Complex and Composite Human Activity Recognition Utilizing Wearable Sensor Data Focusing on Low-Power Low-Cost Low-Compute IoT Devices

by

ATHAR NOOR MOHAMMAD RAFEE
24366014

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
M.Sc. in Computer Science and Engineering

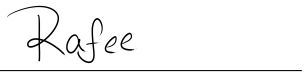
Department of Computer Science and Engineering
Brac University
August 2025

Declaration

It is hereby declared that

1. The thesis submitted is my own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. I have acknowledged all main sources of help.

Student's Full Name & Signature:



ATHAR NOOR MOHAMMAD RAFEE

24366014

Approval

The thesis titled "Complex and Composite Human Activity Recognition Utilizing Wearable Sensor Data Focusing on Low-Power Low-Cost Low-Compute IoT Devices" submitted by

1. ATHAR NOOR MOHAMMAD RAFEE (24366014)

of Fall, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of M.Sc. in Computer Science and Engineering on August 3, 2025.

Examining Committee:

Supervisor:
(Member)

Dr. Jannatun Noor

Associate Professor
Department of Computer Science and Engineering
Director, B.Sc. in Data Science Porgram
United International University

Examiner:
(External)

Dr. Mohammad Nurul Huda

Professor
Department of Computer Science and Engineering
School of Science and Engineering
United International University

Examiner:
(Internal)

Dr. Muhammad Iqbal Hossain

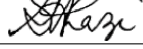
Associate Professor
Department of Computer Science and Engineering
School of Data and Sciences
Brac University

M.Sc. Program Coordinator:
(Member)

Dr. Md. Sadek Ferdous

Professor
Department of Computer Science and Engineering
School of Data and Sciences
Brac University

Head of Department:
(Chair)



Dr. Sadia Hamid Kazi

Associate Professor
Department of Computer Science and Engineering
School of Data and Sciences
Brac University

Ethics Statement

This note is a disclaimer to indicate that the research study in question has not violated the preservation of human rights, welfare or dignity. The honesty and integrity of the research participants are incorporated in the work, which also includes highlighted citations from various reliable sources. No discrimination or disparity has been made in the data collection, and the results have not been manipulated for any prejudiced reasons. Rest assured, I hope my work reflects my respect for all intellectual property and is useful in the long-term welfare of humanity.

Abstract

Human activity recognition is a foundational technology for applications like remote health monitoring, though accurately classifying complex and composite activities often necessitates cloud-based processing. This conventional approach, while powerful, introduces significant latency, power consumption, and privacy concerns, driving a trend toward on-device solutions. However, current on-device systems frequently overlook highly informative sensor modalities. Plantar pressure sensors, in particular, offer a distinct advantage by capturing rich kinetic data related to gait and balance unlike the kinematic data from standard inertial sensors. However, their application within knowledge distillation frameworks remains a significant, unexplored research gap. Hence, this study presents a cost-effective, low-computation system for complex and composite human activity recognition that leverages knowledge-distilled neural networks on a microcontroller unit to minimize reliance on cloud processing. A key contribution of this work is the investigation of plantar pressure sensor data within a knowledge distillation framework, addressing a notable gap in the existing literature. The proposed solution centers around the ESP32-S3 DevKit C1, equipped with a dual-core 240 MHz Tensilica chip, 320 KiB of usable static random access memory, and built-in Wi-Fi and Bluetooth. Significantly, both the teacher and the student models surpass existing state-of-the-art methods, achieving F1 Scores of 99.33%, 98.36%, and 97.68% respectively, in classifying a comprehensive set of 21 activities (15 composite and 6 simple) in CAPPIMU dataset. The distilled student models demonstrate remarkable efficiency, with execution times of 1.83 and 0.64 seconds, memory footprints of only 62 KB and 82 KB, and flash memory usage of approximately 209 KB and 127 KB, while maintaining low power consumption of 210 mW and 215 mW, respectively. Furthermore, validation on external public datasets confirms the generalizability of the proposed models. They not only match the performance of state-of-the-art methods but also exhibit superior computational efficiency, achieving an outstanding balance between accuracy and resource consumption. Lastly, we have developed an end-to-end prototype that integrates the ESP32-S3 with a WitMotion inertial measurement unit sensor. This system autonomously manages data acquisition, feature extraction, and inference in under 7 seconds with a total power consumption of approximately 295 mW. By enabling on-device data processing and inference, our approach significantly reduces dependency on cloud resources, making it highly suitable for power and computationally constrained applications such as remote health monitoring, fitness tracking, and smart home automation.

Keywords: TinyML; Embedded Systems; Microcontroller Units (MCUs); ESP32-S3; Hardware-Software Codesign; Human Activity Recognition (HAR); Composite Human Activity; Complex Human Activity; End-to-End System; AIoT; Knowledge Distillation (KD); Neural Networks; Plantar Pressure (PP); Inertial Measurement Unit (IMU)

Dedication

I wholeheartedly dedicate this M.Sc. thesis to the unwavering pillars of my life, my beloved parents, especially my father whose boundless support and encouragement have illuminated my academic journey, inspiring me to reach new heights. My sincere appreciation goes out to my devoted supervisor, whose continuous counsel and insight have been crucial to the development of this research. Finally, I dedicate this study to all the people who have contributed to the advancement of cyber physical systems and tiny machine-learning-related technologies, and to all the potential beneficiaries of this research, who may benefit from improved safety and well-being.

Acknowledgement

I owe my deepest gratitude to Allah, the Most Merciful, for granting me the strength, patience, and guidance to complete this research. His blessings have been the cornerstone of my academic journey and the driving force behind every milestone I have achieved. I would like to express my heartfelt appreciation to my supervisor, Dr. Jannatun Noor (Associate Professor, CSE, UIU). Her steady guidance, unwavering support, and insightful feedback have shaped this thesis from conception through to its final form. Her dedication to excellence and willingness to challenge me intellectually have elevated the quality of my work and inspired me to push beyond my limits.

My sincere thanks go to Md Abu Obaida Zishan for his thoughtful feedback during the early stage of this work. His meticulous attention to detail and constructive suggestions helped refine my arguments and improve the clarity of my presentation. I am profoundly grateful to my collaborator colleague and friend, John Clear IV of San José State University. From provisioning compute infrastructure to collaborating on formal analysis, hardware and software validation, and visualization, his contributions have been invaluable. John's generous assistance with multiple rounds of editing and his steadfast encouragement sustained me through every phase of this thesis work. In addition to that, I would also like to show my gratitude to the anonymous reviewers for providing invaluable feedback regarding the overall methodology of this study.

I would also like to acknowledge BRAC University for supporting my academic pursuits. The university's resources and funding enabled me to undertake this study, while its intellectual community provided an environment conducive to research and discovery. To the Department of Computer Science and Engineering, thank you for granting access to cloud infrastructure (Azure IoT Hub), software licenses, and technical support. Your facilities have been essential to the experimental and computational aspects of this thesis.

Finally, I believe, only I did not work for my M.Sc. thesis, rather all of my friends and family did the same. I am eternally thankful to my parents and siblings for their unconditional love, encouragement, and understanding. Their patience during the long hours of study, experimentation, and writing has been my greatest source of motivation and comfort.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document.

ADL	Activities of Daily Living
AI	Artificial intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
CP	Cerebral Palsy
DFS	Dynamic Frequency Scaling
DL	Deep Learning
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
DT	Decision Tree
FLOPs	Floating Point Operations
FPU	Floating-Point Unit
GCC	GNU Compiler Collection
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HAR	Human Activity Recognition

IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IoT	Internet of Things
JTAG	Joint Test Action Group
KD	Knowledge Distillation
LOSOCV	Leave-One-Subject-Out Cross-Validation
LSTM	Long Short-Term Memory
MCU	Microcontroller Unit
MFLOPs	Million Floating Point Operations
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
NPU	Neural Processing Units
OPS	Operations
PC	Personal Computer
PCB	Printed Circuit Board
PP	Plantar Pressure
RF	Random Forest
RISC	Reduced Instruction Set Computing
RNN	Recurrent Neural Network
RX	Receive
SD	Subject-Dependent
SDK	Software Development Kit

SoC	System-on-Chip
SRAM	Static Random Access Memory
TD	Typically Developing
TFLM	TensorFlow Lite Micro
TinyML	Tiny Machine Learning
TTL	Transistor–Transistor Logic
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

Chapter 1

Introduction

Human activity recognition has gained significant attention in recent years due to its wide range of applications in areas such as health monitoring, fitness tracking, and smart home automation. Recognizing activities such as walking, sitting, or running enables systems to offer personalized insights and human-centric automation that improves user experience and safety [125], [180], [192]. However, as the complexity of human activities increases, especially in complex and composite activity recognition [44] where multiple activities are performed concurrently or sequentially, the demand for computational resources also rises.

1.1 Motivation

Traditional Human Activity Recognition (HAR) systems often rely on cloud-based processing, where large datasets are transmitted to remote servers (fog or cloud) for analysis. While this approach leverages powerful computing infrastructure, it introduces several drawbacks, including latency, increased energy consumption due to constant wireless communication, and privacy concerns related to data transmission and storage [64], [113]. Furthermore, managing such a large-scale distributed cloud-based solution is increasingly complicated, costlier, and requires expertise. As a result, on-device end-to-end Tiny Machine Learning (TinyML) solution is getting attention like never before [118], [159].

The challenge lies in performing composite human activity recognition on resource-constrained devices, such as microcontrollers, where computational power, memory, and energy are severely limited [140]. With the advent of Internet of Things (IoT), there is a growing need for HAR systems that can process data locally on devices to reduce dependency on cloud servers. On-device processing not only reduces latency and network strain but also addresses privacy concerns by minimizing the need to transmit personal data [201]. However, implementing effective Deep Learning (DL) algorithms on low-power, low-cost Microcontroller Units (MCUs) remains a significant challenge due to hardware limitations. Many state-of-the-art HAR models are computationally intensive, involving complex architectures that are not feasible for MCUs, forcing the adoption of simpler architecture at the cost of accuracy.

1.2 Limitations

Current approaches to solving this problem largely rely on cloud or hybrid models, where raw sensor data is captured on the device and transmitted to a remote server for processing and analysis. Although this offloads the computation from the device to more powerful infrastructure, it introduces latency and requires constant connectivity, which is not ideal for many real-world scenarios. For applications like health monitoring and fitness tracking, where near-real-time feedback is crucial, relying on cloud processing may result in unacceptable delays. Additionally, the energy cost of transmitting data, especially in battery-operated devices, can significantly reduce operational time, making these systems impractical for long-term use in portable applications. Efforts to move HAR onto MCUs have mostly involved basic Machine Learning (ML) models with limited complexity [72], [159], often sacrificing accuracy or the number of classifiable labels to meet hardware constraints. These trade-offs between accuracy, detection capability, and resource usage present a significant barrier to widespread adoption of HAR related tasks.

To address these limitations, recent research has increasingly focused on Knowledge Distillation (KD) [135], [177] as a potent model compression technique. KD facilitates the transfer of learned representations, often referred to as "dark knowledge," from a large, complex Neural Network (NN) (the teacher) to a smaller, more computationally efficient NN (the student). This process enables the deployment of the compact student model on resource-constrained edge devices such as MCUs, single-board computers, and mobile phones, without a substantial loss in predictive performance. A particularly salient application of KD within the HAR domain is multi-sensor fusion [185]. In this paradigm, a sophisticated teacher model is trained on a rich dataset comprising inputs from multiple sensors. Subsequently, a student model is trained on a subset of this data, frequently from only a single sensor. This methodology significantly curtails the student model's complexity while allowing it to retain a significant portion of the teacher's comprehensive understanding, which was distilled from the multi-modal sensory inputs [135]. When the sensors involved capture information from different modalities (e.g., motion and physiological signals), this approach is more specifically termed cross-modal KD.

While existing literature has explored these KD techniques to some extent using conventional inertial sensors like Inertial Measurement Units (IMUs), which include accelerometers, gyroscopes, and magnetometers, a notable and highly informative modality remains largely unexplored in this context. Specifically, the Plantar Pressure (PP), which captures dynamic pressure distribution across the foot-ground interface, offers a rich source of data for analyzing gait, balance, and specific foot-related movements. Unlike IMUs that measure kinematic motion, PP sensors provide kinetic data that can reveal subtle nuances in activities such as walking, running, or standing up. Despite its potential to significantly enhance activity recognition accuracy and provide deeper biomechanical insights, this sensor modality has been conspicuously overlooked within the domain of HAR. The application of PP sensors in the broader HAR domain is itself nascent, with only a limited number of studies [132], [168], [182] investigating their potential, and often in a limited capacity.

Consequently, the application of KD-based methodologies, particularly cross-modal distillation, to PP sensor data represents a significant and unaddressed research gap. Furthermore, there is a distinct lack of comprehensive analysis employing both conventional ML and advanced DL models on this data modality. This gap underscores a compelling opportunity for novel research that could substantially advance the state-of-the-art in wearable HAR systems.

1.3 Our Contribution

As a result, in this study, we propose a prediction platform to complex and composite human activity recognition utilizing PP that is optimized for deployment on resource-constrained microcontroller platforms. Our key contributions are:

1. We design DualCrossTAP, a custom two-branch neural network that achieves 99.33% accuracy, surpassing existing state-of-the-art models in CAPPIMU dataset. Next, we propose two resource-efficient Convolutional Neural Network (CNN) models tailored for low-compute MCUs, balancing high performance with hardware constraints, while maintaining deployability on edge devices with stringent hardware constraints.

2. We adapt a standard KD method that integrates PP and IMU sensors, effectively transferring dark knowledge from the DualCrossTAP model to the student CNN models. The student models achieve up to 98.35% accuracy and up to a 23× compression ratio, outperforming all existing state-of-the-art models. Additionally, we design and propose an ultra light weight Multilayer Perceptron (MLP) model for highly compute constrained IoT devices where the inference latency is as low as 0.02 seconds.

3. To the best of our knowledge, this work is the first to systematically investigate the role of PP sensors within a KD-based sensor fusion context. We uniquely explore bidirectional knowledge transfer, examining distillation from PP to IMU as well as from IMU to PP, thereby addressing a significant research gap in the HAR domain.

4. Subsequently, we perform a comprehensive parameter and Million Floating Point Operations (MFLOPs) analysis on the recent state-of-the-art models, demonstrating our models' superior efficiency-performance balance. We then deploy the student CNN and stand alone MLP models on an ESP32-S3 MCU, showcasing its real-world viability for low-cost, low-power, and resource constrained applications.

5. Next, we demonstrate our models' real-world robustness and generalizability through rigorous validation on two distinct external benchmarks: the KU HAR and NTNU Children HAR datasets. This confirms that our models maintain their superior efficiency-performance balance across different sensor setups, activity types, and demographic groups, underscoring their broad applicability.

6. Finally, we develop a complete end-to-end prototype system for composite HAR, integrating the ESP32-S3 with a WitMotion IMU sensor. This prototype autonomously handles raw sensor data acquisition, on-device preprocess-

ing, feature extraction, and inference, delivering a prediction in under seven seconds while consuming only 295 mW of power.

1.4 Study Organization

The rest of the study is structured as follows: Chapter 2 reviews the relevant literature and compares recent advancement in the HAR domain with respect to TinyML, sensor utilization, practical deployability, and feasibility. Chapter 3 discusses the necessary background knowledge required to understand the rest of the study methods and overall decision-making process. Chapter 4 provides a detailed overview of the utilized datasets (CAPPIMU, KU HAR, NTNU Children HAR), including data preparation, preprocessing pipelines, and the rationale for the chosen train-test splitting strategies. In Chapter 5, we delineate the architecture of our proposed models, discuss the rationale behind key design decisions, and describe the employed KD approach, the selection of the MCU, and the relevant toolchains. Chapter 6 presents our experimental results and performance evaluations, with an emphasis on the ESP32-S3 deployment and a comprehensive discussion of pertinent deployment metrics. Next, Chapter 7 introduces our end-to-end prototype system that integrates the ESP32-S3 with the WitMotion IMU sensor, elaborating on the decision-making process regarding sensor integration, power consumption, and practical implementation challenges. Chapter 8 thoroughly compares our proposed models and our developed end-to-end prototype system against state-of-the-art models and end-to-end HAR systems proposed in recent literatures. Finally, Chapter 9 then addresses the limitations of our study, offers guidelines for future research efforts, summarizes our contributions, and outlines potential directions for future HAR research.

Chapter 2

Related Works

Recent advances in human activity recognition have focused on utilizing resource-constrained devices, such as the MCUs for on-device inference. Our review highlights key studies on classifying Activities of Daily Living (ADL) while addressing different model architectures and optimization strategies.

Daghero et al. [72] proposed efficient 1D CNNs for HAR on MCUs, incorporating hyperparameter optimization and model quantization (floating-point, integer, and mixed precision) to reduce power consumption and memory usage. Their previous work [70] introduced a two-stage hierarchical model combining a Decision Tree (DT) with a 1D CNN, specifically aimed at improving power efficiency. Using the UCI HAPT dataset with a non-overlapping sliding window of 5 seconds and subject-independent validation, they demonstrated that edge-based HAR solutions on MCU could significantly reduce the strain on cloud resources. Similarly, Ismail et al. [94] employed an adaptive CNN architecture design for HAR on MCU, further emphasizing the versatility of CNNs in low-power environments. Despite their different approaches, both studies confirm that CNNs are an effective tool for HAR on resource-constrained devices. Although these studies effectively leveraged resource-constrained MCUs for on-device inference using traditional sensors (i.e., accelerometer, gyroscope, and magnetometer), they did not explore the integration of PP sensors. Investigating the incorporation of PP sensor data within HAR frameworks could provide complementary insights, potentially enhancing feature extraction and recognition accuracy. This gap highlights a promising direction for future research aimed at developing more robust and efficient on-device activity recognition systems which our study addresses.

Several studies have explored the use of both overlapping and non-overlapping sliding windows for segmentation in HAR [45]. Non-overlapping windows, as used by Daghero et al., lower computational complexity but may miss finer-grained activity transitions, while overlapping windows capture smoother transitions at the cost of higher resource consumption. In validation, subject-independent strategies, while more challenging, offer better generalization across different users, whereas subject-dependent validation typically results in higher accuracy [14]. Moreover, when detecting composite human activities, many studies recommend employing longer window sizes [44], [132]. Since composite activities are characterized by a sequence of simpler, individual actions with varying temporal dynamics, a longer window can

effectively integrate and contextualize these sub-activities. This extended temporal coverage aids in reducing ambiguity and misclassification that shorter windows might introduce, thereby enhancing the predictive models' ability to discern complex activity patterns. Investigating these longer windows not only supports improved feature extraction but also provides richer contextual information, ultimately contributing to more robust and accurate activity recognition systems.

Besides that, Contoli et al. [86] investigated the application of TensorFlow compression techniques to HAR using DL models onto an ESP32. Here the authors demonstrate that it is possible and reasonable to deploy complex DL networks onto MCU with a proper optimization strategy to reduce latency and memory footprint. However, the optimization comes at a cost of predictive accuracy degradation.

Hinton et al. [23] demonstrated that KD serves as an effective technique for enhancing the performance of lightweight models. By leveraging a highly accurate teacher model, the logits produced by this teacher can function as soft labels for the student model, thereby improving both the accuracy and generalizability of the resulting lightweight model. Previously, Deng et al. [115] employed KD and data augmentation to train a lightweight HAR model. However, additional research is necessary to understand the limitations and requirements of deploying such a model on resource-constrained devices.

More recent HAR studies, despite focusing on edge deployment and addressing resource constraints, have predominantly covered theoretical analyses, often omitting practical evaluations on resource-constrained devices. For instance, Gonçalves et al. [122] employed a knowledge distillation approach combined with grid search to develop a lightweight student CNN model for edge deployment, resulting in a model 18 to 42 times smaller than the original teacher model. Although this methodology demonstrates significant reductions in model complexity, the authors did not deploy the compressed model on any edge hardware nor evaluate it within a complete end-to-end system. Similarly, AlMuhaideb et al. [108] proposed a custom ResLSTM architecture to alleviate the heavy computational load associated with BiLSTM models. Despite achieving higher accuracy and a reduction in the number of parameters, their work also overlooks practical deployment aspects, as it does not extend to full system integration or testing on actual edge devices. While advancing model efficiency is a valuable step forward, practical implementation and deployability remain crucial to ensure that these models meet end-to-end system requirements in real-world scenarios.

Multi-sensor fusion via knowledge distillation has garnered significant interest in the research community, with recent studies successfully integrating data from multiple sensors into a cohesive framework [97]. However, the use of PP sensors in the context of HAR remains underexplored; to date, no study has systematically examined their effect on activity recognition performance. Moreover, several investigations have largely focused on theoretical performance improvements, neglecting the practical aspects of deployment and on-device performance metrics [177]. Although theoretical advancements are important, it is equally essential to complement these findings with empirical validations, as practical deployment challenges often result in

substantial deviations from theoretical expectations. Additionally, state-of-the-art sensor fusion approaches that incorporate attention mechanisms [185] demonstrate the potential to adaptively refine feature maps, further underscoring the need for comprehensive evaluations that bridge theory and practice.

Chapter 3

Background

Human Activity Recognition (HAR) using wearable sensors has emerged as a cornerstone technology in the evolving landscape of ubiquitous computing and IoT applications. The fundamental premise of HAR lies in its ability to interpret human activities through sensor-derived data patterns, enabling context-aware systems that bridge the physical and digital worlds. Traditional approaches to HAR relied heavily on heuristic methods and manually engineered features, but the advent of ML has revolutionized the field through data-driven pattern recognition capabilities. However, the recent paradigm shift toward edge computing [109] and TinyML [159] introduces stringent constraints on computational resources, power consumption, and model complexity-related challenges that demand innovative solutions at the intersection of ML theory and embedded systems [8] engineering. Contemporary HAR systems mainly face a tripartite challenge: a) achieving high recognition accuracy while maintaining computational efficiency, b) minimizing energy consumption, and c) operating within the strict memory constraints of low-power microcontrollers. This trifecta of requirements has catalyzed research into optimized ML architectures that balance model performance with resource efficiency. The evolution of HAR methodologies has progressed from conventional machine learning techniques through deep NNs, and more recently towards hybrid approaches [71], [76], [82], [202] that combine the strengths of different paradigms. As a result, to better understand the rest of the study and reasoning behind certain design decisions (read Chapter 5), it is necessary to understand some of the foundational ML and DL models as well as relevant concepts of MCU architecture and firmware development.

This chapter provides a succinct overview of the essential background material necessary to contextualize the work presented in this study. Rather than detailing every nuance or elaborating upon each methodological aspect in its entirety, it is designed to furnish the reader with the foundational concepts and refresher information required to navigate the subsequent discussions effectively. The focus here is on presenting a carefully curated selection of topics that support a deeper understanding of the research without overwhelming the reader with superfluous details. We assume that the audience already possesses a working knowledge of the basic principles pertinent to this field. However, for those who wish to explore these concepts in greater depth, a comprehensive list of references is provided throughout the text. These citations serve as gateways to more elaborate expositions and additional scholarly resources. The next section covers tree-based machine learning

models, their inner workings, and common use cases.

3.1 Tree-Based Machine Learning Models

Tree-based models [46] constitute a fundamental family of machine learning algorithms that recursively partition the feature space through hierarchical decision rules. These models offer several inherent advantages for resource-constrained HAR applications:

- **Computational Efficiency**: DTs [181] naturally handle both numerical and categorical data with minimal preprocessing requirements.
- **Interpretability**: The hierarchical structure provides human-readable decision rules critical for system validation.
- **Non-Parametric Nature**: No inherent assumptions about data distributions.
- **Feature Selection**: Automatic relevance detection through split hierarchy.
- **Low Memory Footprint**: Compact model representations suitable for microcontroller deployment.

The subsequent subsections analyze the foundational components of tree-based models, focusing on DTs and their ensemble extension through Random Forests (RFs), while emphasizing the Gini impurity [36], [43] criterion for split optimization.

3.1.1 Decision Trees

A DT constructs [43] a hierarchical structure where internal nodes represent decision rules based on sensor features, branches represent rule outcomes, and leaf nodes store final activity class predictions. The construction process employs a greedy recursive partitioning algorithm that maximizes class separability at each split.

Gini Impurity Criterion

The Gini impurity measures the probability of misclassification for a randomly chosen element from the subset. For a dataset D with C activity classes, the Gini impurity $I_G(D)$ is calculated as:

$$I_G(D) = 1 - \sum_{i=1}^{C} p_i^2 \tag{3.1}$$

where p_i is the proportion of elements belonging to class i in D. The optimal split at each node is determined by maximizing the Gini impurity reduction, formally expressed as:

$$\Delta I_G = I_G(D) - \left(\frac{N_{left}}{N} I_G(D_{left}) + \frac{N_{right}}{N} I_G(D_{right}) \right) \tag{3.2}$$

where N represents the total samples in the parent node, and N_{left}, N_{right} denote samples in the child nodes after splitting.

Recursive Partitioning Steps

The tree construction process follows these key steps:

1. For each sensor feature x_j in feature vector $\mathbf{x}$:
    - Sort feature values
    - Evaluate all possible split thresholds τ
    - Calculate Gini gain for each (feature, threshold) pair
2. Select split with maximum ΔI_G
3. Partition dataset into left/right subsets
4. Recursively apply to child nodes until stopping criteria:
    - Maximum tree depth reached
    - Minimum samples per leaf node
    - Insufficient impurity reduction

The prediction for a new sensor sample $\mathbf{x}^*$ traverses the tree from root to leaf node, returning the majority class in the terminal node. Figure 3.1 shows the prediction process for a typical DT model.

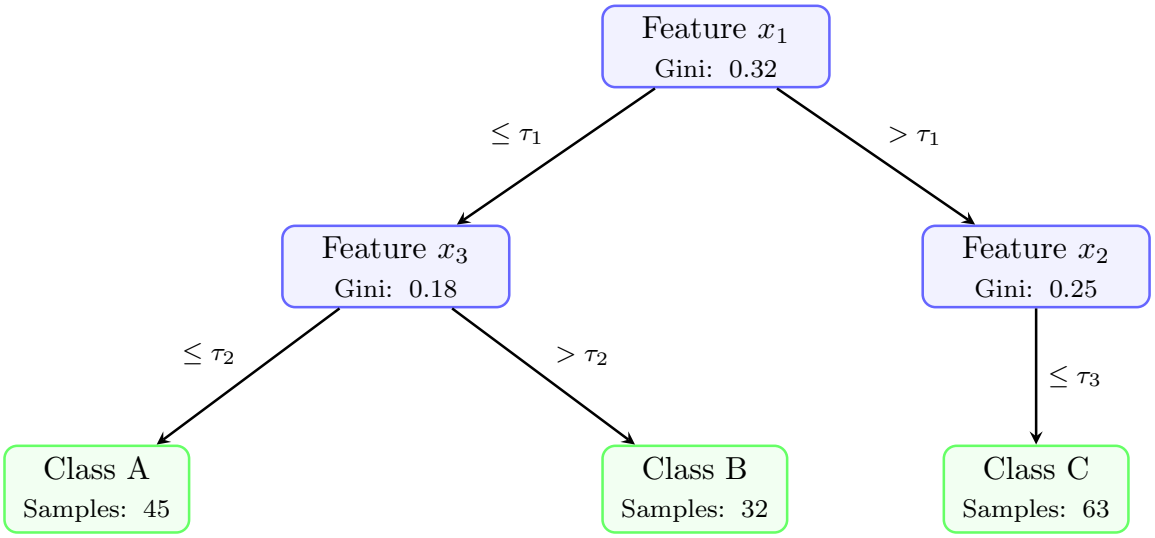


Figure 3.1: Decision Tree architecture demonstrating hierarchical partitioning with Gini impurity values and split thresholds at each node. Leaf nodes show final class predictions with sample counts.

3.1.2 Random Forests

RFs extend DTs through ensemble learning [183], combining multiple decorrelated trees to improve generalization while maintaining computational tractability. This approach addresses the high-variance limitation of individual DTs through two key mechanisms:

- **Bagging (Bootstrap Aggregating)**: Trains each tree on random subsets of the training data with replacement.
- **Feature Randomness**: At each split, considers random subsets of sensor features.

For an ensemble of T trees, the final activity class prediction $\hat{y}$ aggregates individual tree predictions (see Figure 3.2) through majority voting:

$$\hat{y} = \text{mode}\{h_1(\mathbf{x}^*), h_2(\mathbf{x}^*), \ldots, h_T(\mathbf{x}^*)\} \tag{3.3}$$

The probability estimate for class i is computed as:

$$p_i(\mathbf{x}^*) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{I}(h_t(\mathbf{x}^*) = i) \tag{3.4}$$

where $\mathbb{I}(\cdot)$ is the indicator function.

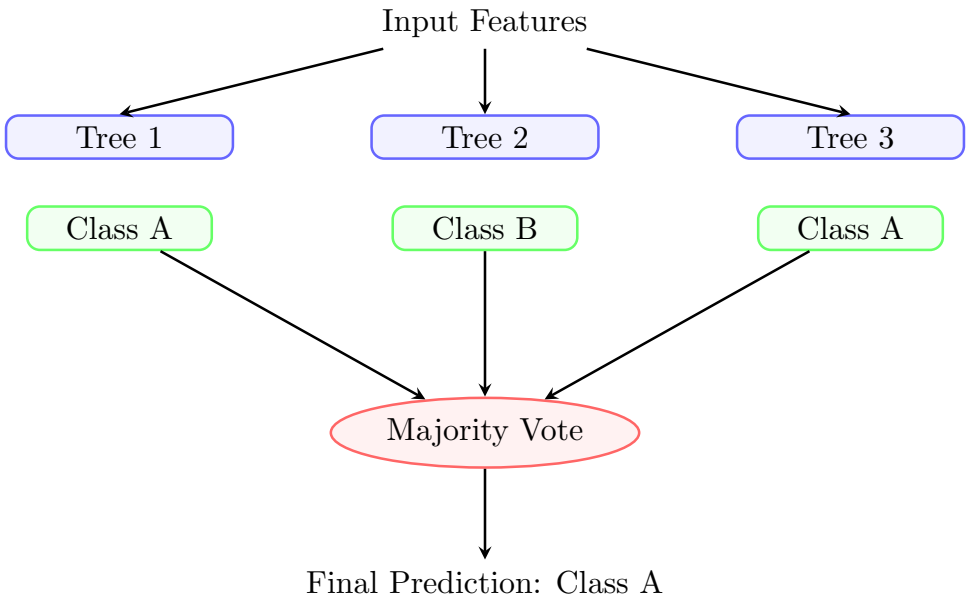


Figure 3.2: Random Forest architecture showing parallel ensemble of decision trees with bootstrap aggregating. Individual tree predictions are combined through majority voting.

Feature Importance Quantification

RFs enable sensor feature relevance analysis through mean Gini impurity reduction:

$$\text{Importance}(x_j) = \frac{1}{T} \sum_{t=1}^{T} \sum_{n \in \mathcal{N}_t(x_j)} \Delta I_G(n) \tag{3.5}$$

where $\mathcal{N}_t(x_j)$ denotes nodes in tree t that split on feature x_j.

Complexity Analysis

The training time complexity for a RF with T trees of depth d on N samples with M sensor features is:

$$\mathcal{O}(T \cdot d \cdot N \cdot \sqrt{M} \cdot \log N) \tag{3.6}$$

This sublinear scaling in feature dimension makes the algorithm particularly suitable for HAR systems with numerous sensor channels while maintaining computational tractability on low-power devices.

3.2 Gradient Boosted Decision Trees

While RFs employ parallel ensemble learning through bagging, gradient boosting techniques construct sequential ensembles that progressively correct residual errors. This approach has produced state-of-the-art results in numerous machine learning competitions and real-world applications, particularly through optimized implementations like XGBoost [28] and LightGBM [34]. These frameworks introduce critical innovations in computational efficiency and algorithmic optimization that align with the constraints of low-power IoT devices.

3.2.1 XGBoost (Extreme Gradient Boosting)

XGBoost [28] revolutionized gradient boosting through systematic engineering optimizations and rigorous handling of sparse sensor data patterns. The framework introduces a regularized objective function that combines predictive loss with model complexity penalties:

$$\mathcal{L}(\phi) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{3.7}$$

where:

- $l(y_i, \hat{y}_i)$ is the differentiable loss function (e.g., squared error for regression, logistic loss for classification)
- $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ penalizes tree (see Figure 3.3) complexity with:
  - T: Number of leaves in tree f_k
  - w: Leaf weights vector
  - γ, λ: Regularization hyperparameters

Additive Training with Taylor Approximation

At each boosting iteration t, the model adds a new tree f_t to minimize:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \tag{3.8}$$

where:

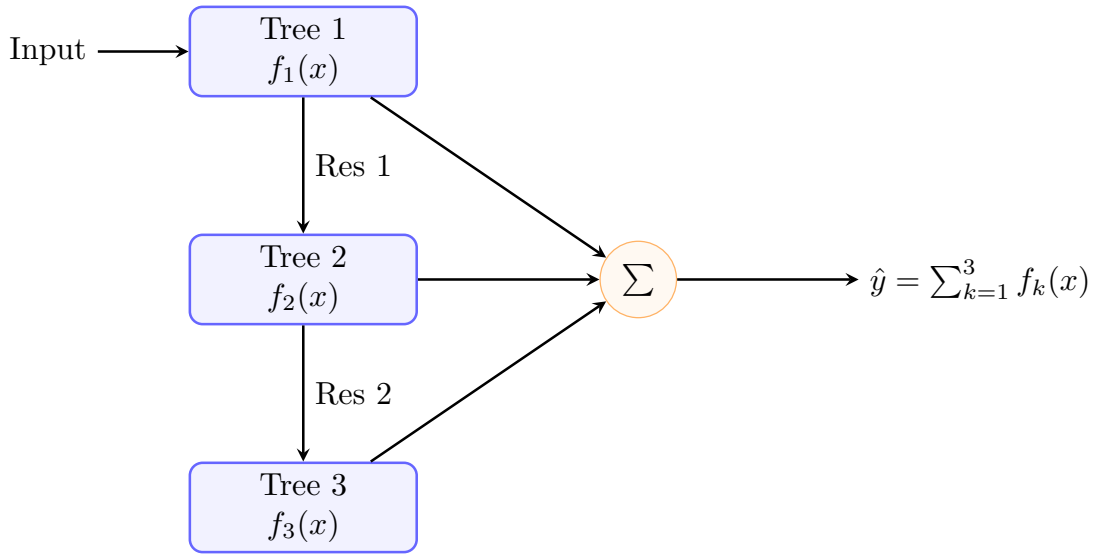
$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

The optimal weight w_j^* for leaf j and corresponding loss reduction are derived as:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{3.9}$$

$$\text{Gain} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{3.10}$$

where I_L, I_R represent instance sets in left/right child nodes after splitting.



$$\Omega(f_k) = \gamma T + \tfrac{1}{2}\lambda \|w\|^2$$

Figure 3.3: XGBoost's additive training process with sequential tree building. Each subsequent tree corrects residuals (Res 1 and Res 2) from previous models, with regularization terms controlling complexity.

