcription requires optimized inference pipelines that can deliver low-latency performance even on moderate hardware configurations.

1. **Language and Multilingual Limitations**

A majority of transcription systems are primarily trained for English or a few major languages. There is a gap in multi-language or code-switching (mixed-language) transcription, especially for Indian regional languages and accents.

1. **Domain Adaptation and Contextual Understanding**

Current models often misinterpret domain-specific terms (e.g., medical, legal, or academic vocabulary). There is a lack of models capable of integrating contextual or semantic understanding to improve accuracy in specialized domains.

1. **Offline and Privacy-Preserving Transcription**

Most existing systems depend on cloud APIs, raising privacy and connectivity concerns. There is limited research on **fully offline transcription models** that ensure **data security** and **local processing**, especially for sensitive applications.

1. **User Interface Integration and Accessibility**

Few transcription systems focus on usability and accessibility. The lack of intuitive interfaces, export options, and easy editing capabilities restricts adoption among non-technical users, educators, and professionals.

**Identified Research Need**

Hence, there is a clear need for a **lightweight, privacy-preserving, and user-friendly transcription system** capable of:

* Operating efficiently on local hardware,
* Supporting multiple audio formats and accents,
* Delivering near real-time transcription with high accuracy, and
* Allowing seamless text export and editing.

This project — **Auto Transcriber** — aims to bridge these research gaps by integrating **AI-based speech recognition**, **noise-resistant preprocessing**, and **React + FastAPI-based real-time UI**, providing a practical, accessible, and secure transcription solution.

**2.5 Summary Of Literature Review**

The literature review reveals several key insights:

1. ERD Tools: While numerous commercial and open-source tools exist, most have limitations in cost, complexity, or specialization

2. Web Technologies: Modern web technologies provide sufficient capabilities for creating sophisticated diagramming applications

3. ReactFlow: Offers an ideal foundation for building interactive ERD tools with its node-based architecture

4. Database Design Process: Well-established methodologies exist that can guide the development of automated tools

This project builds upon these foundations by creating a specialized, accessible ERD tool that bridges the gap between intuitive visual design and practical SQL implementation.

**3. Problem Statement / Definition**

**3.1 Problem Definition**

In today’s digital communication era, a large portion of information is transmitted verbally through meetings, lectures, interviews, podcasts, and videos. However, manually converting these audio conversations into written text is time-consuming, error-prone, and requires significant human effort.

Despite the availability of some transcription software, many existing tools are either expensive, limited to specific languages, or lack accuracy in handling diverse accents and noisy environments.

**Core Problem Areas:**

1. Manual Effort: Human transcription is slow, labor-intensive, and costly.
2. Limited Accessibility: Professional transcription software often requires paid licenses or cloud-based subscriptions.
3. Accuracy Issues: Existing models struggle with accent variations, background noise, and domain-specific vocabulary.
4. Language and Speaker Diversity: Many systems fail to support multi-language or speaker differentiation effectively.
5. Integration Challenges: Users find it difficult to integrate transcription tools with other applications or workflows.

**3.2 Need For Automated Transcription Systems**

**3.2.1 Efficiency Gains**

An AI-powered auto transcriber can dramatically reduce the time and effort required for audio-to-text conversion by:

* Real-Time Transcription: Converting spoken content into text instantly during meetings or recordings.
* Automation: Eliminating the need for manual transcription.
* Batch Processing: Supporting bulk uploads for fast transcription of multiple files.
* Speaker Segmentation: Identifying and labeling different speakers automatically.

**3.2.2 Cost Reduction**

* Open-Source or Custom Solution: Avoids expensive licensing fees of software.
* Reduced Manpower: Minimizes the need for dedicated human transcribers.
* Scalable Deployment: Can be integrated into existing systems or used standalone.

**3.2.3 Quality Improvement**

* High Accuracy: Leverages advanced deep learning models (e.g., Whisper, Wav2Vec, or SpeechRecognition API) for precise results.
* Noise Handling: Preprocessing filters remove background noise for cleaner text.
* Multi-Language Support: Enables transcription of multiple regional and international languages.
* Formatting and Punctuation: Intelligent models automatically insert punctuation, improving readability.

**3.3 Analysis Of Requirements**

**3.3.1 User Analysis**

**Primary Users:**

* Journalists, students, and researchers requiring fast transcription of interviews or lectures.
* Content creators needing subtitles or captions for videos.
* Professionals who want to convert meeting recordings into text notes.
* Educators and transcription services seeking automation tools.

**User Characteristics:**

* Varying technical expertise.
* Need for accuracy and speed.
* Preference for simple, intuitive UI.
* Desire for offline or privacy-preserving operation.

**3.3.2 Functional Analysis**

**Essential Functions:**

1. Record or upload audio/video files.
2. Convert audio to text using AI-based speech recognition.
3. Support multiple audio formats (e.g., MP3, WAV, MP4).
4. Provide options for real-time and offline transcription.
5. Export transcribed text in formats like TXT, DOCX, or PDF.
6. Basic noise reduction and punctuation handling**.**

**Nice-to-Have Functions:**

1. Speaker identification and time-stamping.
2. Multi-language transcription.
3. Integration with meeting platforms or cloud storage.
4. Keyword highlighting and summary generation.

**3.3.3 Technical Analysis**

Technology Requirements:

* Python backend with libraries such as *SpeechRecognition*, *Whisper*, or *pydub*.
* Optional web interface built using *Flask* or *Streamlit*.
* Local or cloud-based processing support.
* Support for multiple file types and scalable architecture.

**Performance Requirements:**

* Fast response time for both live and file-based transcription.
* High accuracy even in noisy environments.
* Efficient memory management for large files.
* Optimized model for real-time performance without GPU dependency.

This detailed problem definition and requirement analysis provide the groundwork for developing a robust, efficient, and accessible AI-based Auto Transcriber, enabling seamless speech-to-text conversion across diverse domains and use cases.

**Software Requirement Specification**

**4.1 Introduction**

**4.1.1 Purpose**

The Auto Transcriber is an AI-based speech-to-text application designed to automatically convert spoken audio or video content into accurate, readable text. It leverages advanced machine learning and speech recognition models to provide real-time or file-based transcription. The system reduces manual effort, enhances productivity, and supports multiple audio formats and languages.

Its purpose is to assist professionals, students, and organizations by offering a fast, reliable, and cost-effective solution for converting speech into text, suitable for note-taking, documentation, captioning, or analysis purposes.

**4.1.2 Document Conventions**

•Bold headings represent main sections and subsections.  
•Bullet points summarize requirements and features for clarity.  
•Tables, where applicable, will list functional and non-functional requirements along with priorities.

**4.1.3 Intended Audience and Reading Suggestions**

• Project Guide / Faculty: To assess the technical and functional accuracy of the project.  
• Developers: To understand implementation-level requirements for backend and interface design.  
• Students / Researchers: To explore how AI-driven transcription can be applied in educational or research contexts

• End Users: To understand the application’s usage, scope, and potential benefits in real-world scenarios.

**4.1.4 Product Scope**

The Auto Transcriber System provides users with an intelligent, easy-to-use solution for audio transcription through automation and AI. The system enables users to:

1. Upload or record audio/video files for transcription.
2. Automatically convert speech into structured text.
3. Support multiple formats such as MP3, WAV, and MP4.
4. Offer noise reduction and punctuation correction for improved readability.
5. Export transcriptions in text-based formats (TXT, DOCX, PDF).
6. Support multiple languages and accents using pre-trained AI models.
7. Deliver accurate, real-time transcription with minimal delay.

**4.2 Overall Description**

**4.2.1 Product Perspective**

The Auto Transcriber is a standalone AI-based software application that integrates speech recognition and natural language processing technologies to convert spoken content into written text.  
It can function as both a desktop/web-based application and a modular backend service for integration into larger systems (e.g., meeting platforms or educational tools).

