

Real-Time Sign Language Chat Application with Speech Output

Bidish Kumar Nayak
Student, Dept. of CSE
GIFT Autonomous, Bhubaneswar
Odisha, India

Bibhuti Kumar Giri
Student, Dept. of CSE
GIFT Autonomous, Bhubaneswar
Odisha, India

Saudamini Samantaray
Assistant Professor, Dept. of CSE
GIFT Autonomous, Bhubaneswar
Odisha, India

Abstract—The proliferation of digital communication technologies has dramatically transformed human interaction, yet individuals with hearing and speech impairments continue to face significant barriers in mainstream communication systems. Sign language remains the primary mode of communication for the deaf and hard-of-hearing community; however, the vast majority of people outside this community are unable to comprehend it. This communication gap necessitates the development of intelligent, technology-driven bridge systems that can translate sign language into readable and audible formats in real time. Traditional solutions such as human interpreters are expensive, logistically complex, and unavailable in many situations.

Recent advances in artificial intelligence, computer vision, and real-time web technologies have opened new possibilities for automated sign language translation. Technologies such as Google's MediaPipe hand landmark detection framework, WebSocket-based real-time communication, and neural text-to-speech synthesis have matured to the point where accurate, low-latency sign language recognition and translation is practically achievable in browser-based applications. However, many existing automated translation systems suffer from critical limitations: they require specialized, expensive hardware such as data gloves or motion capture suits, are designed as standalone offline applications rather than integrated communication platforms, support only a limited vocabulary, and lack real-time performance necessary for fluid conversation.

This project presents the design and implementation of a Real-Time Sign Language Chat Application with Speech Output—a full-stack AI-powered system that enables seamless communication using hand gesture recognition. The gesture recognition engine is built using MediaPipe's hand landmark detection framework, which tracks 21 key points on the human hand in real time through a webcam feed. A rule-based classifier processes the geometric relationships between these landmarks to accurately identify 55 distinct gesture categories: 26 American Sign Language (ASL) alphabetic letters, 10 numeric gestures, and 19 special phrase gestures including greetings, expressions, and commonly used functional words.

Keywords— Sign Language Recognition, MediaPipe, Django Channels, WebSockets, Text-to-Speech, Full-Stack Web Development, Computer Vision, Assistive Technology

I. INTRODUCTION

A. Background

Communication is one of the most fundamental aspects of human existence. For the approximately 70 million deaf people worldwide and the millions more with hearing or speech impairments, sign language serves as the primary mode

of expression and interaction. Sign languages are complete, natural linguistic systems with their own grammar, syntax, and vocabulary, varying by country and region. American Sign Language (ASL) is among the most widely studied and used sign language systems globally. Despite the richness of sign language as a communication modality, the vast majority of hearing individuals lack the knowledge or training to understand it. This creates a persistent communication barrier that affects the social integration, educational access, and employment opportunities of the hearing-impaired community.

Recent advances in artificial intelligence, computer vision, and real-time web technologies have opened new possibilities for automated sign language translation. Technologies such as MediaPipe (Google's hand landmark detection framework), WebSocket-based real-time communication, and neural text-to-speech synthesis have matured to the point where accurate, low-latency sign language recognition and translation is practically achievable in browser-based applications. This project leverages these technologies to build a comprehensive, full-stack Real-Time Sign Language Chat Application that bridges the communication gap between sign language users and the broader population.

Furthermore, the growing ecosystem of open-source AI tools—particularly MediaPipe, TensorFlow.js, and gTTS—makes it increasingly feasible for academic and small-scale projects to achieve results that were previously only accessible to well-funded research laboratories. This project demonstrates the democratization of AI for social good. The challenge of building a system that must function accurately, efficiently, and intuitively in real time presents rich engineering and research problems across multiple advanced computing domains.

B. Problem Statement

Existing digital communication platforms—including messaging applications, video conferencing tools, and social networks—are predominantly designed for hearing users. Sign language users are effectively excluded from these platforms unless a human interpreter is present, which is rarely feasible in informal, spontaneous communication scenarios.

Existing automated sign language translation systems suffer from several critical limitations: (1) they require specialized, expensive hardware such as data gloves or motion capture

suits; (2) they are designed as standalone offline applications rather than integrated communication platforms; (3) they support only a limited vocabulary and cannot generate natural, spoken output; and (4) they lack real-time performance necessary for fluid conversation.

