AI Enhanced Video Conferencing App

Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of

Bachelor of Engineering
in
Information Technology

Submitted by

Chetan Choudhary: (Roll No. 22UITE5010)

Himanshu Sharma: (Roll No. 22UITE5021)

Prajjwal Sharma: (Roll No. 22UITE5030)

Samsuzzaman Ansari: (Roll No. 22UITE5031)

Under the Supervision of

Dr. Shrwan Ram
Professor, CSE

Department of Computer Science and Engineering
MBM University, Jodhpur

May, 2025

Department of Computer Science & Engineering
MBM University, Ratanada, Jodhpur, Rajasthan, India –342011

CERTIFICATE

This is to certify that the work contained in this report entitled **"AI Enhanced Video Conferencing App**" is submitted by the group members Mr. Chetan Choudhary (Roll. No: 22UITE5010), Mr. Himanshu Sharma (Roll No: 22UITE5021), Mr. Prajjwal Sharma (Roll. No: 22UITE5030) and Samsuzzman Ansari (Roll No.: 22UITE5031) to the Department of Computer Science & Engineering, MBM University, Jodhpur, for the partial fulfilment of the requirements for the degree of **Bachelor of Engineering** in **Information Technology**.

They have carried out their work under my supervision. This work has not been submitted else-where for the award of any other degree or diploma.

The project work in our opinion, has reached the standard fulfilling of the requirements for the degree of Bachelor of Engineering in accordance with the regulations of the University.

Dr. Shrwan Ram
Professor, CSE
(Supervisor)

Dr. Shrwan Ram
(Head of Department)

DECLARATION

We, ***Chetan Choudhary, Himanshu Sharma, Prajjwal Sharma and Samsuzzaman Ansari,*** hereby declare that this seminar/project titled **"AI Enhanced Video Conferencing App"** is a record of original work done by me under the supervision and guidance of ***Dr. Shrwan Ram***.

We, further certify that this work has not formed the basis for the award of the Degree/Diploma/Associateship/Fellowship or similar recognition to any candidate of any university and no part of this report is reproduced as it is from any other source without appropriate reference and permission.

(Chetan Choudhary)
VIIIth Sem, Information Technology
Roll No. – 22UITE5010

(Himanshu Sharma)
VIIIth Sem, Information Technology
Roll No. – 22UITE5021

(Prajjwal Sharma)
VIIIth Sem, Information Technology
Roll No. – 22UITE5030

(Samsuzzman Ansari)
VIIIth Sem, Information Technology
Roll No. – 22UITE5031

ACKNOWLEDGEMENT

We extend our deepest gratitude to ***Prof. Dr. Shrwan Ram***, whose invaluable guidance, unwavering support, and insightful feedback have been instrumental throughout this project. His expertise, patience, and encouragement have shaped my understanding and fueled my passion for the subject matter.

We would also like to express my sincere appreciation to ***Prof. Dr. Shrawn Ram***, Head of the Computer Science and Engineering Department, for his constant encouragement and support. His visionary leadership and dedication to academic excellence have provided a conducive environment for learning and growth.

ABSTRACT

The increasing reliance on remote communication has made video conferencing platforms an essential tool for professional, educational, and social interaction. However, traditional video conferencing solutions often fall short in delivering a seamless experience due to limitations like background noise, lack of accessibility for users with hearing impairments or language barriers, and limited intuitive interaction mechanisms. To address these challenges, we have developed an **AI-Enhanced Video Conferencing Platform** that integrates gesture control and real-time transcription to redefine user engagement and accessibility in virtual meetings.

Our platform introduces a range of advanced features powered by artificial intelligence to enhance the quality, usability, and inclusivity of video calls. The system includes **real-time multilingual transcription and translation** using Whisper AI, making conversations accessible to non-native speakers and individuals with hearing impairments.

In addition to audio and accessibility improvements, the platform supports **gesture-based control mechanisms** using MediaPipe. Users can perform actions such as muting/unmuting the microphone, raising a virtual hand, or leaving the meeting simply through predefined hand gestures, enabling a touch-free and more intuitive interaction. The system also includes **WebRTC-powered video and screen sharing**, as well as **real-time chat functionality** through WebSockets to facilitate collaborative work.