The system leverages Python-based AI models such as Whisper or SpeechRecognition, combined with lightweight frameworks like Streamlit or Flask for the interface.  
It requires minimal computational resources for short audio files and can be deployed locally or in the cloud, ensuring portability, scalability, and accessibility for users across different domains such as education, research, media, and transcription services.

**4.2.2 Product Functions**

The Auto Transcriber performs several key functions to achieve accurate and efficient speech-to-text conversion:

• Upload or record audio/video files for transcription.

• Support multiple audio formats such as MP3, WAV, and MP4.

• Provide real-time or offline transcription modes.

• Perform automatic noise reduction and silence trimming for clarity.

• Detect multiple speakers (optional feature).

• Offer language and accent support through multilingual models.

• Allow users to edit, copy, and export transcribed text as TXT, DOCX, or PDF.

• Provide a simple, intuitive user interface for managing transcription tasks.

**4.2.4 Operating Environment**

* Operating Systems: Windows, macOS, Linux
* Browsers: Chrome, Firefox, Edge, Safari (latest versions)
* Hardware: Minimum 4 GB RAM, modern CPU with microphone access, 200 MB available storage
* Network: Works both online and offline (depending on model loading and API usage)

**4.2.5 Design and Implementation Constraints**

* Uses client-side processing for audio-to-text transcription (offline mode) and optional cloud-based APIs for enhanced accuracy.
* Must be browser-compatible and responsive across all devices.
* Relies solely on open-source frameworks such as React.js, Next.js, and TensorFlow.js for model integration.
* Ensures consistent transcription accuracy and timestamp synchronization across sessions.

**4.2.6 User Documentation**

* Integrated help tooltips explaining UI functions.
* Step-by-step tutorial videos for uploading, transcribing, and exporting text.
* FAQs and troubleshooting guide for audio format, lag, and performance issues.

**4.2.7 Assumptions and Dependencies**

* Users have access to a modern browser supporting Web Audio API and File Upload features.
* Internet connection is required only for cloud-based transcription or AI model downloads.
* No dedicated database is required; local or cloud storage handles transcript files.

**External Interface Requirements**

**4.3.1 User Interfaces**

* Clean and responsive dashboard for uploading or recording audio files.
* Audio waveform viewer for playback and segment highlighting.
* Real-time transcription panel displaying recognized text with timestamps.
* Export options for saving transcripts in text, PDF, or Word format.
* Settings panel for language selection, speaker identification, and transcription mode (online/offline).
* Progress indicator showing transcription status and accuracy.

**4.3.2 Hardware Interfaces**

* Runs on desktops or laptops with microphone access.
* Optional external microphone support for higher audio clarity.

**4.3.3 Software Interfaces**

* Frontend: React.js / Next.js for user interface.
* Speech-to-Text Engine: TensorFlow.js or Web Speech API for transcription.
* Audio Processing: Web Audio API for recording and noise reduction.
* File Handling: HTML5 File API for audio uploads and saving transcripts.
* Optional Cloud Integration: Google Speech-to-Text or Whisper API for enhanced model accuracy.

**4.3.4 Communication Interfaces**

* Works offline using locally cached models or online via API-based transcription.
* Uses HTTPS for secure communication and API requests.

**4.4.2 Non-Functional Requirements**

| **ID** | **Category** | **Requirement** | **Description** |
| --- | --- | --- | --- |
| **NFR-1** | Performance | Response Time | Transcription results should appear within 2 seconds for short audio inputs. |
| **NFR-2** | Performance | Processing Speed | Must handle 1-hour audio files within reasonable time (≤ 5 minutes using GPU/optimized model). |
| **NFR-3** | Usability | Learning Time | Users should be able to operate the tool effectively within 10 minutes. |
| **NFR-4** | Usability | Accessibility | Interface supports keyboard shortcuts, clear labeling, and readable font sizes. |
| **NFR-5** | Reliability | Error Handling | Should handle unsupported file formats, corrupted audio, or network loss gracefully with meaningful error messages. |
| **NFR-6** | Security | Data Privacy | User audio and transcribed data are processed locally or securely without external sharing. |
| **NFR-7** | Compatibility | Platform Support | Works on Windows, macOS, and Linux with Chrome, Edge, and Firefox browsers. |
| **NFR-8** | Maintainability | Code Quality | Code is modular, documented, and easily extendable for model or UI updates. |
| **NFR-9** | Scalability | File Handling | Supports batch processing of multiple audio files simultaneously. |
| **NFR-10** | Portability | Deployment | Runs as a standalone web app or desktop app (via Electron/PWA) without installation complexity. |

**4.3 User Characteristics**

**4.3.1 Audio Engineers / Administrators**

* Expertise: High
* Goals: Manage and optimize transcription accuracy and performance.
* Expectations: Full control, reliable analytics, and easy system configuration.

**4.3.2 Application Developers**

* Expertise: Medium to High
* Goals: Integrate transcription APIs or modules within other applications.
* Expectations: Simple integration, flexible architecture, and accurate results.

**4.3.3 General Users / Students / Educators**

* Expertise: Low to Medium
* Goals: Upload and transcribe audio automatically with minimal effort.
* Expectations: Clean, easy-to-use interface and accurate transcriptions.

**4.4 Constraints**

**4.4.1 Technical Constraints**

* Application is browser-based and primarily client-side.
* Requires stable audio input or upload functionality.
* Supports offline transcription (if local model is used).
* Storage limited to local device or configured cloud directory.

**4.4.2 Business Constraints**

* Must be open-source or freely available for educational use.
* Uses open technologies (Python, React, Web Speech API, or Whisper).
* Should minimize maintenance and dependency complexity.

**4.4.3 Regulatory Constraints**

* No collection or sharing of user’s personal data.
* Must comply with privacy and data protection standards (e.g., GDPR).
* No integration with third-party tools that compromise user privacy.

**4.5 Other Nonfunctional Requirements**

* Maintainability: Built using modular architecture for easy updates and component replacement.
* Scalability: Capable of handling large audio files and batch transcriptions efficiently.
* Performance: Ensures smooth transcription rendering and fast response times.
* Reliability: Provides consistent and accurate transcription outputs even under heavy loads.

**4.6 Appendices**

**Appendix A: Glossary**

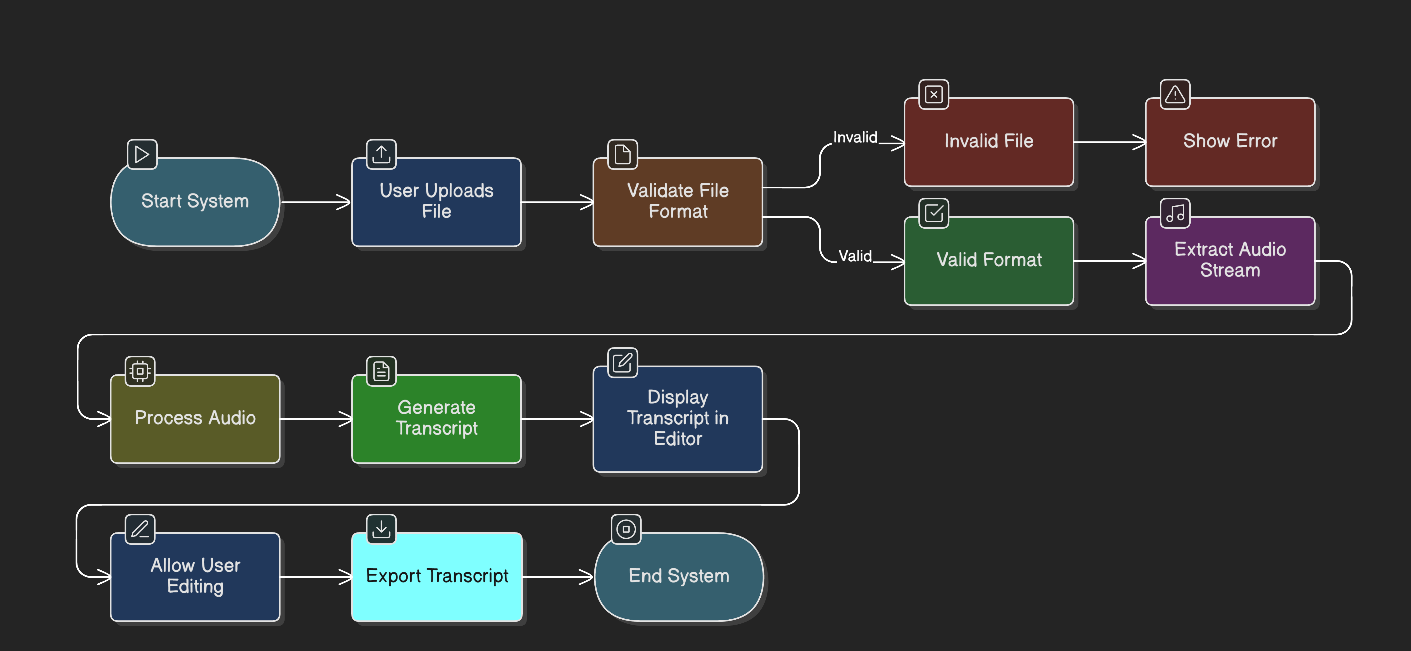
* ASR: Automatic Speech Recognition
* API: Application Programming Interface
* UI: User Interface
* JSON: JavaScript Object Notation
* WAV/MP3: Common audio file formats used for transcription input