Architectural Optimizations for Resource Constraints

XGBoost incorporates several innovations critical for HAR deployments:

- **Approximate Greedy Algorithm**: Utilizes weighted quantile sketch for efficient split candidate proposal.
- **Sparsity-aware Split Finding**: Automatically handles missing sensor values through default direction learning.

- **Block-oriented Parallelization**: Column block data structure enables feature sorting and gradient statistics reuse.
- **Cache-aware Access**: Optimizes memory access patterns for CPU cache efficiency.

The time complexity for building a tree with d depth and m candidate splits per node is:

$$\mathcal{O}(N_{sensors} \cdot d \cdot m \cdot N \log N) \tag{3.11}$$

3.2.2 LightGBM (Light Gradient Boosting Machine)

LightGBM [34] introduces novel gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB) to optimize both time and space efficiency, crucial for memory-constrained IoT devices.

Gradient-Based One-Side Sampling (GOSS)

Traditional boosting uses all instances for split finding. GOSS retains instances with large gradients while randomly sampling those with small gradients:

1. Sort sensor data instances by gradient magnitude $|g_i|$
2. Select top $a \times 100\%$ instances with largest gradients
3. Randomly sample $b \times 100\%$ from remaining instances
4. Amplify sampled small-gradient instances by $\frac{1-a}{b}$

The variance gain estimator becomes:

$$\tilde{V}_j(d) = \frac{1}{n}\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2 + \left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2 \tag{3.12}$$

where A and B represent large-gradient and sampled small-gradient subsets.

Exclusive Feature Bundling (EFB)

EFB reduces feature dimensionality by bundling mutually exclusive sensor channels (features that rarely take non-zero values simultaneously):

$$\text{EFB Cost} = \sum_{i=1}^{M}\sum_{j=1}^{N} \mathbb{I}(f_{ij} \neq 0) \tag{3.13}$$

The bundling problem maps to graph coloring:

- Construct graph with features as vertices
- Add edges between non-exclusive features
- Bundle features with same color (no edge connections)

Leaf-wise Growth Strategy

Contrasting with level-wise tree growth, LightGBM employs depth-first expansion:

$$\text{Gain}(L, R) = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{3.14}$$

where G and H represent sums of gradients and hessians in left/right nodes.

Complexity Analysis

LightGBM achieves training speedup factors of:

$$\text{Speedup} = \mathcal{O}\left(\frac{N_{original}}{aN_{original} + b(1-a)N_{original}} \times \frac{M_{original}}{M_{bundled}}\right) \tag{3.15}$$

Typical implementations see 2–5x memory reduction and 3–6x speed improvements compared to traditional gradient boosting [34], [131].

3.3 Deep Learning Models

DL has revolutionized machine learning by enabling hierarchical feature extraction from raw data. This capability is critical for complex tasks such as Human Activity Recognition, where temporal and spatial patterns are intertwined. In this section, we systematically dissect the foundational architectures of DL, starting from single neurons to modern attention-based models, emphasizing their mathematical formulations and relevance to TinyML.

3.3.1 Perceptron (Artificial Neuron)

The perceptron [40], [50], inspired by biological neurons, is the fundamental unit of neural networks [61]. A perceptron receives inputs $\mathbf{x} \in \mathbb{R}^n$, applies learnable weights $\mathbf{w} \in \mathbb{R}^n$, and produces an output y through an activation function ϕ:

$$y = \phi\left(\mathbf{w}^T\mathbf{x} + b\right), \tag{3.16}$$

where $b \in \mathbb{R}$ is the bias term. Activation functions like the Heaviside step, sigmoid ($\sigma(z) = \frac{1}{1+e^{-z}}$), and ReLU ($\text{ReLU}(z) = \max(0, z)$) introduce non-linearity. While perceptron can model linear decision boundaries, their inability to handle non-linear separable data (e.g., XOR) necessitated multi-layer extensions.

3.3.2 Multilayer Perceptron

MLPs [18] stack multiple perceptron layers, enabling hierarchical representations. Let $\mathbf{a}^{(l)}$ denote the activation of layer l, with weights $\mathbf{W}^{(l)}$ and biases $\mathbf{b}^{(l)}$. The forward propagation for an L-layer MLP is:

$$\mathbf{a}^{(0)} = \mathbf{x}, \tag{3.17}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \text{for } l = 1, \ldots, L, \tag{3.18}$$

$$\mathbf{a}^{(l)} = \phi\left(\mathbf{z}^{(l)}\right). \tag{3.19}$$

The universal approximation theorem [57], [133] states that an MLP with one hidden layer can approximate any continuous function, given sufficient width. However, deep MLPs face challenges such as vanishing gradients [1], [52], [58] and computational inefficiency [203] for high-dimensional data like sensor signals.

3.3.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) [26], [37] exploit spatial locality and translation invariance via convolutional layers. A 2D convolution operation between input $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$ and kernel $\mathbf{K} \in \mathbb{R}^{k\times k\times C\times F}$ is:

$$\mathbf{Y}_{i,j,f} = \sum_{c=1}^{C} \sum_{u=-k/2}^{k/2} \sum_{v=-k/2}^{k/2} \mathbf{X}_{i+u,j+v,c} \cdot \mathbf{K}_{u,v,c,f} + b_f, \tag{3.20}$$

where f indexes output feature maps. Pooling layers (e.g., max-pooling) reduce spatial dimensions, enhancing translational invariance. CNNs excel in extracting spatial hierarchies, making them ideal for image-based activity recognition. Architectures like original LeNet-5 [2] and ResNet [167] have demonstrated state-of-the-art performance.

3.3.4 Residual Connections

Very deep networks suffer from degradation, where accuracy saturates and then declines. Residual Networks (ResNets) address this via skip connections. A residual block computes:

$$\mathbf{a}^{(l+1)} = \phi\left(\mathcal{F}(\mathbf{a}^{(l)}) + \mathbf{a}^{(l)}\right), \tag{3.21}$$

where $\mathcal{F}$ represents stacked nonlinear transformations. This allows gradients to flow directly through layers, enabling training of networks with hundreds of layers. Residual connections are now ubiquitous in architectures targeting complex temporal-spatial patterns.

3.3.5 Recurrent Neural Network

Recurrent Neural Networks (RNNs) [49] process sequential data by maintaining a hidden state $\mathbf{h}_t$ at time t. The update rule is:

$$\mathbf{h}_t = \phi\left(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + \mathbf{b}\right), \tag{3.22}$$

where $\mathbf{x}_t$ is the input at time t. Despite their theoretical ability to capture long-term dependencies, vanilla RNNs suffer from vanishing/exploding gradients [52]. This limitation spurred the development of gated RNN variants [58].

3.3.6 Long Short-Term Memory

Long Short-Term Memorys (LSTMs) [51], [68] mitigate gradient issues via gating mechanisms. An LSTM cell comprises:

- Input gate: $\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$
- Forget gate: $\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$
- Output gate: $\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$
- Cell state: $\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C)$
- State update: $\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$
- Hidden state: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$

Here, $\odot$ denotes element-wise multiplication. LSTMs excel in modeling long-range dependencies in time-series data, such as human activity sequences.

3.3.7 Gated Recurrent Unit

Gated Recurrent Units (GRUs) [31], [58] simplify LSTMs by combining cell and hidden states. Key equations are:

$$\text{Update gate: } \mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z), \tag{3.23}$$

$$\text{Reset gate: } \mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r), \tag{3.24}$$

$$\text{Candidate activation: } \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h), \tag{3.25}$$

$$\text{Hidden state: } \mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \tag{3.26}$$

GRUs achieve comparable performance to LSTMs with fewer parameters, making them attractive for TinyML deployments.

3.3.8 Attention Mechanism

Attention mechanisms [81], [104] dynamically focus on relevant input regions. For a query $\mathbf{q}$, keys $\mathbf{K}$, and values $\mathbf{V}$, scaled dot-product attention computes:

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \tag{3.27}$$

where d_k is the key dimension. Self-attention [93] extends this by using the same input for queries, keys, and values. Attention enables models to capture contextual relationships regardless of distance, overcoming RNN limitations.

3.3.9 Transformer

Transformers rely entirely on attention mechanisms [104], [160]. The encoder-decoder architecture processes inputs in parallel via multi-head attention:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O, \tag{3.28}$$

$$\text{where head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V). \tag{3.29}$$

Positional encodings inject sequence order information. Transformers have set benchmarks in natural language processing and are being adapted for multimodal activity recognition.

3.3.10 Attention Pooling

Attention pooling [48], [53], [166] aggregates features by learning adaptive weights. Given features $\mathbf{H} \in \mathbb{R}^{T \times d}$, the context vector $\mathbf{c}$ is:

$$\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t, \quad \alpha_t = \text{softmax}(f(\mathbf{h}_t)), \tag{3.30}$$

where f is a learnable scoring function. This technique is pivotal in sequence-to-task models common in TinyML, where computational efficiency is critical. The evolution from perceptron to transformers reflects deep learning's trajectory toward modeling complex dependencies efficiently.

3.4 Microcontroller Units and Embedded System Architectures

Modern HAR systems demand specialized hardware architectures that balance computational capability with stringent power and cost constraints. Microcontroller Units (MCUs) [6], [59], [141] have emerged as pivotal components in this domain, offering integrated solutions that combine processing cores, memory subsystems, and peripheral interfaces in single-chip packages. These embedded systems enable energy-efficient execution of sensor data processing pipelines while maintaining the low bill-of-materials [25], [55] costs essential for scalable IoT deployments. This section analyzes the architectural considerations of MCU-based HAR systems, examining processor characteristics, memory hierarchies, hardware accelerators, and development toolchains that collectively enable efficient embedded ML and DL implementations.

3.4.1 Microcontroller Unit Fundamentals

Microcontroller Units represent the cornerstone of low-power embedded systems, integrating essential computing components into unified System-on-Chip (SoC) [9], [92] designs. Figure 3.4 shows a high-level overview of an MCU. Modern 32-bit MCUs typically feature:

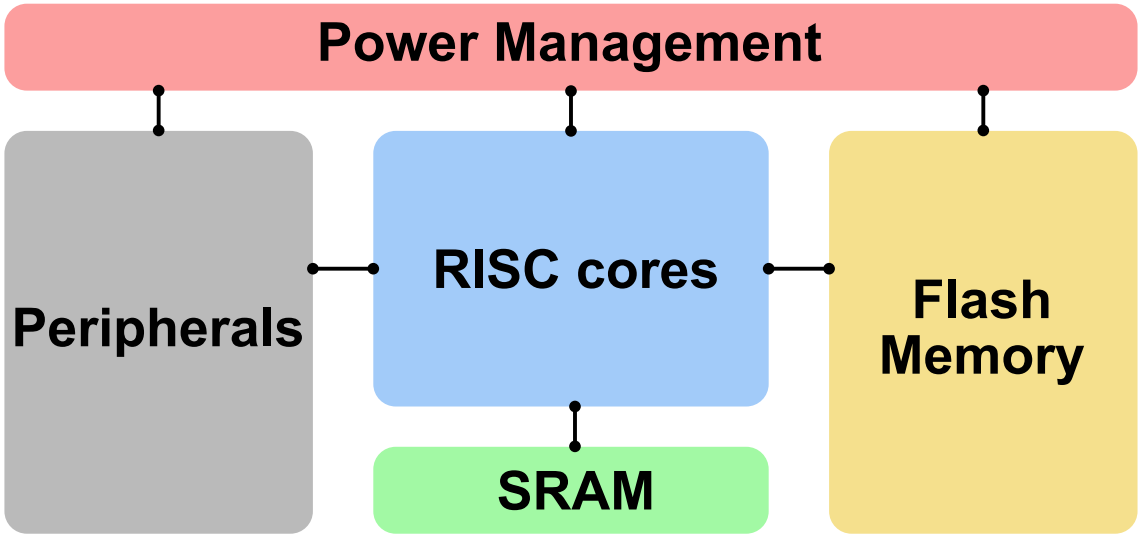


Figure 3.4: High-level overview of a typical RISC MCU architecture.

- 32-bit Reduced Instruction Set Computing (RISC) [7] cores (ARM Cortex-M [29], [78], RISC-V [74], [88]) operating at 10–500 MHz clock frequencies.
- Memory configurations combining Static Random Access Memory (SRAM) [73] (16 KiB-1 MiB) and Flash (64 KiB-8 MiB) [12].
- Advanced power management units supporting multiple low-power modes.
- Digital and analog peripherals for sensor interfacing (I2C, SPI, ADC) [62].
- Hardware security modules for data integrity in distributed deployments.

The evolution of MCUs has been remarkable, from the early 8-bit architectures with limited computational capabilities to modern 32-bit devices featuring sophisticated power management systems and specialized hardware accelerators [65]. Contemporary MCU families like ARM Cortex-M series, Espressif ESP32 [196], Texas Instruments MSP430 [175], and STMicroelectronics STM32 [194] offer varying degrees of performance, power efficiency, and peripheral integration. For instance, the ARM Cortex-M4F combines a 32-bit RISC processor with a Floating-Point Unit (FPU) and Digital Signal Processing (DSP) extensions, making it particularly suitable for the mathematical operations common in HAR and TinyML feature extraction and classification tasks. When selecting an MCU for HAR applications, several critical factors must be considered beyond raw computational power. These include power consumption profiles (active, sleep, and deep sleep modes) [42], [176], available memory resources, peripheral interfaces for sensor integration, debugging capabilities, and the maturity of the development ecosystem. The emergence of specialized MCUs with machine learning accelerators has further expanded the design space, enabling more complex algorithms to be executed efficiently at the edge [126].

MCU selection represents a fundamental design decision that cascades throughout the entire system architecture. A well-chosen MCU platform provides the foundation for balancing the competing demands of computational capability, energy efficiency, and cost-effectiveness; three factors that are particularly critical in wearable and

IoT-based HAR deployments. The following sections explore the key components of MCUs briefly, beginning with the processor architecture that forms the computational heart of these versatile devices.

3.4.2 Processor Architecture Considerations

The processor constitutes the computational core of any microcontroller unit, directly influencing the performance capabilities, power consumption profile, and application suitability for HAR implementations. Most contemporary MCUs designed for low-power applications utilize RISC architectures, which prioritize simplicity and efficiency over complex instruction handling. The predominant architectures in the IoT and wearable HAR space include ARM Cortex-M series (particularly M0+, M3, and M4 variants) [29], RISC-V [7], and proprietary designs like Espressif's Xtensa LX6/LX7 [96], [195].

Clock Generation and Management

The temporal foundation of MCU operation stems from clock generation [24] circuits employing either internal RC oscillators (1–10% accuracy) [199], [200] or external crystal resonators (0.001–0.01% stability) [15]. While 32.768 kHz watch crystals [171] enable low-power real-time clock operation, HAR workloads typically require 16–48 MHz crystals for main clock domains. Advanced MCUs implement Dynamic Frequency Scaling (DFS) [105] through Phase-Locked Loops, allowing runtime adjustments between energy-saving low-frequency modes (e.g., 4 MHz) and high-performance states (80+ MHz) during intensive computation phases.

Instruction Pipeline Efficiency

The relationship between clock frequency (f_{CLK}) and effective performance depends on architectural factors:

$$IPC = \frac{Instructions}{Cycle} \times f_{CLK} \times \eta_{pipeline} \tag{3.31}$$

Where $\eta_{pipeline}$ accounts for pipeline stalls from memory wait states or branch mispredictions (i.e., tree-based ML inference). Modern MCUs employ 3–7 stage pipelines [11] with branch prediction and single-cycle I/O access to maintain > 1.25 DMIPS/MHz efficiency. Furthermore, ARM Cortex-M4's [29] 3-stage Harvard architecture demonstrates particularly favorable characteristics for sensor data processing through simultaneous instruction and data fetch capabilities.

3.4.3 Memory Subsystem Organization

Most Modern MCU nowadays implement hierarchical memory systems [204] with varying technologies, each offering different trade-offs between capacity, speed, power consumption, and persistence. This hierarchy typically includes non-volatile storage (primarily Flash memory), volatile main memory (SRAM or occasionally Dynamic Random Access Memory (DRAM) [3]), and potentially specialized buffer memories or caches [20]. The organization and characteristics of these memory components

fundamentally shape the design approaches for HAR algorithm implementation on resource-constrained devices.

Static Random-Access Memory

SRAM provides high-speed, volatile storage for active data processing in microcontrollers, critical for time-sensitive HAR tasks. Its six-transistor bistable cells enable fast access (10–30 ns), zero refresh requirements, and low active power consumption [147], [148]. However, lower density limits capacities (typically 32 KiB to 4 MiB in MCUs), necessitating careful memory management for sensor buffers and algorithm workspaces [204]. Advanced implementations feature multi-bank architectures (e.g., STM32L4 [205]) with dedicated low-leakage regions to preserve data during sleep states, optimizing energy efficiency without sacrificing critical state retention.

Dynamic Random-Access Memory

DRAM offers higher density (megabyte-scale) through single-transistor cells with capacitor storage but requires periodic refresh cycles, increasing power and complexity. While rarely used in low-power traditional MCUs, it supports advanced applications needing large temporal buffers or complex models. Drawbacks include refresh-related power overhead (20–30%) and intricate timing control. Hybrid solutions like PSRAM [16], [178] merge DRAM density with SRAM-like interfaces, simplifying integration but retaining refresh penalties, as seen in ESP32-S3 (16 MiB) [197]. DRAM remains reserved for high-performance MCUs where capacity justifies trade-offs.

Flash Memory

Non-volatile flash memory stores program code and static HAR parameters (e.g., model weights, constants) with moderate read speeds (20–50 ns) and low standby power [12]. Write/erase operations are slow (milliseconds), energy-intensive, and limited by endurance (10k–100k cycles), restricting flash to infrequently updated data. HAR systems may employ wear-leveling or journaling to manage semi-persistent data (e.g., calibration values), though these add design complexity. Typical MCUs integrate 64 KiB–4 MiB [102], balancing code storage with fixed algorithm requirements.

Cache Memory

Cache hierarchies bridge speed disparities between processors and main memory, reducing access latency [41], [98]. MCU vary from basic instruction caches (4–16 KiB) to multi-level designs (e.g., STM32H7's L1/L2 caches) [13]. For HAR, cache efficiency depends on algorithm access patterns: sequential sensor processing benefits from data caching, while irregular classification steps may not. Multicore MCUs require coherency protocols (hardware-managed or software-driven) [87] to synchronize shared memory views, critical for parallel HAR workloads. Optimizing data alignment and access patterns enhances cache utilization, improving both performance and energy efficiency.

Memory constraints significantly influence HAR algorithm design. For example, storing TensorFlow Lite for Microcontrollers models typically requires 50–200 KiB Flash, while inference operations may need 10–50 KiB SRAM for tensor buffers [117]. However, the actual value depends on the utilized MCU, models, Operations (OPS) [155], and its chipset.

3.4.4 Computational Accelerators

Embedded systems for HAR applications often require specialized hardware accelerators to efficiently process sensor data and execute machine learning algorithms while maintaining low power consumption. These accelerators complement the main processor by handling computationally intensive tasks with greater efficiency [30].

Floating-Point Units

Floating-Point Units are specialized hardware components designed to perform arithmetic operations on floating-point numbers much faster than would be possible using software emulation [99]. For HAR applications, FPUs are critical for efficiently processing sensor data that requires mathematical operations with decimal values. Modern MCUs often include integrated FPUs that significantly accelerate signal processing algorithms commonly used in activity recognition. These units implement IEEE 754 floating-point arithmetic standards [5] and provide hardware support for operations such as addition, multiplication, division, and square root calculations. The inclusion of an FPU can dramatically improve the performance of digital signal processing tasks fundamental to HAR such as filtering, feature extraction, and some machine learning operations [38].

Most Arm Cortex-M4F, M7F, and M33F [29] processors include FPUs that enable single-precision (32-bit) floating-point operations [78], while higher-end processors may support double-precision (64-bit) operations. The integration of FPUs into low-power MCUs has been a crucial development for enabling more sophisticated HAR algorithms on edge devices without sacrificing battery life [159].

Neural Processing Units

Neural Processing Units (NPU) [67] are specialized hardware accelerators designed specifically for artificial intelligence and machine learning workloads. These units are optimized for the matrix multiplications and other operations that dominate neural network inference [63], [80]. NPUs enable significantly faster and more energy-efficient execution of machine learning models compared to running these models on general-purpose CPUs. This is particularly valuable for HAR applications, which often employ neural networks for activity classification tasks. Recent advancements in NPU technology have made it possible to integrate these accelerators into low-power microcontrollers, extending Artificial intelligence (AI) capabilities to edge devices [112], [184]. For instance, the i.MX RT700 crossover microcontrollers from NXP Semiconductors launched in 2024 feature an integrated NPU that facilitates efficient processing of AI algorithms at the edge while maintaining low energy consumption [165], [188]. This enables more complex HAR models to run directly on the device without requiring cloud connectivity. NPUs typically implement quantized

operations that use fixed-point arithmetic rather than floating-point [63], reducing both computational complexity and memory requirements. This makes them particularly suitable for resource-constrained environments where power efficiency is paramount.

Graphics Processing Units

Graphics Processing Unit (GPU) [35] were originally designed for rendering graphics but have found wide application in parallel computing tasks, including machine learning [47]. GPUs feature a highly parallel architecture with many processing cores that can handle thousands of threads simultaneously. While traditional GPUs are power-hungry and unsuitable for battery-powered devices, embedded GPUs with reduced capabilities have been developed for mobile and embedded applications [56], [90]. These scaled-down GPUs maintain the parallel processing architecture, yet it significantly reduces power consumption [85].

When considering HAR applications, embedded GPUs can accelerate certain types of computation, particularly those involving matrix operations and parallel data processing. They are especially useful for more complex HAR systems that may implement computer vision alongside inertial sensor analysis, such as systems that combine video and accelerometer data for more accurate activity recognition [4].

Comparison of Hardware Accelerators for HAR Applications

The following table provides a general rough comparison of different hardware accelerators in terms of their characteristics and suitability for HAR applications. For a more detailed discussion on DL accelerators, we encourage readers to see the study conducted by Cristina Silvano et al. [145].

Table 3.1: Rough comparison between different accelerators for HAR tasks.

Accelerator Type	**Power Efficiency**	**Cost**	**Capability**	**Suitability for HAR**
FPU	High	Low	Medium	Good
NPU	Very High	Medium	High (NN)	Excellent
GPU	Medium	High	Very High	Good
CPU	Low	Low	Low	Okay (simple)

As shown in Table 3.1, NPUs offer the best combination of power efficiency and computational capability for NN-based HAR applications, making them increasingly popular in modern edge AI designs [159]. FPUs provide a good balance for traditional signal processing approaches, while GPUs are typically reserved for more complex systems where power constraints are less stringent. However, for highly resource-constrained and cost-effective MCUs/IoTs devices, extra considerations must be taken into account as discussed in Chapter 5. Besides, the energy efficiency of these accelerators is particularly important for IoT devices implementing HAR, as these often need to operate for extended periods on battery power. The use of specialized accelerators can extend battery life by orders of magnitude compared to implementations that rely solely on general-purpose computing [19].

3.4.5 Toolchain and SDKs for Embedded Development

Developing efficient HAR or in general any application for embedded systems requires appropriate toolchains and Software Development Kits (SDKs) that are optimized for resource-constrained environments. These tools enable developers to write, compile, optimize, and debug code for microcontrollers efficiently.

Compilers for Embedded Systems

Compilers translate human-readable code into machine code that can be executed by the target microcontroller. For embedded applications, compiler efficiency directly impacts both the performance and energy consumption of the final system. Two major compiler frameworks dominate the embedded development landscape: GNU Compiler Collection (GCC) and Clang/LLVM [17], [22]. GCC has long been the standard compiler for embedded systems development, offering support for multiple programming languages and a wide range of target architectures. Its extensive optimizations and broad platform support make it a popular choice for embedded development.

On the other hand, Clang, which uses the LLVM framework as its backend, has gained significant momentum in recent years. LLVM is an open-source project consisting of modular and reusable compiler toolchain technologies, while Clang serves as the C/C++ compiler front end. Major technology companies including Apple, Arm, Google, and Microsoft have heavily invested in these projects. One of the key advantages of Clang is its compatibility with code written for GCC. Clang supports the majority of GCC extensions [208], enabling developers to use existing codebases while benefiting from LLVM's optimization capabilities and modular architectures. This compatibility is particularly valuable when integrating third-party libraries and algorithms common in TinyML applications.

Optimization Levels

Compiler optimization [206], [207] levels significantly impact the performance, code size, and energy consumption of target application (HAR in our case) running on embedded systems. Common optimization levels/flags include:

- *-O0* No optimization, fastest compilation Debugging.
- *-O1* Basic optimizations without significant compilation time increase for the initial testing of implementations.
- *-O2* Moderate optimizations for both speed and size Balancing performance and code size.
- *-O3* Aggressive optimization for speed Compute-intensive algorithms.
- *-Os* Optimization for code size memory-constrained devices.
- *-Ofast* Maximum performance, may affect standards compliance performance-critical sensor processing loops.

When considering TinyML applications on low-power MCUs, the choice of optimization level often involves trade-offs between processing speed, code size, and energy efficiency. Higher optimization levels generally produce faster code but may increase binary size and compilation time. For resource-constrained devices, optimizing for code size (-Os) [207] is often preferred to ensure the application fits within the limited flash memory of the MCUs. Advanced optimizations like link-time optimization and profile-guided optimization can further improve performance by enabling whole-program analysis and runtime behavior-based optimizations, respectively.

Development Platforms

Development of HAR applications for embedded systems is facilitated by Integrated Development Environments (IDEs) and platforms that streamline the workflow from code writing to final deployment. Several development environments are commonly used for embedded HAR development, including MCU vendor-provided IDEs and independent third party IDEs with their own advantages and disadvantages. However, at its core, all IDEs utilize the vendor-provided toolchain, specifically the compiler and related tooling (i.e., firmware flashing tool etc.). In the following sections we briefly discuss regarding the Linux-Based open source embedded development ecosystem.

Linux-Based Development

Linux provides a powerful environment for embedded development with its rich set of open-source tools. Many developers prefer using Linux-based systems for embedded HAR development due to:

- Availability of command-line tools that can be easily scripted and automated unlike proprietary windows.
- Native support for many open-source embedded development tools.
- Compatibility with continuous integration systems.
- Support for cross-compilation environments.

Moreover, Linux-based development workflows typically involve cross-compilation, where code is compiled on a desktop environment (host) for execution on the target microcontroller. This approach leverages the processing power of development machines while producing binaries optimized for resource-constrained targets. Furthermore, effective debugging tools are essential for developing reliable TinyML applications on embedded systems, where limited resources make problems harder to identify and resolve. As a result, Linux-based systems provide first-class support for GNU Debugger, Joint Test Action Group (JTAG), and Profiling tooling out of the box, making it essential for an embedded system development.

In conclusion, the combination of appropriate compilers, optimization techniques, development environments, and debugging tools creates a powerful ecosystem for developing efficient TinyML HAR applications on embedded systems. The right

toolchain can significantly impact both development productivity and the performance characteristics of the final application, making it a critical consideration for TinyML system engineers.

Chapter 4

Dataset

This chapter provides a comprehensive overview of the datasets employed in this study. It presents detailed descriptions, visualization summaries, and the end-to-end preparation workflows for each source. Three publicly available datasets were selected: the CAPPIMU dataset [132], the KU HAR dataset [66], and the NTNU HAR Children dataset [158]. These datasets were sourced directly from the repositories linked in their corresponding publications. Specifically, the CAPPIMU dataset was obtained from a GitHub repository, the KU HAR dataset from Mendeley Data, and the NTNU Children HAR dataset was retrieved from Dataverse. Our primary investigation focuses on the CAPPIMU dataset, with a particular emphasis on the role of plantar pressure signals within a knowledge-distillation framework as discussed in Chapters 1 and 2. The KU HAR and NTNU Children HAR datasets are employed as external benchmarks (see Chapter 5) to assess model performance on data sources that lack plantar pressure channels and exhibit class imbalance.

Prior to any modeling, each dataset was subjected to a uniform preprocessing pipeline. In general, raw sensor recordings were synchronized and resampled to a common rate. Signals were segmented into fixed-length sliding windows with overlap to capture temporal dynamics. The remainder of this chapter is organized as follows. Section 4.1 describes the structure, sensor modalities, and visual characteristics of the CAPPIMU dataset along with its unique preprocessing steps. Similarly, Section 4.2 and Section 4.3 outline the key properties and preprocessing steps for the KU HAR dataset and NTNU Children HAR dataset respectively.

4.1 CAPPIMU Dataset

This study employs the CAPPIMU dataset [132], a multi-sensor dataset uniquely suited for investigating the interplay between IMU and PP sensor data. To the best of our knowledge, it is the only publicly available dataset that combines synchronized readings from accelerometers, gyroscopes, magnetometers, and PP sensors, addressing a critical gap in wearable sensor research. This dual-modality design aligns directly with our objective of evaluating the role of PP data in KD (see Chapter 5), making it the natural choice of our study.

Algorithm 1 Brief CAPPIMU dataset preprocessing pipeline

Require: Base directory $\mathcal{D}_{base}$, Window size W, Step size S, Train ratio ρ.
Ensure: Training set (X_{train}, y_{train}), Testing set (X_{test}, y_{test}).

$\mathbf{D} \leftarrow$ LoadSubjectData($\mathcal{D}_{base}$,"joint/insole_wrist_r.csv") ▷ Load & combine subject CSVs
$\mathcal{F} \leftarrow$ GetFeatureColumns("insole_l/r(p1-8)", "wrist_r(ax/y/z, wx/y/z, ex/y/z)")
$\mathbf{D}[\mathcal{F}] \leftarrow$ ZScoreNormalization($\mathbf{D}[\mathcal{F}]$) ▷ Normalize features $\mathcal{F}$
$\mathbf{D}$['label'] $\leftarrow$ EncodeLabels($\mathbf{D}$['label']) ▷ Map activity strings to integers
Initialize $\mathbf{X}_{tr}, \mathbf{y}_{tr}, \mathbf{X}_{te}, \mathbf{y}_{te} \leftarrow$ empty lists
for all subject $s \in$ UniqueSubjects($\mathbf{D}$) **do**
    $\mathbf{D}_s \leftarrow \mathbf{D}$[subject $= s$].SortByTime()
    Initialize $\mathbf{X}_{tr\_s}, \mathbf{y}_{tr\_s}, \mathbf{X}_{te\_s}, \mathbf{y}_{te\_s} \leftarrow$ empty lists
    for all activity $a \in$ UniqueLabels($\mathbf{D}_s$) **do**
        $\mathbf{D}_{s,a} \leftarrow \mathbf{D}_s$[label $= a$]
        $(\mathbf{X}_{s,a}, \mathbf{y}_{s,a}) \leftarrow$ ApplySlidingWindow($\mathbf{D}_{s,a}, \mathcal{F}, W, S$) ▷ Window data, flatten features, use mode label (highest)
        $(X_{train}, y_{train}), (X_{test}, y_{test}) \leftarrow$ SplitData($\mathbf{X}_{s,a}, \mathbf{y}_{s,a}, \rho$) ▷ Split activity data
        Append (X_{train}, y_{train}) to $(\mathbf{X}_{tr\_s}, \mathbf{y}_{tr\_s})$
        Append (X_{test}, y_{test}) to $(\mathbf{X}_{te\_s}, \mathbf{y}_{te\_s})$
    end for
    Append Combine($\mathbf{X}_{tr\_s}, \mathbf{y}_{tr\_s}$) to $(\mathbf{X}_{tr}, \mathbf{y}_{tr})$ ▷ Combine activities for subject s
    Append Combine($\mathbf{X}_{te\_s}, \mathbf{y}_{te\_s}$) to $(\mathbf{X}_{te}, \mathbf{y}_{te})$
end for
$(X_{train}, y_{train}) \leftarrow$ CombineAll($\mathbf{X}_{tr}, \mathbf{y}_{tr}$) ▷ Combine all subjects
$(X_{test}, y_{test}) \leftarrow$ CombineAll($\mathbf{X}_{te}, \mathbf{y}_{te}$)
SaveData("output/path", $X_{train}, y_{train}, X_{test}, y_{test}$) ▷ Save datasets to text files
return $(X_{train}, y_{train}), (X_{test}, y_{test})$

The dataset comprises recordings from subjects aged 20–25 years performing 21 diverse ADL, ensuring variability in motion patterns and sensor dynamics. Since one of the objectives of this study is to investigate the effect of PP sensor in KD based sensor fusion, we focused on the *insole_wrist_r* files, which include IMU data from the right wrist position shown to yield higher accuracy [132] paired with synchronized PP sensor data. Data preprocessing involved standard scaling [136] to normalize IMU and PP features followed by numerical encoding of activity labels necessary for model training.

