

Article

Efficient Word-Level Sign Language Recognition Using Quantized Spatiotemporal Deep Learning for Low-Power Microcontrollers

Samuel Longwani Kimpinde  and Peter O. Olukanmi *

Department of Electrical and Electronic Engineering Technology, Faculty of Engineering and the Built Environment, University of Johannesburg, Johannesburg 2092, South Africa; 224172460@student.uj.ac.za

* Correspondence: polukanmi@uj.ac.za

Abstract

Deploying efficient sign language recognition models on edge devices advances inclusive, affordable, and privacy-preserving human–computer interaction. Yet most state-of-the-art architectures target server-class hardware and fail under the strict memory, computation, and energy constraints of microcontrollers. This work introduces S3D-Conv1D, a separable spatiotemporal architecture for isolated word-level sign language recognition, tailored for TinyML deployment. While the idea of separating spatial and temporal processing has been explored in earlier models, the novelty here lies in a deployment pipeline designed from the outset for microcontroller-class constraints: every operator has native INT8 support in TensorFlow Lite, CMSIS-NN, and NNoM; the architecture achieves full integer-only execution with competitive accuracy; and the evaluation scale (100 and 300 classes) substantially exceeds prior TinyML sign language recognition studies. Evaluations on datasets show that S3D-Conv1D achieves 98.96% float32 accuracy on WLASL100 with stable cross-dataset generalization (82.5% on SemLex100). After INT8 quantization, accuracy remains high (98.7% on WLASL100) while compressing to 883 KB, the smallest across all evaluated architectures. An ultralight variant further reduces size to 24.7 KB while sustaining 98.5% accuracy on WLASL100 and 77.2% on WLASL300. Quantization-aware training improves stability, particularly at larger vocabulary scales. Among baselines, S3D achieves strong performances but negligible compression (30.3 MB) due to non-quantization-friendly operators. The MobileNet variant generalizes better with 99.4% on WLASL100 and 97.6% accuracy on SemLex100 but remains large at 2.71 MB in INT8 form. CNN + RNN and e-LSTM depend on unsupported recurrent or attention operators. In contrast, S3D-Conv1D meets all operator compatibility requirements, delivers full INT8 execution with a compact sub-1 MB footprint, and real-time performance. These results demonstrate that competitive word-level sign language recognition is achievable under embedded constraints when architectural design prioritizes quantization stability, operator compatibility, and deployment feasibility from the outset.



Academic Editor: Igor Aizenberg

Received: 23 February 2026

Revised: 21 March 2026

Accepted: 23 March 2026

Published: 25 March 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: sign language recognition; TinyML; quantization; spatiotemporal modeling; microcontrollers; embedded deep learning; inclusive technology

1. Introduction

Hearing loss affects more than 1.5 billion people worldwide, with about 430 million requiring rehabilitation services [1]. Sign languages are complete natural languages with their own grammar and cultural grounding, yet significant barriers persist between signers and non-signers [2,3]. DHH individuals face systemic exclusion in education, healthcare,

and public services. Interpreter shortages, inaccessible materials, and dependency on intermediaries undermine autonomy and safety in everyday life [3–6]. These realities highlight the need for tools that enable independent and reliable communication.

Automatic Sign Language Recognition (SLR) offers a potential technological bridge. Recent deep learning advances have enabled accurate sign recognition from video [7–9], but most state-of-the-art systems depend on GPUs or cloud infrastructure, raising concerns about privacy, latency, and accessibility in low-resource settings. TinyML addresses this by enabling inference on ultra-low-power microcontrollers, supporting fully offline, low-latency, and privacy-preserving operation.

Vision-based SLR models typically combine convolutional spatial encoders with recurrent, temporal convolutional, or 3D convolutional modules [9–11]. Although effective on benchmarks, these architectures remain computationally intensive and poorly aligned with microcontroller constraints. Advances in edge inference frameworks such as TensorFlow Lite for microcontrollers [12] and CMSIS-NN [13] have improved quantized inference support, yet integer-only execution remains incompatible with recurrent and attention-based operators used by most SLR models. In practice, promising SLR systems remain confined to desktop or mobile hardware.

Despite strong progress in large-scale and continuous SLR, fully self-contained word-level recognition under embedded deployment constraints remains largely unexplored. The primary obstacles are: (1) the computational cost of spatiotemporal modeling, (2) the difficulty of achieving stable quantization for sequential architectures, and (3) strict memory and operator compatibility requirements of constrained inference stacks.

This work addresses these challenges by systematically studying word-level SLR under realistic embedded deployment constraints, evaluating five spatiotemporal architectures: S3D [14], e-LSTM [15], CNN + RNN, a MobileNet variant [16], and the proposed S3D-Conv1D. The scope is limited to isolated word-level recognition from grayscale video. Continuous recognition, cloud processing, multimodal inputs, and hardware accelerators are excluded. The central research questions are: (i) which spatiotemporal modeling strategies remain accurate and stable under full INT8 quantization, (ii) how architectural choices influence computational cost and operator compatibility on constrained inference stacks, and (iii) whether competitive recognition is achievable with a constrained operator set and without recurrent layers.

While separating spatial and temporal processing is not new, the novelty of this work lies in the complete deployment pipeline designed from the outset for microcontroller constraints. Every operator in S3D-Conv1D has native INT8 support across TFLite, CMSIS-NN, and NNoM, enabling full integer-only execution without Flex delegates, at an evaluation scale (100 and 300 classes) substantially exceeding prior TinyML SLR studies limited to 3–12 classes. The primary objective is not to maximize raw accuracy but to identify architectures occupying the feasible intersection of accuracy, efficiency, and deployability. Fully on-device recognition preserves privacy, operates without connectivity, and remains usable in classrooms, clinics, and remote environments. This supports scalable, low-cost assistive technology.

2. Related Works

Figure 1 conceptually organizes the literature across three intersecting domains: edge AI frameworks, dataset development, and spatiotemporal modeling. Their convergence defines the current research frontier, while the gaps uncovered motivate the direction of this work towards an efficient, fully quantized word-level SLR for microcontrollers.

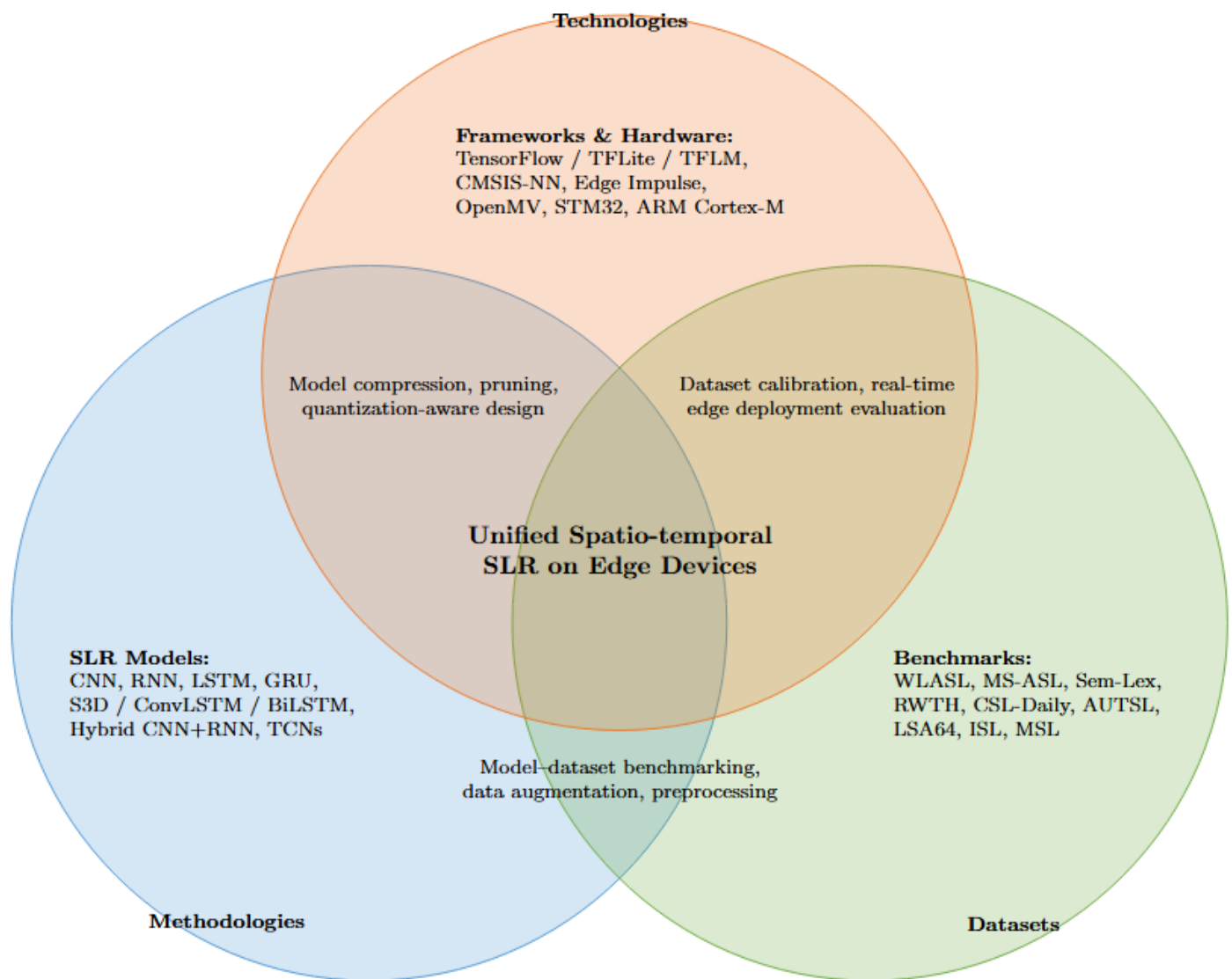


Figure 1. Taxonomy of the literature review showing relationships among methodologies, datasets, and technologies in SLR research on edge.

2.1. Scope and Challenges in Sign Language Recognition

Early SLR systems grew from pattern recognition and computer vision, relying on handcrafted features such as movement trajectories and motion paths. As data availability and compute capacity increased, the field shifted toward deep learning, with mixed CNN, RNN models becoming the dominant paradigm for learning spatiotemporal patterns directly from raw video.

Sign language recognition is more than a computer vision task. It is a communication bridge for DHH individuals across healthcare, education, and everyday interaction [17–19]. Traditional solutions depend on human interpreters who are not always available, affordable, or private. SLR systems promise direct, autonomous communication by translating sign gestures into text or speech in real time.

Sign language is inherently spatiotemporal. Each sign combines hand shape, movement trajectory, facial expression, and rhythm. Word-level recognition is particularly demanding. Words are defined by subtle motion dynamics, and environmental factors such as lighting variation, camera angle, and signer diversity further complicate recognition [18]. Models relying on dense video input also suffer from high frame redundancy, wasting computational resources. CNNs capture spatial features effectively but struggle with motion

continuity. Recurrent networks handle temporal dependencies but are computationally expensive, particularly on edge devices [17–19]. In the broader HCI context, accurate gesture recognition lays the foundation for intuitive human, machine communication [20], where each gesture carries precise semantic meaning.

2.2. Modern Deep Learning Approaches in SLR

2.2.1. Static vs. Dynamic SLR

Sign language recognition divides into static and dynamic settings. Static SLR classifies individual frames, typically targeting alphabets or isolated signs. Dynamic SLR tracks motion across frames, capturing the temporal structure of word- and sentence-level signing. As computational capacity increased, dynamic recognition became the primary research direction. Ref. [21] evaluates CNN, LRCN, and ConvLSTM architectures on WLASL, showing that spatiotemporal fusion via LRCN outperforms purely spatial models, confirming the importance of temporal feature modeling for word-level recognition.

2.2.2. Convolutional and Recurrent Architectures

CNNs extract spatial features, hand shape, finger position, motion boundaries, while recurrent networks track how signs evolve over time [22,23]. Together they enable end-to-end learning from sign language video. Ref. [24] benchmarks I3D, SlowFast, P3D, and R(2 + 1)D on MS-ASL, establishing SlowFast ResNet101 at 92.35% top-1 accuracy as a strong word-level SLR baseline. Ref. [14] proposed S3D in 2018, factorizing 3D convolutions into spatial 2D and temporal 1D operations, achieving near-I3D accuracy at approximately 40% fewer FLOPs. This provided the conceptual foundation for the architecture proposed in this work. Hybrid CNN, RNN designs remain the dominant paradigm. Ref. [10] combines 3D CNNs with BiLSTMs for near-human accuracy on Sports-1M. Ref. [25] achieves 97% on ASL through CNN, RNN multimodal fusion. Ref. [26] reaches 99.51% by integrating the reptile search algorithm (metaheuristic optimizer) into a CNN, LSTM framework.

2.2.3. Hybrid and Multi-Stream Models

Multi-stream architectures combine spatial, temporal, and skeletal cues to build more robust representations [23,27]. Ref. [28] integrates RGB, depth, and wearable sensor data for multimodal SLR, identifying signer variability, limited annotated data, and real-time latency as the primary obstacles. These challenges are directly relevant to this work. Ref. [7] reports over 98% accuracy for sentence-level recognition using depth-based LSTM with Leap Motion sensors, demonstrating the value of 3D motion trajectories for dynamic gesture recognition. Recent hybrids push further. A ViT, CNN, BiLSTM fusion [27] combines local spatial features, global attention, and sequential modeling, though at a parameter and memory cost incompatible with constrained deployment.

2.2.4. Attention and Transformer-Based Architectures

Attention mechanisms allow models to focus on relevant spatiotemporal regions, improving both accuracy and interpretability for long or complex signing sequences [22,29,30]. Transformers extend this by replacing sequential recurrence with self-attention across all frames simultaneously, capturing both local and global temporal dependencies [29,30]. This makes them effective for continuous sign recognition and translation, though their memory and computation demands currently preclude deployment on microcontroller-class hardware.

2.3. Model Compression and Optimization Techniques

Model compression is what makes embedded deep learning practical. Three principal strategies enable complex networks to fit within kilobyte-scale memory budgets: pruning, quantization, and knowledge distillation.

2.3.1. Pruning and Quantization

Pruning removes parameters or neurons that contribute little to model output, reducing size and computation without retraining from scratch [31]. Quantization lowers numerical precision from 32-bit floats to 16-, 8-, or 4-bit integers, yielding smaller models, higher throughput, and lower energy consumption [32]. Applied jointly, these techniques can achieve 92.3% accuracy in ECG anomaly detection at 0.024 mW on wearables [33] and enable gesture recognition on nano drones at 7 FPS using only 72.5 KB of memory [34]. Ref. [35] surveys pruning, quantization, and low-rank factorization in the context of algorithm, hardware co-design for real-time MCU inference, while ref. [36] provides a quantitative meta-analysis of their relative effectiveness. In this work, full INT8 quantization is applied through both PTQ and QAT, targeting the same kilobyte-scale regime demonstrated by these studies.

