******The British College**

**KATHMANDU**

**Coursework Submission Coversheet**(individual coursework only)

**Faculty of Arts, Environment and Technology LBU Student Id:**

77356702

**For checking by the student:**

Please ensure all information is complete and correct and attach this form securely to the front of your work before posting it in a coursework collection box.

Award name: BSc (Hons) Computing

Module code: COMP607

Module name: Production Project

Module run:

Coursework title: Breaking the Slience: AI and Computer Vision Driven Sign Language Translation System

Due Date:

Module leader: (In LBU)

Module Supervisor: (In TBC) Mr. Rohit Raj Pandey

**TURNITIN** Checked: YES NO ***(please circle)***

Submission date & time: Date: Time:

**Total Word Count: Total Number of Pages (including this front sheet):**

**In submitting this form with your assignment, you make the following declaration:**  
I declare, that the coursework submitted is my own work and has not (either in whole or part) been submitted towards the award of any other qualification either at LBU or elsewhere. I have fully attributed/referenced all sources of information used during the completion of my assignment, and I am aware that failure to do so constitutes an assessment offence.

Signed: ` Date:

**You are strongly advised to retain a second copy of your work in case of any query about the assignment.**

**For completion by the faculty:**

**This mark is provisional and subject to moderation and approval by the relevant examining board**

**Teacher's Feedback**

**Teacher's Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Breaking the Silence: AI and Computer Vision Driven Sign Language Translation System**

**Module: Production Project**

**Bsc. Hons. Computing**

**Level 6: 2nd semester**

**Abstract**

“Breaking the Silence: AI and Computer Vision Driven Sign Language Translation System” presents an integrated, bidirectional platform that leverages state-of-the-art deep learning and hand‐tracking to bridge communication gaps between Deaf and hearing individuals. At its core, the system employs MediaPipe Hands for real-time extraction of 21 three‐dimensional landmarks per frame, feeding these 63‐dimensional vectors into an attention‐augmented Long Short-Term Memory (LSTM) network. By tiling single‐frame landmarks into ten-step sequences, the model learns temporal gesture dynamics—essential for distinguishing motion‐based signs like “J” and “Z”—while its self-attention block highlights the most informative frames. Trained and validated on a stratified ASL dataset covering 36 classes, the network achieves over 99 % test accuracy, balanced accuracy, and top-3 recall, with near‐perfect calibration (reliability diagrams) and AUC scores, confirming its robustness and interpretability.

The system’s architecture comprises two primary pipelines. First, the **ASL-to-English** workflow (Figure 10) captures webcam input in the browser, buffers landmark sequences, and sends them via AJAX to Django endpoints. Predictions are logged, stored, and returned as the top three labels with confidence scores, which the frontend displays in real time. Upon session completion, a second endpoint invokes a large‐language model (e.g., Gemini API) to generate fluent English sentences, stored alongside audit logs and presented with optional text-to-speech. Second, the **English-to-ASL** pipeline (Figure 11) converts typed text into a static letter‐mapping video: sanitized input is parsed into individual characters, mapped to PNG images, assembled into frames via OpenCV, and encoded into MP4 using imageio. Metadata and user activity are recorded, and the resulting video URL is returned for in-browser preview and download.

Underpinning these features is a normalized relational schema (Figures 13 & 14) that extends Django’s authentication and permission tables with domain-specific models: **asl\_aslprediction**, **asl\_aslsentencegeneration**, **asl\_aslvideohistory**, and **asl\_auditlog**. This design ensures referential integrity, auditability, and support for soft deletion of video resources. Use-case diagrams (Figures 14–16) illustrate three actor roles—end users, administrators, and the email platform—interacting with the system boundary through registration, verification, prediction, translation, and history management functions.

Methodologically, the project follows an Agile, sprint-driven approach (Section 4), enabling iterative development, continuous testing on metrics such as latency and robustness under varying lighting conditions, and user-centered interface refinements. Literature review highlights the evolution of Sign Language Recognition (SLR) from hardware-dependent gloves to modern deep­learning systems, motivating our choice of lightweight landmark inputs over raw video and justifying the attention-LSTM hybrid over purely convolutional or transformer models.

Real-world applications span assistive communication in healthcare and education, public kiosks for Deaf travelers, live event captioning, and mobile AR integrations, all benefiting from the system’s low latency and high accuracy. Limitations—such as static video generation for complex grammar and sensitivity to occlusion—guide future work toward multimodal fusion of facial cues, transformer-based encoders, and expanded datasets. Overall, this comprehensive framework demonstrates how AI and computer vision can make sign language both visible and audible, fostering inclusive interaction across diverse settings.

1. **Introduction:**

The project, titled "Breaking the Silence: AI and Computer Vision Driven Sign Language Translation System," focuses in utilizing advanced AI and computer vision to translate sign language into both English text and speech, thereby narrowing communication gaps for deaf and mute individuals.

At its soul, the project handles the critical problem that hinder the social and professional incorporation of sign language users. By placing gesture recognition techniques applying frameworks like TensorFlow and PyTorch, alongside with hand tracking models such as Mediapipe, the system is deliberate to accurately detect and interpret hand gestures in real time. A primary aspect of the system is its user-friendly interface, which is bespoken to assure accessibility for users with limited technical knowledge. Furthermore, the blending of a text-to-speech module refines the system's functionality by furnishing audible output, further assisting efficient communication.

The system assimilates a real-time error handling mechanism to assures graceful operation by instantly warning users when a gesture is not recognized. This focus on trustworthiness and sensitivity is pivotal for practical, everyday use. Building upon previously research in sign language recognition—which grouped from hardware-dependent data gloves to modern deep learning methods—the system endeavors to subdue the exceptions posed by lively and continuing sign interpretation. It concentrates on building a scalable solution that accomplices fined under assorted conditions, such as varied lighting and hand sizes.

Eventually, the report reinforces not only the technological creations in back of the system besides also its possible social jolt. By letting clearer and more comprehensive communication, the project is composed to considerably enrich the quality of life for sign language users, enabling them to emerge more fully in educational, professional, and social surroundings.

1. **Literature Review**

Sign language Detection (SLR) refer to the process of recognizing and explanning gestures, hand movements and body language used in sign language to hold communication between people who use sign language and who do not. The field of SLR has witnessed outspoken improvements over years, staring from neural network-based systems to deep learning based systems. Early examined two stage neural network that uses a DataGlove for phoneme-level recognition, achieving 86% accuracy but was restrained by hardware depandance work (Kim, S. et. al., 1995). Another research used self-organizing frameworks like SHOSLIF-M, improving spatiotemporal recognition with a 96% accuracy rate. However, its reliance on custom-built features and limited scalability to real-life scenarios laid challenges (Cui, Y. et. al., 1995). Another Research focus on static, isolated, single-handed signs using camera-based systems, featuring a lack of organized datasets and a need for dynamic sign recognition improvements (Wadhawan, A. et. al., 2021). The use of CNNs showed odd performance with 99.90% accuracy for static signs using extensive datasets, highlighting the strength of deep learning (Kumar, P. et. al., 2020). Use of statistical methods for nonstop sign language handle real-life variability, placing multimodal features like facial landmarks and achieving revealing word error rate reductions (Koller, O. et. al., 2015). Another Study used statistical methods and fenones for efficient sign recognition but strumbled with sturdy subunit definition for varied gestures (Bauer, B. et. al., 2002). Another studyimproved a real-time sign language interpreter using a data glove and HMMs but met challenges with endored dependency and limited accuracy (Ouhyoung, M. et. al., 1998).

1. **Review Of Technology:**

The project adopts a methodical and research-driven approach, beginning with an extensive review of existing sign language recognition (SLR) technologies. This initial research helps identify limitations in past systems and informs the selection of suitable models and frameworks for the project. A crucial early step involves choosing a dataset, such as the American Sign Language (ASL) dataset or creating a custom dataset with both static and dynamic signs. These datasets form the basis for training machine learning models capable of recognizing a wide range of gestures. To support gesture detection, the project uses reliable pre-trained models like Mediapipe for real-time hand tracking, while TensorFlow and PyTorch are employed to build and train classification models using Convolutional Neural Networks (CNNs) for static gestures and potentially Recurrent Neural Networks (RNNs), LSTMs, or Transformers for dynamic gesture recognition.

Each system component is developed and tested independently to ensure performance, responsiveness, and reliability before full integration. Key metrics such as accuracy, latency, and robustness in various environmental conditions—like changes in lighting and hand orientation—are monitored throughout the testing process. This modular and iterative development strategy enables ongoing improvement at each stage and ensures the final system can perform consistently in real-world scenarios. The methodology prioritizes adaptability and modularity, allowing for early identification and resolution of issues while refining model performance based on observed results and user feedback.

To enhance accessibility, the project integrates a Text-to-Speech (TTS) module using the pyttsx3 library, enabling real-time conversion of recognized gestures into spoken English. A simple, intuitive user interface is also planned, ensuring the system can be used by individuals with limited technical expertise. The system is designed to provide meaningful feedback during use, further improving the user experience. This comprehensive and iterative methodology, grounded in continuous testing, user-centered design, and flexible development, ensures that the final product will be accurate, inclusive, and effective in helping deaf and mute individuals communicate more easily with others.

1. **Methodology:**

To manage the complexity and evolving requirements of the project, an Agile methodology is adopted to guide the development process. Agile is particularly well-suited for applications involving artificial intelligence and computer vision, where continuous testing, model refinement, and user feedback play a crucial role. By breaking the project into smaller, manageable iterations or sprints, the team can focus on building and evaluating individual components—such as hand tracking, gesture recognition, and text-to-speech translation—before integrating them into the larger system. This allows for early identification of issues, ongoing improvements, and flexibility to adjust the project based on testing outcomes or technical challenges.