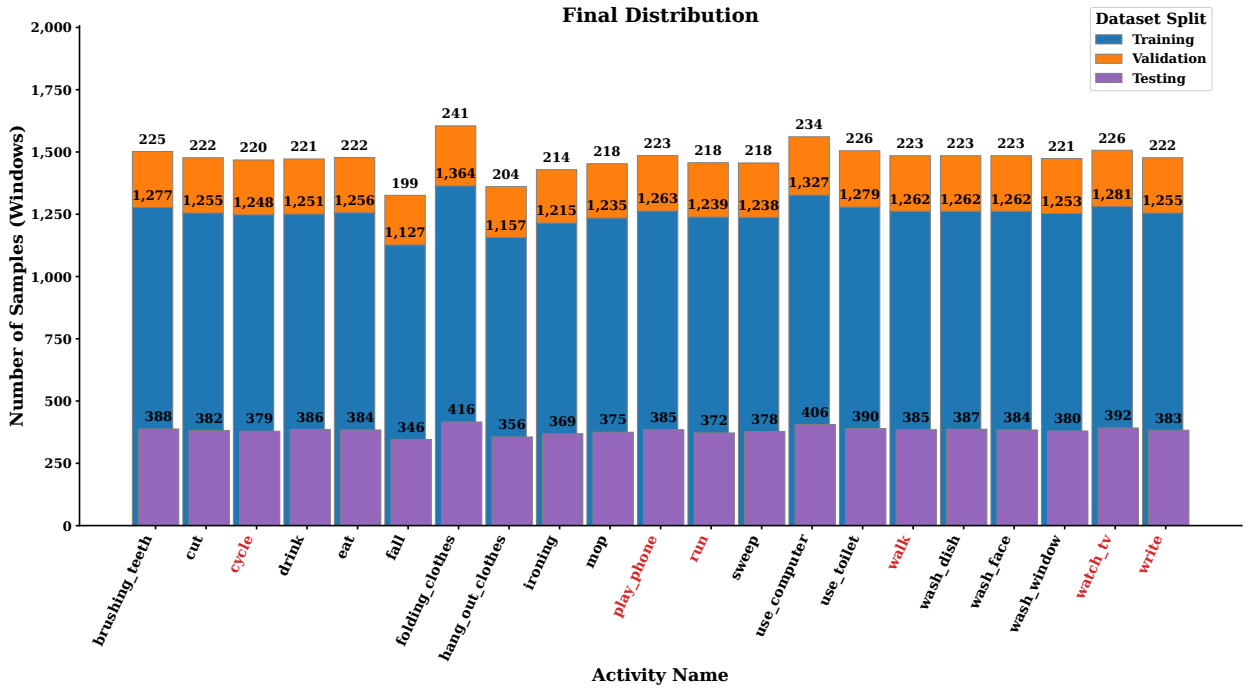


Figure 4.1: Shows the final train, test, and validation distribution of CAPPIMU dataset after preprocessing. Activity names colored in red represent simple type activities, while black-colored names represent composite type activities.

To capture temporal dynamics, a sliding window approach was employed. A window size of 300 samples, corresponding to a six-second window, was selected for two primary reasons. Firstly, the prediction of composite human activities necessitates a longer window to ensure the model captures the complete contextual information inherent in such activities. Secondly, this window size, along with a 50% (three-second) overlap, aligns with the recommendations provided by the dataset authors, thereby facilitating a fair and consistent comparison with existing studies. The 50% overlap was implemented to ensure smooth transitions between consecutive windows and to mitigate potential information loss at the window boundaries [45], [77]. When determining the label for each window, the label with the highest frequency within that window was selected. This technique effectively segments the time-series data into manageable portions for training and testing (see Algorithm 1 for ease of understanding and implementation). To ensure generalizability across subjects while retaining intra-subject variability, we employed a Subject-Dependent (SD) validation strategy, where data from all subjects contributed to both the training and testing phases. The dataset was partitioned into an 80%–20% train-test split, with 15% of the training set allocated for validation (see Figure 4.1).

4.2 KU HAR Dataset

The KU HAR dataset captures eighteen distinct ADL performed by ninety volunteers (seventy-five male and fifteen female) using embedded smartphone sensors (accelerometer and gyroscope). Originally, 1,945 continuous activity recordings were collected, which were later segmented into 20,750 fixed-length subsamples. The full list of activities includes standing, sitting, talking while sitting or standing, repeated

sit–stand and lay–stand motions, object picking, jumping, body-weight exercises (push-ups, sit-ups), various locomotion patterns (walking forward, backward, circular; running; stair ascent and descent), and table tennis.

The dataset archive is organized as follows:

- *Raw_time_domain_data.zip*: 1,945 unprocessed CSV files. Columns 1 and 5 contain timestamps (milliseconds); columns 2–4 record acceleration on the X, Y, and Z axes (m/s^2); columns 6–8 record angular rate on the X, Y, and Z axes (rad/s).
- *Trimmed_interpolated_raw_data.zip*: the same eight-column format, with start/end inactivity trimmed and data resampled to a constant 100 Hz.
- *Time_domain_subsamples.zip*: a single CSV file of 20,750 non-overlapping, three-second windows. Columns 1–300, 301–600, 601–900 hold accelerometer X, Y, Z data; columns 901–1200, 1201–1500, 1501–1800 hold gyroscope X, Y, Z data; column 1801 is the activity label (0–17); column 1802 is the sample length; column 1803 is the subsample index.

To ensure consistency with the preprocessing applied to the CAPPIMU dataset, this study employs the trimmed and interpolated sensor recordings. These files have all idle segments removed at the beginning and end of each recording and are resampled to a uniform 100 Hz rate. This uniform sampling simplifies windowing, avoids variable-length sequences, and reduces potential boundary artifacts during segmentation.

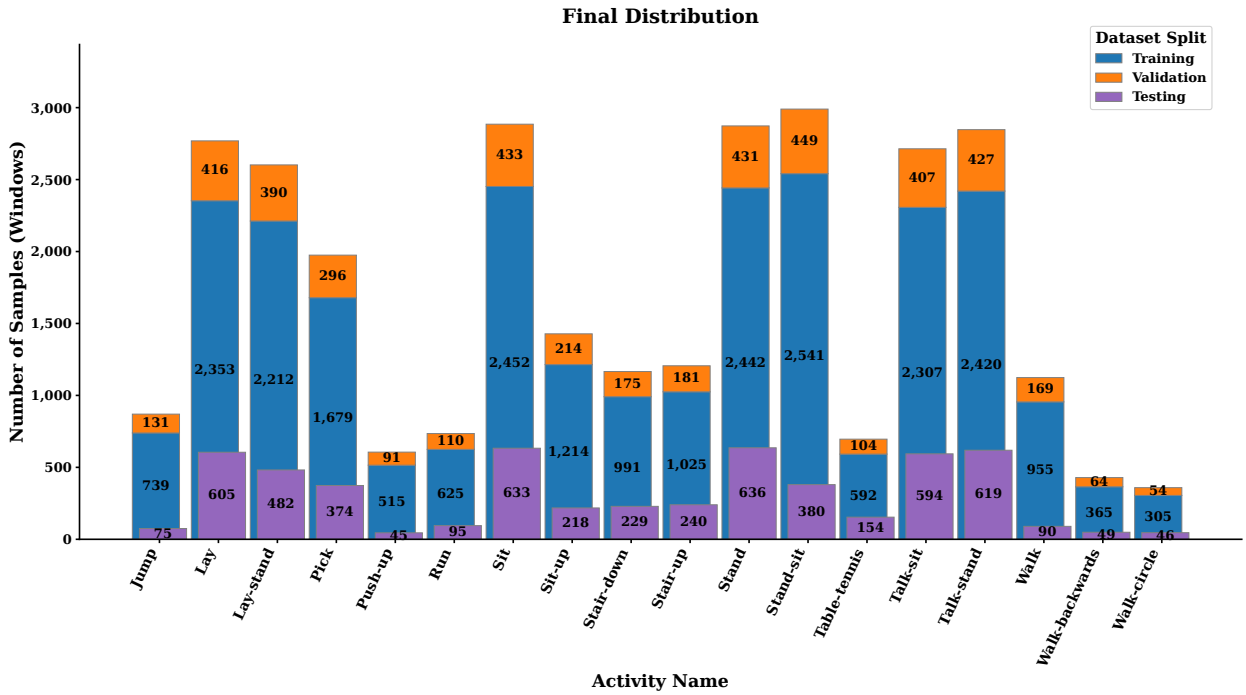


Figure 4.2: Shows the final train, test, and validation distribution of KU HAR dataset after preprocessing. Unlike CAPPIMU dataset, KU HAR dataset exhibits an imbalance distribution of activity labels.

For feature extraction and model training, the interpolated recordings were divided into three-second windows with 50% overlap. This window length balances temporal resolution and computational cost, while the overlap matches the segmentation strategy applied to the CAPPIMU dataset (see Section 4.1). The resulting windows preserve activity dynamics and increase sample diversity.

The data were split into training and test sets using an 80–20 ratio. Within the training set, 15% of windows were held out for validation to monitor convergence and prevent overfitting. The final distribution of activity labels across train, validation, and test partitions is illustrated in Figure 4.2.

4.3 NTNU Children Dataset

The NTNU Children HAR dataset [158] comprises synchronized acceleration recordings and video-based annotations from 79 children. Of these, 16 children have Cerebral Palsy (CP) (Gross Motor Function Classification Scale levels I and II) and 63 are Typically Developing (TD). Each participant wore two three-axial AX3 accelerometers (Axivity Ltd., Newcastle, UK): one on the lower back (at the third lumbar vertebra) and one on the upper thigh (approximately 10 cm above the patella).

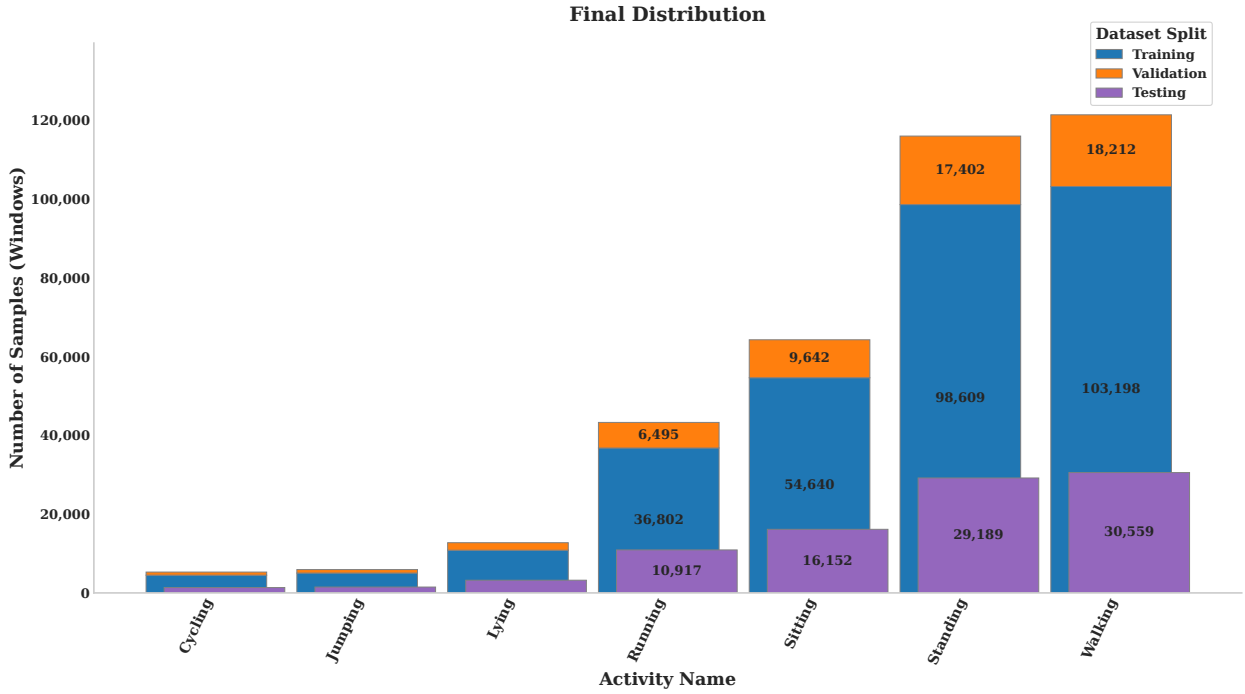


Figure 4.3: Shows the final train, test, and validation distribution of NTNU Children HAR dataset after preprocessing. Unlike CAPPIMU dataset, NTNU Children HAR dataset also exhibits an imbalance distribution of activity labels similar to KU HAR dataset. Due to having a significantly smaller number of samples for *Cycling*, *Jumping*, and *Lying* activities, their corresponding counts (train, test, and validation) could not be plotted in the diagrams.

All recordings were collected in laboratory, gymnasium, and outdoor settings, both

individually and in small groups. Body accelerations were captured at either 100 Hz or 200 Hz and subsequently downsampled to 50 Hz. Video streams were manually annotated using the Anvil video annotation tool (version 6). Twelve activity classes were labeled: sitting, standing, walking, shuffling, stair ascending, stair descending, lying, sit cycling, stand cycling, running, bending, jumping, and various transitions.

Data are provided as subject-specific CSV files that combine synchronized 50 Hz accelerometer signals and annotation streams. Files 001–016 contain children with CP, files 017–120 contain TD children, and are named PM01–16 and TD01–48 respectively. Raw CSV files were loaded, and each three-axis signal channel was inspected for missing samples. No further cleaning was required as the dataset was fully annotated and synchronized at 50 Hz.

Signals were segmented into fixed-length sliding windows of one second (50 samples) with 50% overlap to capture temporal dynamics across activities. For model development we employed a SD split strategy (similar to Section 4.1). All segmentation windows from every participant (following the dataset authors [158], we also combined children with CP and TD children) were pooled and then partitioned into training (80%) and testing (20%) sets at the window level. From within the training windows, 15% were randomly held out as a validation set (see Figure 4.3 for final distribution). This window-level partition ensures that each child contributes data to training, validation, and test subsets, reflecting a subject-dependent evaluation scenario.

Chapter 5

Research Methodology

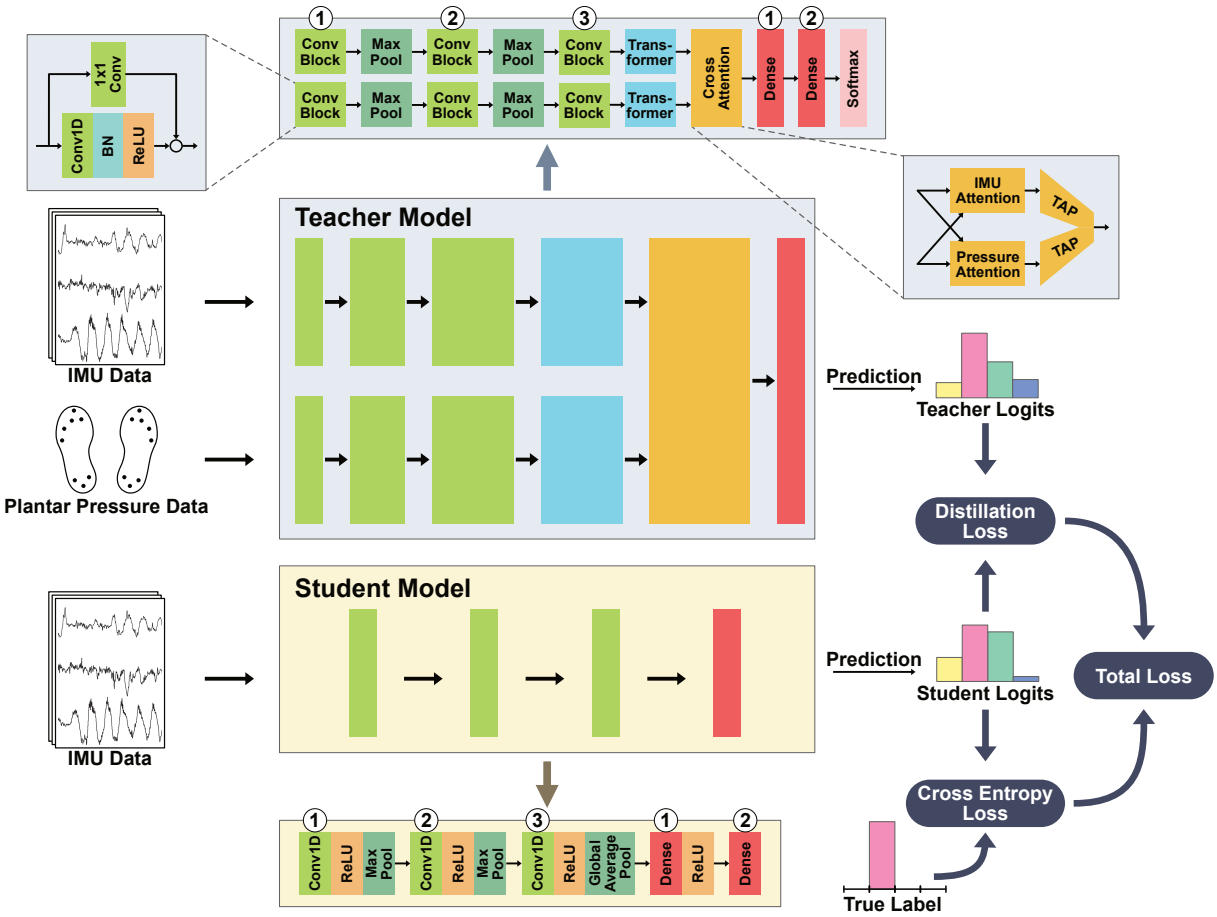


Figure 5.1: Illustrates the architecture of our DualCrossTAP teacher and optimized CNN student models, alongside their knowledge distillation workflow.

This study employs Knowledge Distillation [84] to transfer dark knowledge from a complex teacher model, DualCrossTAP (we named the model DualCrossTAP due to its two-branch architecture, cross-sensor attention, and temporal attention pooling), to lightweight CNN student models, enabling efficient deployment on highly resource-constrained MCUs while prioritizing accuracy and execution speed. The teacher model integrates dual-branch architecture with cross-sensor attention and residual connections to enhance feature representation, while the student models

adopt a minimalist architecture optimized for MCU compatibility. The high-level model architectures and overall methods are outlined in Figure 5.1. Below, we detail the design principles and rationale of the teacher and student models.

5.1 Teacher Model Design

The teacher model, DualCrossTAP, is engineered as a two-branch (dual) architecture to process multimodal data from IMU and PP sensors. The main motivation for such design is the complementary nature of IMU and PP sensors. Extending the analysis conducted by B. Luo et al. [132], we observed that IMU was particularly good at detecting hand-related movements such as *brushing_teeth*, *use_computer*, etc. while PP was good at detecting leg-centric activities such as *walk*, *run*, *cycle*, etc. Hence, a two-branch architecture [134] is ideal to independently extract features from IMU and PP sensors.

For feature extraction within each branch, a series of enhanced *Conv* blocks (see Figure 5.1) are utilized. Each block consists of a 1D convolutional layer, followed by batch normalization and a ReLU activation function. A residual connection is incorporated by adding the input of the block to its output; a 1x1 convolution is applied to the shortcut if the input and output channel dimensions do not match. This convolutional structure is designed to learn hierarchical spatial features from the sensor data. Max pooling layers are interspersed between the convolutional blocks to reduce the dimensionality of the feature maps and provide a degree of translational invariance.

Following the convolutional feature extraction, each branch employs a transformer encoder block [104]. This block consists of a multi-head self-attention mechanism followed by a feed-forward network. Residual connections and layer normalization are applied after both the attention and feed-forward sub-layers. While RNNs, LSTM units, and GRUs are commonly used for sequence modeling in HAR due to their proficiency with temporal dependencies and imbalanced datasets [130], [146], we opted for transformer layers. This decision was based on two primary factors: firstly, the CAPPIMU dataset exhibits a relatively balanced distribution of activity labels (see Figure 4.1), mitigating a key advantage of recurrent architectures in handling class imbalance. Secondly, transformers can leverage parallel processing capabilities of modern accelerators like graphics processing units more effectively than sequential recurrent models [54], leading to significantly faster training times. Furthermore, in scenarios with balanced datasets, transformer architectures often demonstrate superior performance compared to their recurrent counterparts (i.e., RNN, LSTM, GRU) [130]. It is acknowledged that transformer layers are typically more computationally expensive than convolution layer, which can be a significant consideration for resource-constrained MCUs. However, in the context of this study, this computational overhead is permissible for the teacher model, as its primary role is to serve as a source of comprehensive knowledge.

After the individual branch processing, cross-modal attention is applied to allow interaction between the IMU and PP features. Specifically, a multi-head attention mechanism is used where IMU features attend to PP features and vice versa, with

the original features being added to their attended counterparts to enrich the representations. Finally, to aggregate the temporal information from the processed sequences of each modality, a temporal attention pooling layer is utilized [91]. This layer learns to assign different weights to different time steps in the sequence, allowing the model to focus on the most informative segments for the activity recognition task. The attention weights, α_t, for each time step t of an input sequence $X = (x_1, x_2, \ldots, x_T)$ are calculated, using a dense layer with a softmax activation, and then used to compute a weighted sum of the input features. The output, c, can be represented as:

$$\alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)} \tag{5.1}$$

where $e_t = \text{score}(x_t)$ is a learnable score corresponding to the input feature x_t at time step t, e_k represents the same learnable score for the input feature at time step k within the sum, both computed by a small neural network (i.e., a dense layer with a tanh activation), and T is the total number of time steps in the input sequence $X = (x_1, x_2, \ldots, x_T)$.
The final context vector c is then:

$$c = \sum_{t=1}^{T} \alpha_t x_t \tag{5.2}$$

This mechanism allows the model to dynamically emphasize salient temporal features from both the IMU and PP sensor streams before they are concatenated and passed to subsequent dense layers for final classification. The detailed architecture and parameters of the teacher DualCrossTAP model are given in Appendix B. This design yields superior accuracy over existing state-of-the-art models, as validated in Chapter 6.

5.2 MCU and Toolchain Selection

The ESP32-S3 DevKit C1 [103] was selected as the target MCU due to its popularity, balance of computational resources, cost, and compatibility with TinyML workflows. This board features a dual-core Xtensa LX7 processor (240 MHz), 512 KiB SRAM, and 8 MiB Flash, enabling efficient deployment of lightweight models while accommodating memory-intensive operations. To contextualize this choice, Table 5.1 compares alternative MCUs, emphasizing hardware constraints critical for TinyML deployment. Crucially, the selected MCU directly informs the design constraints for the student models; their architectures must be meticulously optimized to ensure that the final compiled binary and runtime memory footprint remains within the available SRAM and Flash memory. This resource-aware design is paramount for successful on-device deployment. Additionally, the utilized method for on-device model testing and validation is visualized and discussed in greater detail in Appendix F.

Similarly, PlatformIO [138] was chosen for its cross-platform board management, enabling seamless transitions between MCUs (e.g., ESP32 to STM32) via unified de-

Table 5.1: Comparison of MCUs that are similar to ESP32-S3 and if desired, could also be used instead of ESP32-S3.

Board	Processor	Dimension (mm)	Price ($)	SRAM/Flash
ESP32-S3 DevKit C-1 [103]	Xtensa LX7 (240 MHz)	25.5 × 18.0	5–8	512 KiB / 8 MiB
Arduino Nano 33 BLE Sense [110]	Cortex-M4 (64 MHz)	18 × 45	25	256 KiB / 1 MiB
STM32 H747XI [149]	Cortex-M7/M4 (480 MHz)	25.5 × 25.5	15–20	1 MiB / 2 MiB
Raspberry Pi Pico W [191]	Cortex-M0+ (133 MHz)	21 × 51	6	264 KiB / 2 MiB
Nordic nRF5340 [187]	Cortex-M33 (128 MHz)	7 × 7	10–12	512 KiB / 1 MiB

pendency resolution. The Arduino framework [137] further ensures portability as its hardware-agnostic Application Programming Interfaces (APIs) allows researchers to adapt this work to other MCUs with minimal code changes, leveraging widespread vendor support. This toolchain combination prioritizes reproducibility while addressing the computational limits of edge devices.

5.3 Student Model Designs

The foremost consideration in designing student models for deployment on resource-constrained MCUs such as the ESP32-S3 is adherence to strict hardware limitations. This necessitates a meticulous selection of network layers and operations, with particular attention given to the resultant total Flash and SRAM consumption. While architectures like RNNs and LSTMs networks are proficient in modeling temporal dependencies inherent in sequential data like IMU and PP signals, their typically substantial computational and memory requirements often render them unsuitable for direct deployment on such severely constrained hardware. The overhead associated with their recurrent connections and gating mechanisms can easily exceed the available resources on MCUs. Consequently, for this study, CNN-based architectures were prioritized, as they can offer a more favorable trade-off between feature extraction capabilities for time-series data and resource efficiency. Besides that, TensorFlow Lite Micro (TFLM) provides optimized kernels for convolution-related operations.

In this context, two distinct CNN-based student models were developed. The first model was engineered with the primary goal of maintaining high classification accuracy, while the second was optimized for superior performance metrics, including inference speed and power efficiency, on the target MCU. The first student model, designated CNN-S1, is conceived as a compact and efficient CNN, specifically tai-

lored for deployment on the ESP32-S3 and similar MCUs (see Section 5.2), whose resource constraints were detailed previously. The principal objective underpinning CNN-S1 is the creation of a lightweight model capable of performing HAR effectively on-device. A secondary, yet crucial, objective is its amenability to knowledge distillation from a more complex teacher model, DualCrossTAP (see Section 5.1), while striving to retain comparable accuracy and precision. To meet these objectives, the architecture of CNN-S1 is intentionally streamlined to minimize its computational and memory footprint. It primarily comprises a sequence of 1D convolutional layers (as illustrated in Figure 5.1), which are particularly adept at extracting salient temporal features from time-series IMU sensor data. These are complemented by max pooling layers to systematically reduce dimensionality and computational load, which is an essential characteristic for low-compute devices. The selection of activation functions, alongside the network's depth and breadth, was carefully calibrated to strike an optimal balance between predictive performance and resource utilization. Subsequently, global average pooling is employed to further curtail computational demands. The terminal stage of CNN-S1 consists of dense layers, which are tasked with classifying the extracted features into distinct 21 activity categories. The overall structure is meticulously designed to ensure that the underlying NN OPS (see [154]), especially when converted for TFLM, are optimized for minimal memory and computational overhead. This design facilitated activity classification with high accuracy while maintaining moderate on-device performance (see Chapter 6).

The second student model, named CNN-S2, shares the foundational architecture (see Appendix C) of CNN-S1 but incorporates a strategic modification aimed at significantly enhancing inference time and reducing its flash memory usage. Specifically, all standard 1D convolutional blocks within CNN-S1 were replaced with 1D depthwise separable convolutional blocks [33], [95]. This architectural adjustment yielded a substantial reduction in the total parameter count by over 42%, leading to markedly lower Flash memory consumption. However, this optimization introduced a trade-off, resulting in increased peak SRAM usage during inference and a discernible reduction in classification accuracy when compared to the CNN-S1 model as detailed in Chapter 6.

5.4 Knowledge Distillation Method

This study employs a standard response-based knowledge distillation technique [79], [83], [106]. The primary objective is not to introduce a novel KD algorithm but rather to adapt and apply established KD principles to a custom requirement: exploring the efficacy of KD-based sensor fusion for HAR using IMU and PP data, specifically targeting resource-constrained MCUs. This investigation addresses a notable gap in the existing HAR literature, as the fusion of IMU and PP sensor modalities through KD for deployment on MCUs has not been explored. The focus on MCUs is particularly pertinent, as these are environments where efficient sensor data utilization and model compression are paramount.

The core of our KD-based sensor fusion strategy is as follows: The teacher model, DualCrossTAP (detailed in Section 5.1), is initially trained using data from both

IMU and PP sensors. Subsequently, during the knowledge distillation process, the student model is trained using input from only a single sensor modality. This approach allows the student model to implicitly learn and benefit from the knowledge of the excluded sensor modality, which is distilled from the more comprehensive understanding of the teacher model. Given the two sensor types, two distinct fusion pathways were investigated:

1. **PP to IMU Fusion**: The student model is trained primarily with IMU sensor data, with knowledge from the PP sensor being distilled from the teacher.

2. **IMU to PP Fusion**: The student model is trained primarily with PP sensor data, with knowledge from the IMU sensor being distilled from the teacher.

Despite exploring both pathways, the strategy prioritizing active IMU data for the student model emerges as the more pragmatic choice for highly resource-constrained devices. This preference is substantiated by several factors. Firstly, the input data dimensionality associated with the IMU sensor is considerably smaller than that of the PP sensor. This directly translates to a reduced input buffer size and a lower number of MFLOPs for the student model. For instance, considering a six-second window and a 50 Hz sampling rate, an IMU with 9 axes (3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer) requires an input buffer for $9 \times 50 \times 6 = 2700$ float (32-bit) values, equating to 10.8 KB. In contrast, a PP sensor system with 16 pressure points (eight pressure points per leg) necessitates an input buffer for $16 \times 50 \times 6 = 4800$ values, or 19.2 KB. Secondly, empirical evidence from existing state-of-the-art models suggests that IMU sensors are inherently capable of distinguishing a broader range of activities more effectively than PP sensors, exhibiting an accuracy advantage of over 12% in some cases [132]. Thirdly, the ergonomic aspects and wearability of the sensors are critical for ensuring user compliance and continuous data acquisition. The IMU sensor, positioned on the right wrist, offers superior comfort and practicality. Its unobtrusive nature allows for consistent wear across a wide range of daily activities, including extended periods of rest and sleep, which is vital for comprehensive and uninterrupted monitoring. On the other hand, the PP sensor, situated on the leg, presents practical limitations for continuous, long-term use. Its placement can be cumbersome and may cause discomfort, particularly during sleep or sedentary states, potentially leading to inconsistent data collection and reduced user adherence. This significantly curtails its suitability for applications requiring persistent, round-the-clock monitoring. Consequently, for deployment on resource-constrained MCUs, selecting the IMU as the primary active sensor for the student model, augmented by distilled knowledge from the PP sensor (PP to IMU Fusion), presents an optimal approach. Nevertheless, for completeness and to provide valuable insights for future research, experiments involving the distillation from IMU to a PP-reliant (IMU to PP Fusion) student model were also conducted. The results, presented in Chapter 6, further corroborate the relative inferiority of the PP sensor as the sole active input in this distilled sensor fusion context.

Algorithm 2 High-level PP to IMU knowledge distillation training process

Require: Pre-trained teacher model $\mathbf{T}$ (accepts IMU and PP data); Student model $\mathbf{S}$ (accepts IMU data only, parameters can be uninitialized or pre-trained); Training dataset $\mathcal{D}_{train} = \{((X_i^{\text{IMU}}, X_i^{\text{PP}}), Y_i)\}_{i=1}^{N}$ where X_i^{IMU} is IMU data, X_i^{PP} is Plantar Pressure data, and Y_i are true labels; Temperature parameter $\tau > 1$; Loss weighting factor $\alpha \in [0, 1]$.

Ensure: Trained student model $\mathbf{S}_{trained}$.

Initialize parameters of student model $\mathbf{S}$ (if not already pre-trained).
for all training epoch e **do**
  for all batch $((X_b^{\text{IMU}}, X_b^{\text{PP}}), Y_b)$ in $\mathcal{D}_{train}$ **do**
    $Z_T \leftarrow \mathbf{T}(X_b^{\text{IMU}}, X_b^{\text{PP}})$ ▷ Compute teacher's logits using both IMU and PP data
    $Z_S \leftarrow \mathbf{S}(X_b^{\text{IMU}})$ ▷ Compute student's logits using only IMU data
    $P_T^{\text{soft}} \leftarrow \text{Softmax}(Z_T/\tau)$ ▷ Calculate teacher's soft targets using temperature τ
    $P_S^{\text{soft}} \leftarrow \text{Softmax}(Z_S/\tau)$ ▷ Calculate student's soft predictions using temperature τ
    $\mathcal{L}_{\text{hard}} \leftarrow \text{CrossEntropyLoss}(\text{Softmax}(Z_S), Y_b)$ ▷ Student's loss against true labels
    $\mathcal{L}_{\text{soft}} \leftarrow \tau^2 \cdot \text{CrossEntropyLoss}(P_S^{\text{soft}}, P_T^{\text{soft}})$ ▷ Student's loss against teacher's soft targets (scaled)
    $\mathcal{L}_{\text{total}} \leftarrow \alpha \cdot \mathcal{L}_{\text{hard}} + (1 - \alpha) \cdot \mathcal{L}_{\text{soft}}$ ▷ Combine hard and soft losses
    Update parameters of $\mathbf{S}$ using gradient of $\mathcal{L}_{\text{total}}$
  end for
end for
$\mathbf{S}_{trained} \leftarrow \mathbf{S}$
return $\mathbf{S}_{trained}$

The employed KD methodology is visually outlined in Figure 5.1. In our primary configuration, the DualCrossTAP teacher model processes inputs from both IMU and PP sensors. Conversely, the student CNN model receives only IMU sensor data as direct input, ensuring a lightweight footprint (see Algorithm 2). This setup facilitates the implicit transfer of knowledge extracted from plantar pressure data by DualCrossTAP to the student CNN, even though the student does not directly process PP data. The knowledge transfer is mathematically formalized using a temperature-scaled Kullback-Leibler (KL) divergence [23] between the logits of the teacher and student models. For a given input $\mathbf{x}$, the total loss function $\mathcal{L}_{\text{total}}$ is a weighted sum of the standard cross-entropy loss and the distillation loss:

$$\mathcal{L}_{\text{total}} = \alpha \underbrace{\mathcal{L}_{\text{CE}}(y, \mathbf{z}_s)}_{\text{Cross-Entropy}} + (1 - \alpha) \underbrace{T^2 \cdot D_{\text{KL}}(\sigma(\mathbf{z}_t/T) \parallel \sigma(\mathbf{z}_s/T))}_{\text{Distillation}} \tag{5.3}$$

In this equation, $\mathbf{z}_t$ and $\mathbf{z}_s$ represent the logits of the teacher and student models, respectively. The temperature parameter T is used to soften the probability distributions derived from the logits, thereby controlling the granularity of the knowledge transferred. The hyperparameter α balances the contribution of the hard target labels (via cross-entropy) and the soft targets from the teacher (via distillation loss). The KL divergence term, D_{KL}, is defined as:

$$D_{\mathrm{KL}} = \mathbb{E}\left[\sum_{i=1}^{C=21} \sigma\left(\frac{\mathbf{z}_t}{T}\right)_i \log\left(\frac{\sigma(\mathbf{z}_t/T)_i}{\sigma(\mathbf{z}_s/T)_i}\right)\right] \tag{5.4}$$

where C is the number of classes (21 in this study), and $\sigma(\cdot)$ denotes the softmax function applied to the temperature-scaled logits. This formulation encourages the student model's output distribution to mimic that of the teacher model.