The frontend of the application is built with **React.js** and styled using **Material UI**, offering a modern, responsive interface across devices. The backend infrastructure is developed using **Node.js and Express**, handling WebSocket communication, API routing, and integration with AI services.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Project

Traditional video conferencing platforms often lacks with limited interactivity that hinders user engagement. To address these limitations, this project presents an advanced video conferencing platform enhanced with artificial intelligence (AI) capabilities including real-time transcription and translation, and gesture-based controls

1.1 Background and Motivation

Traditional video conferencing systems often suffer from significant limitations:

- Lack of accessibility features such as real-time transcription makes participation difficult for users with hearing impairments or those who are non-native speakers.
- Conventional UI-based controls require constant manual interaction, leading to distractions during live discussions.
- Minimal support for natural interaction models (e.g., gesture control or language-based personalization) reduces user engagement and inclusivity.

1.2 Objectives of the Project

The project is driven by the vision of creating a user-centric, immersive, accessible, and efficient remote collaboration experience by integrating Artificial Intelligence (AI) at its core

1.2.1 Functional Objectives

These objectives focus on the core features and functionalities the system must deliver:

1. Live Transcription and Multilingual Translation
    - Use Whisper AI to generate accurate real-time transcriptions of speech during calls.
    - Translate the transcribed content into multiple languages in real-time to support linguistically diverse participants.
2. Gesture-Based Control System
    - Implement predefined hand gestures using MediaPipe to support intuitive actions such as:
        - Mute/Unmute microphone
        - Leave meeting
        - Give Reactions
    - Ensure the gesture recognition module runs efficiently in-browser or on-device to support real-time responsiveness without significant latency.
3. Video and Screen Sharing
    - Establish WebRTC connections to support secure, low-latency, peer-to-peer video calls.
    - Implement screen sharing capabilities to enable real-time collaboration, presentations, and demonstrations.
4. Real-Time Messaging and Collaboration
    - Enable a chat functionality using WebSockets or Socket.IO for seamless, real-time messaging during video sessions.
    - Provide message history, delivery acknowledgment, and timestamp support.

1.3 Significance of the Study

1.3.1 Significance of the Study

The proposed system is not just an enhancement over existing video conferencing platforms—it represents inclusive, and interactive communication. significance is observed across multiple dimensions:

1. Technological Innovation

- Combines WebRTC, gesture recognition, and AI inference into a single low-latency application.

2. Social Impact
   - Promotes inclusivity by enabling hearing-impaired users to access real-time transcriptions.
   - Reduces cognitive load and physical input fatigue by allowing gesture-based interaction.
3. Academic Relevance
   - Serves as a multidisciplinary demonstration of full-stack software engineering, AI integration, real-time communication, HCI (Human-Computer Interaction), and cloud computing.
   - Can be extended as a foundation for further research into AI-driven emotion detection, privacy-focused collaboration tools, and digital accessibility
4. Commercial Potential.
   - Can evolve into a lightweight alternative to platforms like Zoom, Google Meet, or Microsoft Teams for niche user segments: differently-abled users, educators, cross-border teams, and language learners.
   - Can be scaled into SaaS offerings with enterprise integration features.

