AI-Powered Real-Time Sign Language Detection and Transcription.

Project Documentation

Members

Mark Horris Maina-SCT211-0704/2022

Emmanuel Yator SCT211-0622/2022

Sajjad Abdirahim Gedow SCT211-0065/2022

George Githigi Ngugi SCT211-0460/2022

Njoroge Kanyagia SCT211-0009/2022

Edwin Owino SCT211-0725/2021

1. Introduction

1.1 Project Overview

The Sign Language Interpreter Model (SLIM) represents a transformative initiative in assistive technology, designed to bridge the significant communication divide between the deaf and hard-of-hearing community and the hearing world. By leveraging advanced computer vision and machine learning technologies, this project aims to create a seamless, real-time translation system that converts sign language into text and speech, facilitating more natural and effective communication across language barriers.

Sign language is a rich, complex, and fully-developed linguistic system used by millions worldwide. However, the limited understanding of sign languages among the general population creates substantial barriers for deaf individuals in everyday situations—from educational settings to healthcare environments, workplace interactions, and social engagements. SLIM addresses this critical gap by providing an accessible, mobile-based solution that empowers sign language users to communicate independently in diverse environments without requiring an interpreter's physical presence.

The significance of this project extends beyond mere convenience. It represents a fundamental step toward digital inclusivity, aligning with global initiatives for accessibility and equal rights for people with disabilities. By removing communication barriers, SLIM has the potential to enhance educational outcomes, improve employment opportunities, facilitate better access to services, and foster greater social integration for deaf and hard-of-hearing individuals.

Moreover, the mobile application format ensures that this powerful technology is available whenever and wherever needed, transforming spontaneous interactions that would otherwise be challenging without interpretation services. This project embodies the principle that technological innovation should serve to enhance human connection and ensure that no one is excluded from full participation in society due to communication differences.

1.2 Problem Statement

The deaf and hard-of-hearing community faces persistent and multifaceted communication challenges that significantly impact their daily lives and opportunities. These challenges stem from a fundamental linguistic disconnect: while sign languages are complete and complex linguistic systems, they remain inaccessible to the vast majority of the hearing population.

Key challenges include:

Communication Barriers in Everyday Interactions: Simple daily interactions—ordering at restaurants, asking for directions, participating in workplace meetings, or engaging in casual conversations—become laborious or impossible without interpretation. These barriers foster isolation and prevent full participation in society.

Limited Availability of Human Interpreters: Professional sign language interpreters are both costly and in short supply, making them inaccessible for many deaf individuals, particularly for spontaneous interactions or in rural or underserved areas. Scheduling constraints further complicate the use of interpreter services.

Educational Disparities: In educational settings, deaf students often receive inadequate support, leading to lower academic achievement and limited career prospects. Real-time interpretation of lectures and classroom discussions is essential but frequently unavailable.

Healthcare Access Challenges: Medical appointments without proper interpretation can lead to misdiagnosis, inadequate treatment, or inability to provide informed consent, creating dangerous situations and healthcare inequities.

Employment Discrimination: Communication barriers in job interviews and workplace settings contribute to higher unemployment rates among deaf individuals, despite their qualifications and capabilities.

Digital Divide: While technology has improved many aspects of life, many digital platforms and services remain inaccessible to sign language users, creating a new form of exclusion in our increasingly digital society.

Psychological Impact: The constant struggle to communicate can lead to frustration, social isolation, and mental health challenges for deaf individuals, affecting their overall well-being.

Existing technological solutions have significant limitations. Text-based alternatives require literacy skills that may not be present in all sign language users, especially those for whom sign language is their first language. Video relay services still depend on human interpreter availability. Previous sign language recognition technologies have been limited by accuracy issues, restricted vocabulary, inability to process dialects or personal signing styles, and lack of portability for real-world use.

SLIM addresses these shortcomings through an AI-powered solution that can function independently on a mobile device, providing immediate, accurate, and personalized sign language interpretation without reliance on internet connectivity or human interpreters. This approach tackles the core problems of accessibility, availability, and affordability that have historically hindered communication for the deaf and hard-of-hearing community.