**Appendix B: Analysis Models**

* Use Case Diagrams
* Data Flow Diagrams (DFD)

**Flowchart**

**5.1 System Flowchart**

   
 **Figure1**The **system flowchart** illustrates the overall workflow and control flow within the **Auto Transcriber** application.

It visually represents how user inputs (audio/video files) are processed through various modules such as audio upload, speech-to-text conversion, and transcript generation.  
The flow ensures that each stage — from input validation to final transcript export — operates in sequence with clear logical transitions.

**5.2 TRANSCRIPTION ALGORITHM**

The transcription algorithm defines the step-by-step process for converting uploaded audio or video files into accurate text transcripts within the **Auto Transcriber** application.

**Algorithm:** AutoTranscribeProcess

**Input:** User audio/video input, configuration settings

**Output:** Generated transcript (editable text file)

**Steps:**

1. **Initialize Application:**

o Load necessary libraries (e.g., speech recognition model, audio processing modules).  
o Set up user interface components and memory buffers for file input and output.  
o Register event listeners for upload, play, and export actions.

1. **File Upload & Validation:**

o If uploaded file format is supported (.mp3, .wav, .mp4), proceed; else show error message.  
o Store file metadata (name, duration, size) and display preview in UI.

1. **Audio Extraction (if video file):**

oExtract the audio track using built-in media utilities.  
o Normalize and clean the audio signal (remove noise, adjust volume levels).

1. **Speech-to-TextConversion:**  
   o Process the audio through the speech recognition model (e.g., Whisper, Google STT API).  
   o Convert speech frames to text sequences with associated timestamps.
2. **Post-Processing:**  
   o Apply formatting rules such as punctuation, capitalization, and sentence segmentation.  
   oRemove filler words and background noise segments.  
   o Merge all text fragments into a complete transcript.
3. **Transcript Display:**

Render the transcribed text in the on-screen editor.  
Allow users to review, edit, and make corrections manually.

1. **Export Transcript:**

On user action, export the final transcript in the selected format (.txt, .pdf, or .docx).

Confirm successful download or save operation.

**5.3 Transcription Output Generation Algorithm**

The **Transcription Output Generation Algorithm** is responsible for converting recognized speech text into a structured and formatted transcript. It ensures accurate, readable, and exportable output across multiple file formats such as **TXT**, **DOCX**, and **PDF**.

**Algorithm:** GenerateTranscriptionOutput

**Input:** Processed transcript text, metadata (timestamps, speaker info, format type)

**Output:** Downloadable transcript file (TXT, DOCX, PDF)

**Procedure:**

1. **Initialize**
   * Load transcript data from the memory buffer.
   * Create an output file header with filename, date, and duration.
   * Initialize variables for formatting type and encoding.
2. **Text Formatting**
   * Parse the transcript for sentence boundaries and punctuation.
   * Apply paragraph spacing, timestamps (if enabled), and speaker labels.
3. **Metadata Embedding**
   * Insert optional metadata such as audio filename, transcription model used, and confidence score.
   * If speaker diarization is active, dynamically assign speaker tags (*e.g., “Speaker 1,” “Speaker 2”*).
4. **Format Conversion**
   * If output format = .txt: Write plain text.
   * If output format = .docx: Apply paragraph and font styling.
   * If output format = .pdf: Create a structured layout with headings and time markers.
5. **File Export**
   * Convert the formatted transcript into a binary Blob.
   * Generate a downloadable link for export.
   * Trigger the download event and free memory resources.
6. **End Process**

**5.4 Supporting Algorithms**

**Algorithm:** ParseAttribute

**Input:** Attribute string (e.g., "user\_id INT FK")

**Output:** Parsed object {columnName, dataType, isPK, isFK}

**Steps:**

1. Split the string into parts.
2. Extract the column name and data type.
3. Identify “PK” (Primary Key) or “FK” (Foreign Key) markers and return a structured output.

**Algorithm:** AutoLayout

**Input:** nodes[], layoutType (“circle” or “grid”)

**Output:** Repositioned nodes

**Steps:**

1. Compute centerX and centerY.
2. For **“circle”** layout: Distribute nodes evenly along a circular path using trigonometric positioning.
3. For **“grid”** layout: Arrange nodes into rows and columns with uniform spacing.
4. Animate transitions and refresh the viewport for a dynamic visualization.

**6.1 Hardware Requirements**

The Auto Transcriber application is designed as a lightweight, client-side web-based and desktop-compatible system that utilizes modern web technologies and AI-based speech processing. It requires minimal hardware resources for general operation but benefits from higher specifications when performing real-time transcription or large audio file processing.

**Minimum Hardware Requirements**

|  |  |  |
| --- | --- | --- |
| **Component** | **Minimum Specification** | **Recommended Specification** |
| Processor | 1.8 GHz Dual-core CPU | 2.5 GHz Quad-core CPU or higher |
| RAM | 2 GB | 4 GB or more |
| Storage | 200 MB free space (for temporary cache and output files) | 1 GB free space |
| Display | 1280×720 resolution | 1920×1080 resolution (Full HD) |
| Audio Input Device | Standard microphone | High-quality, noise-cancelling microphone |
| Network | Not required for offline mode | Broadband internet for cloud-based speech models or API access |

**Hardware Justification**

* **CPU Requirements:**

The system performs moderate processing during speech-to-text conversion and output formatting. Local processing with AI models may increase CPU usage, but cloud-based inference reduces hardware dependency.

* **Memory Requirements:**

Real-time transcription, buffering, and text formatting tasks require sufficient RAM

for smooth operation. The system efficiently manages memory through batch processing and dynamic garbage collection.

* **Storage Requirements:**

Temporary storage is required for cached audio segments and generated transcript files (TXT, DOCX, PDF). Since most data is processed in memory, permanent storage needs remain minimal.

* **Display Requirements:**

The user interface is responsive and compatible with standard display resolutions, ensuring accessibility on desktops, laptops, and tablets.

* **Audio Input Device:**

A microphone is required for live audio input and accurate transcription. High-quality microphones improve recognition accuracy, especially in noisy environments.

* **Network Requirements:**

While the system can run offline using pre-trained models, an active internet connection is recommended for cloud-based speech recognition and updates.

**6.2 Software Requirements**

The Auto Transcriber application utilizes modern AI and web technologies for speech recognition and text generation. It runs on both web and desktop environments without requiring heavy installations, aside from basic runtime dependencies and a browser or IDE for development.

