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

Samuel Longwani Kimpinde and Peter Olukanmi *

Posted Date: 27 February 2026

doi: 10.20944/preprints202602.1848.v1

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

 doi:10.20944/preprints202602.1848.v1

Article

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

Samuel Longwani Kimpinde and Peter Olukanmi *

University of Johannesburg
* 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 spatiotemporal architecture for isolated word-level sign language recognition, tailored for TinyML deployment. By factorizing spatial and temporal processing into lightweight two-dimensional (2D) convolutions followed by one-dimensional convolution (Conv1D) layers, the model eliminates recurrent dependencies and ensures deterministic, MCU-compatible computation. This design yields predictable latency, bounded activation memory, and stable model size, while supporting full post-training 8-bit integer (INT8) quantization and compatibility with TensorFlow Lite, CMSIS-NN, and NNoM. Three baselines, S3D, CNN+RNN, and attention-based embedded LSTM (e-LSTM), were evaluated using unified preprocessing, quantization, and profiling on WLASL100 and SemLex100 datasets. S3D-Conv1D achieved 98.4% float32 accuracy with superior stability and generalization. After INT8 quantization, it retained accuracy within 0.18% while compressing nearly 4× into sub-megabyte binaries (895.9 KB). Deployment profiling on a CPU-only runner, used as a proxy for microcontroller execution, showed S3D-Conv1D as the only architecture achieving full INT8 execution with real-time indicative performance (23.6 ms). These results demonstrate that efficient, edge-ready sign language recognition requires architectures designed around hardware constraints from the outset, rather than compressing high-capacity models.

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

1. Introduction

Hearing loss affects people across ages, contexts, and regions. More than 1.5 billion people worldwide live with some form of hearing loss, and about 430 million require rehabilitation services [1]. By 2050, that number could reach 2.5 billion. Most people affected live in regions with limited access to healthcare, education, and assistive infrastructure. This global reality underscores the urgency of developing inclusive, user-friendly technologies that bridge communication gaps for Deaf and Hard of Hearing (DHH) communities.

Identity within these groups is diverse: some align with the Deaf community, where deafness represents a cultural and linguistic identity, while others identify as deaf or hard of hearing and may use spoken language or assistive technologies [2]. The distinction between "Deaf" (capital D) and "deaf" or "hard of hearing" is fundamentally about language, culture, and belonging [2].

Sign languages are complete natural languages, each with its own grammar, structure, and cultural grounding. More than 300 sign languages are used worldwide, yet significant barriers remain between signers and non-signers [2,3]. Most hearing people do not understand sign language, and

 doi:10.20944/preprints202602.1848.v1

many DHH individuals remain excluded from everyday communication, education, healthcare, and social participation.

DHH individuals often face systemic barriers in education, healthcare, and public services [4–6]. In the UK, over 11 million people are deaf or hard of hearing, and only about 150,000 use British Sign Language (BSL) [4]. In Wales, shortages of interpreters and the lack of dedicated Deaf mental health services result in long travel distances for essential care [6]. Similar challenges exist globally. In Brazil, for example, nearly 10 million people have hearing loss, and users of LIBRAS continue to face obstacles in education and healthcare despite its official recognition [3]. These structural barriers contribute to broader exclusion and are associated with higher stress, poorer health outcomes, and increased mental health risks among DHH populations [5,6].

Communication gaps are particularly critical in healthcare. Essential processes such as appointment booking often rely on phone calls, forcing Deaf patients to depend on intermediaries or delay care [4]. Interpreter shortages exacerbate the issue: despite legal protections, only about 11% of patients receive equitable access [4]. In practice, patients are frequently left to rely on lip-reading, handwritten notes, or unreliable online services. Miscommunication and rushed consultations undermine autonomy, informed consent, and safety, sometimes leading to incorrect or incomplete treatment [4]. These realities highlight the need for tools that enable independent and reliable communication.

Barriers also persist in education. Universities often promise accessibility but fail to deliver fully inclusive environments. Approximately 39% of students report that lecturers speak too quickly, move while speaking, or refuse to use microphones, making lip-reading or interpreter use difficult [2]. Technical failures, missing captions, poor audio quality, and inaccessible online materials affect a further 28% of students. These gaps hinder learning and increase isolation. Importantly, many Deaf students navigate multiple languages, particularly when the language of instruction differs from their national sign language. Together, these factors significantly raise the risk of academic exclusion [2].

Automatic Sign Language Recognition (SLR) offers a potential technological bridge. Recent advances in deep learning have enabled accurate recognition of signs from video, translating gestures into text or speech [7–9]. However, most state-of-the-art SLR systems depend on powerful GPUs or cloud-based servers. This reliance limits accessibility in low-resource settings and raises concerns about privacy, latency, energy consumption, and network availability. For many real-world scenarios (such as rural environments, classrooms, clinics, or personal devices), cloud dependence is impractical or unacceptable.

TinyML (Tiny Machine Learning) provides a compelling alternative by enabling deep learning inference on ultra-low-power microcontrollers. For sign language recognition, this paradigm enables fully offline operation, low latency, and strong privacy guarantees. In this context, accessibility becomes personal, embedded, and autonomous, rather than centralized and infrastructure-dependent.

Historically, SLR systems have followed two main paradigms: sensor-based and vision-based approaches. Sensor-based systems rely on gloves, inertial sensors, or EMG signals [7,10,11]. While robust to lighting and occlusion, these systems are intrusive and usually impractical for daily use. Vision-based approaches, which infer signs from RGB video, depth data, or extracted keypoints, enable natural and contact-free interaction. Modern vision-based SLR models typically combine convolutional networks for spatial feature extraction with temporal modeling modules such as recurrent networks, temporal convolutions, or 3D convolutions [9,12,13]. Although effective on benchmarks, these architectures remain computationally intensive and poorly aligned with the constraints of microcontroller-class hardware.

Recent advances in edge inference frameworks, including TensorFlow Lite for Microcontrollers [14] and CMSIS-NN [15], have improved support for quantized and memory-efficient inference. Nevertheless, real deployment remains challenging. Integer-only execution does not fully support many commonly used temporal operators, particularly recurrent and attention-based layers. As a result, many SLR models require mixed precision, rely on kernels unavailable in MCU runtimes,

 doi:10.20944/preprints202602.1848.v1

or depend on external accelerators, preventing true microcontroller deployment. In practice, most promising SLR systems remain confined to desktop or mobile-class hardware.

Despite strong progress in large-scale and continuous SLR, fully self-contained word-level sign language recognition on microcontrollers remains largely unexplored. The primary obstacles are threefold: (1) the computational cost of spatio-temporal modeling using 3D convolutions or recurrent layers; (2) the difficulty of achieving stable quantization for sequential architectures; and (3) strict memory and power budgets inherent to MCU-class devices. Addressing these challenges demands architectures designed from the start for quantization, operator support, and predictable execution.

This work addresses these challenges by systematically studying word-level sign language recognition under realistic embedded deployment constraints. We explicitly evaluate architectural suitability for microcontroller-class inference. We evaluate four representative spatiotemporal model families: S3D [16], an embedded-oriented e-LSTM [17], a shallow CNN+RNN (GRU) architecture baseline, and the proposed S3D-Conv1D.

The scope of this study is intentionally constrained to isolate the core research questions. We consider isolated, word-level Sign Language recognition from grayscale video, with deployment-oriented inference evaluation. Continuous sentence recognition, cloud-based processing, multimodal inputs, and hardware accelerators are deliberately excluded. This controlled setting allows us to directly interrogate how spatiotemporal architectures behave when subjected to strict limits on memory and computation.

Within this scope, the central research questions are: (i) which spatiotemporal modeling strategies remain accurate and stable under full INT8 quantization; (ii) how architectural choices influence latency, energy behavior, and operator compatibility on embedded targets; and (iii) whether competitive recognition performance can be achieved with a simple, limited number of operators and without recurrent layers or external accelerators. The primary objective is not to maximize raw accuracy but to identify architectures that occupy the feasible intersection of accuracy, efficiency, and deployability.

Beyond technical considerations, the motivation is fundamentally human-centered. Fully on-device sign language recognition preserves privacy, operates without internet connectivity, and remains usable in classrooms, clinics, public services, and remote environments. By eliminating reliance on cloud infrastructure and complex hardware dependencies, this work supports scalable, low-cost, and trustworthy 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.