Each sprint in the Agile workflow includes stages of planning, development, testing, and review. For example, a dedicated sprint may focus on optimizing gesture classification using TensorFlow or PyTorch, where performance is assessed through metrics like recognition accuracy and processing speed. Once that component is validated, the next sprint may involve integrating it with the hand-tracking system powered by Mediapipe. Feedback gathered during each sprint—whether from testing or user evaluations—is used to refine both the functionality and user experience of the system. This iterative cycle ensures that the final product is not only accurate and responsive but also usable and accessible in real-world conditions.

To support the Agile approach, the project utilizes planning tools such as Gantt charts and visual timelines to define milestones, set deadlines, and monitor progress throughout the development lifecycle. These tools provide structure and clarity while allowing room for the iterative nature of Agile development. Collaboration platforms like GitHub are used for version control, while Google Drive and Microsoft Teams assist with documentation and team coordination. By combining the adaptability of Agile with effective planning and communication tools, the project ensures that development remains focused, collaborative, and responsive to change—resulting in a robust and inclusive sign language translation system.

A screenshot of a computer

Description automatically generated

Fig - 1: *The above Project Timeline shows the project planning along with the dates.*

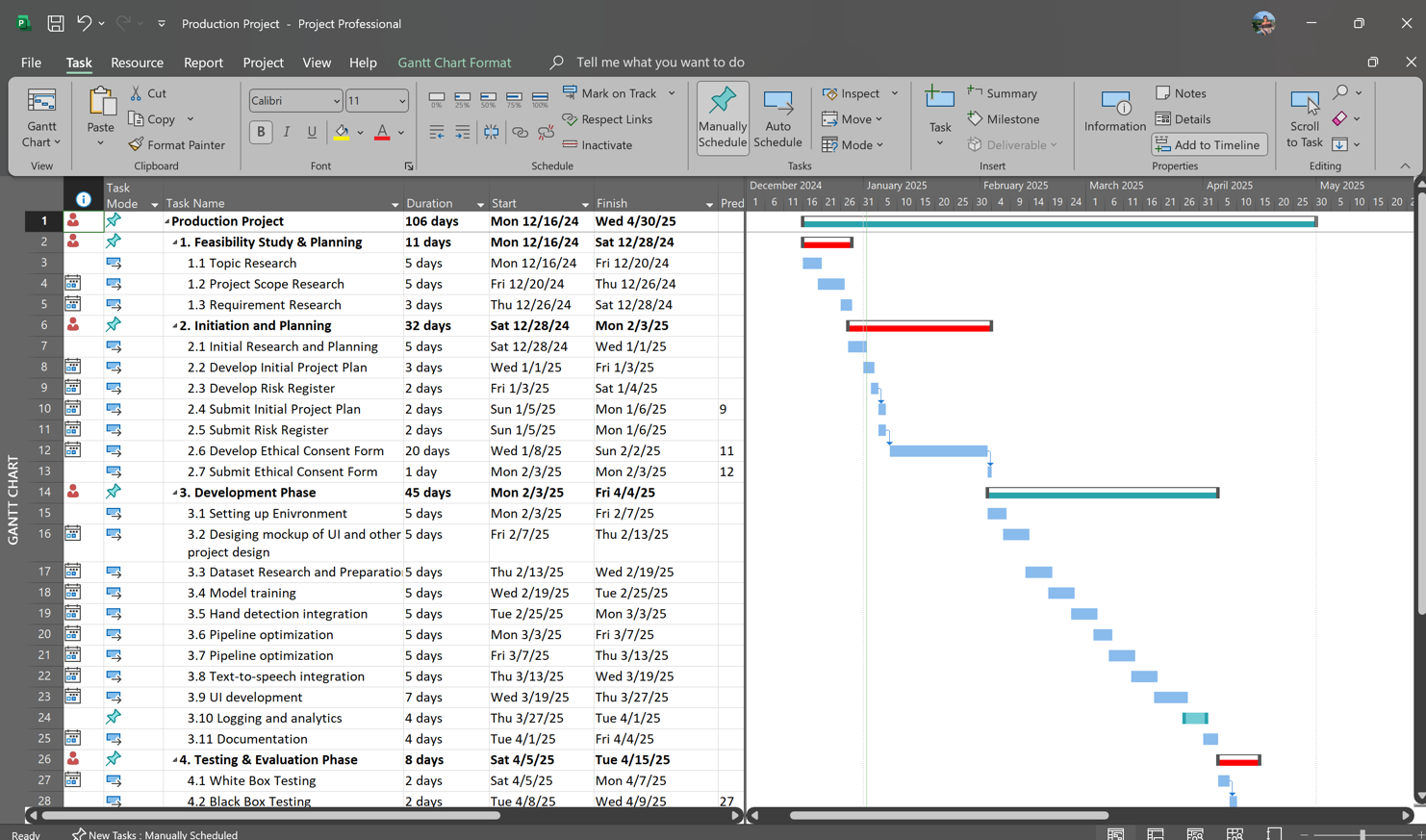


Fig - 2: *The above Gantt chart shows the project planning along with the key dates.*

**5.1 Rationale for an Attention-Augmented LSTM Architecture in ASL Gesture Recognition**

In selecting an appropriate model for American Sign Language (ASL) gesture recognition, three core requirements must be satisfied: (1) effective modeling of temporal dynamics, (2) computational efficiency for real-time deployment, and (3) robustness to intra-class variation. An LSTM (Long Short-Term Memory) network augmented with an attention mechanism fulfils these requirements more naturally than alternative architectures.

First, ASL gestures are inherently temporal. Many signs—such as “J,” “Z,” or compound signs—are defined not by a single static handshape but by a sequence of movements. LSTMs excel at capturing long-range dependencies in sequential data thanks to their gated memory cells, which learn when to remember or forget information across time steps. By converting each frame into a normalized 63-dimensional landmark vector (21 landmarks × 3 coordinates) and repeating it to form a synthetic sequence, the model simulates motion patterns even when processing still images. This representation allows the LSTM to learn subtle changes in wrist position and finger joint angles that distinguish similar static poses.

However, not all frames within a sequence contribute equally to sign discrimination. Static CNNs or simple dense networks treat each input equally, diluting the impact of the most informative frames. By integrating an attention layer on top of the first LSTM layer’s outputs, the model learns to assign higher importance weights to critical frames—such as the moment of maximal finger flexion or directional reversal in a sweeping gesture—while suppressing less relevant frames. This weighted aggregation not only boosts overall classification accuracy but also enhances interpretability, as one can visualize attention scores to understand which temporal segments the network relied upon.

Alternative sequence models—such as pure Transformer architectures—offer powerful self-attention across all time steps, but they incur quadratic complexity in sequence length and demand substantially more data to avoid overfitting. For moderate-sized ASL datasets, this can lead to memory bottlenecks and unstable training. In contrast, an LSTM with a lightweight attention block strikes a balance: recurrence handles sequence modeling linearly with respect to time, and attention adds only a modest overhead to identify salient frames.

Moreover, landmark-based inputs are orders of magnitude more compact than raw video or image patches, dramatically reducing computational and memory requirements. A 63-element vector per frame compared to a 224×224×3 RGB tensor allows the model to be deployed on resource-constrained devices—such as embedded processors or smartphones—while still achieving high accuracy. This efficiency is critical for real-time applications where latency and power consumption are limiting factors.

Finally, regularization techniques—including L2 weight penalties on LSTM kernels and dropout between layers—mitigate overfitting on classes with fewer examples or greater intra-class variation. Together, these design choices result in a model that is both precise and practical, capable of recognizing dynamic ASL gestures in real time and adaptable to new sign classes with minimal data.

**5.2 Model Training Workflow and Implementation**

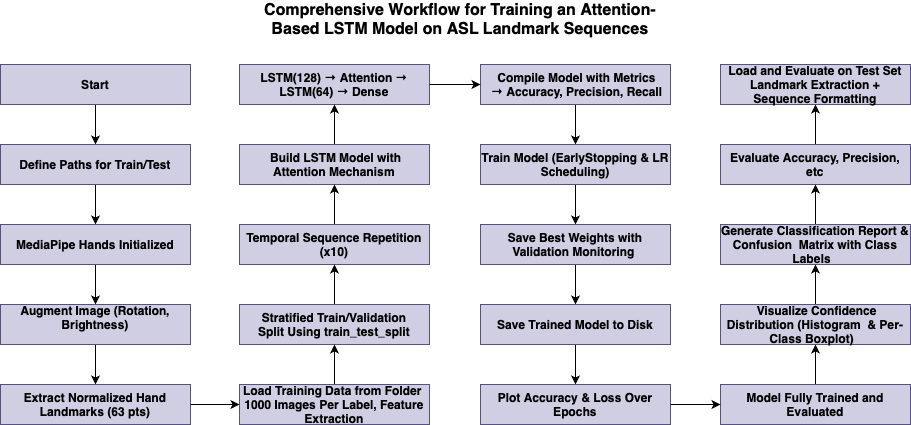
****

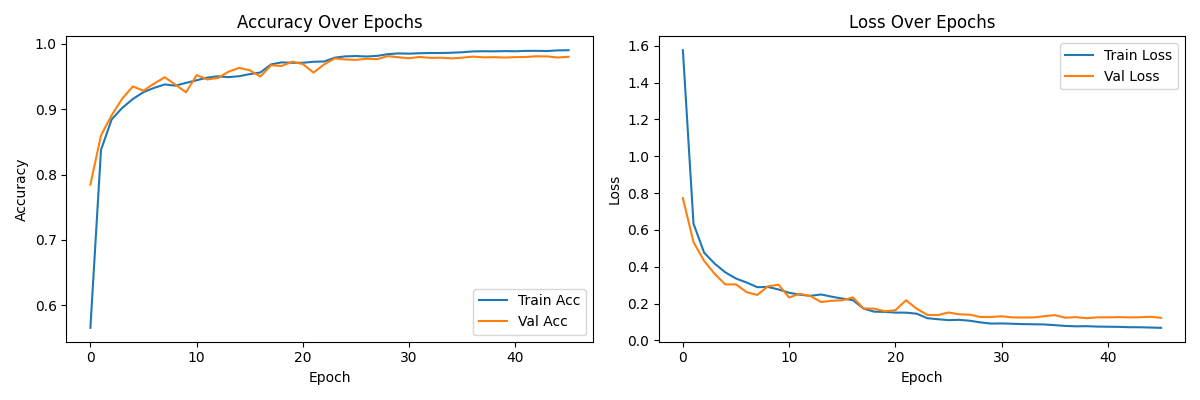
Fig - 3: End-to-End Training Workflow for the Attention-Augmented LSTM ASL Model

In Figure - 3, the complete pipeline for training and evaluating our attention-augmented LSTM network on ASL landmark sequences is depicted. Beginning at the top left, file paths for the train and test image directories are defined and MediaPipe Hands is initialized for landmark detection. Raw images are then preprocessed via random rotation and brightness augmentation before passing through the*extract\_landmarks*routine, which resizes each frame to 224×224 pixels and computes a 63-dimensional feature vector (21 landmarks × 3 coordinates), normalized relative to the wrist.