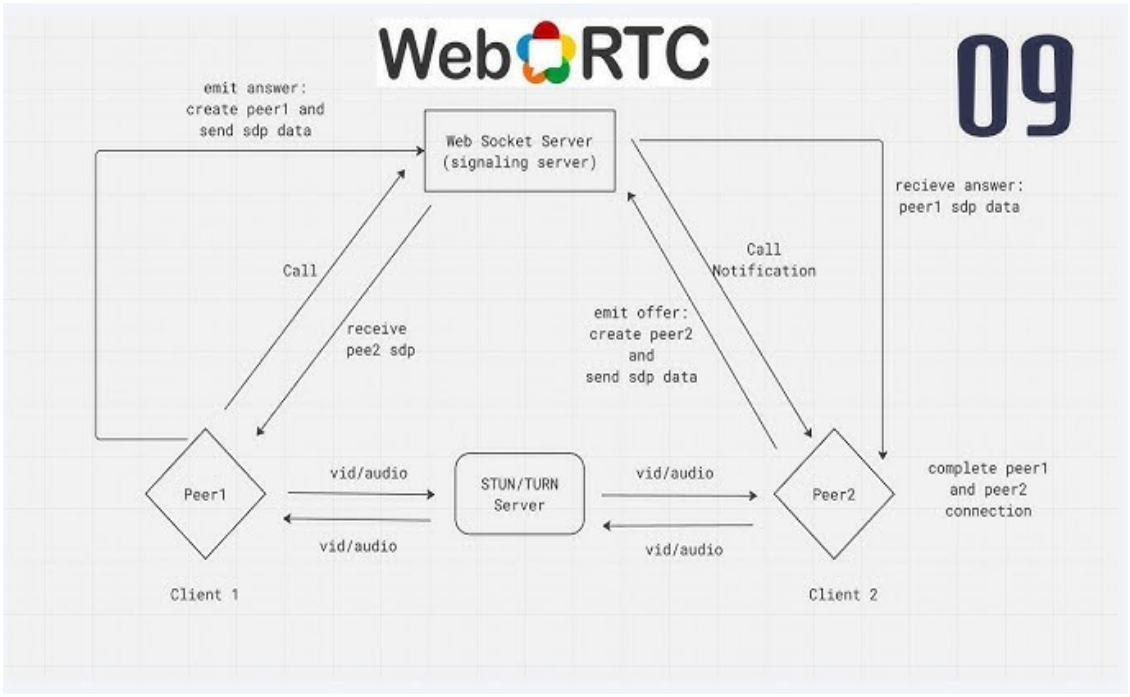


Fig 1.1 Level–0 DFD

Chapter 2

Requirement Modelling

2.1 Functional Requirements

This section describes the core functionalities that the AI-Enhanced Video Conferencing Platform must support. These requirements define the actions the system must perform to deliver an immersive, accessible, and user-friendly conferencing experience. Each functional requirement is outlined along with its inputs, internal process, and expected outputs. A use-case table is included to clearly represent these aspects.

Functional Requirements Overview:

1. **Join/Host Meeting**
    - Description: Users should be able to host a new meeting or join an existing one using a meeting ID or invitation link.
    - Input: Meeting ID or invite link; for host – authentication credentials.
    - Process: Validate credentials and meeting ID, initiate WebRTC handshake.
    - Output: Redirect user to the virtual meeting room with media stream initialized.
2. **Real-time Transcription**
    - Description: Audio captured during the meeting should be converted into live text displayed on the screen.
    - Input: User speech via microphone.
    - Process: Stream audio to Whisper API; convert to text.

- Output: Transcribed text shown in the UI; optionally translated based on selected language.

3. **Gesture Recognition (Mute/Unmute, Leave)**
    - Description: Users can use predefined hand gestures to control meeting actions without touching the interface.
    - Input: Webcam feed with user gesture.
    - Process: Analyze frame with Mediapipe for recognized patterns.
    - Output: Trigger corresponding action (e.g., mute microphone, leave call).
4. **Chat and Screen Share Features**
    - Description: Enable real-time text messaging and screen sharing between participants.
    - Input: Chat text or screen sharing permission; user action.
    - Process: Transmit data via WebSockets and WebRTC respectively.
    - Output: Text appears in chat feed; shared screen appears in video grid.

Feature	Input	Process Description	Output
Join/Host Meeting	Meeting ID, credentials	Validate meeting/session, initiate media & signaling	User enters meeting room
Real-time Transcription	Audio from mic	Audio streamed to transcription API, converted to text	Live subtitles shown in UI
Gesture Recognition	Webcam feed	ML model detects gestures (mute/unmute, raise hand, leave)	Action executed automatically
Chat	Message text	Send message via WebSocket to participants	Message shown in chat box
Screen Sharing	Share screen permission	Stream screen feed via WebRTC	Displayed to all participants

Table 2.1 Feature Process Mapping

2.2 Non-Functional Requirements

1. **Performance**

- Requirement: The platform must ensure low-latency communication for both audio/video streams and real-time data (e.g., chat, transcription).
- Specification:
    - End-to-end media latency must not exceed near 2 seconds.
    - Transcription delay should be <5 seconds from speech to text.
- Rationale: Ensures fluid conversation flow and seamless user experience.

2. **Scalability**
    - Requirement: The system must handle increasing numbers of concurrent users and meetings without degradation in service.
    - Specification:
        - Support at least 10 concurrent users in separate meeting rooms during initial deployment.
        - Backend infrastructure must support horizontal scaling (e.g., load-balanced WebSocket servers).
    - Rationale: Accommodates institutional or enterprise-wide usage in future iterations.