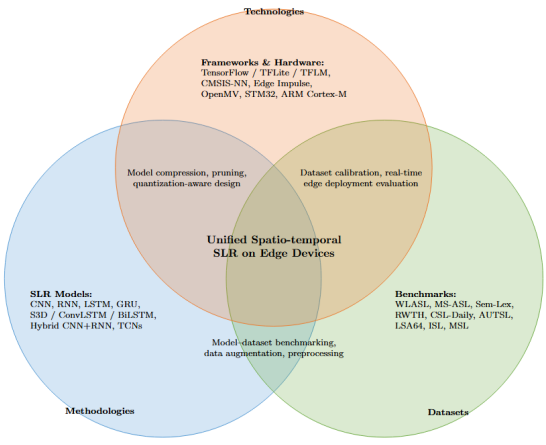

Figure 1. Taxonomy of the literature review showing relationships among methodologies, datasets, and technologies in SLR research on edge.

 doi:10.20944/preprints202602.1848.v1

2.1. Scope and Challenges in Sign Language Recognition (SLR)

Early sign language recognition systems developed incrementally as pattern recognition, computer vision, and sensor modeling. These systems were designed as assistive technology, so they carried both the promise and the limits of their time. Many tried to make sign language more accessible, but they often focused on accuracy first. This meant they sometimes left out inclusivity and real-world use [18].

These early SLR systems relied on handcrafted features such as movement tracking, and motion paths. As data and computing grew, the field shifted to deep learning systems that learn features directly. Mixed CNN–RNN models took center stage, enabling networks to learn spatiotemporal patterns from raw video data.

Societal and Technological Importance of SLR

Sign Language Recognition is more than a computer vision task. It is a social bridge that connects Deaf and hard-of-hearing (DHH) individuals to the digital and spoken world. Despite the global shift toward inclusive technologies, communication barriers persist across healthcare, education, and everyday interaction [19–21]. Traditional solutions often depend on interpreters or human intermediaries, which are not always available, affordable, or private. SLR systems promise direct, autonomous communication by translating sign gestures into text or speech in real time, reducing dependence and restoring autonomy to DHH individuals.

Challenges in Spatiotemporal Modeling for Word-Level Recognition

Sign language is by nature spatiotemporal. Each sign involves a structured combination of hand shape, movement trajectory, facial expression, and rhythm. Capturing these patterns in a machine learning model requires balancing spatial precision with temporal awareness. While CNNs can extract spatial features effectively, they struggle to preserve motion continuity across frames. Recurrent networks like LSTMs and GRUs, on the other hand, handle temporal dependencies but can be computationally expensive, especially for edge devices [19–21].

Word-level SLR presents unique challenges. Unlike isolated gesture recognition, words are defined by subtle motion dynamics and coarticulation between signs. Environmental factors, such as lighting variation, camera angle, and signer diversity, further complicate recognition. Data imbalance and signer-specific motion patterns also lead to poor generalization [20]. Moreover, models that rely on dense video input suffer from high redundancy, wasting computational resources on repetitive frames.

In the broader human–computer interaction (HCI) context, [10] emphasizes that SLR provides one of the most structured and semantically rich modalities for natural communication between humans and machines. Each gesture conveys a precise meaning, and accurate gesture recognition lays the foundation for intuitive HCI systems.

2.2. Modern Deep Learning Approaches in SLR

2.2.1. Static vs. Dynamic SLR

Sign languages appear as static or dynamic. Static SLR uses single images (frames) for alphabet or single signs. Dynamic SLR tracks changes over time, capturing movement between frames, much like recognizing continuous speech.

Early systems focused on static alphabets, but as computational capacity increased, dynamic word and sentence-level SLR became the main research direction. Recent works, such as [22], highlight the growing importance of temporal feature modeling. Their study evaluates CNN, LRCN, and ConvLSTM architectures on the WLASL dataset, showing that spatiotemporal fusion (via LRCN) outperforms purely spatial models.

 doi:10.20944/preprints202602.1848.v1

2.2.2. Convolutional and Recurrent Architectures: CNNs and RNNs.

Deep learning in SLR is centered on two main tools: Convolutional Neural Networks for spatial features, and Recurrent Neural Networks for temporal patterns. CNNs picked up hand shape, finger position, and motion boundaries from frame to frame. RNNs, especially LSTMs, tracked how signs changed over time [23,24]. Together, they let systems read sign language videos end to end.

From a benchmarking perspective, [25] systematically evaluates action recognition backbones (I3D, SlowFast, P3D, R(2+1)D) on MS-ASL. Their results, 92.35% top-1 accuracy, establish SlowFast ResNet101 as a strong baseline for word-level SLR. They emphasize the role of preprocessing (frame trimming, resampling, flipping) in dataset consistency and class balance.

For lightweight 3D architectures, [16] proposed in 2018 the S3D, a separable 3D convolution model that factorizes 3D kernels into spatial 2D and temporal 1D operations. S3D achieves near-I3D accuracy while reducing FLOPS by nearly 40%, providing a conceptual foundation for efficient spatio-temporal models.

Other works further enrich the landscape: [12] combine 3D CNN feature extractors with BiLSTMs for improved temporal awareness, achieving near-human-level accuracy (82.7%) on the Sports-1M dataset. Their hybrid residual network demonstrates that coupling convolutional and recurrent processing remains one of the most effective strategies for continuous sequence modeling. Along the same lines, [26] combines CNN and RNN modules for multimodal fusion of video and inertial data, achieving 97% accuracy on ASL, whereas [27] integrates the Reptile Search Algorithm into a CNN–LSTM framework, achieving 99.51% accuracy while enhancing computational efficiency. All these papers illustrate how hybrid deep learning remains the dominant paradigm in spatiotemporal SLR.

2.2.3. Hybrid and Multi-Stream Models

As research progressed, teams began combining multiple input streams and feature types to build more robust, richer models. Hybrid and multi-stream setups blend spatial, temporal, and skeletal cues, often using CNNs, LSTMs, and graph-based parts to capture different aspects of the signing process [24,28]. A key advantage of multi-stream systems lies in their ability to model heterogeneous input modalities.

Similarly, this multimodality is also reflected in [29], which integrates RGB, depth, and wearable sensor data for multimodal SLR using TensorFlow. The authors identify major obstacles: signer variability, limited annotated data, and real-time latency. Their preprocessing (noise reduction, background subtraction, and keyframe extraction) illustrates how data curation is just as important as model design.

In the same way, [7] demonstrates the power of depth-based LSTM architectures using Leap Motion sensors, reporting over 98% accuracy for sentence-level prediction. Depth data capture 3D motion trajectories that 2D cameras miss, making it valuable for dynamic gesture recognition.

Recent hybrids push this idea further. The ViT–CNN–BiLSTM fusion for Tamil Sign Language [28] uses CNNs for detailed spatial features, ViTs for global attention, and BiLSTMs for sequence modeling.

2.2.4. Attention Mechanisms and Transformer-Based Architectures

Attention mechanisms changed the field. They let models focus on the right spatio-temporal regions in a video, spotting key movements, and eliminating noise. This makes models easier to interpret and more efficient, especially for long, complex signing [23,30,31].

Transformers started in Natural Language processing, but now they excel in vision and multimodal SLR. RNNs process one step at a time, but Transformers use self-attention to scan all frames at once, capturing both local and global patterns [30,31]. This ability to model long-range dependencies makes them strong for continuous-sign recognition and translation.

 doi:10.20944/preprints202602.1848.v1

2.3. Model Compression and Optimization Techniques

Model compression makes TinyML possible. It enables complex networks to fit into just a few kilobytes or megabytes of memory.

Pruning droppes parameters or neurons that add little to the model output [32].

Quantization complements pruning. It shrinks models by lowering precision, from 32-bit floats down to 16-, 8-, or even 4-bit integers [33]. Smaller numbers mean smaller models, higher throughput, and lower energy use.

2.3.1. Pruning & Quantization

Edge AI systems must balance accuracy, latency, and energy. This matters most for real-time, human-focused tasks like sign language or gesture recognition. Pruning and quantization together can achieve 92.3% accuracy in ECG anomaly detection, while cutting power to just 0.024 mW on wearables [34]. Quantizing CNNs to INT8 enables nano drones to recognize gestures at 7 FPS, using only 72.5 kB of memory and achieving 90% accuracy [35]. These exemplify the need for the right model design and energy-aware training, showing that even tiny devices can deliver reliable inference.

[36] further discusses compression strategies such as model pruning, quantization, and low-rank factorization, along with algorithm–hardware co-design to ensure real-time feasibility on MCUs. A broader meta-analysis in [37] rates these techniques quantitatively.

2.3.2. Knowledge Distillation: Transferring Efficiency Across Networks

Knowledge distillation (KD) is a cornerstone of modern model compression and transfer learning. Rather than training a small network from scratch, a large model trains a smaller one [38,39].