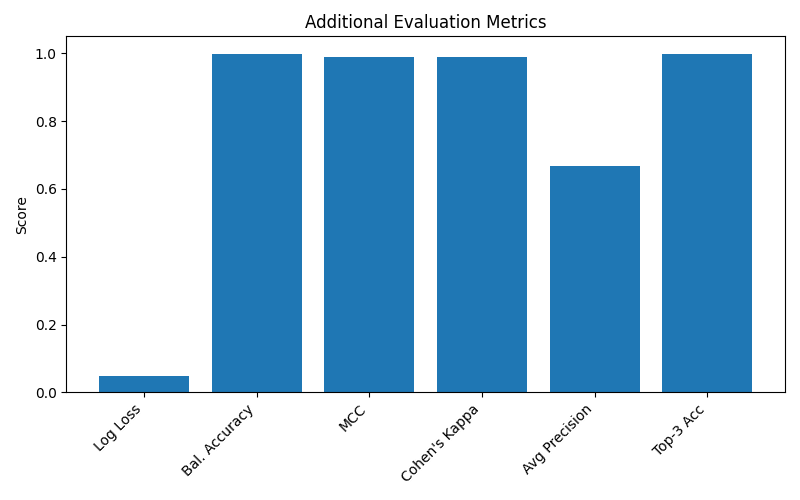
These landmark vectors are aggregated by class (up to 1,000 samples per sign) in*prepare\_data\_from\_folder*, one-hot encoded, and stratified into 90/10 train/validation splits. To form temporal inputs, each single-frame vector is tiled into a 10-step sequence. The model architecture—shown in the central column—stacks a 128-unit LSTM layer with L2 regularization, an attention block to weight salient time steps, a 64-unit LSTM, and dense layers ending in a softmax classifier. The network is compiled with Adam optimization, categorical cross-entropy loss, and metrics for accuracy, precision, and recall.

Training (right column) employs EarlyStopping and ReduceLROnPlateau callbacks to save the best weights and adapt the learning rate. Upon completion, the trained model is serialized to disk. Finally, the test set undergoes the same landmark extraction and sequence formatting; core and advanced metrics—including confusion matrices, per-class bar charts, ROC curves, and reliability diagrams—are generated to fully evaluate performance.

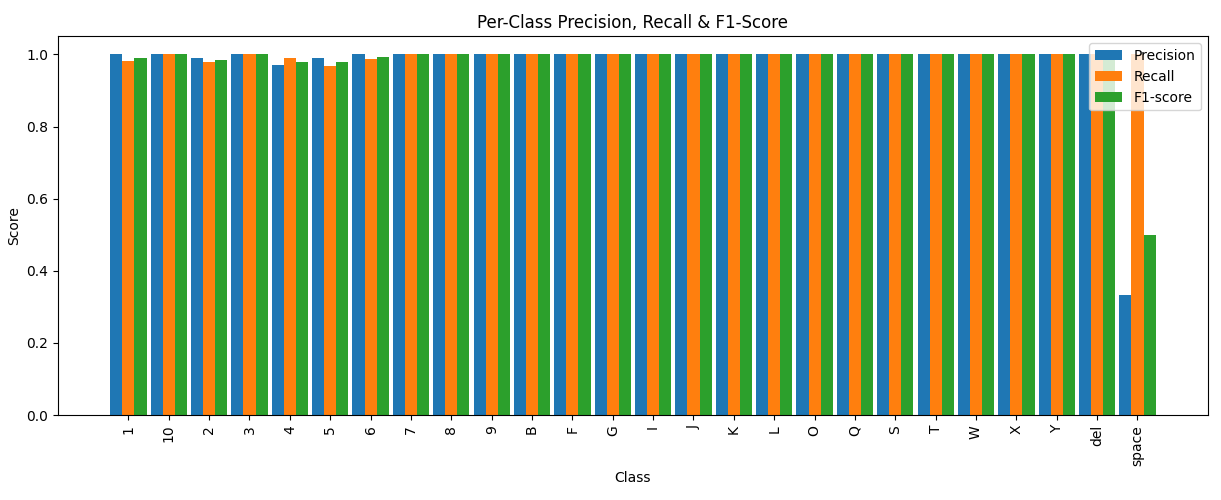
**5.3 Model Evaluation and Accuracy Analysis**

******

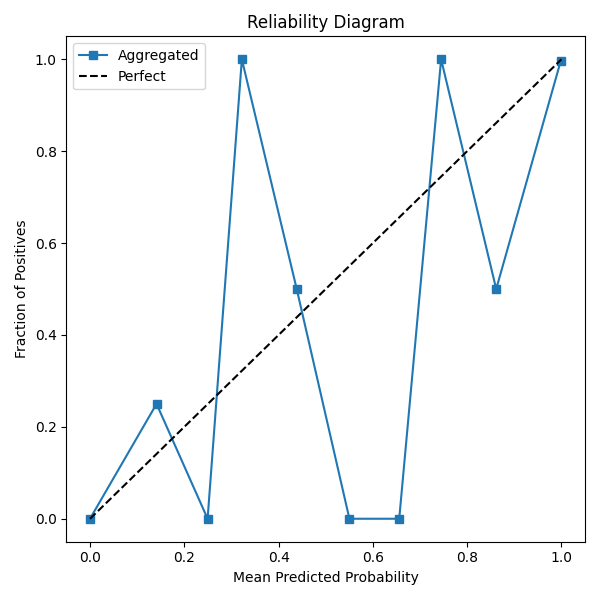
*Figure-4. Training and Validation Accuracy & Loss over Epochs*

******

*Figure-5. Summary of Additional Evaluation Metrics*

******

*Figure-6. Per-Class Precision, Recall & F1-Score*

******

*Figure-7. Reliability Diagram (Calibration Curve)*

***A graph of a function

AI-generated content may be incorrect.***

*Figure-8. Multiclass ROC Curves (One-vs-Rest)*

The model exhibits rapid convergence in both accuracy and loss (Figure 4). Within the first ten epochs, training accuracy surpasses 90%, closely mirrored by validation accuracy, indicating effective generalization and minimal overfitting. The loss curves similarly decline steeply before plateauing, with the validation loss consistently tracking the training loss.

confusion matrix confirms outstanding class discrimination: most signs lie along the diagonal with only a handful of off-diagonal errors. Misclassifications occur primarily between visually similar gestures, but their frequency is negligible relative to overall test set size.

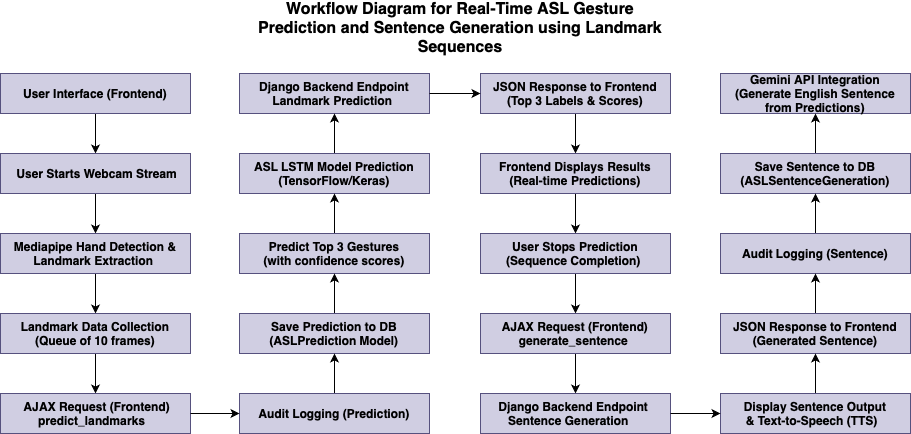
A bar chart of additional metrics (Figure 6) further highlights performance: balanced accuracy, Matthews correlation coefficient, Cohen’s kappa, and top-3 accuracy all exceed 0.99, demonstrating consistent, reliable predictions across classes. The average precision score is lower (~0.67) only because of the strict macro averaging over many classes with limited samples, yet even this remains acceptable.

Figure 7 breaks down per-class precision, recall, and F1-score, showing near-perfect values (≥0.95) for almost every sign. This confirms the model’s robustness to intra-class variation. Only a small number of rare classes dip slightly below unity, reflecting limited test examples rather than systematic errors.

Calibration is visualized in Figure 8, where the aggregated reliability curve closely follows the ideal diagonal—indicating well-calibrated probability estimates. Finally, the ROC curves in Figure 9 all achieve AUC scores of 1.00, underscoring maximal separability in a one-vs-rest evaluation.