1.3 Objectives

The SLIM project is guided by the following comprehensive set of objectives, designed to ensure the development of a robust, user-centered sign language interpretation system:

Primary Objectives:

1. **Develop a High-Performance Sign Language Detection Model**
    - Create a computer vision model capable of accurately detecting and tracking hand movements, facial expressions, and body posture—all critical components of sign language communication.
    - Optimize the model to function effectively under varying lighting conditions, backgrounds, and camera angles.
    - Ensure the system can handle different signing speeds and styles without degradation in performance.
2. **Implement Accurate Sign Language Recognition**
    - Develop deep learning algorithms capable of recognizing discrete signs and continuous signing sequences.
    - Build a comprehensive sign language vocabulary database that covers everyday communication needs.
    - Implement natural language processing capabilities to interpret grammatical structures unique to sign languages.
3. **Enable Real-Time Transcription and Translation**
    - Achieve translation latency under 500 milliseconds to facilitate natural conversation flow.
    - Develop efficient text generation that preserves the semantic meaning of signed communications.
    - Implement speech synthesis capabilities to convert transcribed text into spoken language when needed.
4. **Ensure Mobile Optimization and Accessibility**
    - Develop an intuitive, user-friendly mobile interface adhering to accessibility best practices.
    - Optimize the AI model for mobile processing capabilities while maintaining high accuracy.

- Design the application to function offline, ensuring utility in areas with limited connectivity.
- Minimize battery consumption through efficient processing algorithms.

Secondary Objectives:

5. **Implement Adaptive Learning Capabilities**
    - Develop functionality for the system to learn and adapt to individual signing patterns and preferences.
    - Create mechanisms for users to provide feedback on translation accuracy to improve the system over time.
    - Implement continuous learning algorithms that improve with increased usage.
6. **Facilitate Two-Way Communication**
    - Develop text-to-sign visualization capabilities to enable hearing individuals to communicate with deaf users.
    - Implement speech recognition to convert spoken language into text or visual sign representation.
7. **Ensure Data Privacy and Security**
    - Implement robust data protection measures to safeguard user privacy.
    - Develop secure processing methods that minimize data storage and transmission.
    - Ensure compliance with relevant accessibility and data protection regulations.
8. **Establish Evaluation Frameworks**
    - Develop comprehensive metrics for measuring recognition accuracy, translation quality, and user satisfaction.
    - Establish testing protocols involving deaf users to validate real-world effectiveness.
    - Create benchmarks for performance comparison with existing solutions and future iterations.

These objectives collectively aim to create a transformative tool that provides accurate, efficient, and accessible sign language interpretation, empowering deaf and hard-of-hearing individuals with greater independence in communication across various life contexts.

1.4 Scope

The scope of this project includes the following areas:

1. **Supported Sign Languages**: Initially, the project will focus on supporting widely used sign languages such as American Sign Language (ASL) and British Sign Language (BSL). However, as the application evolves, it can be expanded to accommodate regional and country-specific sign languages, including Kenyan Sign Language (KSL), to meet the diverse needs of users around the world.
2. **Platforms**: The primary platform for the application will be mobile, with native support for both Android and iOS. This ensures that users can access the app from their smartphones or tablets, making it convenient and readily available in any setting. In the future, the application could be expanded to other platforms, such as web browsers or embedded systems (e.g., smart glasses, wearables), to enhance versatility.
3. **Target Users**: The key user groups for this project include:
    - **Deaf and Hard-of-Hearing Individuals**: The primary beneficiaries of the project, who will use the app to communicate with non-sign language users more easily and effectively.
    - **Educational Institutions**: Schools, colleges, and universities can adopt the application as a tool for inclusive education, allowing sign language users to participate fully in classroom activities.
    - **Healthcare Providers**: Hospitals, clinics, and other healthcare facilities can use the app to improve communication between medical staff and deaf patients, ensuring that important information is accurately conveyed.
    - **Government Agencies and Public Services**: Public service providers, such as police departments, post offices, and transportation services, can use the app to better serve individuals with hearing disabilities, promoting inclusivity in public spaces.

2. Literature Review

Existing Solutions for Sign Language Recognition

1.Glove-Based Systems:

How They Work: Use sensor-equipped gloves to detect hand movements and gestures.

