Software Design Specification

Project Title:

SignLink

Project Description:

A Real-Time Pakistan Sign Language Translation and Subtitling into Urdu System for Video Conferencing Platforms

Project Code:

PSLTS-738

Internal Advisor:

Dr. Fahad Maqbool

Project Manager:

Dr. Muhammad Ilyas

Project Team:

i. Muhammad Hassan Shahbaz
ii. Behlol Abbas

Submission Date:

20 DEC,2025

____

Project Manager's Signature

Document Information

Category	Information
Customer	UOS – Department of Computer Science
Project	SignLink
Document	Software Design Specification
Document Version	1.0
Identifier	PSLTS-0738
Status	Draft
Author(s)	Muhammad Hassan Shehbaz Behlol Abbas
Approver(s)	Dr. Muhammad Ilyas
Issue Date	Sept. 15, 2025
Document Location	https://github.com/devHassan007/FYP-SLTS
Distribution	1. Advisor 2. PM 3. Project Office

Definition of Terms, Acronyms and Abbreviations

Term	Description
LSTM	Long-Short Term Memory
PSL	Pakistan Sign Language
TTS	Text To Speech
SDK	Software Development Kit
ASL	American Sign Language
BSL	British Sign Language
API	Application Programming Interface
SLR	Sign Language Recognition
VSL	View Sign Language
NLP	Natural Language Processing
GPU	Graphics Processing Unit

Table of Contents

1. Introduction

1.1 Purpose of Document

This Software Design Document (SDD) outlines the architectural framework, component-level design, and interface specifications for the Sign Link software application. It serves as a detailed technical blueprint that guides developers and engineers throughout the implementation phase, ensuring the system's structure aligns with the requirements defined in the SRS. The document provides clear descriptions of modules, data flows, and design decisions to support effective development, testing, maintenance, and future enhancements. By establishing a well-defined design foundation, the SDD promotes consistency, scalability, and efficient communication among all project stakeholders.

Intended Audience:

- **Developers and ML Engineers:** To implement system components and AI models according to the design specifications.
- **System Architects and Tech Leads:** To review architectural decisions and ensure design consistency.
- **QA Team:** To develop test plans aligned with the system design.
- **Project Managers:** To understand technical complexity and plan resources.
- **Future Maintainers:** To understand the system for debugging and enhancements.
- **Integration Partners:** To understand system interfaces for potential collaboration.

Design Methodology:

We're using **Object-Oriented Design (OOD)** for this project because it offers modularity, reusability, and maintainability—critical for a complex AI-powered application like Sign Link. OOD principles allow us to separate concerns cleanly (recognition, UI, backend services), reuse code across different sign languages and platforms, and easily extend the system with new features without breaking existing functionality.

1.2 Project Overview

SignLink is an accessibility system designed to enable **real-time Pakistan Sign Language (PSL) recognition and Urdu translation** during video conferencing. It integrates with platforms such as **Zoom and Google Meet**, allowing Deaf and hard-of-hearing users to communicate naturally through PSL while hearing participants receive **Urdu subtitles and synthesized Urdu speech**. This enhances digital accessibility, inclusivity, and equal participation in virtual meetings.

What It Does

- Recognizes **PSL gestures in real time** using skeletal keypoint extraction and deep learning.
- Translates recognized PSL glosses into **Urdu text** and generates **Urdu speech output**.
- Integrates as a **plug-and-play add-on** with major video conferencing platforms (via SDKs/virtual devices).
- Provides **continuous PSL recognition**, allowing natural conversation flow.
- Displays on-screen **Urdu subtitles** and injects **Urdu audio** into the meeting for all participants.

How It Works

The system follows a multi-layered architecture:

- **Computer Vision Layer**
  Uses Media Pipe to detect and extract hand landmarks, body pose, and facial key points from the video stream.
- **Machine Learning Pipeline**
  LSTM or Transformer-based deep learning models process extracted keypoints to recognize PSL gestures.
  Models are trained using **custom PSL datasets**, **Multi-VSL**, and additional PSL video samples collected for this project.
- **Translation Module**
  Converts recognized PSL gloss sequences into meaningful **Urdu sentences** using NLP-based rule systems.
- **Text-to-Speech (TTS) Module**
  Generates **Urdu synthesized audio** from translated text for hearing participants.
- **Platform Integration Layer**
  Uses virtual camera and virtual audio output or conferencing APIs (e.g., Zoom SDK) to inject subtitles and audio into meetings.
- **User Interface Layer**
  Provides:
  - system configuration options
  - calibration tools
  - real-time display of Urdu subtitles
  - performance indicators (FPS, confidence, latency)

Why It Matters

SignLink empowers Deaf users by enabling natural PSL communication during online meetings **without depending solely on human interpreters**. It significantly improves accessibility for academic, corporate, and social environments by bridging communication gaps between PSL signers and Urdu-speaking participants. The project also contributes to research in **AI-driven sign language recognition**, helping advance accessibility technologies within Pakistan.

1.3 Scope

This project delivers a prototype for **real-time Pakistan Sign Language (PSL) translation** in controlled indoor environments. The system focuses on accurate PSL gesture recognition, Urdu text generation, and Urdu speech synthesis during live video conferencing sessions. The architecture is modular and prepared for future expansion into more complex PSL grammar, multilingual support, and multi-signer interaction.

In-Scope

- **Recognition of a selected PSL vocabulary**, including commonly used words and short phrases based on a custom PSL dataset.
- **Real-time PSL-to-Urdu translation**, converting recognized glosses into grammatically correct Urdu text.
- **Urdu subtitle generation and Urdu speech synthesis**, with audio injected into the video conferencing platform.
- **Integration with at least one major platform** (starting with Zoom) using SDKs or virtual camera/audio devices.
- **Single-signer recognition** in a controlled, well-lit environment with uncluttered backgrounds.
- **End-to-end latency under 100ms** for recognition and translation.
- **Basic user interface** for configuration, calibration, subtitle settings, and performance monitoring (FPS, confidence, latency).
- **Local on-device processing**, avoiding the need for cloud computation.

Out of Scope

- **Full PSL grammar support**, including detailed facial expressions, body shifts, or advanced non-manual markers beyond selected vocabulary.
- **Recognition of multiple simultaneous signers** in the same video frame.
- **Reverse translation (Urdu speech/text to PSL generation)**.
- **Development of a proprietary video conferencing platform** (system relies on existing platforms like Zoom/Google Meet).
- **Robust operation in poor lighting, occlusions, or cluttered backgrounds** (beyond controlled environments).
- **Mobile, tablet, AR/VR, or wearable device optimization** (prototype is desktop-only).
- **Multilingual output** (only Urdu text and Urdu speech are supported in the prototype).
- **Cloud-based or distributed processing frameworks**, focusing solely on local execution.
- **Enterprise-grade security, encryption, or compliance features**, aside from essential privacy measures.

2. Design Considerations

2.1 Assumptions and Dependencies

2.1.1Design-Specific Assumptions

Hardware Requirements

- Users have webcams capable of **720p resolution at 30 FPS minimum**.
- **CPU:** Intel i5 (8th Generation) or equivalent, with **8 GB RAM minimum**.
- **GPU:** NVIDIA GTX 1060 or equivalent is **recommended** for real-time inference but not mandatory for prototype usage.