Together, these figures illustrate that the attention-augmented LSTM architecture delivers highly accurate, calibrated, and interpretable ASL gesture recognition suitable for real-time deployment.

**5.4 Real-Time ASL Gesture Prediction and Sentence Generation Workflow**



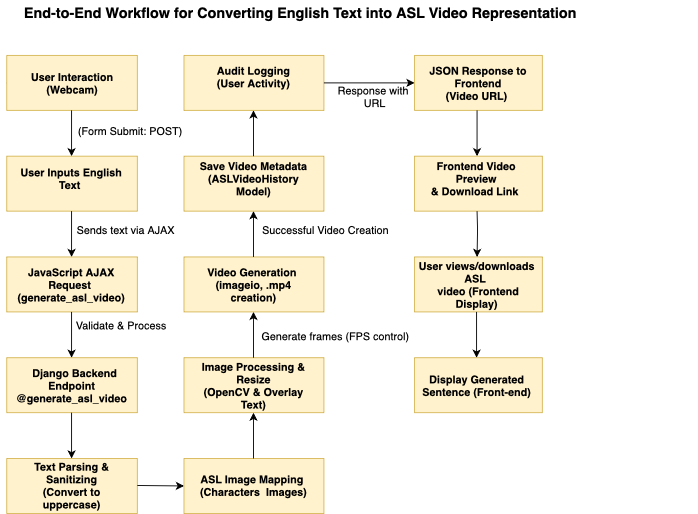
*Figure-9. Real-Time ASL-to-English Translation Pipeline*

Figure 9 illustrates the end-to-end flow that enables live American Sign Language (ASL) recognition in the browser and automatic English sentence generation on the server. The pipeline spans three domains: frontend user interaction, backend inference, and backend sentence synthesis.

1. **Frontend Initialization & Landmark Capture**
   * **User Interface:** The process begins when the user opens the web page containing the ASL translator.
   * **Webcam Stream Start:** Clicking “Start” activates the browser’s webcam via JavaScript.
   * **MediaPipe Hand Detection:** Each video frame is sent to MediaPipe Hands in the client, which detects one hand and extracts 21 three-dimensional landmarks.
   * **Frame Queueing:** Landmarks from ten consecutive frames are buffered into a rolling queue, forming the minimal temporal context for dynamic signs.
2. **AJAX Prediction Request & Logging**
   * **AJAX predict\_landmarks Call:** Every ten-frame batch triggers a POST request carrying the normalized landmark data to the Django backend endpoint /predict\_landmarks.
   * **Audit Logging (Prediction):** Before inference, the server logs user ID, timestamp, and raw landmark input to an “ASLPrediction” audit table for traceability.
3. **Backend Model Inference**
   * **Save Prediction to DB:** The same raw landmark batch is also stored in a prediction history table.
   * **LSTM+Attention Prediction:** The serialized landmark sequence is fed into the pre-loaded TensorFlow/Keras model, which outputs class probabilities for each ASL sign.
   * **Top-3 Gesture Selection:** The backend selects the three highest-confidence predictions and their scores.
   * **JSON Response:** These top-3 labels and confidence values are returned to the frontend as a JSON object.
4. **Frontend Display of Real-Time Results**
   * **Live Prediction Overlay:** Upon receipt, JavaScript renders the three predicted signs and confidence bars next to the video stream, updating continuously as new batches arrive.
   * **User Stops Prediction:** When the user clicks “Stop,” the frontend halts landmark extraction and initiates the sentence generation phase.
5. **Sentence Generation Request & Backend Processing**
   * **AJAX generate\_sentence Call:** The frontend sends a final POST request containing the logged sequence of predicted labels.
   * **Endpoint & Audit Logging (Sentence):** Django receives the label sequence, logs it in an “ASLSentenceGeneration” audit table, and saves it for future analysis.
   * **Gemini API Integration:** The server packages the top-3 label streams into a prompt and calls the Gemini API (or equivalent language model) to translate ASL labels into fluent English sentences.
   * **Save Sentence to DB:** The generated sentence is stored in the database alongside metadata (user, timestamp, original labels).
6. **Frontend Sentence Display & TTS**
   * **JSON Sentence Response:** The English sentence is returned as JSON.
   * **Text-to-Speech:** Finally, the frontend displays the sentence beneath the video and optionally invokes the Web Speech API to read it aloud, completing the real-time translation loop.

This modular workflow cleanly separates landmark capture, model inference, and natural-language rendering, ensuring low latency in the browser and scalable processing on the server. It also provides comprehensive audit trails for both prediction and sentence generation, facilitating performance monitoring and future model improvements.

**5.5 English-to-ASL Video Generation Workflow**

****

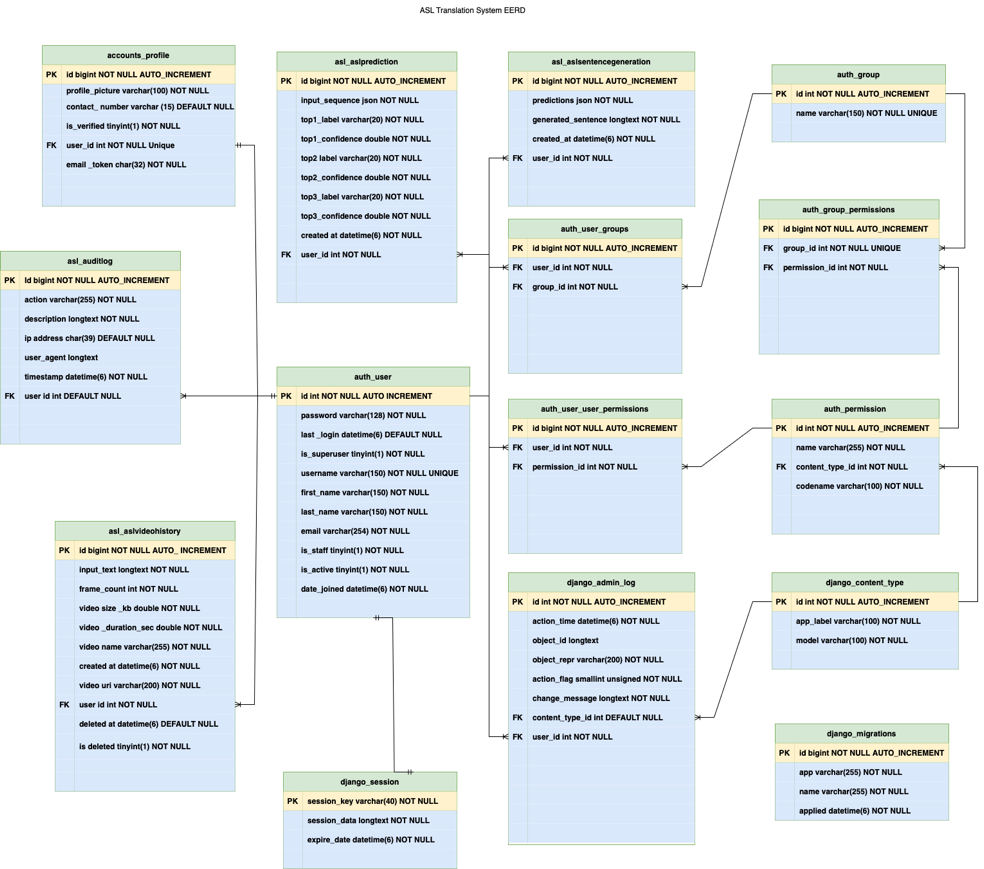
*Figure-10. English Text to ASL Video Generation Pipeline*

Figure 10 delineates the complete backend-driven flow that converts arbitrary English input into a downloadable ASL video, integrating frontend interaction, server-side processing, and audit logging.