Examples: SignAloud Gloves, Flex Sensors.

Limitations:

-Expensive and not portable.

-Require users to wear specialized hardware.

-Limited to predefined gestures.

Vision-Based Systems:

How They Work: Use cameras and computer vision algorithms to detect gestures.

Examples: Microsoft Kinect, OpenCV-based solutions.

Limitations:

-Struggle with complex gestures or fast movements.

-Require high computational power for real-time processing.

-Limited accuracy in diverse lighting conditions or backgrounds.

Deep Learning Approaches:

How They Work: Use neural networks (e.g., CNNs, LSTMs) to classify gestures from video data.

Examples: ASL recognition using TensorFlow, RWTH-PHOENIX dataset for German Sign Language.

Limitations:

Require large, labeled datasets for training.

Often lack real-time capabilities.

Limited to specific sign languages (e.g., ASL, BSL).

Commercial Solutions:

Examples: SignAll, MotionSavvy UNI.

Limitations:

-Expensive and not widely accessible.

-Limited to specific use cases (e.g., workplace communication).

-Lack support for multiple sign languages.

How SignSync Improves Upon Existing Systems

Hardware-Free: Uses standard cameras (webcam, smartphone) without requiring specialized hardware.

Real-Time Performance: Optimized for low-latency, real-time translation using lightweight models.

Inclusivity: Designed with input from the Deaf community for better usability and accuracy.

Scalability: Built to support multiple sign languages (starting with KSL) and dynamic gestures.

Cost-Effective: Open-source and deployable on consumer devices, making it accessible to all.

3. Methodology

3.1 Data Collection

The dataset is sourced from gathering diverse sign language datasets from various sources like the Kaggle notebook *"Kenyan Sign Language Classification 91% Accuracy"* (Hammadi, 2023). It contains KSL gestures captured via webcam, with variations in hand pose, lighting, and background.

Preprocessing Pipeline :

1. Hand Detection : Cvzone’s HandDetector (based on MediaPipe) localized hands in real time using bounding boxes.
2. Cropping and Padding : Hands were cropped using the bounding box coordinates, padded with a 20-pixel offset to avoid truncation.
3. Aspect Ratio Maintenance : Resized crops were padded with white space to ensure a fixed resolution of 300×300 while preserving aspect ratios.
4. Normalization : Backgrounds were standardized to white to reduce noise.

Augmentation :

Explicit augmentation (rotation, flipping, and brightness adjustment) were applied during training. This project leverages the preprocessed data.

3.2 Model Development

Machine Learning/Deep Learning Algorithms :

Three architectures were trained and evaluated:

1. Convolutional Neural Network (CNN) : Optimized for spatial feature extraction and real-time inference.
2. Recurrent Neural Network (RNN) : Explored temporal dynamics via hand keypoints.
3. Transformer : Tested for attention-based feature weighting.

3.3.1 Model Architectures

Deep Learning Models Trained :

1. Convolutional Neural Network (CNN) :
    - Input : $300 \times 300 \times 3$ RGB images with PCA/ICA features.
    - Architecture :
        - 4 convolutional blocks with 3×3 kernels, ReLU activation, and batch normalization.
        - 2×2 max-pooling after each block.
        - Global average pooling followed by a dense layer with softmax activation (7 classes).
    - Rationale : Optimized for spatial feature extraction and real-time inference.
2. Recurrent Neural Network (RNN) :
    - Input : Temporal sequences of hand keypoints (21 keypoints × 30 frames).
    - Architecture :
        - LSTM layers to capture gesture dynamics.
        - Fully connected layers for classification.
    - Limitations: Underperformed due to hardware constraints.
3. Transformer :
    - Input : PCA-reduced feature vectors as tokens.
    - Architecture :
        - Self-attention layers to model spatial relationships.
        - Classification head for gesture recognition.
    - Limitations : Required more data to generalize effectively.

3.4 Training and Evaluation

The Training process involved:

- Framework : TensorFlow 2.18.0 and Keras 3.8.0.
- Hyperparameters :
    - Loss Function: Categorical cross-entropy.
    - Optimizer: Adam (learning rate: 1e-4, decay: 1e-6).
    - Batch Size: 32.
    - Epochs: 50 (early stopping if validation loss plateaued).
- Data Split : 70% training, 15% validation, 15% testing.