2.3.2. Knowledge Distillation

Knowledge distillation transfers the learned representations of a large model into a smaller one rather than training the smaller model from scratch [37,38]. The large teacher model provides soft probability outputs that carry more information than hard labels, guiding the student toward better generalization under a reduced parameter budget.

2.4. TinyML and Edge AI in Sign Language Recognition

Tiny Machine Learning (TinyML) brings intelligence directly onto microcontrollers and low-power devices. It allows models to run locally in real time without relying on the cloud. That means lower latency, less cost, and stronger privacy.

A wide range of studies have explored both static and dynamic SLR under embedded or low-power constraints. Table 1 summarizes major contributions across static image recognition, sensor-based classification, and dynamic gesture modeling. These works collectively show the feasibility of CNN- and RNN-based architectures under TinyML toolchains while exposing persistent challenges in model quantization, operator support, and dataset standardization.

Table 1. Summary of TinyML and edge AI-related works in SLR and embedded ML.

Paper	Static/Dynamic	Dataset	Model	MCU/Toolchain
[39]	Static	Custom glove sensor dataset	CNN-based TinyML model	Raspberry Pi Pico
[40]	Static (ASL digits)	Custom flex-sensor dataset	CNN, KNN, Naïve Bayes, Decision Tree	Arduino Uno
[41]	Static (ASL digits 0–9)	64,000 RGB images	CNN + Residual/Dense Blocks, MobileNet V3	Raspberry Pi/TFLite
[42]	Static (ASL alphabet)	Kaggle ASL A–Z dataset	TinyML CNN classifier on CSV images	MCU (unspecified)
[43]	Static (ASL gestures)	Custom vision dataset	MobileNet V1–V3 (transfer learning)	STM32F746G (Cortex-M7)/TFLM vs. STM32Cube.AI
[44]	Static (images)	Underwater sensor dataset	MobileNet, MobileNet V2, Custom Tiny CNN	ARM Cortex-M4/TFLM
[15]	Dynamic (ISL months)	Custom Indian SL (video)	Mediapipe keypoints + e-LSTM	Raspberry Pi 4/TFLite (INT8)
[45]	Dynamic (sensor HAR)	8 public HAR datasets	RNN, LSTM, GRU, FGRNN, FRNN, UGRNN	Raspberry Pi/TFLM
[35]	Conceptual review	–	–	TFLM, NNoM, Edge Impulse, µTVM
[46]	TinyML overview	–	–	Raspberry Pi Pico, ESP32, Arduino Nano 33 BLE Sense

Edge Metrics and Evaluation

TinyML deployment emphasizes multi-objective metrics that differ from those used in conventional cloud benchmarks. Beyond accuracy, system-level evaluations now include:

- Inference latency (ms or FPS): end-to-end response time; real-time thresholds typically range between 15–30 FPS for gesture or sign recognition [34].
- Energy per inference (mJ or μ W): total power consumption; sub-milliwatt operation is desired for continuous wearable or mobile use [32,33].

2.5. Frameworks and Platforms for Edge AI

The success of TinyML depends on optimized toolchains that bridge ML frameworks and embedded hardware. Ref. [35] surveys the leading deployment solutions: TensorFlow Lite for microcontrollers (TFLM), NNoM, Edge Impulse, and μ TVM, with CMSIS-NN as the underlying kernel library accelerating core operations on Cortex-M cores. TFLM remains the dominant open-source runtime for portable INT8 inference, though it lacks native support for GRU and has limited profiling tools. NNoM compiles networks to plain C with full RNN support and CMSIS-NN integration, at the cost of a smaller community. Edge Impulse provides an end-to-end AutoML pipeline whose EON compiler reduces SRAM usage by 25–55% but sacrifices transparency and portability. From a hardware perspective, ref. [46] catalogs the leading embedded vision boards used in TinyML research and identifies the battery chemistries most commonly used to power them. These frameworks and platforms constitute the first generation of deployable edge AI infrastructure, supporting the migration of SLR models from GPU servers to energy-constrained microcontrollers.

2.6. Hardware-Aware Optimization

Deep learning on embedded devices demands models tailored to specific hardware constraints. Hardware-aware optimization combines architectural profiling, topology search, and accelerator tuning to balance accuracy, latency, and energy [31,32,47]. Performance evaluation in this setting extends beyond accuracy to include MAC counts and GOPS as primary measures of computational workload. For reference, AlexNet requires 1.4 GOPS and ResNet-152 requires 22.6 GOPS [32].

2.7. Energy Efficiency and Privacy

Running sign language recognition on low-power edge devices requires balancing accuracy, energy consumption, and data protection. On-device inference eliminates the need to transmit raw video to external servers, providing strong privacy guarantees by design. Recent work in neuromorphic computing, quantized neural processing, and physical reservoir computing points toward further reductions in inference energy [48,49].

2.8. Incremental Learning for Expanding Sign Vocabularies

A practical limitation of static SLR models is their inability to incorporate new signs without full retraining. Incremental and continual learning address this by enabling models to extend their vocabulary without catastrophic forgetting. Ref. [50] compares incremental learning strategies for a skeleton- and transformer-based isolated SLR model on Peruvian Sign Language, demonstrating that vocabulary expansion is feasible without discarding previously learned representations.

2.9. Datasets and Benchmarks in SLR

Robust datasets are essential for evaluating deep learning models. Ref. [51] combines RGB video, IMU, and sEMG signals in a multimodal continuous SLR dataset of 765 samples across nine activity types. Ref. [8] evaluates on RWTH, RWTHT, and CSL-Daily (over 25,000 samples combined) using Word Error Rate as the principal metric. At the word level, Ref. [24] employs MS-ASL to address signer diversity and background variation, achieving 92.35% top-1 accuracy. Ref. [52] introduces the Sem-Lex Benchmark, 84,000 videos from native ASL signers aligned with ASL-LEX and SignBank, reaching 85% phonological feature recognition with SL-GCN, and identifying demographic bias as a persistent limitation. Ref. [9] evaluates across LSA64, AUTSL, and WLASL, while ref. [53] establishes WLASL as the largest public ASL corpus (2000 glosses, 100+ signers), enabling scalable benchmarking for both pose-based and appearance-based models. Quantization calibration quality has also received attention. The SelectQ method [54] improves 4-bit ResNet-18 accuracy on ImageNet by over 15% through activation-clustering-based calibration sample selection, underscoring that robust quantization requires careful data curation, not only numerical schemes.

Ethical and Reproducibility Considerations

Ethical research with DHH participants requires genuine partnership rather than observation. Deaf communities are close-knit and individuals can be identified even without explicit labeling, making careful data governance and informed consent essential at every stage of collection, processing, and sharing [55–57]. Signed video explanations and QR-coded consent forms help ensure participants understand and remain in control of their data. Reproducibility is further complicated by inconsistent annotation. In WLASL2000, some gloss labels carry translation errors [58], and most datasets label at the video level without frame-by-frame keypoint ground truth, making it difficult to separate pose estimation errors from genuine motion variability [58,59]. Most benchmarks also cover isolated signs or narrow-domain sentences, missing the contextual and dialogic structure of real communication [60].

2.10. Identified Gaps and Design Opportunities

This review shows clear gaps and missed design opportunities for low-power edge deployment. Below we synthesize the main weaknesses found across the literature:

- **Overreliance on heavy preprocessing pipelines:** Many SLR systems depend on full-frame 3D CNNs and pose-estimation toolkits (MediaPipe, OpenPose) that assume abundant CPU/GPU resources and incur costly per-frame preprocessing, making them unsuitable for microcontrollers.
- **Limited vocabulary size and non-representative datasets:** Most TinyML-oriented SLR studies restrict classification to only a few dozen signs and rely on small, lab-controlled, self-collected datasets. This combination limits real-world applicability, hinders reproducibility, and obscures true generalization performance, while large-vocabulary word-level recognition under realistic, open-source benchmarks remains largely unexplored under MCU constraints.
- **Dynamic sign language recognition underexplored in TinyML:** Word-level and continuous SLR remain largely unexplored in the context of low-power deployment. Most TinyML work focuses on static, isolated gesture recognition.
- **Fragmented toolchains and poor cross-hardware portability:** Toolchains, such as TFLM, TinyEngine, NNOM, Edge Impulse, and vendor SDKs exhibit inconsistent serialization formats and missing operators. This means each toolchain uses its own way of saving and representing models, increasing friction and hurting reproducibility.

- Multimodal fusion cost: While multimodal fusion improves accuracy in laboratory settings, it significantly increases compute and memory requirements.
- Sparse field deployments and user studies: Few works include real-user deployments that evaluate robustness across signer diversity (skin tone, clothing, background) or assess social acceptability on practical devices.

Gaps Handled in Our Research

In this work, we address several of the identified gaps directly through targeted design and evaluation choices:

- Eliminating heavy preprocessing pipelines: We remove dependence on pose-estimation frameworks and full-frame 3D CNN stacks.
- Architectures designed for TinyML constraints: we introduce a compact, lightweight architecture that preserves temporal modeling under strict parameter, memory, and quantization constraints.
- Large-vocabulary word-level recognition: We explicitly target a 100-class vocabulary, addressing the underexplored challenge of scaling TinyML-based SLR beyond a few dozen signs and moving closer to real-world linguistic requirements.
- Cross-benchmark robustness assessment: To counter dataset fragmentation and limited reproducibility, we evaluate on open-source, large-scale benchmarks (WLASL and the SemLex benchmark), exposing generalization gaps that are hidden in small, lab-curated datasets.
- Realistic pre-deployment assessment: Rather than claiming deployment readiness without validation, we conduct a systematic pre-deployment hardware assessment. This includes operator-level compatibility analysis, quantization behavior, latency, memory footprint, and energy proxy under microcontroller-like constraints. This bridges the gap between algorithmic evaluation and practical embedded feasibility.

Taken together, the literature makes one point clear: future SLR systems must balance linguistic expressiveness with computational efficiency and must be designed with hardware constraints in mind from the start. This understanding shapes the methodology developed in this work.

3. Proposed Method

3.1. Problem Formulation

This work investigates a lightweight spatio-temporal architecture designed for efficient word-level sign language recognition on low-power hardware.

We consider a video classification problem where each input sample is a sequence of frames

$$X = \{X_t\}_{t=1}^T, \quad X_t \in \mathbb{R}^{H \times W \times C},$$

and the goal is to learn a mapping

$$f_\theta : \mathbb{R}^{T \times H \times W \times C} \rightarrow \Delta^{K-1}. \quad (1)$$

where K denotes the number of target classes. In sign language recognition, K may grow substantially as vocabularies expand, making scalability with respect to K and T a central design constraint.

3.2. Model Architecture

The S3D-Conv1D architecture was inspired by the separable 3D architecture proposed by Xie et al. [14]. It factorizes 3D convolution into separate spatial and temporal operations to reduce computation and improve efficiency.

This subsection presents a mathematical formulation of the proposed S3D-Conv1D architecture, directly grounded in its intuitive operation on video data. Each video is treated as an ordered sequence of individual frames, and spatial and temporal reasoning are deliberately decoupled. This modular design enables clear interpretability, linear-time temporal processing, and predictable memory usage, all essential for scalable and deployable sign language recognition.

Table A1 summarizes the mathematical notation used throughout the paper.

As shown in Figure 2 and described in Algorithm 1, rather than employing full 3D convolutions or recurrent models, S3D-Conv1D factorizes the problem into:

1. A frame-wise spatial encoder based on Conv2D operators;
2. A temporal encoder based on a single Conv1D layer followed by global temporal pooling.

Frames are processed independently in the spatial domain before being aggregated temporally. This architectural decision directly shapes the mathematical structure of the model, the form of its parameterization, and its computational scaling behavior.

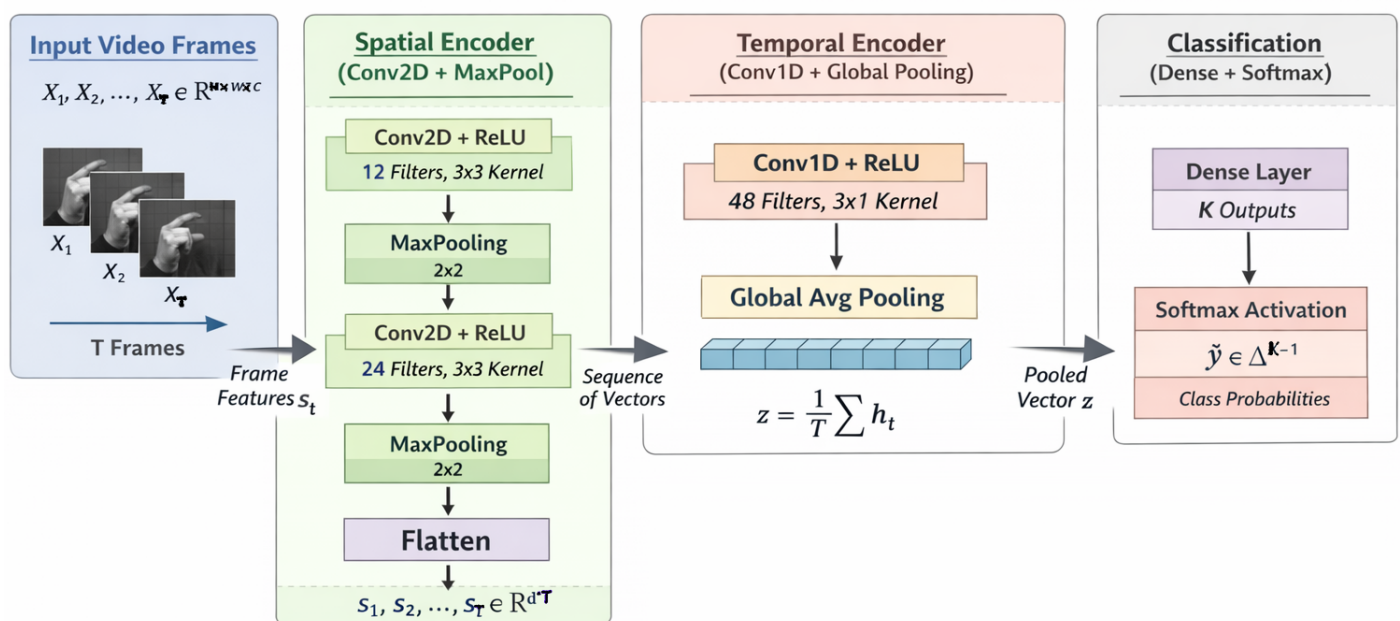


Figure 2. S3D-Conv1D workflow diagram.