Software Environment

- Target Operating Systems:
  - **Windows 10/11** (primary target)
  - **macOS 10.15+**
  - **Linux Ubuntu 20.04+**
- **Python 3.8+** runtime environment available on the system.
- Video conferencing platform SDKs/APIs (Zoom, Google Meet) remain **stable and accessible**.
- Virtual camera and virtual audio drivers can be installed by the user.

Network Conditions

- Stable internet connection for video conferencing (**minimum 5 Mbps**).
- Core sign recognition and translation processing is performed **locally**, requiring no additional bandwidth beyond normal video calls.

User Behavior Assumptions

- Users position themselves within the camera frame with **hands and upper body clearly visible**.
- Signing occurs within a **1–2 meter distance** from the camera.
- Adequate indoor lighting conditions are available.
- Single signer is present in the frame (multi-signer scenarios are out of scope for the prototype).

2.1.2 Dependencies

External Libraries and Frameworks

- **Media Pipe Holistic (Google)** for hand, pose, and facial landmark extraction.
- **TensorFlow / PyTorch** for ML model implementation and inference.
- **OpenCV** for video capture, preprocessing, and frame handling.
- **Zoom SDK / Google Meet APIs** for video conferencing integration.
- **pyttsx3 or Google Text-to-Speech (gTTS)** for Urdu speech synthesis.
- **Virtual audio routing tools** (e.g., VB-Audio Cable) for audio injection.

Datasets

- **Pakistan Sign Language (PSL)** custom dataset.
- **Multi-VSL dataset** for multi-view sign recognition.
- Continuous access to datasets for model retraining and evaluation.

Platform APIs

- Stability of **Zoom Virtual Camera and Audio APIs**.
- Permissions for subtitle overlays within meeting interfaces.
- Continued support for virtual device-based integrations.

Third-Party Services

- Pre-trained **Media Pipe landmark detection models**.
- Text-to-speech engines supporting **Urdu language output**.

2.2 Risks and Volatile Areas

2.2.1High-Risk Areas

Real-Time Performance

- **Risk:** End-to-end latency exceeds the **100 ms** target, affecting conversational flow.
- **Mitigation:** GPU acceleration, optimized model architectures, frame skipping.
- **Contingency:** Reduce vocabulary size or switch to lightweight recognition models.

Platform API Changes

- **Risk:** Zoom or Google Meet restrict or modify SDK/API access.
- **Mitigation:** Implement a platform-agnostic integration layer.
- **Contingency:** Maintain adapters for multiple conferencing platforms.

Recognition Accuracy

- **Risk:** Accuracy drops below **90%** in real-world usage.
- **Mitigation:** Training with diverse signers and environments.
- **Contingency:** Introduce confidence thresholds and visual alerts for uncertain predictions.

Environmental Variability

- **Risk:** Performance degradation due to lighting, background clutter, or occlusion.
- **Mitigation:** Data augmentation and background preprocessing.
- **Contingency:** Provide user setup guidelines and environment calibration tools.

2.2.2Volatile Areas Requiring Flexible Design

Sign Language Vocabulary Expansion

- Modular recognition models to support vocabulary growth.
- Plugin-based architecture for adding new sign languages (future extension).

Platform Integration

- Adapter pattern for isolating platform-specific logic.
- Clean separation between SignLink core engine and conferencing platforms.

ML Model Updates

- Model versioning and rollback support.
- A/B testing framework for evaluating new model versions.

Output Modalities

- Decoupled translation output pipeline.
- Easy extensibility for:
    - Additional TTS voices
    - Subtitle styling
    - New output languages in future versions

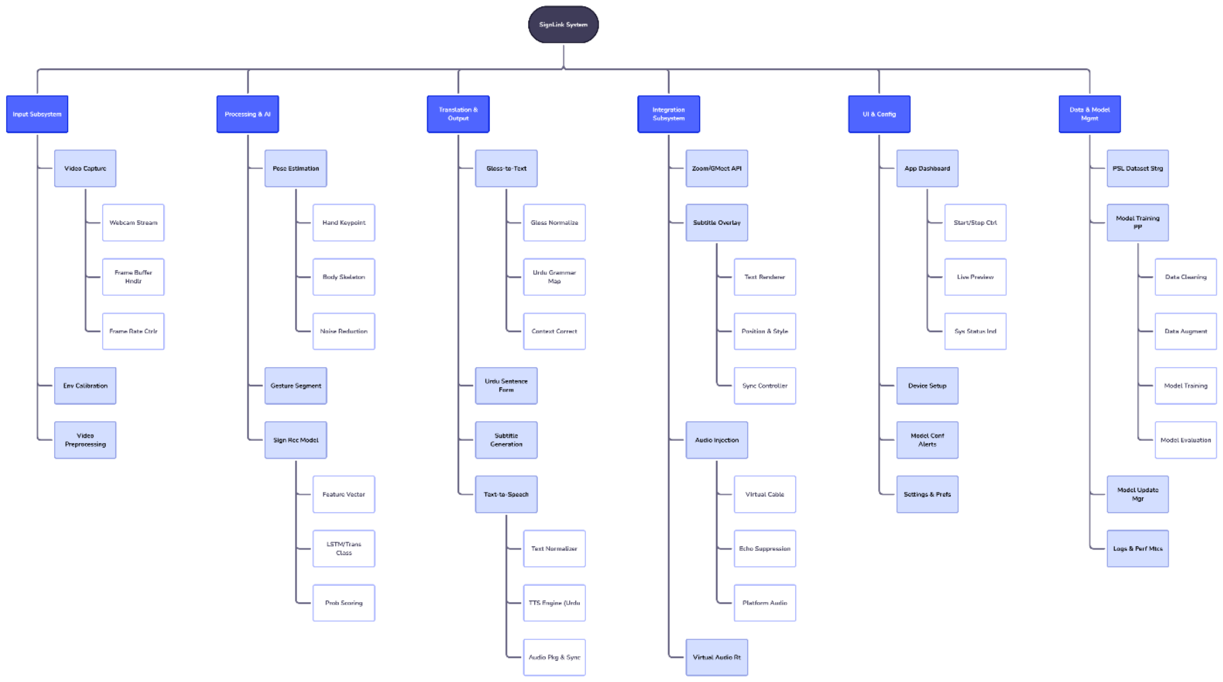
3. System Architecture

3.1 System Level Architecture

High-Level Architecture Overview:

Sign Link follows a **layered architecture pattern** with clear separation of concerns. The system is decomposed into five primary subsystems that interact through well-defined interfaces.