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
[11]	Static	Custom glove sensor dataset	CNN-based TinyML model	Raspberry Pi Pico / TinyML runtime
[40]	Static (ASL digits)	Custom flex-sensor dataset	CNN, KNN, Naïve Bayes, Decision Tree	Arduino Uno (ATmega328P) / TinyML
[41]	Static (ASL digits 0–9)	64,000 RGB images	CNN + Residual/Dense Blocks, MobileNet V3	Raspberry Pi / TFLite (quantized)
[42]	Static (ASL alphabet)	Kaggle ASL A–Z dataset	TinyML CNN classifier on CSV images	MCU (unspecified) / TinyML pipeline
[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
[17]	Dynamic (ISL months)	Custom Indian SL (video)	Mediapipe keypoints + e-LSTM	Raspberry Pi 4 / TFLite (post-training INT8)
[45]	Dynamic (sensor HAR)	8 public HAR datasets	RNN, LSTM, GRU, FGRNN, FRNN, UGRNN	Raspberry Pi / TensorFlow Lite Micro
[36]	Conceptual review	–	–	TinyML tools (TFLM, NNoM, Edge Impulse, µTVM)
[46]	TinyML overview	–	–	Raspberry Pi Pico, ESP32, Arduino Nano 33 BLE Sense

 doi:10.20944/preprints202602.1848.v1

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 [35].
- Energy per inference (mJ or μW): total power consumption; sub-milliwatt operation is desired for continuous wearable or mobile use [33,34].

2.5. Frameworks and Industry Platforms for Edge AI

The success of TinyML hinges on optimized toolchains and firmware that bridge ML frameworks with embedded hardware.

[36] surveys practical deployment solutions including TensorFlow Lite Micro (TFLM), NNoM, Edge Impulse, and µTVM. They emphasize CMSIS-NN, Arm's low-level kernel library that accelerates core NN operations on Cortex-M CPUs. TFLM remains the dominant open-source runtime, translating 8-bit TensorFlow models into portable files executed by an interpreter. However, it still lacks GRU and Conv1D support and has limited profiling tools. NNoM, in contrast, compiles networks into plain C code, supports full RNN layers, and integrates CMSIS-NN for efficiency, but has a smaller community. Edge Impulse offers a proprietary end-to-end AutoML pipeline whose EON compiler reduces SRAM usage by 25–55 % but sacrifices transparency and portability.

From a hardware perspective, [46] catalogs the leading embedded vision boards used in TinyML research, Raspberry Pi 4, Arduino Portenta H7, Pixy2, Jetson Nano, OpenMV Cam H7, and Himax WE-I Plus. Also, identifies the most used batteries to provide power in embedded systems: nickel-metal battery (Ni-MH), sealed lead-acid battery (VRLA), Lithium-ion battery (Li-ion), and Lithium polymer battery (Li-Po).

These frameworks and hardware platforms constitute the first generation of deployable Edge-AI infrastructure, supporting the migration of SLR models from high-end GPUs to energy-aware microcontrollers.

2.6. Hardware-Aware Optimization: Tailoring Networks for Physical Constraints

Deep learning on embedded and edge devices demands models tailored to specific hardware. Hardware-aware optimization meets this need by combining profiling, architecture changes, and accelerator tuning. The goal is to balance accuracy, latency, and energy use [32,33,47]. TinyML, new accelerators, and smarter topology tools have turned this field into one grounded in real metrics and constraints.

Profiling Metrics and Edge-Aware Evaluation.

Performance evaluation in hardware-aware optimization extends beyond accuracy to include MAC and GOPS. They quantify computational workload; for example, AlexNet = 1.4 GOPS, and ResNet-152 = 22.6 GOPS [33].

2.7. Energy Efficiency and Privacy

Running sign language recognition on low-power edge devices means balancing efficiency, energy use, and privacy. Accuracy, power, and data protection set the ground rules for ethical edge AI. Recent work in neuromorphic and quantized neural processing, along with physical reservoir computing, points toward real solutions [48,49].

2.8. Incremental and Continual Learning for Expanding Sign Vocabulary

Incremental and continual learning solve a core problem in deep learning: models that cannot adapt to new data or classes without starting over. In SLR, this problem grows as new signs, variations, and signers keep entering the dataset. A recent study [50] compares incremental learning strategies for a skeleton- and transformer-based ISLR model on Peruvian Sign Language.

 doi:10.20944/preprints202602.1848.v1

2.9. Datasets and Benchmarks in SLR

Robust datasets are the backbone for evaluating deep learning models. [51] combines RGB video, IMU, and sEMG signals to form a multimodal continuous SLR dataset comprising 765 samples across nine activity types. Zhu et al. (2025) in [8] test their MAM-FSD model on RWTH, RWTHT, and CSL-Daily datasets, together exceeding 25,000 samples, and use Word Error Rate as the principal metric, reinforcing evaluation consistency across CSLR tasks.

At the word level, [25] employs MS-ASL to address signer diversity and background variation, achieving 92.35 % top-1 accuracy. They underline the scarcity of large, labeled data and the annotation costs specific to ASL.

[52] introduces the Sem-Lex Benchmark, 84,000 videos from native ASL signers aligned with linguistic databases such as ASL-LEX and SignBank. Using SL-GCN, they reach 85 % phonological feature recognition accuracy and highlight demographic bias as a persistent limitation in existing datasets.

Cross-lingual diversity is extended in [9], where models are evaluated on LSA64 (Argentine SL), AUTSL (Turkish SL), and WLASL (ASL). Similarly, [53] establishes WLASL as the largest public ASL corpus (2,000 glosses, 100+ signers), enabling scalable benchmarking for pose-based and appearance-based models.

2.9.1. Ethical and Reproducibility Issues

Ethical research with Deaf and hard-of-hearing participants involves integrity and partnership. Too often, studies have focused on Deaf communities without including them as equals. This stems from hearing-centered frameworks in fields like linguistics, medicine, and psychology, where the goals have not always aligned with Deaf cultural values. True ethical work means shifting the approach: Deaf participants become collaborators and knowledge creators, not just subjects. They help shape research design, data collection, and interpretation at every step [54].

Deaf communities are close-knit, so people can be recognized even without names or labels. Researchers need to keep careful records of how they handle data and get clear, informed consent for every use: collection, processing, storage, and sharing. Tools like signed video explanations or QR-coded consent forms help ensure participants understand and remain in control [55,56].

2.9.2. Dataset Challenges

Sign language recognition still faces core dataset issues. Four problems shape this landscape: insufficient data, insufficient signer diversity, inconsistent annotation, and shallow context.

SLR datasets, WLASL2000, and How2Sign offer only a few thousand sign classes, and most signers have fewer than ten examples each [57]. This imbalance impedes deep learning, especially for models that require temporal depth or a transformer-based architecture.

Building big sign language datasets takes trained annotators, linguistic skill, and cultural understanding. The process is time-consuming and expensive [58]. Data augmentation (temporal masking, frame swaps, synthetic frames helps a little, but it cannot replace real diversity across signers and domains.

Sign languages vary across countries, regions, and signers. This creates real variability. Speed, motion, facial expression, and hand dominance all can shift how a sign looks [58]. When models train and test on the same signers, they learn personal style rather than language. Academic benchmarks do not match the real world, where every user is new.

Annotation errors block reliable Sign Language Recognition. In WLASL2000, some gloss labels are incorrect due to language ambiguity and translation errors [57]. Most datasets do not include ground truth keypoints or landmarks for body parts. They label at the video level and skip frame-by-frame detail on occlusions, hand overlap, and facial movement [57,59]. Without this, models cannot separate pose errors from real motion. Keypoint-based models depend on outside pose estimators, which often

 doi:10.20944/preprints202602.1848.v1

miss detail in fast or overlapping gestures [59]. These gaps make it harder to reproduce, interpret, and analyze results.

Most datasets show isolated signs or single sentences. They miss the context and dialogue that real communication needs [60]. Benchmarks like PHOENIX-2014T and CSL-Daily stick to narrow topics (weather, scripts, set phrases). This makes training and testing easier, but it limits models' ability to learn grammar, references, or discourse. In real signing, meaning depends on context, topic flow, and conversation. Sentence-level data cannot capture that.

PTQ has taken a leap with smarter calibration data selection. Random sampling often leads to activation mismatch and reduces accuracy. The SelectQ method [61] fixes this by clustering activations and picking better calibration samples. It achieved over 15% higher accuracy for 4-bit ResNet-18 on ImageNet. This shift underscores the need for robust quantization to be well-calibrated, not just numerical schemes.

2.10. Identified Gaps and Design Opportunities

This review shows clear gaps and missed design opportunities for low-power edge deployment. Below we synthesise 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.

 doi:10.20944/preprints202602.1848.v1

- 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

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

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. [16]. 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.

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, and
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.

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.

 doi:10.20944/preprints202602.1848.v1

Table 2. Notation Summary.