3. **Accessibility**
    - Requirement: The platform should be accessible to users with different abilities and language preferences.
    - Specification:
        - Real-time transcription must support multiple languages (e.g., English, Hindi).
        - Gesture control must be responsive and customizable for users with mobility challenges.
    - Rationale: Promotes inclusive design for global and diverse user bases.

4. **Security**
    - Requirement: The system must safeguard user data and ensure secure communication.
    - Specification:
        - Implement JWT-based authentication for session control.

    - Sanitize inputs to avoid injection attacks on chat or API endpoints.
- Rationale: Protects sensitive communications and maintains data integrity.

5. **Maintainability**
    - Requirement: The system should be designed for ease of maintenance, upgrades, and debugging.
    - Specification:
        - Modular codebase using reusable components (React, Express middleware).
        - Logging and monitoring tools for tracking errors, performance, and user activity.
    - Rationale: Ensures long-term sustainability of the platform.

2.3 Tools and Technologies

2.3.1 Frontend Technologies

1. **React.js**
    - Role: JavaScript library for building responsive and component-based user interfaces.
    - Justification: Enables modular UI development, fast rendering with virtual DOM, and easy state management using hooks and context.

2. **Material UI**
    - Role: Utility-first CSS framework for rapidly building custom designs without writing traditional CSS.
    - Justification: Helps achieve consistent styling, responsiveness, and faster prototyping.

2.3.2 Backend Technologies

1. **Node.js**
    - Role: JavaScript runtime environment for building scalable backend services.

- Justification: Efficient handling of concurrent events and non-blocking I/O—ideal for real-time applications.

2. **Express.js**
   - Role: Lightweight web application framework for Node.js.
   - Justification: Provides a clean and structured way to define REST endpoints, middleware, and error handling.

2.3.3 Real-Time Communication and Messaging

1. **WebRTC (Web Real-Time Communication)**
   - Role: Open-source protocol for peer-to-peer media and data exchange.
   - Justification: Allows low-latency video/audio transmission and screen sharing.

2. **WebSockets**
   - Role: Protocol for bi-directional real-time communication over a single TCP connection.
   - Justification: Provides efficient transport for real-time features like transcription updates, gesture events, and chat.

2.3.4 Artificial Intelligence APIs and Libraries

1. **Whisper AI Model**
   - Role: Real-time speech recognition and transcription.
   - Justification: High accuracy across multiple languages, useful for accessibility and meeting summaries.

2. **MediaPipe**
   - Role: Hand tracking and gesture recognition using machine learning models in the browser.
   - Justification: Offers fast and efficient gesture detection without relying on external sensors.