There is a clear need for an accessible, webcam-based, real-time sign language communication system that integrates seamlessly into a familiar chat interface and produces both text and speech output without requiring any specialized hardware. Security vulnerabilities, inefficient backend operations, and poor database structures also affect system reliability and maintainability in real-time communication implementations.

C. Objectives

The primary objectives of this project are as follows:

- To develop a real-time hand gesture recognition system capable of detecting and classifying 55 distinct ASL gestures using MediaPipe and rule-based landmark geometry analysis.
- To implement a full-stack web-based chat application with WebSocket-based real-time bidirectional communication, supporting typing indicators, read receipts, and message persistence.
- To integrate a text-to-speech synthesis module that converts recognized gesture sequences into natural-sounding audio output using Google Text-to-Speech (gTTS).
- To design a user-friendly, accessible interface that provides visual gesture feedback, a categorized gesture reference guide, and supports both dark and light themes.
- To implement a secure authentication system using JSON Web Tokens (JWT) with a custom user model supporting profile management and online presence tracking.
- To containerize the complete application using Docker for reproducible, environment-independent deployment.

D. Scope of the Project

The scope of this project encompasses the following components and boundaries:

- **Gesture Recognition Scope:** The system supports 26 ASL alphabet letters (A-Z), 10 numeric gestures (0-9), and 19 special phrase gestures. The recognition is limited to single-hand gestures captured under standard indoor lighting conditions.
- **Platform Scope:** The application is designed as a web-based system accessible through modern browsers (Chrome, Firefox, Edge) on desktop devices. Mobile browser support is planned as a future enhancement.
- **Communication Scope:** The chat system supports one-to-one real-time messaging with gesture-generated text, TTS audio playback, and standard keyboard text input as a fallback.
- **Language Scope:** The current implementation focuses exclusively on American Sign Language (ASL). Adaptation to other sign language systems is beyond the current scope but is architected to be extensible.

- **Deployment Scope:** The system is designed for local and small-scale cloud deployment using Docker containers. Enterprise-grade scalability is identified as a future enhancement.

II. LITERATURE REVIEW

A. Existing Sign Language Systems

Several significant systems and research efforts have attempted to address the sign language translation challenge. Microsoft's research division demonstrated a sign language recognition system using depth cameras and skeletal tracking to detect and classify a vocabulary of several hundred ASL signs. The system achieved recognition accuracies exceeding 90% on its training vocabulary. However, its reliance on depth-sensing hardware (Kinect) makes it inaccessible for general consumer use, and it does not integrate with any communication platform.

Google introduced a sign language detection feature for Google Meet that uses TensorFlow.js and MediaPipe to detect when a participant is using sign language, triggering an automated notification. This system operates in real time in a browser environment without specialized hardware. However, it only detects signing activity rather than recognizing or translating specific signs. SignAll is a commercial system that uses multiple cameras and computer vision to translate ASL into written English in real time. Despite its impressive capabilities, SignAll requires specialized multi-camera hardware setups and is priced for enterprise use, making it inaccessible for individual users.

Early sign language recognition systems relied on data gloves equipped with sensors to measure finger flexion angles. While highly accurate, these systems require users to wear specialized hardware, which is intrusive and expensive. The dominant research direction in recent years has moved toward vision-based systems that use RGB cameras—including standard webcams—as the sole input device. Deep learning approaches, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated state-of-the-art performance on sign language recognition tasks. However, CNN-based approaches require large labelled training datasets and substantial computational resources for inference.

B. Modern Web and AI Frameworks

Google's MediaPipe framework provides efficient real-time hand landmark detection that tracks 21 3D key points on the hand from a standard RGB webcam. This approach offers several advantages: it runs efficiently on CPU-only hardware, requires no training data from the target domain, and provides a stable set of geometric features that can be analysed using rule-based or lightweight machine learning classifiers.

Real-time communication in web applications has evolved significantly with the introduction of WebSocket technology. WebSockets provide a persistent, full-duplex communication channel between client and server over a single TCP connection, enabling true real-time data exchange without the

overhead of HTTP polling. Django Channels extended the Django web framework—traditionally limited to synchronous HTTP request-response patterns—to support asynchronous WebSocket connections through the ASGI (Asynchronous Server Gateway Interface) specification. Django Channels uses Redis as a channel layer backend, enabling scalable, multi-process WebSocket handling.

The gTTS (Google Text-to-Speech) library provides a straightforward Python interface to the Google Translate TTS API, generating natural-sounding speech from text in multiple languages. While commercial alternatives offer more configuration options, gTTS provides adequate quality for the assistive communication use case with minimal complexity.