Chapter 6

Experimental Evaluation

This chapter details the experimental methodology used to evaluate the performance of our composite HAR system on the resource-constrained device (ESP32-S3). The analysis covers model training, optimization techniques, and deployment on the ESP32-S3 board. The experiments were conducted to assess not only classification accuracy but also computational efficiency, including memory (SRAM) and Flash usage, inference latency, and power consumption. We focused on balancing model performance with the inherent hardware limitations of the ESP32-S3, aiming for an optimized NN solution suitable for real-world, low-power applications utilizing our developed end-to-end prototype.

6.1 Experimental Setup

Our experiments were conducted using a high-performance Personal Computer (PC) equipped with a Ryzen 9 5900X CPU, RTX 3090 (24 GB) GPU, and 128 GB of memory, running Pop!_OS 22.04 LTS with the Linux kernel 6.9.3. This setup provided the necessary computational infrastructure for training and deploying ML models. We used Python 3.10 due to its stability, alongside the Scikit-Learn library 1.6 [10] and the TensorFlow 2.17 framework [21] for dataset preprocessing, model development, training-testing, and optimization. Furthermore, we used PlatformIO core version 6.1.16 [138] with CLion [124] for firmware development as PlatformIO simplifies the process of managing vendor-specific toolchains, libraries, and dependencies. Consequently, the ESP32-S3 Dev Kit C1 [103] is a powerful MCU, with a 32-bit, dual-core Tensilica Xtensa LX7 processor running at 240 MHz, and 320 KiB of SRAM with 8 MiB of Flash memory. Primarily, this MCU was selected due to its popularity, low cost, and power efficiency (see Section 5.2). We tested and collected the inference performance of ESP32-S3 using the testing split transmitted via Universal Asynchronous Receiver/Transmitter (UART) [111], simulating real-time data handling (see Appendix F for an indepth discussion). Next, the power consumption was measured utilizing a KWS-MX18L Universal Serial Bus (USB) power monitor [179]. This specific device was selected due to its suitability for long-term power consumption logging and its pass-through USB design. The latter feature significantly streamlines the development workflow by allowing simultaneous serial communication (for data transmission and debugging) and code deployment to the MCU without requiring disconnection or alternative measurement setups, such as traditional USB multimeters, which often preclude concurrent data operations.

6.2 Model Training and CNN-S1 ESP32-S3 Deployment

The DualCrossTAP model was trained for 50 epochs utilizing the training and validation splits detailed in Chapter 4. A batch size of 32 was employed, and the AdamW optimizer [153] was selected with a learning rate of 0.0001 and a weight decay of 0.0001. To mitigate potential overfitting and enhance generalization, ModelCheckpoint callbacks and ReduceLROnPlateau learning rate scheduling strategies were implemented [198].

As anticipated, the DualCrossTAP model demonstrated exemplary performance, achieving an accuracy of 99.32% and an F1 Score of 99.33%, thereby outperforming existing state-of-the-art models in this dataset. A more granular analysis of DualCrossTAP's performance across each of the 21 activities revealed consistently high F1 Scores. A minor exception was observed for the *play_phone* activity, which registered a slightly lower F1 Score of 97%, while the majority of other activities achieved scores of 98% or higher. To rigorously validate the model's robustness and ensure that the observed performance was not a result of overfitting to the specific train-test split, comprehensive 5-Fold and 10-Fold cross-validation procedures were conducted. During these cross-validation experiments, careful attention was paid to preserving temporal dependencies within the data and preventing data leakage between folds. The results of these cross-validation analyses are presented in Table 6.1 and compared with the initial train-test split results. The cross-validation results consistently affirm the effectiveness of the DualCrossTAP model and its dual-branch architecture for robust IMU and PP sensor fusion. The model demonstrates superior performance on both hand-centric and leg-centric activities, underscoring the significant contribution of the plantar pressure sensor data, a modality often overlooked in previous HAR studies. Furthermore, these findings validate the efficacy of the transformer layer within our architecture, particularly when applied to a dataset with balanced class distributions, as evidenced by the model's capacity for effective learning and generalization. Quantitatively, the DualCrossTAP model comprises a total of 690,711 parameters (2.63 MiB in size) and requires 67.90 MFLOPs for inference. This analysis provides insight into the computational requirements of the teacher model, which serves as the basis for knowledge distillation to the more resource-constrained student models.

Table 6.1: Comparison of DualCrossTAP models' performance across different validations.

Validation	**Accuracy**	**F1 Score (macro)**	**Precision (macro)**	**Recall (macro)**
Train Test Split	99.32% ± 0.35	99.33% ± 0.35	99.34% ± 0.27	99.34% ± 0.33
5-Fold	99.60% ± 0.27	99.56% ± 0.16	99.58% ± 0.14	99.56% ± 0.17
10-Fold	99.63% ± 0.25	99.69% ± 0.16	99.65% ± 0.14	97.79% ± 0.16

Next, for the KD-based PP to IMU sensor fusion, the CNN-S1 student model was trained for 1000 epochs utilizing only IMU data (see Algorithm 2). The training

maintained the same optimizer parameters as the teacher model, with a temperature $T = 3.0$ and a loss component weight $\alpha = 0.75$, as defined in Equations 5.3 and 5.4. The resulting distilled model, CNN-S1, achieved a notable performance with 98.35% accuracy and a 98.36% F1 Score. This demonstrates significant performance retention compared to training the student model without teacher guidance (see Table 6.3) and surpasses existing state-of-the-art models. Furthermore, the CNN-S1 model comprises a total of 51,029 parameters (approximately 199.33 KiB). This represents a compression factor exceeding 13× in terms of size compared to the DualCrossTAP teacher model, with only a minimal reduction in accuracy. The computational cost is also significantly reduced, with the CNN-S1 model requiring only 6.14 MFLOPs, making it substantially less computationally intensive than the teacher model.

Similarly, we conducted experiments on KD-based IMU to PP sensor fusion using the aforementioned temperature and loss component settings. As anticipated and discussed in Section 5.4, the CNN-S1 model in this configuration exhibited considerably worse performance, achieving 86.12% accuracy and an 85.90% F1 Score. With a total parameter count of 51,701 (201.96 KiB) and 6.55 MFLOPs, this strategy is significantly less effective than the PP to IMU fusion approach, primarily due to the substantial accuracy drop of over 12%, rendering it impractical for deployment in scenarios where high accuracy is critical. Nevertheless, it is worth noting that even with only the PP sensor data, this KD-based sensor fusion approach still outperforms existing state-of-the-art models by a small margin (see Table 6.2). The comprehensive comparison between the DualCrossTAP teacher model and the CNN-S1 student model, including the performance of both explored sensor fusion strategies, is presented in Table 6.3.

Table 6.2: Comparison of our optimized CNN-S1 model against state-of-the-art approaches, considering only the PP sensor. To compute *Total Parameters* (referred as *Total Params* in the Table), *Trainable Parameters* (referred as *Trainable Params* in the Table), and *MFLOPs* from the study [132], we referred to their cited GitHub repository. As their models are implemented in PyTorch, we employed *ptflops* [139] and *summary* [157] for MFLOPs and model summary calculations.

Study	**Model**	**Total Params**	**Trainable Params**	**MFLOPs**	**Compute**	**F1 Score**
[132]	DeepConv-LSTM	787,669	787,669	988.48	N/A	63.6%
	FCN	194,005	194,005	18.24		85.8%
	DCNN	23,094,975	23,094,975	901.59		76.0%
	TPN	96,437	96,437	49.45		64.2%
	Shallow-LSTMs	655,573	655,573	914.26		56.9%
	CNN-TA	1,107,786	1,107,786	118.80		75.4%
This Study	**CNN-S1**	**51,701**	**51,701**	**6.55**	**ESP32-S3**	**85.90%**

Table 6.3: Comparison between DualCrossTAP and student CNN-S1 model. The superscript 1 represents KD-based PP to IMU sensor fusion, and superscript 2 represents KD-based IMU to PP sensor fusion in the *Model* column.

Model	**Sensor**	**Accuracy**	**F1 Score (macro)**	**Total Params**	**Trainable Params**	**MFLOPs**
Dual-CrossTAP	IMU & PP	99.32% ± 0.35	99.33% ± 0.35	690,711	688,279	67.90
CNN-S1 without KD	IMU	95.63% ± 0.77	95.56% ± 0.86	51,029	51,029	6.14
CNN-S1 with KD[1]	IMU	98.35% ± 0.21	98.36% ± 0.18			
CNN-S1 without KD	PP	82.28% ± 0.32	82.37% ± 0.25	51,701	51,701	6.55
CNN-S1 with KD[2]	PP	86.12% ± 0.26	85.90% ± 0.11			

Before deploying the CNN-S1 model on the ESP32-S3, we evaluated several quantization methods to further optimize its performance while accommodating the device's strict resource constraints. Table 6.4 provides a detailed comparison of these techniques in terms of accuracy, F1 Score, precision, model size, and deployability. The Float-16 quantization achieves nearly lossless accuracy, precision, and F1 Score while reducing the model size by roughly 50% compared to the default 32-bit float configuration. However, as the ESP32-S3 does not natively support 16-bit floating point operations and TFLM offers no accommodation for this format; the anticipated reduction cannot be fully exploited. Similarly, mixed precision quantization maintains competitive performance (approximately 97.6% accuracy) yet still relies on 16-bit floats. Furthermore, the use of heterogeneous numerical representations (16-bit float and 8-bit integer) across different layers incurs additional memory overhead for quantization and dequantization OPS, making this method less practical despite achieving a model size reduction of over 67%. In contrast, 8-bit integer quantization significantly improves inference latency by leveraging optimized 8-bit integer kernels available in TFLM. However, the associated accuracy drop of roughly 6% precludes its adoption. Hence, the non-quantized CNN-S1 model remains the preferred candidate for deployment on the resource-constrained ESP32-S3.

Table 6.4: Comparison of various quantization methods applied to the student CNN-S1 model.

Method	Accuracy	F1 Score (macro)	Precision (macro)	CNN-S1 Size	Deployable
None	98.35%	98.36%	98.36%	209.108 KB	✔
Float-16	98.34%	98.33%	98.32%	108.116 KB	✖
MP[a]	97.60%	97.63%	97.65%	67.000 KB	✖
Integer-8	92.23%	91.38%	93.05%	67.016 KB	✔

[a] Here MP stands for Mixed Precision.

Table 6.5: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 240 MHz. Execution times were measured as the average of 5000 randomly sampled inference runs, with detailed per-operation latency profiles collected during benchmarking. During the conversion from TensorFlow to TFLite format, the converter replaced all 1D convolutions with 2D convolutions, as native support for 1D convolution is currently unavailable in TFLM. Since 1D convolution is a special case of 2D convolution [60], [116], the TFLite converter introduces a *Reshape* layer to transform 1D data into 2D format, ensuring compatibility. Finally, the last row reports peak SRAM utilization of 62 KB and the total execution time (inference latency) of 1,836,271 micro seconds.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	7	83
	STRIDED_SLICE	32	
	PACK	9	
	RESHAPE	35	
Convolution	CONV_2D	154200	154200
Max Pool	MAX_POOL_2D	6035	6035
Convolution	CONV_2D	517005	517005
Max Pool	MAX_POOL_2D	6011	6011
Convolution	CONV_2D	1137845	1137845
Global Average Pooling	MEAN	11927	11927
Dense	FULLY_CONNECTED	2672	2672
Output (Dense)	FULLY_CONNECTED	446	493
	SOFTMAX	47	
	62 KB (SRAM)		**1836271**

Table 6.6: Performance evaluation of the ESP32-S3 using the CNN-S1 student model at various CPU frequencies. In *DFS* mode, the CPU dynamically operates between a minimum of 80 MHz and a maximum of 240 MHz. The displayed CPU temperatures were recorded using the ESP32-S3's built-in sensor.

Mode	**CPU Frequency (MHz)**	**CNN-S1 Inference Latency (µs)**	**Power Consumption (mW)**	**CPU Temperature (K)**
DFS	min: 80 and max: 240	1836941	212	306.70
Fixed/Constant	240	1836278	210	306.75
	160	2705835	130	305.75
	80	5354446	101	305.70
	40	10959680	64	304.75
	20	23005809	56	303.75
	10	50918089	48	304.75

The finalized CNN-S1 model was converted to TensorFlow Lite format with default optimizations, transpiled to a C header array using xxd tool [127]. We then imported the transpiled C header file in the PlatformIO project, resolved the necessary dependencies for TFLM (for ease of use and deployment, we used our developed PlatformIO library [174]), substituted the appropriate OPS, and deployed via PlatformIO to the ESP32-S3 Dev Kit C1.

The deployed CNN-S1 model attains 98.355% accuracy and 98.365% F1 Score, thereby meeting our performance expectations. Furthermore, the model demonstrates an inference latency of 1,836,288 microseconds approximately (see Table 6.5 for detailed breakdown) and exhibits a peak SRAM usage of 62 KB during operation, with a total model footprint of 209.108 KB. On the ESP32-S3 platform, the overall memory consumption is 91.636 KB of SRAM and 552.649 KB of Flash. Within the SRAM allocation, the CNN-S1 model uses 62 KB while the input buffer requires 10.8 KB; the remaining memory is apportioned to system components such as FreeRTOS and UART etc. Similarly, in Flash memory, the model occupies 209.108 KB, with the rest being utilized by FreeRTOS, UART, string literal, and additional system functionalities. Additionally, the ESP32-S3 operated at a constant 240 MHz without engaging DFS, and an average power consumption of 210 mW was recorded throughout the testing phase. It is important to note that the reported power consumption corresponds to the entire ESP32-S3 Dev Kit C1 (board), rather than solely to the chip itself [150], [152].

In typical use-case scenarios, the battery-powered MCU must balance low power consumption with acceptable performance to extend battery life. To identify the optimal configuration for the ESP32-S3, we evaluated both fixed frequency settings (ranging from 10 to 240 MHz) and DFS mode (see Table 6.6). In our experiments, we measured CNN-S1 inference latency, power consumption, and CPU temperature using the onboard temperature sensor to assess performance and determine whether

additional cooling might be required.

Table 6.6 shows that as the CPU frequency decreases, the inference latency increases markedly. For example, while the fixed 240 MHz configuration produces an inference latency of approximately 1.84 seconds with a power draw of about 210 mW, operating at 160 MHz increases the latency to roughly 2.7 seconds while reducing power consumption to 130 mW. Conversely, frequencies below 80 MHz result in dramatically higher latencies exceeding 10 seconds at 40 MHz and also disable key functionalities such as Wi-Fi and Bluetooth, making these configurations impractical. Notably, when using DFS, the CPU dynamically ramps up to the maximum frequency (240 MHz) during inference, yielding performance identical to the constant 240 MHz frequency. As a result, considering both power efficiency and inference latency, the fixed 160 MHz configuration provides the most balanced performance. Detailed per-layer and per-operation execution time breakdowns for each CPU frequency setting are provided in Appendix D.

6.3 CNN-S2 Training and ESP32-S3 Deployment

Consistent with the training methodology employed for the CNN-S1 student model in the PP to IMU sensor fusion context, the CNN-S2 student model was also trained for 1000 epochs. This training utilized identical optimizer parameters, temperature settings, and loss components as its predecessor. As hypothesized in Section 5.3, the CNN-S2 model demonstrates superior parameter efficiency compared to CNN-S1. Specifically, CNN-S2 comprises 30,288 parameters (approximately 118.31 KiB) and requires 2.19 MFLOPs. This configuration represents a significant reduction of 20,741 parameters relative to CNN-S1. More notably, it achieves substantial compression of 657,991 parameters (an approximate 23× reduction) when compared to the teacher DualCrossTAP model. Furthermore, the reduction in the computational load, evidenced by a decrease of nearly 4 MFLOPs compared to CNN-S1, offers a considerable improvement in terms of inference latency. However, these enhancements in efficiency are accompanied by a marginal trade-off in performance. The CNN-S2 model achieves accuracy and F1 Scores of 97.681% and 97.685% (macro), which, on average, are approximately 0.7% lower than those recorded for CNN-S1. Despite this noticeable decrease, CNN-S2 successfully learned from the teacher DualCrossTAP model, retaining a substantial portion of its performance capabilities, albeit to a lesser extent than CNN-S1. An examination of per-activity classification reveals that CNN-S2 maintained a consistent F1 Score across most activities, with the *play_phone* activity exhibiting the lowest performance. For ease of comprehensive comparative analysis, the performance metrics of CNN-S1 and CNN-S2 are detailed in Table 6.7.

Following a similar investigative approach as with the CNN-S1 model, various quantization methods were explored for the CNN-S2 architecture. The primary objective was to identify an optimal configuration that is less resource-intensive, thereby ensuring efficient deployment on MCUs. A detailed comparative analysis of these techniques, evaluating their impact on accuracy, F1 Score (macro), precision (macro), model size, and deployability, is presented in Table 6.8. For reasons previously articulated, namely limitations within the TFLM framework and the ESP32-S3 hard-

ware, float-16 and mixed precision quantization methods could not be ultimately utilized. This was despite the promising performance of 16-bit floating-point quantization, which, as indicated in Table 6.8, achieved an F1 Score of 97.67% and an accuracy of 97.66%. These metrics are almost identical to those of the non-quantized model (97.68% F1 Score and accuracy) while reducing the model size from 127.984 KB to 68.952 KB. Conversely, the 8-bit integer-quantized model is supported by TFLM and is deployable on the target hardware, offering a comparable model size of 50.856 KB similar to mixed precision. However, this method resulted in a significant degradation in performance, with the F1 Score dropping to 89.53% and accuracy to 89.92% from the 97.68% achieved by the non-quantized model. This substantial decrease in classification performance outweighs the benefits of model size reduction and inference latency. Consequently, considering the trade-off between model size, computational resources, and classification efficacy, the non-quantized (vanilla) CNN-S2 model, with its superior F1 Score of 97.68% and manageable size of 127.984 KB, was determined to be the most suitable choice for deployment on the ESP32-S3. Therefore, all subsequent experimentation and deployment procedures proceeded with the vanilla CNN-S2 model, and the quantized alternatives were discarded.

Table 6.7: Comparisons between CNN-S1 and CNN-S2 student models along with and without KD-based PP to IMU sensor fusion.

Model	**Accuracy**	**F1 Score (macro)**	**Total Parameters**	**Trainable Parameters**	**MFLOPs**
CNN-S1 without KD	95.63% ± 0.77	95.56% ± 0.86	51,029	51,029	6.14
CNN-S1 with KD	98.35% ± 0.21	98.36% ± 0.18			
CNN-S2 without KD	94.22% ± 0.55	94.21% ± 0.49	30,288	30,288	2.19
CNN-S2 with KD	97.68% ± 0.39	97.68% ± 0.35			

Table 6.8: Comparison of various quantization methods applied to the student CNN-S2 model.

Method	**Accuracy**	**F1 Score (macro)**	**Precision (macro)**	**CNN-S2 Size**	**Deployable**
None	97.68%	97.68%	97.92%	127.984 KB	✔
Float-16	97.66%	97.67%	97.92%	68.952 KB	✘
MP[a]	96.49%	96.43%	96.61%	50.840 KB	✘
Integer-8	89.92%	89.53%	91.18%	50.856 KB	✔

[a] Here MP stands for Mixed Precision.

Table 6.9: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 240 MHz. Execution times were measured as the average of 5000 randomly sampled inference runs, with detailed per-operation latency profiles collected during benchmarking. During the conversion from TensorFlow to TFLite format, the converter replaced all 1D separable convolutions with 2D separable convolutions (referred as *Sep Convolution* in *Layer* column), as native support for 1D convolution is currently unavailable in TFLM. Since 1D convolution is a special case of 2D convolution [60], [116], the TFLite converter introduces a *Reshape* layer to transform 1D data into 2D format, ensuring compatibility. Finally, the last row reports peak SRAM utilization of 82 KB and the total execution time (inference latency) of 639,543 micro seconds.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	13	99
	STRIDED_SLICE	36	
	PACK	14	
	RESHAPE	36	
Sep Convolution	DEPTHWISE_CONV_2D	7347	61768
	CONV_2D	54421	
Max Pool	MAX_POOL_2D	6049	6049
Sep Convolution	DEPTHWISE_CONV_2D	12883	188831
	CONV_2D	175948	
Max Pool	MAX_POOL_2D	6012	6012
Sep Convolution	DEPTHWISE_CONV_2D	12844	361670
	CONV_2D	348826	
Global Average Pooling	MEAN	11939	11939
Dense	FULLY_CONNECTED	2679	2679
Output (Dense)	FULLY_CONNECTED	447	496
	SOFTMAX	49	
	82 KB (SRAM)		**639543**

Upon deployment on the ESP32-S3 MCU, the CNN-S2 model performed as anticipated, achieving an accuracy of 97.68% and an F1 Score (macro) of 97.68%. The resource utilization metrics are detailed in Table 6.10. Specifically, the inference latency for the CNN-S2 model was measured at approximately 639,543 microseconds. This represents a significant improvement, being nearly three times lower than the 1,836,271-microsecond latency recorded for the CNN-S1 model. In terms of memory, the CNN-S2 model exhibited a peak SRAM consumption of 82 KB during inference and a Flash memory footprint of 127.984 KB. The increase in model SRAM con-

sumption (82 KB for CNN-S2 compared to 62 KB for CNN-S1) is attributable to the architectural design employing *depthwise separable convolutional* layers, as discussed in Section 5.3. Unlike standard *convolutional* layers, which translate to a single *CONV_2D* operation in TFLM, depthwise separable convolutions correspond to two distinct operations (*DEPTHWISE_CONV_2D* and *CONV_2D*). This structural difference inherently requires more operational memory. However, this design choice yields considerable advantages, including a reduced model Flash size (127.984 KB for CNN-S2 versus 209.108 KB for CNN-S1) and the aforementioned substantial reduction in inference latency. The total system-wide SRAM consumption on the ESP32-S3 platform when running the CNN-S2 model was 111.676 KB. This comprised 82 KB for the CNN-S2 model itself, 10.8 KB for the input buffer, and the remainder allocated to system-level components such as FreeRTOS, UART drivers, and string literals. Similarly, the total Flash memory utilization for the system was 476.457 KB, of which the model accounted for 127.984 KB, with the remainder consumed by other system software and data. Consistent with the testing protocol for CNN-S1, the ESP32-S3 was configured to operate at a constant CPU frequency of 240 MHz, without DFS or other power-saving alterations. Under these conditions, the average power consumption during the entire testing period for CNN-S2 was measured at 215 mW. This represents a minor increase of approximately 5 mW compared to the CNN-S1 model (210 mW). We suspect that this increased power draw is a consequence of the higher number of OPS executed by the *depthwise separable convolutional* layers and the associated rise in active memory utilization.

Table 6.10: Comparison of the deployed CNN-S1 and CNN-S2 student models on the ESP32-S3 MCU. The *Model SRAM* and *Model Flash* columns indicate the SRAM and Flash memory utilization specific to each model, while the *Total SRAM* and *Total Flash* columns represent the overall system-wide SRAM and Flash memory consumption on the ESP32-S3. Lastly, column *IL* represents *Inference Latency* of the respective student model.

Student Model	**F1 Score (macro)**	**IL**	**Model SRAM**	**Model Flash**	**Total SRAM**	**Total Flash**	**Power Draw**
CNN-S1	98.36%	1836271 µs	62 KB	209.108 KB	91.636 KB	552.649 KB	210 mW
CNN-S2	97.68%	639543 µs	82 KB	127.984 KB	111.676 KB	476.457 KB	215 mW

Table 6.11: Performance evaluation of the ESP32-S3 using the CNN-S2 student model at various CPU frequencies. In *DFS* mode, the CPU dynamically operates between a minimum of 80 MHz and a maximum of 240 MHz. The displayed CPU temperatures were recorded using the ESP32-S3's built-in sensor.

Mode	**CPU Frequency (MHz)**	**CNN-S1 Inference Latency (µs)**	**Power Consumption (mW)**	**CPU Temperature (K)**
DFS	min: 80 and max: 240	639529	216	307.75
Fixed/Constant	240	639543	215	307.7
	160	959294	144	306.75
	80	1933541	119	305.75
	40	3958349	75	304.75
	20	8304935	66	303.75
	10	18403205	60	304.75

Table 6.12: Comparison of our optimized CNN-S1 and CNN-S2 models against state-of-the-art approaches, considering only the IMU sensor. To compute *Total Parameters*, *Trainable Parameters*, and *MFLOPs* from the study [132], we referred to their cited GitHub repository. As their models are implemented in PyTorch, we employed *ptflops* [139] and *summary* [157] for MFLOPs and model summary calculations.

Study	**Model**	**Total Params**	**Trainable Params**	**MFLOPs**	**Compute**	**F1 Score**
[132]	DeepConv-LSTM	558,293	558,293	606.03	N/A	82.8%
	FCN	192,213	192,213	17.16		98.1%
	DCNN	22,530,495	22,530,495	507.14		96.6%
	TPN	91,061	91,061	46.47		88.6%
	Shallow-LSTMs	426,197	426,197	531.80		79.1%
	CNN-TA	731,466	731,466	66.96		97.8%
This Study	**CNN-S1**	**51,701**	**51,701**	**6.55**	**ESP32-S3**	**98.36%**
	CNN-S2	**30,288**	**30,288**	**2.19**		**97.68%**

To extend battery life, we conducted further experiments to fine-tune the operational parameters of the CNN-S2 model deployed on the ESP32-S3, similar to CNN-S1. This involved evaluating performance across a range of fixed CPU frequencies (from 10 MHz to 240 MHz) and utilizing DFS. The primary objective was to identify an optimal balance between inference latency and power consumption. Concurrently, CPU temperatures were monitored using the ESP32-S3's internal sensor to assess

whether external thermal management solutions would be necessary. The comprehensive results of this investigation are presented in Table 6.11. A general observation, consistent with initial deployment metrics, is that the CNN-S2 model tends to exhibit slightly higher power consumption at equivalent CPU frequencies compared to CNN-S1. However, the overall patterns of how inference latency, power draw, and CPU temperature scale with frequency for CNN-S2 are strikingly similar to those observed for CNN-S1. Consequently, many of the same conclusions regarding frequency selection apply, albeit with the significant caveat that CNN-S2 is considerably faster across the board. For instance, CNN-S2 operating at a fixed 160 MHz achieves an inference latency of 959,294 µs. This is approximately 1.9 times faster than the CNN-S1 model operating at its peak performance at 240 MHz (which had a latency of 1,836,271 µs, as noted in prior comparisons), and CNN-S2 achieves this superior speed while consuming less power (144 mW at 160 MHz for CNN-S2 versus 210 mW at 240 MHz for CNN-S1). Based on the data in Table 6.11, the 160 MHz fixed frequency again emerges as a highly favorable operating point, offering an excellent compromise between performance and energy efficiency for CNN-S2. While it consumes more power (144 mW) and results in a slightly higher CPU temperature (306.75 K) compared to the 80 MHz setting (119 mW, 305.75 K), the nearly halved inference latency (959,294 µs vs. 1,933,541 µs) justifies this trade-off. The DFS mode, operating between 80 MHz and 240 MHz, delivered performance (639,529 µs latency, 216 mW power) closely mirroring the fixed 240 MHz setting. Across all tested frequencies, the CPU temperatures for CNN-S2 remained modest, ranging from approximately 303.75 K to 307.75 K, indicating that active external cooling is not required for the ESP32-S3 under these workloads. The conclusion is also true for CNN-S1, which depicts a similar temperature range (see Section 6.2). For a more granular understanding of the model's execution characteristics, a detailed breakdown of per-layer and per-operation execution times at the utilized CPU frequencies is provided in Appendix E. Lastly, in Table 6.12 we compare our optimized and deployed CNN-S1 and CNN-S2 models against existing state-of-the-art models which validate the effectiveness of our employed strategy.

6.4 Validation on External Datasets

In the preceding sections, we concentrated exclusively on the CAPPIMU dataset to investigate the influence of plantar pressure sensor data within our bidirectional knowledge distillation framework. This focused analysis yielded valuable insights into the interaction between these sensors and our deep learning pipelines. In this section, we extend our evaluation to additional HAR datasets in order to assess the generalizability of our methods beyond plantar pressure and IMU modalities. In essence, we aim to demonstrate the robustness and adaptability of our models across different contexts.

To achieve this goal, we utilize two well-established public benchmark datasets, namely the KU HAR dataset and the NTNU Children HAR dataset. Each of these datasets presents unique sensor configurations and demographic profiles, offering complementary perspectives on model performance in real-world scenarios. Details on the KU HAR dataset, including its data collection protocol and activity taxonomy, are provided in Section 4.2. A comprehensive overview of the NTNU Children

HAR dataset and its distinctive features are given in Section 4.3.

6.4.1 Validation on KU HAR Dataset

To validate the robustness and adaptability of our proposed knowledge distillation framework, we extended our evaluation to the KU HAR dataset. The model training and testing procedures were identical to the methodology detailed in Chapter 5 for the CAPPIMU dataset. The primary distinction in this validation lies in the data modalities used for knowledge transfer. Instead of IMU and PP sensors, this experiment utilized data from accelerometer and gyroscope sensors. Following the knowledge distillation strategy described in Section 5.4, the complex teacher model, DualCrossTAP, was trained on both accelerometer and gyroscope data to establish a performance benchmark. Subsequently, the lightweight student models, CNN-S1 and CNN-S2, were trained exclusively on the accelerometer data. During this process, knowledge related to the gyroscope modality was distilled from the comprehensive teacher model and transferred to the student models. The accelerometer sensor data was selected primarily because it offers more distinguishable characteristics than gyroscope data, thereby enabling superior performance [14], [101], [143]. The performance metrics obtained from this evaluation are summarized in Table 6.13.

Table 6.13: Obtained performance of DualCrossTAP, CNN-S1, and CNN-S2 models' on KU HAR dataset utilizing our knowledge distillation method.

Model	**Accuracy**	**F1 Score (macro)**	**Total Params**	**Trainable Params**	**MFLOPs**
Dual CrossTAP	98.43% ± 0.65	98.38% ± 0.55	687,508	685,076	66.44
CNN-S1 without KD	93.33% ± 0.27	90.47% ± 0.16	50,066	50,066	5.80
CNN-S1 with KD	94.93% ± 0.31	93.39% ± 0.18			
CNN-S2 without KD	77.46% ± 0.82	71.49% ± 0.75	29,691	29,691	2.06
CNN-S2 with KD	91.08% ± 0.26	89.80% ± 0.21			

The results presented in the table clearly demonstrate the efficacy of the employed knowledge distillation method. The DualCrossTAP teacher model, leveraging both accelerometer and gyroscope data, achieves comparative performance (see Chapter 8) with an accuracy of 98.43% and an F1 Score of 98.38%. This result serves as the upper performance threshold, albeit at a significant computational cost, indicated by its approximately 687,508 parameters and 66.44 MFLOPs.

When trained without KD, the student models, which only process accelerometer data, exhibit considerably lower performance. The CNN-S1 model achieves a respectable accuracy of 93.33%, while the more compact CNN-S2 model struggles,

attaining an accuracy of only 77.46%. However, the application of our KD yields substantial improvements for both student models. For CNN-S1, KD boosts accuracy by 1.6%, and the F1 Score by nearly 3 percent (93.39%). The impact of KD is even more pronounced for the CNN-S2 model. Its accuracy experiences a remarkable increase of over 13.6%, rising to 91.08%. Similarly, its F1 Score improves dramatically by more than 18.3%, reaching 89.80%. This significant enhancement elevates the CNN-S2 model from a poor-performing baseline to a highly effective classifier, demonstrating that our KD framework successfully imparts the rich discriminatory features from the gyroscope modality.

6.4.2 Validation on NTNU Children HAR Dataset

To further assess the generalizability and practical applicability of our framework, we conducted a validation study using the NTNU Children HAR dataset. This dataset is particularly significant as it involves a unique demographic, children, including those with CP. This allowed us to test our method's effectiveness in a more challenging and clinically relevant scenario. The specific data preprocessing pipelines employed for this dataset are already detailed in Section 4.3.

Table 6.14: Obtained performance of DualCrossTAP, CNN-S1, and CNN-S2 models' on NTNU HAR dataset utilizing our knowledge distillation method.

Model	**Accuracy**	**F1 Score (macro)**	**Total Params**	**Trainable Params**	**MFLOPs**
Dual CrossTAP	95.00% ± 0.15	94.22% ± 0.17	684,681	682,249	11.05
CNN-S1 without KD	85.81% ± 0.15	86.99% ± 0.18	48,647	48,647	0.97
CNN-S1 with KD	89.87% ± 0.91	87.64% ± 0.28			
CNN-S2 without KD	85.53% ± 0.82	86.25% ± 0.25	28,272	28,272	0.36
CNN-S2 with KD	91.08% ± 0.26	89.80% ± 0.21			

We applied the same knowledge distillation methodology used in the previous experiments. However, a key difference in this validation lies in the definition of the data modalities. Unlike the previous datasets which utilized different sensor types, the NTNU dataset provides data from two identical sensors (accelerometers) placed at two different body locations: the back and the thigh. Following our framework, the DualCrossTAP teacher model was trained on the combined data from both sensor locations to create a comprehensive, high-performance model. The student models, CNN-S1 and CNN-S2, were subsequently trained using only the data from the back-mounted accelerometer.

The decision to use the back sensor data for the student models was deliberate and based on practical considerations. For younger children, a sensor placement on the back is generally less obtrusive and more comfortable than a sensor on the thigh. This is crucial for ensuring high user adherence and for capturing natural, unhindered movement, especially for children with motor impairments. Furthermore, a sensor on the back provides a more holistic representation of a person's movement, which is often more informative for classifying general physical activities compared to data from a single limb. The performance metrics obtained from this experiment are presented in Table 6.14.

The experimental results from the NTNU dataset, as shown in the Table 6.14, further substantiate the effectiveness of our knowledge distillation strategy. The teacher model, DualCrossTAP, which benefits from sensor data from both the back and thigh, sets a high benchmark with an accuracy of 95.00% and an F1 Score of 94.22%, matching existing state-of-the-art performance (see Chapter 8). This demonstrates the value of using multiple sensor locations for activity recognition in this complex dataset. When trained without KD, the student models, relying solely on the back accelerometer, achieve similar baseline accuracies around 85.5% to 85.8%. This suggests that the back sensor alone provides a solid foundation for activity recognition, but it fails to capture the more nuanced information available from the thigh sensor. Upon applying knowledge distillation, both student models show marked improvement. The CNN-S1 model's accuracy increases by over 4 % to 89.87%. The improvement is even more significant for the more compact CNN-S2 model, which sees its accuracy climb by approximately 5.5% to 91.08% and its F1 Score increase by 3.55%.