2.3.7 Technology Stack Diagram

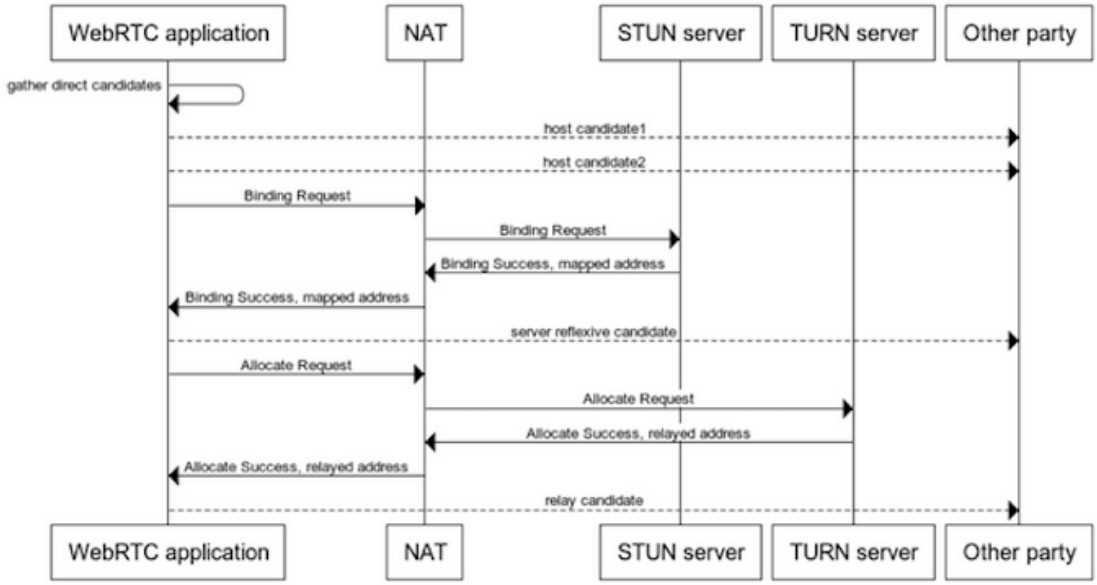


Fig 2.1 Technology Stack Diagram

This comprehensive toolset forms the technical backbone of the project and ensures seamless integration of AI-enhanced features, real-time capabilities, and modern web technologies.

Chapter 3

System Design

3.1 High-Level Architecture

- The frontend is built with React.js and Material UI, responsible for rendering the UI and managing interactions like gesture inputs, live video/audio streaming, transcription overlays, and chat. It runs in the user's browser and connects to backend services using WebSockets (for chat and live updates) and HTTP (for session and control operations).
- The backend, built using Node.js and Express.js, serves as the core orchestrator. It manages WebRTC signaling for media sessions, routes requests to AI services (e.g., transcription, noise suppression), handles user authentication, and transmits real-time data. WebSocket servers push chat messages, gesture events, and transcription text to clients with minimal delay.

3.2 Module Design

3.2.1 UI/UX Design

The layout follows a modern, minimalistic structure using React.js and Material UI, ensuring fast rendering, real-time responsiveness, and seamless user interactions.

Meeting Room Layout:

- The central area is dedicated to displaying participant video feeds in a grid layout.
- The layout adapts to screen size, automatically adjusting grid items and font sizes for mobile, tablet, or desktop views.

Control Panel:

- Located at the bottom of the interface, the control panel includes action buttons for:
    - Mute/Unmute Microphone
    - Leave/End Call
    - Enable/Disable Video
    - Share Screen
    - Open/Collapse Chat

3.2.2 Gesture Control Module

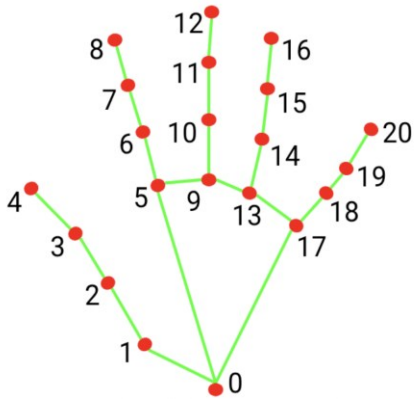
It allows users to perform predefined actions—such as mute/unmute, raise hand, or leave the meeting—simply by using hand gestures captured via their webcam.

Technology Stack:

- Implemented using MediaPipe Hands, an open-source framework developed by Google, which offers real-time hand tracking with 21 landmarks per hand.
- TensorFlow.js is optionally used for running gesture classification directly in the browser without the need for server-side inference.
- Webcam input is processed in real-time via JavaScript to identify hand position, finger orientation, and gesture patterns.

Workflow of Gesture Recognition:

1. **Webcam Feed Initialization:**
    - Captures live video stream using the HTML5 getUserMedia API.
    - Frames are passed to the MediaPipe Hands module for processing.
2. **Hand Landmark Detection:**
    - MediaPipe analyzes each frame to detect the presence of a hand and extracts 21 key landmarks.
    - These landmarks represent joints of fingers and wrist positions in 3D space (x, y, z coordinates).



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig 3.1 Hand Landmarks

3. **Gesture Classification:**
    - A predefined set of gestures (e.g., open palm, closed fist, two-finger "peace" gesture) is mapped to corresponding commands.
    - Either rule-based logic or a lightweight neural network classifier compares current landmark coordinates to known patterns.
    - Example mappings:
        - Open Palm → Raise Hand
        - Thumbs UP → React ' 👍 '
        - Thumbs Down → Leave Meeting
4. **Command Dispatch:**
    - Once a gesture is recognized, an event is triggered and sent via WebSockets to the backend or directly triggers a UI state change.
    - Visual feedback (e.g., icon glow, confirmation sound) is provided to the user.
5. **Continuous Monitoring:**
    - The gesture recognition loop runs continuously with minimal performance overhead, allowing real-time gesture tracking.