System Decomposition:



Element Relationships:

- **UI Layer ↔ Integration Layer:** Configuration data and status updates
- **Integration Layer ↔ Video Input Manager:** Raw video frames
- **Video Input Manager → CV Processing Layer:** Processed frames
- **CV Processing Layer → ML Recognition Layer:** Keypoint sequences
- **ML Recognition Layer → Translation Layer:** Recognized sign IDs
- **Translation Layer → Output Manager:** Text and audio data
- **Output Manager → Integration Layer:** Formatted subtitles and synthesized speech

External System Interfaces:

- **Video Conferencing Platforms:** Bidirectional communication through SDKs
- **Operating System:** Virtual camera/audio device drivers
- **File System:** Model weights, configuration files, log data
- **User Input Devices:** Webcam, microphone (for calibration)

Physical Deployment:

- **Primary Execution:** User's local machine (desktop/laptop)
- **Model Storage:** Local file system (models loaded into RAM during execution)
- **Video Processing:** Main CPU with optional GPU acceleration
- **Network Communication:** Through existing video conferencing client

Global Design Strategies:

- **Error Handling:** Graceful degradation with user notifications; fallback to silent mode on critical failures
- **Logging:** Comprehensive logging at each layer for debugging and performance monitoring
- **Configuration Management:** Centralized configuration service with validation
- **Resource Management:** Frame buffer pools, model caching, memory-efficient data structures

3.2 Sub-System / Component / Module Level Architecture

3.2.1 User Interface Subsystem

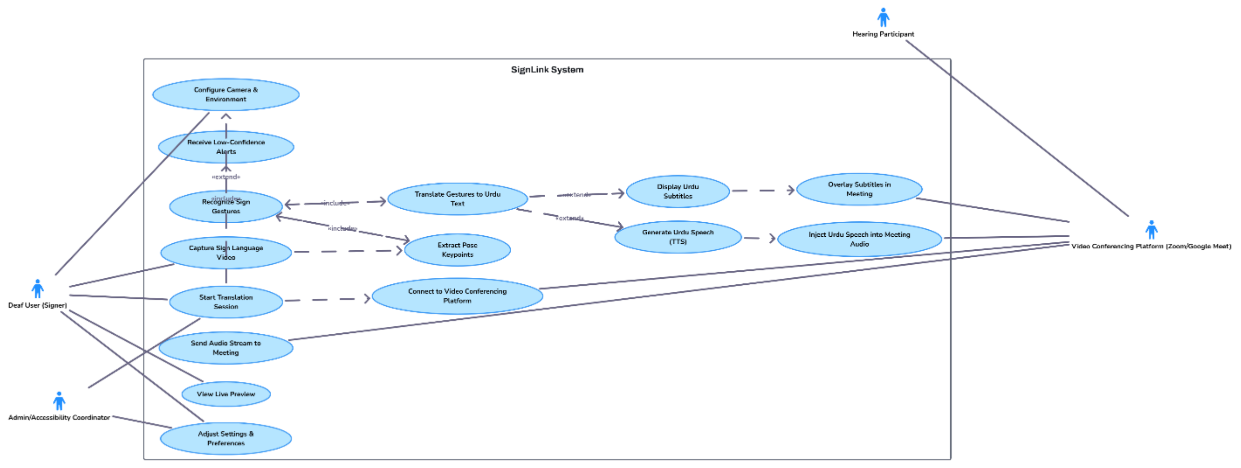
Components:

- **Configuration Panel:** Settings for model selection, platform choice, output preferences
- **Status Monitor:** Real-time display of recognition confidence, FPS, latency
- **Subtitle Renderer:** On-screen text display with customizable positioning and styling
- **Notification Service:** User alerts for errors, warnings, calibration needs

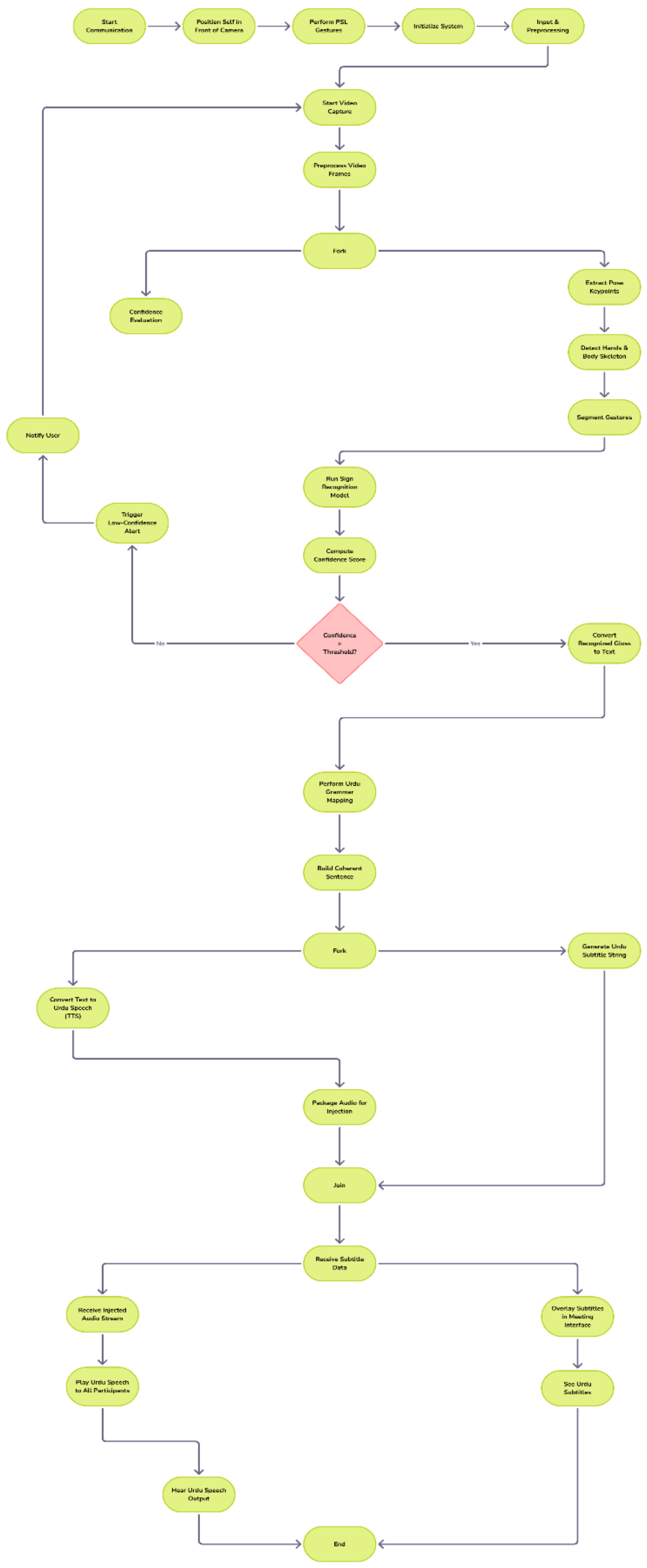
Interactions:

- Receives configuration updates from user
- Sends settings to Platform Integration Layer
- Receives status updates from all lower layers
- Renders subtitles from Translation Layer output

Use Case Diagram:


Hearing Participant
SignLink System
Configure Camera & Environment
Receive Low-Confidence Alerts
«extend»
«include»
Recognize Sign Gestures
«include»
Translate Gestures to Urdu Text
«extend»
Display Urdu Subtitles
Overlay Subtitles in Meeting
«include»
«extend»
Extract Pose Keypoints
Generate Urdu Speech (TTS)
Inject Urdu Speech into Meeting Audio
Capture Sign Language Video
Start Translation Session
Connect to Video Conferencing Platform
Video Conferencing Platform (Zoom/Google Meet)
Deaf User (Signer)
Send Audio Stream to Meeting
View Live Preview
Admin/Accessibility Coordinator
Adjust Settings & Preferences

Communication Flow Diagram:


Start Communication
Position Self in Front of Camera
Perform PSL Gestures
Initialize System
Input & Preprocessing
Start Video Capture
Preprocess Video Frames
Fork
Confidence Evaluation
Extract Pose Keypoints
Detect Hands & Body Skeleton
Segment Gestures
Notify User
Run Sign Recognition Model
Trigger Low-Confidence Alert
Compute Confidence Score
Confidence > Threshold?
No
Yes
Convert Recognized Gloss to Text
Perform Urdu Grammar Mapping
Build Coherent Sentence
Fork
Generate Urdu Subtitle String
Convert Text to Urdu Speech (TTS)
Package Audio for Injection
Join
Receive Subtitle Data
Receive Injected Audio Stream
Overlay Subtitles in Meeting Interface
Play Urdu Speech to All Participants
See Urdu Subtitles
Hear Urdu Speech Output
End