This outcome is particularly important because it proves that our framework can successfully transfer knowledge between different spatial modalities, not just different sensor types. The distilled knowledge from the thigh-mounted sensor effectively compensates for its physical absence in the student models. Consequently, the CNN-S2 model, with KD, achieves an accuracy that is only 4% lower than the dual-sensor teacher model, while being drastically more efficient with only 4.1% of the parameters and 3.3% of the computational cost. This validation confirms that our method can produce highly accurate and efficient single-sensor models, which is a critical requirement for creating practical and user-friendly HAR systems for children.

In conclusion, this validation on the KU HAR and NTNU Children HAR dataset confirms that our KD approach enables the development of lightweight, computationally efficient models that operate on a single sensor modality without a drastic sacrifice in performance. The student models, particularly CNN-S2, offer a compelling trade-off, achieving performance metrics that approach those of the complex teacher model while possessing only a fraction of its parameters.

Chapter 7

End-to-End Prototype Development

While previous sections focused on model evaluation and performance metrics in controlled settings, such analyses may not fully capture the complexities of real-world operational environments. Therefore, to validate the practical viability of our approach and to gain a comprehensive understanding of the engineering aspects involved, we developed a complete end-to-end prototype using CAPPIMU dataset since this dataset demands the most challenging requirements. This endeavor provides critical insights into the entire process, from sensor data acquisition to model inference, benefiting practitioners, engineers, and manufacturers in the development of actual low-cost HAR products.

7.1 Hardware Configuration and IMU Sensor Integration

The hardware configuration for this end-to-end system is foundational. Given our work with IMU data, and to ensure consistency with the CAPPIMU dataset, we selected the precise IMU sensor employed by the dataset's authors. The choice of sensor is crucial, as parameters such as scale factors, internal filters, and axis orientations can vary significantly between different IMU models. Utilizing an alternative sensor would necessitate considerable recalibration and data transformation efforts to align its output with the characteristics of the original dataset, potentially introducing inconsistencies. Consequently, for reasons of fidelity and reproducibility, we employed the WitMotion BWT901CL [162] IMU sensor. This sensor was interfaced with an ESP32-S3 Dev Kit C1 microcontroller via a USB to Transistor–Transistor Logic (TTL) converter. The overall hardware setup is depicted in Figure 7.1. The BWT901CL IMU sensor is powered by a 3.3 V pin from the ESP32-S3, with its ground pin connected to the ESP32-S3's GND pin. The BWT901CL's Transmit (TX) and Receive (RX) pins are connected to General-Purpose Input/Output (GPIO) pins 16 and 15, respectively, on the ESP32-S3 board, facilitating UART communication.

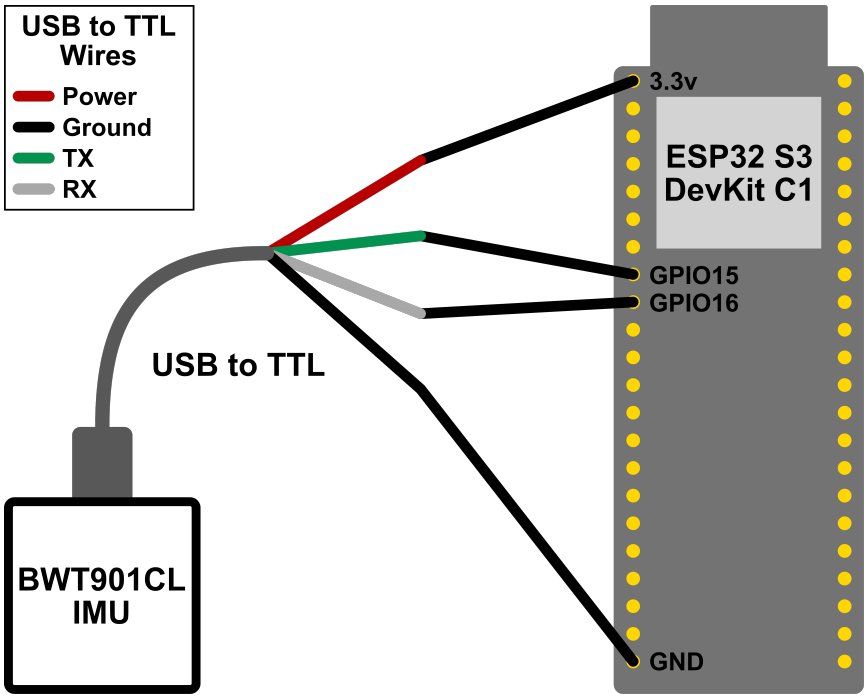


Figure 7.1: High-level representation of the hardware configuration for the end-to-end prediction prototype system, featuring the ESP32-S3 Dev Kit C1 and the WitMotion BWT901CL IMU sensor. The WitMotion BWT901CL IMU sensor is powered by a 3.3 V supply from the ESP32-S3 Dev Kit C1, while the ESP32-S3 Dev Kit C1 itself is powered via USB connection to the workstation.

7.2 High-Level Operational Workflow

Before deploying the machine learning models, it is essential to delineate the operational workflow, from raw data collection to the final inference output. The high-level steps are as follows –

1. **Data Acquisition from IMU Sensor**: The initial step involves collecting a single window of sensor data. In our configuration, this corresponds to 300 timestamps, captured at a sampling rate of 50 Hz over a six-second interval.

2. **Data Preprocessing**: Subsequently, the acquired window of raw sensor data undergoes a two-stage preprocessing. First, the raw 16-bit integer values are converted into their respective physical units using sensor-specific scale factors. Second, the data is normalized using Z-Score normalization, as detailed in Chapter 4 and Equation 7.1, aligning with the methodology used for the CAPPIMU dataset.

3. **Model Inference**: Finally, the preprocessed data window is fed into the deployed TFLM model on the ESP32-S3 to obtain the activity prediction. This cycle of data collection, preprocessing, and inference is then repeated for continuous operation.

7.3 Real-Time Data Collection and Preprocessing

While the inference phase has been discussed previously, the specifics of data acquisition and preprocessing in the deployment context warrant further detail. To streamline data collection from the BWT901CL IMU sensor, we utilized the generic C library provided by WitMotion [163], which is compatible with C/C++ frameworks, including the Arduino framework used for the ESP32-S3. This library simplifies the process of reading sensor data via UART by handling low-level communication protocols and providing a callback mechanism via its API that is invoked when new data is available.

The data received from the sensor via this library is in a 16-bit integer (short) format. These raw integer values for accelerometer, gyroscope, and magnetometer readings must first be converted to meaningful physical units (e.g., m/s^2 for acceleration, $^\circ/s$ for angular velocity). This conversion is achieved by applying specific scale factors, as derived from the sensor's datasheet [164]. For instance, the accelerometer data is scaled using a factor of $(16.0 \times 9.8)/32768.0$, and gyroscope data by $2000.0/32768.0$. After this conversion, the sensor data for each of the nine axes (three for accelerometer, three for gyroscope, and three for magnetometer) is normalized. This normalization employs the Z-Score method (see Equation 7.1), utilizing the per-axis mean (μ) and standard deviation (σ) values (a total of 18 parameters) pre-calculated from the CAPPIMU dataset following dataset's authors. Once these preprocessing steps are completed, the data is in the appropriate format and scale for input to the deployed inference model.

$$z = \frac{x - \mu}{\sigma} \tag{7.1}$$

7.4 Model Deployment and Overall End-to-End System Analysis

For deployment and evaluation on the end-to-end prototype system, both the CNN-S1 and CNN-S2 student models were selected. This choice was guided by their distinct characteristics, as previously detailed in Section 5.3: CNN-S1 is optimized for higher accuracy, whereas CNN-S2 is designed for superior inference latency. To comprehensively assess system performance under varying computational loads, both models were tested on the ESP32-S3 MCU operating at two distinct CPU frequencies: 240 MHz and 160 MHz. This dual-frequency approach allows for an analysis of the trade-offs between processing speed and power consumption, particularly relevant given the inclusion of the IMU sensor as an additional power-consuming component. Although our prior investigations indicated 160 MHz as an optimal frequency for balancing performance and power efficiency, results for 240 MHz are also presented. The higher frequency, despite its increased power draw, offers reduced inference latency, and documenting its performance provides valuable data for practitioners and engineers considering different operational priorities. The detailed performance metrics for these deployment configurations are presented in Table 7.1.

Table 7.1: Performance metrics collected from the end-to-end prototype deploying the student models CNN-S1 and CNN-S2 on the ESP32-S3 at operating frequencies of 240 MHz and 160 MHz. The *F* (*Frequency*) column indicates the constant operating frequency for each model. *RDCT* (Raw Data Collection Time) denotes the duration required to capture one window of data utilizing BWT901CL IMU sensor, while *FCT* (Feature Conversion Time) represents the time taken to convert the raw data into physical units using sensor-specific scale factors. *NT* (Normalization Time) indicates the time required to normalize the data window, and *IL* (Inference Latency) measures the duration from feeding the normalized input to obtaining the model's prediction. Finally, *TS* and *TF* refer to the total allocations of SRAM and Flash memory, respectively, and *PD* reports the corresponding average power draw.

Model	**F (MHz)**	**RDCT (µs)**	**FCT (µs)**	**NT (µs)**	**IL (µs)**	**TS (KB)**	**TF (KB)**	**PD (mW)**
CNN-S1	240	6000010	73	550	1844941	92.22	545.749	275
	160	6000015	95	1025	2705335			195
CNN-S2	240	6000008	77	558	642313	112.26	469.573	295
	160	6000012	93	1028	959294			205

Analyzing the performance of the CNN-S1 model, as shown in Table 7.1, reveals that at an operating frequency of 240 MHz, the total time to process one six-second window of data including inference (sum of RDCT, FCT, NT, and IL) is $6,000,010\,\mu s + 73\,\mu s + 550\,\mu s + 1,844,941\,\mu s = 7,845,574\,\mu s$ (approximately 7.85 seconds). This processing duration is considered acceptable for composite human activity recognition tasks, which often inherently involve longer data windows and do not typically demand near-instantaneous response times [44]. The total system power consumption at this frequency, inclusive of the IMU sensor, was measured at 275 mW. When the ESP32-S3 operating frequency was reduced to 160 MHz, the total processing time for CNN-S1 increased to $6,000,015\,\mu s + 95\,\mu s + 1,025\,\mu s + 2,705,335\,\mu s = 8,706,470\,\mu s$ (approximately 8.71 seconds). In exchange for this increased latency, the power consumption decreased significantly to 195 mW. This duration remains well within practical limits for the target applications.

The total SRAM utilization for the CNN-S1 deployment is 92.22 KB. This comprises several components: the TFLM model requires approximately 62 KB; the input buffer for 300 timestamps (2700 float values) accounts for 2700×4 Bytes = 10.8 KB; the IMU sensor's internal register buffer (for raw 16-bit integer data) consumes about 288 Bytes; temporary storage for IMU feature conversion (scaled accelerometer, gyroscope, and magnetometer values) uses around 12 Bytes; and constants for IMU feature normalization (mean and standard deviation for nine axes) require $9 \times 2 \times 4$ Bytes = 72 Bytes. The remaining SRAM is allocated to FreeRTOS, system libraries (e.g., serial communication, WitMotion library), logging functions, and other temporary variables. The total flash memory footprint for this configuration is 545.749 KB, of which the CNN-S1 model itself occupies 209.108 KB. The remainder is used by the real time operating system, device drivers, libraries, and other system components.

The deployment of the CNN-S2 model, engineered for lower latency, demonstrated faster processing times as anticipated. At 240 MHz, the total processing time was $6,000,008\,\mu s + 77\,\mu s + 558\,\mu s + 642,313\,\mu s = 6,642,956\,\mu s$ (approximately 6.64 seconds). This configuration resulted in a power consumption of 295 mW. At 160 MHz, the CNN-S2 model processed a data window in $6,000,012\,\mu s + 93\,\mu s + 1,028\,\mu s + 959,294\,\mu s = 6,960,427\,\mu s$ (approximately 6.96 seconds), with a corresponding power draw of 205 mW. Compared to CNN-S1, CNN-S2 offers a substantial reduction in inference latency and overall processing time, albeit with a slight increase in power consumption at both frequencies and higher SRAM usage.

The total SRAM consumption for the CNN-S2 deployment is 112.26 KB. The breakdown includes approximately 82 KB for the model's TFLM arena, with the input buffer (10.8 KB), IMU raw data buffer (288 Bytes), feature conversion temporaries (12 Bytes), and normalization constants (72 Bytes) remaining consistent with the CNN-S1 setup. The increased total SRAM is primarily due to the larger arena size required by the CNN-S2 model. The rest of the memory is utilized by the system software and libraries. The flash memory usage for this model is 469.573 KB, with the CNN-S2 model binary contributing 127.984 KB.

The empirical results demonstrate that both the CNN-S1 and CNN-S2 models, when deployed on the ESP32-S3, present viable options for real-world applications. The choice between them, and the selection of operating frequency, will inevitably depend on the specific constraints and priorities of the target use case, such as the relative importance of predictive accuracy, inference speed, and power efficiency. While 240 MHz offers faster processing, the 160 MHz frequency generally provides a more favorable balance between performance and power consumption for both models, significantly reducing energy demand with a manageable increase in latency. These findings underscore the practical feasibility of deploying sophisticated deep learning models like CNN-S1 and CNN-S2 on resource-constrained embedded systems. Notably, this is achieved while maintaining parameter efficiency that is orders of magnitude superior to the original, more complex teacher DualCrossTAP model. As indicated in Table 7.1, even with the overhead of data acquisition and preprocessing, the system achieves inference latencies practical for many human activity recognition scenarios. Furthermore, the memory footprints indicate that sufficient resources remain on the ESP32-S3 for other concurrent tasks or more complex application logic.

7.5 Discussion on Battery Powered Scenarios

Considering battery-powered applications, particularly those aiming for extended operational longevity using compact power sources such as coin cells (e.g., CR2032 [119], often operating around 3.0–3.3 V with capacities in the range of 220–240 mAh [129]), the measured active power consumption figures (195 mW to 295 mW) would indeed lead to rapid battery depletion if the system were maintained in a continuously active state. For instance, a sustained draw of 195 mW at 3.3 V supply (ESP32-S3 requires 3.3 V) translates to an approximate current of 59 mA. This level of current

would exhaust a 220 mAh coin cell in roughly $220\,\text{mAh}/59\,\text{mA} \approx 3.72$ hours. However, it is crucial to contextualize these power measurements as likely representing a conservative, near worst-case scenario for a field-deployed product. This assessment is based on several factors. Firstly, as detailed in Section 6.2, the presented power consumption values pertain to the entire ESP32-S3 Dev Kit C1 evaluation board. In a realistic product designed for power efficiency, a custom-designed Printed Circuit Board (PCB) would be developed. Such PCB would be optimized for the specific application, omitting superfluous components typically found on development kits. These include, but are not limited to, USB ports and their associated bridge circuitry (e.g., USB to UART converters), unutilized GPIO pins, JTAG debugging interfaces, and potentially less efficient power regulation components. The exclusion of such non-essential circuitry directly contributes to lower overall quiescent and active power consumption [150], as well as reduced heat dissipation and a smaller physical dimensions. Secondly, significant power savings could be realized through a more sophisticated execution pipeline and dynamic power management strategies. In the current prototype, the ESP32-S3 was operated at a constant frequency (either 240 MHz or 160 MHz) throughout the entire process. A more power-aware approach could involve manual frequency setting, where the microcontroller operates at a substantially lower frequency (e.g., 20 MHz) during less computationally intensive phases, such as the raw data collection from the IMU sensor. The frequency could then be intelligently ramped up to higher levels (e.g., 160 MHz or 240 MHz) only when necessary for demanding tasks like feature conversion, normalization, and model inference. Furthermore, during the data collection period, the ESP32-S3 could leverage more advanced low-power states, such as the light sleep mode [151]. In light sleep, the CPU is paused, but peripherals and memory can remain active, allowing data acquisition to continue with minimal CPU power draw, thereby further reducing the average energy consumption per cycle. The implementation of these advanced power optimization techniques was beyond the primary scope of this study, which centered on establishing a complete end-to-end functional prototype for the HAR system. Nevertheless, this discussion is included with the intention of providing valuable perspectives for practitioners and engineers focused on developing production-grade, power-efficient embedded systems for composite human activity recognition. The principles outlined highlight feasible pathways to substantially extend battery life in practical deployments.

Chapter 8

Comparison with the State-of-the-Art

This chapter presents a comprehensive comparative analysis, benchmarking the performance of our designed and optimized models against prominent state-of-the-art methodologies proposed in recent literature. A rigorous comparison is fundamental not only for contextualizing our contributions within the existing body of research but also for empirically validating the efficacy and efficiency of our proposed solution. The primary objective is to objectively demonstrate the competitive performance and superior resource optimization of our models, which are specifically tailored for low-power, low-cost, and computationally constrained IoT devices.

While Chapter 6 has already detailed the MCU-specific performance metrics of our models, this chapter broadens the scope to a wider comparative landscape against other scholarly works. This is crucial for establishing the generalizability and robustness of our approach beyond the specific constraints of our target hardware. To ensure a thorough and multifaceted evaluation, our comparative analysis is conducted across three distinct and challenging datasets: CAPPIMU, KU HAR, and the NTNU Children HAR dataset. The specific characteristics, data acquisition protocols, and preprocessing pipelines for these datasets have been extensively discussed in Chapter 4 and Section 6.4. Utilizing multiple datasets with varying complexities and subject demographics allows for a more robust validation of our models' capabilities.

The remainder of this chapter is structured as follows. Section 8.1 initiates the analysis by evaluating our models on the CAPPIMU dataset. This section will also draw comparisons with recently proposed models from the literature that were not explicitly trained on this dataset, thereby underscoring the adaptability of our methods. For direct performance comparisons with previous studies that have utilized the CAPPIMU dataset, the reader is directed to Table 6.2 and Table 6.12. Subsequently, Section 8.2 and Section 8.3 extend this comparative analysis to the KU HAR and NTNU Children HAR datasets, respectively.

This structured, multi-dataset validation serves to rigorously test and confirm the robustness, accuracy, and computational efficiency of our proposed models against a diverse and unique set of activity recognition challenges and state-of-the-art solu-

tions.

8.1 Comparative Analysis on CAPPIMU Dataset

This section initiates our comparative analysis by benchmarking our models against a selection of recently proposed, state-of-the-art architectures from the HAR literature. The CAPPIMU dataset serves as the common ground for this evaluation. A detailed summary of this comparative study is presented in Table 8.1. The primary goal of this analysis is to rigorously evaluate our models not only in terms of predictive accuracy, as established in previous chapters, but also in terms of computational and architectural complexity. Key metrics for this comparison include the total number of parameters, the count of trainable parameters, and the computational load as measured in MFLOPs.

To ensure a fair and direct comparison, a significant challenge had to be addressed: the original models from the literature were designed for and trained on different datasets, with varying input dimensions and classification tasks. To overcome this, we meticulously reimplemented these external models within our experimental framework. It is crucial to note that no modifications were made to the intrinsic architectures of these state-of-the-art models. The only adjustments were to the input layer to match the CAPPIMU dataset's sensor data shape and to the final dense layer's output neurons to align with the 21 activity classes of our study. All consequent architectural adjustments were minimal and necessary to accommodate these input/output changes. A comprehensive discussion of this reimplementation process, including the specific model architectures and metric calculation methodologies, is provided in Appendix A.

A key observation from the data presented in Table 8.1 is the superior balance our models strike between complexity and potential performance. While our teacher model, DualCrossTAP, exhibits a considerable number of parameters, our distilled student models, CNN-S1 and CNN-S2, are significantly more lightweight than all the compared state-of-the-art models. For instance, CNN-S1 and CNN-S2 are orders of magnitude less complex than models like HARCNN, ResLSTM, and HT-AggNet in terms of parameter count and MFLOPs. This highlights the success of our knowledge distillation strategy in creating highly efficient models that are suitable for deployment on resource-constrained hardware such as the ESP32-S3.

In this context, we also introduce a specially engineered MLP model. The rationale behind its design, detailed in Appendix G, was to create an ultra-lightweight model with minimal inference latency. Such a model is particularly valuable for target MCUs that lack specialized hardware like a FPU or DSP extensions, where simpler arithmetic operations are paramount for performance. While the MLP model achieves an exceptionally low computational footprint of approximately 0.25 MFLOPs and is amenable to lossless quantization, its predictive accuracy is significantly lower than that of CNN-S1 and CNN-S2, as shown in Table G.7. This trade-off between extreme computational efficiency and reduced accuracy led to a strategic decision. Given that the primary focus of this thesis is on achieving high accuracy while maintaining substantial efficiency, we concluded that the MLP ar-

chitecture, despite its merits in specific contexts, was not the optimal candidate for the broader HAR challenges presented by the other datasets. Consequently, the MLP model was excluded from subsequent experiments involving the KU HAR and NTNU Children HAR datasets.

Table 8.1: Comprehensive comparison between our optimized models and recent state-of-the-art approaches reported in the literature on CAPPIMU dataset. The evaluation focuses on model complexity through key metrics such as the total number of parameters (referred as *Total Params*), trainable parameters (referred as *Trainable Params*), and MFLOPs, while the *Utilized PP* column indicates whether a PP sensor was incorporated. Due to the differences in datasets used across studies, a fair comparison was achieved by reimplementing the external models with CAPPIMU dataset's input dimensions and output configuration (i.e., number of classification classes). Importantly, no changes were made to the intrinsic model architectures; only the input shape, final Dense layer output neurons, and any consequent adjustments were modified. A detailed discussion of the reimplementation process and metric calculations is provided in Appendix A. Lastly, the MLP-related experimentation and in-depth analysis are provided as supplementary material in Appendix G.

Study	Model	Total Params	Trainable Params	MFLOPs	Utilized PP
[170]	HARCNN	219,763	219,763	9.67	✗
[108]	ResLSTM	856,107	853,227	107.02	✗
[189]	HT-AggNet	2,550,293	2,550,293	17.09	✗
[128]	Static Context CNN	19,666,197	19,666,197	53.07	✗
	Dynamic Context RNN	27,285	27,285	10.62	✗
	Dynamic Context LSTM	12,771	12,771	7.35	✗
This Study	**DualCross TAP**	**690,711**	**688,279**	**67.90**	✓
	CNN-S1	**51,029**	**51,029**	**6.14**	✓
	CNN-S2	**30,288**	**30,288**	**2.19**	✓
	MLP	**126,357**	**126,357**	**0.25**	✗

8.2 Comparative Analysis on KU HAR Dataset

To further assess the generalizability and robustness of our proposed models, this section extends the comparative analysis to the KU HAR dataset. This dataset is widely recognized in the literature and presents a different set of challenges compared

to CAPPIMU, including a distinct class distribution and activity set. By benchmarking against state-of-the-art models that have previously utilized this dataset, we can gain deeper insights into the performance of our models under varied conditions.

Table 8.2 provides an exhaustive comparison between our optimized models and several prominent, recently published methods. The evaluation encompasses model complexity, represented by the total number of parameters, the windowing strategy employed, the model's suitability for low-compute environments, and the overall F1 Score. It is important to note that for many of the referenced studies, specific architectural details such as the total parameter count were not available in the original publications.

Table 8.2: Exhaustive comparison of our optimized models and recent state-of-the-art approaches on the KU HAR dataset. Cells containing "–" denote metrics that were not reported in the original publications or code bases. The *Low Compute* column indicates whether each model can be deployed on resource-constrained microcontrollers such as the ESP32-S3 or similar MCUs (for similar MCUs, see Table 5.1). Lastly, the column *W/O* represents the comparison between the utilized window and overlap size in seconds.

Study	**Model**	**Total Params**	**W/O**	**Low Compute**	**F1 Score**
[144]	WISNet	-	3/0	✗	94.00%
[172]	CFEB	-		✗	97.50%
[186]	Integrated (combined)	-		✗	98.90%
[170]	HARCNN	211,138		✗	98.34%
[108]	ResBiLSTM	569,974		✗	98.72%
[189]	HT-AggNet (L=16)	2,550,034		✗	97.00%
[128]	DCM (Recurrent)	-		✗	99.62%
[156]	HCNN-LSTM	247,930		✗	84.04%
This Study	**DualCross TAP**	**687,508**	**3/1.5**	✗	**98.38%**
	CNN-S1	**50,066**		✓	**93.39%**
	CNN-S2	**29,691**		✓	**90.10%**

A primary observation from the comparative results is that our models exhibit highly competitive performance within this challenging landscape. Our teacher model, DualCrossTAP, achieves an F1 Score of 98.38%, placing it among the top-performing models. It does, however, fall slightly behind the DCM (Recurrent) model, which reports an F1 Score of 99.62%. This minor performance gap can be attributed to

the inherent class imbalance within the KU HAR dataset. As discussed in detail in Section 5.1, our model's architecture, while robust, can be sensitive to such imbalances, which can affect the learning process for minority classes.

However, the most critical insight from this analysis emerges when considering the practical deployability on resource-constrained devices, which is the core objective of this research. As indicated in the *Low Compute* column of Table 8.2, none of the other state-of-the-art models listed, despite their high accuracy, are designed for or capable of being deployed on MCU-level hardware like the ESP32-S3. These models, including the top-performing DCM, are architecturally complex and computationally demanding, requiring the resources of a server or a high-end edge device for inference.

In stark contrast, our student models, CNN-S1 and CNN-S2, stand out as being uniquely advantaged for real-world applications. They are the only models in this comparison that achieve a practical balance between high performance and the stringent constraints of low-power, low-compute hardware. With F1 Scores of 93.39% and 90.10% respectively, they deliver robust and reliable activity recognition while maintaining a minimal memory and computational footprint, as evidenced by their low parameter counts of 50,066 and 29,691. This demonstrates the profound success of our knowledge distillation approach: we have created models that are not just theoretically accurate but are practically deployable at the very edge of the IoT ecosystem.

Finally, while other models may achieve incrementally higher accuracy in a high-resource computational environment, they are fundamentally unsuitable for the target applications of this thesis. Our models, particularly CNN-S1 and CNN-S2, represent a superior solution by providing an exceptional and validated trade-off between predictive accuracy and on-device computational efficiency, a critical requirement for the next generation of wearable and IoT systems.

8.3 Comparative Analysis on NTNU Children HAR Dataset

This section culminates our multi-dataset comparative analysis by evaluating our models on the NTNU Children HAR dataset. This dataset is of particular significance as it introduces a unique and challenging demographic: children, including those with CP. The inclusion of subjects with varied motor abilities makes this dataset an exceptionally robust benchmark for assessing the real-world applicability and generalizability of HAR models, especially for applications in healthcare and assisted living. Successfully performing on this dataset demonstrates a model's ability to handle the irregular and diverse movement patterns inherent in child activity.

Our comparative analysis, summarized in Table 8.3, benchmarks our models against the original study that introduced the dataset. A critical aspect of this comparison is the difference in validation and testing methodologies. The original study employed a rigorous Leave-One-Subject-Out Cross-Validation (LOSOCV) strategy and evalu-

ated their XGBoost model on three distinct subject groups separately: TD children, children with CP performing standardized activities (CP Stan), and children with CP during free play (CP Free). In contrast, our approach utilizes a SD validation and, more importantly, evaluates our models on a single, combined test set that includes all three of these challenging groups. This holistic testing strategy directly assesses the model's capability to generalize across a spectrum of physical abilities within a unified framework, a scenario that is highly representative of real-world deployment.

Table 8.3: Thorough comparison of our optimized models and recent state-of-the-art approaches on the NTNU Children HAR dataset. The *Low Compute* column indicates whether each model can be deployed on resource-constrained microcontrollers such as the ESP32-S3 or similar MCUs (for similar MCUs, see Table 5.1). The column *Validation* represents the employed validation strategy by the corresponding study. In this context, the compared two studies employed *Leave-One-Subject-Out Cross-Validation (LOSOCV)* and *Subject-Dependent (SD)* denoted by its respective acronyms. Next, the column *W/O* represents the comparison between the utilized window and overlap size in seconds. Finally, the superscripts (ranges from 1–4) in the *Model* column denote the employed test groups.

Study	**Model**	**Validation**	**W/O**	**Low Compute**	**Accuracy**
[158]	XGBoost[1]	LOSOCV	1/0	✗	94.00%
	XGBoost[1]		1/0.5	✗	95.00%
	XGBoost[1]		3/0	✗	93.00%
	XGBoost[1]		3/1.5	✗	93.00%
	XGBoost[2]		1/0	✗	94.00%
	XGBoost[2]		1/0.5	✗	94.00%
	XGBoost[2]		3/0	✗	94.00%
	XGBoost[2]		3/1.5	✗	94.00%
	XGBoost[3]		1/0	✗	78.00%
	XGBoost[3]		1/0.5	✗	79.00%
	XGBoost[3]		3/0	✗	77.00%
	XGBoost[3]		3/1.5	✗	78.00%
This Study	**Dual CrossTAP[4]**	**SD**	**1/0.5**	✗	**95.00%**
	CNN-S1[4]			✓	**89.87%**
	CNN-S2[4]			✓	**91.08%**

[1] Tested on TD = Typically Developing children group.
[2] Tested on CP Stan = Cerebral Palsy with standardised activities group.
[3] Tested on CP Free = Cerebral Palsy with Free play group.
[4] Tested on combining all test groups: TD, CP Stan, and CP Free.

The results of this comparison are revealing. Our teacher model, DualCrossTAP, achieves an accuracy of 95.00%. This performance is on par with the highest accuracy reported in the original study, which was also 95.00% using an XGBoost model. However, a crucial distinction must be made: the state-of-the-art XGBoost model achieved this peak performance only on the easiest subgroup, the TD children, with a specific windowing configuration. Its performance saw a substantial decline when evaluated on the CP Free group, dropping to as low as 77.00%. In contrast, our DualCrossTAP model maintains its high 95.00% accuracy across the far more challenging combined group, demonstrating its superior robustness and stability in handling diverse and complex motion data.

The performance of our student models further solidifies the advantages of our approach. The distilled models, CNN-S1 and CNN-S2, achieve impressive accuracies of 89.87% and 91.08%, respectively. It is noteworthy that CNN-S2, the more compact of the two, slightly outperforms CNN-S1, suggesting that its streamlined architecture may offer better generalization on this particularly varied dataset. While these accuracies are below that of our teacher model, they are remarkably high and are significantly more impressive when considering the context of on-device deployment.

This brings us to the core contribution of our research, which is starkly highlighted in the *Low Compute* column of the table. The high-performing XGBoost models from the reference study, being a form of gradient-boosted decision trees, are computationally intensive and not designed for deployment on resource-constrained microcontrollers. In contrast, our CNN-S1 and CNN-S2 models are the only ones in this comparison that are explicitly designed and validated for such low-power environments. They deliver robust, high-accuracy performance on a challenging clinical dataset while retaining the efficiency required for practical, wearable IoT applications.

In conclusion, this final comparative analysis demonstrates that our proposed framework not only competes with state-of-the-art in terms of robustness across diverse user groups but also exceeds the performance of the state-of-the-art. More importantly, it provides a viable pathway to deploy these advanced capabilities on low-cost, low-power hardware, paving the way for accessible and scalable HAR solutions in demanding real-world settings such as pediatric health monitoring.

8.4 Comparative Analysis on Our Developed End-to-End Prototype HAR System

The preceding sections have rigorously benchmarked our models against the state of the art using established, public datasets. While such comparisons are essential for validating algorithmic performance, the ultimate test of a system designed for the IoT edge is its practical viability in a complete, end-to-end implementation. This section, therefore, provides a comparative analysis of our fully developed hardware and software prototype against another recently proposed end-to-end HAR system from the literature that also targets low-compute environments. This analysis shifts the focus from isolated model metrics to a holistic evaluation of a real-world system,

encompassing hardware, data processing, and power consumption.

Table 8.4: Comprehensive performance comparison of our optimized end-to-end composite HAR prototype system against existing low-compute state-of-the-art end-to-end HAR systems. In the table, the *Status* column presents the completeness of the entire system, including hardware and software configuration. Next, *Compute* column represents the respective chip (module) utilized by the respective study. In this context, both the chips (ESP32 and ESP32-S3) were running at constant 240 MHz CPU frequency. Lastly, the column *Power Draw* shows the total power drawn by respective end-to-end HAR systems.

Study	**Model**	**Dataset**	**Status**	**Compute**	**Sensor**	**Total Latency**	**Power Draw**
[190]	DT	HATPT	Partial	ESP32[1]	ACC[a]	2.01123 s	142 mW[*]
	RF	UMAFall				3.05052 s	142 mW[*]
This Study	CNN-S1	CAPPIMU	Complete	ESP32-S3	IMU	7.84557 s	275 mW
	CNN-S2	CAPPIMU				6.64295 s	295 mW

[1] Utilized chip for this MCU (ESP32 AI Thinker) was Tensilica Xtensa LX6.
[a] *ACC* represents Accelerometer Sensor.
[*] The reported power draw does not include the power consumed by the accelerometer since it was simulated by Rafee et al. [190].

Table 8.4 presents a comprehensive performance comparison between our prototype and the system proposed by Rafee et al. Evaluating end-to-end systems presents a unique set of challenges, as direct comparisons are often hindered by differences in hardware, datasets, and the very definition of system completeness. The chosen benchmark study represents one of the few recent efforts to document an end-to-end HAR pipeline on similar microcontroller-class hardware.

A foundational distinction, highlighted in the *Status* column of the table, is that our prototype is a *Complete* system. As detailed in Chapter 7, our implementation integrates a physical WitMotion IMU sensor with the ESP32-S3, managing the entire workflow on-device: from raw data acquisition and preprocessing to final model inference. In contrast, the system by Rafee et al. is described as *Partial*, as the sensor data input was simulated, meaning it did not contend with the real-world challenges and overhead of interfacing with and processing data from a live physical sensor. This makes our prototype a more rigorous and authentic demonstration of a deployable HAR system.