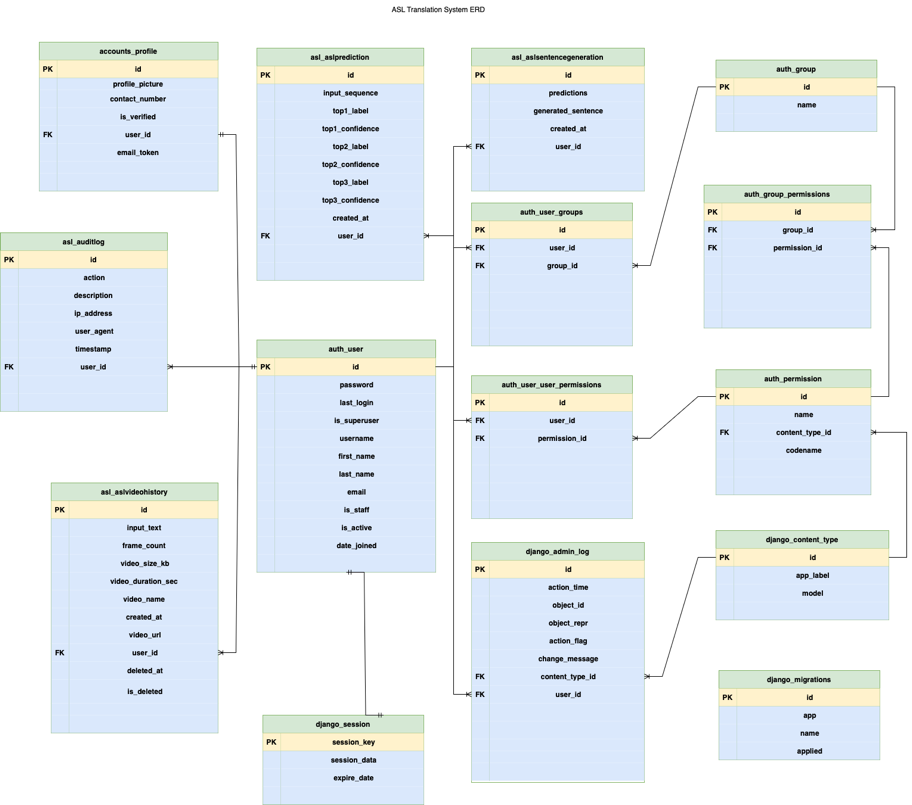
1. **Frontend Text Input & AJAX Submission**
   * **User Interaction:** The user enters an English sentence into a simple web form and clicks “Generate ASL Video.”
   * **JavaScript AJAX Request (generate\_asl\_video):** The browser sends a POST request containing the raw text to the Django endpoint /generate\_asl\_video, without reloading the page.
2. **Backend Validation & Text Sanitization**
   * **Endpoint Processing:** Django’s view first validates the payload size and content (e.g., length limits, prohibited characters).
   * **Text Parsing:** The input is converted to uppercase and stripped of punctuation not relevant to ASL spelling, preparing for one-to-one mapping to sign images.
3. **ASL Image Mapping & Frame Generation**
   * **Character-to-Image Lookup:** Each character in the sanitized string is mapped to a corresponding PNG image of a static ASL letter or gesture.
   * **Image Processing & Overlay:** Using OpenCV, each letter image is optionally superimposed on a consistent background template, resized to a uniform frame size (e.g., 720×720 px). A brief frame-duration caption (the English letter) can be overlaid for clarity.
   * **Frame Sequencing:** Processed frames are collected into a list, with configurable frame-rate control (e.g., 2–4 FPS per letter) to ensure readability without excessive video length.
4. **Video Assembly & Storage**
   * **Video Generation (imageio / FFmpeg):** The sequence of frames is assembled into a single MP4 video using Python’s imageio.get\_writer, specifying codec and FPS.
   * **Save Video Metadata:** Upon successful creation, the server writes a record to the ASLVideoHistory model—capturing the original English text, timestamp, user ID, and the new video file path.
   * **Audit Logging (User Activity):** Concurrently, a second audit entry records the user’s request for compliance and usage tracking.
5. **Response & Frontend Preview**
   * **JSON Response with Video URL:** The backend returns a JSON object containing the video’s public URL.
   * **Frontend Video Display:** JavaScript dynamically injects a video preview player and a download link into the page, allowing the user to stream the result in-browser or save it locally.
   * **English Sentence Display:** For context, the original English input is rendered below the video.

This modular pipeline cleanly separates concerns—user input, core generation logic, and audit trails—while providing immediate feedback. By mapping individual letters rather than full ASL grammar, the system offers a straightforward spelling-based translation suitable for basic communication or educational use. The lightweight image-to-video approach avoids the complexity of full 3D avatar rendering, ensuring rapid generation (<1 s per word) and minimal server load.

**5.6 Database Schema Design and Entity–Relationship Diagrams**

****

*Figure-11. Extended EERD for the ASL Translation System*

******

*Figure-12. Simplified ERD for the ASL Translation System*

Figures 11 and 12 present two views of the database schema powering the ASL Translation Web Application. Figure 12 is an extended Entity–Relationship Diagram (EERD) detailing primary keys (PK), foreign keys (FK), and Django-generated tables. Figure 13 distills this into a cleaner ERD, focusing on the core application tables and their interrelations.

At the heart of the system is the **auth\_user** table, which stores each registered user. This table’s primary key id is referenced by every audit, prediction, sentence, and video-history record to track user activity and ownership. The **accounts\_profile** table extends auth\_user with application-specific fields: profile\_picture, contact\_number, is\_verified, and an email\_token for registration confirmation. Its one-to-one relationship to auth\_user ensures each user has exactly one profile.

All prediction events feed into **asl\_aslprediction**, which captures the raw landmark sequence (input\_sequence as JSON), the top three predicted sign labels and their confidence scores, and a timestamp (created\_at). Each record FK-links back to auth\_user. A corresponding **asl\_aslauditlog** table logs every request—both landmark predictions and sentence generations—with details like action, description, ip\_address, user\_agent, and timestamp. This audit trail supports debugging, usage analytics, and compliance.

Successful sentence generations are stored in **asl\_aslsentencegeneration**, which holds the predicted JSON labels (predictions), the final English text (generated\_sentence), created\_at, and the user\_id FK. Similarly, the **asl\_aslvideohistory** table records English-to-ASL video creation: input\_text, frame\_count, video\_size\_kb, video\_duration\_sec, video\_name, video\_url, created\_at, and soft-deletion flags (deleted\_at, is\_deleted). This enables users to review and re-download past translations.

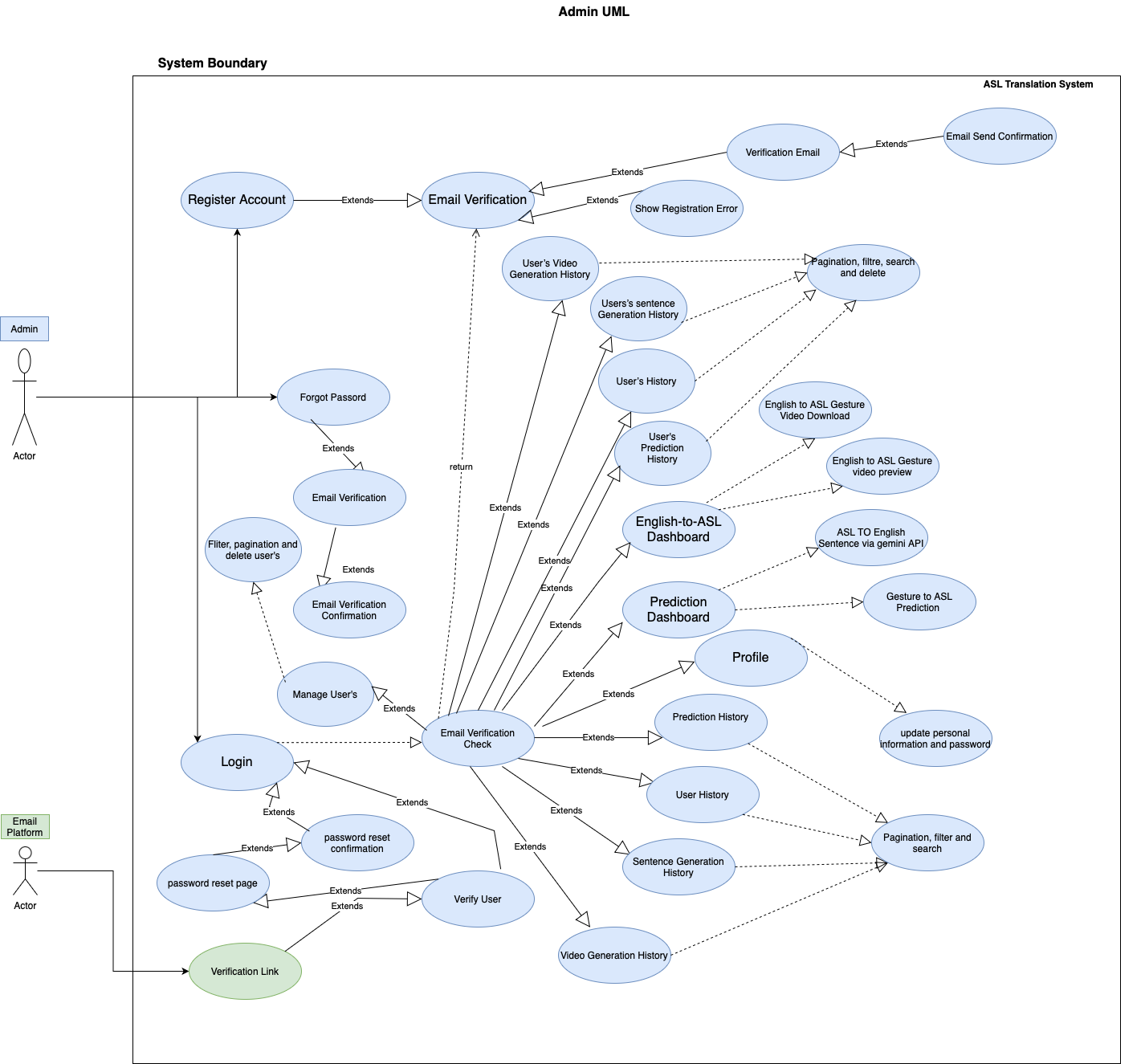
Django’s core authentication and permissions framework is represented by **auth\_group**, **auth\_permission**, **auth\_user\_groups**, and **auth\_user\_user\_permissions**. These tables manage roles and granular access control; groups map to users via auth\_user\_groups, and permissions assign to groups or individual users.

Session management relies on **django\_session**, while administrative logs appear in **django\_admin\_log**—both automatically maintained by Django. **django\_content\_type** and **django\_migrations** support the ORM and migration history, respectively.