3.2.2 Platform Integration Subsystem

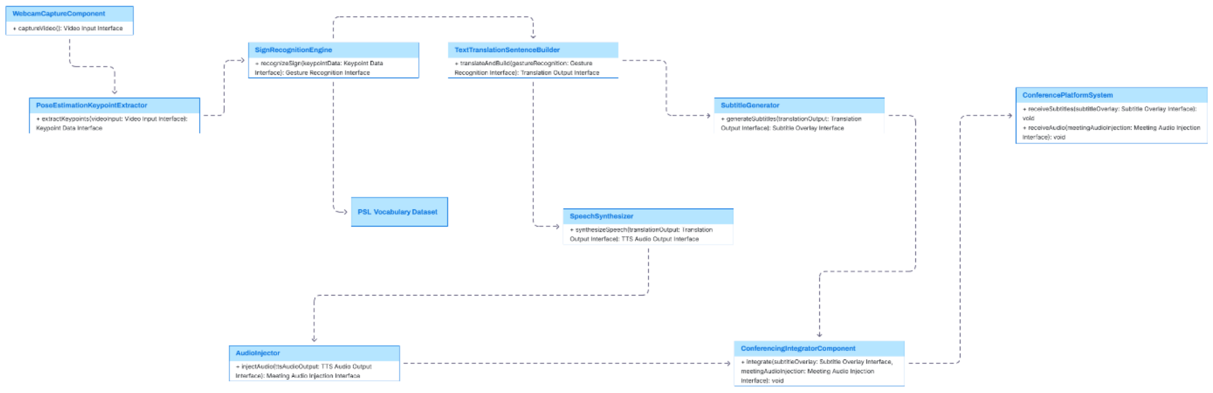
Components:

- **Platform Adapter Factory:** Creates appropriate adapter based on detected platform
- **Zoom Adapter:** Zoom SDK integration for video/audio injection
- **Generic Platform Adapter:** Fallback using virtual devices
- **Virtual Camera Manager:** Creates/manages virtual camera device
- **Virtual Audio Manager:** Creates/manages virtual audio device
- **Stream Router:** Directs video/audio streams between platform and processing pipeline

Interactions:

- Captures video stream from conferencing platform
- Injects synthesized audio into platform audio stream
- Overlays subtitles on video output
- Provides platform status to UI Layer

Component Diagram :


WebcamCaptureComponent
+ captureVideo(): Video Input Interface
PoseEstimationKeypointExtractor
+ extractKeypoints(videoInput: Video Input Interface): Keypoint Data Interface
SignRecognitionEngine
+ recognizeSign(keypointData: Keypoint Data Interface): Gesture Recognition Interface
TextTranslationSentenceBuilder
+ translateAndBuild(gestureRecognition: Gesture Recognition Interface): Translation Output Interface
SubtitleGenerator
+ generateSubtitles(translationOutput: Translation Output Interface): Subtitle Overlay Interface
ConferencePlatformSystem
+ receiveSubtitles(subtitleOverlay: Subtitle Overlay Interface): void
+ receiveAudio(meetingAudioInjection: Meeting Audio Injection Interface): void
PSL Vocabulary Dataset
SpeechSynthesizer
+ synthesizeSpeech(translationOutput: Translation Output Interface): TTS Audio Output Interface
AudioInjector
+ injectAudio(ttsAudioOutput: TTS Audio Output Interface): Meeting Audio Injection Interface
ConferencingIntegratorComponent
+ integrate(subtitleOverlay: Subtitle Overlay Interface, meetingAudioInjection: Meeting Audio Injection Interface): void

3.2.3 Computer Vision Processing Subsystem

Components:

- **Frame Preprocessor:** Resizing, normalization, background blurring
- **Media Pipe Integrator:** Wraps Media Pipe Holistic API
- **Key point Extractor:** Extracts hand (21 points each), pose (33 points), face (468 points) landmarks
- **Sequence Buffer:** Maintains temporal window of key points (e.g., last 30 frames)

- **Data Normalizer:** Normalizes key points to reference frame and scale

Interactions:

- Receives raw frames from Platform Integration
- Outputs normalized key point sequences to ML Layer
- Sends processing metrics to UI Status Monitor

3.2.4 Machine Learning Recognition Subsystem

Components:

- **Model Loader:** Loads and caches trained models
- **Feature Encoder:** Converts key point sequences to model input format
- **LSTM Inference Engine:** Processes temporal sequences for sign recognition
- **Transformer Inference Engine (Alternative):** Attention-based recognition
- **Confidence Estimator:** Calculates prediction confidence scores
- **Vocabulary Manager:** Maps model outputs to sign labels
- **Recognition Smoother:** Temporal filtering to reduce jitter

Interactions:

- Receives keypoint sequences from CV Processing Layer
- Outputs recognized sign IDs with confidence scores to Translation Layer
- Provides inference metrics to UI Layer

3.2.5 Translation & Synthesis Subsystem

Components:

- **Sign-to-Text Translator:** Converts sign sequences to URDU
- **Grammar Corrector:** Applies NLP rules for sentence structure
- **Text-to-Speech Engine:** Synthesizes audio from text
- **Subtitle Formatter:** Formats text for on-screen display
- **Output Queue Manager:** Buffers and synchronizes text/audio output

Interactions:

- Receives recognized signs from ML Layer
- Outputs formatted text to Platform Integration for subtitle display
- Outputs synthesized audio to Platform Integration for audio injection

3.3 Sub-Component / Sub-Module Level Architecture

This section provides a detailed breakdown of each major component in the system into its internal sub-modules and sub-components. It also explains how these sub-components interact with one another to enable the complete functionality of the SignLink System. The purpose of this decomposition is to show the internal architecture of each module beyond the high-level system diagram, clarifying responsibilities, data flow, and module boundaries.

3.3.1 User Interface Module — Sub-Components

The **User Interface (UI)** module handles all user-facing interactions. It consists of the following sub-components:

a. Preview Renderer

- Displays real-time camera feed.
- Renders overlays such as status messages, gesture markers, and subtitles.

b. Interaction Controller

- Manages UI actions (Start Capture, Stop Capture, Change Camera).
- Communicates with VideoCapture module via API or local bridge.

c. Status Display Manager

- Shows system states (Processing, Ready, Error).
- Receives status updates from backend components.

Internal Flow:
User Input → Interaction Controller → Backend Trigger
Backend Messages → Status Display Manager → Preview Renderer

3.3.2 Video Capture Module — Sub-Components

The **Video Capture Module** acquires raw frames from the camera device.

a. Device Handler

- Connects to the selected camera hardware (mobile, webcam, IP camera).
- Retrieves frame streams at the configured frame rate.

b. Frame Buffer Manager

- Temporarily stores frames in a buffer for synchronous processing.
- Implements queueing and timestamping.

c. Frame Exporter

- Sends frames to the Preprocessor module.
- Logs frame metadata into the database (through the Session Manager).

