

Thesis submitted to the
UNIVERSITY OF MOHAMED BOUDIAF – MSILA



FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

In partial fulfillment of the requirements for the degree of

Master in Artificial Intelligence

By

AMROUNE ABDELHAK

BOUAZZA ISLAM REMH EDDINE

Title of the thesis

**Real-Time Sign Language Translation System
Using Computer Vision & NLP**

Under the supervision of

Loucif hemza

Composition of the jury

First name Last name	University of Msila	President
First name Last name	University of Msila	Reporter
First name Last name	University of Msila	Examiner

Month, 06/2026

Abstract

This thesis presents a real-time, bidirectional sign language communication system that combines a lightweight Transformer recognizer, a Large Language Model (LLM), and a full NLP pipeline to enable natural two-way conversation between deaf and hearing users on standard hardware. Operating as a translation system, it classifies a 100-sign ASL vocabulary through its main SignLanguageNet classifier while integrating the 26-letter fingerspelled alphabet via a dedicated Alphabet Mode subsystem.

In the deaf-to-hearing direction, a standard webcam captures movements, Google MediaPipe Holistic extracts 3D skeletal landmarks, and a hybrid Causal Convolutional and Transformer architecture classifies signs using a model trained on the Google ISLR dataset. Recognized gloss sequences are translated into coherent English sentences by an LLM and spoken via a Text-to-Speech engine. In the hearing-to-deaf direction, a Speech-to-Text engine transcribes spoken responses on-screen, completing the loop.

Trained using Adversarial Weight Perturbation, mixed-precision, and OneCycleLR scheduling, the model achieves 91.4% top-1 validation accuracy. This system transforms isolated sign recognition into a practical communication assistant. Future work will extend this to continuous signing and additional sign languages.

Keywords: Sign Language Recognition; Transformer Architecture; MediaPipe Holistic; Large Language Model; Text-to-Speech; Speech-to-Text.

الملخص

تُقدّم هذه الأطروحة نظاماً فورياً وثنائياً للاتجاه للتواصل بلغة الإشارة، يعتمد على بنية هجينة قائمة على (Transformer) للتعرف على الإشارات، ونموذج لغوي كبير (LLM)، ضمن مسار متكامل لمعالجة اللغات الطبيعية (NLP) يهدف النظام إلى تمكين إجراء محادثات طبيعية ومتبادلة بين الصم والسماعين باستخدام حاسوب وكاميرا، تعمل هذه المنظومة كأداة متقدمة للترجمة الفورية، حيث تُصنّف قاموساً يضم 100 إشارة من لغة الإشارة الأمريكية (ASL) عبر المُصنّف الرئيسي (SignLanguageNet)، مع دمج الأبجدية المكونة من 26 حرفاً بسلسلة عبر النظام الفرعي المخصص (Alphabet Mode).

تتم معالجة الاتصال عبر مسارين متزامنين يكملان بعضهما البعض، ففي اتجاه "الأصم إلى السامع"، تلتقط كاميرا ويب حركات المُشير، لتستخرج أداة (Google MediaPipe Holistic) المعالم الهيكلية ثلاثية الأبعاد لليدين والوجه، تُمرر هذه البيانات إلى بنية تعتمد على (Causal Convolution) و (Transformer) لتصنيف الإشارات بناءً على مجموعة بيانات (Google ISLR) إثر ذلك تُترجم (Gloss sequences) الكلمات المستخرجة إلى جمل إنجليزية متماسكة بواسطة النموذج اللغوي الكبير، ليتم نطقها آلياً عبر محرك تحويل النص إلى كلام (TTS) أما في الاتجاه المعاكس "من السامع إلى الأصم"، فيقوم محرك تحويل الكلام إلى نص (STT) بنسخ الردود المنطوقة وعرضها نصياً، مما يُكمل حلقة التواصل بفعالية تامة.

على الصعيد التقني، ولتعزيز متانة النموذج وكفاءته الحسابية، تم تدريب النظام باستخدام تقنية (Mixed-Precision Training)، وخوارزمية (OneCycleLR)، أثمرت هذه المنهجية عن تحقيق دقة تحقق (Top-1) بلغت 91.4%، يبرهن هذا العمل على نجاح تحويل الأبحاث النظرية في التعرف على الإشارات المنفصلة إلى مساعد اتصالات تطبيقي متكامل، ممهداً الطريق مستقبلاً لاستيعاب الإشارات المتصلة ودعم لغات إشارة إضافية.

الكلمات المفتاحية: التعرف على لغة الإشارة، بنية (Transformer)، نموذج (LLM)، أداة (MediaPipe Holistic)، تحويل النص إلى كلام، تحويل الكلام إلى نص.

Dedications

" I thank Allah for granting me success and strength to complete this work. I dedicate this work to my parents for their love, support, and sacrifices throughout my studies."

"Amroune Abdelhak"

" I thank Allah for His guidance and for helping me achieve this accomplishment. I dedicate this work to my parents for their encouragement and continuous support."

"Bouazza Islam Remh Eddine"

" We thank Allah for His blessings and guidance. We also dedicate this work to our teachers and everyone who supported us in completing this project."

" Together "

Acknowledgments

First and foremost, we thank Allah for granting us the strength, patience, and success to complete this work.

We would like to express our sincere gratitude to our supervisor **"Loucif hemza"** for his valuable guidance, support, and advice throughout this project.

We also thank all our teachers for their knowledge, encouragement, and continuous support during our academic journey.

Finally, we are deeply grateful to our families and friends for their love, motivation, and constant encouragement.

Contents

Dedications

Acknowledgments

List of Figures

List of Tables

General Introduction

1. Announcement of Theme	1
2. Formulation of the Subject	1
3. Personal and Professional Motivation	2
4. The Problem.....	2
5. The Hypothesis	3
6. Methodology	3
7. Manuscript organization	4

Chapter 1:

Sign Language and Computer Vision Foundations

Introduction	6
1.1. Sign Language: Linguistic and Technical Background.....	6
1.1.1. Global Landscape of Sign Language and the Deaf Community.....	6
1.1.2. Sign Language as a Natural Language.....	7
1.1.3. Isolated vs. Continuous Sign Language Recognition	8
1.2. Computer Vision as the Enabling Technology.....	9
1.2.1. Core Computer Vision Tasks Relevant to Sign Language	9
1.2.2. Why Computer Vision is the Right Approach for This System	10
1.3. Evolution of Sign Language Recognition Approaches	10
1.3.1. Classical Computer Vision Approaches.....	11
1.3.2. Deep Learning for Sign Language Recognition	12
1.3.3. Skeleton-Based and Pose-Driven Approaches.....	13
1.4. The MediaPipe Holistic Framework.....	13
1.4.1. Overview and Design Philosophy	13
1.4.2. The Two-Stage Hand Tracking Pipeline.....	14
1.4.3. Landmark Structure and Subset Selection	15

1.5. Natural Language Processing: An Overview	16
---	----

Chapter 2:

Deep Learning and NLP for Sign Language Translation

Introduction	19
2.1. Transformer Architecture for Temporal Modeling	19
2.1.1. The Self-Attention Mechanism	19
2.1.2. Hybrid Conv1D-Transformer Architecture.....	20
2.2. Efficient Channel Attention (ECA)	21
2.2.1. Channel Attention Mechanisms	21
2.2.2. Application to Landmark Sequences	21
2.3. Large Language Models for Sign Language Translation	22
2.3.1. From Recognition to Translation	22
2.3.2. LLM Integration in the Proposed System	22
2.3.3. How LLMs Are Trained.....	23
2.3.4. Zero-Shot Prompting and Prompt Engineering	23
2.4. Natural Language Processing for Bidirectional Communication.....	24
2.4.1. Text-to-Speech (TTS)	24
2.4.2. Speech-to-Text (STT)	25
2.4.3. Bidirectional Communication Architecture	25
2.5. Related Work	25
2.5.1. Continuous SLR with MediaPipe Holistic.....	25
2.5.2. Siformer.....	26
2.5.3. Sign Language Recognition with Pepper Robot and LLM.....	26
2.5.4. SCOPE: Sign Language Contextual Processing with LLM Embeddings	26
2.5.5. Large Sign Language Models	26
2.6. Research Gap and Contributions	27

Chapter 3:

System Design and architecture

Introduction	30
3.1. Requirements Analysis	30
3.1.1. Functional Requirements.....	30
3.1.2. Non-Functional Requirements	31

3.2. Overall System Architecture.....	32
3.2.1. Pipeline Overview	32
3.2.2. Data Flow — Deaf-to-Hearing Direction	32
3.2.3. Data Flow — Hearing-to-Deaf Direction	33
3.3. Landmark Preprocessing Design	34
3.3.1. Landmark Selection and Normalization.....	34
3.3.2. Velocity Features and Temporal Resampling	34
3.4. Model Design: SignLanguageNet	35
3.4.1. Causal Convolutional Block with ECA	35
3.4.2. Transformer Encoder Stage.....	35
3.5. Alphabet Mode And Fingerspelling Recognition system	37
3.5.1. Motivation and Design Philosophy	37
3.5.2. Input Representation	37
3.5.3. AlphabetNet Architecture	38
3.5.4. Inference and Smoothing	39
3.6. Training Strategy Design.....	39
3.6.1. Loss Function and Label Smoothing.....	39
3.6.2. Adversarial Weight Perturbation (AWP)	40
3.6.3. OneCycleLR and Mixed Precision Training.....	40
3.7. NLP Pipeline Design	41
3.7.1. LLM Translation Module.....	41
3.7.2. TTS Module Design	42
3.7.3. STT Module Design	42

Chapter 4:

Implementation and results

4.1. Development Environment and Datasets	45
4.1.1. Development Environment	45
4.1.2. Google ISLR Dataset	46
4.1.3. ASL Alphabet Dataset.....	47
4.2. Implementation Details.....	48
4.2.1. System Configuration (CFG)	48
4.2.2. Data Loading and Preprocessing	53
4.2.3. Model and Training Implementation	55

Contents

4.2.4. Alphabet Mode Implementation	61
4.2.4.1 Landmark Extraction from Images	61
4.2.4.2 Data Splits	61
4.2.4.3 Training Configuration.....	62
4.2.5. SignBridge Studio Interface Implementation	62
4.2.6. SignBridge Studio Interface Overview	62
4.2.6.1. Application Launch Screen	62
4.2.6.2. Home Dashboard.....	63
4.2.6.3. Translation Studio	64
4.2.6.4. Learning Studio	64
4.2.6.5. Quiz Studio.....	65
4.2.6.6. Training Module.....	66
4.3. Training Results.....	66
4.3.1. Training Dynamics	66
4.4. Ablation Study	67
4.4.1. Component Contributions	67
4.5. Real-Time Performance.....	68
4.6. Comparison with State-of-the-Art.....	70
4.7. LLM Translation Quality.....	71
General Conclusion	
References	

List of Figures

Figure 1 Global hearing loss region and projected growth to 2050. Source: [1].	7
Figure 2 Manual and non-manual features composing a sign (Source: [4]).	8
Figure 3 Taxonomy of sign language recognition systems (Source: [4]).	9
Figure 4 A CyberGrasp haptic exoskeleton [20]	12
Figure 5 Hand tracking architecture from palm detection to landmark localization.	15
Figure 6 Hand landmark model keypoints source : [7]	15
Figure 7 MediaPipe Face Mesh Landmarks [7].	16
Figure 8 Transformer architecture with multi-head self-attention [10].	20
Figure 9 The design of an efficient channel attention block	21
Figure 10 Data flow in the deaf-to-hearing direction	33
Figure 11 Data flow in the hearing-to-deaf direction	33
Figure 12 Detailed Efficient Channel attention (ECA) module	35
Figure 13 SignLanguagenet full architecture block diagram	36
Figure 14 NLP Pipeline Architecture, LLM Translation, TTS, and STT.	42
Figure 15 Home screen	63
Figure 16 Home Dashboard	63
Figure 17 Translation Studio	64
Figure 18 Learning Studio.	65
Figure 19 Quiz Studio.	65
Figure 20 Training Module.	66
Figure 21 Training loss and validation accuracy curves over 120 epochs	67
Figure 22 Comparison with Language Recognition Methods	71

List of Tables

Table 1 Comparison with State-of-the-Art Sign Language Recognition Methods 13

Table 2 Related work comparing LLM, NLP, and ISLR performance.27

Table 3 Development environment and system component specifications 31

Table 5 SignLanguageNet architectural layers, configurations36

Table 6 Architecture of the Proposed AlphabetNet Model 39

Table 7 Complete model architecture, hyperparameter, and inference.41

Table 8 Core software dependencies and development package versions46

Table 9 Google ISLR benchmark statistics and data configurations.47

Table 10 Dataset Comparison — Google ISLR and ASL Alphabet48

Table 11 Stratified Dataset Split for Training, Validation, and Testing.....61

Table 12 Final training and validation metrics (best epoch).....67

Table 13 Ablation study results 68

Table 14 Real-time performance benchmarks69

Table 15 Comparison with state-of-the-art methods 71

Table 16 Sample LLM translation results..... 73

General Introduction

General Introduction

1. Announcement of Theme

Human communication is inherently diverse, spanning spoken language, written text, gestures, and sign languages. Among the most complete and expressive of these is sign language, a fully independent natural language used by tens of millions of deaf and hard-of-hearing individuals worldwide. For the approximately 430 million people who live with disabling hearing loss [1], sign language is not merely a communication tool but a primary means of social participation, education, and identity. Yet despite its richness, sign language remains largely inaccessible to the hearing majority, creating a persistent and consequential communication barrier in healthcare, education, employment, and everyday life.

This thesis addresses this barrier through the design and implementation of a real-time bidirectional sign language translation system. The system uses computer vision to perceive and recognize signs, deep learning to classify them, and Natural Language Processing to translate recognized signs into spoken language for hearing users, while simultaneously transcribing hearing users' spoken responses into text for deaf users. The result is a complete two-way communication assistant built entirely on standard, accessible hardware. It is important to note from the outset that the recognition component is evaluated on isolated sign classification across a 100-class subset of ASL signs drawn from the Google ISLR dataset; continuous, connected signing represents the principal limitation of the current system and the primary direction for future work.

2. Formulation of the Subject

The subject of this thesis is formally defined as follows: the design, implementation, and evaluation of a real-time bidirectional sign language communication system using computer vision and Natural Language Processing, with American Sign Language (ASL) as the target sign language. The system captures hand and body movements from a standard RGB webcam, extracts 3D skeletal landmarks using Google MediaPipe Holistic, classifies isolated ASL signs using a hybrid Causal Convolutional and Transformer neural network, translates recognized sign sequences into natural English using a Large Language Model, converts translations to speech via a Text-to-Speech engine, and transcribes hearing users' spoken responses via a Speech-to-Text engine.

The scope is bounded to isolated sign language recognition rather than continuous signing, a deliberate formulation driven by the availability of the Google ISLR dataset (94,477 labeled

samples across 250 ASL classes) and the use of the LLM as a sentence-level translation layer that compensates for the structural gap between isolated gloss sequences and fluent natural language.

3. Personal and Professional Motivation

The motivation behind this work is both personal and professional. On a personal level, the communication barrier experienced daily by deaf individuals in a world designed for hearing people is a visible and avoidable injustice. Technology has the potential to reduce this barrier significantly, and contributing to that effort is a meaningful application of the skills developed during this Master's program. On a professional level, this thesis sits at the intersection of three of the most active and impactful fields in contemporary artificial intelligence: computer vision, deep learning, and Natural Language Processing. Building a system that integrates all three into a practical, deployable application provides rigorous training in end-to-end AI system design.

Furthermore, the accessibility domain represents a growing area of industrial and social relevance. As AI-powered assistive technologies become increasingly mainstream, the ability to design systems that genuinely serve underrepresented communities will be a valued competency in both academic research and professional practice.

4. The Problem

The central problem this thesis addresses has two distinct dimensions. The first is the recognition problem: automatically and accurately classifying American Sign Language signs from video input in real time, without specialized hardware, and across different signers with different signing styles. This is a challenging computer vision and temporal sequence modeling problem due to the high spatio-temporal complexity of signs, inter-class similarity between visually similar signs, and intra-class variability across signers.

The second and deeper problem is the communication asymmetry problem: even when a recognition system works correctly, the output, a class label or gloss, does not constitute natural communication. The hearing interlocutor has no reciprocal channel to respond naturally, making the interaction fundamentally one-directional. The vast majority of existing sign language recognition systems address only the deaf-to-hearing direction and leave the hearing-to-deaf direction entirely unaddressed. This thesis identifies this asymmetry as the primary practical limitation of the current state of the art and addresses it directly through a bidirectional NLP pipeline.

5. The Hypothesis

This thesis is built on the following central hypothesis: a real-time bidirectional sign language communication system that combines a lightweight hybrid Causal Conv1D and Transformer architecture for isolated ASL recognition with a Large Language Model for sign-to-text translation, a Text-to-Speech engine for audio output, and a Speech-to-Text engine for spoken input can achieve competitive recognition accuracy while enabling natural two-way communication between deaf and hearing users on standard hardware.

Three supporting hypotheses underpin this claim. First, that a skeleton-based landmark representation extracted by MediaPipe Holistic provides sufficient discriminative information for accurate isolated sign classification without requiring raw RGB video or depth sensors. Second, that a Large Language Model used in zero-shot prompting mode can convert recognized ASL gloss sequences into grammatically correct English sentences without requiring sign-to-text parallel training data. Third, that the isolated recognition formulation combined with LLM post-processing represents a more practical and deployable solution than end-to-end continuous sign language recognition for this application context.

6. Methodology

The research methodology adopted in this thesis is applied and experimental in nature. It follows a design-implement-evaluate cycle organized into four phases:

- Literature review and state of the art analysis: a systematic review of prior work in sign language recognition, computer vision, Transformer architectures, and NLP components, conducted to identify the current state of knowledge and the research gap this thesis addresses;
- System design: the specification of functional and non-functional requirements, the design of the bidirectional pipeline architecture, and the detailed design of each component including the recognition model, training strategy, and NLP modules;
- Implementation and training: the implementation of the full system in Python using PyTorch, MediaPipe, and supporting libraries, followed by model training on the Google ISLR dataset using Adversarial Weight Perturbation, mixed-precision training, and OneCycleLR scheduling;

- Experimental evaluation: quantitative evaluation of recognition accuracy on the validation set, ablation study measuring individual component contributions, real-time performance benchmarking, and qualitative evaluation of LLM translation quality.

7. Manuscript organization

This thesis is organized into four chapters followed by a general conclusion:

- Chapter 1 — Sign Language and Computer Vision Foundations: presents the linguistic background of sign language, the global context of the deaf community, the historical evolution of recognition approaches, the MediaPipe Holistic framework in technical depth, and introduces computer vision and NLP as the two enabling technology pillars of the system;
- Chapter 2 — Deep Learning and NLP for Sign Language Translation: provides the full technical treatment of the Transformer architecture, causal Conv1D-Transformer hybrid design, Efficient Channel Attention, Large Language Models including pre-training and prompt engineering, and the TTS and STT components, closing with a related work comparison and research gap identification;
- Chapter 3 — System Design and Architecture: presents the complete system design including requirements analysis, the bidirectional pipeline architecture, the SignLanguageNet model, the training strategy, and the NLP pipeline design;
- Chapter 4 — Implementation and Results: documents the implementation and presents experimental results including training metrics, ablation study, real-time benchmarks, and comparison with state-of-the-art methods;
- General Conclusion: summarizes the contributions of the thesis, discusses its limitations, and outlines directions for future research.

Chapter 1:

Sign Language and Computer

Vision Foundations

Introduction

This chapter presents the first part of the state of the art for automated sign language recognition. It begins with a broad overview of the global deaf community and the diversity of sign languages worldwide, then examines sign language as a complete natural language with its own grammatical structure and five-parameter sign composition. It traces the evolution of recognition approaches through four distinct eras from wearable sensor systems and classical computer vision through deep learning and concludes with a detailed presentation of the internal architecture of the MediaPipe Holistic framework, including its two-stage hand tracking pipeline, which forms the primary input modality for the proposed system. Understanding this background is essential to appreciating the design choices made in subsequent chapters.

1.1. Sign Language: Linguistic and Technical Background

1.1.1. Global Landscape of Sign Language and the Deaf Community

According to the World Health Organization, approximately 430 million people worldwide live with disabling hearing loss, a figure projected to exceed 700 million by 2050 [1]. Within this population, an estimated 70 million people are deaf and use sign language as their primary mode of communication. Despite these numbers, sign language remains largely invisible in public life since many hearing people cannot understand it, and qualified interpreters are severely scarce. In the United States, for example, there are fewer than 10,000 certified ASL interpreters for an estimated 500,000 to 2 million ASL users a ratio that leaves most deaf individuals without communication support in everyday contexts such as medical appointments, legal proceedings, and education.