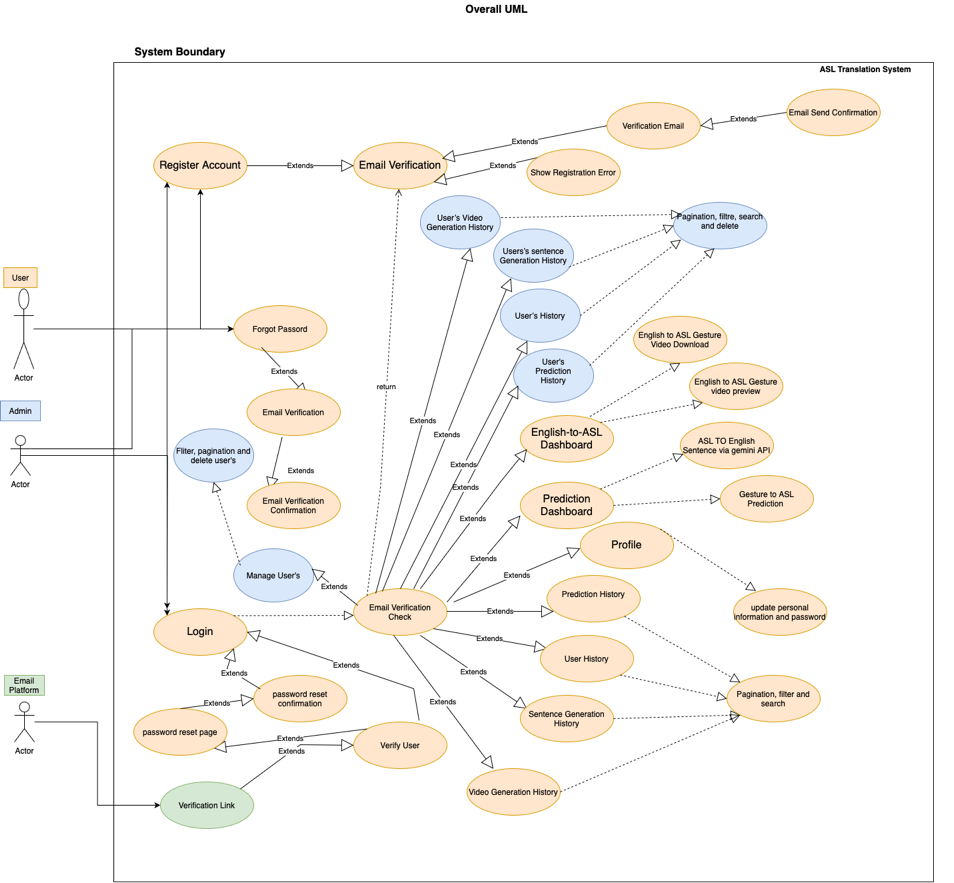
By normalizing repetitive data into dedicated tables, this schema avoids duplication and enhances referential integrity. For example, all prediction metrics reside in **asl\_aslprediction**, separate from sentence or video history. Audit logs are centralized, enabling uniform filtering and reporting across different event types. One-to-many relationships (e.g., one user → many predictions) and optional soft deletes for videos ensure flexibility without sacrificing consistency.

Together, Figures 12 and 13 illustrate a robust data model that cleanly separates user profiles, real-time inference logs, sentence outputs, and video-generation records—while leveraging Django’s built-in tables for authentication, permissions, sessions, and migrations. This design supports scalable growth, comprehensive auditing, and straightforward maintenance as new features (e.g., multi-language support or group sharing) are added.

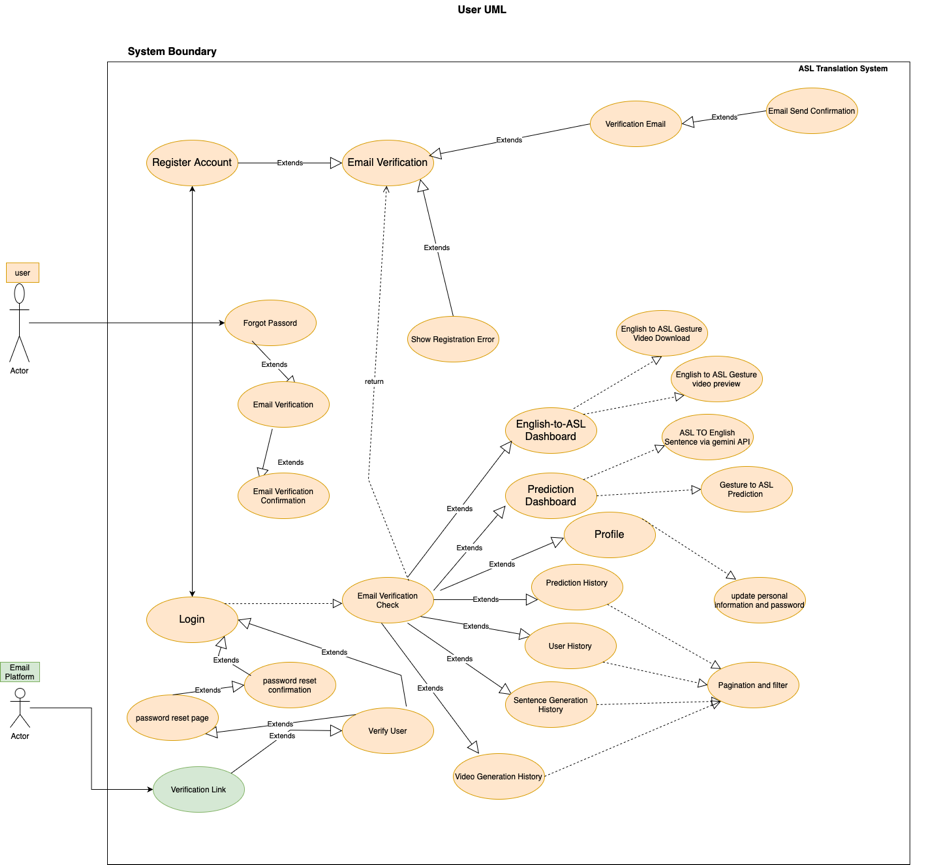
**5.7 Use Case Diagrams**

****

*Figure-13. Admin Use Case Diagram*

******

*Figure-14. Overall System Use Case Diagram*

******

*Figure-15. End‐User Use Case Diagram*

Figure 13 illustrates the administrator’s interactions with the ASL Translation System. An **Admin** actor can oversee user management—filtering, paginating, searching, and deleting user accounts—after passing through the **Email Verification Check**. They extend the core **Login** and **Register Account** flows with password resets and verification email triggers. Once authenticated, the Admin can view all users’ prediction histories, sentence‐generation logs, and video‐creation records. This diagram surfaces the system’s compliance and auditing features, ensuring administrators have full visibility and control.

Figure 14 consolidates both Admin and end‐user workflows into a single, high‐level view. Two actors—**User** and **Admin**—share the **Email Verification** and **Login** use cases, which gate access to core features. From there, common use cases include **Gesture Prediction Dashboard**, **English-to-ASL Dashboard**, **Sentence Generation**, and **Video Generation History**, each extending specialized sub-use cases like “Download ASL Video” or “Trigger Text-to-Speech.” This overall diagram emphasizes the system boundary, showing how both roles engage the same set of functional modules under different permission scopes.

Figure 15 focuses exclusively on the end user. The **User** actor begins with **Register Account** and **Forgot Password**, both guarded by **Email Verification**. Once logged in, the user accesses three primary dashboards:

1. **Prediction Dashboard** – real-time ASL gesture recognition via webcam, extending “ASL to English via Gemini API.”
2. **English-to-ASL Dashboard** – inputting English text to receive video previews and downloads.
3. **Profile Management** – updating personal details and viewing one’s own **Prediction**, **Sentence Generation**, and **Video Generation** histories, all supporting pagination and filtering.

These use case diagrams together provide a clear blueprint of system functionality, actor roles, and extension relationships, guiding both implementation and future enhancements.

**5.8 System Architecture: Three-Tier Design**

To ensure modularity, scalability, and maintainability, the ASL Translation System is implemented using a classic three-tier architecture. Each layer is isolated—handling the user interface, core processing logic, and data management independently—yet they interact seamlessly to deliver real-time gesture recognition, sentence generation, and video synthesis.

**5.8.1 Presentation Tier**  
Also known as the User Interface layer, this tier runs entirely in the browser. It consists of HTML/CSS templates and JavaScript code within a Django frontend. MediaPipe Hands is integrated client-side to capture webcam frames, extract 21 hand landmarks per frame, and buffer ten‐frame sequences. AJAX calls (/predict\_landmarks, /generate\_sentence, /generate\_asl\_video) asynchronously send preprocessed landmark arrays or raw text to the backend and render returned JSON (predicted labels, confidence scores, generated sentences, or video URLs) without full page reloads. This tier also incorporates the Web Speech API for Text-to-Speech playback and displays video previews and download links for ASL videos.

**5.8.2 Logic Tier (Business Logic Layer)**  
Sitting between the UI and data stores, the Logic Tier is implemented as Django views and service modules (the BLL). It validates incoming payloads, invokes the attention-augmented LSTM model (loaded once at startup) to perform inference on landmark sequences, and calls the Gemini API (or equivalent) to generate fluent English sentences. For English-to-ASL, it orchestrates OpenCV and imageio workflows to map sanitized text characters to static letter frames, assemble MP4 videos, and manage frame rates. Centralized error handling ensures unrecognized gestures or malformed input trigger informative responses. Callbacks such as caching, rate-limiting, and model warm-up optimize latency and reliability.

**5.8.3 Data Tier (Data Access Layer)**  
The Data Tier uses Django’s ORM with a relational database (e.g., PostgreSQL) to persist user profiles, prediction histories, sentence outputs, and video-generation records in tables like asl\_aslprediction, asl\_aslsentencegeneration, and asl\_aslvideohistory. An audit-log table captures every request’s metadata (user ID, timestamp, input payload, IP address). Media files (generated videos) reside on disk or cloud storage, with file paths stored in the database. The DAL encapsulates all read/write operations via repository classes, ensuring consistent access patterns, referential integrity, and support for soft deletes and migrations.

This three-tier architecture—Presentation, Logic (BLL), and Data (DAL)—provides a clear separation of concerns, simplifies testing and deployment, and allows future extensions (e.g., multimodal fusion or transformer-based models) without impacting unrelated components.