Symbol	Description
Video Input and Indices	
X_t	Video frame at time step t
X	Input video tensor, $X \in \mathbb{R}^{T\times H\times W\times C}$
t	Temporal index, $t \in \{1,\ldots,T\}$
T	Temporal length (number of frames per video)
H, W	Spatial height and width of each frame
C	Number of input channels per frame
K	Number of target classes (vocabulary size)
Model Parameters and Layers	
f_θ	Neural network classifier parameterized by θ
θ	Trainable floating-point model parameters
θ^*	Optimized parameters after training convergence
θ_q	Quantized INT8 model parameters
$W^{(l)}, b^{(l)}$	Weight tensor and bias vector of layer l
r	Kernel size (spatial or temporal, depending on layer)
$*$	Convolution operator (2D or 1D depending on context)
$\sigma(\cdot)$	ReLU activation function, $\sigma(x) = \max(0, x)$
$\mathrm{MaxPool}(\cdot)$	Spatial max-pooling operator
Feature Representations	
$f_t^{(1)}$	Output feature map of first Conv2D–ReLU–pool block at time t
$f_t^{(2)}$	Output feature map of second Conv2D–ReLU–pool block at time t
s_t	Flattened spatial feature vector at time t
S	Temporal sequence of spatial embeddings, $S = [s_1,\ldots,s_T]$
d	Dimensionality of flattened spatial embedding s_t
d'	Number of temporal filters / temporal embedding dimension
h	Temporal feature map after Conv1D, $h \in \mathbb{R}^{T\times d'}$
h_t	Temporal feature vector at time step t
z	Temporally pooled feature representation after global average pooling
$\hat{y}$	Predicted class probability vector, $\hat{y} \in \Delta^{K-1}$
Complexity and Resource Metrics	
$C_{\text{in}}, C_{\text{out}}$	Input and output channel counts of a convolutional layer
$H_{\text{out}}, W_{\text{out}}$	Spatial resolution of a layer output
$\mathbf{A}_\ell$	Activation tensor of layer ℓ
$\lvert\mathbf{A}_\ell\rvert$	Number of elements in activation tensor $\mathbf{A}_\ell$
$\mathrm{Params}(\cdot)$	Number of learnable parameters of a layer
$\mathrm{MACs}(\cdot)$	Number of multiply–accumulate operations of a layer
FLOPs	Floating-point operations, FLOPs $= \alpha \cdot$ MACs
α	MAC-to-FLOP conversion factor, $\alpha \in \{1, 2\}$
$\lvert\theta\rvert$	Total number of learnable parameters in the model
$\lvert\theta_{\text{feat}}\rvert$	Parameters of the spatiotemporal extractor (Conv2D + Conv1D)
θ_{cls}	Parameters of the final dense classification layer
$\mathcal{O}(\cdot)$	Big-$\mathcal{O}$ asymptotic complexity notation

 doi:10.20944/preprints202602.1848.v1

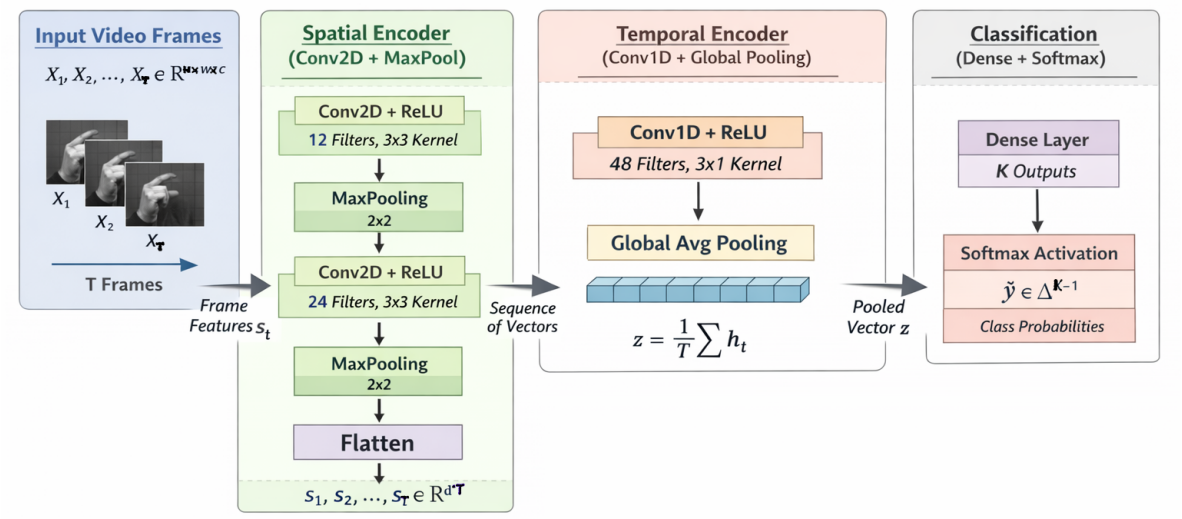

Figure 2. S3D-Conv1D workflow diagram.

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,

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

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:

$$f_t^{(2)} = \text{MaxPool}_{2\times 2}\Big(\sigma\big(W^{(2)} * f_t^{(1)} + b^{(2)}\big)\Big), \tag{3}$$

with

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

yielding

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

Flattening and Spatial Embedding.

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

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

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

 doi:10.20944/preprints202602.1848.v1

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

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:

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

where

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

and

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

Global Temporal Pooling.

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

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

where

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

Classification (Dense Softmax Head).

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

$$\hat{y} = \mathrm{softmax}\big(W^{(4)} z + b^{(4)}\big), \tag{7}$$

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.

 doi:10.20944/preprints202602.1848.v1

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:

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

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:
$W^{(3)} \in \mathbb{R}^{3\times d\times 48}$ (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:

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

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

Flatten features:

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

end for
Construct temporal sequence:

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

Temporal encoding:

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

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:

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

return $\hat{y}$

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.

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

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

 doi:10.20944/preprints202602.1848.v1

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

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

Dense Classification Layer.

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

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

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

$$|\mathbf{A}| = H_{\text{out}} W_{\text{out}} C_{\text{out}}. \tag{11}$$

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

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

with activation size

$$|\mathbf{A}_{\text{1D}}| = T d'. \tag{12}$$

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).

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

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

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

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

Dense Layer MACs.

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

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

 doi:10.20944/preprints202602.1848.v1

4.4. Numerical Cost Assessment

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

Configuration.

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

Unified Cost Summary.

As presented in Table 3, 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.

Table 3. 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	$3^2 \cdot 12 \cdot 24 + 24 = 2{,}616$	$32 \times 32 \times 24 = 24{,}576$	$3^2 \cdot 12 \cdot 24 \cdot 32^2 = 2{,}654{,}208$ per frame
Spatial Conv2D ($T = 24$)	2,736	–	$24 \times (442{,}368 + 2{,}654{,}208) = 74{,}317{,}824$
Temporal Conv1D	$3 \cdot 6144 \cdot 48 + 48 = 884{,}784$	$24 \times 48 = 1{,}152$	$3 \cdot 6144 \cdot 48 \cdot 24 = 21{,}233{,}664$
Dense	$48 \cdot 100 + 100 = 4{,}900$	100	$48 \cdot 100 = 4{,}800$
Total	**892,420**	**49,152**	**95.6 million**

Compute Cost.

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

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 [62]. 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 [63]. Time complexity measures the number of elementary operations required for inference and is expressed using Big-$\mathcal{O}$ notation to describe asymptotic growth [64].

Impact of vocabulary size K and temporal length T.

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.

 doi:10.20944/preprints202602.1848.v1

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.

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 subsection 4.1). Therefore,

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

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 subsection 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.

Worst-case space complexity.

Total inference memory combines parameter storage and peak activations:

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

Thus,

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

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 subsection 4.3. Since S3D-Conv1D is a feed-forward pipeline, the total cost is the sum of per-layer costs.

 doi:10.20944/preprints202602.1848.v1

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

$$\text{MACs}_{\text{spatial}} = T \cdot C_{\text{spatial}}. \tag{19}$$

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:

$$\text{MACs}_{\text{temporal}} = \mathcal{O}(T). \tag{20}$$

Classification.

The final dense layer maps the embedding to K classes:

$$\text{MACs}_{\text{classifier}} = \mathcal{O}(K). \tag{21}$$

Total inference time.

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

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

Hence,

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

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

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

Experiments are performed on two public sign language benchmarks, WLASL dataset [53] and the SemLex benchmark dataset [52], together with a small custom dataset.

The curated WLASL100 dataset is used as the main benchmark. It contains 2 181 videos covering 100 classes. A small custom dataset, ASL–Kimpinde, is created and added to include rare glosses and to support quantization calibration with our domain-specific data, such as skin tone and background conditions. This dataset contains 250 videos from 50 classes, recorded by a single signer under controlled conditions using a standard laptop webcam and a fixed capture protocol. After augmentation, the combined datasets contain 11 194 samples.

Generalization performance is evaluated with SemLex (2023) benchmark dataset using its official release. Only the curated training split is considered, which includes 956 valid videos across 100 classes.

 doi:10.20944/preprints202602.1848.v1

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.

Stratified dataset splitting is used to preserve class distributions and reduce bias toward frequent glosses. Overall, the preprocessing pipeline produces compact, balanced, and deployment-ready datasets designed for efficient spatiotemporal modeling on embedded platforms.