At first glance, the reported total latency of our system (6.6 to 7.8 seconds) appears significantly higher than the benchmark (2 to 3 seconds). However, a breakdown of this metric is crucial for a fair interpretation. The detailed analysis in Table 7.1 reveals that approximately 6 seconds of our total latency is dedicated to the *Raw Data Collection Time* (RDCT). This duration is a function of the required window size for our models and is not indicative of computational inefficiency. The actual on-device processing time, which includes feature conversion, normalization, and inference latency, is substantially lower (approximately 0.64 seconds) for our highly

efficient CNN-S2 model. This demonstrates exceptional processing speed, especially considering our system handles complex 9-axis IMU data on the challenging CAP-PIMU dataset, whereas the benchmark models were simpler (DT and RF) and used only accelerometer data. Furthermore, these simpler DT and RF were trained on UCI HAPT and UMAFall Detection datasets where the total ADL counts are 12 and 15 respectively. Compared to that, our system is much more robust since it is capable of detecting a total of 21 ADL.

A similar nuanced analysis is required for power consumption. Our system's measured power draw of 275 mW to 295 mW is higher than the 142 mW reported for the benchmark. However, a critical point is to notice that their reported power draw explicitly excludes the consumption of the accelerometer sensor itself, as it was simulated. Our measurement, conversely, represents the total power draw of the entire, fully operational system, including the ESP32-S3 development board and the connected IMU sensor under load. The benchmark's figure is therefore an incomplete and optimistic estimation of real-world power consumption. While it is true that their system's power consumption even when the accelerometer is included is going to be lower than ours due to the utilization of tree-based ML models. However, as already established in Chapter 7, in terms of completeness, our system outdid all the recent studies and hence is a practical real world solution.

While surface-level metrics might suggest a mixed comparison, a deeper, context-aware analysis confirms the superior design and validation of our end-to-end prototype. We have successfully built and evaluated a complete, on-device HAR system that handles a more complex sensor suite and a more challenging dataset. It demonstrates highly efficient data processing and provides a transparent, all-inclusive measure of its power requirements. This work, therefore, represents a more robust and realistic blueprint for the development of practical, low-cost, complex and composite HAR systems, offering a more reliable benchmark for future research and engineering endeavors in this domain.

Chapter 9

Limitation, Future Work and Conclusion

Despite the promising results achieved with the proposed KD-based sensor fusion strategy, the developed student models (CNN-S1 and CNN-S2), and the end-to-end prototype, this research is in its foundational stages and presents numerous avenues for further investigation and enhancement.

9.1 Limitations and Future Works

A primary limitation of this study is its reliance on a single dataset, the CAPPIMU dataset. This was necessitated by the fact that CAPPIMU is currently the only publicly accessible dataset that incorporates synchronized (i.e., sensor data is temporally aligned based on its timestamp to preserve spatial and temporal dependencies [44]) data from both PP and IMU sensors. Future research should prioritize the development and public dissemination of additional datasets featuring PP sensor data. Such efforts would not only provide a richer empirical basis for understanding plantar pressure dynamics in human activity recognition but also address the current under-exploration of PP sensor modality within this domain. Furthermore, it is highly recommended that new datasets include raw sensor data in its original integer format alongside the pre-processed data converted into physical quantities. This approach offers distinct advantages for deployment on resource-constrained devices: it obviates the need for the feature conversion step (column *FCT* in Table 7.1), thereby conserving CPU cycles (as evidenced by metrics in Table 7.1). Moreover, working directly with integer data could potentially reduce the necessity for explicit normalization if the raw data from different sensors or axes inherently share comparable scales. Crucially, it would also streamline the application of quantization-aware training [75], [86], facilitating the development of fully integer-based models that can operate efficiently without dedicated floating-point units, leading to potential cost and power savings in hardware.

Another area for expansion pertains to a more exhaustive exploration of PP sensor data. While this study utilized PP data and investigated IMU-to-PP sensor fusion, the full potential of PP sensors, either as a standalone modality or in more diverse fusion configurations, remains largely untapped due to its nascent status in the HAR field. Building upon the sensor fusion strategies explored, future work could

investigate an even more aggressive knowledge distillation approach. Specifically, a teacher model could be trained using comprehensive data from PP and IMU with the objective of distilling this multi-faceted knowledge into a highly compact student model that relies solely on accelerometer data. Such a strategy could lead to exceptionally resource-efficient student models without entirely sacrificing the rich contextual information provided by the other sensors.

The generalizability of the proposed teacher model, DualCrossTAP, also warrants further investigation. Its validation was confined to the CAPPIMU dataset. Therefore, future studies should endeavor to evaluate the performance of DualCrossTAP and the associated distillation methodologies on other benchmark HAR datasets, particularly those involving multi-sensor fusion, to ascertain its robustness and broader applicability. To this end, we suggest the utilization of benchmark datasets such as the UCI HAPT [27] and the UMAFall Detection [39] datasets, as they provide synchronized data from sensors, including accelerometers and gyroscopes, enabling a comprehensive assessment of the model's fusion capabilities. Additionally, for the student models CNN-S1 and CNN-S2, we examined various post-training quantization methods for optimization. However, we did not explore quantization-aware training or pruning [114], which could have significantly optimized the parameters of CNN-S1. Future research should address these aspects or build upon them to further enhance model efficiency and performance.

In terms of model architecture, while this work extensively explored lightweight CNN designs, other efficient architectures, such as those based on dense blocks or alternative lightweight paradigms, were not investigated. An intriguing direction for future research is the application of knowledge distillation from neural networks to simpler, yet highly efficient, tree-based models, such as decision trees or random forest [32]. Lightweight tree-based models are recognized for their exceptional inference speed and minimal resource consumption [69], [142], [190]. Distilling complex neural network knowledge into such models could further drastically reduce computational demands, enabling deployment on even more severely resource-constrained MCUs.

Finally, an analysis of the per-layer and per-operation execution times for CNN-S1 and CNN-S2 (detailed in Appendix D and Appendix E) consistently reveals that the *CONV_2D* operation constitutes the most significant computational bottleneck during inference across all tested ESP32-S3 frequencies. Consequently, a substantial improvement in inference latency could be achieved by designing and integrating a low-cost, low-compute hardware NN accelerator specifically for *CONV_2D* operations [107], [123]. While developing custom hardware accelerators is a challenging endeavor, frameworks and tools such as Apache TVM [121] and the CFU Playground [100] can aid in this process by facilitating the design, simulation, and deployment of such accelerators.

We aim to address these limitations and explore these promising research directions in our future work, with the overarching goal of further refining our sensor fusion and model compression techniques and expanding their practical applicability to a wider array of real-world embedded systems.

9.2 Conclusion

This research has successfully demonstrated a comprehensive approach to developing and deploying resource-efficient Human Activity Recognition systems by leveraging multi-modal data from Plantar Pressure and Inertial Measurement Unit sensors. Through the introduction of the DualCrossTAP teacher model and an effective knowledge distillation strategy, we developed compact yet capable student models (CNN-S1 and CNN-S2). The subsequent deployment and rigorous evaluation of these models on an end-to-end prototype confirmed their practical viability, achieving a crucial balance between predictive performance, inference latency, memory footprint, and power consumption suitable for resource-constrained embedded applications. This work underscores the potential of KD to enable sophisticated sensor fusion and intelligent decision-making on low-cost hardware.

The developed end-to-end prototype evaluation provided valuable insights into real-world operational trade-offs, particularly the quantifiable impact of microcontroller frequency on processing speed versus energy efficiency, and highlighted that while development kit power figures provide a baseline, significant optimizations are attainable through custom PCB design and advanced power management techniques. Our findings illustrate that practical, multi-modal composite HAR systems can be realized without demanding excessive computational resources, thereby broadening the accessibility of such technologies.

While this study marks significant progress, it also opens avenues for future enhancements. Key directions include the development of more diverse, raw-integer-based PP and IMU synchronized datasets, the exploration of even more aggressive distillation strategies, and the potential for hardware acceleration of critical computational operations. Addressing these areas will further propel the deployment of robust and efficient embedded HAR solutions across a wider spectrum of real-world scenarios, from personalized healthcare to pervasive smart environments.

References

[1] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998. DOI: 10.1142/S0218488598000094 [Online]. Available: https://doi.org/10.1142/S0218488598000094

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] F. Horiguchi, "Advanced memory concepts for dram and nonvolatile memories," in *Proceedings of 35th European Solid-State Device Research Conference, 2005. ESSDERC 2005.*, 2005, pp. 181–184. DOI: 10.1109/ESSDER.2005.1546615

[4] E. Schoonderwaldt, N. Rasamimanana, and F. Bevilacqua, "Combining accelerometer and video camera: Reconstruction of bow velocity profiles," Jan. 2006, pp. 200–203.

[5] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008. DOI: 10.1109/IEEESTD.2008.4610935

[6] J.-P. Vasseur and A. Dunkels, "Chapter 11 - smart object hardware and software," in *Interconnecting Smart Objects with IP*, J.-P. Vasseur and A. Dunkels, Eds., Boston: Morgan Kaufmann, 2010, pp. 119–145, ISBN: 978-0-12-375165-2. DOI: https://doi.org/10.1016/B978-0-12-375165-2.00011-9 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123751652000119

[7] T. Wilmshurst, "Chapter 1 - tiny computers, hidden control," in *Designing Embedded Systems with PIC Microcontrollers (Second Edition)*, T. Wilmshurst, Ed., Second Edition, Boston: Newnes, 2010, pp. 3–23, ISBN: 978-1-85617-750-4. DOI: https://doi.org/10.1016/B978-1-85617-750-4.10002-2 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781856177504100022

[8] T. Wilmshurst, "Section 1 getting started with embedded systems," in *Designing Embedded Systems with PIC Microcontrollers (Second Edition)*, T. Wilmshurst, Ed., Second Edition, Boston: Newnes, 2010, p. 1, ISBN: 978-1-85617-750-4. DOI: https://doi.org/10.1016/B978-1-85617-750-4.10001-0 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781856177504100010

[9] In *Advanced Industrial Control Technology*, P. Zhang, Ed., Oxford: William Andrew Publishing, 2010, pp. 811–842, ISBN: 978-1-4377-7807-6. DOI: https://doi.org/10.1016/B978-1-4377-7807-6.10024-5 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781437778076100245

[10] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] P. Barry and P. Crowley, "Chapter 5 - embedded processor architecture," in *Modern Embedded Computing*, P. Barry and P. Crowley, Eds., Boston: Morgan Kaufmann, 2012, pp. 99–152, ISBN: 978-0-12-391490-3. DOI: https://doi.org/10.1016/B978-0-12-391490-3.00005-9 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123914903000059

[12] O. Wong, H. Wong, W. Tam, and C. Kok, "A comparative study of charge pumping circuits for flash memory applications," *Microelectronics Reliability*, vol. 52, no. 4, pp. 670–687, 2012, Advances in non-volatile memory technology, ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2011.09.031 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271411004495

[13] G. Jain, "Chapter 4 - memory models for embedded multicore architecture," in *Real World Multicore Embedded Systems*, B. Moyer, Ed., Oxford: Newnes, 2013, pp. 75–116, ISBN: 978-0-12-416018-7. DOI: https://doi.org/10.1016/B978-0-12-416018-7.00004-3 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124160187000043

[14] M. A. Labrador and O. D. L. Yejas, *Human Activity Recognition: Using Wearable Sensors and Smartphones* (Chapman & Hall/CRC Computer and Information Science Series), 1st. Chapman and Hall/CRC, 2013. DOI: 10.1201/b16098 [Online]. Available: https://doi.org/10.1201/b16098

[15] R. Markell, "Chapter 38 - circuit collection, volume v: Data conversion, interface and signal conditioning products," in *Analog Circuit Design*, B. Dobkin and J. Williams, Eds., Oxford: Newnes, 2013, pp. 961–1081, ISBN: 978-0-12-397888-2. DOI: https://doi.org/10.1016/B978-0-12-397888-2.00038-9 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123978882000389

[16] N. B. Singh, D. K. Rai, and P. Singh, "Design of low power cmos psram," in *2013 International Conference on Advanced Electronic Systems (ICAES)*, 2013, pp. 122–126. DOI: 10.1109/ICAES.2013.6659374

[17] C. Park, M. Han, H. Lee, and S. W. Kim, "Performance comparison of gcc and llvm on the eisc processor," in *2014 International Conference on Electronics, Information and Communications (ICEIC)*, 2014, pp. 1–2. DOI: 10.1109/ELINFOCOM.2014.6914394

[18] G. Singh and M. Sachan, "Multi-layer perceptron (mlp) neural network technique for offline handwritten gurmukhi character recognition," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, 2014, pp. 1–5. DOI: 10.1109/ICCIC.2014.7238334

[19] M. Véstias and H. Neto, "Trends of cpu, gpu and fpga for high-performance computing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–6. DOI: 10.1109/FPL.2014.6927483

[20] M. Wolf, "Chapter 2 - cpus," in *High-Performance Embedded Computing (Second Edition)*, M. Wolf, Ed., Second Edition, Boston: Morgan Kaufmann, 2014, pp. 59–138, ISBN: 978-0-12-410511-9. DOI: https://doi.org/10.1016/B978-0-12-410511-9.00002-2 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124105119000022

[21] M. Abadi et al., *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/

[22] L. Ghica and N. Tapus, "Optimized retargetable compiler for embedded processors - gcc vs llvm," in *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2015, pp. 103–108. DOI: 10.1109/ICCP.2015.7312613

[23] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, 2015. arXiv: 1503.02531 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1503.02531

[24] S. Jamshed, "Chapter 3 - the way the hpc works in cfd," in *Using HPC for Computational Fluid Dynamics*, S. Jamshed, Ed., Oxford: Academic Press, 2015, pp. 41–79, ISBN: 978-0-12-801567-4. DOI: https://doi.org/10.1016/B978-0-12-801567-4.00003-9 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128015674000039

[25] B. V. Kasulaitis, C. W. Babbitt, R. Kahhat, E. Williams, and E. G. Ryen, "Evolving materials, attributes, and functionality in consumer electronics: Case study of laptop computers," *Resources, Conservation and Recycling*, vol. 100, pp. 1–10, 2015, ISSN: 0921-3449. DOI: https://doi.org/10.1016/j.resconrec.2015.03.014 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921344915000683

[26] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[27] J. Reyes-Ortiz, D. Anguita, and L. Oneto, *Smartphone-Based Recognition of Human Activities and Postural Transitions*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C54G7M, 2015.

[28] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794, ISBN: 9781450342322. DOI: 10.1145/2939672.2939785 [Online]. Available: https://doi.org/10.1145/2939672.2939785

[29] J. Yiu, *White paper: Cortex-m for beginners - an overview of the arm cortex-m processor family and comparison*, Accessed: 2024-01-25, 2016. [Online]. Available: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/white-paper-cortex-m-for-beginners-an-overview-of-the-arm-cortex-m-processor-family-and-comparison

[30] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, "Chapter 2 - high-performance embedded computing," in *Embedded Computing for High Performance*, J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, Eds., Boston: Morgan Kaufmann, 2017, pp. 17–56, ISBN: 978-0-12-804189-5. DOI: https://doi.org/10.1016/B978-0-12-804189-5.00002-8 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128041895000028

[31] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1597–1600. DOI: 10.1109/MWSCAS.2017.8053243

[32] N. Frosst and G. Hinton, *Distilling a neural network into a soft decision tree*, 2017. DOI: 10.48550/arXiv.1711.09784 arXiv: 1711.09784 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.09784

[33] A. G. Howard et al., *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017. arXiv: 1704.04861 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1704.04861

[34] G. Ke et al., "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157, ISBN: 9781510860964.

[35] M. Parker, "Chapter 29 - implementation with gpus," in *Digital Signal Processing 101 (Second Edition)*, M. Parker, Ed., Second Edition, Newnes, 2017, pp. 387–393, ISBN: 978-0-12-811453-7. DOI: https://doi.org/10.1016/B978-0-12-811453-7.00029-9 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128114537000299

[36] C. Sheppard, *Tree-based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting*. Amazon Digital Services LLC, 2017, ISBN: 9781975860974. [Online]. Available: https://www.amazon.com/Tree-based-Machine-Learning-Algorithms-Decision/dp/1975860977

[37] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[38] C. C. Aggarwal et al., *Neural networks and deep learning*. Springer, 2018, vol. 10.

[39] E. Casilari and J. A.Santoyo-Ramón, "UMAFall: Fall Detection Dataset (Universidad de Malaga)," Jun. 2018. DOI: 10.6084/m9.figshare.4214283.v7 [Online]. Available: https://figshare.com/articles/dataset/UMA_ADL_FALL_Dataset_zip/4214283

[40] A. Rana, A. Singh Rawat, A. Bijalwan, and H. Bahuguna, "Application of multi layer (perceptron) artificial neural network in the diagnosis system: A systematic review," in *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, 2018, pp. 1–6. DOI: 10.1109/RICE.2018.8509069

[41] B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, "Chapter 3 - modern architectures," in *Parallel Programming*, B. Schmidt, J. González-Domínguez, C. Hundt, and M. Schlarb, Eds., Morgan Kaufmann, 2018, pp. 47–75, ISBN: 978-0-12-849890-3. DOI: https://doi.org/10.1016/B978-0-12-849890-3.00003-4 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128498903000034

[42] S. Ahmadi, "Chapter 6 - internet of things (nb-iot and massive mtc)," in *5G NR*, S. Ahmadi, Ed., Academic Press, 2019, pp. 747–787, ISBN: 978-0-08-102267-2. DOI: https://doi.org/10.1016/B978-0-08-102267-2.00006-3 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780081022672000063

[43] H. Ahmed and A. K. Nandi, "Decision trees and random forests," in *Condition Monitoring with Vibration Signals: Compressive Sampling and Learning Algorithms for Rotating Machines*. 2019, pp. 199–224. DOI: 10.1002/9781119544678.ch10

[44] L. Chen and C. D. Nugent, "Composite activity recognition," in *Human Activity Recognition and Behaviour Analysis: For Cyber-Physical Systems in Smart Environments*. Cham: Springer International Publishing, 2019, pp. 151–181, ISBN: 978-3-030-19408-6. DOI: 10.1007/978-3-030-19408-6_7 [Online]. Available: https://doi.org/10.1007/978-3-030-19408-6_7

[45] A. Dehghani, O. Sarbishei, T. Glatard, and E. Shihab, "A quantitative comparison of overlapping and non-overlapping sliding windows for human activity recognition using inertial sensors," *Sensors*, vol. 19, no. 22, p. 5026, 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6891351/

[46] C. Kern, T. Klausch, and F. Kreuter, "Tree-based machine learning methods for survey research," in *Survey research methods*, vol. 13, 2019, p. 73.

[47] M. Madiajagan and S. S. Raj, "Chapter 1 - parallel computing, graphics processing unit (gpu) and new hardware for deep learning in computational intelligence research," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah, Ed., Academic Press, 2019, pp. 1–15, ISBN: 978-0-12-816718-2. DOI: https://doi.org/10.1016/B978-0-12-816718-2.00008-7 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128167182000087

[48] H. Phan et al., *Spatio-temporal attention pooling for audio scene classification*, 2019. arXiv: 1904.03543 [cs.SD]. [Online]. Available: https://arxiv.org/abs/1904.03543

[49] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *arXiv preprint arXiv:1912.05911*, 2019. [Online]. Available: https://arxiv.org/abs/1912.05911

[50] J. Singh and R. Banerjee, "A study on single and multi-layer perceptron neural network," in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, 2019, pp. 35–40. DOI: 10.1109/ICCMC.2019.8819775

[51] R. C. Staudemeyer and E. R. Morris, *Understanding lstm – a tutorial into long short-term memory recurrent neural networks*, 2019. arXiv: 1909.09586 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1909.09586

[52] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*, 2019, pp. 1–4. DOI: 10.1109/ICSCC.2019.8843652

[53] S. Yang, Y. Yuan, S. Wei, and Y. Ren, "An attention pooling based channel accumulation for target classification of harmonic radar," in *2019 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, 2019, pp. 56–60. DOI: 10.1109/ICTA48799.2019.9012910

[54] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, "A comparison of transformer and lstm encoder decoder models for asr," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2019, pp. 8–15. DOI: 10.1109/ASRU46091.2019.9004025

[55] C. W. Babbitt, H. Madaka, S. Althaf, B. Kasulaitis, and E. G. Ryen, "Disassembly-based bill of materials data for consumer electronic products," *Scientific Data*, vol. 7, no. 1, p. 251, Jul. 2020, ISSN: 2052-4463. DOI: 10.1038/s41597-020-0573-9 [Online]. Available: https://doi.org/10.1038/s41597-020-0573-9

[56] G. Campeanu, J. Carlson, and S. Sentilles, "Component-based development of embedded systems with gpus," *Journal of Systems and Software*, vol. 161, p. 110 488, 2020, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.110488 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219302626

[57] Y. Lu and J. Lu, "A universal approximation theorem of deep neural networks for expressing probability distributions," *Advances in neural information processing systems*, vol. 33, pp. 3094–3105, 2020.

[58] A. Rehmer and A. Kroll, "On the vanishing and exploding gradient problem in gated recurrent units," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1243–1248, 2020, 21st IFAC World Congress, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1342 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320317481

[59] H. Ali et al., "A survey on system level energy optimisation for mpsocs in iot and consumer electronics," *Computer Science Review*, vol. 41, p. 100 416, 2021, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2021.100416 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000563

[60] T. Contributors, *Dilated and causal convolutions on microcontrollers*, Accessed: 2024-10-19, 2021. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/48567

[61] Y. Gong, "Neuron perceptron model based on artificial neural network about decision-making behavior of macaque auditory cortex," in *Proceedings of the 2nd International Symposium on Artificial Intelligence for Medicine Sciences*, ser. ISAIMS '21, Beijing, China: Association for Computing Machinery, 2021, pp. 449–453, ISBN: 9781450395588. DOI: 10.1145/3500931.3501007 [Online]. Available: https://doi.org/10.1145/3500931.3501007

[62] P. Itterheimová, F. Foret, and P. Kubáň, "High-resolution arduino-based data acquisition devices for microscale separation systems," *Analytica Chimica Acta*, vol. 1153, p. 338 294, 2021, ISSN: 0003-2670. DOI: https://doi.org/10.1016/j.aca.2021.338294 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003267021001203

[63] K. J. Lee, "Chapter seven - architecture of neural processing unit for deep neural networks," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, ser. Advances in Computers, S. Kim and G. C. Deka, Eds., vol. 122, Elsevier, 2021, pp. 217–245. DOI: https://doi.org/10.1016/bs.adcom.2020.11.001 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300887

[64] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021, ISSN: 0360-0300. DOI: 10.1145/3469029 [Online]. Available: https://doi.org/10.1145/3469029

[65] K. R. Raghunathan, " History of Microcontrollers: First 50 Years," *IEEE Micro*, vol. 41, no. 06, pp. 97–104, Nov. 2021, ISSN: 1937-4143. DOI: 10.1109/MM.2021.3114754 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MM.2021.3114754

[66] N. Sikder and A.-A. Nahid, "Ku-har: An open dataset for heterogeneous human activity recognition," *Pattern Recognition Letters*, vol. 146, pp. 46–54, 2021, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2021.02.024 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865521000933

[67] T. Tan and G. Cao, *Deep learning on mobile devices through neural processing units and edge computing*, 2021. arXiv: 2112.02439 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2112.02439

[68] C. B. Vennerød, A. Kjærran, and E. S. Bugge, *Long short-term memory rnn*, 2021. arXiv: 2105.06756 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2105.06756

[69] C. Aytekin, *Neural networks are decision trees*, 2022. DOI: 10.48550/arXiv.2210.05189 arXiv: 2210.05189 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2210.05189

[70] F. Daghero, D. Jahier Pagliari, and M. Poncino, "Two-stage human activity recognition on microcontrollers with decision trees and cnns," *arXiv preprint arXiv:2206.07652*, 2022. [Online]. Available: https://arxiv.org/abs/2206.07652

[71] F. Daghero, D. J. Pagliari, and M. Poncino, "Two-stage human activity recognition on microcontrollers with decision trees and cnns," in *2022 17th Conference on Ph.D Research in Microelectronics and Electronics (PRIME)*, 2022, pp. 173–176. DOI: 10.1109/PRIME55000.2022.9816745

[72] F. Daghero et al., "Human activity recognition on microcontrollers with quantized and adaptive deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 4, Aug. 2022, ISSN: 1539-9087. DOI: 10.1145/3542819 [Online]. Available: https://doi.org/10.1145/3542819

[73] S. L. Harris and D. Harris, "5 - digital building blocks," in *Digital Design and Computer Architecture*, S. L. Harris and D. Harris, Eds., Morgan Kaufmann, 2022, pp. 236–297, ISBN: 978-0-12-820064-3. DOI: https://doi.org/10.1016/B978-0-12-820064-3.00005-2 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128200643000052

[74] "B - risc-v instruction set summary," in *Digital Design and Computer Architecture*, S. L. Harris and D. Harris, Eds., Morgan Kaufmann, 2022, p. 544, ISBN: 978-0-12-820064-3. DOI: https://doi.org/10.1016/B978-0-12-820064-3.00016-7 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128200643000167

[75] M. Kirtas et al., "Quantization-aware training for low precision photonic neural networks," *Neural Networks*, vol. 155, pp. 561–573, 2022, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2022.09.015 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608022003598

[76] S. Mekruksavanich and A. Jitpattanakul, "Fallnext: A deep residual model based on multi-branch aggregation for sensor-based fall detection," *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252418046

[77] S. Mekruksavanich and A. Jitpattanakul, "Fallnext: A deep residual model based on multi-branch aggregation for sensor-based fall detection," *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252418046

[78] H. P B, S. R. Anireddy, J. F T, and V. R, "Introduction to arm processors & its types and overview to cortex m series with deep explanation of each of the processors in this family," in *2022 International Conference on Computer Communication and Informatics (ICCCI)*, 2022, pp. 1–8. DOI: 10.1109/ICCCI54379.2022.9740768

[79] N. Passalis, M. Tzelepi, and A. Tefas, "Chapter 8 - knowledge distillation," in *Deep Learning for Robot Perception and Cognition*, A. Iosifidis and A. Tefas, Eds., Academic Press, 2022, pp. 165–186, ISBN: 978-0-323-85787-1. DOI: https://doi.org/10.1016/B978-0-32-385787-1.00013-0 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323857871000130

[80] N. Shrivastava and S. R. Sarangi, *Seculator: A fast and secure neural processing unit*, 2022. arXiv: 2204.08951 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2204.08951

[81] D. Soydaner, "Attention mechanism in neural networks: Where it comes and where it goes," *Neural Computing and Applications*, vol. 34, no. 16, pp. 13 371–13 385, Aug. 2022, ISSN: 1433-3058. DOI: 10.1007/s00521-022-07366-3 [Online]. Available: https://doi.org/10.1007/s00521-022-07366-3

[82] S. Ziyabari, L. Du, and S. Biswas, "Multi-branch resnet-transformer based deep hybrid approach for short-term spatio-temporal solar irradiance forecasting," in *2022 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2022, pp. 1–5. DOI: 10.1109/ECCE50734.2022.9947670

[83] W. Chen, Y. Li, Z. Tian, and F. Zhang, "2d and 3d object detection algorithms from images: A survey," *Array*, vol. 19, p. 100 305, 2023, ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2023.100305 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005623000309

[84] K. T. Chitty-Venkata, S. Mittal, M. Emani, V. Vishwanath, and A. K. Somani, "A survey of techniques for optimizing transformer inference," *Journal of Systems Architecture*, vol. 144, p. 102 990, 2023, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2023.102990 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762123001698

[85] P.-H. Chou, C. Wang, and C.-S. Mei, "Applicability of deep learning model trainings on embedded gpu devices: An empirical study," in *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, 2023, pp. 1–4. DOI: 10.1109/MECO58584.2023.10155048

[86] C. Contoli and E. Lattanzi, "A study on the application of tensorflow compression techniques to human activity recognition," *IEEE Access*, vol. 11, pp. 48 046–48 058, 2023. DOI: 10.1109/ACCESS.2023.3276438

[87] K. D. Cooper and L. Torczon, "Chapter 14 - runtime optimization," in *Engineering a Compiler (Third Edition)*, K. D. Cooper and L. Torczon, Eds., Third Edition, Philadelphia: Morgan Kaufmann, 2023, pp. 713–755, ISBN: 978-0-12-815412-0. DOI: https://doi.org/10.1016/B978-0-12-815412-0.00020-6 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128154120000206

[88] E. Cui, T. Li, and Q. Wei, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023. DOI: 10.1109/ACCESS.2023.3246491

[89] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100 857, Apr. 2023, ISSN: 2210-5379. DOI: 10.1016/j.suscom.2023.100857 [Online]. Available: http://dx.doi.org/10.1016/j.suscom.2023.100857

[90] A. P. Diéguez and M. A. López, "Performance tuning for gpu-embedded systems: Machine-learning-based and analytical model-driven tuning methodologies," in *2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2023, pp. 129–140. DOI: 10.1109/SBAC-PAD59825.2023.00022

[91] M.-G. Gan and Y. Zhang, "Temporal attention-pyramid pooling for temporal action detection," *IEEE Transactions on Multimedia*, vol. 25, pp. 3799–3810, 2023. DOI: 10.1109/TMM.2022.3166025

[92] D. Gonzalez-Arjona, V. Pesce, A. Colagrossi, and S. Silvestrini, "Chapter thirteen - on-board implementation," in *Modern Spacecraft Guidance, Navigation, and Control*, V. Pesce, A. Colagrossi, and S. Silvestrini, Eds., Elsevier, 2023, pp. 685–712, ISBN: 978-0-323-90916-7. DOI: https://doi.org/10.1016/B978-0-323-90916-7.00013-5 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323909167000135

[93] Z. Huang, M. Liang, J. Qin, S. Zhong, and L. Lin, " Understanding Self-attention Mechanism via Dynamical System Perspective," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2023, pp. 1412–1422. DOI: 10.1109/ICCV51070.2023.00136 [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV51070.2023.00136

[94] W. N. Ismail, H. A. Alsalamah, M. M. Hassan, and E. Mohamed, "Auto-har: An adaptive human activity recognition framework using an automated cnn architecture design," *Heliyon*, vol. 9, no. 2, e13636, 2023, ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2023.e13636 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844023008435

[95] V. Kamath and A. Renuka, "Deep learning based object detection for resource constrained devices: Systematic review, future trends and challenges ahead," *Neurocomputing*, vol. 531, pp. 34–60, 2023, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2023.02.006 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223001388

[96] K. Lehniger and P. Langendörfer, "Through the window: Exploitation and countermeasures of the esp32 register window overflow," *Future Internet*, vol. 15, no. 6, 2023, ISSN: 1999-5903. DOI: 10.3390/fi15060217 [Online]. Available: https://www.mdpi.com/1999-5903/15/6/217

[97] M. Mardanpour, M. Sepahvand, F. Abdali-Mohammadi, M. Nikouei, and H. Sarabi, "Human activity recognition based on multiple inertial sensors through feature-based knowledge distillation paradigm," *Information Sciences*, vol. 640, p. 119 073, 2023, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2023.119073 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025523006588

[98] T. Martin, "Chapter 6 - cortex-m7 processor," in *The Designer's Guide to the Cortex-M Processor Family (Third Edition)*, T. Martin, Ed., Third Edition, Newnes, 2023, pp. 203–230, ISBN: 978-0-323-85494-8. DOI: https://doi.org/10.1016/B978-0-323-85494-8.00007-3 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323854948000073

[99] T. Martin, "Chapter 9 - practical dsp for cortex-m microcontrollers," in *The Designer's Guide to the Cortex-M Processor Family (Third Edition)*, T. Martin, Ed., Third Edition, Newnes, 2023, pp. 313–353, ISBN: 978-0-323-85494-8. DOI: https://doi.org/10.1016/B978-0-323-85494-8.00010-3 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323854948000103

[100] S. Prakash et al., "Cfu playground: Full-stack open-source framework for tiny machine learning (tinyml) acceleration on fpgas," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 157–167. DOI: 10.1109/ISPASS57527.2023.00024

[101] J. C. Ruiz-Garcia, R. Tolosana, R. Vera-Rodriguez, and C. Moro, "Carefall: Automatic fall detection through wearable devices and ai methods," *arXiv preprint arXiv:2307.05275*, 2023. [Online]. Available: https://arxiv.org/abs/2307.05275

[102] STMicroelectronics, *STM32F334K4 datasheet - mainstream arm cortex-m4 mcu with fpu, 40 mhz cpu, 16 kbytes flash, 4 kbytes sram, peripherals*, https://www.st.com/resource/en/datasheet/stm32f334k4.pdf, Accessed January 4, 2025, 2023.

[103] E. Systems, *Esp32-s3-devkitc-1 user guide*, Accessed: 2024-09-29, 2023. [Online]. Available: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-devkitc-1/index.html

[104] A. Vaswani et al., *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762

[105] M. Wolf, "Chapter 3 - cpus," in *Computers as Components (Fifth Edition)*, ser. The Morgan Kaufmann Series in Computer Architecture and Design, M. Wolf, Ed., Fifth Edition, Morgan Kaufmann, 2023, pp. 97–159. DOI: https://doi.org/10.1016/B978-0-323-85128-2.00003-7 [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323851282000037

[106] C. Yang, X. Yu, Z. An, and Y. Xu, *Categories of response-based, feature-based, and relation-based knowledge distillation*, 2023. arXiv: 2306.10687 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2306.10687

[107] S. Ahmadifarsani, R. Stahl, P. van Kempen, D. Mueller-Gritschneder, and U. Schlichtmann, "Towards rapid exploration of heterogeneous tinyml systems using virtual platforms and tvm's uma," in *Proceedings of the 2023 Workshop on Compilers, Deployment, and Tooling for Edge AI*, ser. CODAI '23, Hamburg, Germany: Association for Computing Machinery, 2024, pp. 6–10, ISBN: 9798400703379. DOI: 10.1145/3615338.3618121 [Online]. Available: https://doi.org/10.1145/3615338.3618121

[108] S. AlMuhaideb, L. AlAbdulkarim, D. M. AlShahrani, H. AlDhubaib, and D. E. AlSadoun, "Achieving more with less: A lightweight deep learning solution for advanced human activity recognition (har)," *Sensors*, vol. 24, no. 16, 2024, ISSN: 1424-8220. DOI: 10.3390/s24165436 [Online]. Available: https://www.mdpi.com/1424-8220/24/16/5436

[109] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa, and V. Zizzo, "Edge computing and cloud computing for internet of things: A review," *Informatics*, vol. 11, no. 4, 2024, ISSN: 2227-9709. DOI: 10.3390/informatics11040071 [Online]. Available: https://www.mdpi.com/2227-9709/11/4/71

[110] Arduino, *Nano 33 BLE Sense Rev2 Hardware Documentation*, https://docs.arduino.cc/hardware/nano-33-ble-sense-rev2/, Accessed: 2024-09-10, 2024.