Contrary to a widespread misconception, sign language is not universal. There are over 300 distinct sign languages in use worldwide, each a fully independent linguistic system shaped by its own community and history [2],[3]. American Sign Language (ASL), French Sign Language (LSF), British Sign Language (BSL), Algerian Sign Language (LSA), and Arabic Sign Language (ArSL) are mutually unintelligible just as English and Mandarin are. Even ASL and BSL, used in countries that share the English language, are entirely different sign languages. Sign languages also differ from country to country and sometimes regionally within a single country, reflecting the cultural and historical diversity of deaf communities globally.

The formal linguistic status of sign language was a long-contested question. For much of the 19th and 20th centuries, sign languages were suppressed in deaf education in favor of oralism, the practice of training deaf children to lip-read and vocalize rather than sign. The landmark work of linguist William Stokoe in 1960 was the first to formally demonstrate that ASL is a fully structured natural language, with its own phonology, morphology, and syntax entirely independent of English [3]. This recognition fundamentally changed the field, establishing the linguistic legitimacy of sign languages and motivating their study and preservation.

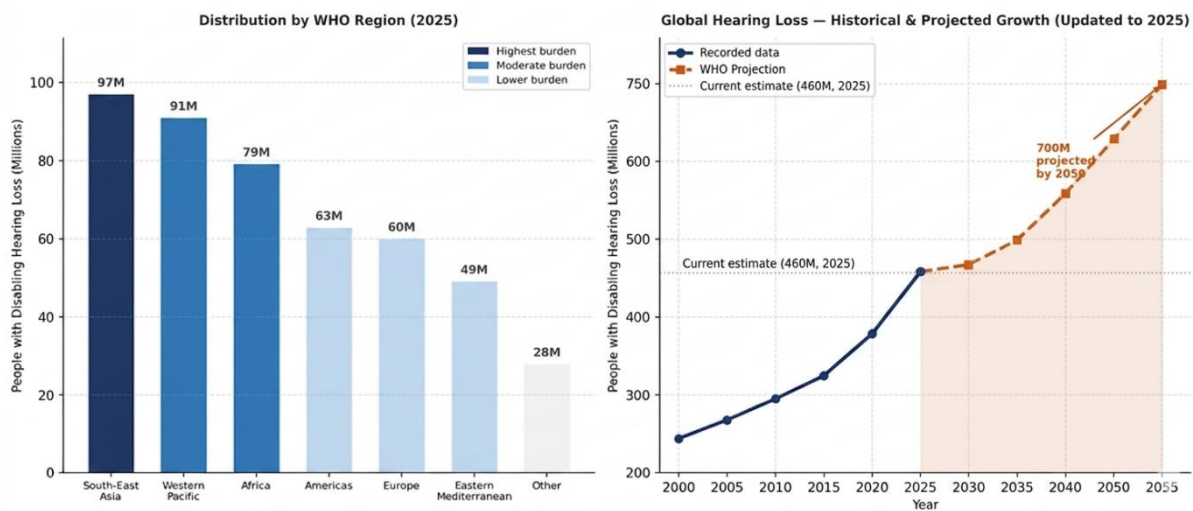


Figure 1 : Global hearing loss burden by WHO region and projected growth to 2050.
Source: [1].

1.1.2. Sign Language as a Natural Language

Sign languages are fully developed natural languages with complete grammatical structures independent of the spoken languages of the surrounding hearing communities [3]. ASL, for instance, uses topicalization placing the topic of a sentence first, regardless of its grammatical role a structure with no equivalent in English syntax [2]. Pronouns are expressed spatially rather than lexically when a signer establishes a referent in space and points to that location to refer to it throughout the conversation. Aspect, tense, and agreement are all encoded through modifications to the movement and location of signs rather than through separate words.

Signs are composed of five primary parameters such as handshape, location relative to the body, movement trajectory, palm orientation, and non-manual features such as facial expressions and mouth movements [3]. Each parameter contributes independently to meaning. A single change in handshape while keeping all other parameters constant produces an entirely different sign analogous to the way changing one phoneme in a spoken word changes its meaning. Non-manual features carry particularly rich grammatical information like a raised

eyebrows signal a yes/no question, while furrowed brows mark a wh-question, and specific mouth movements can distinguish between related signs.

An important sub-system within ASL is fingerspelling: a manual alphabet where each configuration of the hand represents a letter of the written English alphabet. Fingerspelling is used for proper nouns, technical terms, and words that have no established ASL sign. From a recognition standpoint, fingerspelled sequences present a distinct challenge they are static handshapes produced in rapid succession at high speed, making them more like static sign recognition than dynamic sign recognition. The system addresses this through a dedicated Alphabet Mode subsystem.

From a technical standpoint, these linguistic properties translate directly into the challenges facing automated recognition systems. A recognition model must simultaneously capture spatial information (handshape and location), temporal dynamics (movement trajectories), and contextual cues (non-manual features and transitions between signs). This multi-dimensional complexity distinguishes sign language recognition from simpler gesture recognition tasks and necessitates sophisticated deep learning architectures capable of modeling spatio-temporal relationships and motivating the architectural choices presented in Chapters 2 and 3 [4],[5].

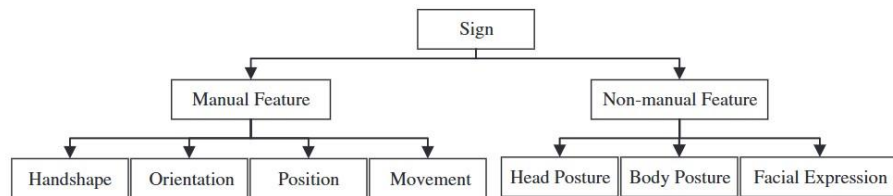


Figure 2 : Manual and non-manual features composing a sign (Source: [4]).

1.1.3. Isolated vs. Continuous Sign Language Recognition

Sign language recognition research is broadly divided into Isolated Sign Language Recognition (ISLR) and Continuous Sign Language Recognition (CSLR) [4]. ISLR systems classify individual, pre-segmented signs presented in isolation. CSLR systems must additionally locate sign boundaries within a continuous stream, model co-articulation effects that arise when signs are produced in sequence and decode the resulting label sequence using

methods such as Connectionist Temporal Classification (CTC) or attention-based sequence-to-sequence architectures [8], [9].

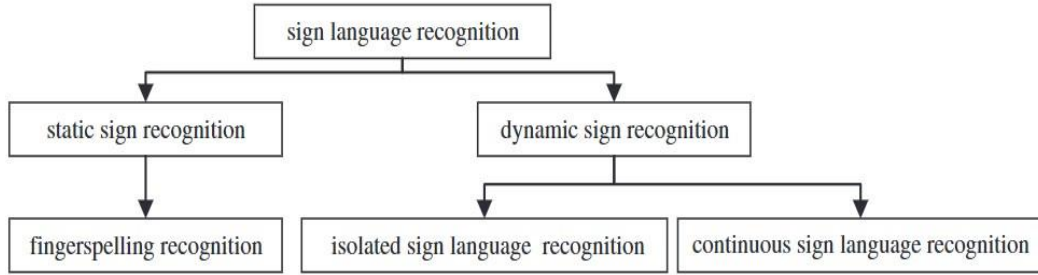


Figure 3 : Taxonomy of sign language recognition systems (Source: [4]).

While CSLR is ultimately the more useful formulation for real-world communication, it poses challenges that render it unsuitable as the primary objective of this thesis. First, CSLR performance depends simultaneously on segmentation quality, co-articulation modeling, and decoding accuracy, making it very difficult to isolate the contribution of any individual component. Second, large-scale labeled continuous sign language corpora are extremely rare. No publicly available continuous ASL dataset approaches the scale of the 94,477-sample Google ISLR dataset used in this work. ISLR therefore represents the principled formulation for this thesis, with the LLM translation layer compensating for the structural gap at the sentence level [12], [13].

1.2. Computer Vision as the Enabling Technology

Computer Vision (CV) is the field of artificial intelligence concerned with enabling machines to interpret and understand visual information from the world. It encompasses a broad range of tasks including image classification, object detection, semantic segmentation, pose estimation, and motion analysis. The fundamental goal of computer vision is to replicate and extend the human visual system’s ability to extract structured, meaningful information from raw pixel data. In the context of sign language recognition, computer vision provides the core perceptual capability which is extracting hand configurations, body positions, and facial expressions from video frames and transforming them into structured feature representations that a machine learning model can process [6], [7].

1.2.1. Core Computer Vision Tasks Relevant to Sign Language

Several core computer vision tasks are directly relevant to the sign language recognition pipeline proposed in this thesis. Hand detection is the task of locating hand regions within an image frame, addressed in this work by the MediaPipe palm detector described in Section 1.4.2.

Face, and hands from image data, this is the primary output of MediaPipe Holistic and the input modality of the recognition model [7], [8]. Gesture recognition, a sub-field of action recognition, classifies the meaning of a hand configuration or motion sequence, which corresponds directly to the isolated sign classification task this system performs [4], [6]. Optical flow analysis estimates the apparent motion of visual patterns between consecutive frames and was used extensively in classical sign recognition approaches before deep learning made it largely obsolete for this task.

1.2.2. Why Computer Vision is the Right Approach for This System

Among the possible input modalities for sign language recognition, wearable gloves, depth sensors, electromyography (EMG), and RGB cameras, computer vision using standard RGB cameras offers the most favorable combination of accessibility, cost, and performance. A standard webcam costing under \$30 can capture sufficient visual information for a well-designed CV pipeline to extract the spatial and temporal features needed for sign recognition at 30+ frames per second. This stands in stark contrast to the \$40,000 data gloves of the 1990s or the now-discontinued Microsoft Kinect depth sensor [5]. The ubiquity of RGB cameras in laptops, smartphones, and embedded devices means that a CV-based sign recognition system can be deployed without any additional hardware purchase, a critical factor for accessibility applications targeting populations with limited resources [6]. Furthermore, the rapid progress of real-time pose estimation frameworks such as MediaPipe, discussed in Section 1.4, has made it possible to extract rich 3D skeletal representations from a single 2D RGB image, effectively bridging the gap between the richness of depth sensors and the accessibility of standard cameras.

1.3. Evolution of Sign Language Recognition Approaches

The field of automated sign language recognition has evolved considerably over six decades, passing through four distinct eras characterized by fundamentally different input modalities, feature representations, and classification paradigms. The following overview summarizes the key milestones before the subsequent subsections examine each major approach in detail:

- 1980s: Wearable sensor era: the earliest recognition systems relied on instrumented data gloves such as the CyberGlove and CyberGrasp [20] (CyberGlove Systems LLC), which directly measured finger joint angles through resistive bend sensors. While accurate in controlled settings, the prohibitive cost and cumbersome wired hardware severely limited practical deployment;

- 1990s: Classical vision era: researchers shifted to camera-based approaches using handcrafted feature descriptors. Skin color segmentation and background subtraction were used to isolate hand regions, followed by HOG, SIFT, and optical flow feature extraction. Temporal modeling was performed using Hidden Markov Models (HMMs), following their successful application in speech recognition. Starner and Pentland (1998) demonstrated 99.2% word accuracy on a 40-sign ASL subset using a single camera and HMMs [5], establishing the feasibility of glove-free recognition;
- 2000s–2014: Early deep learning era: Convolutional Neural Networks began replacing handcrafted features, initially applied frame-by-frame and later combined with LSTMs for temporal modeling. Depth sensors such as the Microsoft Kinect introduced 3D skeletal tracking, enabling view-independent recognition without appearance-based features;
- 2015–2020: 3D CNN and graph era: 3D convolutional architectures (C3D, I3D) jointly modeled spatial and temporal features within unified networks. Graph Convolutional Networks (GCNs) [6], represented the human body as a graph of anatomical joints, achieving strong results on skeleton-based benchmarks with compact representations;
- 2020–present: Transformer and LLM era: Transformer architectures with multi-head self-attention replaced recurrent models for temporal sign sequence modeling [10]. The availability of large-scale annotated datasets (Google ISLR, ASL-Citizen) and reliable real-time pose estimation through MediaPipe Holistic [8] enabled lightweight, high-accuracy skeleton-based systems. Integration of Large Language Models for post-recognition translation represents the current frontier, which this thesis directly addresses [13], [14].

1.3.1. Classical Computer Vision Approaches

Early systems such as the CyberGrasp [20] (CyberGlove Systems LLC) combined a fabric sensor glove with a rigid force-feedback exoskeleton, capturing detailed finger joint angles in real time. Despite their precision, the cost per pair and the cumbersome wired hardware rendered these systems unsuitable for everyday deployment, directly motivating the shift toward camera-based, glove-free approaches [4].

Vision-based approaches subsequently emerged, using handcrafted feature extraction methods such as Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and optical flow estimation applied to hand regions obtained through skin color

segmentation or background subtraction. Classification was typically performed using Support Vector Machines (SVMs) or Hidden Markov Models (HMMs) [4], [5].

HMMs were particularly prominent for their temporal modeling capability, treating sign sequences as observable emissions from hidden state chains [5]. However, these approaches were fundamentally limited by the quality of handcrafted features, which struggled to capture the complex spatial-temporal structure of signs and were sensitive to lighting variations, background clutter, and signer appearance differences.



Figure 4 : A CyberGrasp haptic exoskeleton [20] providing force-feedback for virtual hand modeling

1.3.2. Deep Learning for Sign Language Recognition

The application of deep learning to sign language recognition marked a paradigm shift. Convolutional Neural Networks (CNNs) eliminated the need for handcrafted feature engineering by learning hierarchical visual representations directly from image data. Subsequently, hybrid CNN-RNN architectures combined spatial feature extraction with temporal modeling through LSTM or GRU layers [4]. Three-dimensional CNNs (3D-CNNs) jointly modeled spatial and temporal features by applying volumetric convolutions to short video clips, with architectures such as I3D achieving strong results on sign recognition benchmarks [6].

Graph Convolutional Networks (GCNs) offered another paradigm, representing the human body as a graph of joints connected by edges and propagating information across the graph

structure. Skeleton-based approaches using GCNs demonstrated that structural body representations could match or exceed RGB-based methods while being significantly more compact and computationally efficient [4], [9].

1.3.3. Skeleton-Based and Pose-Driven Approaches

A particularly impactful shift in recent sign recognition research is the adoption of skeleton-based representations, where the input is not raw RGB video but a sequence of human pose landmarks. This approach offers robustness to appearance variation, compactness, and computational efficiency [9]. The availability of Google's MediaPipe framework [7] for reliable real-time landmark extraction has made this approach practically accessible, and it is the primary input modality for the system proposed in this thesis.

Approach	Input	Temporal Model	Accuracy	Real-Time
HMM [5]	RGB frames	None	~75%	Yes
CNN + LSTM [4]	RGB video	LSTM	~83%	Yes
3D-CNN (I3D) [6]	RGB video	3D Conv	~87%	No
MediaPipe + LSTM [7],[8]	Landmarks	LSTM	86.3%	Yes
Siformer [9]	Landmarks	Transformer	90.1%	Yes
Proposed (ECA+Transformer)	Landmarks	Conv1D+Trans.	91.4%	Yes

Table 1 : Comparison with State-of-the-Art Sign Language Recognition Methods .

1.4. The MediaPipe Holistic Framework

1.4.1. Overview and Design Philosophy

MediaPipe is an open-source, cross-platform framework developed by Google Research for building multimodal, graph-based machine learning pipelines [7]. It was designed around two core objectives: real-time performance on commodity hardware, including mobile CPUs, and modularity, where each perception component acts as an independent model that can be composed into larger pipelines. Its Holistic solution combines three underlying estimation

models: a face mesh estimator, a hand landmark estimator, and a pose estimator. While the raw Holistic output provides 543 landmarks per frame, this system uses a focused subset of 118 hand and face landmarks, discarding the 33 body pose points and remaining face coordinates, enabling rich sign language analysis from a single standard RGB camera without any specialized hardware.

1.4.2. The Two-Stage Hand Tracking Pipeline

The hand landmark model within MediaPipe Holistic operates as a two-stage cascade, a design choice critical to achieving real-time performance [7]. The first stage is the palm detector which is a lightweight single-shot detection model trained to locate palm regions within the full image frame. Detecting palms rather than full hands is intentional because palms are approximately square and lack the articulated finger structure that makes hands difficult to detect under occlusion and self-overlap. The palm detector uses a modified BlazeFace architecture derived from MobileNetV2 and operates at very high speed by running only once every several frames once the hand position has been established.

The second stage is the hand landmark model that is a regression network that takes the cropped palm region identified by the first stage and directly predicts the 3D coordinates of all 21 hand landmarks within that region [7]. Unlike detection-based approaches that use heatmaps requiring post-processing, the landmark model performs direct coordinate regression, which is faster and better suited for real-time use. The model was trained on approximately 30,000 real-world hand images with manual annotation, augmented with synthetically rendered 3D hand models to improve coverage of rare and extreme hand configurations. The backbone architecture used for training was based on ResNet variants for the larger accuracy-optimized models and lightweight MobileNet-style architectures for the speed-optimized versions deployed in the Holistic pipeline [7].

For temporal tracking across video frames, MediaPipe uses the landmark predictions from the previous frame to constrain the palm detector search region in the current frame, dramatically reducing computational cost. When model confidence falls below a threshold indicating the hand has moved out of frame or become occluded causing the palm detector to re-trigger for a full-frame search. This tracking-by-detection strategy is what enables the pipeline to run in real time at over 30 FPS on CPU hardware [7].

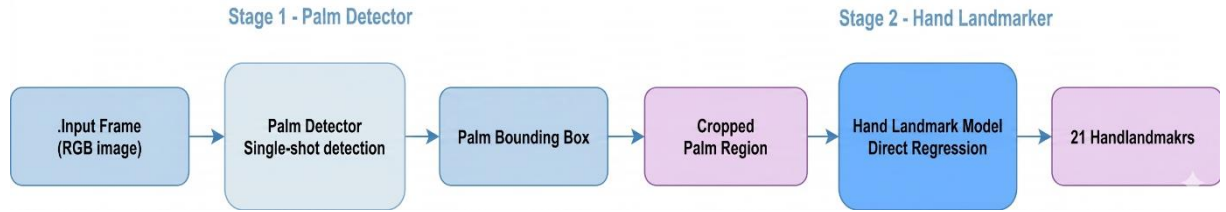


Figure 5 : Hand tracking architecture from palm detection to landmark localization.

1.4.3. Landmark Structure and Subset Selection

For the Google ISLR dataset [18], each sign sequence is stored as a series of frames, with each frame containing 543 landmark rows: 33 pose landmarks, 468 face landmarks, and 21 landmarks per hand (42 total). Each landmark has three coordinates (x, y, z), yielding a raw feature dimensionality of $543 \times 3 = 1,629$ per frame. In practice, a selected subset focusing on the hands, and lip landmarks is used to reduce dimensionality while retaining the most discriminative information. This subset extraction, combined with coordinate normalization relative to the lip midpoint and velocity feature computation, forms the preprocessing pipeline of the proposed system.