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

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 these Key settings:

Table 4. Training configuration and hyperparameters for S3D-Conv1D.

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	`tf.data` with shuffle, cache, batch, prefetch

 doi:10.20944/preprints202602.1848.v1

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.

Algorithm 3. This algorithm defines the core learning procedure of S3D-Conv1D, combining frame-wise spatial encoding with temporal convolution and global pooling.

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

Require: Preprocessed dataset $\tilde{\mathcal{D}}$, 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 θ^*, $\mathcal{M}$

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.

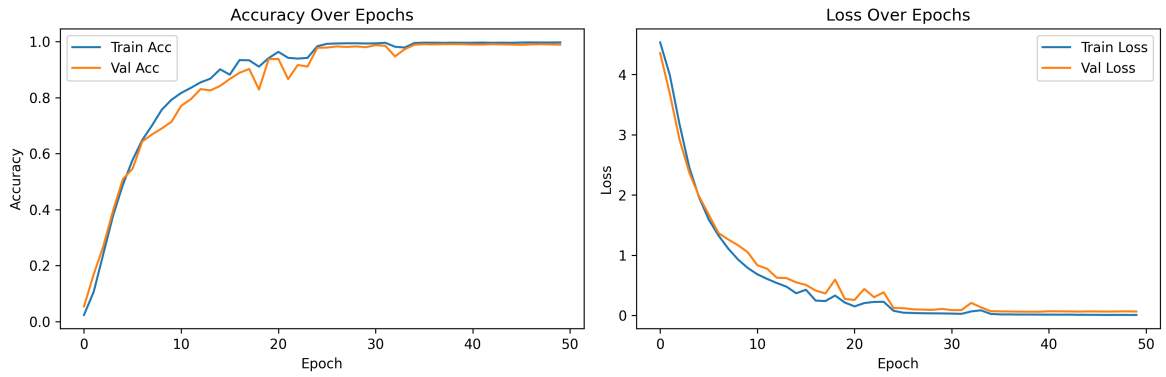

Figure 3. S3D-Conv1D Training Plots.

6.3. Quantization

As established in the TinyML literature, integer quantization provides the most effective trade-off between accuracy, efficiency, and footprint within kilobyte-scale budgets.

Post-training quantization converts 32-bit floating-point weights and activations into 8-bit integers, reducing model size by approximately 4× while enabling faster and more energy-efficient integer arithmetic on targets such as ARM Cortex-M cores. In this work, full-integer quantization is employed: all weights, activations, inputs, and outputs are represented in INT8, ensuring inference runs entirely in integer space with no floating-point operations.

Quantization is performed using TensorFlow Lite with `Optimize.DEFAULT`, restricted to integer-based operations (int8) only, and configured for integer-only input and output tensors. Weights and activations are quantized using per-tensor or per-channel affine scaling, balancing numerical precision

 doi:10.20944/preprints202602.1848.v1

and memory efficiency. Inputs are pre-quantized, while INT8 logits are optionally dequantized for probability estimation when required.

Accurate calibration is critical to preserve spatiotemporal dynamics. Activation ranges are determined using a representative dataset that captures both spatial and temporal statistics. Calibration is performed on the augmented `X_test` split of the combined WLASL and ASL–Kimpinde datasets, ensuring domain fidelity and robust range estimation.

Trained Keras models are first exported in FP32 format and then converted to INT8 using the TensorFlow Lite converter. Both floating-point and quantized models are validated with the TensorFlow Lite Interpreter to verify accuracy retention and numerical stability, including softmax consistency.

Algorithm 4. This algorithm applies post-training INT8 quantization and evaluates numerical robustness and deployment readiness through accuracy, footprint, and hardware-oriented metrics.

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
Export trained model to TensorFlow Lite format
Apply affine INT8 post-training quantization using $\mathcal{R}$
Restrict operators to TFLite INT8 built-ins
Export quantized model f_{θ_q}
Evaluate f_{θ_q} on $\mathcal{T}$
Compute accuracy and macro F1-score
Estimate operation count via `tflite_flops` [65]
Measure INT8 model size and binary footprint
Measure INT8 inference latency (20-run average) and throughput
Perform MCU-level deployability analysis
return f_{θ_q}, quantized evaluation metrics

6.4. Results

This section presents a consolidated evaluation of recognition performance, class-wise behavior, cross-dataset generalization, and quantization–compression effects across all benchmarked models (S3D, CNN+RNN, e-LSTM and S3D-Conv1D). Results are reported on WLASL100 + ASL–Kimpinde and SemLex100 datasets under identical preprocessing, training, and evaluation conditions.

6.4.1. Float32 Recognition Performance

Table 5 summarizes float32 accuracy, precision, recall, and macro F1-scores across both datasets. S3D, CNN+RNN, and the proposed S3D-Conv1D model all achieve near-saturated performance ($\geq$ 98.5% accuracy) on WLASL dataset, confirming that high recognition rates are attainable under controlled lexical conditions. In contrast, e-LSTM underperforms substantially, indicating insufficient spatial modeling capacity.

Table 5. Float32 accuracy, precision, recall, and F1-score across datasets.

Model (Float32)	WLASL100 dataset				SemLex100 dataset			
	Accuracy (%)	Precision	Recall	F1	Accuracy (%)	Precision	Recall	F1
S3D	99.29	0.99	0.99	0.99	64.29	0.78	0.64	0.64
CNN+RNN	99.29	0.99	0.99	0.99	89.62	0.91	0.90	0.89
e-LSTM	44.20	0.75	0.44	0.53	59.78	0.75	0.60	0.64
S3D-Conv1D (ours)	98.48	0.99	0.98	0.98	82.48	0.84	0.82	0.82

Performance divergence becomes pronounced on SemLex100. CNN+RNN achieves the highest accuracy (89.6%), benefiting from explicit recurrent temporal modeling, while S3D-Conv1D follows closely at 82.5%, outperforming both S3D and e-LSTM.

 doi:10.20944/preprints202602.1848.v1

6.4.2. Class-Wise Behavior and Failure Modes

Class-wise analysis (Figure 4) reveals robustness differences that are not visible in aggregate metrics. On WLASL100, three models achieve perfect recognition for at least one class (F1 = 1.00), while e-LSTM peaks lower at 0.88, reflecting weaker spatial encoding. Worst-class behavior differentiates the models: S3D and CNN+RNN degrade moderately (minimum F1 $\geq$ 0.78), whereas e-LSTM collapses entirely on at least one class (F1 = 0.00). The proposed S3D-Conv1D maintains a controlled lower bound (F1 $\approx$ 0.75), avoiding catastrophic failure.

On SemLex100, increased semantic overlap amplifies these effects. Although all models still achieve F1 = 1.00 on at least one class, worst-class performance diverges sharply. S3D and S3D-Conv1D drop to F1 $\approx$ 0.20, CNN+RNN retains higher robustness (F1 $\approx$ 0.50), and e-LSTM again collapses to F1 = 0.00.

Overall, class-wise results show that high mean accuracy can mask severe per-class failures. Architectures with heavy spatiotemporal coupling or compact recurrent representations exhibit breakable behavior on challenging signs, while S3D-Conv1D remains competitive across datasets despite the simplicity of its arcchitecture.

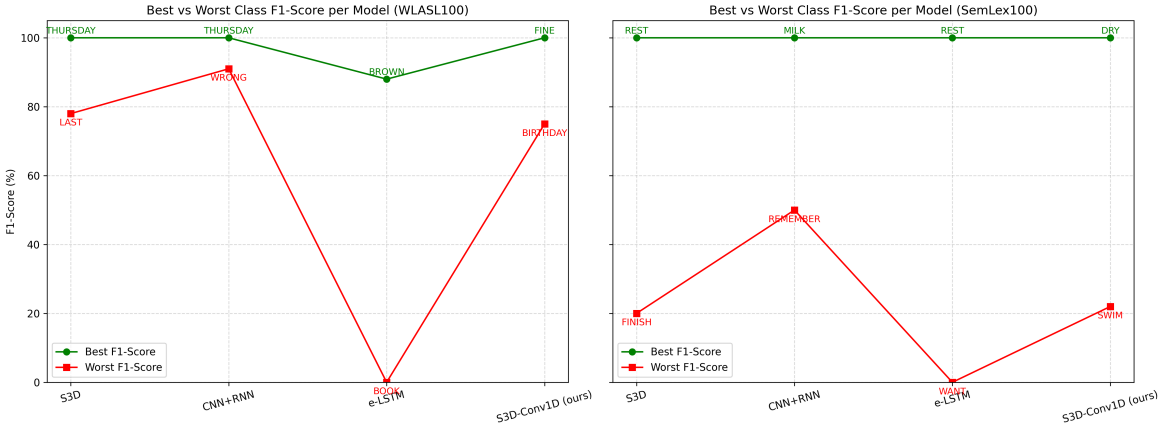

Figure 4. Classwise F1-Score.