Algorithm 1 S3D-Conv1D Architecture for Sign Language Recognition

Require: Video sequence $X = \{X_t\}_{t=1}^T$, with $X_t \in \mathbb{R}^{H \times W \times C}$; number of classes K

Ensure: Predicted class probabilities $\hat{y} \in \Delta^{K-1}$

Define mapping:

$$f_\theta : \mathbb{R}^{T \times H \times W \times C} \rightarrow \Delta^{K-1}$$

Spatial encoder parameters:

$W^{(1)} \in \mathbb{R}^{3 \times 3 \times C \times 12}$ (Conv2D, stride 1, padding “same”, activation ReLU)

$W^{(2)} \in \mathbb{R}^{3 \times 3 \times 12 \times 24}$ (Conv2D, stride 1, padding “same”, activation ReLU)

MaxPooling layers with window 2×2

Temporal encoder parameters:

$W^{(3)} \in \mathbb{R}^{k \times d \times 48}$ (Conv1D, stride 1, padding “same”, activation ReLU)

Classifier parameters:

$W^{(4)} \in \mathbb{R}^{d' \times K}$ (Dense layer, activation Softmax)

for $t = 1$ to T **do**

Frame-wise spatial encoding:

$$f_t^{(1)} = \text{MaxPool}_{2 \times 2}(\text{ReLU}(W^{(1)} * X_t + b^{(1)}))$$

$$f_t^{(2)} = \text{MaxPool}_{2 \times 2}(\text{ReLU}(W^{(2)} * f_t^{(1)} + b^{(2)}))$$

Flatten features:

$$s_t = \text{Flatten}(f_t^{(2)}), \quad s_t \in \mathbb{R}^d$$

end for

Construct temporal sequence:

$$S = [s_1, s_2, \dots, s_T] \in \mathbb{R}^{T \times d}$$

Temporal encoding:

$$h = \text{ReLU}(W^{(3)} * S + b^{(3)}), \quad h \in \mathbb{R}^{T \times d'}$$

Global temporal average pooling:

$$z = \frac{1}{T} \sum_{t=1}^T h_t, \quad z \in \mathbb{R}^{d'}$$

Dense layer:

$$y = W^{(4)} z + b^{(4)}, \quad y \in \mathbb{R}^K$$

Softmax activation:

$$\hat{y} = \text{softmax}(y), \quad \hat{y} \in \Delta^{K-1}$$

return $\hat{y}$

3.2.1. Spatial Encoding (Frame-Wise Conv2D Backbone)

Each frame is passed through a shared spatial encoder composed of two Conv2D-ReLU-MaxPooling blocks. Conceptually, this stage scans each frame using small sliding windows to detect salient patterns such as edges and hand contours.

First Conv2D Block. A 3×3 convolution with stride 1 and same padding is applied using 12 filters. The convolutional kernel examines 3×3 pixel neighborhoods at a time, ensuring fine-grained spatial coverage. Formally,

$$f_t^{(1)} = \text{MaxPool}_{2 \times 2}(\sigma(W^{(1)} * X_t + b^{(1)})), \quad (2)$$

where

$$W^{(1)} \in \mathbb{R}^{3 \times 3 \times C \times 12}, \quad b^{(1)} \in \mathbb{R}^{12},$$

and $\sigma(\cdot)$ denotes the ReLU activation. The subsequent 2×2 max pooling operation retains only the strongest activations in each local region, reducing spatial resolution while preserving dominant features:

$$f_t^{(1)} \in \mathbb{R}^{H/2 \times W/2 \times 12}.$$

Second Conv2D Block. The process is repeated with increased representational capacity using 24 filters:

$$f_t^{(2)} = \text{MaxPool}_{2 \times 2} \left(\sigma(W^{(2)} * f_t^{(1)} + b^{(2)}) \right), \quad (3)$$

with

$$W^{(2)} \in \mathbb{R}^{3 \times 3 \times 12 \times 24}, \quad b^{(2)} \in \mathbb{R}^{24},$$

yielding

$$f_t^{(2)} \in \mathbb{R}^{H/4 \times W/4 \times 24}.$$

3.2.2. Flattening and Spatial Embedding

The final spatial feature maps are flattened into a one-dimensional vector for each frame:

$$s_t = \text{Flatten}(f_t^{(2)}), \quad (4)$$

where

$$s_t \in \mathbb{R}^d, \quad d = H/4 \cdot W/4 \cdot 24.$$

This transformation converts each frame into a compact embedding vector, producing a temporal sequence

$$S = [s_1, s_2, \dots, s_T] \in \mathbb{R}^{T \times d}.$$

3.2.3. Temporal Modeling (Conv1D over Frame Embeddings)

Temporal dependencies are modeled by applying a one-dimensional convolution across the sequence of frame embeddings. With kernel size 3, stride 1, and same padding, the Conv1D layer captures short-range motion patterns:

$$h = \sigma(W^{(3)} * S + b^{(3)}), \quad (5)$$

where

$$W^{(3)} \in \mathbb{R}^{3 \times d \times d'}, \quad b^{(3)} \in \mathbb{R}^{d'}, \quad d' = 48,$$

and

$$h \in \mathbb{R}^{T \times d'}.$$

3.2.4. Global Temporal Pooling

To summarize the entire video sequence into a fixed-length representation, global average pooling is applied across time:

$$z = \text{GAP}_{\text{time}}(h) = \frac{1}{T} \sum_{t=1}^T h_t, \quad (6)$$

where

$$z \in \mathbb{R}^{d'}.$$

3.2.5. Classification (Dense Softmax Head)

Finally, the pooled temporal representation is mapped to class probabilities using a fully connected layer:

$$\hat{y} = \text{softmax}(W^{(4)}z + b^{(4)}), \quad (7)$$

where

$$W^{(4)} \in \mathbb{R}^{d' \times K}, \quad b^{(4)} \in \mathbb{R}^K, \quad \hat{y} \in \Delta^{K-1}.$$

This layer-wise formulation explicitly reflects how S3D-Conv1D processes videos: spatial features are extracted independently from each frame using small, consistent convolutional kernels, then aggregated temporally using a lightweight Conv1D operator and global averaging.

4. Measurable Quantities

This subsection analyzes the quantities that determine the deployability of the proposed S3D-Conv1D architecture on microcontroller-class hardware. In such constrained environments, feasibility is governed by three factors:

1. Parameter count, which defines Flash memory usage;
2. Activation footprint, which determines peak SRAM requirements;
3. Multiply–accumulate operations (MACs), which dominate inference latency and energy.

4.1. Parameters and Learnable Weights

The number of learnable parameters is determined by kernel dimensions and channel widths and is independent of the input sequence length T for convolutional and temporal layers.

4.1.1. Conv2D Layers

For a 2D convolution with kernel size $r \times r$, C_{in} input channels, and C_{out} output channels, the parameter count is

$$\text{Params}(\text{Conv2D}) = r^2 C_{\text{in}} C_{\text{out}} + C_{\text{out}}. \quad (8)$$

4.1.2. Temporal Conv1D Layer

The temporal one-dimensional convolution layer applies d' filters of width r over frame embeddings $s_t \in \mathbb{R}^d$, yielding

$$\text{Params}(\text{Conv1D}) = r d d' + d'. \quad (9)$$

4.1.3. Dense Classification Layer

The final classifier maps a pooled representation $z \in \mathbb{R}^{d'}$ to K classes, resulting in

$$\text{Params}(\text{Dense}) = d' K + K. \quad (10)$$

4.2. Activation Memory Footprint

Activation memory corresponds to intermediate tensors stored during inference. For a convolutional or pooling layer producing

$$\mathbf{A} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}},$$

the activation size is

$$|\mathbf{A}| = H_{\text{out}} W_{\text{out}} C_{\text{out}}. \quad (11)$$

In the spatial encoder, successive pooling halves the spatial resolution, ensuring that the peak activation occurs in early layers and is bounded by the fixed input resolution.

For the temporal one-dimensional convolution layer, activations have shape

$$\mathbf{A}_{1D} \in \mathbb{R}^{T \times d'},$$

with activation size

$$|\mathbf{A}_{1D}| = T d'. \quad (12)$$

Thus, activation memory scales linearly with T and is independent of the vocabulary size K . The classifier output of size K does not contribute to peak activation storage.

4.3. Multiply–Accumulate Operations (MACs)

Inference computation is measured using multiply–accumulate operations (MACs).

4.3.1. Conv2D MACs

For a two-dimensional convolution layer with kernel size $r \times r$, C_{in} input channels, C_{out} output channels, and output spatial resolution $(H_{\text{out}}, W_{\text{out}})$, each output element requires $r^2 C_{\text{in}}$ MACs. Summed over all output locations and channels, this yields

$$\text{MACs}(\text{Conv2D}) = r^2 C_{\text{in}} C_{\text{out}} H_{\text{out}} W_{\text{out}}. \quad (13)$$

4.3.2. Conv1D MACs

For the temporal one-dimensional convolution layer, each of the T temporal positions produces d' outputs, each requiring $r \cdot d$ MACs. The total temporal cost is therefore

$$\text{MACs}(\text{Conv1D}) = r d d' T. \quad (14)$$

4.3.3. Dense Layer MACs

The dense classifier computes K dot products of length d' , yielding

$$\text{MACs}(\text{Dense}) = d' K. \quad (15)$$

4.4. Numerical Cost Assessment

To assess practical inference cost, representative values are substituted into the derived expressions.

We consider $T = 24$ grayscale frames of size 64×64 and a vocabulary of $K = 100$ classes.

As presented in Table 2, the model contains 892,420 parameters, corresponding to a Flash footprint of approximately 900 KB under 8-bit integer (INT8) quantization. The largest activation tensor, with 49,152 elements, requires about 50 KB of RAM. With buffer reuse and runtime overheads, total RAM usage remains below 96–128 KB, consistent with mid-range microcontrollers.

The total inference cost is 95.6 M MACs, dominated by per-frame spatial convolutions and a lightweight temporal Conv1D backbone. The dense classifier contributes negligibly.

Table 2. Numerical cost and memory footprint of S3D-Conv1D under INT8 quantization.

Component	Parameters	Peak Activations	MACs
Conv2D-1	$3^2 \cdot 1 \cdot 12 + 12 = 120$	$64 \times 64 \times 12 = 49,152$	$3^2 \cdot 1 \cdot 12 \cdot 64^2 = 442,368$ per frame
Conv2D-2	$3^2 \cdot 12 \cdot 24 + 24 = 2616$	$32 \times 32 \times 24 = 24,576$	$3^2 \cdot 12 \cdot 24 \cdot 32^2 = 2,654,208$ per frame
Spatial Conv2D ($T = 24$)	2736	–	$24 \times (442,368 + 2,654,208) = 74,317,824$
Temporal Conv1D	$3 \cdot 6144 \cdot 48 + 48 = 884,784$	$24 \times 48 = 1152$	$3 \cdot 6144 \cdot 48 \cdot 24 = 21,233,664$
Dense	$48 \cdot 100 + 100 = 4900$	100	$48 \cdot 100 = 4800$
Total	892,420	49,152	95.6 million

5. Computational Complexity Analysis of S3D-Conv1D

This section analyzes the computational complexity of the proposed S3D-Conv1D model during inference. Classical complexity theory was developed for algorithms with input-dependent control flow, whereas neural network inference executes a fixed computation graph once per input. Nevertheless, complexity analysis remains necessary to assess deployability, scalability, and real-time feasibility, especially under embedded and microcontroller constraints.

Complexity is expressed using Big- $\mathcal{O}$ notation [61]. The analysis considers worst-case memory usage and worst-case arithmetic cost. It characterizes how inference time and memory requirements evolve as either the input sequence length T or the vocabulary size K increases.

Asymptotic analysis describes scalability but does not ensure feasibility under strict hardware limits. Numerical operation counts provide exact values but hide scaling trends. We therefore combine closed-form expressions for parameters, activations, and MACs with asymptotic analysis. This approach enables both reliable deployment validation and principled scalability assessment.

Space complexity refers to total memory usage, including parameters and intermediate buffers [62]. Time complexity measures the number of elementary operations required for inference and is expressed using Big- $\mathcal{O}$ notation to describe asymptotic growth [63].

5.1. Inference Model and Assumptions

The input is a video sequence

$$X = \{X_t\}_{t=1}^T, \quad X_t \in \mathbb{R}^{H \times W \times C},$$

processed by a trained S3D-Conv1D model using INT8 precision. Inference follows a fixed sequential pipeline. Spatial features are first extracted using Conv2D layers. Temporal dynamics are then modeled with a Conv1D layer. A final dense layer produces class scores. Each layer is executed exactly once per input, and execution does not depend on input content.

Worst-case analysis is used since embedded deployment requires strict upper bounds on memory usage and execution time.

5.2. Space Complexity

Memory usage during inference consists of three components:

- Model parameters, stored in non-volatile memory (Flash/ROM), including convolutional and classifier weights.
- Intermediate activations, stored in SRAM, including input tensors, feature maps, and temporary convolution buffers.
- Runtime overhead, including stack memory and bookkeeping buffers.

5.2.1. Model Parameters

All convolutional layers use fixed kernel sizes and channel counts. Their parameter size does not depend on the sequence length T . Only the final classifier depends on the number of classes K (see Section 4.1). Therefore,

$$\mathcal{O}_{\text{params}} = \mathcal{O}(K). \quad (16)$$

5.2.2. Activation Memory

Inference feasibility is determined by peak activation memory. Layers execute sequentially, and intermediate buffers are reused or released. The dominant activation terms are (see Section 4.2):

- Spatial feature maps of bounded size $\mathcal{O}(1)$.
- Temporal one-dimensional convolution activations of size $\mathcal{O}(T)$.

Classifier outputs of size $\mathcal{O}(K)$ are short-lived and non-dominant.

5.2.3. Worst-Case Space Complexity

Total inference memory combines parameter storage and peak activations:

$$\mathcal{O}_{\text{space}} = \mathcal{O}_{\text{params}} + \mathcal{O}_{\text{act}} = \mathcal{O}(K) + \mathcal{O}(T). \quad (17)$$

Thus,

$$\mathcal{O}_{\text{space}} = \mathcal{O}(T + K). \quad (18)$$

This bound corresponds to the maximum supported sequence length and vocabulary size. It provides a strict upper limit required for safe embedded deployment.

5.3. Time Complexity

Inference time is dominated by arithmetic operations, mainly multiply–accumulate (MAC) operations, formulated in Section 4.3. Since S3D-Conv1D is a feed-forward pipeline, the total cost is the sum of per-layer costs.