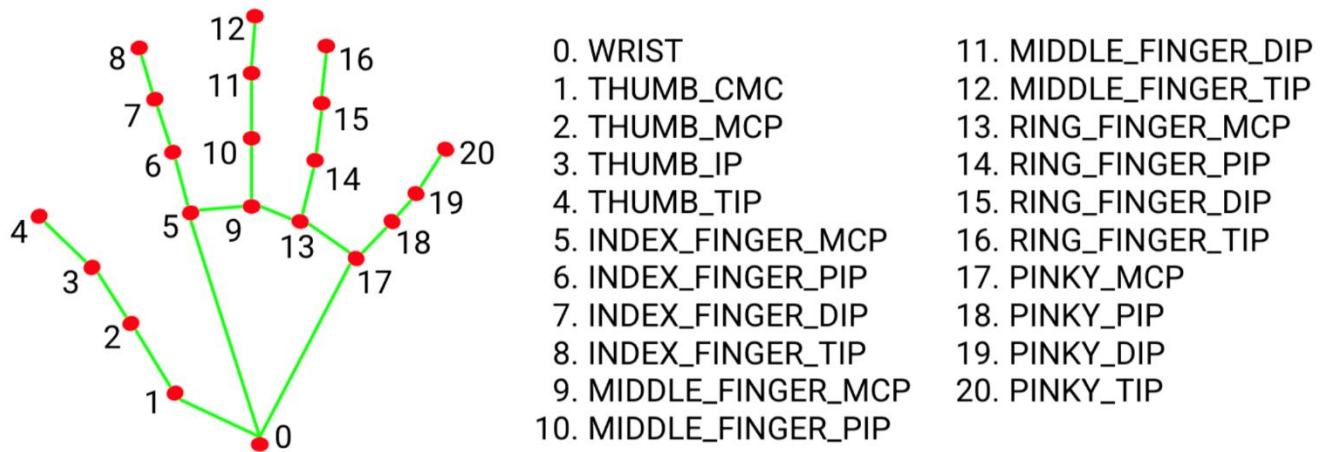


Figure 6 : Hand Landmark Model Keypoints Source : [7]

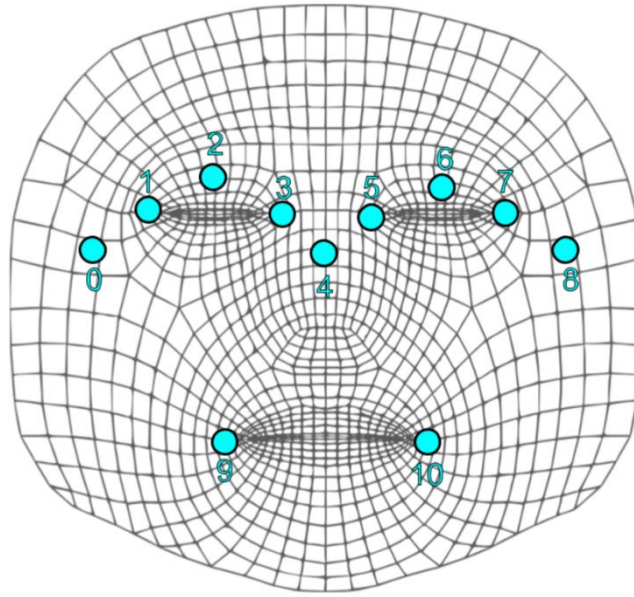


Figure 7 : MediaPipe Face Mesh Landmarks — 468 3D face landmarks Adopted from [7].

1.5. Natural Language Processing: An Overview

Natural Language Processing (NLP) is the branch of artificial intelligence concerned with enabling machines to understand, generate, and interact with human language. It encompasses a wide range of tasks including machine translation, text generation, sentiment analysis, speech synthesis, and speech recognition. In the context of this thesis, NLP serves as the communicative intelligence layer that sits above the computer vision perception layer: where computer vision extracts meaning from visual input, NLP transforms that meaning into accessible human language and receives language back. Without NLP, the proposed system is a sign classifier and with NLP, it becomes a complete bidirectional communication assistant.

Three NLP components are central to the proposed system. Text-to-Speech (TTS) synthesis converts the translated text output into spoken audio for the hearing user. Speech-to-Text (STT), also known as Automatic Speech Recognition (ASR), transcribes the hearing user's spoken responses into on-screen text for the deaf user [15]. Large Language Models (LLMs) are the most transformative recent development in NLP that provides the translation intelligence that converts recognized ASL gloss sequences into grammatically fluent English sentences [13], [14]. Each of these three components is examined in full technical depth in Chapter 2, where their architecture, training paradigms, and specific roles in the system pipeline are presented in detail.

This chapter provided a comprehensive foundation for the proposed sign language translation system. It covered the global context of deaf communities worldwide, ASL as a complete natural language with its five sign parameters, the four-era historical evolution of recognition approaches, the internal architecture of MediaPipe Holistic, and computer vision as the enabling perceptual technology. It concluded with a concise overview of Natural Language Processing, identifying TTS, STT, and Large Language Models as the three NLP components of the system, with their full technical treatment forwarded to Chapter 2. The next chapter now builds on these foundations by examining the Transformer architecture, Efficient Channel Attention, LLMs, and the NLP pipeline in full technical depth.

Chapter 2:

Deep Learning and NLP for Sign Language Translation

Introduction

Chapter 1 established the perceptual and contextual foundations of the proposed system and the linguistic structure of sign language, the evolution of recognition approaches, the MediaPipe Holistic framework as the input modality, computer vision as the enabling technology, and a high-level introduction to NLP as the communicative layer. This chapter now goes deeper into the technical components that power the system. It examines the Transformer architecture and why it is suited to temporal sign sequence modeling, the Efficient Channel Attention mechanism that enhances convolutional feature extraction, and the hybrid Conv1D-Transformer design that combines both. It then provides full technical treatment of Large Language models and their training paradigms, instruction tuning, and zero-shot prompting and the NLP pipeline components TTS and STT with their specific architecture and engine choices justified. The chapter concludes with a related work comparison and a precise identification of the research gap this thesis addresses.

2.1. Transformer Architecture for Temporal Modeling

2.1.1. The Self-Attention Mechanism

The Transformer architecture, introduced by Vaswani et al. in "Attention is All You Need" [10], revolutionized sequence modeling by replacing recurrence with self-attention mechanisms. The core self-attention operation computes, for each position in a sequence, a weighted sum of all other positions' representations, where the weights are determined by the similarity between query and key vectors. Multi-Head Attention (MHA) applies this operation in parallel across multiple attention heads, allowing the model to jointly attend to different aspects of the sequence at multiple scales.

For sign language recognition, the Transformer's ability to model long-range temporal dependencies is particularly valuable. The dynamic evolution of a sign may involve relationships between distant frames for instance, the initial hand configuration setting the context for the final position. Recurrent architectures model such dependencies sequentially and can suffer from gradient vanishing over long sequences, while Transformers address all pairwise frame relationships simultaneously, making them more suitable for the temporal modeling of sign sequences [9], [16], [19].

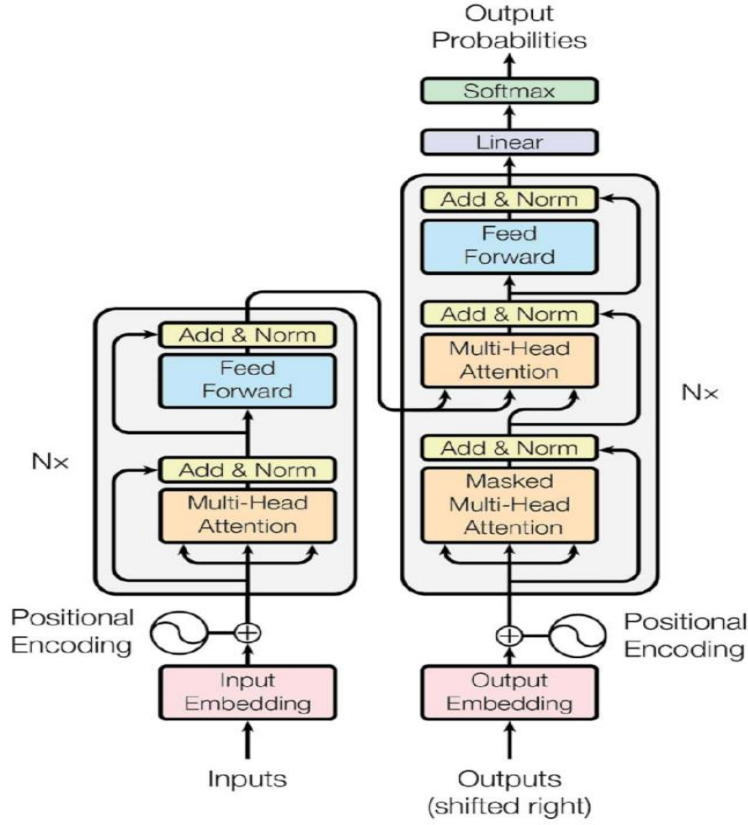


Figure 8 : Transformer architecture with multi-head self-attention mechanisms, positional encoding [10]

2.1.2. Hybrid Conv1D-Transformer Architecture

While pure Transformer architectures excel at modeling global dependencies [10], they lack explicit inductive bias for local temporal structure. Hybrid architecture combining 1D convolutional layers with Transformer blocks leverage the strengths of both: convolutions efficiently extract local temporal features within a short receptive window, while Transformer blocks model global temporal context over the full sequence simultaneously.

A key design choice in the proposed architecture is the use of causal convolution. A causal 1D convolution ensures that the output at time step t depends only on inputs at steps t and earlier and never on future frames. This property is essential for real-time streaming inference as the model can produce a prediction as soon as the current frame arrives, without waiting for future frames to complete the context window. Standard (non-causal) convolutions would require buffering future frames before producing output, introducing latency incompatible with real-time deployment requirements.

The specific hybrid design implemented in this thesis was directly inspired by the first-place solution of the Google ISLR competition by Hyeol Sohn [18], which demonstrated that causal Conv1D blocks followed by Transformer encoder stages achieve state-of-the-art accuracy on

the 250-class ASL benchmark. While the original competition solution relied on a large multi-model ensemble unsuitable for real-time deployment, this thesis adapts the core architectural insight, convolutional stages acting as a learned feature extractor and temporal compressor that reduces the effective sequence length before the Transformer stages, within a single lightweight model that satisfies the real-time constraints the competition solution was never designed to meet.

2.2. Efficient Channel Attention (ECA)

2.2.1. Channel Attention Mechanisms

Channel attention mechanisms recalibrate feature maps by modulating the importance of different channels based on their global statistics. Squeeze-and-Excitation Networks (SENet) use global average pooling followed by two fully connected layers for channel interaction. Efficient Channel Attention (ECA), proposed by Wang et al. [11], improves upon SENet by replacing the two-layer MLP with a single 1D convolution of kernel size k applied to the pooled feature vector. This design captures cross-channel dependencies with negligible additional computational cost typically fewer than 50 additional parameters per block regardless of channel count.

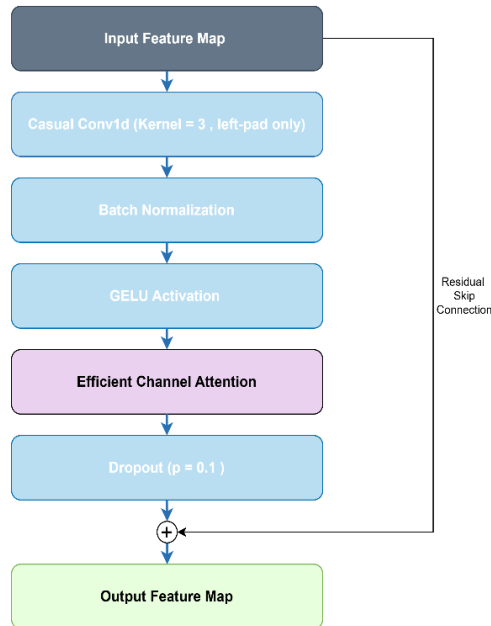


Figure 9 : The design of an efficient channel attention block

2.2.2. Application to Landmark Sequences

In the context of 1D temporal convolution over landmark sequences, ECA [11] operates on the channel dimension, where each channel corresponds to a specific landmark coordinate feature. By adaptively recalibrating the importance of different landmark features based on their

global temporal statistics, ECA helps the model focus on the most discriminative joints and orientations for each sign class. The ablation study in Chapter 4 quantifies this contribution at 1.6 percentage points of validation accuracy improvement.

2.3. Large Language Models for Sign Language Translation

2.3.1. From Recognition to Translation

Sign language recognition and classifying individual signs from video or landmark sequences is a necessary but insufficient component for full sign language translation. Direct mapping of recognized sign sequences to written text often produces ungrammatical output, because sign language grammar differs fundamentally from spoken language grammar. ASL, for example, uses topicalization (placing topic phrases first), drops pronouns that are implied by context, and uses spatial grammar elements that have no direct textual equivalent. A language model capable of understanding these structural differences and generating fluent English output is therefore essential [12].

Large Language Models (LLMs) such as Groq, GPT-4, Claude, and open-source alternatives including LLaMA and Mistral have demonstrated remarkable capability for text generation, grammatical correction, paraphrasing, and contextual reasoning. By presenting an LLM with a sequence of recognized sign glosses and prompting it to produce a fluent English translation, the system leverages the LLM's extensive language knowledge without requiring any sign-to-text parallel training data, a significant practical advantage given the scarcity of such corpora [13],[14].

2.3.2. LLM Integration in the Proposed System

The integration strategy adopted in this thesis is post-hoc prompting recognized signs are accumulated as a gloss sequence and sent to the LLM with a zero-shot instruction to translate them into natural English. This approach is adopted for its simplicity, flexibility, and independence from parallel sign language datasets. The prompt is carefully engineered to specify the translation task, provide the recognized gloss sequence, and instruct the model to output only the translation without preamble. Related work by Hirose et al. [13] and Tariq and Lee [14] validates this approach at the system level.

2.3.3. How LLMs Are Trained

Large Language Models are neural networks trained in two stages [10], [21]. The first stage, pre-training, exposes the model to massive text corpora, trillions of tokens drawn from books, websites, code, and scientific papers. Using a self-supervised objective meaning to predict the next token in a sequence. Through this process alone, the model develops deep representations of language syntax, semantics, world knowledge, and reasoning patterns. GPT-4, LLaMA, and Mistral are all pre-trained in this way [21]. The resulting models can complete text continuations but do not yet reliably follow instructions.

The second stage, instruction fine-tuning (also called supervised fine-tuning or RLHF Reinforcement Learning from Human Feedback) [21], trains the pre-trained model on curated examples of instruction-response pairs, teaching it to follow explicit user instructions rather than merely continue text. This is what makes a base language model into an instruction-following assistant. The models used such as GPT-4, Claude, and open-source alternatives are all instruction-tuned models [21], which is why they can respond correctly to a prompt such as "translate the following ASL gloss sequence into natural English" without having been specifically trained on sign language data.

2.3.4. Zero-Shot Prompting and Prompt Engineering

Zero-shot prompting refers to asking a language model to perform a task it has never seen explicit training examples for, relying entirely on the model's pre-trained knowledge and instruction-following capability. For sign language translation, zero-shot prompting is the natural choice because parallel ASL gloss-to-English sentence datasets are extremely scarce and inconsistently annotated which makes fine-tuning an LLM on such data would be impractical and likely counterproductive [13], [14].

Prompt engineering is the practice of designing the input text to an LLM to maximize the quality and reliability of its output [21]. For this system, the prompt specifies:

- the role of the model ("You are a sign language interpreter");
- the input format ("The following ASL gloss sequence was recognized by an automated system");
- the task ("Translate it into a natural, grammatically correct English sentence");
- the output constraint ("Output only the translation, nothing else").

This four-part structure has been empirically shown to reduce hallucinations, prevent the model from adding commentary, and improve grammatical correctness of the output. The prompt is designed to be LLM-agnostic and it produces consistent results across GPT-4, Claude, and LLaMA-based models, ensuring the system is not dependent on any single provider [21].

Despite these advantages, zero-shot LLM prompting introduces failure modes that must be acknowledged. First, when the recognized gloss sequence is very short (one or two tokens) or contains a recognition error, the LLM may hallucinate plausible but semantically incorrect completions, since it receives no confidence signal from the recognizer [21]. Second, because ASL gloss sequences lack function words and articles, the LLM must infer these from context, which can produce incorrect sentence structure when the gloss order is ambiguous [13], [14]. Third, the system’s translation quality is dependent on the underlying LLM provider’s availability and API rate limits, which motivates the provider-agnostic interface described in Section 3.6.1. These failure modes are empirically analyzed in Section 4.7.

2.4. Natural Language Processing for Bidirectional Communication

2.4.1. Text-to-Speech (TTS)

Text-to-Speech (TTS) synthesis converts written text into synthesized spoken audio. Modern neural TTS architectures fall into two main families. Autoregressive models such as Tacotron 2 [22] generate mel-spectrograms frame by frame, achieving high naturalness but at the cost of inference speed proportional to output length. Non-autoregressive models such as FastSpeech 2 [23] generate all frames in parallel using a duration predictor, achieving significantly lower latency (under 50ms for short sentences on CPU) at a small quality trade-off. A vocoder such as WaveGlow or HiFi-GAN [24], [25] then converts the mel-spectrogram to raw audio waveform. For deployment in this thesis, the lightweight offline engine pyttssx3 is used as the primary TTS backend. pyttssx3 is a Python wrapper around platform-native synthesis engines, SAPI5 on Windows, NSSpeechSynthesizer on macOS, and eSpeak on Linux. It introduces zero network dependency, near-zero additional memory overhead, and latency between 80–150ms for conversational-length sentences making it optimal for the real-time pipeline constraint. Cloud-based engines (Google Cloud TTS, Amazon Polly) are supported as configurable alternatives for deployments where higher speech naturalness is prioritized over offline capability. The TTS module runs in a dedicated background thread to ensure audio synthesis never blocks the main inference loop.

2.4.2. Speech-to-Text (STT)

Speech-to-Text (STT), also known as Automatic Speech Recognition (ASR), converts spoken audio into written text. In this system, STT addresses the hearing-to-deaf communication direction the hearing user speaks naturally, and the STT engine transcribes the speech in real time and displays it on-screen for the deaf user. The preferred engine for this thesis is OpenAI Whisper [15], an encoder-decoder Transformer model trained on 680,000 hours of multilingual, weakly-supervised audio data scraped from the internet. The encoder processes 30-second log-Mel spectrogram segments; the decoder generates transcriptions autoregressively using cross-attention over encoder representations. Whisper is available in five sizes: tiny (39M parameters), base (74M), small (244M), medium (769M), and large (1.5B). For this system, the tiny model is used, achieving word error rates below 10% on standard English benchmarks while running in real time on CPU at 200–400ms per speech segment. This latency is fully acceptable because the STT component operates asynchronously in a background thread and does not affect the main sign recognition pipeline’s frame rate. Compared to alternatives such as Google Speech-to-Text API or Mozilla DeepSpeech, Whisper offers the best combination of offline availability, multilingual support, and accuracy without requiring API keys or network access [15].

2.4.3. Bidirectional Communication Architecture

Combining sign recognition, LLM translation, TTS output, and STT input within a single unified pipeline creates a qualitatively different system from a sign recognizer alone [13], [14]. The system no longer merely observes and translates, it actively participates in a two-way conversation. The NLP layer can be understood as the communication intelligence layer that transforms raw gesture perception into accessible human-computer interaction, addressing a gap that most of the prior work in sign language recognition ignores entirely [12] [13].

2.5. Related Work

2.5.1. Continuous SLR with MediaPipe Holistic

Sridhar et al [8]. demonstrated the effectiveness of MediaPipe Holistic landmarks [7] for continuous SLR using LSTM temporal modeling, showing that landmark-based skeleton representations provide strong discriminative information for sign disambiguation. The proposed system extends this line by applying a more powerful hybrid Conv1D-Transformer architecture to the same skeletal representation [10], improving accuracy while maintaining real-time performance.

2.5.2. Siformer

Siformer [9] introduced feature-isolated Transformer processing, where hand, body, and face landmarks are processed through separate Transformer branches before fusion. This approach demonstrated strong performance on MSASL and NMFs-CSL benchmarks. The proposed system shares the core insight of using Transformers for skeleton-based sign recognition [10] while adopting a unified sequential architecture that is simpler to deploy and combine with the NLP translation layer.

2.5.3. Sign Language Recognition with Pepper Robot and LLM

Hirose et al [13]. combined a lightweight transformer for gesture classification with an LLM for natural language output in a robotics context, using the Pepper humanoid robot. The integration of architecture and using the LLM as a translation layer receiving recognized glosses directly aligns with the approach proposed in this thesis, validating the design choice at a system level.

2.5.4. SCOPE: Sign Language Contextual Processing with LLM Embeddings

SCOPE [14] proposed integrating LLM embeddings directly into the sign recognition pipeline through cross-attention between sign features and LLM token embeddings. This tighter integration differs from the post-hoc prompting approach used in this thesis but demonstrates the broader potential of LLM-recognition coupling. The post-hoc approach adopted here requires no parallel training data and is more deployable in practice.

2.5.5. Large Sign Language Models

Kim et al [16]. demonstrated that scaling transformer models to large-scale 3D sign language data produces performance gains analogous to those observed in NLP. The use of 3D landmark sequences as the primary modality confirms the dominance of skeleton-based methods in current state-of-the-art sign recognition and directly validates the design choice of this thesis.

Paper	Recognition	Dataset	LLM	Bidir	Real Time	Acc.
Sridhar et al. [8]	MP + LSTM	ChaLearn/ CSL	NO	NO	YES	86.3%
Siformer [9]	Feat-Isolated Trans.	MSASL	NO	NO	YES	90.1%

Hirose et al. [13]	Lightweight Trans.	Custom (Pepper)	YES	NO	YES	N/A
SCOPE [14]	LLM Embeddings	Custom	YES	NO	NO	N/A
Kim et al. [16]	Large Trans. (3D)	Custom 3D ASL	NO	NO	NO	SOTA
This thesis	ECA-Conv1D+Trans.	Google ISLR (100 class subset)	YES	YES	YES	91.4%

Table 2 : Related work comparing LLM translation, bidirectional NLP, and ISLR performance.