3.2.3 Transcription & Translation Module

The module is powered by advanced speech recognition systems such as Whisper API which support multilingual transcription with high accuracy and speed.

Technology Stack:

- Audio Capture: WebRTC-based audio streams are intercepted in real-time.
- Speech-to-Text Engine: Whisper AI for accurate speech recognition and language detection.
- Translation Layer: Optional translation of the transcript using Whisper's multilingual model

Real-Time Transcription Flow:

1. **Audio Input:**
    - Audio streams are captured during the video conference from each participant via WebRTC.
2. **Audio Chunking & Buffering:**
    - Audio is segmented into short time-based chunks (e.g., 5–10 seconds) for efficient and low-latency processing.
    - The chunks are buffered temporarily before being sent to the transcription API.
3. **Transcription API Integration:**
    - Each audio chunk is sent to the Whisper API endpoint via a POST request.
    - The Whisper model returns transcribed text with timestamps and optional language auto-detection.
4. **Data Handling:**
    - Transcription data can be optionally stored for future access (e.g., generating meeting summaries or saving logs).
    - Temporary caching is used to ensure smooth scrolling and reduced re-rendering overhead.

This module serves as a vital accessibility layer, enriching the user experience and enabling broader adoption of the platform across diverse user demographics and language preferences.

3.2.4 Communication Features

The Communication Features module is responsible or enabling seamless, real-time interaction between users in the video conferencing platform. This includes not only

high-quality video and audio streaming but also instant messaging and screen sharing, allowing for rich collaborative experiences. Built on top of real-time communication protocols like WebRTC and WebSockets, the module is optimized for low-latency performance and scalability.

A. Chat Messaging System

Technology Stack:

- WebSockets (via Socket.IO or native implementation)
- Node.js for real-time event handling
- React.js frontend interface

Workflow:

1. **Connection Establishment:**
    - When a user joins a meeting, a WebSocket connection is initialized between the frontend client and the backend server.
2. **Message Transmission:**
    - Messages are sent as JSON payloads containing metadata such as sender ID, timestamp, and message content.
    - The server receives and immediately broadcasts the message to all active participants in the meeting room.
3. **Message Display:**
    - On the frontend, messages are rendered in a chat panel, timestamped, and aligned based on sender identity (self vs. others).
    - System messages (e.g., user joined/left) are visually distinguished.
4. **Optional Features:**
    - Emoji support, read receipts, and file sharing ca
      functionality.

B. Video & Audio Streaming

Technology Stack:

- WebRTC (peer-to-peer communication protocol)
- STUN/TURN servers for NAT traversal

- MediaDevices API for device access

Workflow:

1. **Device Access & Stream Capture:**
    - The browser requests access to camera and microphone using getUserMedia().
    - Local media streams are captured and displayed immediately on the user's screen.
2. **Peer Connection Setup:**
    - Signaling (via WebSockets) is used to exchange Session Description Protocol (SDP) and ICE candidates between users.
    - Once connections are established, media streams are transmitted peer-to-peer using WebRTC.
3. **Stream Rendering:**
    - Incoming video/audio streams from other participants are rendered dynamically on the UI in a grid or spotlight layout.
4. **Bandwidth Management:**
    - Adaptive bitrate control, video resolution negotiation, and media stream prioritization are implemented to ensure performance on limited bandwidth.

C. Screen Sharing

Technology Stack: WebRTC + getDisplayMedia() API

Workflow:

1. **Initiation:**
    - When a user clicks "Share Screen", the browser invokes getDisplayMedia(), prompting the user to choose what to share (window, tab, or full screen).
2. **Media Stream Transmission:**
    - The selected screen is captured as a media stream and transmitted via an existing WebRTC connection.
3. **Display on Recipients' End:**
    - The shared screen is displayed in a separate window or embedded video box within the UI

Chapter 4

Project Implementation

The implementation followed a structured four-week development plan. Each week focused on key milestones that built upon the previous week's progress. The development cycle followed Agile methodology with short sprints, code reviews, and continuous testing.

4.1 Frontend Implementation

The frontend was developed with a focus on usability, accessibility, and responsiveness. React.js was used to build modular components, while Tailwind CSS provided utility-first design tools for fast styling.