6.4.3. Cross-Dataset Generalization

Cross-dataset evaluation exposes robustness differences that are not evident from single-dataset accuracy. Models that nearly saturate WLASL100 do not necessarily transfer well to SemLex100, which exhibits greater semantic overlap and intra-class variability.

The full S3D model suffers a pronounced degradation, with an accuracy drop exceeding 35%, indicating sensitivity to dataset-specific spatiotemporal statistics. The e-LSTM baseline shows inconsistent transfer behavior, while CNN+RNN generalizes well but relies on recurrent operators that are less suitable for embedded deployment.

S3D-Conv1D demonstrates the most stable degradation among deployable architectures, preserving a strong balance between spatial discrimination and temporal abstraction.

6.4.4. Quantization Robustness

INT8 results are reported in Table 6. S3D and CNN+RNN preserve accuracy under mixed or hybrid quantization, but rely on partial floating-point execution. e-LSTM exhibits severe quantization sensitivity, with WLASL100 accuracy collapsing to 20.1 %, indicating numerical instability in low-precision recurrent representations.

In contrast, S3D-Conv1D maintains accuracy within a narrow tolerance ($\Delta < 1$ % on WLASL100; about 2.6 % on SemLex100) under full INT8 execution, confirming that its operator homogeneity and temporal Conv1D formulation are inherently quantization-friendly.

 doi:10.20944/preprints202602.1848.v1

Table 6. INT8 accuracy, precision, recall, and F1-score across datasets.

Model (INT8)	WLASL100 dataset				SemLex100 dataset			
	Accuracy (%)	Precision	Recall	F1	Accuracy (%)	Precision	Recall	F1
S3D	99.20 (mixed)	0.99	0.99	0.99	60.38 (mixed)	0.78	0.64	0.64
CNN+RNN	99.29 (hybrid)	0.99	0.99	0.99	89.62 (hybrid)	0.91	0.90	0.89
e-LSTM	20.09 (INT8)	0.35	0.20	0.23	56.95 (INT8)	0.73	0.57	0.61
S3D-Conv1D	98.30 (INT8)	0.98	0.98	0.98	79.92 (INT8)	0.82	0.80	0.80

6.4.5. Compression and Model Size

Table 7 and Figure 5 summarize compression behavior. While S3D achieves negligible size reduction due to operator constraints, CNN+RNN and e-LSTM compress by approximately four times but remain prohibitively large. S3D-Conv1D compresses from 3.5 MB to under 0.9 MB (× 3.9) while preserving competitive accuracy, resulting in the smallest and most stable deployable model.

Table 7. Accuracy and model size comparison before and after quantization (WLASL100).

Model	Acc. Float32 (%)	Acc. INT8 (%)	Δ Acc. (%)	Size Float32 (KB)	Size INT8 (KB)	Compres. Ratio
S3D	99.29	99.20	-0.09	31,354.88	31,023.00	× 1.0
CNN+RNN	99.29	99.29	0.00	24,954.88	6,727.00	× 3.7
e-LSTM	44.20	20.09	-24.11	30,883.84	7,914.00	× 3.9
S3D-Conv1D (ours)	98.48	98.30	-0.18	3,501.00	895.90	× 3.9

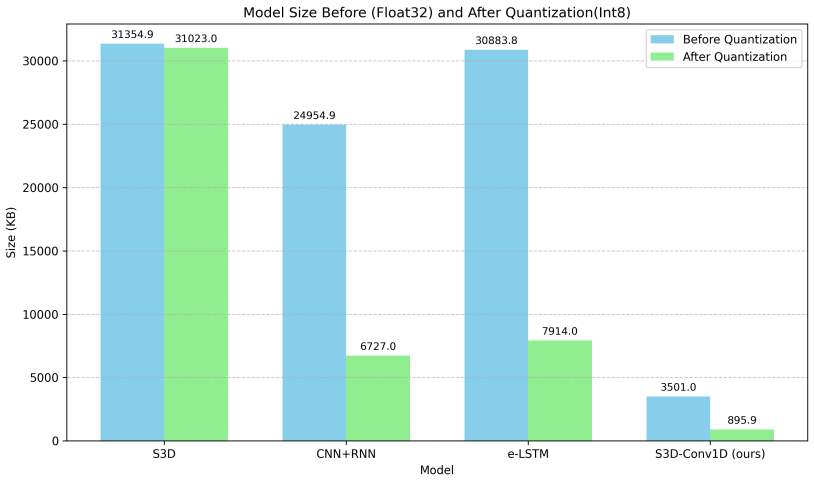

Figure 5. Model size comparison.

6.4.6. Profiling Pipeline

To contextualise deployment-oriented profiling, inference benchmarks were executed on a lightweight CPU-only laptop (Intel(R) Celeron(R) N4500 @ 1.10 GHz). While architecturally distinct from microcontroller cores such as the STM32U585 Cortex-M33, this modest hardware avoids GPU acceleration and provides latency behaviour indicative of TinyML-class targets.

The Table 8 below shows the core deployment metrics measured on this laptop runner. Sizes correspond to TFLite binary sizes obtained post-export; FLOPs are tflite_flops [65] estimates in MFLOPs; latency represents the median per-inference runtime across 20 executions; and the energy proxy is computed as CPU utilisation (%) multiplied by latency (arbitrary units), providing a stable comparative indicator of computational cost under uniform conditions.

The S3D model shows a very large latency due to run-time reliance on TF Select ops and a non-optimized graph. CNN+RNN is computationally heavier (1.27 GFLOPs) and slower than our

 doi:10.20944/preprints202602.1848.v1

S3D-Conv1D in practice; e-LSTM is fast but underperforms accuracy-wise, and requires extra memory to handle Mediapipe dependencies.

Table 8. Model deployment and runtime summary (measured on laptop).

Model	Params (from TFLM)	Model size (KB)	FLOPs (10^6)	Latency (ms)	FPS	Energy proxy
S3D	8,027,492	31,022.8	0.20	9,126.35	0.11	912,634.7
CNN+RNN	6,389,604	6,726.4	1,267	84.89	11.78	1,655.44
e-LSTM	7,906,212	7,913.8	0.60	39.29	25.45	2,448.08
S3D-Conv1D (ours)	892,420	970.2	191.11	23.67	42.25	1,559.87

S3D-Conv1D is the only architecture that delivering the best trade-off with a high recognition accuracy, low latency, and model size for on-device deployment.

6.4.7. Operator-Level Deployability Constraints

A neural network model is deployable on embedded hardware only if all its constituent operations are natively supported by the target inference libraries. The use of unsupported operators or fallback mechanisms (such as Flex operators in TensorFlow Lite) prevents true on-device execution and typically exceeds the memory and latency budgets of microcontrollers.

Table 9 summarizes the compatibility of the core operations used in this work with three representative embedded inference stacks: TensorFlow Lite [14], CMSIS-NN [15], and NNoM [66]. These libraries cover a broad spectrum of microcontroller-class deployments, ranging from ARM Cortex-M CPUs to lightweight vendor-agnostic runtimes.

Table 9. Operation compatibility with TensorFlow Lite, CMSIS-NN, and NNoM.

Operation	TensorFlow Lite	CMSIS-NN	NNoM
Conv2D (Quantized)	Yes (INT8 support)	Yes (optimized kernels)	Yes (fully supported)
DepthwiseConv2D	Yes	Yes (optimized for ARM)	Yes (supported since v1.1)
Conv3D (Factorized / Separable)	Partial (via Conv2D + Conv1D split)	Partial (via Conv2D emulation)	Partial (via stacked Conv2D + Conv1D)
Conv1D	Yes	Yes (optimized kernels)	Yes (via Conv2D wrapper)
MaxPooling2D	Yes	Yes (optimized)	Yes (supported)
AveragePooling2D	Yes	Yes	Yes (supported)
GlobalAveragePooling1D/2D	Yes	Yes	Yes (GAP layer)
Flatten	Yes	Yes	Yes
Dense (Fully Connected)	Yes (INT8)	Yes	Yes
Softmax	Yes	Yes (INT8)	Yes
GRU / LSTM	Yes (FlexOps / unrolled)	No	Partial (custom)
BatchNormalization	Yes (fused)	No (folded offline)	Yes
ReLU / ReLU6	Yes	Yes	Yes
Dropout	Ignored at inference	Ignored	Ignored
Reshape / Permute	Yes	Yes	Yes
Temporal Conv1D / TCN	Yes	Partial	Yes

6.4.8. End-to-End Deployability Assessment

To consolidate operator support, quantization feasibility, runtime performance, and energy behavior, Table 10 presents a deployability checklist across all evaluated architectures.

Table 10. Deployability checklist across candidate architectures.