C. Research Gap

Despite significant progress in individual component technologies, the literature reveals a clear gap: there is no existing open-source, accessible system that integrates all of the following capabilities in a single platform:

- Real-time, webcam-based ASL gesture recognition without specialized hardware requirements.
- Integration with a full-featured, persistent messaging platform rather than operating as a standalone translation tool.
- Text-to-speech output that converts recognized gesture sequences into audio, making the communication accessible to hearing users who are not looking at the screen.
- A user-friendly interface with an integrated gesture reference guide, category-based detection modes, and real-time visual feedback.
- A complete DevOps solution with containerized deployment for reliable, reproducible setup.

This project addresses the identified research gap by delivering a complete, integrated sign language chat application that combines state-of-the-art computer vision, a production-grade web framework, real-time WebSocket communication, TTS synthesis, and an accessible interface.

III. SYSTEM OVERVIEW

A. Proposed System

The proposed system is a Real-Time Sign Language Chat Application that enables users to communicate through hand gestures captured by a standard webcam. The system translates gestures into text and speech, integrating this capability into a familiar chat messaging interface. The application supports two types of users: sign language users who communicate via gesture input, and text-based users who receive gesture-generated messages as text and audio. Both user types interact through the same interface, with the gesture input panel available as an optional input modality.

The system comprises five primary subsystems that operate in concert: (1) the gesture capture and preprocessing module, (2) the gesture recognition and classification engine, (3) the real-time messaging backend, (4) the text-to-speech synthesis module, and (5) the web-based user interface. These subsystems are supported by a persistent data layer (MySQL),

a message broker (Redis), and a containerized deployment environment (Docker).

B. System Architecture

The system follows a three-tier client-server architecture enhanced with WebSocket capabilities for real-time communication:

- **Presentation Layer (Frontend):** Consists of HTML5, CSS3, and JavaScript served by Django's template engine. The interface includes the chat window, gesture capture panel, gesture reference guide, and authentication forms. The webcam feed is accessed through the browser's MediaDevices API, and frames are transmitted to the backend gesture recognition API via HTTP POST requests encoded as base64 image data.
- **Application Layer (Backend):** Implemented in Python using the Django framework, configured with the ASGI server (Daphne) to support both synchronous HTTP requests and asynchronous WebSocket connections simultaneously. Django Channels provides the WebSocket routing and consumer logic, while REST API endpoints handle authentication, message retrieval, and gesture detection under JWT security.
- **Data Layer:** Consists of a MySQL relational database for persistent storage of user profiles, messages, conversation metadata, and gesture event logs. Redis serves as the channel layer backend for Django Channels, managing WebSocket group membership and message routing between consumer instances.

C. Key Features

The system integrates several advanced functionalities to optimize communication and accessibility:

- **Real-Time Gesture Capture:** Accesses local user webcam media streams to perform rapid, high-frequency frame capture arrays.
- **Multi-Category Gesture Engine:** Evaluates 55 independent gesture mapping conditions distributed across letters, numbers, and custom phrases.
- **Dual Modality Translation:** Dispatches concurrent text streaming representations accompanied by asynchronous TTS speech synthesis audio routes.
- **Familiar Chat Canvas:** Renders a clean UI complete with theme selectors, typing triggers, read receipts, and a tabbed visual lookup panel.

D. User Modules

The platform manages interactions across defined user profiles:

- **Sign Language User Module:** Provides access to authorization systems, webcam configuration toggles, categorization constraints dropdown matrices, and visual text input string compilation buffers.
- **Text User Module:** Enforces access control layers to engage in standard keyboard text input streams while enabling immediate rendering loops for incoming gesture-translated strings and audio payloads.

- **Administrator Module:** Isolates system oversight permissions to supervise master user registers, audit Web-Socket thread channels, and adjust detection settings parameters.