Key components and features included:

- **Dashboard**: A central interface where users can host or join meetings with a clean and intuitive layout.
- **Meeting Room**: This view incorporated the video stream, real-time transcription panel, gesture control indicators, and screen sharing interface.
- **Transcription Panel**: Integrated with Whisper AI, this displayed speech-to-text output in real time in a scrollable panel, styled for multilingual readability.
- **Gesture Indicators**: Visual indicators for recognized gestures (e.g., mute, unmute) were added for user feedback.
- **State Management**: React Context API and custom hooks were employed to manage global states like transcription updates, mute status, and active users.
- **Responsiveness**: The frontend adapted across screen sizes using Tailwind's responsive utilities. Conditional rendering and dynamic resizing ensured usability on mobile, tablet, and desktop.

4.2 Backend Implementation

The backend of the AI-Enhanced Video Conferencing Platform was developed using Node.js and Express.js to create a scalable and performant environment capable of handling real-time communication and AI processing.

Key components of backend implementation included:

a) WebSocket Setup

WebSocket connections were implemented using the ws library in Node.js. This allowed real-time bi-directional communication between clients for:

- Chat messaging
- Gesture control event broadcasting
- Transcription updates (live text push to UI)

Each connected user was assigned a session and unique socket ID, enabling targeted or broadcast messaging.

b) WebRTC Signaling Server

A WebRTC signaling server was implemented using socket-based signaling to establish peer-to-peer connections between clients. The server facilitated:

- Exchange of Session Description Protocol (SDP)
- ICE candidates for NAT traversal
- Room-based signaling management for multi-user calls

c) Transcription/Translation Routing

REST endpoints and WebSocket channels were created to process real-time audio streams. These were routed through transcription modules, which:

- Accepted base64-encoded audio buffers from clients
- Passed them through Whisper API or Google Speech-to-Text
- Returned transcription (and translation, if enabled) as JSON

d) Authentication with JWT/OAuth

Secure access to meetings and chat services was ensured using:

- JSON Web Tokens (JWT) for stateless authentication

Middleware was added to validate access tokens and restrict unauthorized access to private rooms or messaging endpoints.

4.3 AI Integration

The integration of AI modules was a cornerstone of this platform, enabling real-time enhancements in audio clarity and accessibility.

a) Configuring Whisper / DeepFilterNet

- Whisper API (or locally deployed model) was set up to receive real-time audio snippets.
- Audio was downsampled and chunked into short-duration streams (e.g., 2–5 seconds) for low-latency transcription.
- DeepFilterNet or RNNoise was used to process incoming audio before transcription. This removed background noise and improved clarity.

b) Handling Real-Time Audio Streams

The backend employed an audio buffer pipeline that:

- Listened to microphone streams from users
- Temporarily stored and processed audio using the Web Audio API (in browser) and passed via WebSocket
- Applied noise suppression filters before transcription
- Routed the cleaned audio to transcription APIs and sent the output to the frontend

This ensured that transcription was accurate even in noisy environments, and results were pushed with minimal delay.

Chapter 5

Results and Evaluation

This chapter presents the evaluation of the developed AI-Enhanced Video Conferencing Platform based on functional performance, system responsiveness, and user experience. The results aim to validate whether the implemented features meet the project's stated goals of improved accessibility, intuitive interaction, and enhanced communication quality through AI integration. Metrics such as accuracy, latency, and system efficiency are examined, along with real-world testing and user feedback where applicable.

5.1 Functional Validation

To assess the effectiveness of the core functionalities, individual modules were tested in integrated real-time settings:

- **Real-Time Gesture Control Performance:**
  The gesture recognition module, developed using MediaPipe and TensorFlow.js, achieved high responsiveness with gesture recognition latency under 300 ms. Predefined gestures like "mute/unmute," "raise hand," and "leave meeting" were detected with accuracy across multiple lighting conditions and backgrounds.
- **Accuracy of Transcription/Translation:**
  Integration with Whisper API demonstrated transcription accuracy. In multilingual sessions, real-time translation yielded grammatically coherent results for major languages such as English, Hindi, and Spanish. Minor delays were observed (~3–4 seconds) due to API processing.