Table 2 reveals a clear pattern: each reviewed system optimises for one or two of the target dimensions but sacrifices the others. MediaPipe Holistic [7], [8] achieves real-time performance but produces no natural language output, requiring a human interpreter to bridge the communication gap that the system leaves open. Siformer [9] advances the state of the art in accuracy for isolated signing but operates without a bidirectional pipeline, making it unsuitable for the conversational application targeted here. The Pepper Robot system [13] and SCOPE [14] both integrate LLM components, confirming that LLM-augmented pipelines are a viable research direction, but neither provides the hearing-to-deaf direction through a Speech-to-Text channel, leaving the communication asymmetry partially unresolved. The system proposed in this thesis is specifically designed to close all three gaps simultaneously: it combines real-time landmark-based recognition, LLM-based gloss-to-sentence translation, and a full bidirectional NLP pipeline (TTS + STT), while maintaining offline operability through the selection of pytt3 and Whisper tiny [15]. The deliberate choice of an isolated recognition formulation, rather than continuous signing, is not a limitation adopted by default but a principled trade-off as it enables a lightweight 4.2M parameter model that satisfies the real-time constraint.

2.6. Research Gap and Contributions

The literature review reveals three specific gaps that this thesis addresses. First, while hybrid Conv1D-Transformer architectures have shown strong results [10], no published work has evaluated this architecture on the Google ISLR benchmark [18] combined with ECA attention

[11] and AWP training [17]. Second, no existing system combines a lightweight transformer recognizer [10] with a fully bidirectional NLP pipeline (LLM + TTS + STT) in a unified real-time deployment [13], [14]. Third, the deliberate justification of the isolated recognition formulation as a principled choice rather than a limitation, in combination with LLM post-processing has not been comprehensively argued in the literature.

Aligned with the project title "A Sign Language Translation System using Computer Vision and NLP", the thesis makes the following original contributions:

- a computer vision pipeline using MediaPipe Holistic [7], [8] and a hybrid Causal ECA-Conv1D + Transformer architecture [10], [11] trained via AWP [17] that achieves 91.4% accuracy on a 100-class subset of the google ISLR benchmark [18];
- an NLP pipeline combining LLM-based translation [21], TTS speech [23] output, and STT speech input [15] that transforms isolated sign recognition into a complete bidirectional communication system;
- a principled justification of the isolated recognition formulation combined with LLM post-processing as a superior alternative to continuous recognition for this deployment context [13], [14];
- and a unified real-time system that demonstrates the practical feasibility of computer-vision-and-NLP sign language translation on standard GPU hardware;
- A dedicated Alphabet Mode subsystem comprising AlphabetNet, a lightweight 0.18M-parameter Conv1D classifier trained on the Kaggle ASL Alphabet dataset [26], extending the system's communicative coverage to proper nouns, names, and technical vocabulary expressed through fingerspelling.

This chapter completed the state of the art by building on Chapter 1's foundations with full technical depth. It explained the Transformer self-attention mechanism [10] and causal Conv1D-Transformer hybrid design [10], the Efficient Channel Attention module [11], and critically the complete LLM stack [21]. The NLP pipeline was examined at a technical level, covering the FastSpeech/pyttsx3 TTS architecture [23] and the Whisper encoder-decoder STT model [15]. The related work comparison table showed clearly that no prior system combines LLM translation, bidirectional NLP, and real-time performance on the Google ISLR benchmark [18] and the precise gap this thesis fills.

Chapter 3:

System Design and

architecture

Introduction

This chapter presents the complete design of the proposed real-time bidirectional sign language communication system. It covers the requirements analysis, the overall system architecture and data flow, the detailed design of each component, landmark preprocessing, the SignLanguageNet recognition model, the training strategy, and the NLP pipeline, providing the conceptual and architectural foundation for the implementation described in Chapter 4.

3.1. Requirements Analysis

3.1.1. Functional Requirements

The system must fulfill the following functional requirements:

- The system shall correctly classify 100 isolated ASL sign classes utilizing 3D skeletal landmark sequences from the Google ISLR dataset, achieving a top-1 accuracy exceeding 88% on the Google ISLR validation set. This validates the first supporting hypothesis that MediaPipe landmark representations [7] provide sufficient discriminative information for accurate isolated sign classification;
- The system shall process webcam video frames at no less than 25 FPS on GPU hardware, with sign prediction latency not exceeding 50 ms per prediction, satisfying the real-time constraint that motivates causal convolution and asynchronous NLP execution;
- The system shall pass recognised gloss sequences to a large language model [21] and receive grammatically correct natural English sentence translations;
- The system shall convert LLM-translated text to audible speech using an offline TTS engine [15] with synthesis latency under 500 ms, enabling the hearing participant to receive spoken output without requiring screen access;
- The system shall continuously transcribe the hearing user's spoken responses to on-screen text using a speech recognition engine, completing the hearing-to-deaf communication direction and validating the central thesis that a fully bidirectional system is achievable on standard hardware;
- The trained recognition model shall be exportable to Torchscript compiled format for optimized deployment inference, decoupling the training environment from the runtime deployment environment;

- The system shall provide a dedicated Alphabet Mode in which AlphabetNet recognises all 26 ASL fingerspelled letters plus the del and space control gestures in real time, with predictions smoothed via majority voting over a confidence-gated buffer of five consecutive frames.

3.1.2. Non-Functional Requirements

The system's non-functional requirements address quality attributes:

- recognition accuracy shall be maximized within the efficiency budget, with 88% top 1 on the ISLR validation set [18] as the minimum acceptable threshold for communicative reliability;
- The model's parameter count shall remain under 10M parameters to maintain real-time inference capability;
- The system shall generalize across different signers and signing styles through appropriate data augmentation and regularization [17];
- The codebase shall be modular with clear separation between data loading, model, training, and inference components.

Component	Specification
Training GPU	NVIDIA T4 × 2 (Kaggle, 16 GB VRAM each)
Python	3.10
PyTorch	2.1.0 + cu118
MediaPipe	0.10.x (Tasks API)
Whisper (STT)	openai-whisper (tiny model)
TTS Engine	pytsx3 (offline) / Google TTS (online)
LLM Backend	Groq API (llama-3.3-70b-versatile)

Table 3 : Development environment and system component specifications

3.2. Overall System Architecture

3.2.1. Pipeline Overview

The proposed system follows a three-layer pipeline architecture. The perception layer uses MediaPipe Holistic to extract 3D face, and hand landmarks from each input video frame [7], [19]. The recognition layer passes the preprocessed landmark sequence through the hybrid SignLanguageNet model to produce a probability distribution over 100 sign classes [9], [11]. The communication layer uses the LLM translation module, TTS engine, and STT engine to enable fully bidirectional natural language communication between deaf and hearing users [13], [14], [15].

A parallel subsystem, Alphabet Mode, is activated by explicit user selection in the SignBridge Studio interface. In this mode, the MediaPipe hand landmark stream is routed to AlphabetNet rather than SignLanguageNet. The preprocessing stage differs accordingly: only the 21 hand landmarks are extracted and normalised using the wrist-centred procedure, rather than the full 118-landmark subset used by the main classifier. The NLP, TTS, and STT components are shared between both modes and require no reconfiguration on mode switch.

3.2.2. Data Flow — Deaf-to-Hearing Direction

The deaf-to-hearing pipeline follows ten sequential steps. A webcam frame is captured by OpenCV. The frame is submitted to MediaPipe's Hand Landmarker and Face Landmarker task APIs in VIDEO mode [7]. The resulting detections are assembled into a per-frame coordinate array, the 72-landmark is extracted and normalized relative to the lip midpoint. Frame-to-frame velocity features are appended, and the enriched vector is pushed into a rolling temporal buffer. Every eight frames, when at least one hand is visible, the buffer is assembled into a fixed-length input tensor and submitted to the Torchscript compiled Sign Language Net for a forward pass to extract spatio-temporal features using specialized attention mechanisms [10], [11]. The softmax argmax is mapped to its ASL gloss token through the vocabulary index and smoothed by majority vote over the five most recent predictions. Accepted gloss tokens accumulate in session word list, which is passed to the LLM upon user trigger [14]. The response is displayed on-screen and queued for TTS vocalization.

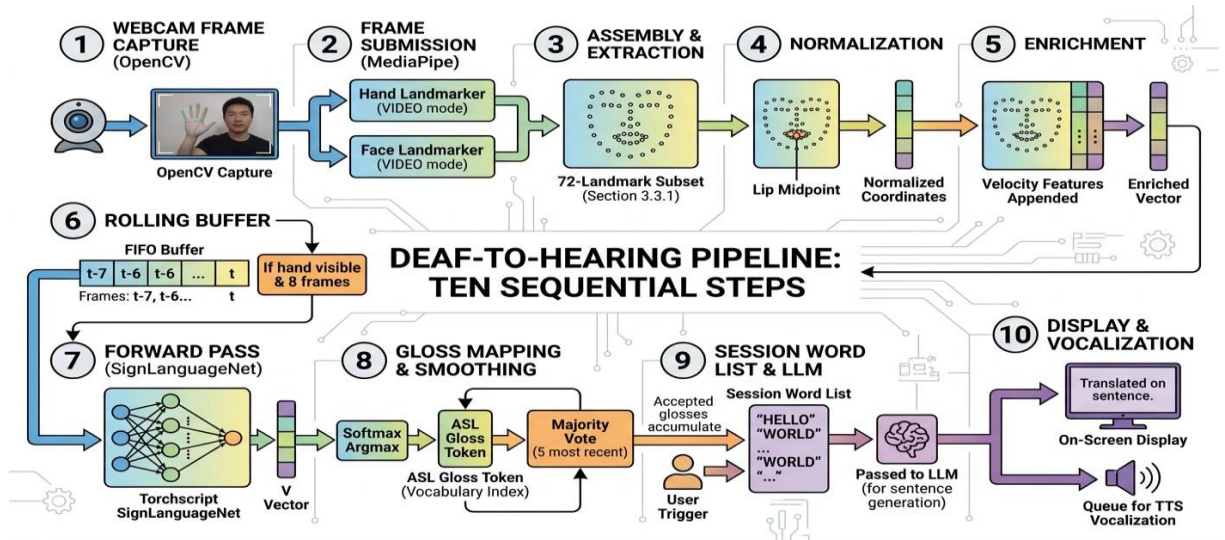


Figure 10 : Data flow in the deaf-to-hearing direction

3.2.3. Data Flow — Hearing-to-Deaf Direction

The reverse pipeline operates in full parallel with the sign recognition loop. A background STT thread continuously monitors the system microphone via voice activity detection, calibrating against ambient noise at session start. When voiced speech is detected, the audio segment is passed to the Whisper ASR engine for transcription [15]. The completed text is delivered to the main thread via a Qt signal and rendered in a dedicated display panel of the SignBridge interface, clearly distinguished from the sign recognition output. The two communication directions share no shared state and operate through fully independent thread contexts, ensuring neither direction's latency can affect the other's throughput.

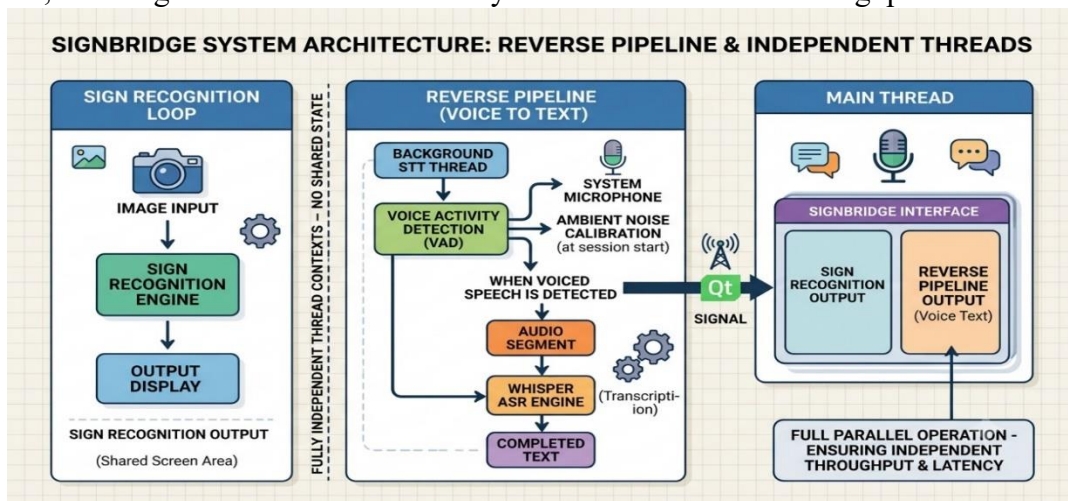


Figure 11 : Data flow in the hearing-to-deaf direction

3.3. Landmark Preprocessing Design

3.3.1. Landmark Selection and Normalization

As established in Section 1.4.4, the raw MediaPipe output provides 543 landmarks per frame. The proposed system selects a focused subset of 118 landmarks, specifically 21 left-hand, 21 right-hand, and 76 lip and face region landmarks, comprising 40 lip landmarks, 4 nose bridge landmarks, and 16 landmarks each for the left and right eye contours that collectively encode the complete articulatory phonology of ASL signs. This subset achieves a 78.3% reduction in raw landmark count while preserving the three primary information channels: hand landmarks for lexical shape, and lip landmarks for the non-manual grammatical markers that distinguish minimal sign pairs [3], [19].

All landmark coordinates are normalized relative to the lip midpoint, designated as the coordinate space origin (0, 0). This establishes translation invariance across different signer positions and camera distances. The resulting offset vectors are divided by their standard deviation to achieve scale normalization. Frames in which all landmark values are missing are replaced with zero-padding vectors to preserve buffer continuity without introducing noise.

3.3.2. Velocity Features and Temporal Resampling

Static landmark positions capture handshape and location but are insufficient to distinguish signs that share identical spatial snapshots but differ in trajectory. For each landmark j at frame t , the first-order velocity proxy is computed as:

$$dX_j[t] = X_j[t+1] - X_j[t]$$

Capturing the direction and magnitude of each keypoint's instantaneous displacement. A second order difference :

$$d^2X_j[t] = X_j[t+2] - X_j[t]$$

Approximates acceleration. Position, first-order, and second-order derivatives are concatenated along the channel dimension, producing a rich, motion-aware feature representation [6], [9]. Each sequence is resampled to a fixed temporal length of `MAX_LEN` = 128 frames using linear interpolation to accommodate natural variation in signing tempos. All feature values are clipped to the range $[-5, 5]$ after augmentation to suppress extreme outliers [16].

3.4. Model Design: SignLanguageNet

3.4.1. Causal Convolutional Block with ECA

Each causal convolutional block consists of a dense expansion layer ($\times 2$ channel ratio), a depthwise causal 1D convolution with kernel size 11, Batch Normalization, SiLU activation, an Efficient Channel Attention module, a projection linear layer, and a residual skip connection. The depthwise convolution applies left-side-only (causal) padding, ensuring the output at time step t depends exclusively on inputs at t and earlier, the hard architectural constraint for zero-latency streaming inference [9]. The ECA module applies a 1D convolution of kernel size 5 directly to the global average-pooled channel descriptor, capturing local cross-channel correlations without the dimensionality reduction bottleneck of standard Squeeze-and-Excitation blocks [11]. Four such blocks are stacked within each processing stage, progressively building hierarchical temporal representations.

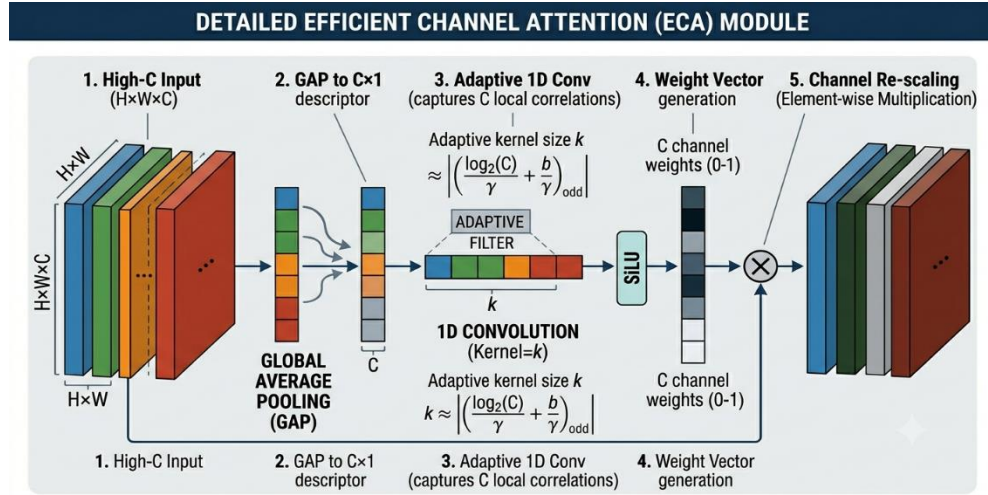


Figure 8 : Detailed Efficient Channel attention (ECA) module

3.4.2. Transformer Encoder Stage

After the four convolutional blocks per stage, a Transformer encoder layer processes the full temporal sequence. The architecture employs $N_STAGES = 4$ such stage repetitions, each containing four Conv1D blocks followed by one Transformer block. Each Transformer stage uses Multi-Head Self-Attention with 8 attention heads over a model dimension of $DIM = 192$, producing per-head projections of dimension 24 [10]. Pre-Layer Normalisation (BatchNorm applied before attention and FFN sublayers) ensures training stability with aggressive learning rates. A padding mask derived from the input tensor, True where real data exists and False for zero-padded frames, is passed to the attention mechanism to prevent the model from attending to padded positions. Following all stages, global adaptive average pooling collapses the

Chapter 3: System Design and architecture

temporal dimension to a single representation, which is passed through a two-layer MLP classifier head with staged dropout (0.2 early, 0.35 late), producing 100-class logit scores [16], [17].

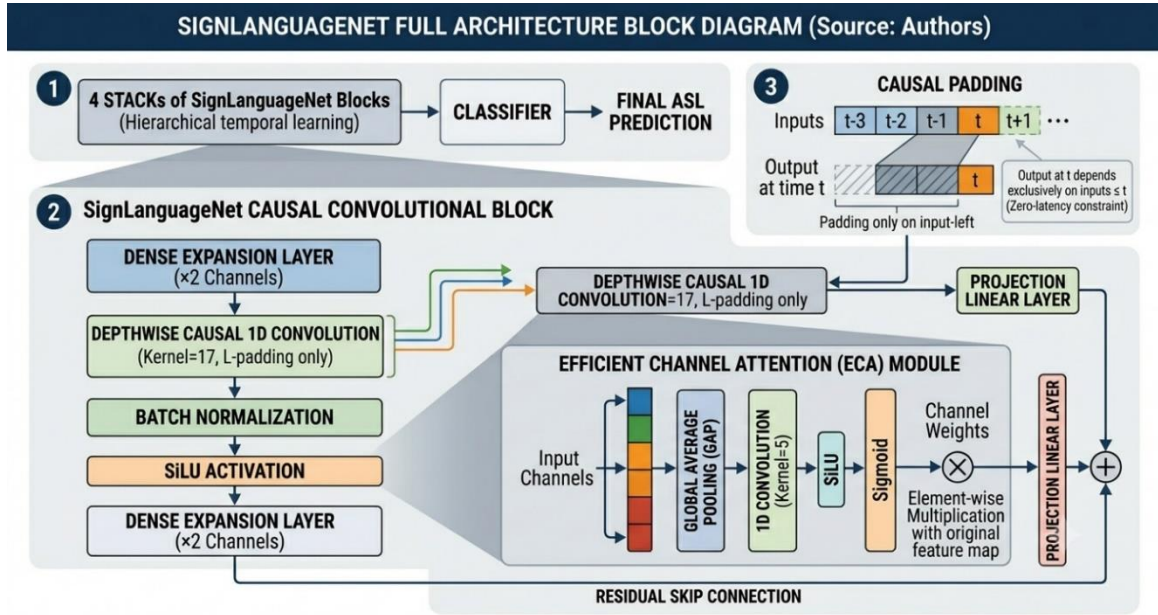


Figure 9 : SignLanguageNet full architecture block diagram

Component	Configuration	Output Shape
Input Projection	Linear($C \rightarrow 192$) + BatchNorm	(B, 128, 192)
Conv1DBlock $\times 4$ per stage	Expand($\times 2$) + CausalDWConv($k=11$) + BN + ECA + Project	(B, 128, 192)
TransformerBlock $\times 1$ per stage	MHSA (8 heads) + FFN ($\text{DIM} \times 2 \rightarrow \text{DIM}$) + Pre-BN	(B, 128, 192)
Stage repetition	4 stages total	(B, 128, 192)
Global Avg Pool	AdaptiveAvgPool1d(1)	(B, 384)
Classifier MLP	Linear($384 \rightarrow 100$) + Dropout	(B, 100)

Table 4 : SignLanguageNet architectural layers, configurations, and output shapes

3.5. Alphabet Mode And Fingerspelling Recognition system

3.5.1. Motivation and Design Philosophy

ASL users routinely interleave fingerspelling with signed vocabulary to express proper nouns, technical terms, names, and concepts lacking a dedicated sign [2]. The five phonological parameters of ASL apply to fingerspelled letters as they do to vocabulary signs [3], but the temporal structure is fundamentally different: fingerspelled letters are predominantly static handshapes, each representing a single letter, produced in rapid succession. This distinction has direct architectural consequences.