3.3 Real-Time Processing

Frameworks and Technologies Used for Real-Time Processing

Real-time processing in sign language detection requires a well-structured technology stack to ensure minimal latency, high accuracy and seamless interaction between the system components. The following technologies are employed:

Frontend Technologies

- **Dart**: Dart is used as the primary programming language for the frontend due to its efficiency in building high-performance mobile applications. It supports just-in-time (JIT) compilation, which speeds up development through hot-reloading and ahead-of-time (AOT) compilation, ensuring optimized execution for real-time processing. Dart's strong type system also helps in maintaining application stability.
- **Flutter**: Chosen for its cross-platform capabilities, Flutter allows for the development of a single codebase that runs smoothly on both Android and iOS devices. This significantly reduces development time and ensures a consistent user interface (UI). Flutter's reactive UI framework provides smooth animations, essential for displaying real-time transcriptions. Additionally, its built-in gesture detection features support seamless user interaction, improving accessibility for deaf and non-deaf users alike.

Backend Technologies

- **Python**: Python is used in the backend because of its powerful ecosystem for machine learning and data processing. It offers extensive libraries for deep learning, making it the ideal choice for training and deploying sign language detection models. Python's simplicity and readability accelerate development and debugging, ensuring a robust backend infrastructure.
- **Requests**: This library is used to establish communication between the frontend and the backend. It ensures smooth data transmission, enabling the mobile application to send captured sign language gestures to the backend for real-time processing and receive accurate transcriptions with minimal delay.
- **protobuf (Protocol Buffers)**: Used for efficient data serialization, protobuf allows structured data to be transmitted between the backend and frontend in a compressed and optimized format. This significantly reduces the amount of data transferred, improving processing speed and minimizing latency in real-time applications.
- **Werkzeug**: This utility library is essential for handling HTTP requests and server-side processing. It enhances the performance of the Flask backend by managing routing, debugging, and request handling efficiently, ensuring that API responses are processed in real-time without bottlenecks.

Optimization Techniques for Efficiency

To ensure the real-time processing of sign language detection and transcription, various optimization techniques are applied:

1. **Model Quantization**: Reduces model size and computation by converting floating-point weights into lower precision formats, such as INT8 or FP16. This significantly improves inference speed on edge devices while maintaining accuracy.
2. **Edge Processing**: Offloading computations to edge devices (such as smartphones or embedded systems) minimizes the reliance on cloud servers, reducing network latency and improving response times.
3. **Parallel Processing & Multi-threading**: The system leverages multi-threading and parallel processing to simultaneously capture video frames, process model inference, and render UI elements, ensuring minimal latency and seamless performance.

4. **Adaptive Frame Rate Control**: Dynamically adjusts the processing frame rate based on system load, ensuring smooth video feed and optimizing resource utilization without compromising accuracy.
5. **Efficient Data Handling**: Utilizes optimized data structures and memory management techniques, reducing processing overhead and ensuring smooth real-time execution.
6. **Streaming Buffering**: Implements intelligent buffering techniques to prevent lag and ensure continuous video feed processing, maintaining an uninterrupted user experience.
7. **Hardware Acceleration**: The system takes advantage of hardware acceleration using GPU-based processing for deep learning models, enhancing inference speed and efficiency.
8. **Low-Latency Data Transfer**: Implements fast serialization and deserialization of data using protobuf, ensuring efficient communication between system components.

3.4 System Architecture

High-Level System Design

The system architecture comprises several integrated modules that work together to achieve real-time sign language detection and transcription. The key components include:

1. **Camera Input Module**
    - Captures video streams in real time using OpenCV.
    - Adjusts resolution, brightness and contrast dynamically for optimal gesture detection.
    - Ensures efficient frame capture with minimal latency.
2. **Preprocessing Module**
    - Utilizes OpenCV and NumPy for image normalization, noise reduction and frame resizing.
    - Applies contrast enhancement and adaptive thresholding to improve model accuracy.
    - Uses SciPy for additional signal processing techniques when required.
3. **Model Inference Engine**
    - Runs a deep learning model trained on sign language gestures using TensorFlow.