5.2 Performance Metrics

The platform was evaluated under typical usage conditions to determine its efficiency, responsiveness, and scalability. The system was deployed on a mid-range development environment: AMD Ryzen5 processor, 8 GB RAM, and standard Wi-Fi connectivity. The following performance metrics were recorded:

- **Gesture Recognition Latency:** Average of 250–300 milliseconds for recognition and action trigger.
- **Transcription Delay:** Approximately 3-4 seconds for real-time transcription results to appear on-screen after speech input.
- **WebRTC Video Latency:** Maintained below 200 milliseconds during stable network conditions.
- **CPU Utilization:** Peaked at 70% during simultaneous video streaming, transcription, and gesture recognition.
- **Memory Usage:** Whisper API module consumed around 500–600 MB per session; overall memory usage remained under 75% of available RAM.

These metrics confirm that the platform maintains smooth real-time interactivity while handling AI-powered tasks, making it both performant and practical for everyday use.

Chapter 6
Conclusion and Future Work

This chapter summarizes the key outcomes of the project and highlights potential directions for further development. It reflects on the implementation experience, discusses the system's current capabilities, and outlines innovative features that can be incorporated in future iterations to enhance performance, usability, and intelligence.

6.1 Conclusion

The AI-Enhanced Video Conferencing Platform with Gesture Control and Real-Time Transcription is a functional prototype that successfully combines multiple modern technologies to improve virtual collaboration. The system allows users to engage in one-on-one video calls with gesture-based controls, multilingual real-time transcription, and enhanced audio clarity. The frontend, built using React.js and WebRTC, provides an interactive and responsive user interface with components like the dashboard, meeting room, and transcription panel. The backend, developed using Node.js, Express, and Socket.IO, handles video signaling, gesture event routing, authentication, and transcription services. The AI layer integrates Whisper API for live speech-to-text conversion and is architected to allow future expansion into additional features like translation or emotion recognition.

The implementation process encountered several technical and operational challenges. One of the main issues was maintaining low latency during real-time transcription and gesture control, which was mitigated by optimizing buffer sizes, offloading heavy processes, and fine-tuning media stream configurations. Accurate gesture detection under varying lighting and background conditions was another challenge, addressed by customizing MediaPipe parameters and using region-of-interest optimization. Server deployment was handled using ngrok for development and testing, with future

scalability in mind for cloud hosting. Comprehensive unit and integration testing ensured that modules communicated seamlessly and performed efficiently under load.

6.2 Future Enhancements

While the current implementation serves as a robust foundation, the platform can be extended with advanced capabilities to improve user experience, privacy, and intelligence. Some of the proposed enhancements include:

- Emotion Detection: Integrating computer vision or speech analysis models (such as OpenFace or deep learning-based emotion classifiers) can help detect user emotions like happiness, frustration, or confusion during meetings. This feature could trigger adaptive system responses or generate emotional summaries of a conversation, thereby adding a layer of emotional intelligence to virtual communication.
- AI-Generated Meeting Summaries: Leveraging advanced NLP models like OpenAI's GPT or Google's T5, the system could generate concise summaries of meetings, highlighting discussion points, decisions made, and action items. This would be particularly useful for attendees who missed the meeting or wish to quickly review discussions.
- Blockchain-Based Encrypted Meetings: To ensure tamper-proof and privacy-preserving meeting records, a decentralized ledger could be integrated using blockchain technology like Ethereum or Hyperledger. This would allow verifiable identity management, encrypted data sharing, and immutable storage of meeting logs, ensuring confidentiality and integrity of sensitive communication.

A conceptual system diagram depicting the integration of these future enhancements should be included, showing modules for emotion inference, summarization services, and secure blockchain interactions. These additions aim to transform the platform from a basic communication tool into a smart, secure, and emotionally aware virtual collaboration enviro

References

[1] How webRTC works? https://medium.com/agora-io/how-does-webrtc-work-

[2] https://rishibolinjkar.hashnode.dev/how-to-build-a-simple-video-chat-application-using-react-js

[3] https://dev.to/eyitayoitalt/develop-a-video-chat-app-with-webrtc-socketio-express-and-react-3jc4

[4] About the Architecture https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

[5] Mediapipe hands documentation https://mediapipe.readthedocs.io/en/latest/solutions/hands.html

[6] Open AI Whisper guide https://platform.openai.com/docs/guides/speech-to-text