IV. METHODOLOGY

chains, the algorithm calculates finger extension states by comparing the fingertip landmark Y_{tip} against its corresponding proximal interphalangeal joint Y_{pip} :

$$E_{finger} = \begin{cases} 1 & \text{if } Y_{tip} < Y_{pip} - \tau \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

A. Workflow of the System

The application executes an automated, multi-tiered pipeline that orchestrates webcam image streams, geometry classification, and real-time message dissemination:

- The user activates gesture capture mode in the browser. JavaScript accesses the webcam via the MediaDevices API.
- At configurable intervals (150ms–200ms), a frame is captured from the video stream and encoded as a base64 JPEG string.
- The frontend sends an AJAX POST request to the `/api/detect-gesture/` endpoint with the base64 frame data.
- The Django view decodes the frame into a NumPy array, performs BGR to RGB color conversion, and passes it to the GestureDetector service.
- The engine processes the frame through the MediaPipe pipeline and returns the classified gesture name and category badge.
- The frontend accumulates the response into a local text buffer and updates the UI layout dynamically.
- Upon pressing the Space key or Send button, the accumulated string is sent as a WebSocket message to the ChatConsumer.
- The consumer persists the payload to MySQL, invokes async gTTS routines, and broadcasts the completed message structure to all active clients in the conversation group via Redis.

B. Authentication Process

Authentication is implemented using Django REST Framework SimpleJWT. The registration endpoint validates username, email uniqueness, and password strength. Upon successful login via the token pair view, the server yields JWT access tokens expiring in 15 minutes and refresh tokens valid for 7 days.

The frontend uses an Axios interceptor to capture 401 responses, automatically requesting a new access token via the refresh route to sustain an uninterrupted real-time messaging session. User presence flags (`is_online`) and corresponding timestamps are updated via Django signals tied directly to the lifecycle of the consumer's socket handshake.

C. Gesture Classification Process

The classification engine maps hand anatomy coordinates into discrete categories using a rule-based geometric approach over MediaPipe landmarks. MediaPipe tracks 21 distinct 3D anatomical points. For the index, middle, ring, and pinky finger

where τ represents a calibrated distance threshold. The thumb utilizes horizontal plane checking (X_{tip} vs. X_{ip}) due to its structural orientation.

The evaluation outputs a five-element binary vector representing extension profiles. A closed fist `[0, 0, 0, 0, 0]` maps to letter 'A' or 'E' depending on thumb proximity conditions, while an open palm `[1, 1, 1, 1, 1]` maps to number 5. Inter-finger angles, palm orientations, and spatial distances resolve tracking ambiguities.

D. Payment Processing Methodology

This subsection is adapted to represent the text-to-speech media generation tracking routine utilized within the messaging flow instead of commercial monetary checkouts. The text-to-speech pipeline takes compiled message strings and translates them into streaming media assets using gTTS.

When a gesture-generated string is committed via the AsyncWebsocketConsumer, the backend schedules an asynchronous task using thread capabilities to safely hit the Google Translate TTS API without obstructing core WebSocket routing loops. The API yields a unique MP3 file saved to persistent media directories. The relative static file path URL is appended into the broadcast payload, allowing recipient frontends to instantly stream the audio asset on demand using native browser media elements.

E. Use Case Diagram Description

The use case configuration establishes the functional scope across all actors interacting with the system application. The Sign Language User interacts with account registry, webcam frame processing, token authentication, and gesture string buffering use cases. The Text User engages with message views, text input loops, and TTS playback triggers. The Admin interacts with interface dashboards, rate adjustments, and connection auditing screens.

V. SYSTEM DESIGN

A. Use Case Diagram Description

The use case configuration emphasizes behavioral separation between assistive sign-input pipelines, standard keyboard inputs, and background administration settings. It models authentication wrappers across all state-changing activities to fulfill system compliance metrics.

B. Database Design

The database schema utilizes a relational design enforced via MySQL to maintain data integrity and performance up to Third Normal Form (3NF) requirements. The tables are structured as follows:

- **Users Table:** Manages primary login keys, email strings, password hashes, avatars, online flags, and last-seen temporal logs.
- **Conversations Table:** Tracks persistent interaction rooms mapped across participants using a structural junction entity mapping framework.
- **Messages Table:** Mapped via foreign keys to parent conversations, containing content fields, message type descriptors (text/gesture/audio), raw gesture metadata blobs, audio path links, and boolean read state flags.

- **Chatbot Rules / Reference Table:** Stores static node configurations, visual guidance assets, definition lookups, and mapping identifiers.

C. Data Flow Diagram

The Data Flow Diagram charts structural data transformations across consecutive processing zones:

- **Level 0 (Context Diagram):** Sets the system perimeter, capturing webcam video arrays from signing nodes and rendering message strings and synthesized audio objects across recipient targets.
- **Level 1 (Operational Decomposition):** Tracks validation paths where client-side JavaScript handles local frame capture and encoding before backend views invoke sanitization, database CRUD lookups, and Redis group distribution channels.
- **Level 2 (Predictor Deep-Dive):** Focuses exclusively on the gesture detection engine, illustrating how raw image blocks pass through OpenCV conversions and MediaPipe landmark mapping arrays to emit JSON responses back to frontend compilation buffers.