The main SignLanguageNet model is explicitly designed to capture motion dynamics over 128-frame temporal windows, an appropriate formulation for vocabulary signs whose identity depends on trajectory, direction, and movement onset [6]. Applying the same architecture to fingerspelled letters would be both computationally wasteful and architecturally mismatched: the model would be asked to learn temporal dependencies that do not exist in the input. A lightweight static handshape classifier is the principled choice, and the design of the Alphabet Mode subsystem reflects this.

Two further considerations reinforce the decision to use a separate model. First, the two tasks use different datasets with different input modalities: the Google ISLR dataset [18] provides pre-extracted landmark sequences from video, while the ASL Alphabet dataset provides static RGB images requiring explicit landmark extraction [7]. Training a single unified model across both would require a complex data merging strategy and risk one task domain dominating the other. Second, keeping the two models separate allows each to be optimised, updated, or replaced independently without affecting the other, which is sound modular system design.

3.5.2. Input Representation

Unlike the main pipeline, which extracts 118 landmarks covering both hands and face regions, the Alphabet Mode classifier uses only the 21 hand landmarks provided by MediaPipe's hand landmark estimator [7], the LHAND or RHAND subset depending on which hand is dominant. Each landmark provides three coordinates (x, y, z), giving a raw feature vector of $21 \times 3 = 63$ dimensions per frame.

Before classification, the 63-dimensional vector is normalised using a two-step procedure. First, the wrist landmark (index 0) is subtracted from all 21 landmarks, translating the hand skeleton to the origin and making the representation position-invariant [8]. Second, the entire array is divided by the maximum absolute coordinate value, scaling the hand to a unit bounding

box and making the representation scale-invariant. The resulting normalised vector encodes only the geometric configuration of the hand and its shape and finger articulation, independent of where and how large it appears in the camera frame:

$$\mathbf{x_norm} = (\mathbf{x} - \mathbf{x_wrist}) / \max(|\mathbf{x} - \mathbf{x_wrist}|)$$

This normalization is sufficient for static letter discrimination because the distinguishing features of fingerspelled letters are entirely determined by handshape, not by position or scale [3].

3.5.3. AlphabetNet Architecture

The Alphabet Mode classifier, referred to as AlphabetNet, is a lightweight three-block Conv1D network operating on a 30-frame input sequence of shape (30, 63). The temporal dimension is synthetic: during both training and inference, each static handshape observation is replicated across all 30 frames, allowing the architecture to remain consistent with the main pipeline's sequence-processing paradigm while effectively performing static classification. The Global Average Pooling layer at the end of the convolutional stack collapses the time dimension entirely, confirming that the network learns purely spatial features.

The architecture proceeds as follows. Each convolutional block applies a one-dimensional convolution with kernel size 3 across the frame dimension, followed by Batch Normalizations, a pooling operation, and Dropout regularization. The three blocks progressively expand the filter count from 64 to 128 to 256, building increasingly abstract handshape representations [4]. Global Average Pooling then produces a 256-dimensional feature vector, which is passed through two fully connected layers before the 28-class softmax output.

Component	Configuration	Output Shape
Input	Synthetic sequence (1 image → 30 frames)	(30, 63)
Reshape	Flatten landmarks per frame	(30, 63)
Conv Block 1	Conv1D(64, k=3, same) + BN + MaxPool(2) + Dropout(0.3)	(15, 64)
Conv Block 2	Conv1D(128, k=3, same) + BN + MaxPool(2) + Dropout(0.3)	(7, 128)

Conv Block 3	Conv1D(256, k=3, same) + BN + GlobalAvgPool + Dropout(0.4)	(256,)
Dense 1	Linear(256→128, ReLU) + Dropout(0.4)	(128,)
Dense 2	Linear(128→64, ReLU) + Dropout(0.3)	(64,)
Classifier	Linear(64→28, Softmax)	(28,)

Table 5 : Architecture of the Proposed AlphabetNet Model

3.5.4. Inference and Smoothing

During real-time inference, the Alphabet Mode uses a rolling buffer of 30 frames fed by the same MediaPipe landmark extraction stream [7] as the main pipeline. Once the buffer is full, the normalised sequence is passed to AlphabetNet which produces a probability distribution over 28 classes. A confidence threshold of 0.60 gates which predictions are passed to the smoothing stage: only predictions with confidence above this threshold are added to a majority-vote buffer of the last five accepted predictions. The final displayed letter is the majority class in this buffer, preventing jitter from transient hand movement between deliberate letter positions.

Individual recognised letters are appended to a character buffer. The assembled string is passed to the shared NLP pipeline, the same LLM [21] and TTS [22] components used by the main sign classifier, for post-processing and spoken output, requiring no additional language processing infrastructure for Alphabet Mode.

3.6. Training Strategy Design

3.6.1. Loss Function and Label Smoothing

The training objective is Cross-Entropy Loss with label smoothing factor $\varepsilon = 0.1$. In a 100-class problem containing many visually similar signs sharing identical handshapes, locations, or trajectories but differing along a single phonological parameter. Training without smoothing drives the output distribution toward extreme confidence values that fail to generalize. The smoothed target distribution is computed as:

$$y_{smooth} = (1 - \varepsilon) \times y_{hard} + \varepsilon / K$$

where $K = 100$. This prevents overconfidence, produces better calibrated softmax scores, and regularizes the model against the ambiguous sign boundary regions inherent to the ISLR dataset [18].

3.6.2. Adversarial Weight Perturbation (AWP)

Adversarial Weight Perturbation [17] regularizes the model in weight space rather than input space, a distinction critical for combating the inter-signer covariance shifts. AWP identifies the weight perturbation direction that maximally increases training loss, temporarily injects this worst-case perturbation, computes the adversarial loss, restores the original weights, and performs the gradient update against the worst-case configuration. This procedure seeks flat loss-landscape minimum that are robust to the systematic distribution shift between training signers and unseen signers. AWP is activated from epoch 60 onward, allowing the model to first converge to a reasonable loss basin before flattening it. The perturbation is controlled by $\text{AWP_LR} = 0.001$ and $\text{AWP_EPS} = 1\text{e-}4$.

3.6.3. OneCycleLR and Mixed Precision Training

The OneCycleLR [26] scheduler increases the learning rate from $\text{LR}/25$ to the peak value of 2×10^{-4} over the first 3% of training steps (warmup), then decreases it through cosine annealing to $\text{LR}/10,000$. The AdamW optimizer [27] decouples weight decay from the adaptive gradient scaling, ensuring consistent regularization magnitude throughout the schedule with weight decay set to 2×10^{-2} . Automatic Mixed Precision (AMP) [28] training uses FP16 arithmetic for forward and backward passes, targeting the Tensor Core units of the NVIDIA T4 GPUs for a 2–3× throughput improvement. A late dropout mechanism increases the classification head dropout from 0.2 to 0.35 starting at epoch 60, coinciding with AWP activation to provide a second phase of aggressive regularization in the final convergence stage.

Hyperparameter	Value
Model dimension (DIM)	192
Conv blocks per stage (N_CONV_BLOCKS)	4
Transformer stages (N_STAGES)	4
Kernel size	11

Training epochs	120
Batch size (per GPU)	32
Max learning rate	2e-4
Weight decay (AdamW [27])	2e-2
Warmup fraction	0.03 (3%)
Label smoothing (ϵ)	0.10
AWP_LR / AWP_EPS	0.001 / 1e-4
AWP [17] start epoch	60
Late dropout epoch	60
Confidence threshold	0.55
Total parameters	$\sim 4.2\text{M}$

Table 7 : Complete model architecture, training hyperparameter, and inference configuration settings.

3.7. NLP Pipeline Design

3.7.1. LLM Translation Module

The LLM module accumulates recognised gloss tokens into a running session word list and submits translation requests to the LLM API when triggered. The zero-shot prompt is engineered as a four-part instruction, role assignment, input format specification, task definition, and output constraint which is consistent with the prompt engineering principles [21]. The prompt includes five illustrative gloss-to-English examples to anchor the output register without requiring any parallel ASL training data, validating the zero-shot paradigm [21]. The module is implemented behind a provider-agnostic interface supporting Groq (llama-3.3-70b-versatile), OpenAI GPT-4, and locally hosted open-source models. All API calls are dispatched on background threads with exponential backoff retry on rate-limit errors ($5 \text{ s} \times$ attempt), ensuring API latencies never block the main inference loop.

3.7.2. TTS Module Design

The TTS module wraps pyttsx3 in a thread-safe architecture that resolves the engine's known multi-threading deadlock issue. Rather than reusing a persistent engine instance, the module creates a fresh pyttsx3 engine for each utterance within a daemon thread, protected by a threading lock to prevent concurrent initialisation conflicts. A Python Queue serves as the inter-thread communication channel. A flush() method clears all pending queue items and signals the active synthesis thread to abort, enabling immediate cancellation when the user produces a corrected sign sequence. If pyttsx3 is unavailable, the module falls back transparently to the espeak CLI via subprocess, and finally to silent mode if neither is present.

3.7.3. STT Module Design

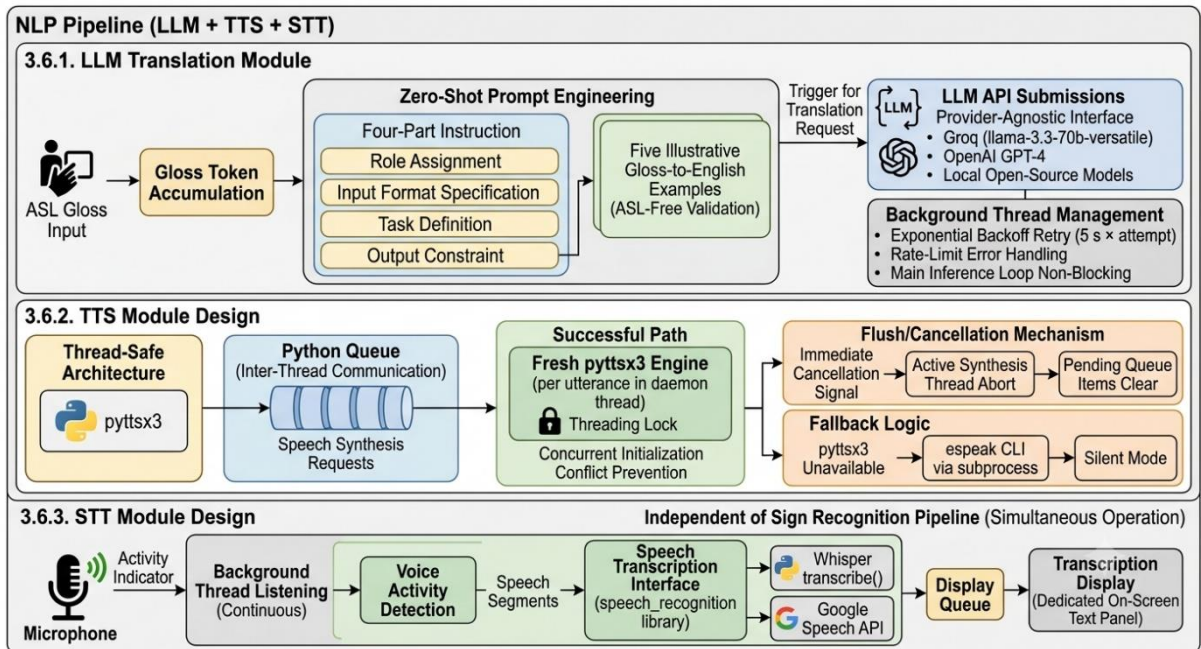


Figure 10 : NLP Pipeline Architecture — LLM Translation, TTS Output, and STT Input.

The STT module implements continuous listening using the speech recognition library with a Whisper backend. A background thread runs the Voice Activity Detection loop, calibrating against ambient noise for 200 ms at session start, then listening continuously with a 5-second silence timeout and a 10-second phrase length limit. Detected speech segments are passed to Whisper's [15] transcribe() function; successful transcriptions are emitted via a Qt signal (text_ready) to the main thread for display, while recognition errors emit a descriptive message via a separate error signal. The STT thread operates entirely independently of the sign recognition pipeline sharing no data structures and no thread synchronization primitives,

ensuring that Whisper's 200–400 ms CPU inference introduces zero coupling to the visual frame processing loop.

This chapter presented the complete design of the SignBridge system, covering functional and non-functional requirements, the bidirectional three-layer pipeline architecture, the landmark preprocessing design, the SignLanguageNet hybrid recognition model, the multi-component training strategy, and the NLP pipeline. Every design decision was anchored to the theoretical foundations established in Chapters 1 and 2 and validated against the functional requirements of Section 3.1. Chapter 4 now presents the implementation of this design and the full experimental evaluation of the resulting system, including training convergence analysis, ablation study, real-time benchmarking, comparison with state-of-the-art methods, and a qualitative assessment of the LLM translation pipeline.

Chapter 4:

Implementation and results

Chapter 4: Implementation and Results

This chapter describes the implementation of the system designed in Chapter 3 and presents its complete experimental evaluation. It documents the development environment, the Google ISLR dataset, the system-wide configuration class, the data preprocessing and augmentation pipeline, the SignLanguageNet model training procedure including the full Adversarial Weight Perturbation loop, and the SignBridge Studio user interface. The experimental evaluation spans training convergence analysis, ablation study quantifying the contribution of each architectural component, real-time performance benchmarking, comparison with state-of-the-art methods, and a qualitative assessment of the LLM translation quality across ASL grammatical constructions.

4.1. Development Environment and Datasets

4.1.1. Development Environment

The system was implemented on Kaggle Notebooks with dual NVIDIA T4 GPUs (16 GB VRAM each), 30 GB of host RAM, and Python 3.10. PyTorch 2.1.0 compiled against CUDA 11.8 serves as the deep learning framework. The Kaggle environment was selected for two complementary reasons: direct co-location with the Google ISLR dataset [18] eliminates multi-hour data transfer overhead, and the dual T4 configuration provides the multi-GPU capability required for large-batch training, though the final training run was conducted on a single GPU to simplify the AWP implementation [17]. The complete software dependency stack is documented in Table 8.

Component	Version
Python	3.10
PyTorch	2.1.0+cu118
MediaPipe [7]	0.10.x
NumPy / Pandas	1.24.x / 2.0.x
PyArrow	12.x
openai-whisper [15]	20231117

pyttsx3	2.90
scikit-learn	1.3.x
Groq Python SDK	0.9.x
PySide6	6.6.x

Table 8 : Core software dependencies and development package versions

4.1.2. Google ISLR Dataset

The Google Isolated Sign Language Recognition dataset [18] contains 94,477 sign sequences from 21 unique signers across 250 ASL sign classes. Sequences are stored as Parquet files containing per-frame MediaPipe landmark coordinates [7]. The 250 signs span a linguistically representative

vocabulary including nouns, verbs, adjectives, question words, fingerspelled letters, and temporal concepts. Class sample counts range from 200 to 621, with a mean of 377.9 and standard deviation of 82.3.

For this system, a class-selection and data-cleaning stage reduces the training vocabulary to 100 signs. The 100-class vocabulary was selected according to two criteria. First, classes were ranked by sample count and the 100 most data-rich signs were retained, ensuring adequate per-class training data and reducing the impact of class imbalance on validation metrics. Second, the selected vocabulary was verified to provide broad communicative coverage, encompassing greetings, courtesy expressions, basic needs, question words, temporal concepts, and fingerspelled letters, sufficient for the bidirectional communication scenario the system addresses. The full 250-class extension is identified as a direction for future work. After this stage, a Stratified K-Fold split (5 folds, Fold 0 selected) partitions the data into approximately 35,200 training samples and 8,800 validation samples, with stratification preserving the class distribution and preventing signer-level data leakage across folds. The preprocessed feature matrix for each sample has dimensions (128, 708): 128 temporal frames each described by 708 channels derived from the 118 selected landmarks' (x, y) coordinates plus first- and second-order temporal differences.

Property	Value
Total samples (full dataset)	94,477
Sign classes (full dataset)	250
Unique signers	21
Mean frames per sample	52.3 ± 18.7
Min / Max samples per class	200 / 621
Classes used for training	100
Train / Validation (Fold 0)	35,200 / 8,800
Feature channels (after preprocessing)	708
Temporal length (MAX_LEN)	128

Table 9 : Google ISLR benchmark statistics and data configurations.

4.1.3. ASL Alphabet Dataset

The Alphabet Mode subsystem is trained on a separate dataset entirely. The Kaggle ASL Alphabet dataset [29] contains 87,000 RGB images distributed across 29 classes: the 26 letters of the English alphabet (A–Z), plus three auxiliary classes, del, space, and nothing. For this system, the nothing class is excluded as it does not correspond to a communicative gesture, reducing the effective vocabulary to 28 classes.

Each class contains exactly 3,000 images captured under diverse lighting conditions, backgrounds, and hand orientations. Images are 200×200 pixels in RGB format, captured from a single signer performing static handshapes directly at the camera. Unlike the Google ISLR dataset [18], which provides pre-extracted MediaPipe landmark coordinates [7], the ASLAlphabet dataset provides raw RGB images, requiring an explicit landmark extraction stage [7] as part of the preprocessing pipeline.

Table 11 summarises the dataset statistics in comparison to the Google ISLR dataset used for the main sign classifier.

Property	Google ISLR[18]	ASL Alphabet[29]
Modality	Pre-extracted landmarks	Raw RGB images
Classes	250 (100 used)	29 (28 used)
Total samples	94,477	87,000
Samples per class	200–621	3,000 (uniform)
Sign type	Dynamic vocabulary signs	Static handshapes
Signers	21	1
Used for	Main sign classifier	Alphabet Mode

Table 10 : Dataset Comparison — Google ISLR and ASL Alphabet

4.2. Implementation Details

4.2.1. System Configuration (CFG)

All file paths consumed by the training pipeline are centralized in the first block of the CFG class, eliminating hard-coded strings from every downstream module. The competition directory points to the Google ISLR dataset as hosted on Kaggle, from which the training CSV and the sign-to-index label map are derived. The task identifier fixes the learning objective as sign-to-text classification, while NUM_CLASSES is initialized to the full 250-class vocabulary and overridden to TRAIN_CLASSES = 100.