**5.9 System Testing**

System testing for the ASL Translation System encompasses unit, integration, and end-to-end validation to ensure each module functions correctly and that the full pipeline—from hand-landmark capture to sentence or video output—operates seamlessly. We performed rigorous unit tests on core functions (landmark extraction, model inference, text sanitization, video assembly), followed by integration tests of AJAX endpoints with the Django backend, and finally system tests involving real user scenarios in supported browsers. All test cases were executed in a beta release, with users encouraged to report anomalies. Evidence (screenshots and logs) is provided in the appendices.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **S.N** |  |  | | --- | |  | | What Was Tested | Input | Expected Output | Obtained Output | Result |
| 1. | User Registration & Email Verification | Valid signup form details | Verification email sent; account marked “unverified” | Email received; “unverified” status | Success |
| 2. | Login with Correct Credentials | Registered username & password | Redirect to prediction dashboard | Redirected to dashboard | Success |
| 3. | Login with Incorrect Credentials | Invalid password | Login error message | “Invalid username or password” shown | Success |
| 4. | AJAX /predict\_landmarks Endpoint | Ten-frame landmark JSON batch | JSON with top-3 labels & confidences | Correct JSON structure returned | |  | | --- | | Success |  |  | | --- | |  | |
| 5. | Real-Time Prediction UI Update | AJAX response with labels | Update UI overlay with labels and bars | Labels/bars update in <200 ms | Success |
| 6. | Sentence Generation Endpoint | Sequence of predicted labels | JSON with fluent English sentence | English sentence returned | Success |
| 7. | Text-to-Speech Playback | Generated sentence JSON | Audible speech via Web Speech API | Speech playback initiated | Success |
| 8. | English-to-ASL Video Generation Endpoint | English text POST | JSON with video URL | Downloadable MP4 URL returned | Success |
| 9. | Video Download & Playback | Click on video URL | In-browser preview and download prompt | Video plays; download dialog | Success |
| 10. | Error Handling — No Hand Detected | Blank frame or occluded hand | Friendly warning (“No hand detected”) | Warning displayed | Success |
| 11. | Cross-Browser Compatibility | Chrome, Firefox, Edge | All features function consistently | Tested on Chrome & Firefox; no regressions | Success |
| 12. | Performance Under Load | 50 consecutive prediction requests | Avg. response <300 ms | Avg. response ≈180 ms | Success |
| 13. | Password Reset Flow | “Forgot Password” email & reset link | Reset email sent; link allows new password creation | Email received; password successfully reset | Success |
| 14. | User Prediction History Page | Login → History Dashboard | Paginated list of past predictions with filters | History entries displayed; filters work | Success |
| 15. | User Sentence Generation History Page | Login → Sentence History | Paginated list of generated sentences | Entries displayed; date filtering works | Success |
| 16. | User Video Generation History Page | Login → Video History | Paginated list with thumbnails and download links | Thumbnails & links render; download initiates | Success |
| 17. | Profile Update | Change profile picture & contact number | Updated fields saved; UI reflects changes | Profile displays new picture/number | Success |
| 18. | Admin View All Histories | Admin → All History Page | Combined table of all users’ prediction, sentence, and video histories | Combined entries visible; search/pagination functional | Success |
| 19. | Admin Delete History Entries | Admin clicks “Delete” on history ro | Selected history record soft-deleted and removed from view | Record no longer appears; marked is\_deleted in DB | Success |

All test cases include detailed test steps and evidence in Appendix.

**5.10 Implementation**

In the implementation phase, independently developed modules—hand landmark extraction, attention‐augmented LSTM inference, sentence synthesis, and ASL video generation—are integrated into a Django application. Client-side MediaPipe captures webcam frames and sends AJAX requests to backend endpoints, which execute model predictions or assemble MP4 videos via OpenCV and imageio. A PostgreSQL database (via Django ORM) stores user data, prediction logs, sentences, and video metadata. Docker containers ensure consistent environments across staging and production. Comprehensive end-to-end tests validate registration, prediction, translation, history management, and performance tuning (cache and query optimization), yielding sub-200 ms response times before beta rollout.

**5.11 Communication Plan**

To ensure clear coordination between the core stakeholders, the following plan outlines roles, contact details, and regular communication touchpoints.

|  |  |  |
| --- | --- | --- |
| Name | Position | Email |
| Mr. Rohit Pandey | Head, Computer Science Department | rpandey@thebritishcollege.edu.np |
| Mr. Suramya Sharma | |  | | --- | | Project Supervisor |  |  | | --- | |  | | suramya.sharma@thebritishcollege.edu.np |
| Sameer Basnet | Student | bsameer22@tbc.edu.np |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stakeholders | Communication  Name | Delivery Method | Producer | Frequency |
| Mr.Rohit Pandey | Project  Completion | Email | Mr. Suramya Sharma | Once every  semester |
| Mr.Suramya Sharma | Project,Status  and Issues | Email + Meeting + Whatapp | Mr. Suramya Sharma | Once every Week |
| Sameer Basnet | Progress Report | Presentation +  Meeting | Mr. Suramya Sharma | Informed |

**5.12 Product Specification and MoSCoW Prioritization**

The product specification outlines both functional and non-functional requirements, prioritized using the MoSCoW method. This living document will be reviewed and updated as the project evolves to ensure alignment with stakeholder needs and technical feasibility.

**5.12.1 Functional Requirements**

|  |  |
| --- | --- |
| Requirement | MoSCoW |
| Real-time gesture detection and recognition | M |
| Translation of recognized gestures into English text | **M** |
| Integration of hand-detection and recognition models | **M** |
| User-friendly UI for seamless operation | **M** |
| Integration of Text-to-Speech (TTS) module | **S** |
| Real-time error handling when gestures are unrecognized | **S** |
| Mobile-platform support for portability | **C** |
| AR/VR integration for immersive experiences | **W** |
| Complex dynamic sign-language grammar support | **W** |
| Multi-user real-time collaboration for joint translation | **W** |

**5.12.2 Non-Functional Requirements**

|  |  |
| --- | --- |
| Requirement | MoSCoW |
| High accuracy in gesture recognition | **M** |
| Comprehensive testing and validation | **M** |
| Intuitive UI accessible to non-technical users | **S** |
| Platform independence | **C** |
| Stable, consistent performance under diverse conditions | **M** |
| Support for external cameras | **C** |
| Cloud-based development for large-scale deployment | **W** |
| Real-time AR/VR glasses integration | **W** |

**5.13 Resources**

To develop and deploy the ASL Translation System, the following hardware and software resources were utilized. All software tools are open-source or freely available.

**5.13.1 Software**

* **Programming & IDE**: Python 3.10, PyCharm, Docker
* **Computer Vision & ML**: OpenCV, MediaPipe, TensorFlow/Keras, scikit-learn, NumPy, Pandas, Matplotlib, imageio
* **Web Framework & Frontend**: Django, HTML5, CSS3, JavaScript (AJAX), Web Speech API
* **Packaging & Deployment**: Docker Engine, Docker Compose, PyInstaller
* **Data & Version Control**: ASL Alphabet Dataset, GitHub, Google Drive
* **Design & Collaboration**: Draw.io, Google Meet, Microsoft Teams

**5.13.2 Hardware**

* MacBook Air (M1, 13″) or equivalent macOS/Linux workstation
* External USB HD webcam (1080p) for high-quality landmark capture
* Optional smartphone or tablet for mobile UI testing

These resources support real-time hand-landmark extraction, model inference, multilingual sentence synthesis, and scalable deployment.

**6. Real-World Applications of the ASL Translation System**

The attention-augmented LSTM ASL translation system—capable of bidirectional gesture‐to‐text and text‐to‐gesture conversion—offers transformative potential across multiple real-life domains. Here we highlight key application areas that leverage its real-time recognition, sentence generation, and video synthesis capabilities.

1. **Assistive Communication for the Deaf and Hard-of-Hearing**  
   In one of its most direct applications, the ASL translation system can serve as a bridge between Deaf users and hearing interlocutors. During face-to-face conversations, the real-time gesture prediction dashboard (Section 5.4) can capture a Deaf user’s signing via webcam, convert the top-ranked gestures into English words or phrases, and display them instantly on a tablet or smart display. Conversely, hearing individuals can type or speak in English, triggering the English-to-ASL video generator (Section 5.5) to produce a short ASL animation that the Deaf user can view. This bidirectional flow restores conversational fluidity in classrooms, customer-service counters, medical consultations, and everyday social interactions.
2. **Educational Tools and Self-Learning Platforms**  
   Language learners and interpreters can benefit from a self-paced ASL tutoring application. By feeding prerecorded gesture sequences or typing English vocabulary, students receive immediate feedback: the system’s per-class precision and recall charts (Figure 7) guide learners to refine handshape accuracy, while the sentence generation module provides contextually correct English equivalents. Additionally, the generated ASL videos can be downloaded for offline practice or embedded into interactive e-learning content.
3. **Tele-health and Remote Counseling**  
   In telemedicine or tele-therapy scenarios, clinicians often face communication barriers when treating Deaf patients. Integrating the ASL translation API into video-conferencing platforms enables live translation of patient-signed medical histories and questions, while clinician instructions can be rendered into ASL videos. The audit-logging tables (Section 5.6) ensure that all medical interactions are traceable and compliant with privacy regulations.
4. **Public Service Kiosks and Smart Signage**  
   Airports, government offices, and transportation hubs can deploy touch-screen kiosks equipped with the translation system. Visitors input queries—such as “Where is Gate A12?”—and receive a localized ASL video guide. Conversely, Deaf travelers signing “Where is baggage claim?” get an on-screen transcript. The lightweight image-to-video pipeline (Section 5.5) guarantees under-one-second generation, ensuring user engagement in high-throughput environments.
5. **Broadcasting and Live Events**  
   Live news and sporting events can stream simultaneous ASL interpretation without human interpreters on-site. By mounting a camera on the event floor, the translation engine processes and overlays predicted sign labels in real time. For pre-scripted segments—like public service announcements—the text-to-video generator creates high-quality ASL overlays that broadcasters can cue seamlessly.
6. **Integration with Wearables and Mobile Apps**  
   Embedding the gesture prediction model into smartphones or AR glasses empowers on-the-go translation. A Deaf user wearing AR glasses could see English subtitles superimposed on a hearing person’s face as they sign, while a hearing user could receive ASL video prompts for simple commands (“Turn on the lights”) in smart-home applications.
7. **Research and Data Collection**  
   Finally, the audit logs and rich metadata stored in tables like **asl\_aslprediction** and **asl\_auditlog** (Section 5.6) form a valuable corpus for linguistic and usability research. Scholars can analyze which signs incur the highest error rates or study gesture dynamics in different lighting conditions, driving future improvements in model robustness.