5.3.1. Spatial Processing

Each frame is processed independently by the Conv2D backbone. Let C_{spatial} denote the constant per-frame cost. The total spatial cost is

$$\text{MACS}_{\text{spatial}} = T \cdot C_{\text{spatial}}. \quad (19)$$

5.3.2. Temporal Modeling

The one-dimensional convolution layer operates along the temporal dimension with fixed kernel width and channel size. Its cost scales linearly with sequence length:

$$\text{MACS}_{\text{temporal}} = \mathcal{O}(T). \quad (20)$$

5.3.3. Classification

The final dense layer maps the embedding to K classes:

$$\text{MACS}_{\text{classifier}} = \mathcal{O}(K). \quad (21)$$

5.3.4. Total Inference Time

The total inference cost is the sum of feature extraction and classification costs:

$$\mathcal{O}_{\text{time}} = \mathcal{O}_{\text{feat}} + \mathcal{O}_{\text{cls}} = \mathcal{O}(T) + \mathcal{O}(K). \quad (22)$$

Hence,

$$\mathcal{O}_{\text{time}} = \mathcal{O}(T + K). \quad (23)$$

This additive form follows from sequential execution, with no nested recomputation between spatial and temporal stages. Worst-case timing is required for real-time systems, where latency bounds must hold for all valid inputs.

5.4. Discussion

Practical sign language systems must support vocabulary growth and variable input duration. In S3D-Conv1D, the vocabulary size K affects only the final classification layer, while spatial and temporal feature extraction remain unchanged. This separation bounds the cost of vocabulary expansion.

The temporal length T influences inference through repeated spatial processing and explicit temporal modeling. Each additional frame increases computation and intermediate activations linearly.

The resulting linear scaling in both time and space complexity is well suited to embedded platforms. By separating spatial feature extraction from explicit temporal modeling, S3D-Conv1D achieves predictable inference cost while preserving temporal expressiveness. This balance enables reliable real-time deployment under strict memory and latency constraints without sacrificing model capacity.

The absence of quadratic or state-dependent terms distinguishes S3D-Conv1D from recurrent and attention-based models. This property ensures predictable latency and bounded memory usage, which are critical for microcontroller deployment.

6. Experimental Evaluation

6.1. Datasets

The primary benchmark is a curated WLASL subset with 2181 videos across 100 classes and 119 signers [53]. Large-scale datasets like WLASL support generalization but are collected under heterogeneous recording conditions. Background, lighting, camera, and signer differences shift feature distributions across layers, and a model trained and calibrated exclusively on such data often fails to recognize signs from signers whose appearance, skin tone, or style is not well represented in the dataset. To address this, a small custom dataset, ASL-Kimpinde [64], was created with two roles: augmenting underrepresented WLASL classes and providing signer-specific calibration data for quantization. This illustrates a broader principle: signer variation is a genuine source of distribution shift, and domain-aware calibration is a necessary step when moving from benchmark evaluation to individual use. The ASL-Kimpinde scope (250 videos, 50 classes, and a single signer) is a recognized limitation. Greater signer diversity in both augmentation and calibration subsets would strengthen cross-signer generalization. More broadly, multimodal pipelines combining raw frames with lightweight keypoint representations, optimized for MCU inference, offer a path toward systems that generalize across signers without requiring signer-specific data.

After augmentation, the combined dataset totals 11,194 samples.

Generalization performance is evaluated with the SemLex (2023) benchmark dataset using its official release [52]. Only the curated training split is considered, comprising 1253 valid videos across 100 classes and totaling 7412 samples after augmentation. This restriction was chosen to keep computational complexity tractable and to balance dataset statistics with WLASL100, enabling fairer cross-dataset comparisons. We acknowledge that this choice makes direct comparison with prior work less straightforward but provides a controlled setting aligned with the scope of this study.

A unified preprocessing pipeline is applied to all datasets to ensure consistency and reproducibility. Videos are decoded using OpenCV, converted to grayscale, resized to 64×64 , normalized to the $[0, 1]$ range, and stored as .npy files. Grayscale conversion reduces memory usage and stabilizes activation ranges, which is beneficial for INT8 calibration. The NumPy format accelerates training and provides deterministic data access.

Video lengths vary between approximately 20 and 100 frames. All sequences are therefore aligned to a fixed temporal length of $T = 24$ frames. Longer videos are subsampled to retain key motion, while shorter ones are padded. This temporal normalization balances motion coverage with embedded memory constraints and ensures a consistent temporal receptive field during both training and inference.

Data augmentation is applied to address limited data and domain shift. Using Vidaug, each sequence undergoes lightweight spatial and temporal transformations, including random rotation up to 10° , Gaussian blur, and frame-rate resampling to $\{10, 15, 30, 50\}$ FPS. These transformations simulate viewpoint changes, low-quality imaging, and variable signing speeds commonly observed on real devices. All augmentations remain compatible with integer-only inference.

As shown in Table 3, the datasets were curated into training, validation, and test subsets with a fixed 70/20/10 split, and the table also reports the number of samples per subset obtained after augmentation. The official WLASL benchmark does not enforce a fully signer-disjoint split; our protocol follows this convention, meaning that reported accuracies reflect within-distribution performance. Cross-signer generalization is partially evaluated via SemLex100 (Section 6.5.2), which provides an entirely independent signing population and recording setup.

Algorithm 2 shows how Preprocessing converts raw videos into fixed-length, normalized sequences with temporal alignment, resolution control, and augmentation for robustness.

Algorithm 2 Dataset Preprocessing

Require: Video dataset $\mathcal{D} = \{(V_i, y_i)\}_{i=1}^N$, target length T , resolution (H, W)

Ensure: Preprocessed dataset $\tilde{\mathcal{D}}$

```

1: for each video  $V_i \in \mathcal{D}$  do
2:   Decode frames  $\{F_t\}_{t=1}^{T_i}$ 
3:   Resize to  $H \times W$ , convert to grayscale
4:   Normalize intensities:  $F_t \leftarrow F_t/255$ 
5:   Temporally resample frames
6:   Subsample or pad to length  $T$ 
7:   Apply augmentation  $\mathcal{A}(\cdot)$ 
8: end for
9: return  $\tilde{\mathcal{D}} = \{(X_i, y_i)\}$ 

```

Table 3. Dataset splits used in experiments (70/20/10 fixed split).

Dataset	Train	Val	Test	Total
WLASL100 + ASL-Kimpinde	7835	2239	1120	11,194
SemLex100 benchmark	5188	1482	742	7412
WLASL300 + ASL-Kimpinde	11,085	3167	1584	15,836

For each video, multiple variants are generated: the original sequence, a spatially augmented version, temporally resampled sequences at each target FPS, and their augmented counterparts. This strategy increases data diversity while preserving label semantics and temporal structure. Spatial transformations improve robustness to camera noise and

minor viewpoint changes, while temporal resampling explicitly trains the model to handle variable signing speeds, all while remaining lightweight and fully compatible with embedded deployment.

6.2. Training Procedure and Optimization

The S3D-Conv1D architecture trains under the following key settings. As presented in Table 4, the training configuration and hyperparameters include optimizer choice, learning rate, batch size, and number of epochs.

Table 4. Training configuration and hyperparameters.

Component	Setting
Optimizer	Adam
Base learning rate	$\eta = 1 \times 10^{-3}$
Loss function	Categorical cross-entropy with softmax
Evaluation metrics	Top-1/Top-5 accuracy, precision, recall, F1-score
Batch size	32
Training duration	Up to 100 epochs
Early stopping	Patience = 10 epochs
Learning rate schedule	ReduceLROnPlateau (factor 0.5, patience 5)
Checkpointing	Best model saved based on validation performance
Numerical precision	Full precision (mixed precision disabled)
Data pipeline	<code>tf.data</code> with shuffle, cache, batch, prefetch

These choices enabled stable convergence under the sparse, noisy gradients typical of word-level signing and the small per-class sample sizes in the datasets.

As presented in Algorithm 3, this procedure defines the core learning workflow of S3D-Conv1D.

Algorithm 3 Model Construction, Training, and Floating-Point Evaluation

Require: Preprocessed dataset $\tilde{D}$, epochs E , batch size B
Ensure: Optimized model parameters θ^* , evaluation metrics $\mathcal{M}$
 Define input tensor $X \in \mathbb{R}^{T \times H \times W \times C}$
 Build S3D-Conv1D architecture:
 Stratified split into training, validation, and test sets
 Initialize model parameters θ
for epoch = 1 to E **do**
 for each mini-batch $\mathcal{B}$ of size B **do**
 Forward pass through spatial encoder (per frame)
 Stack frame-wise features into temporal sequence
 Forward pass through Conv1D and temporal pooling
 Compute predictions and loss
 Update parameters using Adam optimizer
 end for
end for
 Evaluate model on test set
 Compute Top-1, Top-5 accuracy, precision, recall, and macro F1-score
return $\theta^*, \mathcal{M}$

The model converged in under 100 epochs (see Figure 3). This shows stable optimization and effective preprocessing. S3D-Conv1D also showed a stable learning curve, even though it trained both spatial and temporal kernels from scratch.

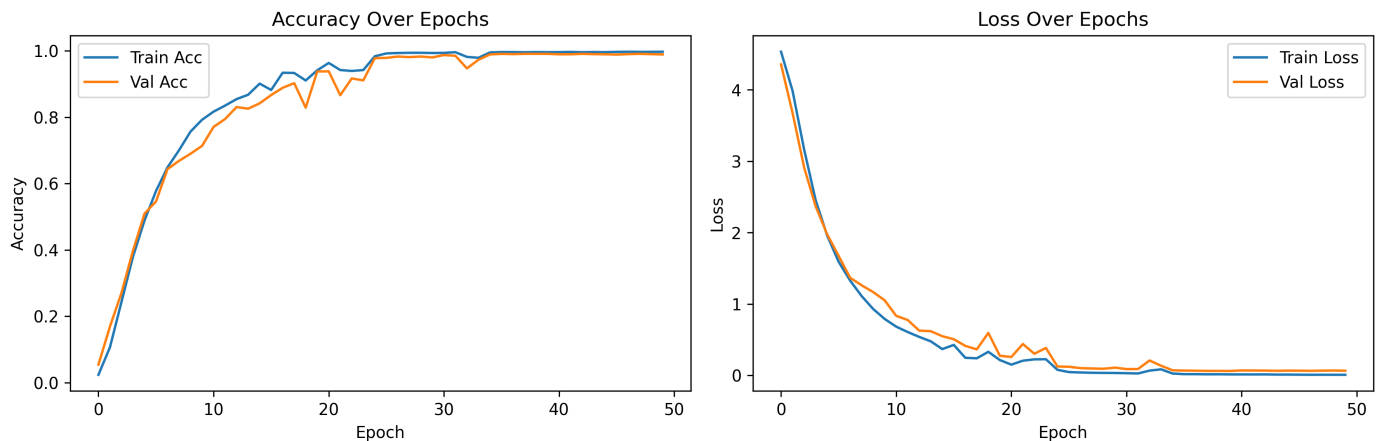


Figure 3. S3D-Conv1D Training Plots on WLASL100 + ASL-Kimpinde.

6.3. Quantization

Integer quantization reduces model size and inference cost by mapping floating-point weights and activations to 8-bit integers. This work applies two complementary strategies: post-training quantization (PTQ) and quantization-aware training (QAT), both targeting full INT8 representation with no floating-point fallback.

6.3.1. Quantization Formulation

Both strategies use affine INT8 quantization, mapping a floating-point value $x \in [x_{\min}, x_{\max}]$ to $x_q \in [-128, 127]$ via:

$$x_q = \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor + z, -128, 127\right), \quad s = \frac{x_{\max} - x_{\min}}{255}, \quad z = \text{clip}\left(\left\lfloor -\frac{x_{\min}}{s} \right\rfloor, -128, 127\right) \quad (24)$$

where s is the scale and z the zero-point. The quantization error is bounded by $|x - s(x_q - z)| \leq s/2$, so accurate range estimation is critical: an overly wide range increases the step size and reduces precision, while an overly narrow range causes clipping.

The workflow in Figure 4 highlights two complementary strategies: post-training quantization (PTQ) and quantization-aware training (QAT), which are detailed in the following subsections.

6.3.2. Post-Training Quantization

PTQ converts a trained FP32 model to INT8 in a single calibration pass with no weight updates. A representative calibration set $\mathcal{R}$ is passed through the model to record per-layer activation ranges, from which s and z are computed. All weights, activations, inputs, and outputs are then re-exported as INT8 using the TFLite converter with TFLITE_BUILTINS_INT8 as the sole operator set. Calibration is performed on the training split of the combined WLASL and ASL-Kimpinde dataset.

As detailed in Algorithm 4, the post-training quantization procedure transforms a trained FP32 model into its INT8 form.

Quantization Workflow: PTQ vs. QAT

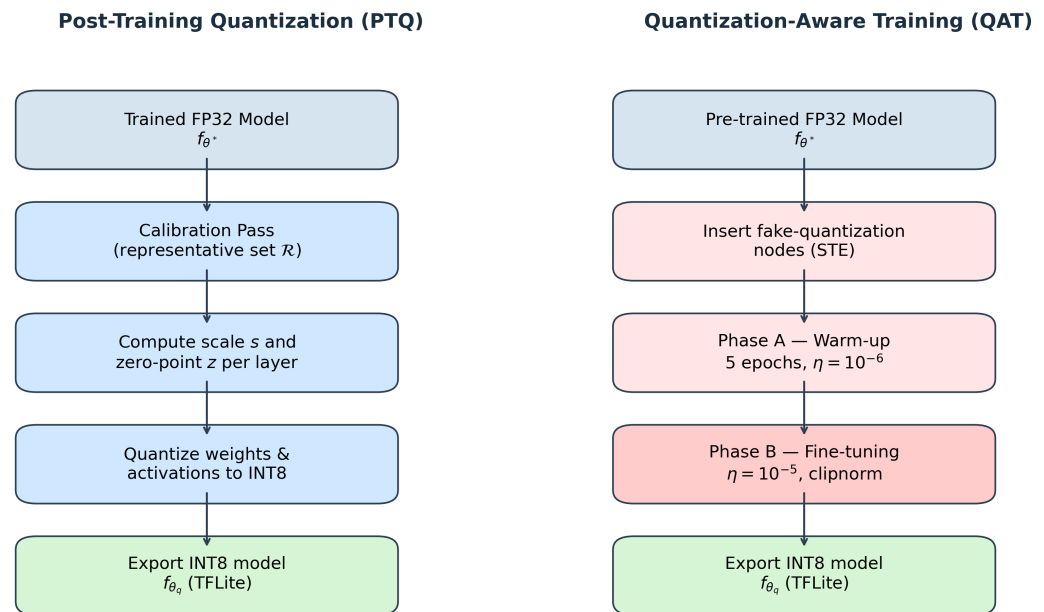


Figure 4. Quantization workflow comparing (PTQ) and (QAT).