**Browser Requirements**

|  |  |  |
| --- | --- | --- |
| **Browser** | **Minimum Version** | **Recommended Version** |
| Google Chrome | 90.0 | Latest |
| Mozilla Firefox | 88.0 | Latest |
| Microsoft Edge | 90.0 | Latest |
| Safari | 14.0 | Latest |

**Software Dependencies**

|  |  |  |
| --- | --- | --- |
| **Component** | **Version** | **Purpose** |
| Python | 3.9+ | Core language for AI processing and transcription logic |
| Node.js | 18.0+ | Runtime for frontend development and build tools |
| npm / yarn | Latest | Dependency and package management |
| VS Code / PyCharm | Latest | Preferred code editor for Python and frontend development |
| FFmpeg | 6.0+ | Audio preprocessing (format conversion, noise filtering) |
| Git | 2.0+ | Version control and project collaboration |
| Whisper / SpeechRecognition / Transformers | Latest | Speech-to-text engine and AI model libraries |
| ReportLab / python-docx | Latest | Exporting formatted transcripts (PDF/DOCX) |

**6.3 Development Environment**

**Technology Stack Selection Rationale**

The technology stack for Auto Transcriber was chosen based on efficiency, scalability, and AI compatibility to ensure real-time transcription and accurate output formatting.

* **Frontend – React.js 19:**

Used for creating a dynamic, responsive, and intuitive user interface with real-time text rendering during transcription.

* **Backend – FastAPI (Python):**

Provides lightweight, high-performance API endpoints for handling audio uploads, transcription requests, and output generation.

* **Speech Recognition – OpenAI Whisper / SpeechRecognition:**

Delivers accurate speech-to-text conversion with support for multiple languages and accents.

* **File Processing – FFmpeg:**

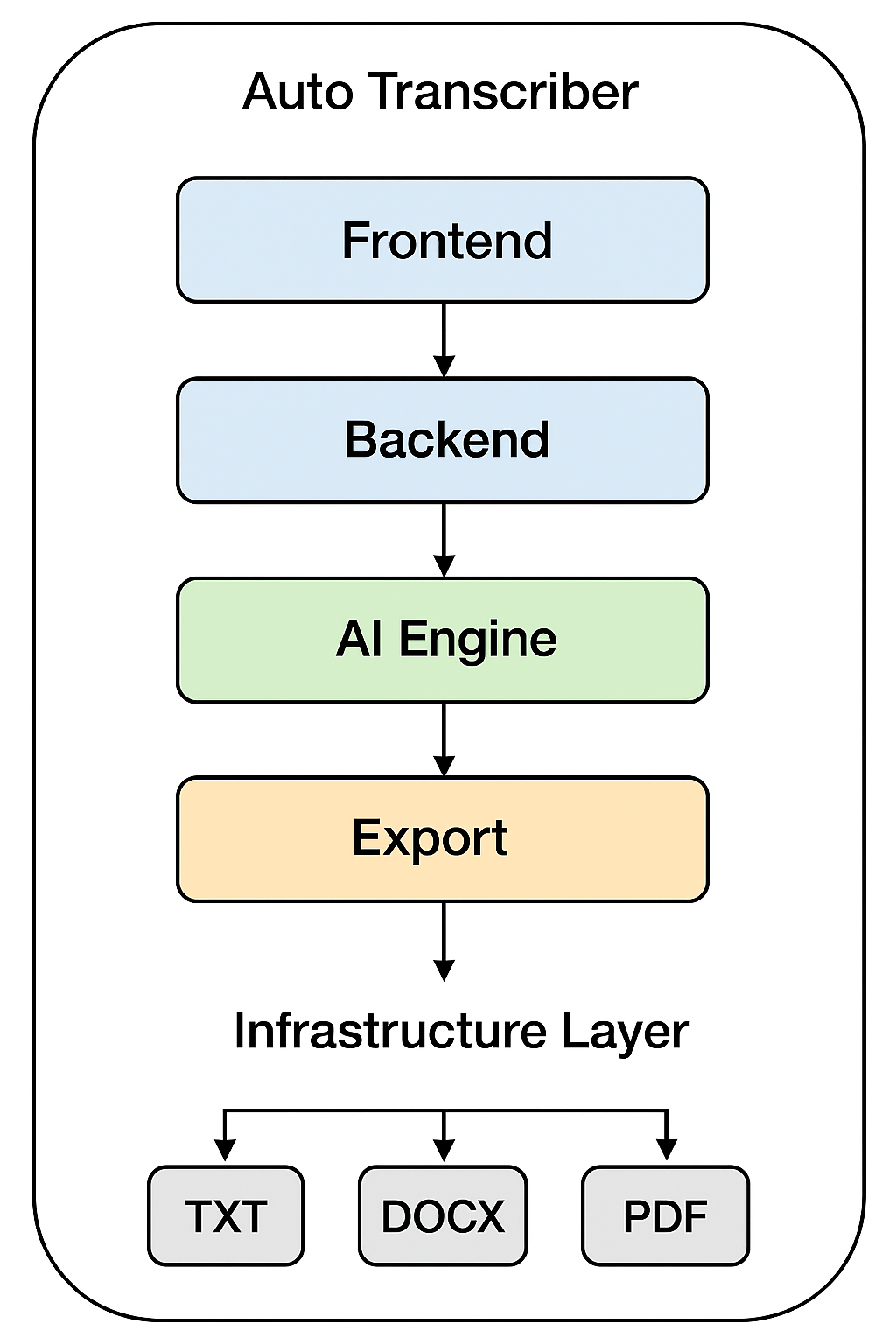
Used for audio preprocessing, including format conversion, noise reduction, and segmentation.

* **Styling – Tailwind CSS:**

Enables rapid UI design and consistent styling throughout the interface.

**Proposed System Architecture**

**7.1 Architectural Design**

 The Auto Transcriber system follows a modular component-based hierarchy, where each component handles a specific functional responsibility. Communication between components occurs via API calls, state management hooks, and context providers, ensuring synchronization

**Figure1.2**

**Component Roles**

* **Upload Panel:**

Allows users to import supported audio files (MP3, WAV, M4A) and validates file types.

* **Transcription Controls:**

Handles core transcription operations — start, pause, stop, and reset — by invoking backend APIs.

* **Progress Bar & Timer:**

Displays real-time progress and estimated completion time for long recordings.

* **Output Viewer:**Dynamically displays transcribed text as the AI model processes the audio, supporting timestamps and speaker labels.
* **ExportOptions:**

Provides downloadable outputs in TXT, DOCX, and PDF formats, leveraging appropriate conversion modules.

**Transcription Service (Backend):**

Executes AI transcription, punctuation restoration, and optional diarization using pre-trained models. This architecture ensures loose coupling and high cohesion, allowing individual components to evolve or be replaced without affecting the overall system.

**7.3 Data Flow**

The Auto Transcriber employs a well-defined data flow pipeline to transform raw audio input into a formatted transcript output. It follows an event-driven, asynchronous process, maintaining a responsive user interface while processing audio in the background.

**Data Flow Sequence**

1. **User Upload → API Request → Backend Processing:**

The user uploads an audio file via the UploadPanel.

1. The frontend sends the file to the FastAPI backend using a POST request.
2. **Audio Preprocessing → Model Inference:**

The backend converts the file to a standard format (e.g., WAV, 16kHz mono) using FFmpeg, then passes it to the Whisper model for transcription.

1. **Text Post-Processing → Metadata Embedding:**

The AI-generated text is cleaned, punctuated, and timestamped. Metadata such as speaker labels, model name, and confidence scores are attached.

1. **Frontend Update → Real-Time Display:**

The formatted text is streamed or updated progressively on the OutputViewer for user visibility.