3.3.3 Preprocessor Module — Sub-Components

The Preprocessor transforms raw frames into a normalized format suitable for ML models.

a. Resolution Scaler

- Resizes frames to the required ML model input dimension (e.g., 256×256).

b. Normalization Engine

- Adjusts lighting, contrast, and removes noise.
- Converts color spaces (RGB → grayscale or model-specific format).

c. Frame Serializer

- Packages the processed frame into a tensor or structured array.
- Forwards it to the Pose Estimator.

3.3.4 Pose Estimator Module — Sub-Components

The Pose Estimator extracts key points (hands, face, body).

a. Model Loader

- Loads and initializes ML model (Media Pipe, Open Pose, Blaze Pose).

b. Key point Extractor

- Performs inference on processed frames.
- Produces landmarks + confidence scores.

c. Data Formatter

- Converts model output to a structured format (JSON).
- Sends to Gesture Recognizer and stores in the database (Key points table).

3.3.5 Gesture Recognizer Module — Sub-Components

The Gesture Recognizer identifies sign-language gestures from a sequence of key points.

a. Sequence Buffer

- Stores sliding window of key points (temporal sequence).
- Handles sequence segmentation (start/end detection).

b. Classification Engine

- Runs the ML or rule-based classifier (LSTM, Transformer, HMM).
- Outputs gesture label + confidence.

c. Gesture Validator

- Checks confidence threshold.
- Ensures gesture is reliable and not noise.

d. Gesture Event Generator

- Emits recognized gesture events.
- Notifies Translator + stores gesture in database.

3.3.6 Translation Module — Sub-Components

The Translator converts gestures into natural-language sentences.

a. Gloss Mapper

- Maps gesture labels to glossary text.

- Example: SIGN_HELLO → "HELLO"

b. Grammar Rule Engine

- Applies grammar templates and linguistic rules.
- Converts gloss into fluent sentences (e.g., Urdu).

c. Text Output Formatter

- Finalizes translated output.
- Sends result to Subtitle Generator and Text To Speech.

3.3.7 Subtitle Generator Module — Sub-Components

a. Timing Calculator

- Uses gesture start/end timestamps to determine subtitle intervals.

b. Subtitle Builder

- Creates subtitle objects with text, timing, and styling.

c. Subtitle Exporter

- Stores subtitles in database.
- Sends them to Platform Integrator for display.

3.3.8 Text-to-Speech (TTS) Module — Sub-Components

a. Voice Model Manager

- Loads the voice model selected for TTS (male/female/accented voices).

b. Speech Synthesizer

- Converts translated text to audio waveform.

c. Audio Packager

- Formats output into MP3/WAV.
- Sends to Platform Integrator and stores metadata in Audio Output table.

3.3.9 Platform Integrator Module — Sub-Components

Responsible for delivering subtitles/audio to the target platform.

a. API Connector

- Handles authentication and communication with external platforms.

b. Subtitle Dispatcher

- Sends subtitles in real-time to the selected platform (web client, Zoom API, etc.)

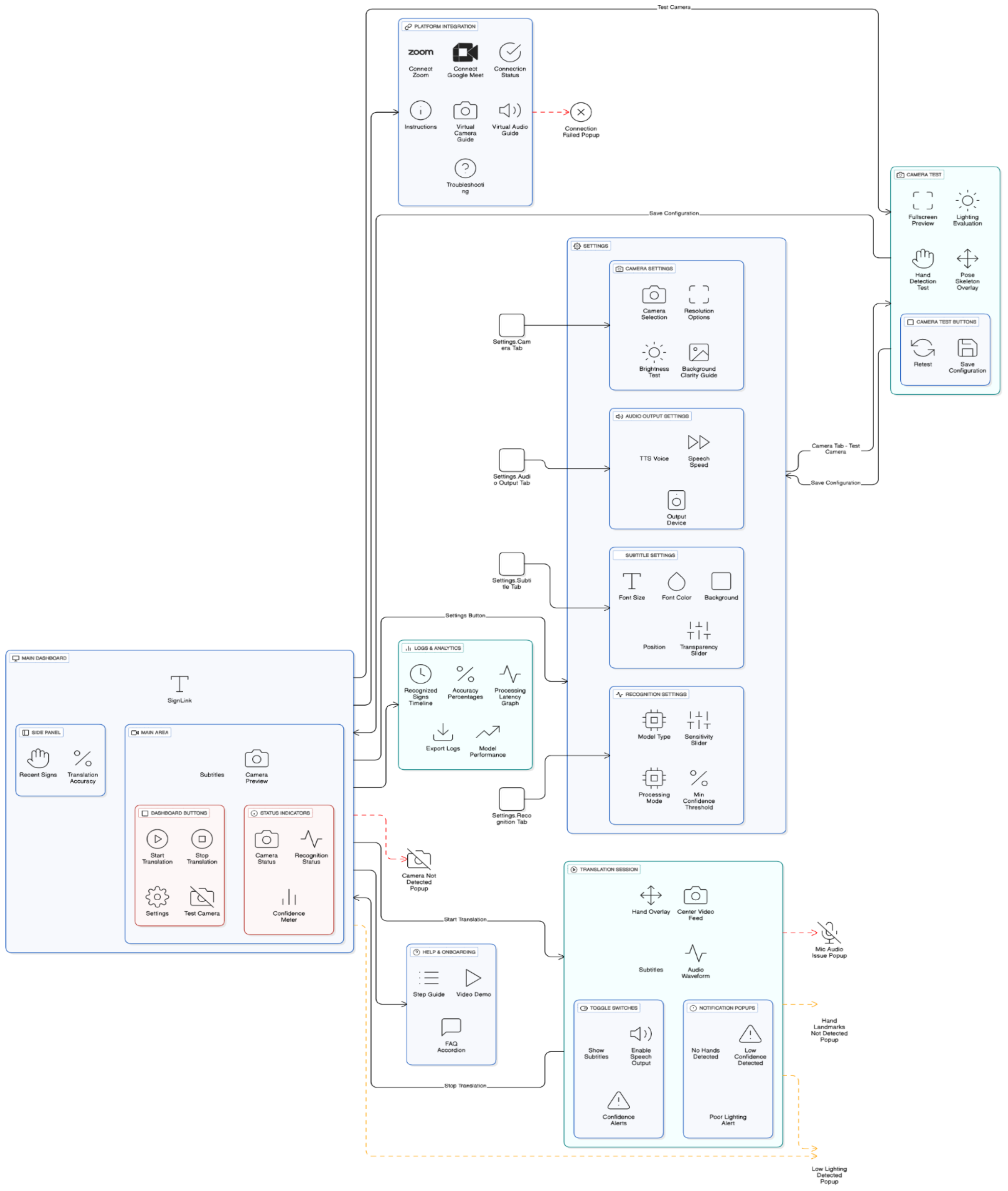
c. Audio Stream Sender

- Streams generated audio to the platform or client.

d. Delivery Monitor

- Confirms successful delivery and logs errors.

Accessibility-Optimized UI Flow :



4. Design Strategies

Strategy 1: Modular and Layered Architecture

The system is organized into clearly defined modules (UI, Business Logic, Data Access), allowing each component to evolve independently.
Reasoning: Supports scalability and easier maintenance.
Trade-offs: Requires careful interface definition between modules.
Impact Areas:

- **Future Extension:** New features can be added without affecting existing modules.
- **System Reuse:** Individual modules can be reused across different parts of the project.

Strategy 2: Consistent User Interface Paradigm

The system follows a uniform UI structure and interaction flow to ensure a smooth user experience.
Reasoning: Reduces user confusion and maintains predictable behavior.
Trade-offs: Limits experimentation with alternative UI layouts.
Impact Areas:

- **Future Extension:** New screens maintain the same UI logic.
- **Reuse:** Reusable UI templates and components.