Algorithm 4 Post-Training Quantization and Deployment-Aware Evaluation

Require: Trained FP32 model f_{θ^*} , calibration set $\mathcal{R}$, test set $\mathcal{T}$

Ensure: INT8 model f_{θ_q} , quantized evaluation metrics

- 1: Export trained model to TensorFlow Lite format
 - 2: Apply affine INT8 post-training quantization using $\mathcal{R}$
 - 3: Restrict operators to TFLite INT8 built-ins
 - 4: Export quantized model f_{θ_q}
 - 5: Evaluate f_{θ_q} on $\mathcal{T}$; Compute accuracy and macro F1-score
 - 6: **return** f_{θ_q} , quantized evaluation metrics
-

6.3.3. Quantization-Aware Training

QAT inserts fake-quantization nodes into the computation graph during fine-tuning, simulating INT8 rounding in the forward pass via the straight-through estimator [65]. Activation ranges are tracked as exponential moving averages over training batches, yielding more accurate scale estimates than the single-pass calibration of PTQ. A two-phase schedule is used: a 5-epoch warm-up at $\eta = 10^{-6}$ to stabilize the range trackers, followed by fine-tuning at $\eta = 10^{-5}$ with gradient clipping ($\|\nabla\|_2 \leq 1.0$) and early stopping on the validation set.

As described in Algorithm 5, the quantization-aware training procedure incorporates simulated INT8 operations during fine-tuning, enabling the model to adapt to quantization effects while maintaining numerical robustness.

Algorithm 5 Quantization-Aware Training (QAT)**Require:** Pre-trained FP32 model f_{θ^*} , training set $\mathcal{D}$, validation set $\mathcal{V}$, test set $\mathcal{T}$ **Ensure:** Fine-tuned INT8-equivalent model $f_{\tilde{\theta}}$, deployment metrics

- 1: Insert fake-quantization nodes into f_{θ^*} via `quantize_apply()`
- 2: Initialize per-layer scale trackers from FP32 weight ranges
- 3: **Phase A—warm-up:** fine-tune 5 epochs at $\eta = 10^{-6}$ (trackers stabilize)
- 4: **Phase B—main:** fine-tune up to 50 epochs at $\eta = 10^{-5}$, `clipnorm` = 1.0, early stopping on $\mathcal{V}$
- 5: Convert $f_{\tilde{\theta}}$ to TFLite INT8 via representative dataset
- 6: Export quantized model f_{θ_q}
- 7: Evaluate f_{θ_q} on $\mathcal{T}$; compute accuracy, macro F1
- 8: **return** f_{θ_q} , quantized evaluation metrics

6.4. Ablation Study

We systematically evaluate three design dimensions: (i) temporal Conv1D kernel size $k \in \{1, 3, 5, 7\}$, (ii) spatial block (Conv2D vs. DepthwiseConv2D), and (iii) temporal resolution $T \in \{8, 16, 24\}$. All experiments use post-training INT8 quantization (PTQ) and are reported as mean $\pm$ std over three runs. The baseline is $k = 3$, $T = 24$, Conv2D.

6.4.1. Effect of Temporal Kernel Size k

The kernel size k controls the local temporal receptive field of the Conv1D layer. A larger k capture of motion patterns over more consecutive frames but this also increases the number of parameters and enlarges the range of activations, which can make INT8 quantization unstable. Tables 5 and 6 report the results.

Table 5. Effect of kernel size k on accuracy ($T = 24$, PTQ INT8, mean $\pm$ std, 3 runs). WLASL100 + ASL-Kiminde dataset.

k	Dataset	Top-1	Top-3	Top-5	Macro F1	INT8 Acc.
1	WLASL100	0.991 $\pm$ 0.003	0.998 $\pm$ 0.004	0.998 $\pm$ 0.003	0.99 $\pm$ 0.01	98.51 $\pm$ 0.10%
3	WLASL100	0.985 $\pm$ 0.001	0.996 $\pm$ 0.006	0.997 $\pm$ 0.006	0.98 $\pm$ 0.01	98.60 $\pm$ 0.97%
5	WLASL100	0.968 $\pm$ 0.014	0.987 $\pm$ 0.006	0.991 $\pm$ 0.004	0.97 $\pm$ 0.01	88.15 $\pm$ 11.47%
7	WLASL100	0.986 $\pm$ 0.001	0.996 $\pm$ 0.002	0.997 $\pm$ 0.001	0.99 $\pm$ 0.01	98.15 $\pm$ 0.64%
1	WLASL300	0.671 $\pm$ 0.160	0.771 $\pm$ 0.171	0.819 $\pm$ 0.060	0.66 $\pm$ 0.07	38.15 $\pm$ 13.95%
3	WLASL300	0.868 $\pm$ 0.038	0.920 $\pm$ 0.024	0.933 $\pm$ 0.017	0.87 $\pm$ 0.04	77.09 $\pm$ 8.33%
5	WLASL300	0.766 $\pm$ 0.022	0.851 $\pm$ 0.027	0.876 $\pm$ 0.026	0.76 $\pm$ 0.03	53.36 $\pm$ 11.02%
7	WLASL300	0.843 $\pm$ 0.042	0.909 $\pm$ 0.024	0.932 $\pm$ 0.013	0.84 $\pm$ 0.04	75.86 $\pm$ 6.26%

Table 6. Hardware efficiency across kernel sizes ($T = 24$, PTQ INT8).

k	Parameters	Size (KB)	FLOPs (M)
1	302,596	394.2	162.8
3	892,420	883.1	191.1
5	1,482,244	1546.2	219.4
7	2,072,068	2122.2	247.7

Three effects of k are clearly observable. First, on WLASL-100 the task is largely saturated regardless of kernel size; all configurations except $k = 5$ achieve above 98% INT8 accuracy, confirming that short-range spatial features dominate recognition at this scale. Second, $k = 5$ shows unstable PTQ INT8: accuracy spans 75.7% to 98.4% across three runs, with a standard deviation (SD) = 11.47%. Notably, $k = 7$ does not exhibit this behavior (SD = 0.64%), so the instability is not a simple function of kernel size. Third, on WLASL300

the kernel size affects both accuracy and stability more strongly: $k = 3$ achieves the best Top-1 (0.868) and the most stable INT8 behavior, while $k = 1$ collapses to 38.15% INT8 accuracy. This confirms that temporal context is necessary at larger vocabulary scales. $k = 7$ approaches $k = 3$ accuracy but has much higher memory and computational costs (Table 6). $k = 3$ is therefore the optimal choice across accuracy, quantization stability, and computational efficiency.

6.4.2. Effect of Vocabulary Size K

Moving from $K = 100$ to $K = 300$ introduces two compounding challenges. First, The dense layer grows from 4900 to 14,700 parameters (see Table 2). Second, the semantic overlap between classes increases, as WLASL300 contains more signs with similar hand configurations and trajectories that are trivially separable at 100 classes but not at 300 (Figure 5). This explains the consistent pattern observed across all ablation dimensions: every configuration achieves near-saturated performance on WLASL100 but shows substantially higher variance and lower INT8 accuracy on WLASL300.

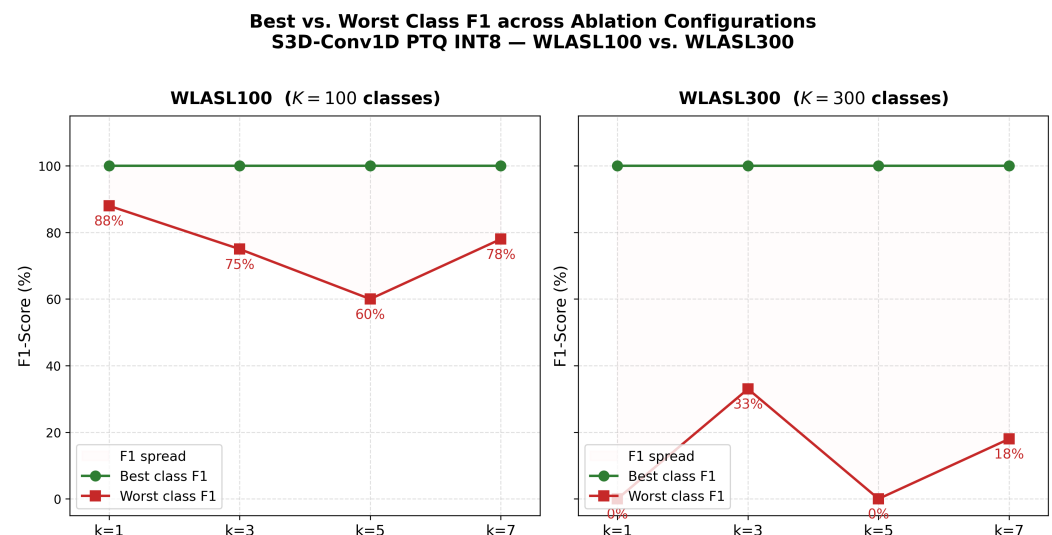


Figure 5. Best vs. Worst Class F1 across Ablation Configurations, PTQ INT8: WLASL100 vs. WLASL300.

6.4.3. Effect of Temporal Resolution T

The temporal resolution T determines how many consecutive frames are presented to the model. A larger T provides more motion context but increases input memory, and FLOPs linearly. Table 7 reports results for $T \in \{8, 16, 24\}$.

Table 7. Effect of temporal resolution T on accuracy and efficiency (PTQ INT8, mean $\pm$ std, 3 runs).

T	Dataset	Top-1	Top-3	INT8 Acc.	FLOPs (M)	Size (KB)
24	WLASL100	0.985 ± 0.025	0.996 ± 0.003	$98.60 \pm 0.97\%$	191.1	883.1
16	WLASL100	0.984 ± 0.005	0.994 ± 0.005	$98.09 \pm 0.52\%$	127.4	940.5
8	WLASL100	0.979 ± 0.003	0.995 ± 0.001	$96.22 \pm 1.12\%$	63.7	910.8
24	WLASL300	0.868 ± 0.038	0.920 ± 0.024	$77.09 \pm 8.33\%$	191.1	883.1
16	WLASL300	0.755 ± 0.069	0.842 ± 0.038	$54.97 \pm 9.24\%$	127.4	940.5
8	WLASL300	0.811 ± 0.061	0.886 ± 0.033	$70.67 \pm 9.85\%$	63.7	910.8

On WLASL100, reducing T from 24 to 8 cuts FLOPs by $3\times$, with only a modest drop in INT8 accuracy. This confirms that spatial features dominate recognition at 100 classes and that very short temporal windows are sufficient. On WLASL300, $T = 24$ achieves the best INT8 accuracy (77.09%) and the most stable results across runs. $T = 16$ and $T = 8$ both show higher variance under PTQ, with no consistent ordering between them.

6.4.4. Effect of Quantization Strategy: PTQ vs. QAT

Post-training quantization (PTQ) estimates activation ranges from the calibration set after training; quantization-aware training (QAT) inserts fake-quantization nodes during fine-tuning, constraining weight distributions to quantization-friendly ranges. Table 8 compares both strategies.

Table 8. PTQ vs. QAT for S3D-Conv1D ($k = 3$, $T = 24$, mean $\pm$ std, 3 runs).

Method	Dataset	FP32 Acc.	INT8 Acc.	Δ Acc.	INT8 Size
PTQ	WLASL100	98.96 $\pm$ 0.36%	98.60 $\pm$ 0.97%	−0.36 $\pm$ 0.62%	883 KB
QAT	WLASL100	98.42 $\pm$ 0.37%	98.10 $\pm$ 0.65%	−0.32 $\pm$ 0.32%	880 KB
PTQ	WLASL300	85.67 $\pm$ 3.60%	67.76 $\pm$ 8.43%	−17.91 $\pm$ 10.21%	898 KB
QAT	WLASL300	88.16 $\pm$ 0.27%	86.94 $\pm$ 0.48%	−1.22 $\pm$ 0.12%	890 KB

On WLASL100, PTQ and QAT are statistically equivalent: both achieve ≈ 98.1 – 98.6% INT8 accuracy with negligible and stable accuracy drop. The architecture’s operator homogeneity makes it PTQ-stable at this scale. On WLASL300, the two strategies diverge substantially. PTQ suffers a mean accuracy loss of $17.91 \pm 10.21\%$, with INT8 accuracy ranging from 60.2% to 76.8% across runs, a spread of 16.6% indicating that calibration-set-derived thresholds are unreliable at larger vocabulary scales. QAT nearly eliminates this gap (-1.22%), confirming that fine-tuning with quantization-aware gradients effectively constrains weight ranges for WLASL300. The quantization parameters in Table 9 confirm that QAT produces tight, symmetric weight distributions across all layers, consistent with low quantization error.

Table 9. S3D-Conv1D QAT kernel quantization parameters (WLASL100, run 1).

Layer	Shape	$x_{\min}$	$x_{\max}$	Scale	Zero-Point
conv2d_1	(3, 3, 1, 12)	−0.683	1.286	0.00772	88
conv2d_2	(3, 3, 12, 24)	−1.481	1.087	0.01007	127
temporal_conv1d	(3, 6144, 48)	−1.203	1.172	0.00931	127
classifier	(48, 100)	−1.617	0.429	0.00802	127

6.4.5. DepthwiseConv2D

Figure 6 compares Conv2D and DepthwiseConv2D spatial blocks in S3D-Conv1D. DepthwiseConv2D achieves large efficiency gains with negligible accuracy loss at 100 classes, but the reduced spatial capacity becomes limiting at 300 classes.

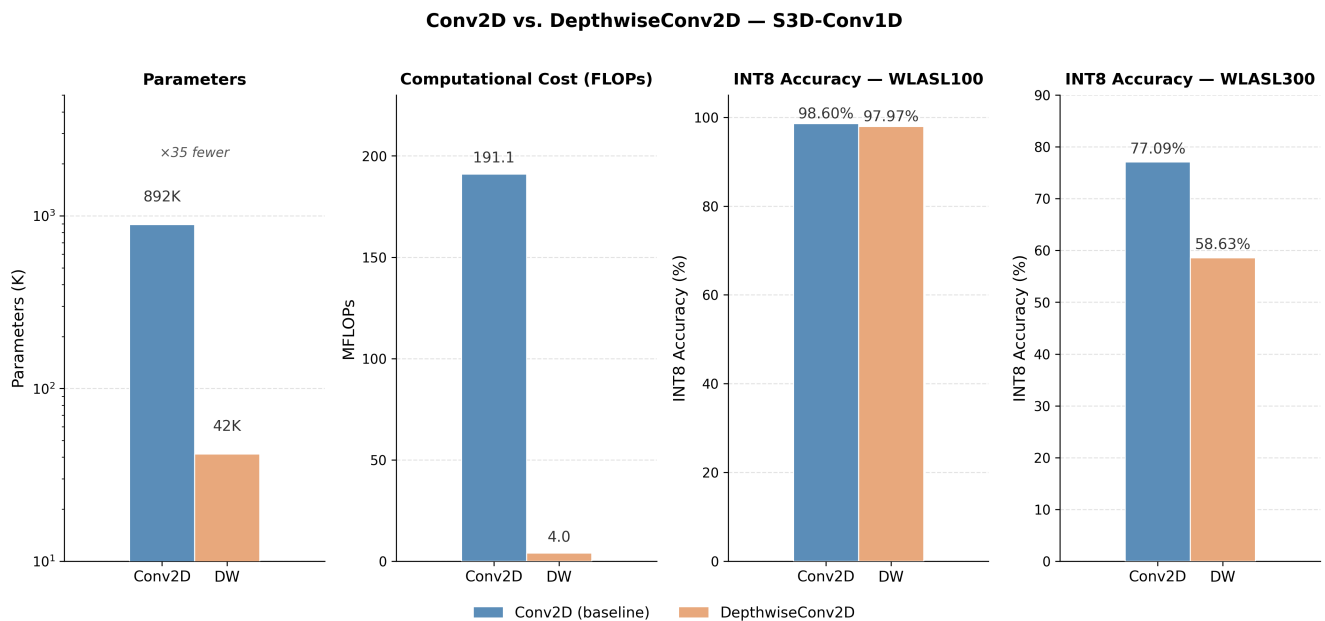


Figure 6. Conv2D vs. DepthwiseConv2D spatial blocks in S3D-Conv1D (PTQ, $k = 3$, $T = 24$, 3-run mean $\pm$ std).