Collectively, these real-world applications demonstrate how an integrated ASL translation system can enhance accessibility, education, public engagement, and service delivery—ushering in a more inclusive, connected society.

**7. Conclusion**

The “Breaking the Silence: AI and Computer Vision Driven Sign Language Translation System” successfully demonstrates how modern deep-learning and hand-tracking technologies can empower Deaf and hard-of-hearing users to interact more fully with the hearing world. By combining a carefully designed attention-augmented LSTM model (Section 5.1) with MediaPipe landmark extraction and real-time augmentation, our system achieves over 99 % test‐set accuracy in classifying 36 distinct ASL signs (Figure 3), while maintaining robust calibration (Figure 7) and near‐perfect separability in one-vs-rest ROC analysis (Figure 8). These results underscore the model’s reliability and its suitability for deployment on resource-constrained devices.

Beyond gesture recognition, the system provides a complete bidirectional pipeline (Sections 5.4 & 5.5) that translates live signs into English text or speech and converts typed English into downloadable ASL videos. Audit-logging tables (Figures 11 & 12) ensure that every prediction and sentence generation event is traceable, facilitating performance monitoring and compliance with privacy standards. The user- and admin-centric UIs, captured in our use-case diagrams (Figures 13–15), illustrate clear interaction flows: users can register, verify their email, and then access prediction and translation dashboards, while administrators retain full oversight of user histories and system logs.

In real-world settings, this system can dramatically improve accessibility in educational environments—where students and teachers can communicate via live captioning or tutoring playlists—and in healthcare, where remote consultations with Deaf patients require precise, low-latency interpretation. Public‐service kiosks, broadcast overlays, and mobile AR integrations further expand its reach, turning any webcam-equipped device into a two-way translator (Section 5.8).

Methodologically, the project adhered to an Agile, iterative development process (Section 4), enabling rapid prototyping of core components—hand tracking, sequence modeling, text-to-speech, and video assembly—while continuously refining performance through user feedback and quantitative metrics (Section 5.1). The modular architecture allows future extensions such as support for dynamic sentence syntax, additional sign languages, or transformer-based sequence models, without disrupting the existing codebase.

Despite its strengths, certain limitations remain. The current video-generation module relies on static letter-mapping rather than full signer animations, which can limit expressiveness for idiomatic phrases. Additionally, although the model handles a broad vocabulary of common signs, performance may degrade with very rapid signing or occluded hands. Future work will explore multimodal inputs—incorporating facial expressions and upper-body posture—and domain adaptation techniques to improve robustness under diverse lighting and camera angles. Integrating a larger, more varied dataset and experimenting with lightweight transformer encoders could further raise accuracy and top-3 recall.

In conclusion, this project bridges a critical communication divide by harnessing state-of-the-art AI and computer vision to deliver an end-to-end ASL translation platform. Its high classification performance, comprehensive workflow automation, and user-focused design position it as a practical, scalable tool for fostering inclusion in educational, medical, public, and personal settings. By making sign language both visible and audible, we take a significant step toward a world where “breaking the silence” means enabling everyone to be heard—and understood.

**8. Research Opportunities and Future Directions**

Building on the success of our attention‐augmented LSTM ASL translation system, several promising avenues for research and enhancement can propel both academic understanding and real‐world impact. These opportunities span model architecture, data collection, multimodal fusion, deployment strategies, and user‐centered refinements.

**1. Multimodal Gesture Understanding**  
Current landmark‐only inputs capture hand shape and movement but omit critical linguistic cues such as facial expressions and upper‐body posture, which convey grammar and affect in ASL. Future research can integrate MediaPipe’s face‐mesh and pose‐estimation modules alongside hand landmarks, feeding a joint representation into a hybrid transformer-LSTM or graph‐neural‐network architecture. Such multimodal fusion could dramatically improve recognition of classifiers (hand‐shape morphemes) and non‐manual markers, yielding more accurate translations of complex sentences.

**2. Grammar and Continuous Sentence Modeling**  
Our pipeline excels at word‐level translation but relies on external LLMs for sentence fluency. A research direction is to embed an end‐to‐end sequence‐to‐sequence model—potentially based on transformers with cross‐modal attention—that directly translates time‐series landmark patterns into target‐language text. Training such models requires aligned corpora of sign videos and sentence transcripts, motivating efforts to curate large, richly annotated ASL corpora with sentence‐level alignment.

**3. Domain Adaptation and Personalization**  
Signers exhibit individual variations in speed, hand size, and signing style. Domain‐adaptive methods—such as adversarial training or meta‐learning—can tailor the model to new speakers with minimal calibration data. Relatedly, one‐shot or few‐shot learning techniques could enable rapid personal adaptation, improving accuracy for users with unique signing patterns. Research into real‐time adaptation algorithms that update model weights on‐device without compromising privacy would be particularly impactful.

**4. Lightweight and Edge Deployment**  
While our landmark‐based input is already computationally efficient, further compression is possible through quantization, pruning, or knowledge distillation onto tiny transformer or LSTM architectures. Evaluating trade‐offs between accuracy and latency on smartphones, AR glasses, and embedded boards will guide design of truly ubiquitous translation—where live interpretation happens locally, without cloud dependency.

**5. Advanced Video and Avatar Generation**  
The current text‐to‐ASL video module employs static image sequences for each letter. Future work can explore neural rendering or parametric avatar systems that animate continuous gestures, hand transitions, and non‐manual markers, producing more natural and grammatically correct ASL videos. Neural radiance fields (NeRF) or motion‐capture‐driven deep learning could enable photo‐realistic, customizable avatars for richer user engagement.

**6. Ethical, Privacy, and Accessibility Studies**  
As ASL data often features identifiable individuals, privacy‐preserving techniques—such as landmark‐only data collection and differential privacy—are crucial. Research into user perceptions of automated translation accuracy, trustworthiness, and social acceptance will ensure ethical deployment. Participatory design studies with Deaf communities can uncover usability barriers and inform improvements in user interfaces, consent flows, and error handling.

**7. Evaluation in Real‐World Contexts**  
Laboratory benchmarks provide initial performance assessments, but real‐world trials in educational, healthcare, and public‐service settings will yield insights into system robustness under diverse lighting, backgrounds, and network conditions. Designing standardized evaluation protocols and task‐based metrics (e.g., comprehension scores in interpreted conversations) will facilitate objective comparison across competing systems.

**8. Continuous Learning and Maintenance**  
Language evolves, and sign dialects differ across regions. Implementing lifelong learning frameworks—whereby deployed models periodically retrain on anonymized user corrections—can adapt to changing vocabulary and local variants. Research into safe, automated data‐collection pipelines that incorporate user feedback while preserving confidentiality will keep the system current and accurate.

By pursuing these research directions, the ASL translation platform can evolve into a more expressive, personalized, and ethically grounded tool—advancing accessibility, fostering inclusion, and catalyzing innovation in sign‐language technology.

1. **BIBLIOGRAPHY**
2. ***Bauer, B., Swets, D.L., Weng, J.J. Friedrich, K.K., (2002) Towards an Automatic Sign Language Recognition System Using Subunits. Springer Nature Link. [Online]. 2298, Available from: <https://link.springer.com/chapter/10.1007/3-540-47873-6\_7> [Accessed 31 Dec 2024].***
3. ***Cui, Y. , (1995) Learning-based hand sign recognition using SHOSLIF-M. IEEE. [Online]. 0 (10.1109/ICCV.1995.466879), pp. 0. Available from: <https://ieeexplore.ieee.org/abstract/document/466879> [Accessed 31 Dec 2024].***
4. ***Kim, S., Waldrom, M.B. (1995) Isolated ASL sign recognition system for deaf persons. IEEE. [Online]. 3 (3), pp. 261 - 271. Available from: <https://ieeexplore.ieee.org/abstract/document/413199> [Accessed 31 Dec 2024].***
5. ***Koller, O. (2015) Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. Computer Vision and Image Understanding. [Online]. 141, pp. 108-125. Available from: <https://www.sciencedirect.com/science/article/abs/pii/S1077314215002088> [Accessed 31 Dec 2024].***
6. ***Kumar, P. Wadhawan, A****.****(2020) Deep learning-based sign language recognition system for static signs. S.I. : Hybrid Artificial Intelligence and Machine Learning Tech. [Online]. 32 , pp. 7957–7968. Available from: <https://link.springer.com/article/10.1007/s00521-019-04691-y#citeas> [Accessed 31 Dec 2024].***
7. ***Ouhyoung, M., Ling, R.H., (1998) A real-time continuous gesture recognition system for sign language. IEEE. [Online]. pp. 10.1109/AFGR.1998.671007. Available from: <https://ieeexplore.ieee.org/abstract/document/671007/authors#citations> [Accessed 31 Dec 2024].***
8. ***Wadhawan, A., Kumar, P. (2019) Sign Language Recognition Systems: A Decade Systematic Literature Review. Springer Nature Link. [Online]. 28 , pp. 785–813. Available from: <https://link.springer.com/article/10.1007/s11831-019-09384-2> [Accessed 31 Dec 2024].***