Metric	S3D-Conv1D	CNN+RNN	e-LSTM	S3D (full)
Full INT8	Yes	No	No	No
Flex Ops required	**No**. Fully supported INT8 kernels	**Yes**. GRU requires NPU or FlexOps	**Yes**. Unsupported on MCUs	**Yes**. BatchNorm unavailable on MCUs
Real-time ($\leq$ 50 ms)	Yes	Borderline	No	No
Energy (qualitative)	Low	Moderate	High	Very High
Deployability rating	Ready	Needs NPU	Not feasible	Not deployable

7. Conclusions

This study investigated the feasibility of performing accurate word-level sign language recognition entirely on microcontroller-class hardware, without reliance on cloud computation, external

 doi:10.20944/preprints202602.1848.v1

accelerators, or unsupported operators. To address these constraints, we proposed S3D-Conv1D, a separable spatiotemporal architecture explicitly designed for embedded deployment, and evaluated it against separable S3D, CNN+RNN, and e-LSTM models using a unified training, quantization, and profiling framework.

Experiments on WLASL100 and SemLex100 show that S3D-Conv1D attains competitive float32 accuracy with stable per-class behavior and improved cross-dataset generalization. Following post-training INT8 quantization, the model maintains near-equivalent accuracy while achieving approximately 4× parameter reduction, a sub-megabyte binary size, and fully integer execution, capabilities not jointly satisfied by the compared baselines.

System-level profiling corroborates these results. Although separable S3D offers strong accuracy, its depth and operator complexity result in excessive latency and energy cost. CNN+RNN models remain computationally intensive and depend on Flex operators or NPUs, while e-LSTM, despite low latency, suffers from reduced accuracy and poor quantization robustness. In contrast, S3D-Conv1D consistently achieves real-time inference, low MAC counts, reduced activation memory, and full INT8 compatibility, relying exclusively on natively supported operators across TensorFlow Lite, CMSIS-NN, and NNoM.

These empirical findings are supported by the accompanying complexity analysis. By decoupling spatial feature extraction, temporal modeling, and classification, S3D-Conv1D exhibits linear time complexity with respect to sequence length and linear space complexity with respect to vocabulary size, with vocabulary growth confined to the final classification layer. This bounded scaling ensures predictable memory usage, stable latency, and straightforward vocabulary expansion, properties essential for deterministic microcontroller deployment.

In short, the results demonstrate that TinyML-ready word-level sign language recognition is achievable when architectural design prioritizes quantization stability, operator support, and hardware constraints. S3D-Conv1D illustrates that a carefully constrained convolutional temporal backbone can surpass more expressive recurrent models once real deployment limitations are enforced, providing a practical and reproducible path toward fully embedded sign language recognition.

7.1. 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.

7.2. 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.

 doi:10.20944/preprints202602.1848.v1

7.3. Future Research Directions

Future work should extend S3D-Conv1D toward continuous signing, integrate multimodal inputs such as depth, IMU, or skeletal data, and explore quantization-aware training to further stabilize 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.

7.4. Final Remarks

This dissertation 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, Samuel Longwani Kimpinde and Peter Olukanmi; methodology, Samuel Longwani Kimpinde; investigation, Samuel Longwani Kimpinde; validation, Peter Olukanmi; writing—original draft preparation, Samuel Longwani Kimpinde; supervision, Peter Olukanmi. 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; WLASL2000-Resized is available at https://www.kaggle.com/datasets/sttaseen/wlasl2000-resized; and the SemLex Benchmark dataset can be accessed at https://github.com/leekezar/SemLex. All datasets 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), Microsoft Copilot, and NotebookLM (Google) 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:

SLR	Sign Language Recognition
ASL	American Sign Language
WSLR	Word-level Sign Language Recognition
DHH	Deaf and Hard-of-Hearing
MCU	Microcontroller Unit
TFLM	TensorFlow Lite for Microcontrollers
CMSIS-NN	ARM Cortex Microcontroller Software Interface Standard for Neural Networks
TinyML	Machine Learning on Tiny/Resource-Constrained Devices
Conv1D	One-Dimensional Convolution
Conv2D	Two-Dimensional Convolution

 doi:10.20944/preprints202602.1848.v1

References

1. Organization, W.H. Deafness and hearing loss. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss, 2023. Accessed: 2025-10-18.
2. 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. *Higher Education Research & Development* **2025**, *0*, 1–17, [https://doi.org/10.1080/07294360.2025.2515214]. https://doi.org/10.1080/07294360.2025.2515214.
3. 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. *Lecture Notes in Networks and Systems* **2021**, *260*, 498 – 505. Cited by: 0, https://doi.org/10.1007/978-3-030-80829-7_62.
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 and Social Care in the Community* **2025**, *2025*. https://doi.org/10.1155/hsc/8867224.
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. *Frontiers in Public Health* **2021**, *9*. Cited by: 7; All Open Access, Gold Open Access, Green Open Access, https://doi.org/10.3389/fpubh.2021.615474.
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. *Journal of Public Health* **2024**, *46*, e520–e527, [https://academic.oup.com/jpubhealth/article-pdf/46/3/e520/58915014/fdae112.pdf]. https://doi.org/10.1093/pubmed/fdae112.
7. Kumar, N.; Ahmed, R.; H, V.B.; Salvi, S.; Panjwani, Y. Continuous Sign Language Recognition Using Leap Motion Sensor. IEEE, pp. 1160–1165. https://doi.org/10.1109/aic61668.2024.10730854.
8. Zhu, Q.; Li, J.; Yuan, F.; Gan, Q. Continuous sign language recognition based on motor attention mechanism and frame-level self-distillation. *Machine Vision and Applications* **2025**, *36*. https://doi.org/10.1007/s00138-024-01633-0.
9. Tang, H.Q.; Vo, T.D.N.; Tran, T.P.; Pham, D.D. Two-Stream S3D Architecture for Word-Level Sign Language Recognition, 2024. https://doi.org/10.1145/3654522.3654559.
10. Khomami, S.A.; Shamekhi, S. Persian sign language recognition using IMU and surface EMG sensors. *Measurement* **2021**, *168*, 108471. https://doi.org/10.1016/j.measurement.2020.108471.
11. B, S.K.; P, R.; Hiremath, R.B.; Ramadurgam, V.S.; Shaw, D.K. Survey on implementation of TinyML for real-time sign language recognition using smart gloves. IEEE. https://doi.org/10.1109/icerect56837.2022.10060135.
12. Shi, X.; Jiao, X.; Meng, C.; Bian, Z. 3D Sign language recognition based on multi-path hybrid residual neural network. ACM. https://doi.org/10.1145/3529836.3529943.
13. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D. Temporal Convolutional Networks for Action Segmentation and Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* **2017**, pp. 156–165.
14. TensorFlow Lite — Interpreter API. https://www.tensorflow.org/lite/. Accessed: 2025-10-18.
15. CMSIS-NN: Efficient Neural Network Kernels for ARM Cortex-M Processors. https://developer.arm.com/Tools%20and%20Software/Software%20Development%20Tools/CMSIS/CMSIS-NN. Accessed: 2025-10-18.
16. 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, 2018, pp. 305–321.
17. Sharma, V.; Sharma, A.; Saini, S. Real-time attention-based embedded LSTM for dynamic sign language recognition on edge devices. *Journal of Real-Time Image Processing* **2024**, *21*. https://doi.org/10.1007/s11554-024-01435-7.
18. Sandjaja, I.; Alsharoa, A.; Wunsch, D.; Liu, J. Survey of Hidden Markov Models (HMMs) for Sign Language Recognition (SLR). 2024. Cited by: 3, https://doi.org/10.1109/ICPS59941.2024.10640040.
19. Abhishek, Y.; Sumanathilaka, D. End-to-End Sign Language Recognition Pipeline: Towards Energy Efficient Modeling. *Procedia Computer Science* **2025**, *265*, 483–490. 20th International Conference on Future Networks and Communications/ 22nd International Conference on Mobile Systems and Pervasive Computing/15th International Conference on Sustainable Energy Information Technology (FNC/MobiSPC/SEIT 2025), https://doi.org/https://doi.org/10.1016/j.procs.2025.07.208.