6.4.6. Ultralight Variant

Combining $T = 8$, $k = 1$, DepthwiseConv2D, and QAT yields the ultralight configuration (Table 10). It is the most compact design evaluated, showing negligible accuracy loss at 100 classes and a clear efficiency–accuracy trade-off at 300 classes.

Table 10. Ultralight S3D-Conv1D configuration ($T = 8$, $k = 1$, DepthwiseConv2D, QAT INT8). All values are mean $\pm$ std over 3 independent runs.

Metric	WLASL100	WLASL300
FP32 Accuracy	98.90 $\pm$ 0.55%	77.22 $\pm$ 4.00%
QAT INT8 Accuracy	98.51 $\pm$ 0.89%	74.11 $\pm$ 5.16%
Accuracy Drop	−0.39 $\pm$ 0.36%	−3.11 $\pm$ 1.73%
INT8 Size	24.7 KB	34.9 KB
FLOPs	0.94 M	0.96 M

6.4.7. Ablation Summary

Across all ablation dimensions, the baseline configuration ($k = 3$, $T = 24$, Conv2D) occupies the most robust operating point: it achieves the best Top-1 on WLASL300, the most stable INT8 quantization across runs and kernel sizes, and the best cross-scale generalization. On WLASL100 alone, $k = 1$ achieves marginally higher accuracy and the ultralight variant offers a compelling efficiency trade-off at under 35 KB; however, neither generalizes as reliably to WLASL300 as the baseline.

6.5. Comparative Evaluation with Baseline Architectures

Four baseline architectures are evaluated alongside S3D-Conv1D. S3D [14] is included as the reference separable spatiotemporal model, factorizing 3D convolutions into spatial and temporal components for video understanding. CNN + RNN represents the dominant paradigm in sign language recognition, combining a convolutional spatial encoder with a recurrent temporal module (GRU). The e-LSTM [15] is the closest prior work to our scope, an attention-based LSTM architecture explicitly designed for edge deployment on embedded devices. MobileNet Variant uses a pretrained MobileNetV2 backbone for

spatial feature extraction, exploiting transfer learning, followed by temporal global average pooling and a dense classifier. All models are trained and evaluated under identical preprocessing, training, and evaluation conditions (Section 6.1).

6.5.1. Float32 Recognition Performance

S3D, CNN + RNN, the MobileNet variant, and S3D-Conv1D all achieve above 98% on WLASL100, confirming that the 100-class task is largely saturated under this dataset and preprocessing protocol. The discriminating dimension is SemLex100. CNN + RNN generalizes best (89.6%), benefiting from recurrent temporal modeling. S3D-Conv1D achieves 82.5%, outperforming both S3D (64.3%) and e-LSTM (59.8%). The MobileNet variant achieves the highest SemLex100 accuracy (97.6%), which reflects the strong transfer learning capacity of the pretrained MobileNetV2 backbone. As summarized in Table 11, the float32 accuracy and macro F1 scores across datasets highlight the comparative strengths of each architecture.

Table 11. Float 32 accuracy and macro F1 across datasets (mean $\pm$ std).

Model	WLASL100			SemLex100		
	Acc. (%)	Prec.	F1	Acc. (%)	Prec.	F1
S3D	99.29 $\pm$ 0.15	0.99	0.99	64.29 $\pm$ 1.50	0.78	0.64
CNN + RNN	99.29 $\pm$ 0.10	0.99	0.99	89.62 $\pm$ 1.00	0.91	0.89
e-LSTM	44.20 $\pm$ 2.50	0.75	0.53	59.78 $\pm$ 2.00	0.75	0.64
MobileNet variant	99.46 $\pm$ 0.03	1.00	0.99	97.57 $\pm$ 0.13	0.98	0.98
S3D-Conv1D	98.96 $\pm$ 0.36	0.99	0.98	82.48 $\pm$ 1.50	0.84	0.82

6.5.2. Cross-Dataset Generalization

S3D drops more than 35 pp from WLASL100 to SemLex100, indicating sensitivity to dataset-specific spatiotemporal statistics. CNN + RNN generalizes well (89.6%) but relies on GRU operators that are unsupported in CMSIS-NN and require Flex delegates in TFLite, preventing integer-only execution. MobileNet variant achieves the highest SemLex100 accuracy (97.6%), but its size exceeds the memory budget of constrained inference stacks. S3D-Conv1D demonstrates a stable degradation profile, dropping 16.5 pp from WLASL100 to SemLex100 while remaining fully INT8-compatible.

6.5.3. Class-Wise Behavior and Failure Modes

Class-wise analysis (Figure 7) reveals robustness differences not visible in aggregate accuracy. On WLASL100, S3D, CNN + RNN, MobileNet variant, and S3D-Conv1D all achieve F1 = 1.00 on at least one class; e-LSTM peaks at 0.88. Worst-class behavior is more revealing: CNN + RNN degrades moderately (F1 = 0.91), S3D and S3D-Conv1D show lower bounds of 0.78 and 0.75, respectively, MobileNet variant reaches 0.86, and e-LSTM collapses to F1 = 0.00 on at least one class.

On SemLex100, worst-class performance diverges sharply: S3D and S3D-Conv1D drop to F1 $\approx$ 0.20, CNN + RNN retains F1 $\approx$ 0.50, MobileNet variant maintains the highest lower bound (F1 = 0.67), and e-LSTM collapses to F1 = 0.00. High mean accuracy therefore masks severe per-class failures in architectures with heavy spatiotemporal coupling or compact recurrent representations, while the pretrained MobileNetV2 backbone provides more robust per-class discrimination at the cost of operator compatibility.



Figure 7. Best and worst class F1-score per model on WLASL100 and SemLex100.

6.5.4. INT8 Quantization Performance

Table 12 presents the performance of quantized versions of the evaluated models.

Table 12. PTQ INT8 accuracy across datasets.

Model	WLASL100			SemLex100		
	FP32 (%)	INT8 (%)	Δ	FP32 (%)	INT8 (%)	Δ
S3D (mixed)	99.29	99.20	−0.09	64.29	60.38	−3.91
CNN + RNN (hybrid)	99.29	99.29	0.00	89.62	89.62	0.00
e-LSTM (INT8)	44.20	20.09	−24.11	59.78	56.95	−2.83
MobileNet variant (INT8)	99.46	96.36	−3.10	97.57	91.04	−6.53
S3D-Conv1D (INT8)	98.96	98.60	−0.36	82.48	79.92	−2.56

S3D and CNN + RNN preserve accuracy under their respective quantization schemes but cannot achieve full INT8 execution: S3D requires Conv3D and standalone BatchNormalization, which have no INT8 kernels in CMSIS-NN; CNN + RNN's GRU requires Flex delegates. e-LSTM collapses from 44.2% to 20.1% under INT8, confirming numerical instability in quantized recurrent representations. MobileNet variant achieves competitive PTQ accuracy on both datasets.

S3D-Conv1D achieves full INT8 execution with $\Delta < 1$ pp on WLASL100 under PTQ, and $\Delta \approx 2.6$ pp on SemLex100, confirming that its operator set (Conv2D, Conv1D, GAP, Dense, Softmax) is inherently quantization-friendly. Under QAT, the accuracy drop on WLASL100 reduces further to 0.32 pp.

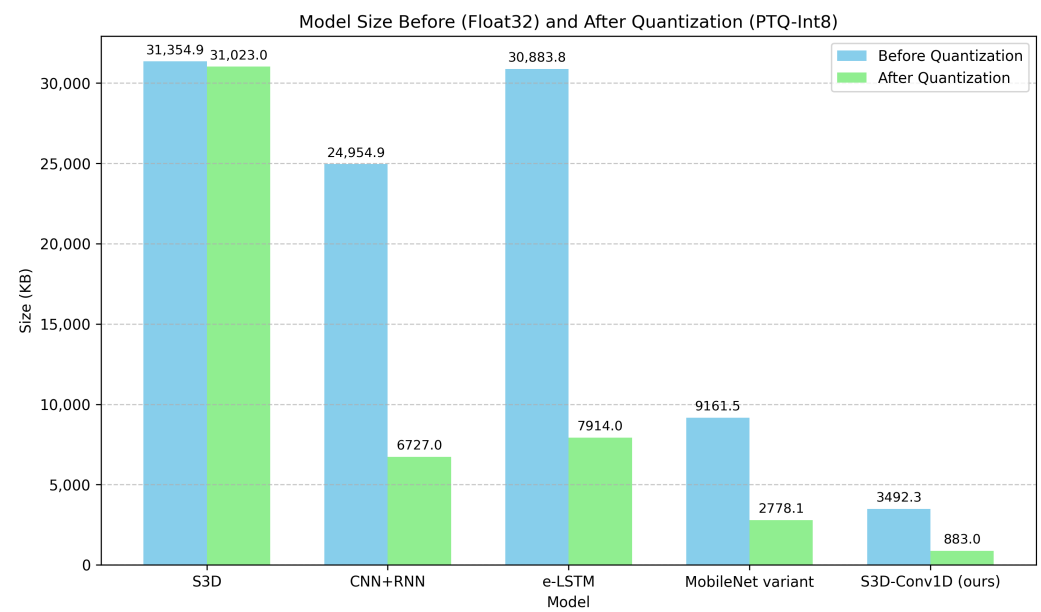
6.5.5. Compression and Runtime

Table 13 presents a summary of model compression.

S3D achieves negligible compression due to Conv3D and BatchNormalization constraints. CNN + RNN, e-LSTM, and MobileNet variant compress substantially but remain above microcontrollers' memory budget. S3D-Conv1D compresses from 3.5 MB to 883 KB at a $\times 4$ ratio, the smallest result across all evaluated architectures. S3D-Conv1D is $3\times$ smaller than the next smallest baseline in INT8. Figure 8 illustrates the comparative model sizes before and after PTQ INT8 quantization, highlighting the significant compression achieved by S3D-Conv1D.

Table 13. Model compression summary (PTQ INT8).

Model	Params (K)	FP32 (KB)	INT8 (KB)	Ratio	Lat. (ms)
S3D	8027	31,355	31,023	×1.0	9126
CNN + RNN	6390	24,955	6727	×3.7	84.9
e-LSTM	7906	30,884	7914	×3.9	39.3
MobileNet variant	2386	9162	2778	×3.3	1895
S3D-Conv1D	892	3492	883	×4.0	26.0

**Figure 8.** Model size before and after PTQ INT8 quantization.

6.5.6. Operator Compatibility

A model is fully compatible with constrained inference only if all its operators are natively supported in INT8 without Flex delegates. Table 14 maps the critical operators of each architecture to their support status across TFLite, CMSIS-NN, and NNoM frameworks.

Table 14. Operator compatibility across inference stacks. ✓ = INT8 supported; ~ = partial or Flex delegate required; × = not supported.

Operator	TFLite	CMSIS-NN	NNoM	Used by
Conv2D (INT8)	✓	✓	✓	All
Conv1D	✓	✓	~	S3D-Conv1D
DepthwiseConv2D	✓	✓	✓	S3D-Conv1D (DW variant), MobileNetV2
GlobalAveragePooling	✓	✓	✓	S3D-Conv1D, MobileNetV2
Dense/Softmax	✓	✓	✓	All
GRU/LSTM	~ (Flex)	×	~	CNN + RNN, e-LSTM
Custom Attention	×	×	×	e-LSTM
Conv3D/Sep. 3D	~	~	~	S3D
BatchNorm (standalone) ~ (fused)	~	×	✓	S3D
Inverted Residuals	✓	~	✓	MobileNetV2
ReLU/ReLU6	✓	✓	✓	S3D-Conv1D, MobileNetV2

6.5.7. Runtime Profiling

Inference benchmarks were executed on a CPU-only laptop (Intel Celeron N4500, 1.10 GHz, single-threaded TFLite execution). FLOPs are estimated via `tf_lite_flops` [66];

latency is the median over 20 executions. The energy proxy is CPU utilization (%) multiplied by latency, used solely as a comparative indicator of computational cost under uniform conditions. These figures are not equivalent to embedded hardware measurements and cannot be compared across platforms. FLOPs counts are reported as indicators of architectural complexity, not energy estimates. As reported in Table 15, the runtime summary captures FLOPs, latency, throughput, and energy proxy values across all evaluated architectures, highlighting the efficiency gains of lightweight models such as S3D-Conv1D compared to heavier baselines.

Table 15. Runtime summary (CPU proxy, INT8, 20-run median latency).

Model	FLOPs (10^6)	Lat. (ms)	FPS	Energy Proxy
S3D	0.20	9126	0.11	912,635
CNN + RNN	1267	84.9	11.8	1655
e-LSTM	0.60	39.3	25.5	2448
MobileNet variant	105.4	1895	0.53	189,547
3D-Conv1D	191.1	26.0	38.4	1560

S3D's large latency reflects runtime reliance on TF Select ops and an unoptimized graph. CNN + RNN is computationally heavier (1.27 GFLOPs) and slower. e-LSTM achieves low latency but fails on accuracy and requires Mediapipe as an external dependency for keypoint extraction. MobileNet variant's 1895 ms latency reflects the cost of processing 24 frames through a full MobileNetV2 backbone. S3D-Conv1D achieves the best balance of accuracy, latency, and model size among all evaluated architectures.

7. Conclusions

7.1. Discussion

This study investigated word-level sign language recognition with MCU deployment constraints, without reliance on cloud computation, external accelerators, or unsupported operators. To address these constraints, we proposed S3D-Conv1D, a separable spatiotemporal architecture explicitly designed for constrained inference, and evaluated it against S3D, CNN + RNN, e-LSTM, and a MobileNet variant using a unified training, quantization, and profiling framework on WLASL and the SemLex benchmark datasets.