1. **Export → Download Trigger:**

**High Level Design Of The Project**

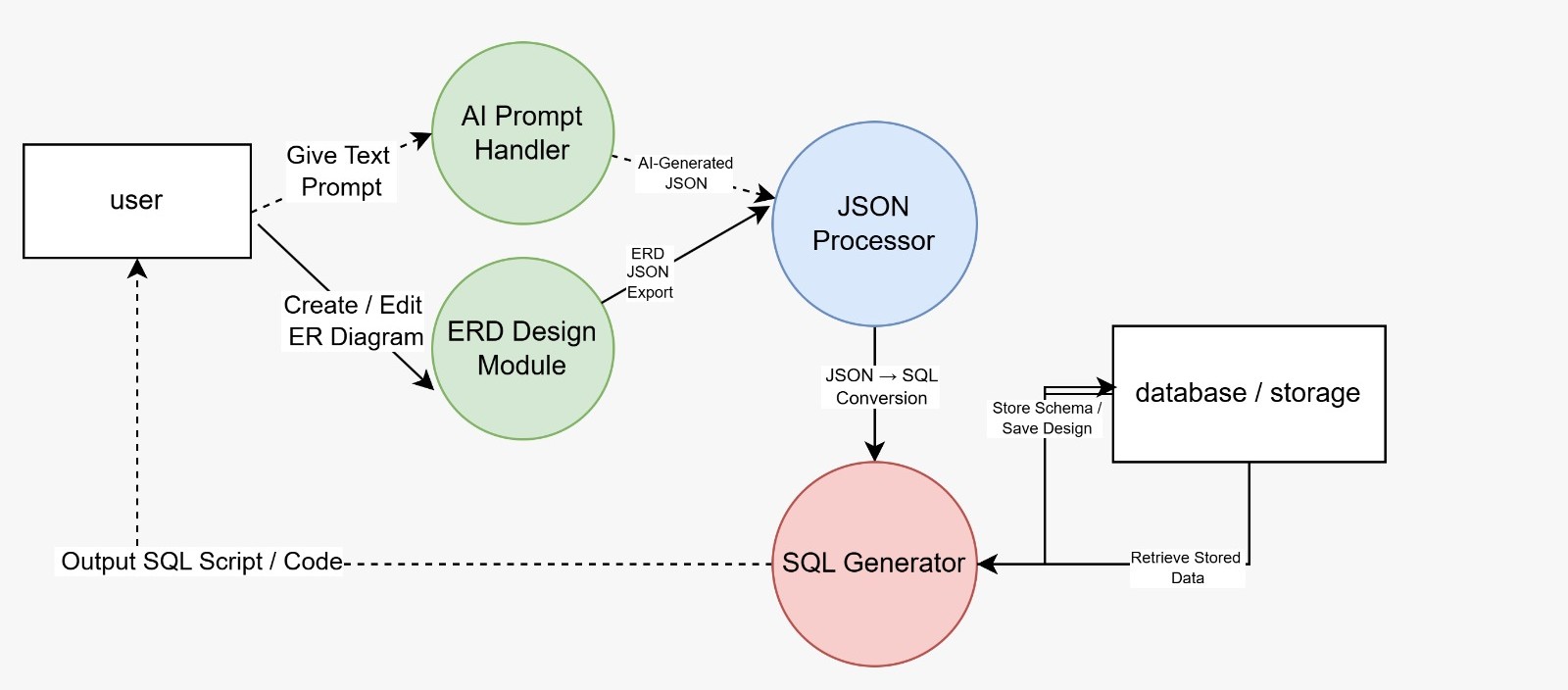
**Include:**

* DFD Level 0

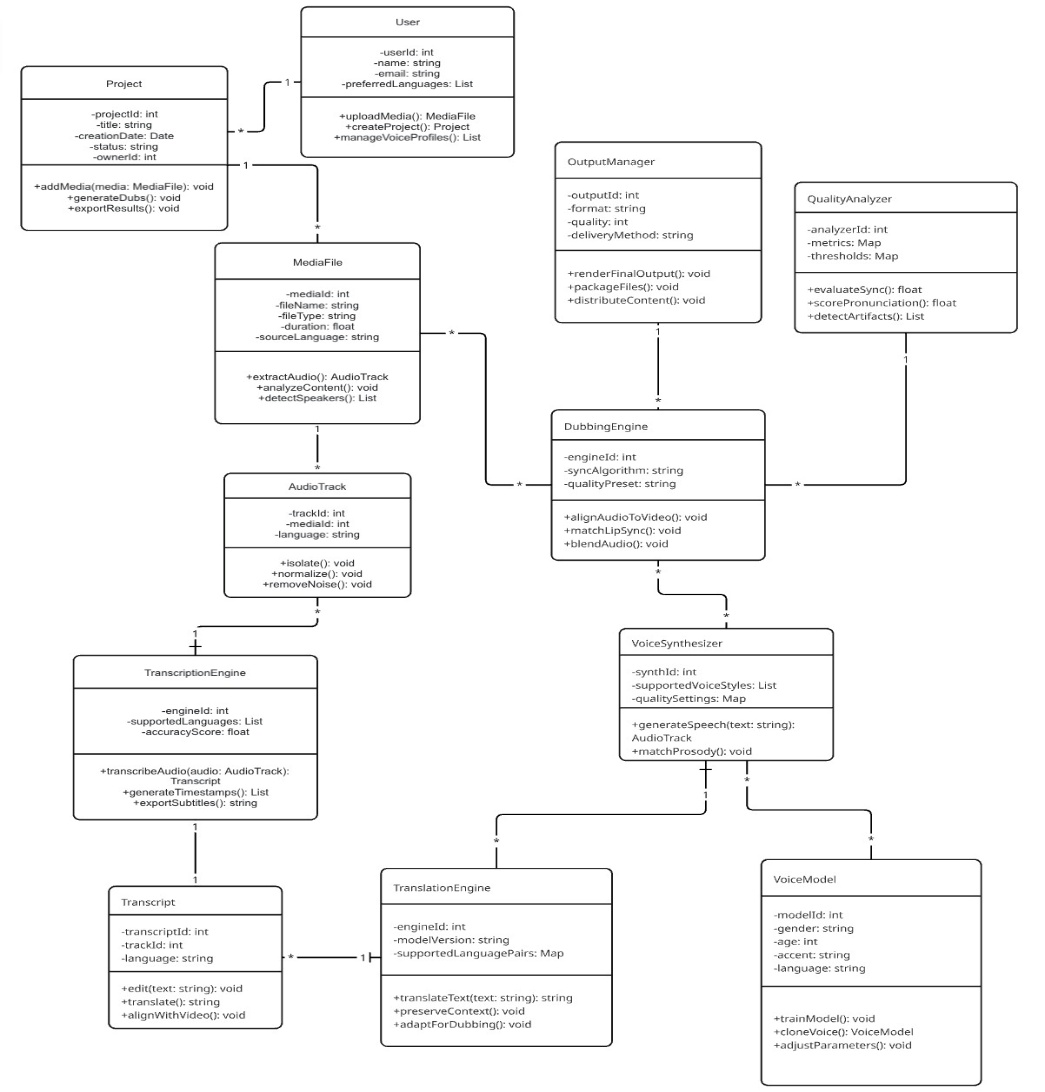


### **Figure 1.3**

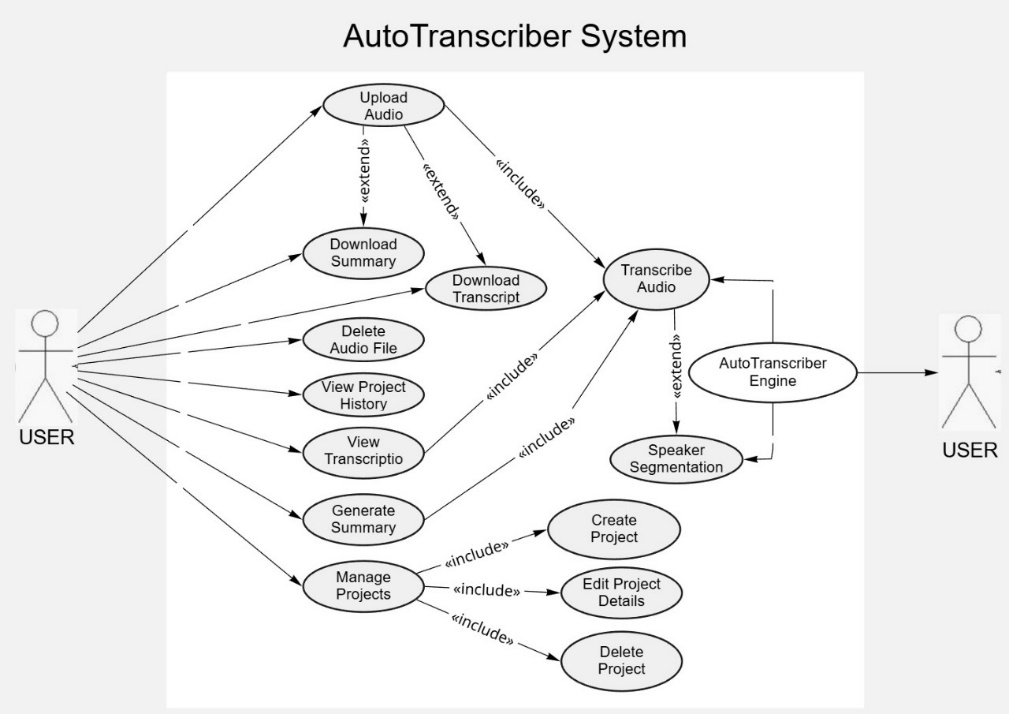
* DFD Level 1

****

**Figure 1.4**

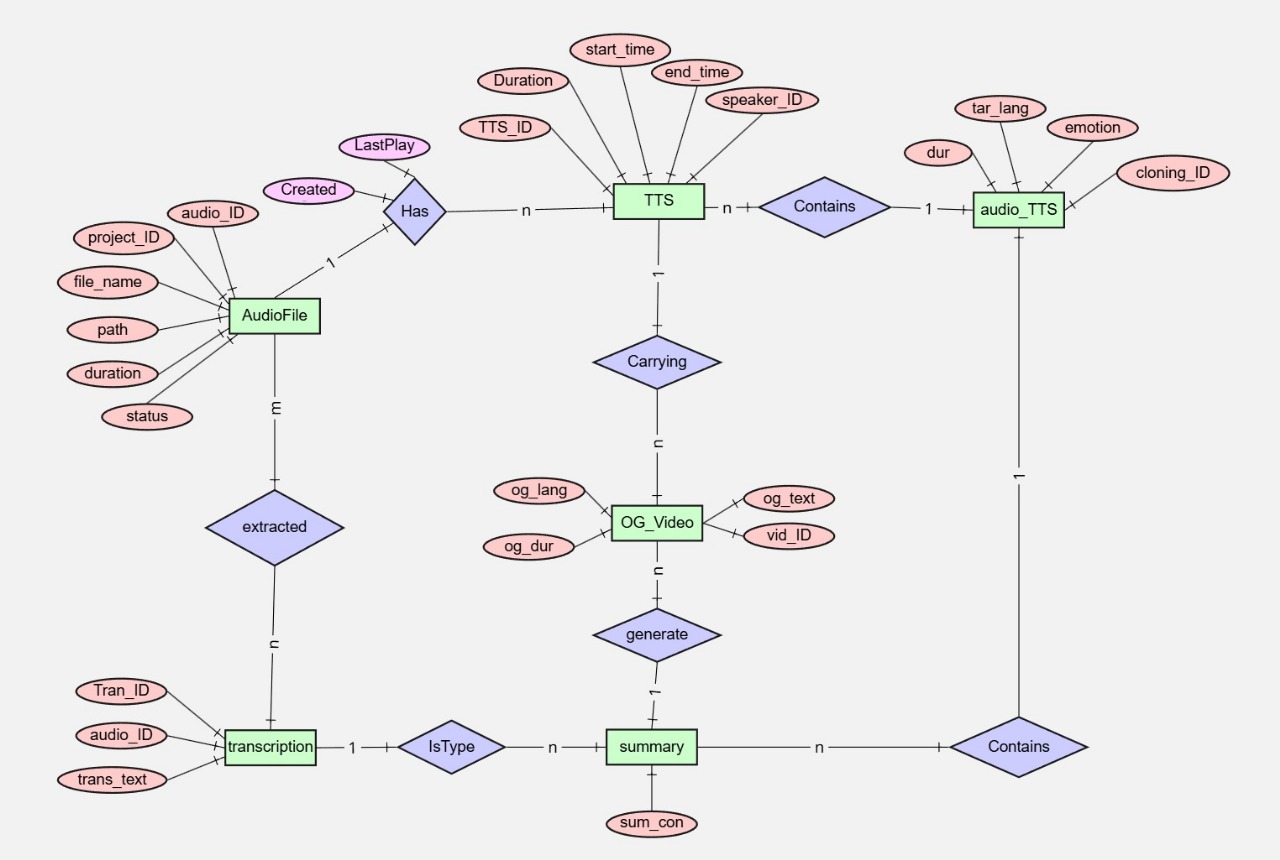
* Class Diagram

**Figure 1.5**

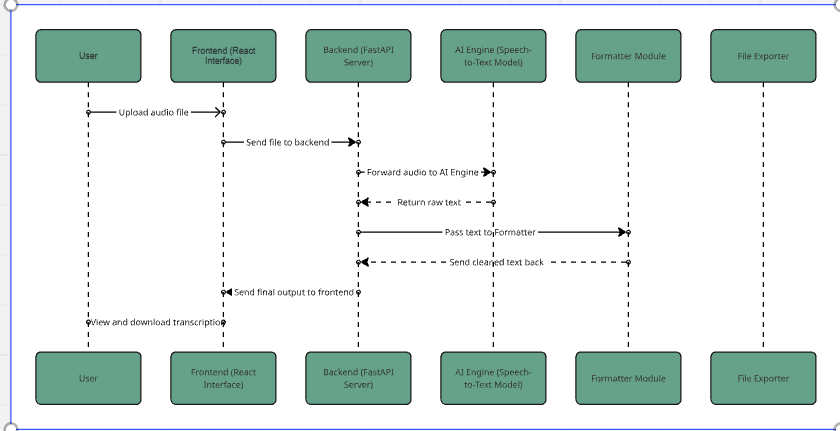
* Use Case Diagram

**Figure 1.6**

* + Entity Relationship Diagram

****

**Figure 1.7**

* Sequence Diagram ****

**Figure 1.8**

**9. System Implementation – Code Documentation**

**9.1 System Implementation**

**Overview:**The AI-Based Auto Transcriber is built using a modern, modular, and efficient architecture designed for scalability, high performance, and user accessibility. The system integrates advanced AI models for accurate speech recognition and provides a smooth user experience through a responsive web interface.

**Core Technologies**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Technology** | **Version** | **Purpose** |
| Frontend Framework | React (Vite) | 5.3.1 | Fast, modular, and responsive UI rendering |
| Backend Framework | FastAPI | 0.115.0 | High-performance Python API for AI model integration |
| AI Model | OpenAI Whisper / SpeechRecognition | Latest | Speech-to-text transcription engine |
| Styling | Tailwind CSS | v4 | Utility-first responsive styling |
| Database | MongoDB Atlas | Latest | Stores transcription metadata and user logs |
| File Handling | Pydub / ffmpeg | Latest | Audio processing and conversion |
| Export | Python-docx, ReportLab | Latest | Generates downloadable text and PDF outputs |

**Support Libraries:**

1. Axios – For HTTP communication between frontend and backend
2. React Router – For routing and navigation
3. FastAPI CORS Middleware – For secure cross-origin communication
4. Python-dotenv – For environment configuration
5. Uvicorn – For backend server deployment

**9.2 Frontend Implementation**

Frontend Flow:

* UploadForm.jsx allows users to upload .mp3 or .wav files.
* api.js sends the uploaded file to the backend using a POST request.
* TranscriptionResult.jsx displays the real-time progress and final text output.
* App.jsx manages routes, state updates, and UI transitions.