[111] Arduino, *Serial - arduino reference*, Accessed on 2024-06-09, 2024. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/communication/serial/

[112] *Article 95*, Accessed: 2024-02-07, 2024. [Online]. Available: https://www.semicone.com/article-95.html

[113] L. Cao et al., "Cost optimization in edge computing: A survey," *Artificial Intelligence Review*, vol. 57, no. 11, p. 312, Oct. 2024, ISSN: 1573-7462. DOI: 10.1007/s10462-024-10947-4 [Online]. Available: https://doi.org/10.1007/s10462-024-10947-4

[114] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10 558–10 578, 2024. DOI: 10.1109/TPAMI.2024.3447085

[115] S. Deng et al., "Lhar: Lightweight human activity recognition on knowledge distillation," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 11, pp. 6318–6328, 2024. DOI: 10.1109/JBHI.2023.3298932

[116] T. Developers, *Tensorflow lite converter api*, Accessed: 2024-09-21, 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter

[117] G. A. Edge, *Litert for microcontrollers overview*, https://ai.google.dev/edge/litert/microcontrollers/overview, Accessed April 4, 2024, Aug. 2024.

[118] A. Elhanashi, P. Dini, S. Saponara, and Q. Zheng, "Advancements in tinyml: Applications, limitations, and impact on iot devices," *Electronics*, vol. 13, no. 17, 2024, ISSN: 2079-9292. DOI: 10.3390/electronics13173562 [Online]. Available: https://www.mdpi.com/2079-9292/13/17/3562

[119] Energizer, *Energizer cr2032 datasheet*, Accessed: 2024-05-28, 2024. [Online]. Available: https://data.energizer.com/pdfs/cr2032.pdf

[120] M. Ficco, A. Guerriero, E. Milite, F. Palmieri, R. Pietrantuono, and S. Russo, "Federated learning for iot devices: Enhancing tinyml with on-board training," *Information Fusion*, vol. 104, p. 102 189, 2024, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2023.102189 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253523005055

[121] A. S. Foundation, *Apache tvm: Open deep learning compiler stack*, Accessed: 2024-12-28, 2024. [Online]. Available: https://github.com/apache/tvm

[122] P. H. N. Gonçalves, H. Bragança, and E. Souto, "Efficient human activity recognition on wearable devices using knowledge distillation techniques," *Electronics*, vol. 13, no. 18, 2024, ISSN: 2079-9292. DOI: 10.3390/electronics13183612 [Online]. Available: https://www.mdpi.com/2079-9292/13/18/3612

[123] Z. Huang, K. Zandberg, K. Schleiser, and E. Baccelli, "Riot-ml: Toolkit for over-the-air secure updates and performance evaluation of tinyml models," *Annals of Telecommunications*, 2024, ISSN: 1958-9395. DOI: 10.1007/s12243-024-01041-5 [Online]. Available: https://doi.org/10.1007/s12243-024-01041-5

[124] JetBrains, *Clion: A cross-platform ide for c and c++ by jetbrains*, Accessed on: August 16, 2024, 2024. [Online]. Available: https://www.jetbrains.com/clion/

[125] H. Kaur, V. Rani, and M. Kumar, "Human activity recognition: A comprehensive review," *Expert Systems*, vol. 41, no. 11, e13680, 2024. DOI: https://doi.org/10.1111/exsy.13680 eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.13680. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13680

[126] A. R. Khouas, M. R. Bouadjenek, H. Hacid, and S. Aryal, *Training machine learning models at the edge: A survey*, 2024. DOI: https://doi.org/10.48550/arXiv.2403.02619 arXiv: 2403.02619 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2403.02619

[127] C. Kormanyos, *Xxd: Hex-dump-type utility*, https://github.com/ckormanyos/xxd, Accessed: 2024-09-29, 2024.

[128] P. Kumar and S. Suresh, "Deep context model (dcm): Dual context-attention aware model for recognizing the heterogeneous human activities using smartphone sensors," *Evolving Systems*, vol. 15, no. 4, pp. 1475–1486, 2024. DOI: 10.1007/s12530-024-09570-z [Online]. Available: https://doi.org/10.1007/s12530-024-09570-z

[129] Leafony, *Leafony av01 cr2032 documentation*, Accessed: 2024-05-28, 2024. [Online]. Available: https://docs.leafony.com/en/docs/leaf/power/av01/

[130] C. S. Leite, H. Mauranen, A. Zhanabatyrova, and Y. Xiao, *Transformer-based approaches for sensor-based human activity recognition: Opportunities and challenges*, 2024. arXiv: 2410.13605 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2410.13605

[131] S. Li, X. Dong, D. Ma, B. Dang, H. Zang, and Y. Gong, "Utilizing the lightgbm algorithm for operator user credit assessment research," *arXiv preprint arXiv:2403.14483*, 2024. [Online]. Available: https://arxiv.org/abs/2403.14483

[132] B. Luo, Q. Qiu, T. Zhu, and Z. Liu, "Cappimu: A composite activities dataset for human activity recognition utilizing plantar pressure and imu sensors," in *Intelligent Information Processing XII*, Z. Shi, J. Torresen, and S. Yang, Eds., Cham: Springer Nature Switzerland, 2024, pp. 87–100, ISBN: 978-3-031-57919-6.

[133] C. Monico, "An elementary proof of a universal approximation theorem," *arXiv preprint arXiv:2406.10002*, 2024.

[134] P. Moreau, D. Durand, J. Bosche, and M. Lefranc, "Two-branch neural network using two data types for human activity recognition," *IEEE Sensors Journal*, vol. 24, no. 2, pp. 2216–2227, 2024. DOI: 10.1109/JSEN.2023.3332290

[135] A. Moslemi, A. Briskina, Z. Dang, and J. Li, "A survey on knowledge distillation: Recent advancements," *Machine Learning with Applications*, vol. 18, p. 100 605, 2024, ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2024.100605 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827024000811

[136] F. Pedregosa et al., *Standardscaler*, https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html, Accessed: 2024-10-04, 2024.

[137] PlatformIO, *Arduino Framework - PlatformIO Documentation*, https://docs.platformio.org/en/latest/frameworks/arduino.html, Accessed: 2024-05-10, 2024.

[138] PlatformIO, *Platformio: Professional collaborative platform for embedded development*, Accessed on: August 16, 2024, 2024. [Online]. Available: https://platformio.org/

[139] ptflops Contributors, *Ptflops: A python package for calculating flops in pytorch models*, Accessed on: October 31, 2024, 2024. [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch

[140] L. Qi, J. Fan, H. Cai, and Z. Fang, "A survey of emerging memory in a microcontroller unit," *Micromachines*, vol. 15, no. 4, 2024, ISSN: 2072-666X. DOI: 10.3390/mi15040488 [Online]. Available: https://www.mdpi.com/2072-666X/15/4/488

[141] L. Qi, J. Fan, H. Cai, and Z. Fang, "A survey of emerging memory in a microcontroller unit," *Micromachines*, vol. 15, no. 4, 2024, ISSN: 2072-666X. DOI: 10.3390/mi15040488 [Online]. Available: https://www.mdpi.com/2072-666X/15/4/488

[142] A. N. M. Rafee, A. Dutta, A. Haque, A. Rahman, and A. Barua, *Edge-optimized machine learning models for real-time personalized health monitoring on wearables*, J. Noor, Ed. Brac University, 2024. [Online]. Available: http://hdl.handle.net/10361/22912

[143] U. Saha, S. Saha, T. Kabir, S. A. Fattah, and M. Saquib, *Decoding human activities: Analyzing wearable accelerometer and gyroscope data for activity recognition*, 2024. arXiv: 2310.02011 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2310.02011

[144] H. Sharen, L. Jani Anbarasi, P. Rukmani, A. H. Gandomi, R. Neeraja, and M. Narendra, "Wisnet: A deep neural network based human activity recognition system," *Expert Systems with Applications*, vol. 258, p. 124 999, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.124999 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417424018669

[145] C. Silvano et al., *A survey on deep learning hardware accelerators for heterogeneous hpc platforms*, 2024. arXiv: 2306.15552 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2306.15552

[146] I. Sonata and Y. Heryadi, "Comparison of lstm and transformer for time series data forecasting," in *2024 7th International Conference on Informatics and Computational Sciences (ICICoS)*, 2024, pp. 491–495. DOI: 10.1109/ICICoS62600.2024.10636892

[147] M. Srinu, E. S. Rao, and P. C. Sekhar, "Design of low power sram cells with increased read and write performance using read - write assist technique," *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 7, p. 100 381, 2024, ISSN: 2772-6711. DOI: https://doi.org/10.1016/j.prime.2023.100381 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772671123002760

[148] A. Srivastav, S. K. Tripathi, U. Tiwari, and S. K. Mandal, "Comprehensive study of low-power sram design topologies," *Recent Advances in Electrical & Electronic Engineering*, vol. 17, no. 9, pp. 849–858, 2024, ISSN: 2352-0973. DOI: https://doi.org/10.2174/0123520965275861231027060817 [Online]. Available: https://www.benthamdirect.com/content/journals/raeeng/10.2174/0123520965275861231027060817

[149] STMicroelectronics, *STM32H747XI Microcontrollers Microprocessors*, https://www.st.com/en/microcontrollers-microprocessors/stm32h747xi.html, Accessed: 2024-05-10, 2024.

[150] E. Systems, *Current consumption measurement of modules*, Accessed: 03-May-2024, 2024. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/current-consumption-measurement-modules.html

[151] E. Systems, *Esp-idf sleep modes documentation*, Accessed: 2024-05-28, 2024. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/system/sleep_modes.html

[152] E. Systems, *Power management - esp32-s3 - esp-idf programming guide*, Accessed: 2024-03-12, 2024. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/system/power_management.html

[153] TensorFlow, *Adamw optimizer*, Accessed: 2024-08-12, 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/AdamW

[154] TensorFlow, *Tensorflow keras api documentation*, Accessed: 2024-10-03, 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/ops

[155] TensorFlow, *Tensorflow::ops namespace reference*, https://www.tensorflow.org/api_docs/cc/namespace/tensorflow/ops, Accessed April 4, 2024, 2024.

[156] D. J. Tilley and U. Martinez-Hernandez, "Hierarchical deep learning for human activity recognition integrating postural transitions," *IEEE Sensors Journal*, vol. 24, no. 24, pp. 40 305–40 312, 2024. DOI: 10.1109/JSEN.2024.3491352

[157] torchinfo Contributors, *View model summaries in pytorch!* Accessed on: October 31, 2024, 2024. [Online]. Available: https://github.com/TylerYep/torchinfo

[158] M. F. Tørring, A. Logacjov, S. M. Brændvik, A. Ustad, K. Roeleveld, and E. M. Bardal, "Validation of two novel human activity recognition models for typically developing children and children with Cerebral Palsy," *PLOS ONE*, vol. 19, no. 9, e0308853, Sep. 23, 2024, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0308853

[159] V. Tsoukas, A. Gkogkidis, E. Boumpa, and A. Kakarountas, "A review on the emerging technology of tinyml," *ACM Comput. Surv.*, vol. 56, no. 10, Jun. 2024, ISSN: 0360-0300. DOI: 10.1145/3661820 [Online]. Available: https://doi.org/10.1145/3661820

[160] R. E. Turner, *An introduction to transformers*, 2024. arXiv: 2304.10557 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2304.10557

[161] P. R. Verma, N. P. Singh, D. Pantola, and X. Cheng, "Neural network developments: A detailed survey from static to dynamic models," *Computers and Electrical Engineering*, vol. 120, p. 109 710, 2024, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2024.109710 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790624006372

[162] Wit-Motion, *Bwt901cl: 9-axis bluetooth gyroscope accelerometer sensor*, Accessed: 03-May-2024, 2024. [Online]. Available: https://www.wit-motion.com/BLE/50.html

[163] WITMOTION, *Witstandardprotocol jy901 - arduino sdk*, https://github.com/WITMOTION/WitStandardProtocol_JY901/tree/main/Arduino/Arduino_sdk, Accessed: 26-May-2024, 2024.

[164] L. WitMotion Shenzhen Co., *Fcc datasheet for bwt901cl*, Accessed: 26-May-2024, 2024. [Online]. Available: https://m.media-amazon.com/images/I/81q25mmcAyL.pdf

[165] W. G. Wong, *Nxp's i.mx rt700 crossover microcontroller incorporates the eiq neutron neural processing unit*, Accessed: 2025-02-07, 2024. [Online]. Available: https://www.electronicdesign.com/technologies/embedded/digital-ics/processors/microcontrollers/article/55142460/electronic-design-nxps-imx-rt700-crossover-microcontroller-incorporates-the-eiq-neutron-neural-processing-unit

[166] T. Zhang, *Network level spatial temporal traffic forecasting with hierarchical-attention-lstm*, 2024. DOI: 10.48130/dts-0024-0021 [Online]. Available: http://www.maxapress.com/article/doi/10.48130/dts-0024-0021

[167] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, no. 4, p. 99, 2024. DOI: 10.1007/s10462-024-10721-6 [Online]. Available: https://link.springer.com/article/10.1007/s10462-024-10721-6

[168] Y. Zhu, B. Luo, Q. Qiu, and T. Zhu, "Multimodal fusion contrastive learning framework based on insole and wristband," in *4th International Conference on Internet of Things and Smart City (IoTSC 2024)*, SPIE, vol. 13224, 2024, pp. 96–101. DOI: 10.1117/12.3034835

[169] M. A. O. Zishan, H. Shihab, S. S. Islam, M. A. Riya, G. M. Rahman, and J. Noor, "Dense neural network based arrhythmia classification on low-cost and low-compute micro-controller," *Expert Systems with Applications*, vol. 239, p. 122 560, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122560 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423030622

[170] E. Abdellatef, R. M. Al-Makhlasawy, and W. A. Shalaby, "Detection of human activities using multi-layer convolutional neural network," *Scientific Reports*, vol. 15, no. 1, p. 7004, 2025. DOI: https://doi.org/10.1038/s41598-025-90307-6 [Online]. Available: https://www.nature.com/articles/s41598-025-90307-6

[171] W. Elektronik, *We-xtal watch crystals design kit brochure*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.we-online.com/catalog/media/o166617v410%20BCF_WE-XTAL-WatchCrystals_DesignKit-Brochure_830001BR.pdf

[172] X. Guo, Y. Kim, X. Ning, and S. D. Min, "Enhancing the transformer model with a convolutional feature extractor block and vector-based relative position embedding for human activity recognition," *Sensors*, vol. 25, no. 2, 2025, ISSN: 1424-8220. DOI: 10.3390/s25020301 [Online]. Available: https://www.mdpi.com/1424-8220/25/2/301

[173] S. Heydari and Q. H. Mahmoud, "Tiny machine learning and on-device inference: A survey of applications, challenges, and future directions," *Sensors*, vol. 25, no. 10, 2025, ISSN: 1424-8220. DOI: 10.3390/s25103191 [Online]. Available: https://www.mdpi.com/1424-8220/25/10/3191

[174] Inmoresentum, *Esp32tflmwrapper*, https://registry.platformio.org/libraries/inmoresentum/ESP32TFLMWrapper, PlatformIO library for ESP32 TensorFlow Lite Micro wrapper, 2025.

[175] T. Instruments, *Msp430 microcontrollers overview*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.ti.com/microcontrollers-mcus-processors/msp430-microcontrollers/overview.html

[176] L. R. Jason Tollefson, *Low power ewc overview*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.mouser.com/applications/low-power-ewc-low-power/

[177] M. Kagiyama and T. Okita, *Knowledge distillation for reservoir-based classifier: Human activity recognition*, 2025. arXiv: 2505.22985 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2505.22985

[178] M. A.-A. Kaiser, S. Sunder, C. Mathew, M. Rakowski, A. P. Jacob, and A. R. Jaiswal, *Design of energy-efficient cross-coupled differential photonic-sram (psram) bitcell for high-speed on-chip photonic memory and compute systems*, 2025. arXiv: 2503.19544 [physics.optics]. [Online]. Available: https://arxiv.org/abs/2503.19544

[179] Keweisi, *Keweisi kws-mx18l digital display detector voltmeter*, Accessed: 2025-01-14, 2025. [Online]. Available: https://www.amazon.com/KWS-MX18L-Digital-Display-Detector-Voltmeter/dp/B0CGTG7JNQ

[180] M. T. R. Khan, E. Ever, S. Eraslan, and Y. Yesilada, "Human activity recognition using binary sensors: A systematic review," *Information Fusion*, vol. 115, p. 102731, 2025, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2024.102731 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253524005098

[181] S. Kobourov et al., "The influence of dimensions on the complexity of computing decision trees," *Artificial Intelligence*, vol. 343, p. 104322, 2025, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2025.104322 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370225000414

[182] H. Li, I. Anatoljevic Baryskievic, A. Antonovich Baryskievic, and V. Yurevich Tsviatkou, "A novel integration of bodily-kinesthetic intelligence (bki) and feature mining methodology: Applications in fall risk assessments," *IEEE Sensors Journal*, vol. 25, no. 5, pp. 8721–8736, 2025. DOI: 10.1109/JSEN.2025.3531296

[183] C. Mary M., J. H.S., and A. S., "Explainable optimal random forest model with conversational interface," *Engineering Applications of Artificial Intelligence*, vol. 145, p. 110 134, 2025, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2025.110134 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197625001344

[184] J. Millar, Y. Huang, S. Sethi, H. Haddadi, and A. Madhavapeddy, *Benchmarking ultra-low-power μnpus*, 2025. arXiv: 2503.22567 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2503.22567

[185] A. Muniasamy and F. Asiri, "Advanced human activity recognition on wearables with multi-scale sparse attention," *Multimedia Tools and Applications*, Jan. 2025, ISSN: 1573-7721. DOI: 10.1007/s11042-024-20475-6 [Online]. Available: https://doi.org/10.1007/s11042-024-20475-6

[186] A. Muniasamy and F. Asiri, "Advanced human activity recognition on wearables with multi-scale sparse attention," *Multimedia Tools and Applications*, Jan. 2025, ISSN: 1573-7721. DOI: 10.1007/s11042-024-20475-6 [Online]. Available: https://doi.org/10.1007/s11042-024-20475-6

[187] Nordic Semiconductor, *nRF5340 Advanced Bluetooth LE Audio SoC*, https://www.nordicsemi.com/Products/nRF5340, Accessed: 2025-02-10, 2025.

[188] NXP, *I.mx rt700 crossover mcus*, Accessed: 2025-02-07, 2025. [Online]. Available: https://www.nxp.com/products/i.MX-RT700

[189] J. Park, D.-W. Kim, and J. Lee, "Ht-aggnet: Hierarchical temporal aggregation network with near-zero-cost layer stacking for human activity recognition," *Engineering Applications of Artificial Intelligence*, vol. 149, p. 110 465, 2025, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2025.110465 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197625004658

[190] A. N. M. Rafee, M. A. O. Zishan, and J. Noor, "Decision tree-based real-time personalized human activity recognition and fall detection utilizing cost effective highly resource-constrained microcontrollers," *Available at SSRN 5146354*, 2025. DOI: 10.2139/ssrn.5146354 [Online]. Available: https://ssrn.com/abstract=5146354

[191] Raspberry Pi Foundation, *Pico Series Microcontrollers Documentation*, https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html, Accessed: 2025-02-10, 2025.

[192] N. sedaghati, S. ardebili, and A. Ghaffari, "Application of human activity/action recognition: A review," *Multimedia Tools and Applications*, Jan. 2025, ISSN: 1573-7721. DOI: 10.1007/s11042-024-20576-2 [Online]. Available: https://doi.org/10.1007/s11042-024-20576-2

[193] M. Sinigaglia et al., *Maestro: A 302 gflops/w and 19.8gflops risc-v vector-tensor architecture for wearable ultrasound edge computing*, 2025. arXiv: 2503.04581 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2503.04581

[194] STMicroelectronics, *Stm32 32-bit arm cortex mcus overview*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[195] C. D. Systems, *Xtensa lx7 processor product brief*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.cadence.com/en_US/home/resources/product-briefs/xtensa-lx7-processor-pb.html

[196] E. Systems, *Esp devkits | espressif systems*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.espressif.com/en/products/devkits

[197] E. Systems, *Esp32-s3*, Accessed: 2025-01-01, 2025. [Online]. Available: https://www.espressif.com/en/products/socs/esp32-s3

[198] T. Team, *Callback*, https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback, TensorFlow Keras Callback API Documentation, 2025.

[199] E. Tutorials, *Rc oscillator tutorial*, Accessed: 2025-01-25, 2025. [Online]. Available: https://www.electronics-tutorials.ws/oscillator/rc_oscillator.html

[200] E. Tutorials, *Rc oscillator tutorial*, Accessed: 2025-02-25, 2025. [Online]. Available: https://www.electronics-tutorials.ws/oscillator/rc_oscillator.html

[201] X. Wang et al., "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–39, Apr. 2025, ISSN: 1557-7341. DOI: 10.1145/3724420 [Online]. Available: http://dx.doi.org/10.1145/3724420

[202] B. Xiong, J. Lou, W. Ni, Z. Su, and S. Huang, "A multi-branch adaptive model with hybrid time—frequency loss to the enhanced joint moment prediction of prosthetic control and human motion applications," *Applied Sciences*, vol. 15, no. 4, 2025, ISSN: 2076-3417. DOI: 10.3390/app15041678 [Online]. Available: https://www.mdpi.com/2076-3417/15/4/1678

[203] Q. Zhao, F. Wang, W. Wang, T. Zhang, H. Wu, and W. Ning, "Research on intrusion detection model based on improved mlp algorithm," *Scientific Reports*, vol. 15, no. 1, p. 5159, 2025.

[204] R. E. Corporation, *Mcu course: Basic structure and operation*, https://www.renesas.com/en/support/engineer-school/mcu-01-basic-structure-operation, Accessed: 2025-01-27.

[205] *Stm32l4 series - stmicroelectronics*, https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html, Accessed: 2025-01-30.

[206] G. Documentation, *Optimize options*, Accessed: 2025-02-07, n.d. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[207] L. Project, *Clang command guide*, Accessed: 2025-02-07, n.d. [Online]. Available: https://clang.llvm.org/docs/CommandGuide/clang.html

[208] L. Project, *Clang language extensions*, Accessed: 2025-02-07, n.d. [Online]. Available: https://clang.llvm.org/docs/LanguageExtensions.html

Appendices

Appendix A

Architectures of the Reimplemented Models for Comparative Study

In this chapter, we describe the architecture and reimplementation process for each study incorporated in our comparative analysis. Generally, the core architecture of each model remains unchanged; instead, adjustments were made to the input shape, the number of classification classes, and other minor parameters to accommodate changes in input and output specifications. For studies that provided a complete codebase for model training, testing, and evaluation, we directly utilized that, and when necessary, extended the provided implementations to compute metrics such as *Total Parameters*, *Trainable Parameters*, and *MFLOPs*. In instances where the codebase was not provided, we adhered to the methodologies detailed in the respective papers by reimplementing the proposed architectures in TensorFlow and employing the *summary* and *profiler* APIs to calculate the required metrics.

When the codebase was provided, we employed the corresponding framework's APIs to compute the metrics. For example, for PyTorch implementations, we made use of the *summary* [157] and *ptflops* [139] libraries. The subsequent sections offer a comprehensive discussion of each model's architecture and detail the reimplementation strategies employed for an effective comparison with our study.

A.1 HARCNN

We implemented the HARCNN model [170] in TensorFlow by strictly adhering to the architecture presented in the original paper. The only modifications made were to the input configuration and the final Dense output layer. Further details regarding this implementation are provided in Table A.1.

Table A.1: Modified HARCNN model architecture and parameters. The input shape is (300, 9) representing IMU sensor data with 9 features and 300 time steps. "N" denotes the batch size.

Stage	**Layer Type**	**Parameters**	**Output Shape (N, ...)**
IMU Input	Input	Shape: (300, 9)	(N, 300, 9)
Conv Block 1	Conv1D	Filters: 32, Kernel: 5, Stride: 2, Padding: Same	(N, 150, 32)
	ReLU	-	
Conv Block 2	Conv1D	Filters: 48, Kernel: 3, Stride: 2, Padding: Same	(N, 75, 48)
	ReLU	-	
Concat Block 1	Conv1D (Branch A)	Filters: 48, Kernel: 3, Stride: 1, Padding: Same	(N, 75, 48)
	ReLU	-	
	Conv1D (Branch B)	Filters: 48, Kernel: 1, Stride: 1, Padding: Same	(N, 75, 48)
	ReLU	-	
	Concatenate	Axis: -1	(N, 75, 96)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Same	(N, 38, 96)
Conv Block 3	Conv1D	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 38, 64)
	ReLU	-	
Concat Block 2	Conv1D (Branch A)	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 38, 64)
	ReLU	-	
	Conv1D (Branch B)	Filters: 64, Kernel: 1, Stride: 1, Padding: Same	(N, 38, 64)
	ReLU	-	
	Concatenate	Axis: -1	(N, 38, 128)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Same	(N, 19, 128)

Continued on next page

Table A.1 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
Conv Block 4	Conv1D	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 19, 128)
	ReLU	-	
Concat Block 3	Conv1D (Branch A)	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 19, 128)
	ReLU	-	
	Conv1D (Branch B)	Filters: 128, Kernel: 1, Stride: 1, Padding: Same	(N, 19, 128)
	ReLU	-	
	Concatenate	Axis: -1	(N, 19, 256)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Same	(N, 10, 256)
Feature Flattening	Flatten	-	(N, 2560)
Dense Block	Dense	Units: 21	(N, 21)
	ReLU	-	
Output	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

A.2 ResLSTM

For the ResLSTM model [108], we extended the original TensorFlow implementation provided by the authors by incorporating our custom input and output configurations. In addition, we made the necessary adjustments for calculating both the total number of parameters and the MFLOPs (see Table A.2 for details).

Table A.2: ResLSTM architecture and parameters. The model takes three parallel input streams, each with shape (300, 3) representing sensor data with 3 features (3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer) and 300 time steps. Here "N" denotes the batch size.

Stage	Layer Type	Parameters	Output Shape (N, ...)
Input Stream 1	Input	Shape: (300, 3)	(N, 300, 3)
Input Stream 2	Input	Shape: (300, 3)	(N, 300, 3)

Continued on next page

Table A.2 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
Input Stream 3	Input	Shape: (300, 3)	(N, 300, 3)
Shared Processing Branch: CNN_ResBLSTM_Attention Sub-module (Applied to each input stream. Input to this branch is (N, 300, 3), output is (N, 128))			
CNN Block 1	Conv1D	Filters: 256, Kernel: 3, Padding: Same, Act: Swish	(N, 300, 256)
	BatchNormalization	-	(N, 300, 256)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Valid	(N, 150, 256)
	Dropout	Rate: 0.2	(N, 150, 256)
CNN Block 2	Conv1D	Filters: 128, Kernel: 3, Padding: Same, Act: Swish	(N, 150, 128)
	BatchNormalization	-	(N, 150, 128)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Valid	(N, 75, 128)
	Dropout	Rate: 0.2	(N, 75, 128)
CNN Block 3	Conv1D	Filters: 64, Kernel: 3, Padding: Same, Act: Swish	(N, 75, 64)
	BatchNormalization	-	(N, 75, 64)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Valid	(N, 37, 64)
	Dropout	Rate: 0.2	(N, 37, 64)
CNN Block 4	Conv1D	Filters: 32, Kernel: 3, Padding: Same, Act: Swish	(N, 37, 32)
	BatchNormalization	-	(N, 37, 32)
	MaxPooling1D	Pool Size: 2, Strides: 2, Padding: Valid	(N, 18, 32)
	Dropout	Rate: 0.2	(N, 18, 32)
ResBLSTM Block	LayerNormalization	-	(N, 18, 32)
	Bidirectional LSTM (x)	Units: 64, Return Sequences: True	(N, 18, 128)
	LayerNormalization	-	(N, 18, 128)
	Bidirectional LSTM (y)	Units: 64, Return Sequences: True	(N, 18, 128)
	Add	Inputs: [x, y]	(N, 18, 128)

Continued on next page

Table A.2 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
Attention Mechanism	Attention (Custom)	Step Dim: 18	(N, 128)
Fusion and Output Head			
Feature Concatenation	Concatenate	From 3 branches, Input per branch: (N, 128)	(N, 384)
Classifier	Dropout	Rate: 0.3	(N, 384)
	Dense	Units: 21	(N, 21)
	Activation	Type: Softmax	(N, 21)

A.3 HT-AggNet

Similarly, for the HT-AggNet [189] model, we extended the original PyTorch implementation (see Table A.3) by incorporating our custom input and output configurations.

Table A.3: The architecture and parameters of the HT-AggNet model are outlined below. The input data, consisting of IMU sensor measurements, has a shape of $(N, 9, 300)$, where 9 represents the number of features, 300 denotes the time steps, and "N" is the batch size. The model is constructed with $L = 8$ TAggBlocks, utilizes an internal channel dimension of $nc\_o = 128$, and is modified to perform classification across 21 classes.

Stage	Layer Type	Parameters	Output Shape (N, ...)
IMU Input	Input	Shape: (9, 300) (Features, Timesteps)	(N, 9, 300)
Stem Convolution	Conv1D	In: 9, Out: 128, Kernel: 5, Padding: Same	(N, 128, 300)
Initial Pooling	MaxPool1d	Kernel: 3, Stride: 2, Padding: 1	(N, 128, 150)
TAggBlock Processing Loop (L=8 iterations) (Input x to the first block is (N, 128, 150). The l_feat (N, 128, 150) from block $i-1$ is input x to block i.) (Each block i also outputs a g_feat (N, 16, 1) for final aggregation.)			

Continued on next page

Table A.3 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
TAggBlock (generic)	Grouped Conv1D (*gconv*)	In: 128, Out: 128, Kernel: 5, Padding: Same, Groups: 8, Bias: False	(N, 128, 150)
	BatchNorm1d	Features: 128	(N, 128, 150)
	ReLU	In-place after BatchNorm	(N, 128, 150)
	ChannelShuffle	Groups: 8	(N, 128, 150) (l_feat)
	Temp. Global Conv1D (*gconv*)	In: 128, Out: 16 (128/L), Kernel: 150 (input T)	(N, 16, 1) (g_feat)
Output Head			
Feature Aggregation	Concatenate (g_feats)	L=8 tensors of (N, 16, 1) along channel dim	(N, 128, 1)
Flattening	View	Target shape: (N, -1)	(N, 128)
Classifier	Fully Connected	In: 128, Out: 21	(N, 21)
	LogSoftmax	Dim: 1	(N, 21)

A.4 Deep Context Models

Similar to the HARCNN model discussed previously in Appendix Section A.1, we implemented the proposed models in TensorFlow by closely following the methodologies outlined in the original research papers. Detailed information regarding the architectures and parameter configurations is presented in Appendix Subsection A.4.1, Subsection A.4.2, and Subsection A.4.3.

A.4.1 Static Context CNN

Table A.4: The architecture and parameters of the Static Context CNN model are detailed below. The input shape is $(300, 9, 1)$, which represents IMU sensor data reshaped to include a single channel, with 9 units as the width (features) and 300 time steps as the height. Here, "N" denotes the batch size.

Stage	Layer Type	Parameters	Output Shape (N, ...)
IMU Input (reshaped)	Input	Shape: (300, 9, 1)	(N, 300, 9, 1)

Continued on next page

Table A.4 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
Conv Block	Conv2D	Filters: 256, Kernel: (3,3), Padding: Same	(N, 300, 9, 256)
	ReLU	-	
	MaxPooling2D	Pool Size: (2,2)	(N, 150, 4, 256)
	Dropout	Rate: 0.25	(N, 150, 4, 256)
Feature Flattening	Flatten	-	(N, 153600)
Dense Block 1	Dense	Units: 128	(N, 128)
	ReLU	-	
	Dropout	Rate: 0.25	(N, 128)
Output Block	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

A.4.2 Dynamic Context RNN

Table A.5: The architecture and parameters of the Dynamic Context RNN model are outlined below. The input shape is $(300, 9)$, representing IMU sensor data captured over 300 time steps with 9 features, where "N" denotes the batch size. Additionally, the RNN layer was unrolled to ensure compatibility with TFLM.

Stage	Layer Type	Parameters	Output Shape (N, ...)
IMU Input	Input	Shape: (300, 9) (Timesteps, Features)	(N, 300, 9)
Recurrent Layer	RNN (SimpleRNNCell)	Units: 128, Unroll: True	(N, 128)
Dropout Layer 1	Dropout	Rate: 0.2	(N, 128)
Dense Block 1	Dense	Units: 64	(N, 64)
	ReLU	-	
Dropout Layer 2	Dropout	Rate: 0.5	(N, 64)
Output Block	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

A.4.3 Dynamic Context LSTM

Table A.6: The architecture and parameters of the Dynamic Context LSTM model are detailed below. The model accepts IMU sensor data with an input shape of $(300, 9)$, where 9 features are recorded over 300 time steps, and "N" denotes the batch size. Similar to the Dynamic Context RNN model, the LSTM layer was unrolled to ensure TFLM compatibility.

Stage	**Layer Type**	**Parameters**	**Output Shape (N, ...)**
IMU Input	Input	Shape: (300, 9)	(N, 300, 9)
LSTM Layer 1	LSTM	Units: 30, Return Sequences: True, Unroll: True	(N, 300, 30)
LSTM Layer 2	LSTM	Units: 30, Unroll: True	(N, 30)
Output	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

Appendix B

Teacher Model Architecture

In the below Table B.1 we present the detailed DualCrossTAP teacher model's architecture and parameters necessary to reproduce our study.

Table B.1: DualCrossTAP Teacher model architecture and parameters. Input shapes are based on IMU data with 9 features and PP data with 16 features, both sampled at 300 time steps. "N" denotes the batch size. Moreover, the optimal parameters are selected via an extensive grid search.