Experiments show that S3D-Conv1D achieves competitive float32 accuracy on WLASL100 ($98.96 \pm 0.36\%$) with stable per-class behavior and consistent cross-dataset generalization (82.5% on SemLex100). Under PTQ INT8, it maintains near-equivalent accuracy ($\Delta < 1$ pp on WLASL100, $\Delta \approx 2.6$ pp on SemLex100) while compressing to 883 KB at a $\times 4$ ratio. It is the smallest across all evaluated architectures. Under QAT, the accuracy drop on WLASL100 reduces to 0.32 pp, and on WLASL300 QAT recovers the PTQ accuracy gap from 17.91 pp to 1.22 pp, confirming that quantization-aware fine-tuning is necessary for stable deployment at larger vocabulary scales.

An ablation study across kernel size k , temporal resolution T , and spatial block type establishes that the baseline configuration ($k = 3$, $T = 24$, Conv2D) occupies the most robust operating point across accuracy, quantization stability, and cross-scale generalization. An ultralight variant ($T = 8$, $k = 1$, DepthwiseConv2D, QAT INT8) achieves $98.51 \pm 0.89\%$ on WLASL100 at 24.7 KB and 0.94 MFLOPs, and $74.11 \pm 5.16\%$ on WLASL300 at 34.9 KB, the most compact fully INT8-compatible configuration identified in this study.

Among the baselines, the MobileNet variant achieves the highest accuracy on SemLex100 dataset (97.6%) through transfer learning from a pretrained backbone, but at a 9.1 MB float32 size and 1.895 s inference latency, it is incompatible with constrained de-

ployment. CNN + RNN and e-LSTM depend on recurrent and attention operators that are either unsupported or require Flex delegates in CMSIS-NN and NNoM, preventing integer-only execution. S3D-Conv1D is the only architecture in this study that simultaneously satisfies all operator compatibility requirements, full INT8 execution with no Flex delegates, sub-1 MB model size, and low inference latency on the CPU proxy.

These results demonstrate that competitive word-level sign language recognition can be achieved under microcontroller deployment constraints when architectural design prioritizes quantization stability and operator compatibility from the outset. S3D-Conv1D illustrates that a carefully constrained convolutional temporal backbone, combined with a domain-aware calibration strategy and systematic ablation of design parameters, can satisfy real deployment limitations that more expressive recurrent and large backbone models cannot.

To support reproducibility, the training scripts, source code, experiments and trained models (FP32 and INT8) for S3D-Conv1D are openly available in the GitHub repository <https://github.com/Samuel-KIMPINDE/S3D-Conv1D-for-SLR-for-Microcontrollers> (accessed on 22 March 2026), with a citable release archived on Zenodo: <https://doi.org/10.5281/zenodo.18758927>.

7.2. Limitations and Challenges

Despite these advances, several limitations remain. Dataset diversity is restricted; existing benchmarks underrepresent signer variability, environmental conditions, and non-manual linguistic features such as facial expressions, constraining generalization. Temporal modeling in S3D-Conv1D captures short- to mid-range dependencies, but extending to continuous or sentence-level recognition would require deeper temporal context or attention mechanisms that currently exceed microcontroller resource budgets.

Quantization remains an inherent trade-off. Although S3D-Conv1D exhibits strong robustness to post-training INT8 quantization, precision loss due to activation clipping and calibration bias cannot be fully eliminated. The system is also limited to isolated word-level recognition, leaving continuous decoding, grammar modeling, and multilingual scalability as open challenges. Ensuring fairness across signers, dialects, and recording conditions will require broader datasets and ethically grounded evaluation practices.

7.3. Broader Implications and Ethical Scope

This research demonstrates the practical potential of TinyML to enable private, offline, and inclusive sign language recognition. By combining open datasets, public toolchains, and affordable hardware, it advances reproducible and accessible AI for assistive technology. On-device inference eliminates cloud dependency, preserves user privacy, and enables deployment in connectivity-constrained or sensitive environments.

Responsible deployment must go beyond technical performance. Continuous fairness auditing, transparent reporting of failure cases, and collaboration with deaf communities are essential to ensure equitable outcomes. The proposed architecture and evaluation pipeline align with these principles by emphasizing transparency, reproducibility, and deployment realism.

7.4. Future Research Directions

Future work should extend S3D-Conv1D toward continuous signing, integrate multimodal inputs such as depth, IMU, or skeletal data to further stabilize and improve performance. Structured pruning and knowledge distillation offer promising avenues for additional compression, particularly by leveraging larger separable or recurrent models as teachers. Hardware-aware neural architecture search could further refine designs under strict Flash and SRAM budgets.

Direct deployment and power profiling on physical platforms such as AE3 Edge [67], and UNO Q [68] will strengthen empirical validation. User-centered studies are also essential to evaluate usability, interpretability, and long-term acceptance in real-world settings.

7.5. Final Remarks

This work demonstrates that accurate, efficient, and robust sign language recognition is achievable on microcontroller-class hardware through principled, deployment-aware spatiotemporal model design. By introducing S3D-Conv1D and rigorously evaluating it against strong separable and recurrent baselines, this work clarifies the architectural and systems-level requirements for true edge readiness. As embedded hardware continues to evolve, such transparent, efficient, and inclusive approaches will be central to delivering trustworthy assistive technologies that operate autonomously at the edge.

Author Contributions: Conceptualization, S.L.K. and P.O.O.; methodology, S.L.K.; investigation, S.L.K.; validation, P.O.O.; writing—original draft preparation, S.L.K.; supervision, P.O.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The large-scale datasets used in this study are publicly available. WLASL-Processed can be found at <https://www.kaggle.com/datasets/risangbaskoro/wlasl-processed> (accessed on 22 March 2026); WLASL2000-Resized is available at <https://www.kaggle.com/datasets/sttaseen/wlasl2000-resized> (accessed on 22 March 2026); and the SemLex Benchmark dataset can be accessed at <https://github.com/leekezar/SemLex> (accessed on 22 March 2026). ASL-Kimpinde can be found at <https://zenodo.org/records/18754451> (accessed on 22 March 2026). The source code, training scripts, and trained models (FP32 and INT8) for the S3D-Conv1D architecture are openly available in the GitHub repository <https://github.com/Samuel-KIMPINDE/S3D-Conv1D-for-SLR-for-Microcontrollers> (accessed on 22 March 2026), with a citable release archived on Zenodo: <https://doi.org/10.5281/zenodo.18758927>. All datasets and code are openly accessible and were used in accordance with their respective licenses.

Acknowledgments: The authors thank the University of Johannesburg for providing the research environment. During the preparation of this manuscript, the authors used ChatGPT (OpenAI, version GPT-5.4, free tier), Microsoft Copilot (Microsoft 365 integration, free tier, March 2026 release), and NotebookLM (Google, Gemini 3 engine, free tier) to assist with paraphrasing, conciseness of text, and organization of the literature review. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASL	American Sign Language
CMSIS-NN	Cortex Microcontroller Software Interface Standard—Neural Networks
CNN	Convolutional Neural Network
Conv1D	One-Dimensional Convolution
Conv2D	Two-Dimensional Convolution
CPU	Central Processing Unit
DHH	Deaf and Hard-of-Hearing
DW	Depthwise (as in DepthwiseConv2D)
EMA	Exponential Moving Average
F1	F1-Score (harmonic mean of precision and recall)
FLOP	Floating-Point Operation
FPS	Frames Per Second

GAP	Global Average Pooling
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HCI	Human–Computer Interaction
INT8	8-bit Integer Representation
ISL	Indian Sign Language
KD	Knowledge Distillation
LSTM	Long Short-Term Memory
MAC	Multiply–Accumulate Operation
MCU	Microcontroller Unit
NNoM	Neural Network on Microcontroller
PTQ	Post-Training Quantization
QAT	Quantization-Aware Training
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
S3D	Separable 3D Convolution
SLR	Sign Language Recognition
SRAM	Static Random-Access Memory
STE	Straight-Through Estimator
TFLite	TensorFlow Lite
TFLM	TensorFlow Lite for Microcontrollers
TinyML	Machine Learning on Tiny/Resource-Constrained Devices
ViT	Vision Transformer
WLASL	Word-Level American Sign Language dataset
WSLR	Word-Level Sign Language Recognition

Appendix A. Notation Summary

Table A1. Notation summary.

Symbol	Description
<i>Video input and indices</i>	
X	Input video tensor, $X \in \mathbb{R}^{T \times H \times W \times C}$
X_t	Video frame at time step t
t	Temporal index, $t \in \{1, \dots, T\}$
T	Number of frames per video (temporal length)
H, W	Spatial height and width of each frame
C	Number of input channels per frame
K	Vocabulary size (number of output classes)
<i>Model parameters and layers</i>	
f_θ	Neural network parameterized by θ
θ	Trainable floating-point parameters
θ^*	Optimized parameters after training
θ_q	Quantized INT8 parameters
$W^{(l)}, b^{(l)}$	Weight tensor and bias vector of layer l
k	Kernel size (spatial or temporal, context-dependent)
$*$	Convolution operator (2D or 1D, context-dependent)
$\sigma(\cdot)$	ReLU activation, $\sigma(x) = \max(0, x)$
MaxPool($\cdot$)	Spatial max-pooling operator

Table A1. Cont.

Symbol	Description
<i>Feature representations</i>	
$f_t^{(1)}, f_t^{(2)}$	Output feature maps of Conv2D blocks 1 and 2 at time t
s_t	Flattened spatial feature vector at time t
S	Temporal sequence of spatial embeddings, $S = [s_1, \dots, s_T]$
d	Dimensionality of flattened spatial embedding s_t
d'	Number of temporal Conv1D filters
h	Temporal feature map after Conv1D, $h \in \mathbb{R}^{T \times d'}$
z	Pooled representation after GlobalAveragePooling1D
$\hat{y}$	Predicted class probability vector, $\hat{y} \in \Delta^{K-1}$
<i>Complexity and resource metrics</i>	
C_{in}, C_{out}	Input and output channel counts of a convolutional layer
H_{out}, W_{out}	Spatial resolution of a layer output
$\mathbf{A}_\ell$	Activation tensor of layer ℓ
$ \mathbf{A}_\ell $	Number of elements in $\mathbf{A}_\ell$
$\text{Params}(\cdot)$	Number of learnable parameters of a layer
$\text{MACs}(\cdot)$	Number of multiply–accumulate operations of a layer
FLOPs	Floating-point operations; $\text{FLOPs} = \alpha \cdot \text{MACs}$
α	MAC-to-FLOP conversion factor, $\alpha \in \{1, 2\}$
$ \theta $	Total number of learnable parameters
$ \theta_{\text{feat}} $	Parameters of the spatiotemporal feature extractor
θ_{cls}	Parameters of the dense classification layer
$\mathcal{O}(\cdot)$	Asymptotic complexity notation
<i>Quantization</i>	
x	Floating-point value to be quantized
x_q	Quantized INT8 integer value
s	Quantization scale factor
z	Quantization zero-point
$x_{\min}, x_{\max}$	Observed minimum and maximum of a tensor
$\mathcal{R}$	Representative calibration dataset (PTQ)
$\mathcal{D}, \mathcal{V}, \mathcal{T}$	Training, validation, and test sets
η	Learning rate
$\mathcal{L}$	Training loss

References

1. World Health Organization. Deafness and Hearing Loss. 2023. Available online: <https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss> (accessed on 18 October 2025).
2. Lopez, M.H.; Vergara, L.G.L. Accessibility to Ambulatory and Emergency Services for Deaf People in the Context of a University Hospital: A Macroergonomic Approach. *Lect. Notes Netw. Syst.* **2021**, *260*, 498–505. [\[CrossRef\]](#)
3. Tannenbaum-Baruchi, C.; Kugel, A.; Rapaport, C. Needs and barriers in bachelor’s degree programs: Perspectives from deaf and hard-of-hearing students and higher education institutions in Israel. *High. Educ. Res. Dev.* **2025**, *45*, 255–271. [\[CrossRef\]](#)
4. Wilson-Menzfeld, G.; Gates, J.; Jackson-Corbett, C.; Erfani, G. Communication Experiences of Deaf/Hard-of-Hearing Patients During Healthcare Access and Consultation: A Systematic Narrative Review. *Health Soc. Care Community* **2025**, *2025*, 8867224. [\[CrossRef\]](#)
5. Bodenmann, P.; Singy, P.; Kasztura, M.; Graells, M.; Cantero, O.; Morisod, K.; Malebranche, M.; Smith, P.; Beyeler, S.; Sebaï, T.; et al. Developing and Evaluating a Capacity-Building Intervention for Healthcare Providers to Improve Communication Skills and Awareness of Hard of Hearing and D/deaf Populations: Protocol for a Participative Action Research-Based Study. *Front. Public Health* **2021**, *9*, 615474. [\[CrossRef\]](#) [\[PubMed\]](#)
6. Terry, J.; Meara, R.; England, R. “They still phone even though they know I’m deaf”: Exploring experiences of deaf people in health services in Wales, UK. *J. Public Health* **2024**, *46*, e520–e527. [\[CrossRef\]](#)