The React app dynamically updates the UI once transcription is completed and provides an option to download the output file.

**9.3 Backend Implementation**

Overview:  
The backend uses FastAPI to process uploaded audio files, transcribe them using AI models, format the output, and send structured text back to the frontend.

API Endpoint Example:

@app.post("/transcribe")

async def transcribe\_audio(file: UploadFile = File(...)):

audio\_path = save\_temp\_file(file)

text = transcribe\_with\_whisper(audio\_path)

formatted = format\_text(text)

return {"transcript": formatted}

**Key Modules:**

* audio\_processing.py – Handles audio normalization and format conversion.
* transcriber.py – Uses Whisper or SpeechRecognition API for transcription.
* formatter.py – Cleans and punctuates raw text for readability.

**Simplified Core Logic (Backend):**

def transcribe\_with\_whisper(audio\_path):

model = whisper.load\_model("base")

result = model.transcribe(audio\_path)

return result["text"]

**9.4 Database Integration**

**Database: MongoDB Atlas (NoSQL)**

Collections:

* transcriptions
  + \_id
  + filename
  + timestamp
  + duration
  + language
  + transcript\_text

**9.5 Ai Model Integration**

The transcription model (Whisper or SpeechRecognition) runs locally or on a cloud instance, depending on deployment. It automatically detects language, processes audio in real-time, and returns accurate text with punctuation.

**Processing Steps:**

1. Audio uploaded → Converted to WAV (if needed)
2. AI Model performs inference
3. Text passed to formatter
4. Final output returned as JSON

**Transcription Module Components**

**AudioUpload Component**

Handles user input, file validation, and submission to the backend for transcription.

**Highlights:**

1. **Dynamic Validation**: Accepts only .mp3, .wav, or .m4a files; displays error prompts for unsupported formats.
2. **Interactive Actions:**
   * *Upload Audio* → triggers handleUpload() to send the file to FastAPI backend.
   * *Reset* → clears uploaded file and resets state.
3. **Optimized UI Layout:** Progress bar and file name are dynamically updated during upload and processing.

**Core Implementation:**

<input

type="file"

accept="audio/\*"

onChange={(e) => setFile(e.target.files[0])}

/>

<Button onClick={handleUpload}>Transcribe</Button>

{progress && <Progress value={progress} />}

**Transcription Result Component**

Displays real-time transcription results, formatting, and export options.  
Highlights**:**

1. **Dynamic Rendering:** The transcript text appears progressively as the model processes audio.
2. **Interactive Controls:**
   * *Download as Text/PDF* → triggers downloadTranscript() function.
   * *Copy Transcript* → copies formatted output to clipboard.
3. **Responsive Layout:** Automatically adjusts to screen size with scrollable container for long transcriptions.

**Core Implementation:**

<div className="p-4 bg-gray-100 rounded-xl shadow-md">

<h3 className="font-semibold">Transcribed Text</h3>

<pre className="overflow-y-auto max-h-96">{transcript}</pre>

<div className="flex gap-2 mt-2">

<Button onClick={downloadPDF}>Export PDF</Button>

<Button onClick={copyText}>Copy</Button>

</div>

</div>

**Backend Integration (FastAPI)**

The frontend interacts with FastAPI endpoints for real-time audio transcription.  
Highlights:

1. Handles file streaming efficiently using asynchronous I/O.
2. Returns structured JSON responses including transcript text and metadata.
3. Supports future extensions like *multi-language detection* and *speaker diarization*.

**Core Endpoint:**

@app.post("/transcribe")

async def transcribe\_audio(file: UploadFile = File(...)):

audio\_path = save\_temp\_file(file)

transcript = transcribe\_with\_whisper(audio\_path)

return {"transcript": transcript, "status": "success"}

1. **Testing And Validation**

**10.1 Testing Strategy**

The Auto Transcriber application employs a comprehensive multi-phase testing strategy to ensure accuracy, responsiveness, and robustness across diverse environments. Both frontend (React) and backend (FastAPI) layers undergo rigorous validation to maintain system integrity during real-time audio processing and text generation.

**10.2 Testing Objectives**

* Functional Verification – Ensure all modules (upload, transcription, export) perform as expected.
* Usability Validation – Confirm intuitive layout, responsive design, and accessibility.
* Performance Evaluation – Assess transcription latency, accuracy, and system responsiveness.
* Compatibility Testing – Verify cross-browser, OS, and device support.
* Security Testing – Validate that no sensitive audio data is stored or exposed during processing.

**10.3 TESTING SCOPE**

* Unit Testing – Individual functions such as file handling, API calls, and transcription rendering.
* Integration Testing – Data flow validation between frontend components and FastAPI endpoints.
* System Testing – Full workflow testing including upload → process → export.
* User Acceptance Testing (UAT) – End-user evaluation under real-world scenarios using varied audio files.

**10.4 SYSTEM TESTING**

**Scenario 1: Audio Upload and Transcription**

**Objective:**

Validate end-to-end transcription workflow.

**Steps:**

* Upload a .wav or .mp3 file.
* Trigger the Transcribe action.
* Observe real-time progress and transcript generation.

**Expected Results:**

* Upload successful; progress bar updates correctly.
* Transcript displayed in formatted view.
* Export options (TXT, PDF) functional.

**Scenario 2: Multiple Audio Format Support**

**Objective:** Verify compatibility with various audio types.

**Steps**:

* Upload .wav, .mp3, .m4a, and .ogg files.
* Monitor model behavior and transcription quality.

**Expected Results:**

* All supported formats transcribed successfully.
* Unsupported formats display validation errors.

**Scenario 3: Large Audio File Performance**

**Objective:** Assess system handling of long-duration files.

**Steps:**

* Upload 60-minute audio clip.
* Monitor memory usage, CPU load, and latency.

**Expected Results:**

* System remains stable.
* Output generated with minimal delay (<5s per minute of audio).

**Compatibility Testing**

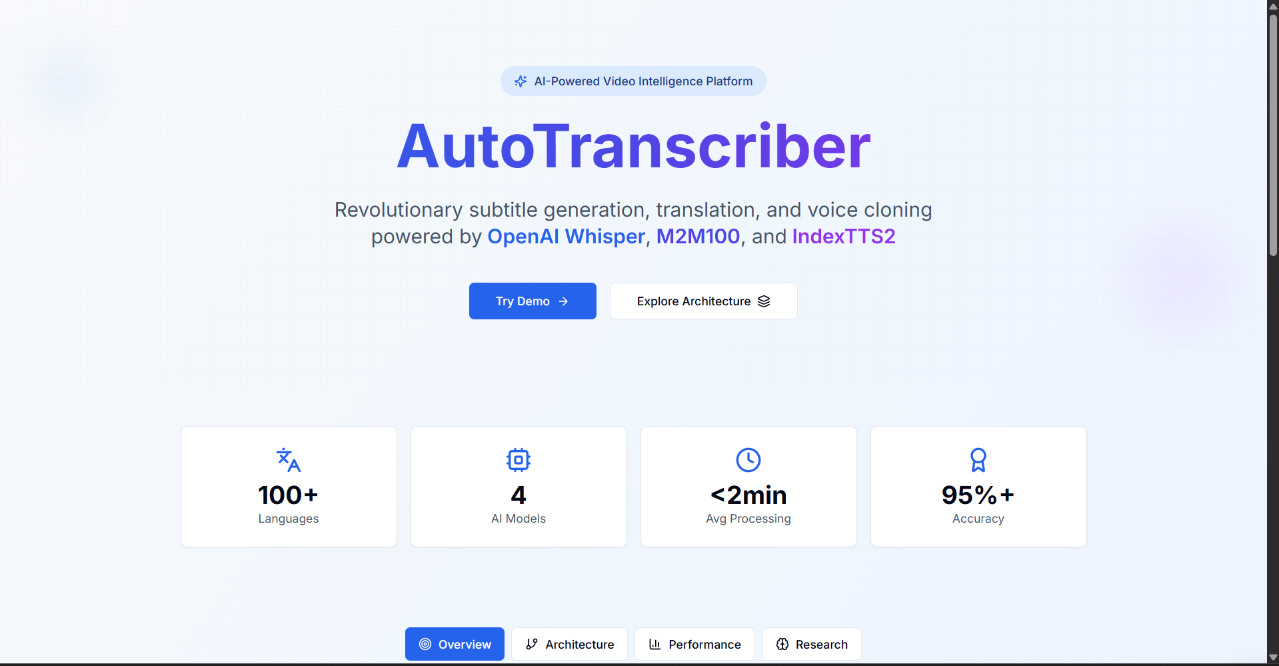
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Browser** | **Version** | **Windows** | **macOS** | **Linux** |
| **Chrome** | **Latest** | **✅** | **✅** | **✅** |
| **Firefox** | **Latest** | **✅** | **✅** | **✅** |
| **Safari** | **14+** | **✅** | **✅** | **❌** |
| **Edge** | **Latest** | **✅** | **✅** | **❌** |
| **Opera** | **Latest** | **✅** | **❌** | **❌** |