 doi:10.20944/preprints202602.1848.v1

20. 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), 2025, pp. 1–5. https://doi.org/10.1109/INCET64471.2025.11139951.
21. 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), 2025, pp. 1–6. https://doi.org/10.23919/INDIACom66777.2025.11115571.
22. Das, S.; Yadav, S.K.; Samanta, D., Isolated Sign Language Recognition Using Deep Learning; Springer Nature Switzerland, 2024; pp. 343–356. https://doi.org/10.1007/978-3-031-58181-6_29.
23. Chinchmalatpure, S.; Matale, 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., Singapore, 2025; pp. 75–89.
24. 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), 2018, pp. 1–4. https://doi.org/10.1109/3DTV.2018.8478467.
25. Radhakrishnan, S.; Mohan, N.C.; Varma, M.; Varma, J.; Pai, S.N. Cross Transferring Activity Recognition to Word Level Sign Language Detection. IEEE, pp. 2445–2452. https://doi.org/10.1109/cvprw56347.2022.00273.
26. 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. IEEE. https://doi.org/10.1109/accai58221.2023.10199569.
27. 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. https://doi.org/10.1016/j.heliyon.2023.e23252.
28. 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), 2025, pp. 1–6. https://doi.org/10.1109/AIMLA63829.2025.11040563.
29. Astya, R.; Agrawal, A.; Chauhan, A.; Sony, A.K.; Thakur, K.R.; Nand, P. Enhancing Sign Language Detection using Tensor Flow. IEEE. https://doi.org/10.1109/icac3n60023.2023.10541807.
30. Xie, P.; Zhao, M.; Hu, X. PiSLTRc: Position-Informed Sign Language Transformer With Content-Aware Convolution. *IEEE Transactions on Multimedia* **2022**, *24*, 3908–3919. https://doi.org/10.1109/TMM.2021.3109665.
31. 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), 2025, pp. 1–6. https://doi.org/10.1109/GCON65540.2025.11173323.
32. Saint-Germain, L.; Le Gal, B.; Baldacci, F.; Crenne, J.; Jego, 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), 2022, pp. 1–6. https://doi.org/10.1109/SiPS55645.2022.9919244.
33. 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 Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **2024**, *17*, 15862–15884. https://doi.org/10.1109/JSTARS.2024.3452321.
34. Hizem, M.; Bousbia, L.; Ben Dhiab, Y.; Aoueileyine, M.O.E.; Bouallegue, R. Reliable ECG Anomaly Detection on Edge Devices for Internet of Medical Things Applications. *Sensors* **2025**, *25*. Cited by: 6; All Open Access, Gold Open Access, Green Open Access, https://doi.org/10.3390/s25082496.
35. 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), 2025, pp. 95–100. https://doi.org/10.1109/RCAR65431.2025.11139603.
36. Lê, M.T.; Wolinski, P.; Arbel, J. Efficient Neural Networks for Tiny Machine Learning: A Comprehensive Review **2023**. https://doi.org/10.48550/arxiv.2311.11883.
37. Dantas, P.V.; Sabino Da Silva, W.; Cordeiro, L.C.; Carvalho, C.B. A comprehensive review of model compression techniques in machine learning. *Applied Intelligence* **2024**, *54*, 11804–11844. https://doi.org/10.1007/s10489-024-05747-w.

 doi:10.20944/preprints202602.1848.v1

38. 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), 2024, pp. 1–7. https://doi.org/10.1109/INTERCON63140.2024.10833501.
39. 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), 2023, pp. 3990–3996. https://doi.org/10.1109/CAC59555.2023.10451266.
40. Kumar, S.; Poongodan, R.; Hiremath, R.B.; Ramadurgam, V.S.; Shaw, D.K. Implementation of sign language recognition with TinyML using smart gloves. AIP Publishing. https://doi.org/10.1063/5.0198896.
41. Yu, A.; Qian, C.; Guo, Y. A Real-Time Hand Gesture Recognition System on Raspberry Pi: A Deep Learning-Based Approach. IEEE. https://doi.org/10.1109/ccnc51664.2024.10454652.
42. A. Dabwan, B.; E. Jadhav, M.; A. Ismil, O.; A. Hassan, E.; A. Farah, E.; A. Mohammad, A.; Ali, Y.A. Detecting Gesture Language for Deaf and Mute People Using on Ultra-Low-Power TinyML Model. IEEE, pp. 1–5. https://doi.org/10.1109/icdsns62112.2024.10691053.
43. Trpcheska, A.; Zevnik, F.; Bader, S. Towards Real-Time Vision-Based Sign Language Recognition on Edge Devices. IEEE, Vol. abs/2201.09679, pp. 1–6. https://doi.org/10.1109/sas60918.2024.10636604.
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. IEEE. https://doi.org/10.1109/it61232.2024.10475765.
45. Lalapura, V.S.; Bhimavarapu, V.R.; Amudha, J.; Satheesh, H.S. A Systematic Evaluation of Recurrent Neural Network Models for Edge Intelligence and Human Activity Recognition Applications. *Algorithms* **2024**, *17*, 104. https://doi.org/10.3390/a17030104.
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. https://doi.org/10.3390/a17110476.
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 Robotic and Sensors Environments (ROSE), 2024, pp. 1–7. https://doi.org/10.1109/ROSE62198.2024.10591116.
48. Shingu, T.; Uchiyama, H.; Watanabe, T.; Ohno, Y. Electrochemical reservoir computing based on surface-functionalized carbon nanotubes. *Carbon* **2023**, *214*, 118344. https://doi.org/https://doi.org/10.1016/j.carbon.2023.118344.
49. Rutishauser, G.; Scherer, M.; Fischer, T.; Benini, L. 7 µJ/inference end-to-end gesture recognition from dynamic vision sensor data using ternarized hybrid convolutional neural networks. *Future Generation Computer Systems* **2023**, *149*, 717–731. https://doi.org/https://doi.org/10.1016/j.future.2023.07.017.
50. Huamani-Malca, J.; Bejarano, G. Comparing Incremental Learning Approaches for a Growing Sign Language Dictionary. *Communications in Computer and Information Science* **2024**, *2142 CCIS*, 97 – 106. Cited by: 0, https://doi.org/10.1007/978-3-031-63616-5_7.
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. IEEE, Vol. 37, pp. 310–315. https://doi.org/10.1109/eecr60807.2024.10607314.
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. Export Date: 11 June 2025; Cited By: 7, https://doi.org/10.1145/3597638.3608408.
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. IEEE. https://doi.org/10.1109/wacv45572.2020.9093512.
54. 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, 2015; chapter 1, pp. 5–20, [https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118346013.ch1]. https://doi.org/https://doi.org/10.1002/9781118346013.ch1.
55. Kolbe, V. OPEN SCIENCE VERSUS DATA PROTECTION – CHALLENGES AND SOLUTIONS IN SIGN LANGUAGE ACQUISITION STUDIES. *Hrvatska Revija Za Rehabilitacijska Istrazivanja* **2022**, *58*, 109 – 120. Cited by: 5; All Open Access, Gold Open Access, https://doi.org/10.31299/hrri.58.si.6.

 doi:10.20944/preprints202602.1848.v1

56. Parks, E. Listen First: Dialogic Research Ethics With Caribbean Signing Communities. *Ethics & Behavior* **2017**, *29*. https://doi.org/10.1080/10508422.2017.1395338.
57. 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. Cited by: 0; All Open Access, Gold Open Access, https://doi.org/10.1109/ACCESS.2025.3591123.
58. Liu, T.; Tao, T.; Zhao, Y.; Li, M.; Zhu, J. A signer-independent sign language recognition method for the single-frequency dataset. *Neurocomputing* **2024**, *582*, 127479. https://doi.org/https://doi.org/10.1016/j.neucom.2024.127479.
59. Guan, M.; Wang, Y.; Ma, G.; Liu, J.; Sun, M. MSKA: Multi-stream keypoint attention network for sign language recognition and translation. *Pattern Recognition* **2025**, *165*. Cited by: 6, https://doi.org/10.1016/j.patcog.2025.111602.
60. Liu, Y.; Zhang, W.; Ren, S.; Huang, C.; Yu, J.; Xu, L. SCOPE: sign language contextual processing with embedding from LLMs. AAAI Press, 2025, AAAI'25/IAAI'25/EAAI'25. https://doi.org/10.1609/aaai.v39i6.32612.
61. Zhang, Z.; Gao, Y.; Fan, J.; Zhao, Z.; Yang, Y.; Yan, S. SelectQ: Calibration Data Selection for Post-training Quantization. *Machine Intelligence Research* **2025**, *22*, 499 – 510. Cited by: 0, https://doi.org/10.1007/s11633-024-1518-0.
62. Wikipedia contributors. Big O notation. https://en.wikipedia.org/wiki/Big_O_notation, 2025. Accessed: 2025-12-18.
63. Wikipedia contributors. Space complexity. https://en.wikipedia.org/wiki/Space_complexity, 2025. Accessed: 2025-12-18.
64. Wikipedia contributors. Time complexity. https://en.wikipedia.org/wiki/Time_complexity, 2025. Accessed: 2025-12-18.
65. lisosia. tflite-flops: Roughly calculate FLOPs of a tflite model. https://github.com/lisosia/tflite-flops, 2021. Accessed: 2025-10-22.
66. Ma, J. A higher-level Neural Network library on Microcontrollers (NNoM), 2020. https://doi.org/10.5281/zenodo.4158710.
67. OpenMV. OpenMV AE3 – Python Programmable AI Camera for Embedded Vision. https://openmv.io/products/openmv-ae3. Accessed: June 28, 2025.
68. Arduino. Arduino UNO Q Documentation. https://docs.arduino.cc/hardware/uno-q/, 2025. Accessed: 2025-10-22.