7. Kumar, N.; Ahmed, R.; H, V.B.; Salvi, S.; Panjwani, Y. Continuous Sign Language Recognition Using Leap Motion Sensor. In *2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1160–1165. [\[CrossRef\]](#)
8. Zhu, Q.; Li, J.; Yuan, F.; Gan, Q. Continuous sign language recognition based on motor attention mechanism and frame-level self-distillation. *Mach. Vis. Appl.* **2025**, *36*, 7. [\[CrossRef\]](#)
9. Tang, H.Q.; Vo, T.D.N.; Tran, T.P.; Pham, D.D. Two-Stream S3D Architecture for Word-Level Sign Language Recognition. In *Proceedings of the 2024 9th International Conference on Intelligent Information Technology*; ACM: Rochester, NY, USA, 2024; pp. 365–370. [\[CrossRef\]](#)
10. Shi, X.; Jiao, X.; Meng, C.; Bian, Z. 3D Sign language recognition based on multi-path hybrid residual neural network. In *Proceedings of the 2022 14th International Conference on Machine Learning and Computing*; ACM: Rochester, NY, USA, 2022. [\[CrossRef\]](#)
11. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D. Temporal Convolutional Networks for Action Segmentation and Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2017; pp. 156–165.
12. TensorFlow Lite—Interpreter API. Available online: <https://www.tensorflow.org/lite/> (accessed on 18 October 2025).
13. CMSIS-NN: Efficient Neural Network Kernels for ARM Cortex-M Processors. Available online: <https://developer.arm.com/Tools%20and%20Software/Software%20Development%20Tools/CMSIS/CMSIS-NN> (accessed on 18 October 2025).
14. Xie, S.; Sun, C.; Huang, J.; Tu, Z.; Murphy, K. Rethinking Spatiotemporal Feature Learning: Speed-Accuracy Trade-offs in Video Classification. In *Proceedings of the European Conference on Computer Vision (ECCV)*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 305–321.
15. Sharma, V.; Sharma, A.; Saini, S. Real-time attention-based embedded LSTM for dynamic sign language recognition on edge devices. *J. Real-Time Image Process.* **2024**, *21*, 53. [\[CrossRef\]](#)
16. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2018; pp. 4510–4520.
17. Abhishek, Y.; Sumanathilaka, D. End-to-End Sign Language Recognition Pipeline: Towards Energy Efficient Modeling. *Procedia Comput. Sci.* **2025**, *265*, 483–490. [\[CrossRef\]](#)
18. Patil, A.; Hatti, S.; Shinde, A.; Sonvalkar, P.K.; Mulay, Y. Smart Vision for Silent Voices: A Review of Sign Language Recognition Technologies. In *Proceedings of the 2025 6th International Conference for Emerging Technology (INCET)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–5. [\[CrossRef\]](#)
19. Midha, N.; Bansal, S. Exploring Sign Language in Complex Background: Techniques, Datasets and Challenges. In *Proceedings of the 2025 12th International Conference on Computing for Sustainable Global Development (INDIACom)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6. [\[CrossRef\]](#)
20. Khomami, S.A.; Shamekhi, S. Persian sign language recognition using IMU and surface EMG sensors. *Measurement* **2021**, *168*, 108471. [\[CrossRef\]](#)
21. Das, S.; Yadav, S.K.; Samanta, D. *Isolated Sign Language Recognition Using Deep Learning*; Springer Nature Switzerland: Cham, Switzerland, 2024; pp. 343–356. [\[CrossRef\]](#)
22. Chinchmalatpure, S.; Matala, A.; Mane, S.; Petkar, V.; Kavathe, S.; Kokate, P.D. Deep Learning-Based Dynamic Sign Language Recognition System. In *Proceedings of the Smart Computing Paradigms: Advanced Data Mining and Analytics*; Simic, M., Bhateja, V., Azar, A.T., Lydia, E.L., Eds.; Springer: Singapore, 2025; pp. 75–89.
23. Konstantinidis, D.; Dimitropoulos, K.; Daras, P. Sign language recognition based on hand and body skeletal data. In *Proceedings of the 2018—3DTV-Conference: The True Vision—Capture, Transmission and Display of 3D Video (3DTV-CON)*; IEEE: Piscataway, NJ, USA, 2018; pp. 1–4. [\[CrossRef\]](#)
24. Radhakrishnan, S.; Mohan, N.C.; Varma, M.; Varma, J.; Pai, S.N. Cross Transferring Activity Recognition to Word Level Sign Language Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Piscataway, NJ, USA, 2022; pp. 2445–2452. [\[CrossRef\]](#)
25. Amutha, S.; Shanmukh, N.; Naidu, A.V.S.R.P.; Kumar, P.V.; Narayana, G.S.S. Real-Time Sign Language Recognition using a Multimodal Deep Learning Approach. In *Proceedings of the 2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*; IEEE: Piscataway, NJ, USA, 2023. [\[CrossRef\]](#)
26. Alsolai, H.; Alsolai, L.; Al-Wesabi, F.N.; Othman, M.; Rizwanullah, M.; Abdelmageed, A.A. Automated sign language detection and classification using reptile search algorithm with hybrid deep learning. *Heliyon* **2024**, *10*, e23252. [\[CrossRef\]](#)
27. Kumar, R.D.; Saravanan, K.; Nallusamy, C.; Sakthivel, K. Transformative Ai-Driven Tamil Sign Language Recognition and Speech Synthesis Using ViT-CNN. In *Proceedings of the 2025 3rd International Conference on Artificial Intelligence and Machine Learning Applications Theme: Healthcare and Internet of Things (AIMLA)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6. [\[CrossRef\]](#)
28. Astya, R.; Agrawal, A.; Chauhan, A.; Sony, A.K.; Thakur, K.R.; Nand, P. Enhancing Sign Language Detection using Tensor Flow. In *Proceedings of the 2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*; IEEE: Piscataway, NJ, USA, 2023. [\[CrossRef\]](#)

29. Xie, P.; Zhao, M.; Hu, X. PiSLTRC: Position-Informed Sign Language Transformer With Content-Aware Convolution. *IEEE Trans. Multimed.* **2022**, *24*, 3908–3919. [\[CrossRef\]](#)
30. Vandana; Kamal; Saharia, S. Attention at Hand: A Comprehensive Survey of Transformer-based Hand Gesture and Landmark Detection Models. In *Proceedings of the 2025 IEEE Guwahati Subsection Conference (GCON)*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6. [\[CrossRef\]](#)
31. Saint-Germain, L.; Le Gal, B.; Baldacci, F.; Crenne, J.; Jegou, C.; Loty, S. Methodology to Adapt Neural Network on Constrained Device at Topology level. In *Proceedings of the 2022 IEEE Workshop on Signal Processing Systems (SiPS)*; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6. [\[CrossRef\]](#)
32. Xiao, P.; Zhang, C.; Guo, Q.; Xiao, X.; Wang, H. Neural Networks Integer Computation: Quantizing Convolutional Neural Networks of Inference and Training for Object Detection in Embedded Systems. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2024**, *17*, 15862–15884. [\[CrossRef\]](#)
33. Hizem, M.; Bousbia, L.; Ben Dhiab, Y.; Aoueilayine, M.O.E.; Bouallegue, R. Reliable ECG Anomaly Detection on Edge Devices for Internet of Medical Things Applications. *Sensors* **2025**, *25*, 2496. [\[CrossRef\]](#) [\[PubMed\]](#)
34. Zhang, Y.; Su, Z.; Meng, L. Hand Gesture Recognition for Real-Time Nano Drone Control on RISC-V. In *Proceedings of the 2025 IEEE International Conference on Real-Time Computing and Robotics (RCAR)*; IEEE: Piscataway, NJ, USA, 2025; pp. 95–100. [\[CrossRef\]](#)
35. Lê, M.T.; Wolinski, P.; Arbel, J. Efficient Neural Networks for Tiny Machine Learning: A Comprehensive Review. In *ACM Transactions on Intelligent Systems and Technology*; ACM: Rochester, NY, USA, 2023. [\[CrossRef\]](#)
36. Dantas, P.V.; Sabino Da Silva, W.; Cordeiro, L.C.; Carvalho, C.B. A comprehensive review of model compression techniques in machine learning. *Appl. Intell.* **2024**, *54*, 11804–11844. [\[CrossRef\]](#)
37. Salazar, M.; Bandara, D.; Bejarano, G. Evaluating Model Compression Techniques for On-Device Browser Sign Language Recognition Inference. In *Proceedings of the 2024 IEEE XXXI International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–7. [\[CrossRef\]](#)
38. Chen, N.; Mao, J.; Peng, Z.; Yi, J.; Tao, Z.; Wang, Y. Knowledge distillation-based lightweight network for power scenarios inspection. In *Proceedings of the 2023 China Automation Congress (CAC)*; IEEE: Piscataway, NJ, USA, 2023; pp. 3990–3996. [\[CrossRef\]](#)
39. Kumar, S.; Rachna, P.; Hiremath, R.B.; Ramadurgam, V.S.; Shaw, D.K. Survey on implementation of TinyML for real-time sign language recognition using smart gloves. In *Proceedings of the 2022 Fourth International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*; IEEE: Piscataway, NJ, USA, 2022. [\[CrossRef\]](#)
40. Kumar, S.; Poongodan, R.; Hiremath, R.B.; Ramadurgam, V.S.; Shaw, D.K. Implementation of Sign Language Recognition with TinyML Using Smart Gloves. In *AIP Conference Proceedings*; AIP Publishing: Melville, NY, USA, 2024. [\[CrossRef\]](#)
41. Yu, A.; Qian, C.; Guo, Y. A Real-Time Hand Gesture Recognition System on Raspberry Pi: A Deep Learning-Based Approach. In *Proceedings of the 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*; IEEE: Piscataway, NJ, USA, 2024. [\[CrossRef\]](#)
42. Dabwan, B.A.; Jadhav, M.E.; Ismil, O.A.; Hassan, E.A.; Farah, E.A.; Mohammad, A.A.; Ali, Y.A. Detecting Gesture Language for Deaf and Mute People Using on Ultra-Low-Power TinyML Model. In *Proceedings of the 2024 International Conference on Data Science and Network Security (ICDSNS)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–5. [\[CrossRef\]](#)
43. Trpcheska, A.; Zevnik, F.; Bader, S. Towards Real-Time Vision-Based Sign Language Recognition on Edge Devices. In *Proceedings of the 2024 IEEE Sensors Applications Symposium (SAS)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–6. [\[CrossRef\]](#)
44. Krivokapic, B.; Tomovic, S.; Radusinovic, I.; Jovanovic, A. Implementation and Performance Evaluation of Convolutional Neural Network models for Low-Power Microcontrollers with Constrained Resources. In *Proceedings of the 2024 28th International Conference on Information Technology (IT)*; IEEE: Piscataway, NJ, USA, 2024. [\[CrossRef\]](#)
45. Lalapura, V.S.; Bhimavarapu, V.R.; Amudha, J.; Sathesh, H.S. A Systematic Evaluation of Recurrent Neural Network Models for Edge Intelligence and Human Activity Recognition Applications. *Algorithms* **2024**, *17*, 104. [\[CrossRef\]](#)
46. Beltrán-Escobar, M.; Alarcón, T.E.; Rumbo-Morales, J.Y.; López, S.; Ortiz-Torres, G.; Sorcia-Vázquez, F.D.J. A Review on Resource-Constrained Embedded Vision Systems-Based Tiny Machine Learning for Robotic Applications. *Algorithms* **2024**, *17*, 476. [\[CrossRef\]](#)
47. Mnif, M.; Sahnoun, S.; Kaaniche, M.; Atitallah, B.B.; Fakhfakh, A.; Kanoun, O. Ultra-Fast Edge Computing Approach for Hand Gesture Classification Based on EIT Measurements. In *Proceedings of the 2024 IEEE International Symposium on Robot and Sensors Environments (ROSE)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1–7. [\[CrossRef\]](#)
48. Shingu, T.; Uchiyama, H.; Watanabe, T.; Ohno, Y. Electrochemical reservoir computing based on surface-functionalized carbon nanotubes. *Carbon* **2023**, *214*, 118344. [\[CrossRef\]](#)
49. Rutishauser, G.; Scherer, M.; Fischer, T.; Benini, L. 7 J/inference end-to-end gesture recognition from dynamic vision sensor data using ternarized hybrid convolutional neural networks. *Future Gener. Comput. Syst.* **2023**, *149*, 717–731. [\[CrossRef\]](#)
50. Huamani-Malca, J.; Bejarano, G. Comparing Incremental Learning Approaches for a Growing Sign Language Dictionary. In *Proceedings of the Annual International Conference on Information Management and Big Data; Communications in Computer and Information Science*; Springer: Cham, Switzerland, 2024; Volume 2142, pp. 97–106. [\[CrossRef\]](#)

51. Chen, Y.; Li, J.; Lin, S.; Xu, Y.; Yang, C. A BiLSTM and CTC Based Multi-Sensor Information Fusion Frame for Continuous Sign Language Recognition. In *Proceedings of the 2024 10th International Conference on Electrical Engineering*; IEEE: Piscataway, NJ, USA, 2024; Volume 37, pp. 310–315. [\[CrossRef\]](#)
52. Kezar, L.; Thomason, J.; Daniels, A.; Connor, B.; Ferster, R.; Berger, L.; Caselli, N.; Sehyr, Z.; Pontecorvo, E. The Sem-Lex Benchmark: Modeling ASL Signs and their Phonemes. In *Proceedings of the ASSETS 2023—Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility*; ACM: Rochester, NY, USA, 2025. [\[CrossRef\]](#)
53. Li, D.; Opazo, C.R.; Yu, X.; Li, H. Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*; IEEE: Piscataway, NJ, USA, 2020. [\[CrossRef\]](#)
54. Zhang, Z.; Gao, Y.; Fan, J.; Zhao, Z.; Yang, Y.; Yan, S. SelectQ: Calibration Data Selection for Post-training Quantization. *Mach. Intell. Res.* **2025**, *22*, 499–510. [\[CrossRef\]](#)
55. Singleton, J.L.; Martin, A.J.; Morgan, G. Ethics, Deaf-Friendly Research, and Good Practice When Studying Sign Languages. In *Research Methods in Sign Language Studies*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2015; Chapter 1, pp. 5–20. [\[CrossRef\]](#)
56. Kolbe, V. Open science versus data protection—Challenges and solutions in sign language acquisition studies. *Hrvat. Rev. Za Rehabil. Istraz.* **2022**, *58*, 109–120. [\[CrossRef\]](#)
57. Parks, E. Listen First: Dialogic Research Ethics With Caribbean Signing Communities. *Ethics Behav.* **2017**, *29*, 156–166. [\[CrossRef\]](#)
58. Batnasan, G.; Otgonbold, M.E.; Ali Memon, Q.; Shih, T.K.; Gochoo, M. Head and Hands Tunneling Pipeline for Enhancing Sign Language Recognition. *IEEE Access* **2025**, *13*, 127926–127940. [\[CrossRef\]](#)
59. Guan, M.; Wang, Y.; Ma, G.; Liu, J.; Sun, M. MSKA: Multi-stream keypoint attention network for sign language recognition and translation. *Pattern Recognit.* **2025**, *165*, 111602. [\[CrossRef\]](#)
60. Liu, Y.; Zhang, W.; Ren, S.; Huang, C.; Yu, J.; Xu, L. SCOPE: Sign language contextual processing with embedding from LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI Press: Washington, DC, USA, 2025. [\[CrossRef\]](#)
61. Wikipedia Contributors. Big O Notation. 2025. Available online: https://en.wikipedia.org/wiki/Big_O_notation (accessed on 18 December 2025).
62. Wikipedia Contributors. Space Complexity. 2025. Available online: https://en.wikipedia.org/wiki/Space_complexity (accessed on 18 December 2025).
63. Wikipedia Contributors. Time Complexity. 2025. Available online: https://en.wikipedia.org/wiki/Time_complexity (accessed on 18 December 2025).
64. Kimpinde, S.L. *ASL-Kimpinde Set: Minimal Dataset for Sign Language Recognition*, Version 1.0; Zenodo: Geneva, Switzerland, 2026. [\[CrossRef\]](#)
65. Yin, P.; Lyu, J.; Zhang, S.; Osher, S.; Qi, Y.; Xin, J. Understanding straight-through estimator in training activation quantized neural nets. *arXiv* **2019**, arXiv:1903.05662. [\[CrossRef\]](#)
66. lisosia. Tflite-Flops: Roughly Calculate FLOPs of a Tflite Model. 2021. Available online: <https://github.com/lisosia/tflite-flops> (accessed on 22 October 2025).
67. OpenMV. OpenMV AE3—Python Programmable AI Camera for Embedded Vision. Available online: <https://openmv.io/products/openmv-ae3> (accessed on 28 June 2025).
68. Arduino. Arduino UNO Q Documentation. 2025. Available online: <https://docs.arduino.cc/hardware/uno-q/> (accessed on 22 October 2025).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.