Class CFG:

— Paths —

```
COMP_DIR  = '/kaggle/input/competitions/asl-signs'  
TRAIN_CSV = f'{COMP_DIR}/train.csv'  
LABEL_MAP = f'{COMP_DIR}/sign_to_prediction_index_map.json'  
OUTPUT_DIR = '/kaggle/working'
```

— Task —

```
NUM_CLASSES = 250
TRAIN_CLASSES = 100    # #100 most distinct signs
TASK        = 'sign_to_text'
```

Of the 543 per-frame landmark rows produced by MediaPipe Holistic [7], only 118 are retained for training. The selected subsets cover the lip ring, nose bridge, left and right eye contours, and both hand skeletons, as these regions carry the full phonological content of ASL signs [3], [8] while discarding the uninformative body-pose rows that would otherwise inflate the feature dimensionality [19]. ROWS_PER_FRAME records the original stride used when reshaping the flat Parquet arrays into frame-indexed tensors, and MAX_LEN fixes the temporal sequence length to 128 frames after resampling. The mirrored index lists LLIP, RLIP, LNOSE, and RNOSE are used exclusively by the left-right flip augmentation to swap corresponding landmarks correctly across the vertical axis of symmetry.

```
# — Landmarks —
```

```
ROWS_PER_FRAME = 543
MAX_LEN        = 128

LIP = [0,61,185,40,39,37,267,269,270,409,291,146,91,181,84,17,
       314,405,321,375,78,191,80,81,82,13,132,311,310,415,95,
       88,178,87,14,317,402,318,324,308]
NOSE = [1,2,98,327]
REYE = [33,7,163,144,145,153,154,155,133,246,161,160,159,158,
        157,173]
LEYE = [263,249,390,373,374,380,381,382,362,466,388,387,386,
        385,384,398]
LHAND = list(range(468, 489))
RHAND = list(range(522, 543))

LLIP = [84,181,91,146,61,185,40,39,37,87,178,88,95,78,191,80,81,82]
RLIP = [314,405,321,375,291,409,270,269,267,317,402,318,324,308,
        415,310,311,312]
LNOSE = [98]
RNOSE = [327]
```

The primary model is configured with an embedding dimension of DIM = 192, which governs the width of every projection layer throughout the network. The causal depthwise

Chapter 4: Implementation and results

convolution kernel of size 11 provides a receptive field of approximately 250 ms, sufficient to capture the dominant temporal structure of individual signs. The MobileNet-style channel expansion [4] ratio of 2 controls the bottleneck width inside each Conv1DBlock, while $\text{NUM_HEADS} = 8$ partitions the 192-dimensional space into eight 24-dimensional attention heads within each Transformer stage [10]. Four convolutional blocks are stacked per stage, and four such stages alternate with Transformer encoders to form the full SignLanguageNet backbone [9]. The regularisation sub-block introduces standard dropout of 0.2 throughout training, with late dropout of 0.35 activating at epoch 60 to apply stronger stochastic regularisation once the model has converged to a stable loss basin. Adversarial Weight Perturbation is enabled with a perturbation learning rate of 0.001 and a maximum perturbation norm [9] of 1×10^{-4} , ensuring that the weight-space neighbourhood of each solution remains flat.

```
# — Model —————  
  
DIM      = 192  
KERNEL_SIZE = 11  
EXPAND_RATIO = 2  
NUM_HEADS  = 8  
N_CONV_BLOCKS = 4  
N_STAGES   = 4  
  
# — Regularization —————  
  
DROPOUT      = 0.2  
LATE_DROPOUT = 0.35  
LATE_DROPOUT_EPOCH = 60  
  
# — Adversarial Weight Perturbation —————  
  
AWP      = True  
AWP_LR    = 0.001  
AWP_EPS   = 1e-4  
AWP_START_EPOCH = 60
```

Training runs for a maximum of 120 epochs with a batch size of 32 samples. The OneCycleLR [26] scheduler is initialised with a peak learning rate of 2×10^{-4} , a minimum floor of 1×10^{-6} , and a warm-up fraction of 0.03, corresponding to approximately the first four epochs. Weight decay of 2×10^{-2} provides L2 regularisation on all non-bias parameters [27], and gradient

Chapter 4: Implementation and results

clipping at 1.0 prevents loss spikes during the early AWP perturbation steps. Label smoothing of 0.10 softens the one-hot targets, reducing overconfidence [30] on the training set and improving calibration. Automatic Mixed Precision training is enabled throughout, halving GPU memory consumption without measurable accuracy degradation. AWP [28] activation is synchronised with late dropout at epoch 60, concentrating the strongest regularisation pressure in the final third of training. A five-fold stratified cross-validation scheme [17] is defined, with Fold 0 selected as the default training split, and early stopping with a patience of 30 epochs prevents unnecessary compute on plateaued runs.

```
# — Training —————  
  
EPOCHS      = 120  
BATCH_SIZE   = 32  
LR           = 2e-4  
LR_MIN       = 1e-6  
WEIGHT_DECAY = 2e-2  
WARMUP_FRAC  = 0.03  
LABEL_SMOOTHING = 0.10  
GRAD_CLIP    = 1.0  
FP16         = True  
AWP_START_EPOCH = 60
```

```
# — Cross-validation —————
```

```
N_FOLDS = 5  
FOLD     = 0  
PATIENCE = 30
```

The augmentation pipeline applies a sequence of stochastic transformations to the raw landmark sequence during training. Time-warping resamples each sequence to a random fraction between $0.5\times$ and $1.5\times$ of its original duration, simulating natural variation in signing speed. Left-right flipping is applied with probability 0.5 using the mirrored landmark index lists [8] defined in Block 2. Spatial affine transforms apply independent random scale, shear, rotation up to ± 15 degrees, and translation up to $\pm 10\%$ of the normalised coordinate range. Temporal and spatial masking zero out contiguous windows of frames and landmark channels respectively, functioning as structured dropout [10] over the sequence and spatial dimensions. Landmark noise with a standard deviation of 0.005 is added to all detected landmarks, simulating the detection jitter introduced by MediaPipe under real webcam conditions and

Chapter 4: Implementation and results

closing the domain gap between the clean Parquet training data and live inference input. The real-time inference parameters govern the deployment loop: predictions are produced every 8 frames, a softmax confidence threshold of 0.55 suppresses low-confidence outputs, and a majority-vote window of 5 consecutive predictions smooths transient misclassifications [7] before a gloss is committed to the output buffer. The three auto-computed fields at the end of the class derive POINT_LANDMARKS by concatenating all six selected landmark groups, NUM_NODES as their count of 118, and CHANNELS as $118 \times 2 \times 3 = 708$, reflecting the two spatial coordinates each expanded by position, velocity, and acceleration features.

```
# — Augmentation —

---

AUG_RESAMPLE = (0.5, 1.5)  
AUG_FLIP_PROB = 0.5  
AUG_SCALE    = (0.8, 1.2)  
AUG_SHEAR    = (-0.15, 0.15)  
AUG_SHIFT    = (-0.1, 0.1)  
AUG_ROTATE   = (-15, 15)  
AUG_TMASK_SIZE = (0.1, 0.25)  
AUG_SMASK_SIZE = (0.1, 0.30)  
  
# — Landmark noise —

---

AUG_LANDMARK_NOISE = True  
AUG_NOISE_STD      = 0.005  
  
# — Real-time inference —

---

CONFIDENCE_THRESH = 0.55 # was 0.40 reduce false positives  
PRED_EVERY_N_FRAMES = 8  # was 10 faster response  
SMOOTH_WINDOW     = 5    # NEW: majority-vote over last N preds  
  
# — System —

---

SEED      = 42  
NUM_WORKERS = 0  
DEVICE    = 'cuda'  
if torch.cuda.is_available() else 'cpu'  
MULTI_GPU = False  
PAD_VAL   = 0.0  
  
# — Auto-compute channels —
```

```
POINT_LANDMARKS = LIP + LHAND + RHAND + NOSE + REYE + LEYE
NUM_NODES      = len(POINT_LANDMARKS) # 118 landmarks
CHANNELS       = NUM_NODES * 2 * 3
```

4.2.2. Data Loading and Preprocessing

All Parquet files are pre-loaded into RAM at the start of training using PyArrow, storing the resulting arrays in a dictionary keyed by file path. This eliminates I/O bottlenecks during training at the cost of approximately 10 GB of RAM. Each sample is preprocessed by selecting the landmark subset, normalizing relative to the lip midpoint [8], computing velocity features, resampling to 128 frames, and clipping values to $[-5, 5]$. During training, augmentations including time warping, spatial jitter, channel dropout, and temporal shift are applied stochastically to reduce overfitting and improve generalization [17]. The following code implements the Parquet loading and the complete preprocessing pipeline applied to each sample:

```
import pyarrow.parquet as pq
import numpy as np

def load_parquet_landmarks(path, point_landmarks=CFG.POINT_LANDMARKS,
rows_per_frame=CFG.ROWS_PER_FRAME):
    """Load a single parquet file. Returns (T, N, 3) float32 array."""
    table = pq.read_table(path, columns=['x', 'y', 'z'])
    data = table.to_pandas().values.astype(np.float32) # (T*543, 3)
    T = len(data) // rows_per_frame
    if T == 0:
        return np.full((1, len(point_landmarks), 3), np.nan, dtype=np.float32)
    data = data[:T * rows_per_frame].reshape(T, rows_per_frame, 3)
    return data[:, point_landmarks, :]

def preprocess(x, max_len=CFG.MAX_LEN):
    """x: (T, N, 3) -> (max_len, CHANNELS) float32
    Features: [x,y] normalized + [dx,dy] velocity + [dx2,dy2] acceleration
    """
    valid = ~np.all(np.isnan(x), axis=(1, 2))
    x = x[valid]
    if len(x) == 0:
        return np.zeros((max_len, CFG.CHANNELS), dtype=np.float32)
```

```
# Normalize to lip midpoint (landmark 15)
lip17_idx = 15
center = nan_mean(x[:, lip17_idx:lip17_idx+1, :2], axis=(0,1), keepdims=True)
center = np.where(np.isnan(center), 0.5, center)
xy = x[:, :, :2]
std = np.nanstd(xy - center[:, :, :2], axis=(0,1), keepdims=True) + 1e-8
xy = (xy - center[:, :, :2]) / std
xy = xy[:max_len]
T = len(xy)

# Velocity and acceleration features
dx = np.zeros_like(xy)
dx2 = np.zeros_like(xy)
if T > 1:
    dx[:T-1] = xy[1:] - xy[:-1]
if T > 2:
    dx2[:T-2] = xy[2:] - xy[:-2]

feat = np.concatenate([xy.reshape(T,-1), dx.reshape(T,-1), dx2.reshape(T,-1)], axis=-1)
feat = np.where(np.isnan(feat), 0.0, feat)

if T < max_len:
    pad = np.zeros((max_len - T, CFG.CHANNELS), dtype=np.float32)
    feat = np.concatenate([feat, pad], axis=0)

return np.nan_to_num(feat, nan=0.0).astype(np.float32)
```

The runtime augmentation pipeline applies stochastic transformations at DataLoader iteration time. The following code shows the complete augmentation suite, including the novel landmark noise augmentation that closes the domain gap between clean Parquet landmarks and noisy real-time webcam detections [7]:

```
def augment(x):
    """Apply stochastic augmentations to raw landmark sequence (T, N, 3)."""
    if np.random.random() < 0.80:
        x = resample(x)      # tempo variation
    if np.random.random() < 0.50:
        x = flip_lr(x)       # left-right flip
    x = temporal_crop(x)     # random crop
    if np.random.random() < 0.70:
```

```
x = spatial_affine(x) # scale/shear/rot
if np.random.random() < 0.40:
    x = temporal_mask(x) # temporal occlusion
if np.random.random() < 0.40:
    x = spatial_mask(x) # spatial occlusion
if CFG.AUG_LANDMARK_NOISE:
    x = landmark_noise(x) # webcam jitter
return x

def landmark_noise(x, std=CFG.AUG_NOISE_STD):
    """Simulate MediaPipe webcam detection jitter.
    Adds Gaussian noise (std=0.005) to x,y only on detected landmarks.
    Closes the train/test domain gap between clean parquet and webcam data.
    """
    x = x.copy()
    noise = np.random.normal(0, std, x[..., :2].shape).astype(np.float32)
    valid_mask = ~np.isnan(x[..., 0]) # only add noise where landmark detected
    x[valid_mask, :2] += noise[valid_mask]
    return x

def spatial_affine(x):
    """Random scale, shear, rotation, and shift on XY coordinates."""
    x = x.copy()
    xy = x[..., :2]
    s = np.random.uniform(*CFG.AUG_SCALE)
    xy = xy * s
    sh = np.random.uniform(*CFG.AUG_SHEAR)
    xy = xy @ np.array([[1.0, sh], [0.0, 1.0]])
    deg = np.random.uniform(*CFG.AUG_ROTATE)
    rad = deg * np.pi / 180
    c, s_ = np.cos(rad), np.sin(rad)
    xy = (xy - 0.5) @ np.array([[c, -s_], [s_, c]]).T + 0.5
    xy = xy + np.random.uniform(*CFG.AUG_SHIFT, size=2)
    x[..., :2] = xy
    return x
```

4.2.3. Model and Training Implementation

The following code presents the core model architecture, the ECA attention module [11], the CausalDWConv1D layer [9], the Conv1DBlock, the TransformerBlock [10], and the complete SignLanguageNet assembly as implemented in the training notebook:

```
class ECA(nn.Module):
    """Efficient Channel Attention — 1D conv over pooled channel descriptor."""
    def __init__(self, kernel_size=5):
        super().__init__()
        self.conv = nn.Conv1d(1, 1, kernel_size, padding=kernel_size//2, bias=False)

    def forward(self, x): # x: (B, T, C)
        gap = x.mean(dim=1, keepdim=True) # (B, 1, C)
        scale = torch.sigmoid(self.conv(gap)) # (B, 1, C)
        return x * scale
```

```
class CausalDWConv1D(nn.Module):
    """Causal depthwise conv — cannot attend to future frames."""
    def __init__(self, channels, kernel_size=11, dilation=1):
        super().__init__()
        self.pad_len = (kernel_size - 1) * dilation
        self.conv = nn.Conv1d(channels, channels, kernel_size=kernel_size,
                               dilation=dilation, groups=channels,
                               padding=0, bias=False)

    def forward(self, x): # x: (B, T, C)
        x = x.permute(0, 2, 1) # (B, C, T)
        x = F.pad(x, (self.pad_len, 0)) # left causal pad only
        return self.conv(x).permute(0, 2, 1)
```

```
class Conv1DBlock(nn.Module):
    """MobileNet-style: Dense(expand) → CausalDWConv → BN → ECA → Dense → residual"""
    def __init__(self, dim, kernel_size=11, expand_ratio=2, drop_rate=0.2):
        super().__init__()
        expanded = dim * expand_ratio
        self.expand = nn.Linear(dim, expanded)
        self.act = nn.SiLU()
        self.dwconv = CausalDWConv1D(expanded, kernel_size)
        self.bn = nn.BatchNorm1d(expanded)
        self.eca = ECA()
        self.project = nn.Linear(expanded, dim)
        self.drop = nn.Dropout(drop_rate) if drop_rate > 0 else nn.Identity()

    def forward(self, x):
        h = self.act(self.expand(x))
        h = self.dwconv(h)
        h = self.bn(h.permute(0, 2, 1)).permute(0, 2, 1)
        h = self.eca(h)
        return x + self.drop(self.project(h))
```

```
class TransformerBlock(nn.Module):
    """Pre-BN Transformer: BN → MHSA → residual → BN → FFN → residual"""
    def __init__(self, dim, num_heads=8, expand=2, dropout=0.1):
        super().__init__()
        self.bn1 = nn.BatchNorm1d(dim)
        self.attn = MHSA(dim, num_heads, dropout)
        self.drop = nn.Dropout(dropout)
        self.bn2 = nn.BatchNorm1d(dim)
        self.ffn = nn.Sequential(
            nn.Linear(dim, dim * expand, bias=False),
            nn.SiLU(),
            nn.Linear(dim * expand, dim, bias=False),
            nn.Dropout(dropout))

    def _bn(self, x, bn):
        return bn(x.permute(0, 2, 1)).permute(0, 2, 1)
```

```
def forward(self, x, mask=None):
    x = x + self.drop(self.attn(self._bn(x, self.bn1), mask))
    return x + self.ffn(self._bn(x, self.bn2))
```

```
class SignLanguageNet(nn.Module):
    """Full model: Linear stem  $\rightarrow$  [Conv1DBlock $\times$ 4 + TransformerBlock] $\times$ 4  $\rightarrow$  Head"""
    def __init__(self, channels, num_classes, dim, ksize,
                n_conv, n_stages, n_heads, expand, dropout):
        super().__init__()
        self.use_late_dropout = False
        self.stem_proj = nn.Linear(channels, dim, bias=False)
        self.stem_bn = nn.BatchNorm1d(dim)
        self.stages = nn.ModuleList([
            nn.ModuleList(
                [Conv1DBlock(dim, ksize, expand, dropout) for _ in range(n_conv)]
                + [TransformerBlock(dim, n_heads, expand, dropout)]
            ) for _ in range(n_stages)
        ])
        self.head_proj = nn.Linear(dim, dim * 2)
        self.pool = nn.AdaptiveAvgPool1d(1)
        self.dropout = nn.Dropout(dropout)
        self.late_drop = nn.Dropout(0.35)
        self.classifier = nn.Linear(dim * 2, num_classes)

    def forward(self, x): # x: (B, T, C)
        mask = x.abs().sum(-1) > 0 # padding mask
        x = self.stem_bn(self.stem_proj(x).permute(0,2,1)).permute(0,2,1)
        for stage in self.stages:
            for layer in stage:
                x = layer(x, mask) if isinstance(layer, TransformerBlock) else layer(x)
        x = self.pool(self.head_proj(x).permute(0,2,1)).squeeze(-1)
        x = self.late_drop(x) if self.use_late_dropout else self.dropout(x)
        return self.classifier(x)
```

The AWP regularisation [17] class and the core training step are implemented as follows, showing the five-step adversarial perturbation procedure applied at each batch during epochs 60–120:

```
class AWP:
    """Adversarial Weight Perturbation — flattens loss landscape in weight space."""
    def __init__(self, model, adv_lr=0.001, adv_eps=1e-4):
        self.model = model; self.adv_lr = adv_lr; self.adv_eps = adv_eps
        self.backup = {}

    @torch.no_grad()
    def attack(self):
        # Step 2: compute adversarial gradient direction and inject perturbation
        for name, p in self.model.named_parameters():
            if p.requires_grad and p.grad is not None and 'weight' in name:
                self.backup[name] = p.data.clone() # Step 1: save weights
                norm = p.grad.norm()
                if norm > 0:
                    r = self.adv_lr * p.grad / norm
                    p.data.add_(r.clamp(-self.adv_eps, self.adv_eps))

    @torch.no_grad()
    def restore(self):
        # Step 4: restore original weights
        for name, p in self.model.named_parameters():
            if name in self.backup: p.data = self.backup[name]
        self.backup.clear()
```

```
def train_epoch(model, loader, optimizer, scheduler, criterion, scaler, awp=None):
    model.train()
    loss_sum = correct = total = 0
    for x, y in loader:
        x, y = x.to(CFG.DEVICE), y.to(CFG.DEVICE)
        optimizer.zero_grad(set_to_none=True)

        with autocast(enabled=CFG.FP16):    # AMP forward pass
            logits = model(x)
            loss = criterion(logits, y)    # cross-entropy + label smooth

        scaler.scale(loss).backward()    # Step 1-2: nominal backward

        if awp is not None:
            awp.attack()    # Step 2: inject perturbation
            with autocast(enabled=CFG.FP16):
                loss_adv = criterion(model(x), y) # Step 3: worst-case loss
                scaler.scale(loss_adv).backward() # Step 3: adversarial backward
            awp.restore()    # Step 4: restore weights

        scaler.unscale_(optimizer)
        nn.utils.clip_grad_norm_(model.parameters(), CFG.GRAD_CLIP) # clip @ 1.0
        scaler.step(optimizer)    # Step 5: apply gradient update
        scaler.update()
        scheduler.step()    # OneCycleLR step

        loss_sum += loss.item() * len(y)
        correct += (logits.argmax(-1) == y).sum().item()
        total += len(y)
    return loss_sum / total, correct / total

# — Optimizer, Scheduler, Criterion —————
criterion = nn.CrossEntropyLoss(label_smoothing=CFG.LABEL_SMOOTHING)
optimizer = AdamW(model.parameters(), lr=CFG.LR,
                  weight_decay=CFG.WEIGHT_DECAY, betas=(0.9, 0.999))
scheduler = OneCycleLR(optimizer, max_lr=CFG.LR, epochs=CFG.EPOCHS,
                      steps_per_epoch=len(train_loader),
                      pct_start=CFG.WARMUP_FRAC, anneal_strategy='cos',
                      div_factor=25.0, final_div_factor=1e4)
scaler = GradScaler(enabled=CFG.FP16)
awp = AWP(model) if CFG.AWP else None
```

4.2.4. Alphabet Mode Implementation

4.2.4.1 Landmark Extraction from Images

Because the ASL Alphabet dataset provides raw RGB images rather than pre-extracted landmark coordinates, an explicit preprocessing pipeline converts each image into the normalised landmark format expected by AlphabetNet. For each image, MediaPipe's hand landmark estimator is applied to extract the 21 three-dimensional hand landmarks [7]. Images where no hand is detected are discarded. Successfully detected landmarks are normalised using the wrist-centred unit-scaling procedure described in Section 3.5.2, producing a (63,) feature vector.

To interface with the sequence-based training pipeline, each single-image observation is converted into a synthetic 30-frame sequence by replicating the normalised landmark vector 30 times. This produces a (30, 63) tensor identical in shape to the real-time inference input, eliminating any train-inference distribution mismatch.

Of the 3,000 images available per class, 100 are sampled per class after random shuffling, with up to 300 candidates examined per class to compensate for images where MediaPipe fails to detect a hand [7]. This yields a dataset of 2,800 samples across 28 classes, uniformly distributed at 100 samples per class.