- Utilizes MediaPipe for hand tracking and pose estimation to extract key gesture features.
- Optimized for real-time execution with reduced latency and high inference accuracy.

4. **Post-Processing Module**
   - Transforms raw model outputs into meaningful transcriptions.
   - Implements filtering techniques to refine detected words based on context.
   - Uses NumPy for efficient data handling and processing of model outputs.
5. **User Interface (UI) Integration**
   - Displays real-time transcriptions and implements interactive elements allowing user feedback and correction.
   - Uses Matplotlib for visual debugging and representation of detected gestures.
6. **API Layer**
   - Facilitates seamless communication between the frontend and backend.
   - Utilizes Requests for handling API calls between system components.
   - Ensures efficient data exchange while maintaining real-time performance.
7. **Storage Module**
   - Stores processed sign language data locally or in the cloud for future analysis.
   - Uses Protobuf for efficient serialization and data structuring.
   - Handles retrieval and storage of past transcriptions for learning improvements.

4. Implementation

4.1 Tools and Technologies

To build a fast, reliable, and scalable mobile app for facilitating communication between deaf and non-deaf users, we'll leverage a combination of cross-platform frameworks, machine learning tools, and lightweight libraries.

- **TensorFlow**: A powerful deep learning framework used for model training and inference. It supports optimized computation for neural networks, allowing efficient model deployment on both edge and cloud devices.
- **OpenCV**: An essential computer vision library that handles video capture, frame extraction and preprocessing. It enhances images using techniques such as noise reduction, contrast enhancement, and edge detection, which improve gesture recognition accuracy.
- **MediaPipe**: Provides pre-trained models and pipelines for hand tracking, pose estimation and gesture recognition, allowing real-time detection of sign language movements with minimal computational overhead.
- **NumPy & SciPy**: NumPy provides efficient data handling and array computations, while SciPy offers advanced image processing functions crucial for enhancing gesture recognition accuracy.
- **Matplotlib**: Used for data visualization and debugging, particularly in tracking model performance and analyzing real-time detection accuracy.
- **Requests**: Enables seamless API interactions and data exchange between different system components, facilitating efficient real-time processing.
- **Pillow**: Used for image manipulation, including resizing and format conversion, to optimize frames before feeding them into the model.

4.2 Software Development Workflow

To ensure a streamlined process, maintain code quality, and enable future scaling, we'll adopt a structured workflow with version control, CI/CD, and a deployment strategy tailored to a mobile app.

- **Version Control:**
  - **Git**: The backbone of our version control, hosted on **GitHub**. We'll use a simple branching strategy:
    - `main`: Stable, production-ready code.
    - `dev`: Integration branch for new features (e.g., gesture recognition, chat).
    - Feature branches (e.g., `feature/gesture-model`, `feature/chat-ui`): Short-lived branches for specific tasks, merged into `dev` via pull requests (PRs).

- **CI/CD Integration:**
  - **GitHub Actions**: Our CI/CD tool of choice. It's built into GitHub, free for small projects, and highly customizable. The workflow will include:
    - **Continuous Integration (CI)**: On every PR to `dev` or `main`, GitHub Actions will:
      - Run Flutter unit tests (e.g., for chat logic or emoji translation).
      - Lint the Dart code for consistency using `flutter analyze`.
      - Build the app to catch errors early.
    - **Continuous Deployment (CD)**: For deployment-ready builds:
      - Automatically generate APK (Android) and IPA (iOS) files on merges to `main`.
      - Upload artifacts to GitHub Releases for easy access by users.
- **Deployment Strategy:**
  - **Edge Deployment (On-Device)**: Since the app operates offline or on LAN for now, all ML models (gesture recognition, speech-to-text) and features run locally on the mobile device. TensorFlow Lite and Google ML Kit ensure efficient edge performance.
    - **Model Optimization**: The sign language ML model will be trained in Python, converted to TensorFlow Lite format, and optimized (e.g., quantization) to minimize size and latency on phones.
    - **Local Storage**: SQLite handles emoji/sticker data, and app settings (e.g., voice vs. text preference) are stored in shared preferences.