Strategy 3: Centralized Data Management

A centralized database design is used for storing, retrieving, and updating all system data.
Reasoning: Ensures consistency, avoids data duplication, and simplifies auditing.
Trade-offs: The central DB can become a performance bottleneck if not optimized.
Impact Areas:

- **Data Management:** Improves integrity and persistence.
- **Future Enhancement:** Easier migrations and schema updates.

Strategy 4: Concurrency Control & Synchronization

The system maintains controlled access to shared data through safe update mechanisms (e.g., transaction management or locking).
Reasoning: Prevents conflicts when multiple users or processes access the same data.

Trade-offs: Can increase response time during peak usage.
Impact Areas:

- **Concurrency:** Ensures reliable multi-user operations.
- **Data Integrity:** Protects against inconsistent states.

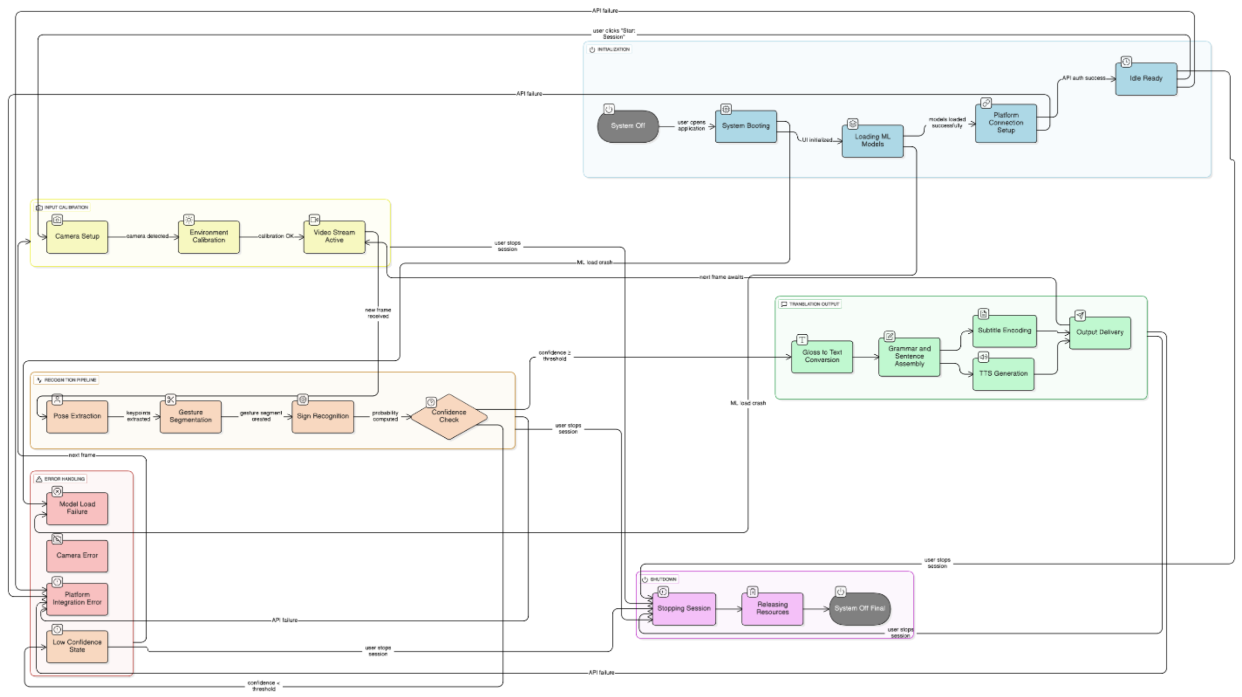
Strategy 5: Reusable Service Components (Service-Oriented Thinking)

Common operations (authentication, validation, file processing, etc.) are implemented as services accessible across modules.
Reasoning: Minimizes duplication and standardizes logic across the system.
Trade-offs: Slight overhead in managing shared services.
Impact Areas:

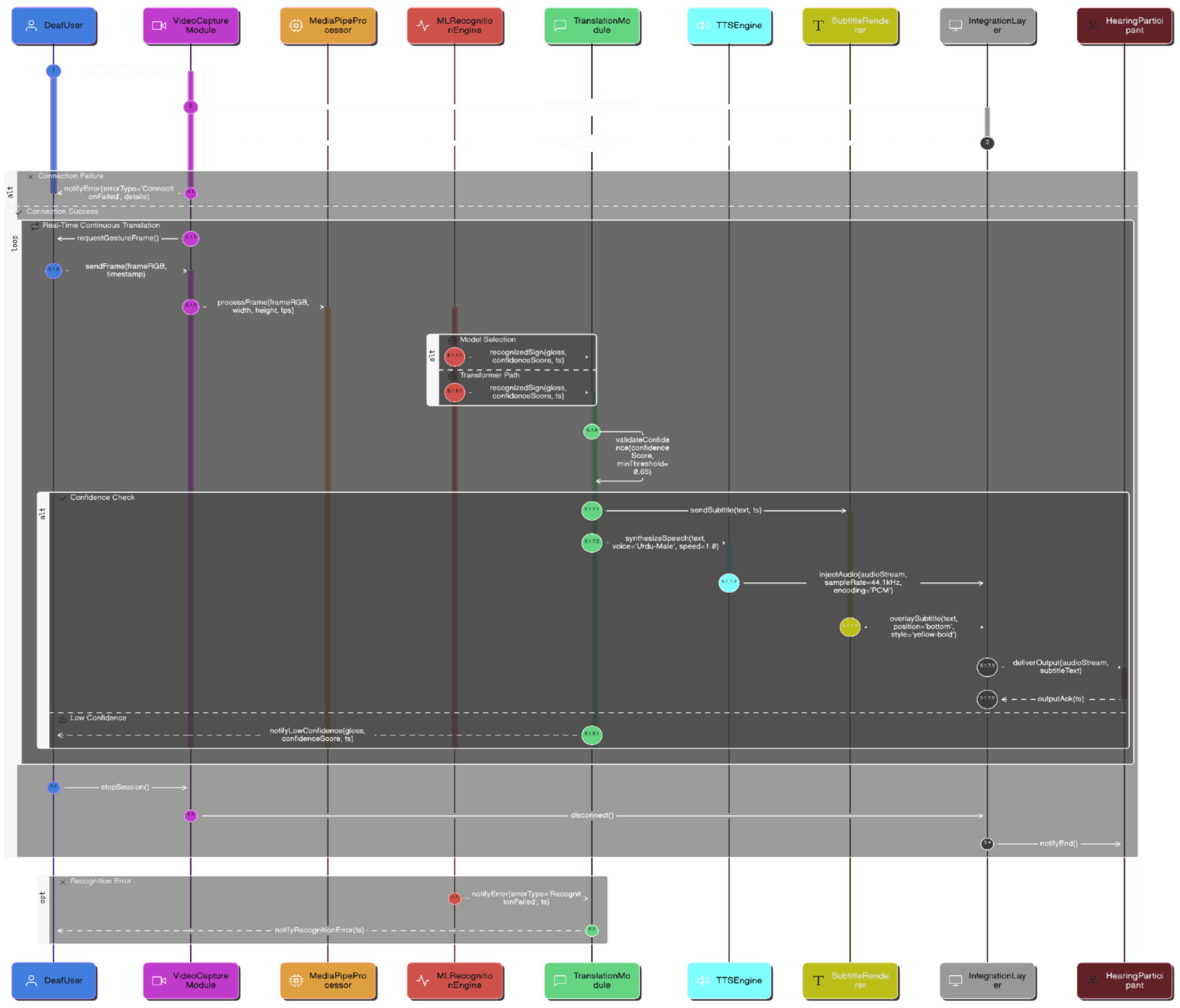
- **System Reuse:** Strong reuse of services throughout the project.
- **Future Enhancements:** Easy integration of new features using existing services.