D. Class Diagram

The object-oriented structure organizes system logic into distinct components. The `CustomUser` class encapsulates authentication states and ties directly to the `Conversation` aggregator. The `Message` model handles multi-format data and tracks read receipts via simple flags.

The active runtime operations are driven by service classes: `GestureDetector` handles frame landmark conversion and rule metrics parsing, while `ChatConsumer` maintains client connections and routes asynchronous events across the Redis channel layer.

VI. IMPLEMENTATION

A. Frontend Implementation

The presentation layer is implemented using HTML5, CSS3, JavaScript, and Tailwind CSS templates. Layouts are structured through Django template inheritance centered on a master configuration template. Theme profiles switch dynamically via body data attributes and persist choices across browser storage targets. The visual thread renders bubble blocks based on identity keys, showing custom category badges and speaker icons for translated elements.

Backend Implementation

The application logic runs on Python 3.11 with Django 4.2 and Django REST Framework. Asynchronous workflows deploy via a Daphne ASGI server initialization. URL routers divide incoming requests: standard API requests pass down traditional view routes, while WebSocket channels route directly into specific consumer loops. Security controls filter all traffic, verifying authentication tokens via clear headers.

B. Database Implementation

The storage architecture is managed on a MySQL database service. Performance indexes optimize lookup tasks over user strings, keys, and tracking parameters. The

system sets database flags to drop child objects automatically through cascading parameters whenever parent profiles are deleted.

C. Asynchronous Integration Engine

Real-time message routing is handled by the Redis service acting as a channel broker for group data sharing. Media processing jobs call the gTTS API to generate audio files, returning file link objects back into the consumer loops.

D. Admin Dashboard and Containerization

Administrators monitor active instances, user accounts, and token states through an admin panel interface. The complete stack is containerized into microservices using Docker and orchestrated with Docker Compose. The container image copies local setups and resolves external environment variables from hidden configuration scripts.

VII. RESULTS AND ANALYSIS

A. Authentication Results

Testing over 20 independent profiles verified the security workflows. The system correctly blocks mismatched parameters, manages strength constraints, and drops expired tracking elements with appropriate server responses. Token updates seamlessly refresh active sessions via frontend interceptors.

B. Real-Time Communication Benchmarks

Message routing tests on local networks measured an average message latency of 47ms. Typing status displays updated smoothly within specified limits, and read receipts changed state accurately upon interface focus.

C. Gesture Recognition Accuracy Analysis

The tracking model was evaluated under fluorescent lighting conditions across a test group of 5,500 gesture samples performed by five volunteers. The system achieved an overall classification accuracy of 90.3%. The numeric category achieved the highest accuracy (94.1%) due to its highly distinct landmark profiles. Structural phrase mappings showed lower accuracy metrics (86.0%) due to the challenges of capturing movement transitions within static frame rules. Ambiguities between closed shapes (such as letter profiles A, E, and S) accounted for minor misclassification rates.

D. Performance Latency Benchmarks

The application profile was tested on an Intel Core i5 processor machine with 16GB of RAM and no external GPU acceleration. Image tracking tasks completed in an average of 143ms per frame, supporting a processing rate of approximately 7 frames per second.

System load tests verified connection stability up to 127 concurrent WebSocket instances, significantly exceeding the baseline design target of 50 users. Database message query lookups completed in 42ms.

E. Operational Usability Surveys

Usability was evaluated through a structured study with 15 volunteer participants. The system achieved an average System Usability Scale (SUS) score of 78.4, corresponding to a "Good" rating. Approximately 93% of participants reported that the visual reference guide helped them learn and sign character sets effectively.

VIII. DISCUSSIONS

A. Advantages of the System

The architectural design and empirical validation of the Real-Time Sign Language Chat Application demonstrate distinct conceptual and operational advantages over both contemporary standalone algorithmic translation utilities and conventional, text-centric real-time communication frameworks:

- **Ubiquitous Hardware Accessibility:** Unlike historical assistive frameworks that depend heavily on cost-prohibitive, proprietary, or intrusive peripheral hardware—such as cybernetic data gloves, infrared depth-sensing matrices, or multi-camera laboratory arrays [cite: 129, 203, 204]—the proposed system executes entirely via standard consumer-grade RGB webcams integrated into ubiquitous personal computing devices[cite: 133, 208]. By offloading high-fidelity spatial telemetry capture to native browser API streams, the barrier to user onboarding is completely minimized[cite: 93, 255].
- **Highly Granular Core Dataset Mappings:** The platform boasts an engineered vocabulary layer spanning 55 distinct sign classifications[cite: 86, 155]. By carefully balancing 26 alphabetical characters for continuous fingerspelling fallback, 10 numerical expressions for symbolic and data processing usage, and 19 contextual, high-frequency conversational phrase shortcuts[cite: 86, 155], the vocabulary framework effectively addresses complex linguistic needs without requiring heavy localized asset compilation.
- **Direct Workspace Integration Lifecycle:** Standalone translator programs disrupt conversational flow by forcing users to toggle between an isolated visual inference terminal and an independent communication channel[cite: 130]. By embedding the browser-based webcam frame loop directly inside an active WhatsApp-style messaging layout [cite: 90, 393], this architecture establishes a fluid human-computer interaction canvas where gestural inputs

naturally compile into native communication channels without layout disruptions[cite: 584].

- **Dual Cross-Modality Output Pathways:** To fully resolve the communicative asymmetry that exists between signing individuals and non-signing hearing users, the platform emits parallel presentation channels[cite: 245]. Every gesture sequence processed through the backend engine triggers simultaneous text string presentation layers and asynchronous, server-side neural speech synthesis streams[cite: 83, 84, 492]. This mechanism ensures that the message remains fully accessible even when the recipient is not actively viewing the interface canvas[cite: 234].
- **Democratized Open-Source Infrastructure Stack:** By structuring the full-stack blueprint around open-source components—specifically Python, Django Channels, Redis, MySQL, OpenCV, and MediaPipe [cite: 87, 88, 266]—the ecosystem completely removes licensing dependencies and enterprise vendor lock-in challenges[cite: 587]. Organizations can easily fork, containerize, customize, and spin up isolated instances using automated orchestration tools[cite: 94, 149].

B. Limitations

Despite satisfying its baseline functional criteria and clearing structural performance benchmarks under ideal operational environments[cite: 95, 607], deep algorithmic and infrastructure limitations persist within this specific technical framework:

- **Algorithmic Optimization Bias for Static Telemetry:** Because the underlying classifier relies on localized, frame-by-frame geometric calculus over isolated coordinate blocks[cite: 86, 456], it lacks structural temporal awareness. Consequently, complex dynamic signs requiring spatial tracking across explicit time sequences—such as kinetic hand movements or transitions across the signing space [cite: 545, 590]—must be approximated using static poses caught at peak acceleration boundaries, which introduces inherent tracking instability[cite: 590].
- **Single-Hand Kinematic Invariance:** The classification engine's configurations enforce a maximum hand restriction parameter (*max_num_hands* = 1) to preserve low execution latency limits over CPU runtimes[cite: 451, 458]. This constraint restricts structural tracking to single-hand signs, rendering the application incapable of processing bimanual or symmetrical gestures[cite: 592, 593]. Such shapes are either forced into single-hand approximations or dropped entirely from the lexical dictionary[cite: 593].
- **Linguistic Vulnerability to Ambient Illuminance Fluctuation:** System accuracy relies directly on the pixel contrast arrays provided to the MediaPipe processing pipeline[cite: 93, 213]. Empirical testing reveals that while the system maintains high accuracy under controlled fluorescent indoor illumination (60–80 lux) [cite: 542], spatial tracking degrades rapidly when exposed to

low-light environments (< 30 lux) or back-lit environments[cite: 558, 594]. This behavior drastically limits the reliability of the system in outdoor or poorly lit rooms[cite: 594].

- **External Network Dependency for Audio Processing:** The text-to-speech engine relies on the Google Translate API via the gTTS package to synthesize audio streams[cite: 228, 490]. Because file processing requires external server handshakes, the voice playback feature remains completely non-functional during network drops or fully offline operation[cite: 597]. Replacing this with an offline tool requires installing heavy local models that complicate resource utilization[cite: 598].
- **Desktop Layout Constraints and Missing Device Portability:** The presentation layer's elements—such as side-by-side chat panels, floating webcam streams, and large visual reference guides [cite: 393, 398]—are meticulously optimized for horizontal view designs[cite: 157, 599]. This structural layout fails to translate well to vertical mobile browser viewports, which introduces severe scrolling overlaps and layout distortions due to limited screen real estate[cite: 600].