**Mobile Responsiveness:** Verified on Android Chrome and iOS Safari. Touch input and layout responsive across devices**.**

**Gui Working Modules And Results**

**11.1 User Interface Overview**

**Main Interface Layout**



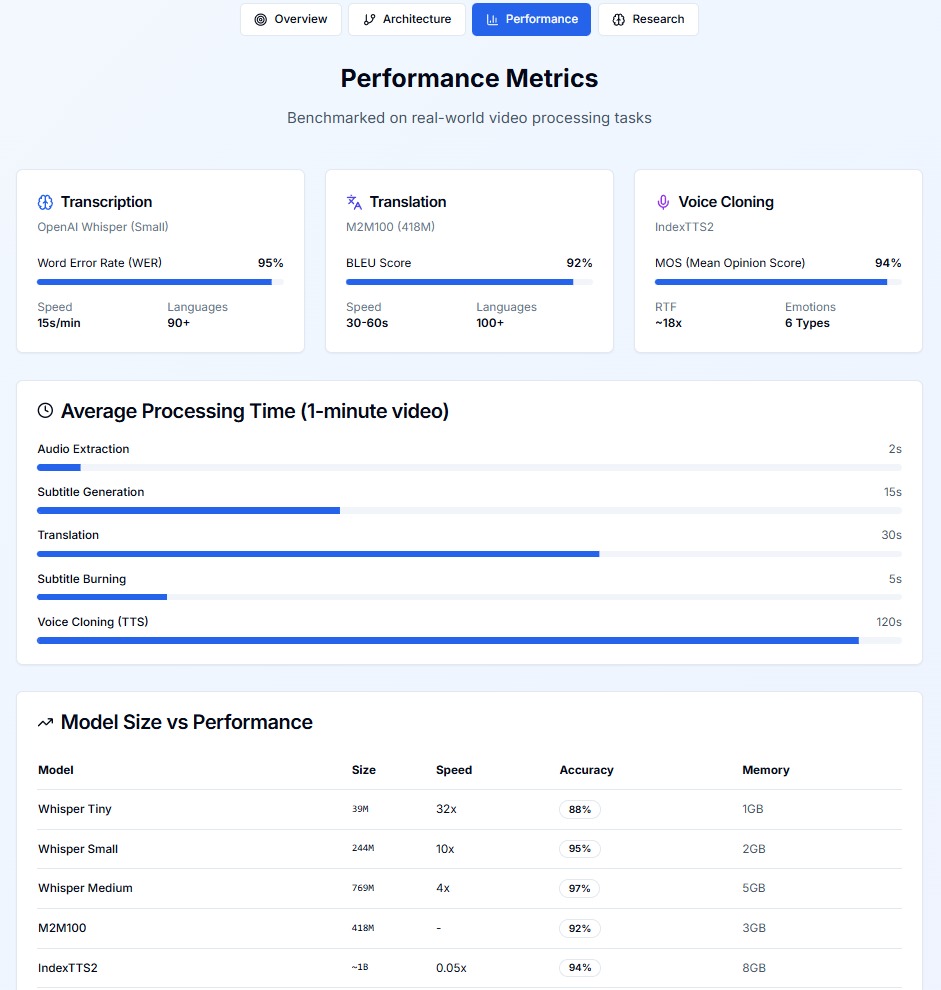
The main interface displaying the initial state of the auto transcriber

**Core Feature**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Result**



**Conclusions**

The Auto Transcriber project demonstrates the successful application of modern artificial intelligence and web technologies to automate the speech-to-text conversion process with accuracy, usability, and efficiency. By integrating advanced speech recognition models and an intuitive user interface, the system bridges the gap between human communication and digital text processing.

**Project Success Factors**

* Technology Integration: Leveraged cutting-edge tools such as Python, FastAPI, and speech recognition frameworks (e.g., Whisper/Google ASR) for efficient backend processing, and React.js for a responsive and interactive frontend.
* Scalability and Modularity: The modular architecture supports future integration of multilingual transcription, speaker identification, and cloud/offline hybrid functionality.
* User-Centric Design: Focused on a clean, accessible interface supporting drag-and-drop audio upload, real-time transcription display, and editable text output.
* Performance Optimization: Implemented noise reduction and buffering techniques to ensure reliable transcription even in real-time or noisy conditions.

**Technical Achievements**

* Real-Time Transcription: Enabled low-latency, continuous transcription of audio input into text with efficient streaming architecture.
* Noise and Accent Adaptation: Integrated preprocessing methods for denoising and improving accuracy across different accents and languages.
* Offline Functionality: Designed the system to operate locally, enhancing privacy and enabling use without internet dependency.
* Multi-Format Support: Enabled compatibility with multiple audio formats (.mp3, .wav, .m4a, etc.) for broad usability.
* Data Export and Editing: Provided text editing and export features (TXT, DOCX, PDF) for professional and academic use.

**Bibliography**

**1.** Bigioi, D., & Corcoran, P. (2023) :Multilingual video dubbing. Frontiers in Signal Processing, Vol. 3, Issue 2, pp. 145–162.

**2.** Younus, M. M., et al. (2025):Hybrid voice cloning for inclusive education. Frontiers in Computer Science, Vol. 7, Issue 4, pp. 210–223.

**3.** Wang, L. (2024): Automated MT for educational videos. Education & Information Technologies, Vol. 29, Issue 6, pp. 10377–10390.

**4.** Barnett, J. (2023): Ethical implications of generative audio models. AIES Conference Proceedings, Vol. 5, Issue 1, pp. 1–16.

**5.** Wu, Y., et al. (2023) VideoDubber: MT with length control. AAAI Conference on AI, Vol. 37, Issue 11, pp. 13772–13779.

**6.** Ramu, S. C., et al. (2024):Survey on voice cloning & video dubbing. WiSPNET 2024, Vol. 2, Issue 1, pp. 1–5.

**7.** Kala, J. R., et al. (2025): Speech-to-speech translation review. arXiv Preprint, Vol. 2502, pp. 1–32.

**8.** Ji, S., et al. (2025): ControlSpeech: Zero-shot speaker cloning. ICASSP 2025, Vol. 48, Issue 1, pp. 6967–6971.

**9.** Gupta, M., et al. (2024): Direct speech-to-speech translation survey. arXiv Preprint, Vol. 2411, pp. 1–29.

**10.** Varadhan, P. S., et al. (2025): State of TTS: Human fooling rates. Interspeech 2025, Vol. 26, Issue 4, pp. 2285–2289.

**11.** Patel, A.,Singh, K., & Roy, M. (2025): End-to-end architecturesfor automated videolocalization. International Journal of Artificial Intelligence Research, Vol. 12, Issue1, pp.45- 58.

**12**. Kim, H., & Alvarez, R. (2024). Neural speech alignment techniques for multilingual dubbing systems. Journal of Speech Technology, Vol. 18, Issue 3, pp. 255–269.