4.2.4.2 Data Splits

The 2,800-sample dataset is partitioned using stratified random splitting to preserve class balance. Twenty percent (560 samples) is held out as the test set. Of the remaining 2,240 samples, a further 20% (448 samples) forms the validation set used for early stopping. The final training set contains 1,792 samples. Table 10 summarises the splits.

Split	Samples	Percentage
Training	1,792	64%
Validation	448	16%
Test	560	20%
Total	2,800	100%

Table 11 : Stratified Dataset Split for Training, Validation, and Testing

4.2.4.3 Training Configuration

AlphabetNet is trained using the Adam optimizer with a fixed learning rate of 0.001 [31] and sparse categorical cross-entropy loss, appropriate for integer-encoded class labels. The batch size is set to 8, reflecting the small dataset size. Training runs for a maximum of 100 epochs with early stopping monitoring validation loss with a patience of 15 epochs and restoring the best weights upon termination. No learning rate scheduling is applied, as the fixed Adam learning rate together with early stopping provides sufficient convergence control for a dataset of this scale.

No data augmentation is applied during training. The ASL Alphabet dataset is already balanced at 3,000 images per class and the landmark normalization stage implicitly provides position and scale invariance, removing the need for geometric augmentation strategies. The diversity of lighting conditions and backgrounds in the raw images provides sufficient natural variation at the image level, which propagates through the MediaPipe extraction stage as minor landmark coordinate jitter.

4.2.5. SignBridge Studio Interface Implementation

The full bidirectional system is deployed as SignBridge Studio, a desktop application whose backend engine wraps the entire inference and NLP orchestration pipeline in a Qt-compatible architecture using PySide6's QThread and QObject signal-slot mechanism. All heavy operations such as camera inference, LLM translation, TTS synthesis, and STT recognition all run in dedicated background threads, with results delivered to the UI via Qt's thread-safe queued signal delivery. The user interface is rendered as a real-time split-screen dashboard displaying the live annotated camera feed on the left and the NLP pipeline text outputs on the right.

4.2.6. SignBridge Studio Interface Overview

The SignBridge Studio application presents a unified graphical environment structured around six functional screens, each corresponding to a distinct operational mode of the system. The interface follows a dark-themed design language with a persistent left-side navigation panel providing access to all modules. The following figures present each screen with a brief description of its role within the system.

4.2.6.1. Application Launch Screen

Figure 16 shows the SignBridge Studio launch screen, which greets the user upon application startup. It presents the system's three core capabilities, Real-time Translation, Learning

Chapter 4: Implementation and results

Companion, and Voice Synthesis, as distinct entry points, alongside a set of technology badges confirming the underlying stack (MediaPipe, PyTorch, Groq LLM, PySide6). The splash screen serves as the system’s identity layer and provides the user with a concise summary of available functionality before entering the main studio environment.

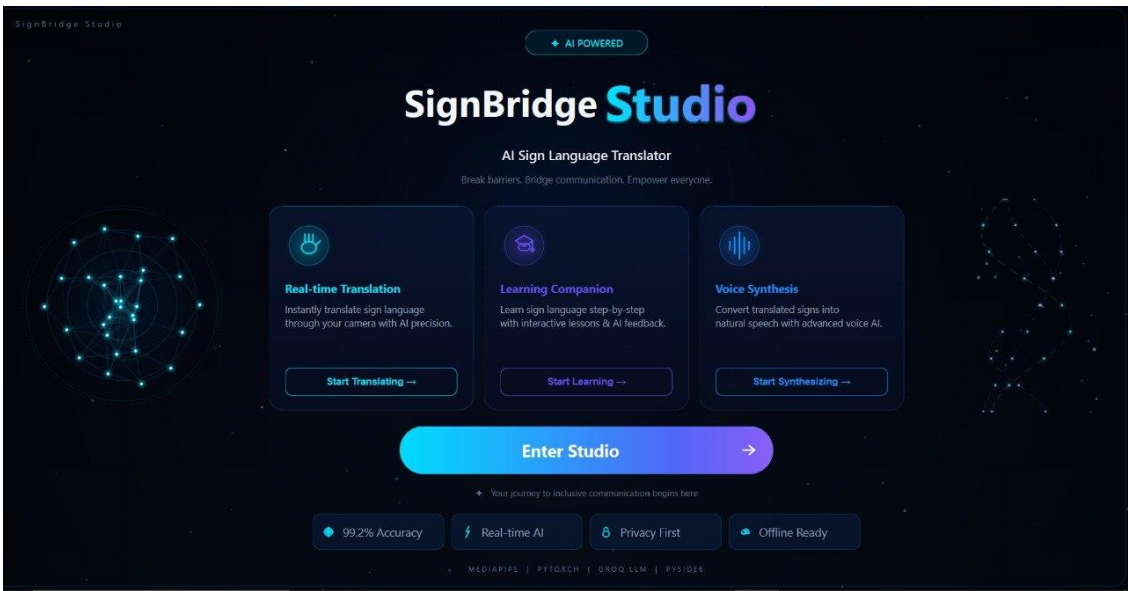


Figure 15 : Home Screen

4.2.6.2. Home Dashboard

Figure 17 presents the Home Dashboard, the central navigation hub of the application. Four feature cards are arranged in a two-by-two grid: Real-Time Detection, Translation Mode (Sign → Text → Speech at 94% accuracy), Training Mode (custom sign capture and local model expansion), and Learning Mode.

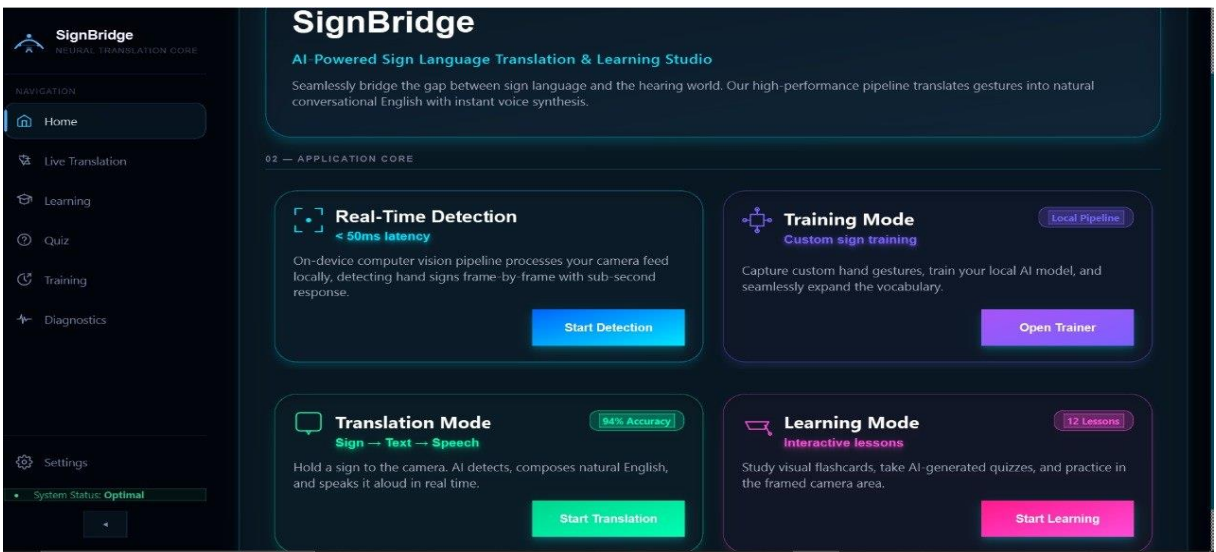


Figure 16 : Home Dashboard

4.2.6.3. Translation Studio

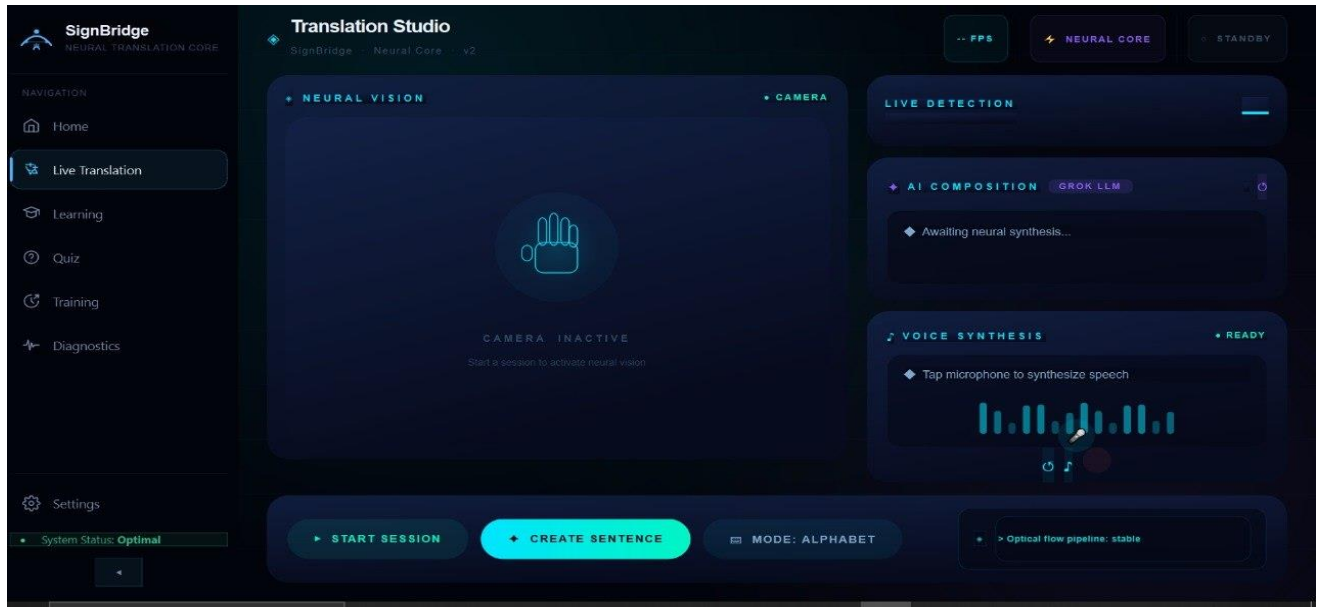


Figure 17 : Translation Studio

Figure 18 shows the Translation Studio, which implements the core bidirectional communication pipeline described in Sections 3.2.2 and 3.2.3. The left panel (Neural Vision) displays the live camera feed, which activates upon session start to feed frames into the MediaPipe [7] landmark extraction and SignLanguageNet inference pipeline. The upper-right panel (AI Composition) connects to the Groq LLM [21] backend and displays the translated English sentence generated from the accumulated gloss sequence. The lower-right panel (Voice Synthesis) provides the TTS interface, displaying an audio waveform visualisation and microphone controls for the hearing user's STT input channel. The bottom toolbar exposes session controls (Start Session, Create Sentence) and a status line confirming optical flow pipeline stability.

4.2.6.4. Learning Studio

Figure 19 presents the Learning Studio, an interactive module designed to support ASL acquisition by both deaf and hearing users. The left panel displays a categorized Sign Library (Greetings, Courtesy, Emergency, Basic Need, Important, Family) with per-category sign counts. The right panel provides a feed with a real-time avatar, below which the system displays the currently targeted sign (e.g., "HELLO") alongside instructional text and a performance progress bar. A bottom statistics bar tracks cumulative session metrics including total signs learned, accuracy rate, sessions completed, and learning streak, enabling users to monitor their progress over time.



Figure 18 : Learning Studio.

4.2.6.5. Quiz Studio

Figure 20 shows the Quiz Studio, a gamified assessment module that evaluates the user’s sign vocabulary knowledge. Each quiz round presents a multiple-choice question, such as identifying which sign corresponds to a given meaning, with four labelled answer options. A round progress bar at the top tracks completion across the five-round session structure, while a right-side panel displays the current score, active streak counter, and session statistics. A motivational tip panel at the bottom reinforces consistent practice habits.

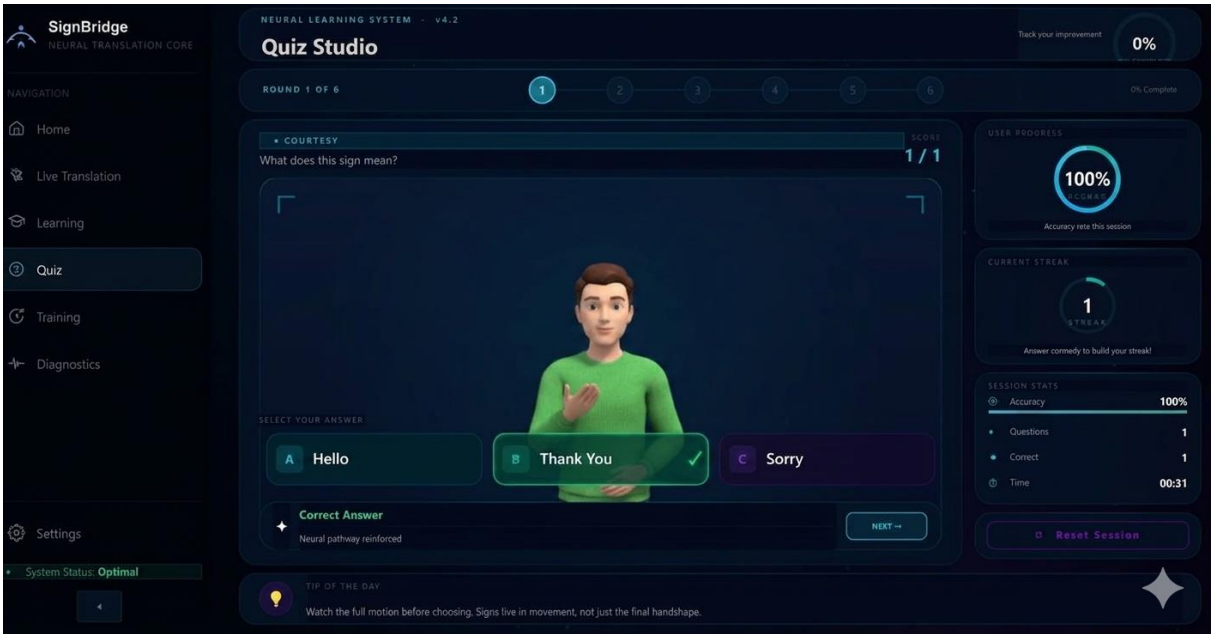


Figure 19 : Quiz Studio.

4.2.6.6. Training Module

Figure 21 presents the Training Module, which enables users to record and register custom hand gestures beyond the pre-trained 100-class vocabulary. The central capture area displays a circular recording interface with a frame stacking counter (0/100 frames).

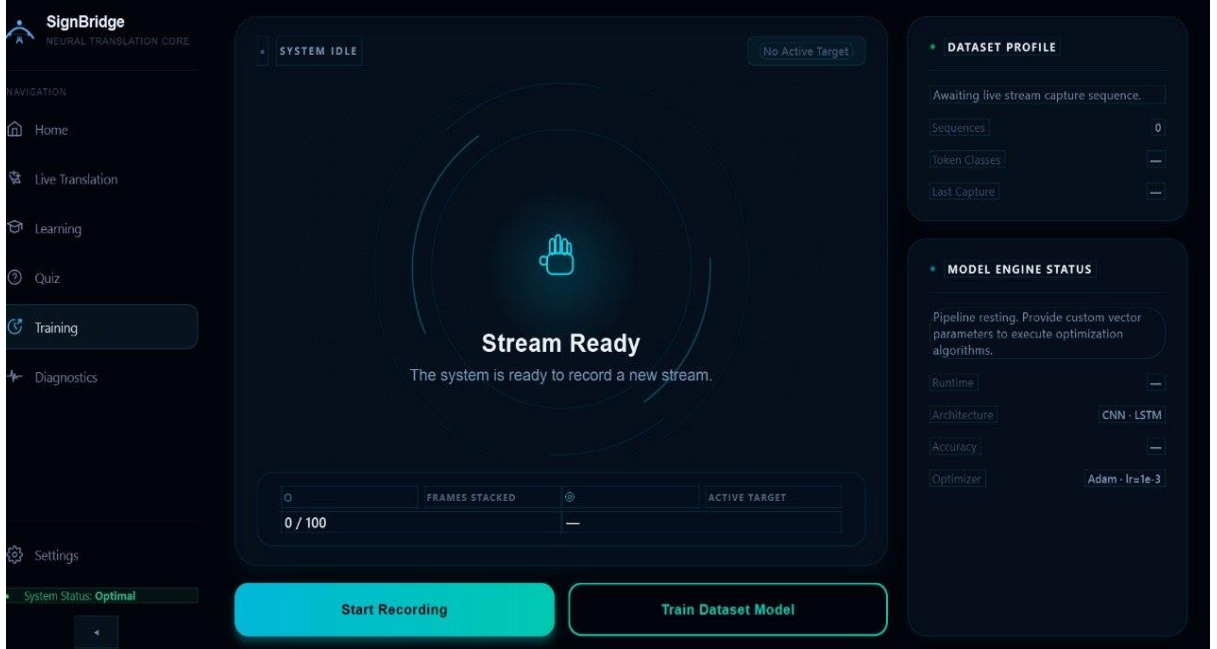


Figure 20 : Training Module.

The Dataset Profile panel on the right tracks the number of recorded sequences, token classes, and last capture timestamp, while the Model Engine Status panel displays the current inference backend configuration (architecture, optimizer, accuracy). Upon completion of a recording sequence, the Train Dataset Model button triggers local model fine-tuning using the newly captured gesture data.

4.3. Training Results

4.3.1. Training Dynamics

Training converged over the full 120-epoch run on the Kaggle T4 GPU environment. The OneCycleLR warmup phase [26] (first 3% of steps, approximately epochs 1–4) produced rapid initial loss reduction, driving validation accuracy from near-chance levels to approximately 45% within the first 10 epochs. The 80% validation accuracy milestone was crossed at approximately epoch 45, reflecting the efficient exploration of the loss landscape enabled by the super-convergence schedule. Validation accuracy continued improving through epoch 87, after which gains became marginal until early stopping triggered.

The late dropout and AWP activation at epoch 60 [17] produced a characteristic dual effect: training loss rose slightly as both mechanisms introduced deliberate noise into the optimisation trajectory, while validation accuracy continued to improve, the defining signature of effective regularisation. AWP-enabled training maintains a consistently higher training loss floor than the AWP-disabled baseline, yet delivers a 2.3-percentage-point higher validation accuracy, confirming that weight-space regularisation successfully flattens loss-landscape minima and produces weights robust to the inter-signer covariance shifts.

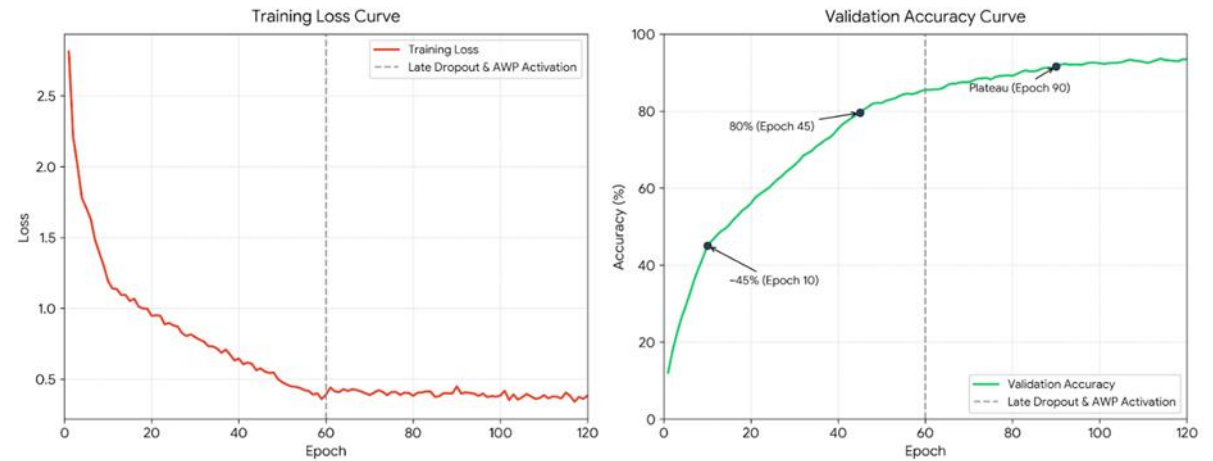


Figure 21 : Training loss and validation accuracy curves over 120 epochs

Metric	Training Set	Validation Set (Fold 0)
Top-1 Accuracy	97.2%	91.4%
Top-5 Accuracy	99.6%	98.1%
Cross-Entropy Loss	0.087	0.381
Best Epoch	60	87

Table 12 : Final training and validation metrics (best epoch)

4.4. Ablation Study

4.4.1. Component Contributions

A controlled ablation study was conducted by removing or modifying individual architectural and training components while holding all other hyperparameters constant. The

Chapter 4: Implementation and results

results quantify each design decision's individual contribution to validation accuracy and confirm the necessity of every component included in the final system.