C. Real-World Applicability

The practical architectural choices behind the Real-Time Sign Language Chat Application make it immediately viable for real-world deployment across several inclusive community spaces, service domains, and organizational workflows:

- **Inclusive Workspaces and Enterprise Communication Support:** Corporate environments can deploy the system locally on existing intranet servers to facilitate spontaneous, cross-departmental collaboration between deaf employees and hearing staff members without scheduling certified human interpreters for routine stand-ups.
- **Public Service Access and Healthcare Intake Systems:** Hospitals, emergency response rooms, and reception counters can utilize desktop terminal installations to capture immediate medical intake logs, manage initial patient registration inquiries, and eliminate crucial communication delays during time-sensitive triage processes.
- **Digital Customer Service Points and Helpdesk Infrastructure:** Government service offices, banking halls, and retail helpdesks can incorporate the web application onto touchscreens or customer terminals, allowing signing citizens to handle automated ticket requests independently.
- **Adaptive Educational Environments and Interactive Sign Learning:** Accessible classrooms can employ the system as an active teaching assistant where non-signing peers can engage in peer-to-peer messaging exchanges with signing students while utilizing the built-in gesture reference panels to study ASL mechanics organically.

D. Scalability and Future Improvements

To evolve this system from an academic prototype into an enterprise-ready assistive platform, the system architecture supports several structural roadmap upgrades:

- **Transitioning to Deep Learning Sequence Classifiers:** Future iterations plan to swap the rule-based coordinate math for a hybrid Convolutional Neural Network and Long Short-Term Memory network architecture (CNN-LSTM)[cite: 637]. Training this architecture on high-density data corpuses like WLASL will help the engine expand its vocabulary from 55 simple patterns to thousands of lexical forms[cite: 640].
- **Bimanual Telemetry and Temporal Spatial Mapping:** By elevating tracking constraints to accommodate dual-hand landmark processing[cite: 637], the engine can capture complex spatial data trajectories. This advancement will allow the system to map spatial vectors over continuous time frames, enabling support for intricate grammar rules and body-anchored signs.
- **Mobile Ecosystem Migration via Component Frameworks:** To deliver true communication portability, the application framework can be adapted into hybrid mobile apps via cross-platform development toolkits like Flutter or React Native[cite: 637]. This migration will involve restructuring the layout into a portrait interface, utilizing device camera modules, and optimizing processing speeds via specialized on-device runtimes.
- **Video Relay Service (VRS) Infrastructure Integration:** Linking the platform with professional Video Relay Service APIs will bridge the gap between automated translation tools and human support networks[cite: 641]. The software can dynamically route simple casual chats through automated AI loops and smoothly upgrade to live video interpreter connections for legally or medically sensitive interactions[cite: 642].
- **Offline Edge Audio Generation and AR Guidance Overlays:** Integrating specialized local TTS engines (such as Coqui TTS or pyttsx3) will entirely remove the application's external internet dependencies[cite: 598, 637]. Furthermore, the frontend view can incorporate an augmented reality (AR) bounding box overlay to guide users' hands into optimal lighting zones, significantly reducing classification dropouts under challenging tracking environments[cite: 579, 637].

IX. FUTURE SCOPE

A. Deep Learning Classifier Integration

Near-term upgrades plan to replace the rule-based geometric classifier with a deep learning model combining Convolutional Neural Networks and Long Short-Term Memory networks (CNN-LSTM). Training the model on datasets like WLASL or the ASL-LEX corpus will expand the recognizable vocabulary to thousands of signs, supporting natural sign language communication rather than basic finger-spelling.

B. Bimanual and Temporal Tracking

Future iterations will extend the MediaPipe implementation to simultaneously track both hands, enabling the system to process more complex bimanual signs. Adding temporal sequence

tracking will improve the recognition of dynamic gestures that depend on motion paths over time.

C. Mobile Application Deployment

To improve portability, development plans include building dedicated mobile applications using React Native or Flutter. Mobile ports will leverage native device camera access and adjust interface designs to accommodate smaller smartphone screens.

D. Video Relay Service (VRS) Frameworks

Long-term plans involve integrating the platform with professional Video Relay Service (VRS) infrastructure. This will create a more complete digital communication ecosystem for the deaf community, combining automated gesture translation for casual chats with on-demand interpreter access for complex interactions.

Additional enhancements will focus on offline speech synthesis tools (such as Coqui TTS), multi-user group chat rooms, and augmented reality (AR) visual overlays to guide users' hand positioning within the camera frame.