5. Detailed System Design

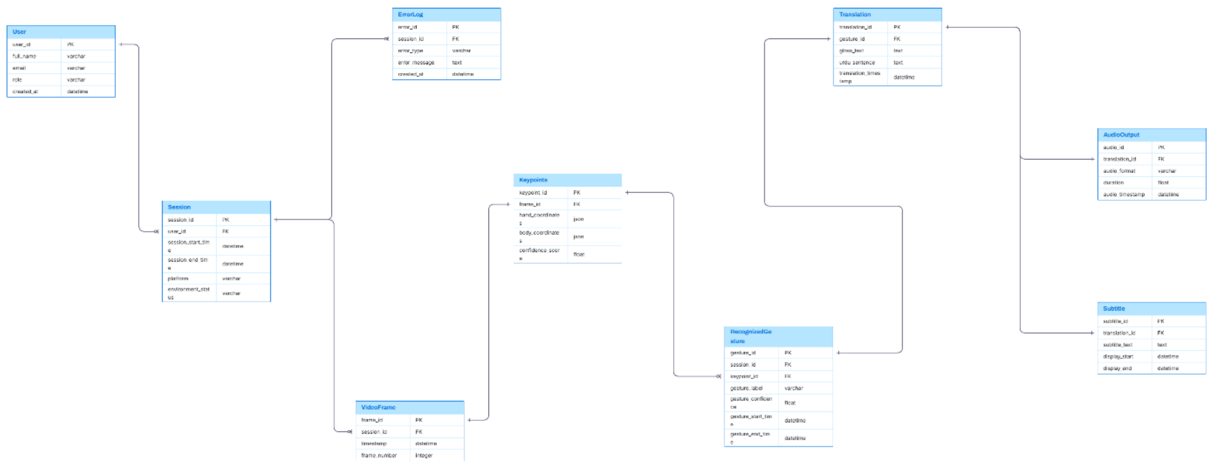
State Change Diagram:



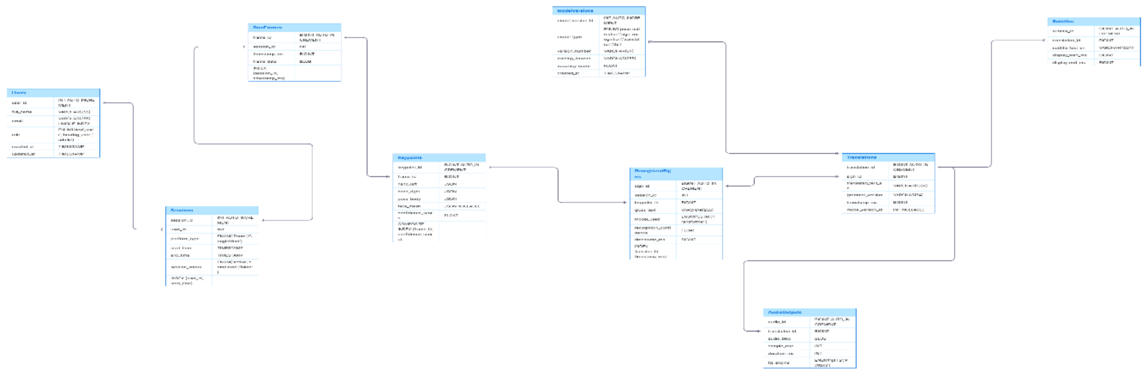
Sequence Diagram:


DeafUser
VideoCapture Module
MediaPipeProcessor
MLRecognitionEngine
TranslationModule
TTSEngine
SubtitleRenderer
IntegrationLayer
HearingParticipant
alt
Connection Failure
notifyError(errorType='ConnectionFailed', details)
Connection Success
loop
Real-Time Continuous Translation
requestGestureFrame()
sendFrame(frameRGB, timestamp)
processFrame(frameRGB, width, height, fps)
alt
Model Selection
recognizedSign(gloss, confidenceScore, ts)
Transformer Path
recognizedSign(gloss, confidenceScore, ts)
validateConfidence(confidenceScore, minThreshold=0.65)
alt
Confidence Check
sendSubtitle(text, ts)
synthesizeSpeech(text, voice='Urdu-Male', speed=1.0)
injectAudio(audioStream, sampleRate=44.1kHz, encoding='PCM')
overlaySubtitle(text, position='bottom', style='yellow-bold')
deliverOutput(audioStream, subtitleText)
outputAck(ts)
Low Confidence
notifyLowConfidence(gloss, confidenceScore, ts)
stopSession()
disconnect()
notifyEnd()
opt
Recognition Error
notifyError(errorType='RecognitionFailed', ts)
notifyRecognitionError(ts)

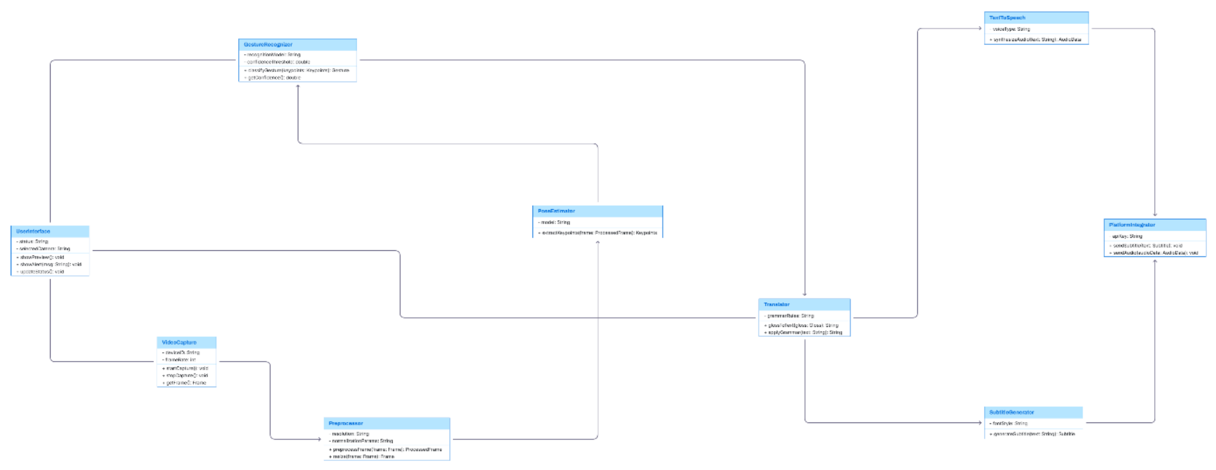
ER Diagram:



Physical data models:

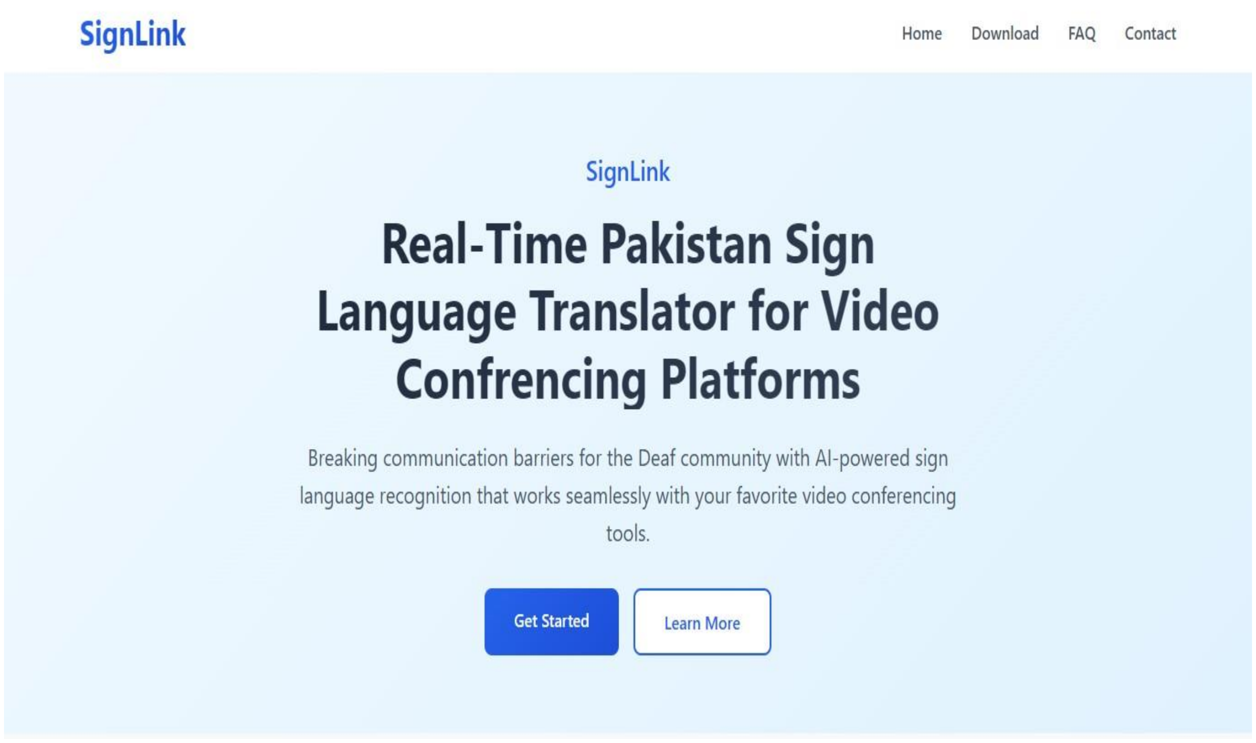


Class Diagram:


GestureRecognizer
- recognitionModel: String
- confidenceThreshold: double
+ classifyGesture(keypoints: Keypoints): Gesture
+ getConfidence(): double
UserInterface
- status: String
- selectedCamera: String
+ showPreview(): void
+ showAlert(msg: String): void
+ updateStatus(): void
PoseEstimator
- model: String
+ extractKeypoints(frame: ProcessedFrame): Keypoints
VideoCapture
- deviceID: String
- frameRate: int
+ startCapture(): void
+ stopCapture(): void
+ getFrame(): Frame
Preprocessor
- resolution: String
- normalizationParams: String
+ preprocessFrame(frame: Frame): ProcessedFrame
+ resize(frame: Frame): Frame
TextToSpeech
- voiceType: String
+ synthesizeAudio(text: String): AudioData
PlatformIntegrator
- apiKey: String
+ sendSubtitle(text: Subtitle): void
+ sendAudio(audioData: AudioData): void
Translator
- grammarRules: String
+ glossToText(gloss: Gloss): String
+ applyGrammar(text: String): String
SubtitleGenerator
+ fontStyle: String
+ generateSubtitle(text: String): Subtitle

GUI :

Home Page :

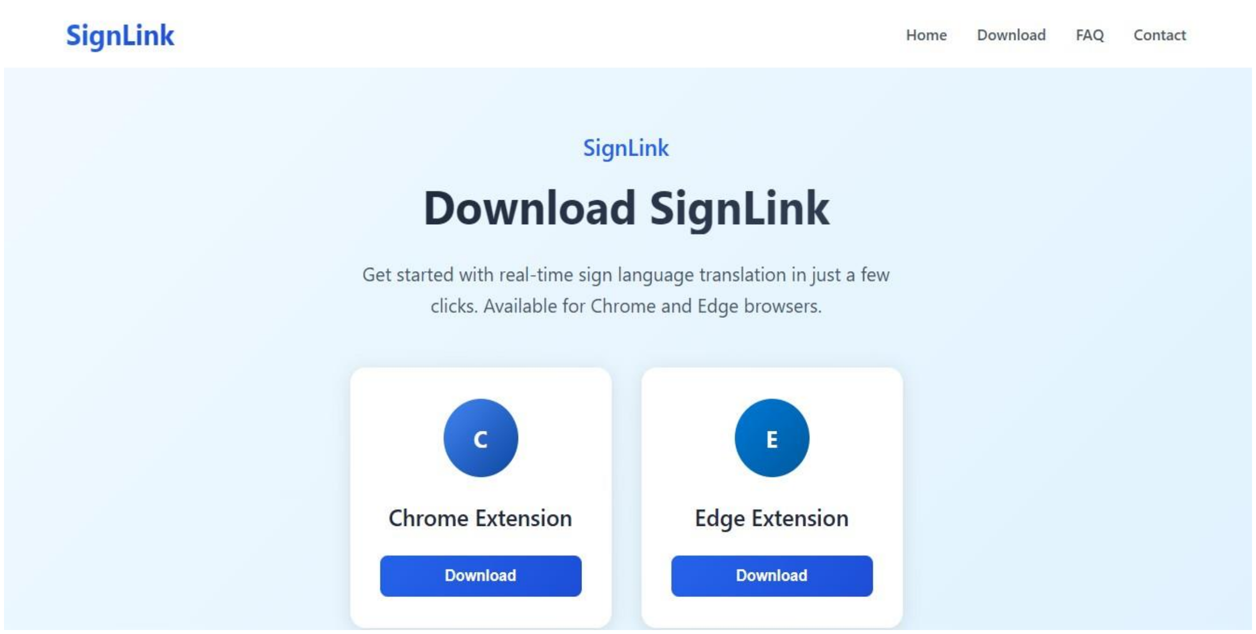


SignLink

Real-Time Pakistan Sign Language Translator for Video Confrencing Platforms

Breaking communication barriers for the Deaf community with AI-powered sign language recognition that works seamlessly with your favorite video conferencing tools.

Download Page :



Download SignLink

Get started with real-time sign language translation in just a few clicks. Available for Chrome and Edge browsers.

Chrome Extension

Edge Extension

6. References

Ref. No.	Document Title	Date of Release/ Publication	Document Source
PSLTS-738-Proposal	Project Proposal	Oct 10, 2025 Dec 4, 2025	https://github.com/devHassan007/FYP-SLTS
PSLTS-738-SRS	SRS	Oct 20, 2025 Dec 7, 2025	https://github.com/devHassan007/FYP-SLTS
PSLTS-738-SRS	Diagrams	Dec 14,2025	https://github.com/behlolabbas/Diagrams
Design Doc	UI	Dec 20, 2025	https://fyp-pslts.vercel.app/