    - **LAN/Pass-Phone Mode**: For LAN, we'll use Flutter's built-in Dart networking (e.g., `dart:io`) to enable device-to-device chat over a local network. For pass-phone, the app runs as a single-instance chat, passing between users without networking.
    - **Scalability Note**: This edge-first approach allows easy scaling later—e.g., adding a cloud backend (Firebase, AWS) for networked chat—without rewriting the core app.

5. Testing and Evaluation

5.1 Performance Metrics

To ensure the effectiveness of our AI-powered real-time sign language interpreter, we will measure:

- **Accuracy:** The percentage of correctly classified signs compared to the actual gestures.
- **Latency:** The time taken to process and translate a sign into text/speech (measured in milliseconds).
- **Efficiency:** Resource consumption (CPU/GPU usage, memory footprint) during real-time processing.
- **Robustness:** The system's ability to handle variations in lighting, camera angles, and background noise.

5.2 User Testing

To validate usability and real-world performance, we will conduct:

- **Pilot Testing with Deaf Users & Sign Language Experts**
    - Collaborate with **Kenyan Sign Language (KSL) speakers** to test accuracy in natural settings.
    - Evaluate **ease of use, responsiveness, and clarity** of the interpreted text/speech output.
    - Collect feedback on errors, misinterpretations, and user experience.
- **Scenario-Based Testing**
    - Users will test the system in **different environments** (e.g., classrooms, hospitals, public offices).
    - Test under **varying conditions** (e.g., low lighting, background movement, different skin tones).
- **Iterative Improvements**

    - Gather user feedback via surveys and interviews.
    - Fine-tune the model based on real-world performance gaps.

5.3 Model Evaluation

We will use standard machine learning evaluation techniques to assess model performance:

- **Confusion Matrix:** To analyze classification errors by comparing predicted vs. actual labels.
- **F1-Score:** Measures model balance between precision and recall, ensuring it performs well across all sign classes.
- **Precision & Recall:**
    - **Precision:** How many predicted signs were correct?
    - **Recall:** How many actual signs were correctly identified?
- **Cross-Validation:** Splitting the dataset into multiple training and testing sets to prevent overfitting.
- **Real-Time Testing:** Measuring model accuracy and response time when deployed on a mobile device.

6. Conclusion and Future Work

Key Takeaways

SignSync is an innovative, AI-powered solution that bridges the communication gap between the Deaf and hearing communities.

By combining computer vision and NLP, it provides real-time, accurate translation of sign language into text and speech.

The project prioritizes inclusivity, accessibility, and scalability, making it a practical tool for everyday use.

Future Work

Expand to More Sign Languages:

Add support for British Sign Language (BSL), Indian Sign Language (ISL), and others.

Collaborate with native signers to collect diverse datasets.

Enhance Real-Time Accuracy:

Improve gesture recognition for complex and dynamic gestures.

Optimize models for faster inference on edge devices.

Incorporate Facial Expressions and Body Language:

Add support for non-manual markers (e.g., facial expressions, head movements) to improve translation accuracy.

Integrate offline capabilities for areas with limited internet access.

Community-Driven Improvements:

Open-source the project to encourage contributions from developers and the Deaf community.

Continuously refine the system based on user feedback.

Integration with Assistive Devices:

Partner with smart glasses or AR/VR devices for hands-free translation.

Explore integration with hearing aids or other assistive technologies.

7. References

1. Hammadi, S. (2023). *Kenyan Sign Language Classification 91% Accuracy* . Kaggle. https://www.kaggle.com/code/salimhammadi07/kenyan-sign-language-classification-91-accuracy .
2. OpenCV Team. (2023). *OpenCV: Open Source Computer Vision Library* . https://opencv.org .
3. Thrun, T., et al. (2023). *MediaPipe Hands: High-Fidelity Hand Tracking* . https://mediapipe.dev .
4. Rathod, V. (2023). *Cvzone: Computer Vision Library* . https://github.com/cvzone/cvzone .
5. Chollet, F., et al. (2023). *Keras: Deep Learning Library for Python* . https://keras.io
6. Abadi, M., et al. (2023). *TensorFlow: Open-Source Machine Learning Framework* . https://www.tensorflow.org .