X. CONCLUSION

This project has successfully achieved its primary objective: the design and implementation of a Real-Time Sign Language Chat Application with Speech Output that enables people with hearing and speech impairments to communicate effectively through hand gestures in a familiar digital chat interface. The system demonstrates that advanced AI and computer vision technologies—specifically MediaPipe's real-time hand landmark detection and rule-based geometric gesture classification—can be integrated into practical, accessible web applications without requiring specialized hardware, extensive training datasets, or expensive computational resources.

The evaluation results confirm that the system achieves its functional and performance requirements: overall gesture recognition accuracy of 90.3%, WebSocket message delivery latency of 47ms on LAN, and a System Usability Scale score of 78.4/100. The security analysis confirmed the effectiveness of the JWT authentication, CSRF protection, and input validation mechanisms. The project also demonstrates a robust full-stack engineering approach using current industry best practices to ensure that the system is both maintainable and extensible.

In conclusion, the developed application represents a meaningful contribution to inclusive communication technology. It demonstrates that the technology is ready for sign language communication to be integrated into mainstream digital communication platforms, and that academic projects can meaningfully advance the state of accessible technology.

REFERENCES

- [1] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C. Chang, and M. Grundmann, "MediaPipe Hands: On-device Real-time Hand Tracking," *arXiv preprint arXiv:2006.10214*, 2020.
- [2] L. Pigou, S. Dieleman, P. J. Kindermans, and B. Schrauwen, "Sign Language Recognition using Convolutional Neural Networks," in *European Conference on Computer Vision (ECCV) Workshops*, 2015.
- [3] N. Pugeault and R. Bowden, "Spelling It Out: Real-Time ASL Fingerspelling Recognition," in *Proceedings of the 1st IEEE Workshop on Consumer Depth Cameras for Computer Vision*, 2011.
- [4] Django Software Foundation, "Django Documentation (Version 4.2)," 2023. [Online]. Available: <https://docs.djangoproject.com/en/4.2/>
- [5] Django Channels Development Team, "Channels Documentation (Version 4.0)," 2023. [Online]. Available: <https://channels.readthedocs.io/>
- [6] World Health Organization, "Deafness and Hearing Loss," WHO Fact Sheet, 2023. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>
- [7] Venkata Ramana, P. (2024). AI-driven predictive analytics in ERP systems for proactive supply chain optimization. *International Journal of Research in Information Technology and Computing*, 8(4).
- [8] Maturi, S. Y. (2023). Crowdsourced frontier: Unveiling autonomous adversarial cybercapabilities via open AI competition. *International Journal of Intelligent Systems and Applications in Engineering*, 11(1s), 275–284.
- [9] Manoharan, D. (2024). Governance-Oriented Quality Engineering Framework for Healthcare EDI Modernization. *International Journal of Multidisciplinary on Science and Management IJMSM*, 1(2).
- [10] B. Fang, J. Co, and My. Zhang, "DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation," in *Proceedings of ACM SenSys*, 2017.
- [11] D. Li, C. Rodriguez, X. Yu, and H. Li, "Word-Level Deep Sign Language Recognition from Video: A New Large-Scale Dataset and Methods Comparison," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2020.
- [12] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, and M. Grundmann, "MediaPipe: A Framework for Building Perception Pipelines," *arXiv preprint arXiv:1906.08172*, 2019.
- [13] A. Wadhawan and P. Kumar, "Sign Language Recognition Systems: A Decade Systematic Literature Review," *Archives of Computational Methods in Engineering*, vol. 28, no. 3, pp. 785–813, 2021.
- [14] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, Internet Engineering Task Force (IETF), 2011. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc6455>
- [15] Doragacharla, V. R. (2026). Building Real-Time Pricing Systems for Modern Retail. Available at SSRN 6451760.
- [16] Adabala, P. K. (2024). Utilizing predictive analytics to improve efficiency and decision-making in ERP-connected supply chains. *International Journal of Intelligent Systems and Applications in Engineering*, 12(22s), 2465
- [17] Mozilla Developer Network, "MediaDevices.getUserMedia() Web API Documentation," 2023. [Online]. Available: <https://developer.mozilla.org/>
- [18] Redis, "Redis Documentation," 2023. [Online]. Available: <https://redis.io/documentation>
- [19] Docker Inc., "Docker Documentation," 2023. [Online]. Available: <https://docs.docker.com/>
- [20] gTTS Contributors, "gTTS Google Text-to-Speech Python Library Documentation," 2023. [Online]. Available: <https://gtts.readthedocs.io/>