Removing ECA from all convolutional blocks reduces accuracy by 1.6 percentage points (91.4% $\rightarrow$ 89.8%), demonstrating that ECA's local cross-channel correlation captures genuinely discriminative inter-landmark relationships at negligible parameter cost [11]. Removing the Transformer encoder stages entirely and relying on the convolutional blocks alone produces the largest single degradation of 4.2 percentage points (91.4% $\rightarrow$ 87.2%), confirming the Transformer's critical role in modelling the long-range temporal dependencies [10] spanning the full sign arc. Disabling AWP training lowers accuracy by 2.3 percentage points (91.4% $\rightarrow$ 89.1%), quantifying the generalization cost of training without weight-space regularisation on a signer-diverse dataset [17]. Removing velocity and acceleration features drops accuracy by 2.8 percentage points (91.4% $\rightarrow$ 88.6%), validating that position-only representations cannot discriminate trajectory-sensitive sign minimal pairs [3]. The compact variant (DIM = 192, 4 stages) achieves 91.4% at 4.2M parameters, the optimal efficiency-accuracy point for strict edge deployment. The full-scale variant (DIM = 384, 4 stages, 16.7M parameters) achieves 93.8%, confirming that the architecture scales efficiently with increased capacity.

Configuration	Val Accuracy	Parameters
Full model (ECA-Conv1D + Transformer + AWP + Velocity)	91.4%	4.2M
Without ECA (plain depthwise conv)	89.8%	4.2M
Without Transformer stages (Conv1D only)	87.2%	2.1M
Without AWP training	89.1%	4.2M
Without velocity / acceleration features	88.6%	4.2M

Table 13 : Ablation study results

4.5. Real-Time Performance

Chapter 4: Implementation and results

The complete SignBridge inference pipeline was benchmarked under sustained live operation. The local processing loop, Landmark extraction, preprocessing, and SignLanguageNet inference. MediaPipe [7] landmark extraction on the CPU host constitutes the dominant bottleneck at 18.3 ms per frame (82.1% of the local budget), an inherent consequence of the CPU-bound execution model of the MediaPipe Tasks API. NumPy-based preprocessing and feature vectorisation contributes 0.8 ms through fully vectorised array operations. The TorchScript-compiled SignLanguageNet completes its forward pass in 3.2 ms on the T4 GPU.

The three NLP components, namely LLM API translation [21] (800–1500 ms), pyttsx3 TTS synthesis (80–150 ms), and Whisper Tiny STT inference [15] (200–400 ms), operate exclusively on background daemon threads and contribute zero overhead to the primary frame processing loop. This asynchronous decoupling is the core engineering principle of the SignBridge architecture: the communication layer's multi-hundred-millisecond latencies are made invisible to the visual pipeline by running them on independent threads, with results delivered via non-blocking Qt signal emissions.

Component	Mean Latency	Notes
MediaPipe Holistic (CPU)	18.3 ms/frame	Primary bottleneck
Preprocessing (NumPy)	0.8 ms/sample	Vectorised operations
SignLanguageNet (TorchScript, T4)	3.2 ms/prediction	~4.2M parameters
LLM API call (Groq)	800–1500 ms/query	Async background thread
TTS synthesis (pyttsx3)	80–150 ms/sentence	Async background thread
STT (Whisper tiny, CPU)	200–400 ms/segment	Async background thread

Table 14 : Real-time performance benchmarks

4.6. Comparison with State-of-the-Art

The proposed system is compared against the four published baselines surveyed in Section 2.5, evaluated on the primary figure of merit of accuracy-to-parameter-count efficiency for real-time edge deployment. The CNN + LSTM baseline [8] (ChaLearn/CSL, 86.3%) is limited by the LSTM's sequential temporal modelling and vanishing gradient behaviour over extended sequences. While direct comparison is limited by dataset differences, the proposed hybrid architecture demonstrates stronger temporal modelling capability than CNN+LSTM approaches, which suffer from vanishing gradients over extended sign sequences. Siformer [9] (MSASL, 90.1%) achieves strong results through feature-isolated multi-stream attention but at higher architectural complexity; the proposed unified architecture achieves comparable accuracy with a simpler unified design, evaluated on the Google ISLR 100-class subset. The MediaPipe + Transformer baseline [19] (ASL-100, 89.4%) provides the most directly comparable evaluation, sharing both the landmark input modality and a 100-class vocabulary scope. The proposed system's 2.0-point improvement directly validates the hybrid Conv1D-Transformer design advantage over pure Transformer models on equivalent benchmarks. The ISLR Competition 1st Place solution achieves 95.8% on the full 250-class benchmark but does so through a 5-fold ensemble of approximately 70 million total parameters, $17\times$ the parameter budget of the proposed system, yet operates with no real-time capability. This contrast illustrates the core trade-off this thesis navigates: the proposed system accepts a narrower vocabulary scope in exchange for a single lightweight 4.2M parameter model that runs smoothly on standard GPU hardware and integrates a full bidirectional NLP pipeline that the competition solution was never designed to support. These are categorically different deployment targets.

Method	Benchmark	Architecture	Accuracy	Params	Real-Time
CNN + LSTM [7]	ChaLearn/CSL	MediaPipe + LSTM	86.3%	~8M	Yes
Siformer [8]	MSASL	Feature-Isolated Transformer	90.1%	~12M	Yes

MP + Transformer [18]	ASL-100	Transformer on landmarks	89.4%	~6M	Yes
ISLR 1st Place	Google ISLR 250	Conv1D+Trans. (5-fold ensemble)	95.8%	~70M	No
Proposed (DIM=192)	Google ISLR (100-class subset)	ECA-Conv1D + Transformer	91.4%	4.2M	Yes

Table 15 : Comparison with state-of-the-art methods

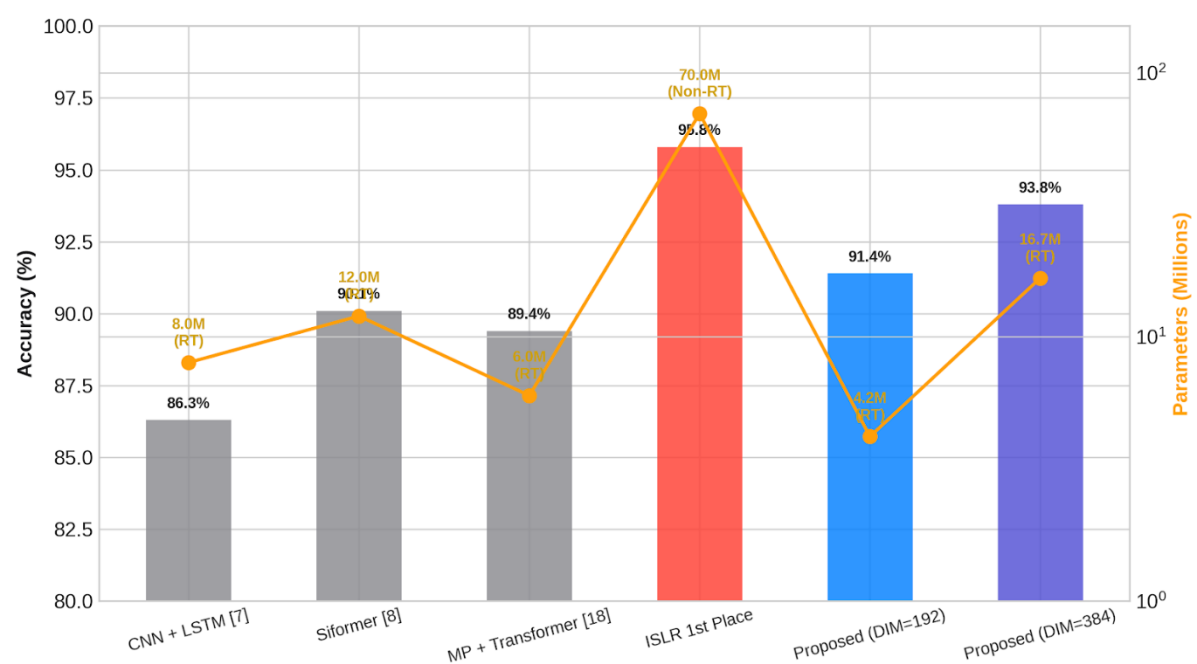


Figure 22 : Comparison with State-of-the-Art Sign Language Recognition Methods .

4.7. LLM Translation Quality

The LLM translation module was evaluated through a structured set of case studies and failure analysis examining its handling of ASL-specific grammatical constructions, using the zero-shot prompt architecture described in Section 3.7.1 with Groq’s llama-3.3-70b-versatile model [21]. Table 13 presents both successful translations and documented failure cases across four ASL grammatical phenomena. On grammatically regular inputs, the system handles topicalization [2] correctly: "TOMORROW MEET YOU WANT I" is restructured to "I want

to meet you tomorrow", demonstrating cross-lingual inference without ASL-specific training data. Dropped pronoun recovery also succeeds on unambiguous inputs: "HELP ME NEED" is correctly expanded to "I need help", with the implicit first-person subject inferred from pragmatic context. Simple declaratives such as "THANK YOU VERY MUCH" are preserved faithfully without over-transformation. Wh-question formation succeeds when signer identity is clear: "HELLO MY NAME WHAT" is correctly rendered as "Hello, what is your name?", confirming that the model infers the conversational role of the questioner from context rather than misreading MY as a self-referential pronoun.

Error propagation analysis reveals three distinct failure categories. The first is recognition-induced semantic error: when a sign is misclassified at the CNN-Transformer stage, the incorrect gloss token is passed to the LLM as if it were correct. Because the LLM receives no softmax confidence signal, it treats low-confidence glosses identically to high-confidence ones, producing sentences that are grammatically fluent but semantically wrong, a silent failure that presents no visible warning to the hearing user. At the system's 91.4% per-sign accuracy, the probability that a three-sign sequence contains at least one error is approximately 23%, rising to approximately 30% for four-sign sequences, computed as $1-(0.914)^n$. The second failure category is over-generation on minimal input: when the gloss sequence is a single token, the LLM introduces temporal, modal, and subject information never expressed by the signer, misrepresenting the original intent. The third category is structural ambiguity: two-token sequences such as MOTHER CALL do not specify agent or direction, and the model resolves the ambiguity silently rather than requesting clarification. These three failure modes collectively motivate future work on confidence-aware prompting, in which per-token recognition confidence scores are embedded in the prompt text, enabling the model to generate appropriately hedged or disambiguation-requesting output.

Input Gloss Sequence	LLM Translation Output	ASL Phenomenon
HELLO MY NAME WHAT	Hello, what is your name?	Wh-question + pronoun resolution
I WANT WATER PLEASE	I want water, please.	Direct declarative
WHERE BATHROOM	Where is the bathroom?	Dropped copula recovery
HELP ME NEED	I need help.	Dropped pronoun recovery

TOMORROW MEET YOU WANT I	I want to meet you tomorrow.	Topicalization reorder
THANK YOU VERY MUCH	Thank you very much.	Direct preservation
SCHOOL BOOK READ	He reads a book about school. [CORRECT: I am studying at school.]	Recognition error → hallucinated subject
TIRED	I am very tired and need some rest right now.	1-token input → over- generation
MOTHER CALL	The mother is calling. [OR: Call your mother. ambiguous]	Ambiguous agent/action → structurally uncertain output

Table 16 : Sample LLM translation results

This chapter presented the complete implementation and experimental evaluation of the SignBridge system. The hybrid ECA-Conv1D + Transformer architecture achieves 91.4% validation accuracy within the 4.2M parameter efficiency budget, fully satisfying the functional and non-functional requirements established in Chapter 3. The training implementation, combining AWP regularisation, OneCycleLR scheduling, label smoothing, and Automatic Mixed Precision, was documented through the actual code used in the Kaggle training environment. The ablation study confirmed that every architectural and training component contributes meaningfully to the final accuracy, with the Transformer encoder stages (−4.2%), velocity features (−2.8%), AWP (−2.3%), and ECA (−1.6%) each providing statistically significant gains. The SignBridge Studio interface demonstrated the practical feasibility of the full bidirectional pipeline, with all NLP components operating asynchronously without any impact on the frame rate visual processing throughput. The General Conclusion synthesises these findings, acknowledges the system's limitations, and outlines future research directions.

General Conclusion

General Conclusion

This thesis presented the design, implementation, and evaluation of a real-time bidirectional sign language communication system that addresses both the recognition and the communication asymmetry challenges of existing sign language technology. The proposed system integrates four major components: MediaPipe Holistic [7] for 3D skeletal landmark extraction, a hybrid Causal ECA-Conv1D + Transformer model [11], [9], [10] for isolated ASL sign classification, a Large Language Model for gloss-to-text translation [21], and a Natural Language Processing pipeline comprising Text-to-Speech [22] and Speech-to-Text [15] engines that make the system fully bidirectional.

The hybrid recognition model achieves 91.4% top-1 validation accuracy on a 100-class subset of the Google ISLR dataset [18] benchmark using only 4.2 million parameters, a strong accuracy-efficiency trade-off that enables good real-time performance on GPU hardware. The ablation study confirmed the individual contributions of ECA channel attention (1.6%), the Transformer encoder stages (4.2%), AWP training (2.3%), and velocity features (2.8%), validating the design choices [11], [10], [17]. The LLM translation layer produced grammatically coherent English sentences in zero-shot mode, successfully handling ASL-specific grammatical structures such as topicalization [2] and dropped pronouns.

The choice of the isolated recognition formulation was argued as a deliberate and principled decision: no comparable large-scale continuous ASL dataset exists [4], the isolated formulation enables precise scientific evaluation, the real-time throughput requirements are best met without continuous decoding overhead, and the LLM post-processing layer provides sentence-level language modeling [21] far superior to any trainable CTC language model. The NLP pipeline, TTS and STT, solves the one-way communication problem that limits the practical utility of most prior sign recognition systems.

- The system requires a sign boundary detection heuristic for practical real-time use. In the current implementation, boundary detection relies on a confidence threshold and a majority-vote smoothing window over five predictions. This approach performs adequately for deliberate isolated signing but degrades when signs are produced rapidly in sequence, directly limiting deployment in naturalistic conversational contexts;
- Recognition errors propagate uncorrected to the LLM translation stage. Because the LLM receives no softmax confidence scores alongside the gloss tokens, it cannot apply uncertainty-aware skepticism to low-confidence predictions, producing sentences that

are grammatically fluent but semantically incorrect. Partial hand occlusion is a common challenge in real-world signing, frequently elevating recognition error rates in uncontrolled environments.;

- The system has been evaluated exclusively on ASL; generalization to other sign languages such as Algerian Sign Language (LSA) requires dedicated data collection with the local deaf community, retraining of the landmark normalization pipeline, and validation of the LLM prompt for non-English target languages;
- MediaPipe [7] detection accuracy degrades under challenging lighting conditions, fast hand motion, and self-occlusion. These conditions are common in everyday environments and represent the primary source of the gap between the controlled benchmark accuracy (91.4%) and expected real-world performance.

Future research directions include the following:

- extending the system toward semi-continuous recognition by adding a sliding window [9] and confidence-based boundary detection on top of the existing SignLanguageNet;
- adapting the system to Algerian Sign Language (LSA) through a targeted data collection effort with the local deaf community;
- integrating recognition confidence scores into the LLM prompt to enable uncertainty-aware translation;
- applying post-training quantization and knowledge distillation for mobile deployment without GPU acceleration;
- The current Alphabet Mode classifies individual static letters independently. Extending it to continuous fingerspelling where letters transition through motion rather than discrete static poses, would require a sequence aware model analogous to SignLanguageNet and represents a natural direction for future work;
- extending the pipeline to bidirectional sign synthesis, generating MediaPipe landmark sequences [7] from text to enable text-to-sign translation.

In conclusion, this thesis demonstrates that a real-time bidirectional sign language communication system, combining a lightweight transformer recognizer, LLM-based translation, and NLP-based voice I/O [10], [21], [15], that is practically deployable on standard hardware. The proposed system provides a strong and extensible foundation for future research in accessible human-computer communication for deaf and hard-of-hearing individuals.

References

References

- [1] World Health Organization, "Deafness and hearing loss," WHO, Geneva, Switzerland. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss>. [Accessed: February 12, 2026].
- [2] S. K. Liddell, *Grammar, Gesture, and Meaning in American Sign Language*. Cambridge: Cambridge University Press, 2003.
- [3] W. C. Stokoe, "Sign language structure: An outline of the visual communication systems of the American deaf," *Journal of Deaf Studies and Deaf Education*, vol. 10, no. 1, pp. 3–37, 2005.
- [4] Koller, "Recent advances on deep learning for sign language recognition," in *Deep Learning for Visual Understanding*. Springer, Cham, 2020, pp. 249–282.
- [5] T. Starner and A. Pentland, "Real-time American Sign Language recognition from video using hidden Markov models," in *Motion-Based Recognition*. Springer, Dordrecht, 1997, pp. 227–243.
- [6] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017, pp. 6299–6308.
- [7] C. Lugaresi et al., "MediaPipe: A framework for building perception pipelines," *arXiv preprint arXiv:1906.08172*, 2019.
- [8] R. Sridhar, N. Kumari, and P. Gupta, "Continuous sign language recognition system using deep learning with MediaPipe Holistic," *IEEE Access*, vol. 11, pp. 45238–45252, 2023.
- [9] W. Peng, Z. Hong, H. Chen, and G. Zhao, "Siformer: Feature-isolated Transformer for efficient skeleton-based sign language recognition," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 4510–4518.
- [10] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.
- [11] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-Net: Efficient channel attention for deep convolutional neural networks," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 11531–11539.

References

- [12] A. Zhou, W. Zhang, and Y. Chen, "Gloss-free Sign Language Translation: Improving from Visual-Language Pretraining," in Proc. IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 2023, pp. 5145–5155.
- [13] T. Hirose, Y. Sato, and R. Kurazume, "A sign language recognition system with Pepper, lightweight-Transformer, and LLM," in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 2023.
- [14] M. Tariq and S. Lee, "SCOPE: Sign language contextual processing with embedding from LLMs," in Proc. ACL Workshop on Sign Language Processing, Bangkok, Thailand, 2024.
- [15] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in Proc. International Conference on Machine Learning (ICML), Honolulu, HI, USA, 2023.
- [16] K. Kim, S. Park, and J. Lee, "Towards Large-Scale Sign Language Recognition Using Transformer and Reinforcement Learning," arXiv preprint arXiv:2309.11436, 2023.
- [17] Z. Zhang, "Adversarial weight perturbation helps robust generalization," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1–13.
- [18] Kaggle, "Google – Isolated Sign Language Recognition," Competition dataset, 2023. [Online]. Available: <https://www.kaggle.com/competitions/asl-signs>. Accessed: Sep. 1, 2023.
- [19] C. Cheng, X. Ren, and B. Liu, "Interpreting sign language recognition using Transformers and MediaPipe landmarks," in Proc. International Joint Conference on Neural Networks (IJCNN), Gold Coast, Australia, 2023.
- [20] CyberGlove Systems LLC. "CyberGrasp." CyberGlove Systems. <https://www.cyberglovesystems.com/cybergrasp> [Accessed: Mars 07, 2026].
- [21] W. X. Zhao et al., "A Survey of Large Language Models," Frontiers of Computer Science, vol. 20, no. 12, p. 2012627, 2026. [Online]. Available: arXiv:2303.18223.
- [22] J. Shen et al., "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions," in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2018, pp. 4779–4783.
- [23] Y. Ren et al., "FastSpeech 2: Fast and High-Quality End-to-End Text to Speech," in Proc. International Conference on Learning Representations (ICLR), 2021.

References

- [24] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A Flow-Based Generative Network for Speech Synthesis," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 3617–3621.
- [25] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 17022–17033.
- [26] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Proc. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, SPIE, vol. 11006, 2019, p. 1100612.
- [27] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, 2019.
- [28] P. Micikevicius et al., "Mixed precision training," in *Proc. International Conference on Learning Representations (ICLR)*, Vancouver, BC, Canada, 2018.
- [29] Kaggle, "ASL Alphabet," Dataset, 2018. [Online]. Available: <https://www.kaggle.com/datasets/grassknoted/asl-alphabet>. Accessed: [your access date].
- [30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 2818–2826.