Stage	Layer Type	Parameters	Output Shape (N, ...)
IMU Branch			
IMU Input	Input	Shape: (2700,)	(N, 2700)
	Reshape	Target Shape: (300, 9)	(N, 300, 9)
Conv Block 1	Conv1D	Filters: 32, Kernel: 3, Stride: 1, Padding: Same	(N, 300, 32)
	Batch Norm	-	
	ReLU	-	
	Residual Add	(Shortcut 1x1 Conv if needed)	
	MaxPooling1D	Pool Size: 2	(N, 150, 32)
Conv Block 2	Conv1D	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 150, 64)
	Batch Norm	-	
	ReLU	-	
	Residual Add	(Shortcut 1x1 Conv if needed)	
	MaxPooling1D	Pool Size: 2	(N, 75, 64)
Conv Block 3	Conv1D	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 75, 128)
	Batch Norm	-	
	ReLU	-	

Continued on next page

Table B.1 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
	Residual Add	(Shortcut 1x1 Conv if needed)	
Transformer (IMU Features)	MultiHeadAttention	Heads: 4, Key Dim: 32	(N, 75, 128)
	Dropout	Rate: 0.1	
	LayerNorm	Epsilon: 1e-6 (Residual Add)	
	Dense (FFN)	Units: 128 (ReLU), Units: 128 (Input Dim)	
	Dropout	Rate: 0.1	
	LayerNorm	Epsilon: 1e-6 (Residual Add)	
Plantar Pressure Branch			
PP Input	Input	Shape: (4800,)	(N, 4800)
	Reshape	Target Shape: (300, 16)	(N, 300, 16)
Conv Block 1	Conv1D	Filters: 32, Kernel: 3, Stride: 1, Padding: Same	(N, 300, 32)
	Batch Norm	-	
	ReLU	-	
	Residual Add	(Shortcut 1x1 Conv if needed)	
	MaxPooling1D	Pool Size: 2	(N, 150, 32)
Conv Block 2	Conv1D	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 150, 64)
	Batch Norm	-	
	ReLU	-	
	Residual Add	(Shortcut 1x1 Conv if needed)	
	MaxPooling1D	Pool Size: 2	(N, 75, 64)
Conv Block 3	Conv1D	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 75, 128)
	Batch Norm	-	
	ReLU	-	
	Residual Add	(Shortcut 1x1 Conv if needed)	
Transformer (PP Features)	MultiHeadAttention	Heads: 4, Key Dim: 32	(N, 75, 128)
	Dropout	Rate: 0.1	
	LayerNorm	Epsilon: 1e-6 (Residual Add)	
	Dense (FFN)	Units: 128 (ReLU), Units: 128 (Input Dim)	

Continued on next page

Table B.1 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
	Dropout	Rate: 0.1	
	LayerNorm	Epsilon: 1e-6 (Residual Add)	
Cross-Modal Interaction			
IMU Attends PP	MultiHeadAttention	Heads: 4, Key Dim: 32 (Query: IMU, Value: PP)	(N, 75, 128)
	Add	(IMU Features + IMU Attended)	
PP Attends IMU	MultiHeadAttention	Heads: 4, Key Dim: 32 (Query: PP, Value: IMU)	
	Add	(PP Features + PP Attended)	
Temporal Pooling and Classification Head			
IMU Path	TemporalAttention	Dense Units: 1 (tanh), Softmax over time	(N, 128)
PP Path	TemporalAttention	Dense Units: 1 (tanh), Softmax over time	
Concatenate	Concatenate	(IMU Pooled, PP Pooled)	(N, 256)
Dense Block 1	Dense	Units: 512, Activation: Swish, Regularizer: L2	(N, 512)
	Batch Norm	-	
	Dropout	Rate: 0.6	
Dense Block 2	Dense	Units: 256, Activation: Swish, Regularizer: L2	(N, 256)
	Batch Norm	-	
	Dropout	Rate: 0.5	
Output	Dense	Units: 21 (num_classes), Activation: Softmax	(N, 21)

Appendix C

Student Model Architecture

In this chapter we provide the designed student CNN models' architecture and parameters in Section C.1 and Section C.2 respectively. The primary difference between these two models is that CNN-S1 utilizes *Conv1D* whereas CNN-S2 utilizes *SeparableConv1D*. Other than that, the entire architecture is identical to one another.

C.1 CNN-S1 model architecture

Table C.1: Student CNN-S1 model architecture and parameters. The input shape is (300, 9) representing IMU sensor data with 9 features with 300 time steps. "N" denotes the batch size.

Stage	**Layer Type**	**Parameters**	**Output Shape (N, ...)**
IMU Input	Input	Shape: (300, 9)	(N, 300, 9)
Conv Block 1	Conv1D	Filters: 32, Kernel: 3, Stride: 1, Padding: Same	(N, 300, 32)
	ReLU	-	
	MaxPooling1D	Pool Size: 2	(N, 150, 32)
Conv Block 2	Conv1D	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 150, 64)
	ReLU	-	
	MaxPooling1D	Pool Size: 2	(N, 75, 64)
Conv Block 3	Conv1D	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 75, 128)
	ReLU	-	
Feature Pooling	GlobalAveragePooling1D	-	(N, 128)

Continued on next page

Table C.1 – continued from previous page

Stage	Layer Type	Parameters	Output Shape (N, ...)
Dense Block	Dense	Units: 128, Activation: ReLU	(N, 128)
	Dropout	Rate: 0.5	
Output	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

C.2 CNN-S2 model architecture

Table C.2: Student CNN-S2 model architecture and parameters. The input shape is (300, 9) representing IMU data with 9 features with 300 time steps. "N" denotes the batch size. This model employs separable convolutions for improved efficiency compared to the previous CNN-S1.

Stage	Layer Type	Parameters	Output Shape (N, ...)
IMU Input	Input	Shape: (300, 9)	(N, 300, 9)
Conv Block 1	SeparableConv1D	Filters: 32, Kernel: 3, Stride: 1, Padding: Same	(N, 300, 32)
	ReLU	-	
	MaxPooling1D	Pool Size: 2	(N, 150, 32)
Conv Block 2	SeparableConv1D	Filters: 64, Kernel: 3, Stride: 1, Padding: Same	(N, 150, 64)
	ReLU	-	
	MaxPooling1D	Pool Size: 2	(N, 75, 64)
Conv Block 3	SeparableConv1D	Filters: 128, Kernel: 3, Stride: 1, Padding: Same	(N, 75, 128)
	ReLU	-	
Feature Pooling	GlobalAveragePooling1D	-	(N, 128)
Dense Block	Dense	Units: 128, Activation: ReLU	(N, 128)
	Dropout	Rate: 0.5	
Output	Dense	Units: 21 (num_classes)	(N, 21)
	Softmax	-	

Appendix D

CNN-S1 Per-Operations Execution Time Breakdown

In Table D.1 to Table D.7, we present per-layer and per-operation execution time for the mentioned frequencies in Table 6.6.

Table D.1: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 240 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	7	85
	STRIDED_SLICE	32	
	PACK	9	
	RESHAPE	37	
Convolution	CONV_2D	154202	154202
Max Pool	MAX_POOL_2D	6037	6037
Convolution	CONV_2D	517005	517005
Max Pool	MAX_POOL_2D	6011	6011
Convolution	CONV_2D	1137845	1137845
Global Average Pooling	MEAN	11927	11927
Dense	FULLY_CONNECTED	2674	2674
Output (Dense)	FULLY_CONNECTED	443	492
	SOFTMAX	49	
	62 KB (SRAM)		**1836278**

Table D.2: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3 running on DFS mode with minimum CPU frequency set to 80 MHz and maximum frequency set to 240 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	8	111
	STRIDED_SLICE	49	
	PACK	17	
	RESHAPE	37	
Convolution	CONV_2D	154227	154227
Max Pool	MAX_POOL_2D	6047	6047
Convolution	CONV_2D	517003	517003
Max Pool	MAX_POOL_2D	6015	6015
Convolution	CONV_2D	1138395	1138395
Global Average Pooling	MEAN	11938	11938
Dense	FULLY_CONNECTED	2682	2682
Output (Dense)	FULLY_CONNECTED	449	523
	SOFTMAX	74	
	62 KB (SRAM)		**1836941**

Table D.3: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 160 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	15	129
	STRIDED_SLICE	43	
	PACK	17	
	RESHAPE	54	
Convolution	CONV_2D	232123	232123
Max Pool	MAX_POOL_2D	9092	9092
Convolution	CONV_2D	778217	778217
Max Pool	MAX_POOL_2D	9049	9049

Continued on next page

Table D.3 – Continued from previous page

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Convolution	CONV_2D	1655945	1655945
Global Average Pooling	MEAN	17953	17953
Dense	FULLY_CONNECTED	2783	2783
Output (Dense)	FULLY_CONNECTED	462	544
	SOFTMAX	82	
62 KB (SRAM)			**2705835**

Table D.4: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 80 MHz.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	15	188
	STRIDED_SLICE	54	
	PACK	20	
	RESHAPE	99	
Convolution	CONV_2D	469359	469359
Max Pool	MAX_POOL_2D	18351	18351
Convolution	CONV_2D	1573541	1573541
Max Pool	MAX_POOL_2D	18290	18290
Convolution	CONV_2D	3234039	3234039
Global Average Pooling	MEAN	36291	36291
Dense	FULLY_CONNECTED	3656	3656
Output (Dense)	FULLY_CONNECTED	606	731
	SOFTMAX	125	
62 KB (SRAM)			**5354446**

Table D.5: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 40 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	30	370
	STRIDED_SLICE	103	
	PACK	40	
	RESHAPE	197	
Convolution	CONV_2D	960615	960615
Max Pool	MAX_POOL_2D	37560	37560
Convolution	CONV_2D	3220352	3220352
Max Pool	MAX_POOL_2D	37436	37436
Convolution	CONV_2D	6620034	6620034
Global Average Pooling	MEAN	74317	74317
Dense	FULLY_CONNECTED	7486	7486
Output (Dense)	FULLY_CONNECTED	1256	1510
	SOFTMAX	254	
	62 KB (SRAM)		**10959680**

Table D.6: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 20 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	57	733
	STRIDED_SLICE	200	
	PACK	86	
	RESHAPE	390	
Convolution	CONV_2D	2016317	2016317
Max Pool	MAX_POOL_2D	78829	78829
Convolution	CONV_2D	6760170	6760170
Max Pool	MAX_POOL_2D	78648	78648

Continued on next page

Table D.6 – Continued from previous page

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Convolution	CONV_2D	13896236	13896236
Global Average Pooling	MEAN	156055	156055
Dense	FULLY_CONNECTED	15706	15706
Output (Dense)	FULLY_CONNECTED	2611	3115
	SOFTMAX	504	
62 KB (SRAM)			**23005809**

Table D.7: Per-layer and per-operation execution time for CNN-S1 on ESP32-S3, running at a constant CPU frequency of 10 MHz.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	106	1453
	STRIDED_SLICE	399	
	PACK	171	
	RESHAPE	777	
Convolution	CONV_2D	4464553	4464553
Max Pool	MAX_POOL_2D	174732	174732
Convolution	CONV_2D	14969229	14969229
Max Pool	MAX_POOL_2D	174195	174195
Convolution	CONV_2D	30746369	30746369
Global Average Pooling	MEAN	345729	345729
Dense	FULLY_CONNECTED	34890	34890
Output (Dense)	FULLY_CONNECTED	5740	6939
	SOFTMAX	1199	
62 KB (SRAM)			**50918089**

Appendix E

CNN-S2 Per-Operations Execution Time Breakdown

Tables E.1 to E.6 present the per-layer and per-operation execution time for the specified frequencies in Table 6.11, excluding 240 MHz. A detailed breakdown of inference latency at 240 MHz is already provided in Table 6.9 and is omitted here to avoid redundancy, as the inference latency remains unchanged.

Table E.1: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3 running on DFS mode with minimum CPU frequency set to 80 MHz and maximum frequency set to 240 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	10	114
	STRIDED_SLICE	46	
	PACK	18	
	RESHAPE	40	
Sep Convolution	DEPTHWISE_CONV_2D	7347	61761
	CONV_2D	54414	
Max Pool	MAX_POOL_2D	6052	6052
Sep Convolution	DEPTHWISE_CONV_2D	12878	188820
	CONV_2D	175942	
Max Pool	MAX_POOL_2D	6009	6009
Sep Convolution	DEPTHWISE_CONV_2D	12837	361656
	CONV_2D	348819	
Global Average Pooling	MEAN	11932	11932

Continued on next page

Table E.1 – Continued from previous page

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Output (Dense)	FULLY_CONNECTED	2677	3185
	FULLY_CONNECTED	448	
82 KB (SRAM)			**639529**

Table E.2: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 160 MHz.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	11	133
	STRIDED_SLICE	46	
	PACK	21	
	RESHAPE	55	
Sep Convolution	DEPTHWISE_CONV_2D	11039	92931
	CONV_2D	81892	
Max Pool	MAX_POOL_2D	9095	9095
Sep Convolution	DEPTHWISE_CONV_2D	19384	284216
	CONV_2D	264832	
Max Pool	MAX_POOL_2D	9046	9046
Sep Convolution	DEPTHWISE_CONV_2D	19320	542608
	CONV_2D	523288	
Global Average Pooling	MEAN	17949	17949
Output (Dense)	FULLY_CONNECTED	2776	3316
	FULLY_CONNECTED	463	
82 KB (SRAM)			**959294**

Table E.3: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 80 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	12	188
	STRIDED_SLICE	55	
	PACK	22	
	RESHAPE	99	
Sep Convolution	DEPTHWISE_CONV_2D	22261	187823
	CONV_2D	165562	
Max Pool	MAX_POOL_2D	18359	18359
Sep Convolution	DEPTHWISE_CONV_2D	39194	574662
	CONV_2D	535468	
Max Pool	MAX_POOL_2D	18285	18285
Sep Convolution	DEPTHWISE_CONV_2D	39056	1093546
	CONV_2D	1054490	
Global Average Pooling	MEAN	36290	36290
Output (Dense)	FULLY_CONNECTED	3652	4388
	FULLY_CONNECTED	608	
	82 KB (SRAM)		**1933541**

Table E.4: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 40 MHz.

Layer	**Operation**	**Operation Execution Time (µs)**	**Layer Total Execution Time (µs)**
Reshape	SHAPE	25	374
	STRIDED_SLICE	106	
	PACK	47	
	RESHAPE	196	
Sep Convolution	DEPTHWISE_CONV_2D	45564	384477
	CONV_2D	338913	
Max Pool	MAX_POOL_2D	37573	37573

Continued on next page

Table E.4 – Continued from previous page

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Sep Convolution	DEPTHWISE_CONV_2D	80207	1176555
	CONV_2D	1096348	
Max Pool	MAX_POOL_2D	37428	37428
Sep Convolution	DEPTHWISE_CONV_2D	79987	2238622
	CONV_2D	2158635	
Global Average Pooling	MEAN	74320	74320
Output (Dense)	FULLY_CONNECTED	7477	9000
	FULLY_CONNECTED	1262	
	82 KB (SRAM)		**3958349**

Table E.5: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 20 MHz.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	50	749
	STRIDED_SLICE	212	
	PACK	94	
	RESHAPE	393	
Sep Convolution	DEPTHWISE_CONV_2D	95653	807034
	CONV_2D	711381	
Max Pool	MAX_POOL_2D	78839	78839
Sep Convolution	DEPTHWISE_CONV_2D	168420	2467800
	CONV_2D	2299380	
Max Pool	MAX_POOL_2D	78648	78648
Sep Convolution	DEPTHWISE_CONV_2D	167711	4696984
	CONV_2D	4529273	
Global Average Pooling	MEAN	156073	156073

Continued on next page

Table E.5 – Continued from previous page

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Output (Dense)	FULLY_CONNECTED	15670	18808
	FULLY_CONNECTED	2616	
82 KB (SRAM)			**8304935**

Table E.6: Per-layer and per-operation execution time for CNN-S2 on ESP32-S3, running at a constant CPU frequency of 10 MHz.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Reshape	SHAPE	97	1492
	STRIDED_SLICE	423	
	PACK	187	
	RESHAPE	785	
Sep Convolution	DEPTHWISE_CONV_2D	212061	1788460
	CONV_2D	1576399	
Max Pool	MAX_POOL_2D	174759	174759
Sep Convolution	DEPTHWISE_CONV_2D	373264	5474536
	CONV_2D	5101272	
Max Pool	MAX_POOL_2D	174195	174195
Sep Convolution	DEPTHWISE_CONV_2D	371522	10402104
	CONV_2D	10030582	
Global Average Pooling	MEAN	346018	346018
Output (Dense)	FULLY_CONNECTED	34660	41641
	FULLY_CONNECTED	5753	
82 KB (SRAM)			**18403205**

Appendix F

MCU on-Device Testing Workflow

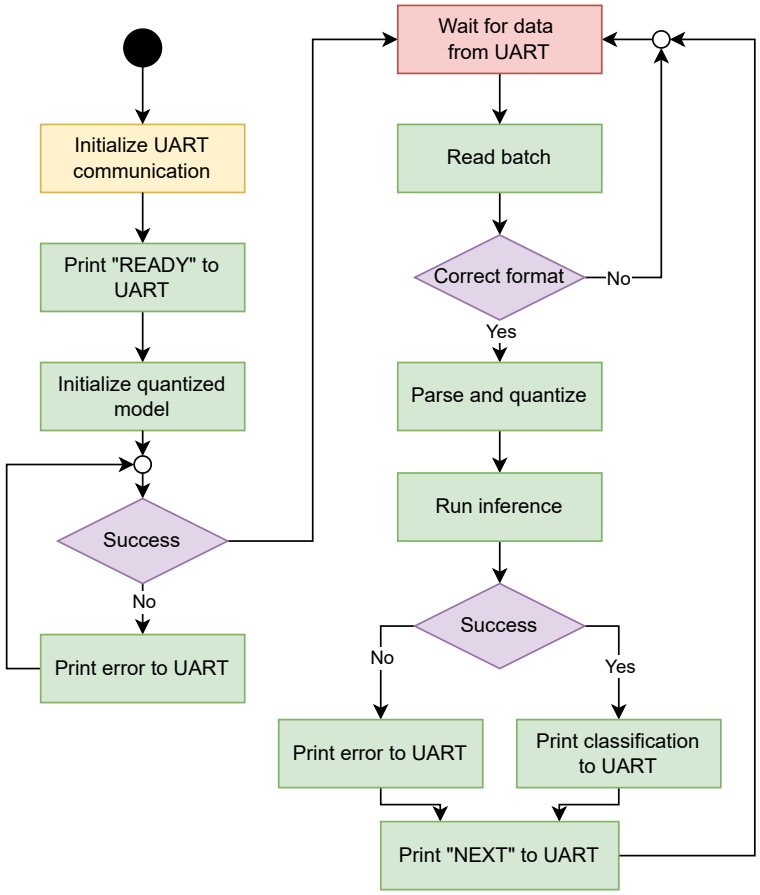


Figure F.1: Diagram showing the MCU inference and workstation communication workflow. The process begins with initializing UART serial communication, printing "READY" and initializing the model. If successful, the MCU enters the main processing loop where it waits for serial data, parses and quantizes it, runs inference using the model, and prints the result or error to UART before repeating the loop.

This chapter details a robust and generalized workflow for the on-device testing [120], [173], [201] of machine learning models deployed on MCUs. The process involves a coordinated effort between a host workstation and the target MCU to facilitate data transfer, inference execution, and result analysis. This process is crucial for

validating the real-world performance of a model under the hardware and software constraints of an embedded system. The workflow is designed to be model-agnostic, applicable to various ML and DL architectures, and adaptable to different MCU platforms and communication interfaces.

F.1 Introduction and Overview

Deploying ML and DL models on resource-constrained devices like MCUs, a practice often referred to as TinyML, presents unique challenges. While model performance is typically evaluated in a high-level simulation environment during development, this does not capture the potential performance degradation or operational issues that can arise from hardware limitations, memory constraints, and quantization effects. Therefore, on-device testing is an indispensable step to verify that the model behaves as expected on the target hardware.

This section outlines a comprehensive workflow that establishes a communication bridge between a workstation (in this case our PC) and an MCU. The workstation acts as a test coordinator, sending data samples and receiving inference results, while the MCU performs the actual on-device computation. This setup allows for systematic, automated testing of the deployed model with large datasets, enabling the generation of detailed performance metrics directly from the target device.

F.2 System Architecture and Components

The testing architecture is composed of two primary components: the Workstation and the MCU. Their roles and interactions are central to the workflow.

F.2.1 Workstation

The workstation serves as the master controller for the testing process. Its responsibilities include:

- **Data Management:** Loading and preparing the test dataset, which consists of input samples and their corresponding ground truth labels.
- **Test Orchestration:** Sequentially sending individual test samples to the MCU.
- **Data Reception:** Listening for and receiving the inference results computed by the MCU.
- **Synchronization:** Managing the flow of data to prevent buffer overflows and ensure that the workstation and MCU remain in sync.
- **Performance Analysis:** Aggregating the predictions, comparing them against the true labels, and generating a final classification report.

A high-level scripting language, such as Python, is typically used on the workstation for its powerful data handling libraries and ease of serial (UART) communication

management. During our testing phase, we've utilized Python since many of our codebases' components were written in Python.

F.2.2 Microcontroller Units

The MCU is the target device where the ML model is deployed. It acts as a slave node in this workflow, responding to the commands from the workstation. Its primary functions are:

- **Model Hosting:** Storing the quantized ML model in its flash memory and loading it into RAM for execution.
- **Data Reception:** Receiving test data samples sent from the workstation via a serial interface.
- **Preprocessing:** Performing any necessary data transformations, such as quantization (see Equation F.1), to match the model's input requirements.
- **Inference:** Executing the ML model on the received data to generate a prediction.
- **Result Transmission:** Sending the prediction back to the workstation.

The firmware on the MCU is typically written in a low-level language like C/C++ to ensure maximum performance and control over the hardware. In this study, we utilized vendor-provided Arduino wrapper around ESP-IDF.

F.3 Communication Protocol

A well-defined communication protocol is essential for reliable data exchange. A simple, yet effective, handshake-based protocol over a standard serial interface (i.e., UART) is employed. This protocol ensures that both devices are ready before any data is transferred, preventing data loss and synchronization errors.
The protocol relies on a set of simple string-based signals:

- `READY`: Upon initialization, the MCU sends this signal to the workstation to indicate that it has successfully loaded the model and is ready to begin the testing process.
- `START`: The workstation prefixes each data sample with this signal. When the MCU receives `START`, it knows that the subsequent data stream constitutes a single input vector for inference.
- `NEXT`: After the MCU completes an inference and sends the result, it transmits this signal to the workstation. This informs the workstation that the MCU is prepared to receive the next data sample.

This protocol creates a synchronized, lock-step interaction, as illustrated in Figure F.2 and Figure F.1.

F.4 Workstation Operations

The workstation-side logic is managed by a script that handles the entire testing lifecycle. To manage the asynchronous nature of serial communication, a multi-threaded approach is used in our study, with one thread for sending data and another for receiving results.

F.4.1 Data Preparation and Loading

The first step is to load the test dataset, which is typically stored in text files. The input features (`X_test`) and the ground truth labels (`y_test`) are loaded into memory.

F.4.2 Data Transmission and Synchronization

The core of the workstation's operation is a loop that iterates through each sample in the test set.

1. **Wait for Signal:** For every sample after the first, the sender thread waits for the `NEXT` signal from the MCU. This ensures the MCU is not overwhelmed with data.
2. **Send Data:** The workstation sends the `START` signal, followed by the space-delimited floating-point values of the input sample. A newline character (`\n`) marks the end of the sample.
3. **Event Signaling:** After sending the data, the sender thread uses an event flag to notify the reader thread that it should start listening for a response. The sender then waits for a corresponding event from the reader, indicating that the result has been received.

F.4.3 Result Reception

The reader thread runs in parallel and is responsible for capturing the output from the MCU.

1. **Wait for Signal:** The reader thread waits for the event flag from the sender.
2. **Read and Parse:** Once signaled, it reads from the serial port until it captures the inference result. It must be robust enough to parse the expected integer classification, potentially ignoring any other debug messages or logs that the MCU might output.
3. **Store Result:** The validated result is written to a prediction file (`y_pred.txt`).
4. **Event Signaling:** The reader thread sets an event flag to signal the sender thread that the transaction is complete, allowing the next sample to be sent.

F.5 MCU Operations

The MCU firmware is designed to be a lean, efficient inference engine that communicates with the workstation.

F.5.1 Initialization

In the `setup()` function, the MCU performs the following critical tasks:

1. Initializes the serial communication at a predetermined baud rate (e.g., 115200).
2. Initializes the TFLM interpreter, allocating a memory region (the "tensor arena") for the model's inputs, outputs, and intermediate tensors.
3. Loads the ML model, which is typically compiled into the program as a byte array.
4. Sends the initial `READY` signal to the workstation.

F.5.2 Main Execution Loop

The `loop()` function continuously checks the serial buffer for incoming data.

1. **Wait for Start Signal:** The MCU waits for a line starting with the `START` signal.
2. **Data Parsing:** Upon receiving `START`, it parses the subsequent space-separated string of numbers, converting them into a floating-point array. This array represents the input vector for the model.
3. **Preprocessing and Quantization:** ML models on MCUs are most of the time quantized to use integer arithmetic (e.g., `INT8`) for efficiency. This step involves taking the floating-point input array and applying a quantization function. This function maps each float value to its corresponding 8-bit integer representation based on a pre-calculated scale and zero-point.

$$q = \text{round}\left(\frac{f}{\text{scale}} + \text{zero\_point}\right) \tag{F.1}$$

   where q is the quantized value and f is the floating-point value.
4. **Inference Execution:** The quantized input tensor is passed to the TFLM interpreter, which runs the model to produce an output tensor.
5. **Result Transmission:** The index of the highest value in the output tensor (the predicted class) is extracted. This integer result is printed to the serial port, sending it back to the workstation.
6. **Signal Readiness:** Finally, the MCU prints the `NEXT` signal to the serial port, indicating it is ready for the next iteration.

F.6 Post-Processing and Analysis

Once the workstation has finished sending all test samples and has received all predictions, the final phase of the workflow begins.
A separate script on the workstation reads the ground truth labels (`y_test.txt`) and the MCU's predictions (`y_pred.txt`). Using a library like Scikit-learn [10], it generates a standard classification report. This report provides key metrics such as:

- **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations.
- **Recall (Sensitivity):** The ratio of correctly predicted positive observations to all observations in the actual class.
- **F1 Score:** The weighted average of Precision and Recall.

This report gives a definitive, quantitative measure of the model's performance when running on the actual target hardware.

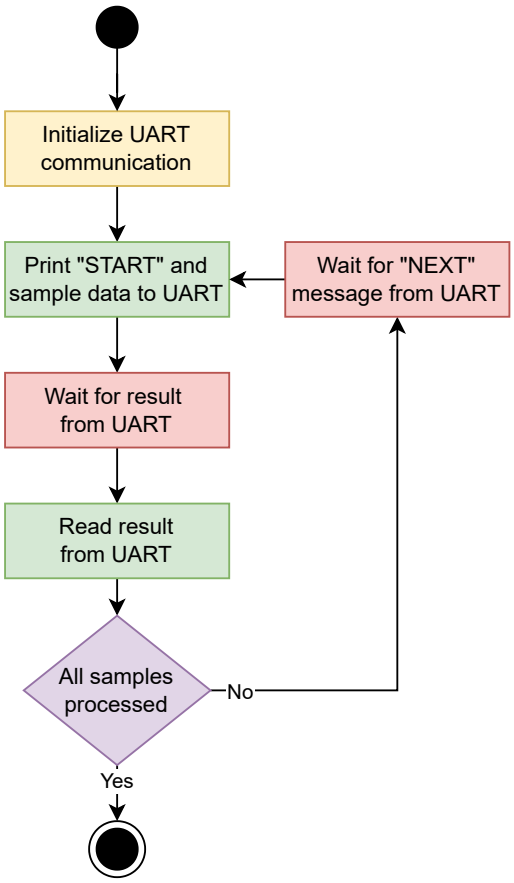


Figure F.2: Diagram illustrating the workstation to MCU communication workflow. The process begins with initializing UART serial communication and sending the first sample to the UART conneciton. The PC then waits for the result, and sends the next sample until all samples are processsed.

F.7 Ending Remarks

The on-device testing workflow described in this chapter provides a systematic and reliable method for validating the performance of embedded machine learning models. By establishing a clear communication protocol and division of labor between a workstation and an MCU, it enables automated, large-scale testing directly on the target hardware. This process is critical for identifying and mitigating issues related to quantization, memory limitations, and other hardware-specific constraints, ultimately ensuring that the deployed model is both accurate and robust.

Appendix G

Supplementary Experimentation on MLP with CAPPIMU Dataset

In this chapter, we describe the experimental procedure and present the results of applying a standard MLP model to the CAPPIMU dataset. As established in Chapter 3, MLP architectures often deliver greater compute efficiency than CNN models. This motivates our investigation into how an MLP performs on the CAPPIMU dataset, especially under the tight memory and processing constraints of MCUs. Chapters 5 and Chapter 6 introduced two CNN-based designs (CNN-S1 and CNN-S2) that achieve high classification accuracy but still impose notable computational demands. In contrast, fully connected layers which are the building blocks of MLP are inherently well suited to resource-constrained hardware [169]. Accordingly, we explore whether an optimized MLP can reduce inference latency and resource usage while maintaining competitive accuracy. We also examine the trade-off between speed and performance, as faster inference may come at the cost of lower accuracy (see Chapter 3). Figure G.1 illustrates the high-level methodology for our utilized MLP model. The sections that follow detail the methods, results, and analysis of this investigation.

G.1 Employed Methods

In this section we briefly outline the employed methods for the performance evaluation of our MLP model. We start our discussion with the CAPPIMU dataset preprocessing specific to this investigation.

G.1.1 CAPPIMU Dataset Preprocessing

The preprocessing pipeline applied to the CAPPIMU dataset closely follows the procedure described in Section 4.1, with specific adaptations to accommodate a computationally efficient MLP model. Unlike the CNN architectures previously proposed, our goal is to minimize Floating Point Operations (FLOPs) and inference latency on the ESP32-S3. To achieve this, we configure:

- A two-second sliding window with a one-second (50%) overlap (smaller window size; see Section 5.3).

- IMU data recorded at the right wrist, which provides more discriminative features (see Chapter 5) and reduces input dimensionality (9 channels vs. 16 for PP sensors).

We adopt the same train–test split (80% train and 20%) referenced in Section 4.1, reserving 15% of the training set for validation.

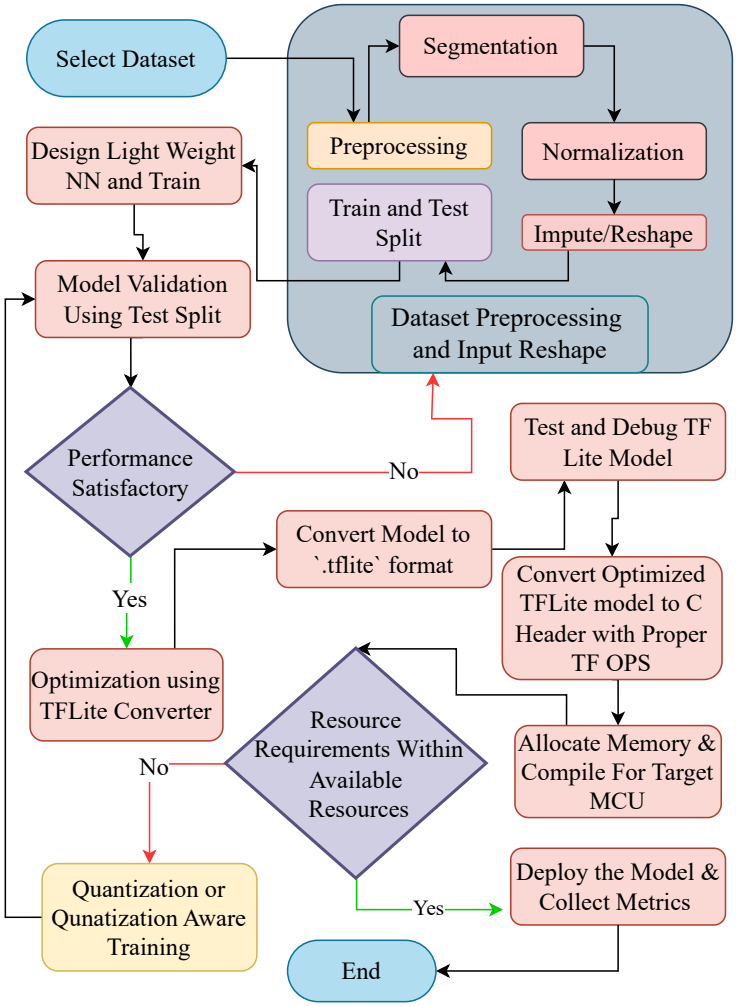


Figure G.1: Employed high-level MLP model training, testing, and validation methodology.

G.1.2 MLP Model Training and Validation

We design and train a lightweight NN model tailored for low-computation environments. A MLP architecture (see Section G.2.1) is selected due to its simplicity and efficiency. The network is trained on the training dataset using a stochastic gradient descent optimizer (ADAM [153]), with a suitable learning rate and loss function. During training, we monitor key metrics such as accuracy and loss to ensure the model is learning effectively. Following the training process, we validate the model using the test split. This step assesses the model's ability to generalize

to unseen data. Key performance metrics, such as accuracy, F1 Score, and precision are assessed per activity respectively. If the model does not meet the desired performance criteria, we return to the training step to fine-tune the network or adjust hyperparameters.

G.1.3 TensorFlow Lite Conversion

Once the model achieves satisfactory performance, it is converted into the TensorFlow Lite (*.tflite*) format with the default preset optimization. TensorFlow Lite is optimized for running machine learning models on edge devices, making them suitable for deployment on resource-constrained devices. The conversion process preserves the model's architecture while enabling efficient execution on low-powered hardware.

G.1.4 Optimization using TensorFlow Lite Converter for TensorFlow Lite Micro

Further optimization of the model is performed using TensorFlow Lite's converter tools, which effectively substitute efficient TensorFlow OPS [154] for TFLite Micro as opposed to using the all-ops resolver. This step is essential for an efficient memory footprint.

G.1.5 Conversion to C Header with TensorFlow Operations

Once the optimized TensorFlow Lite model is ready, it is converted into a C header file using xxd [127] command line tool. This conversion process prepares the model for integration into the target MCU (ESP32-S3) firmware, enabling the device to interpret and run the model using TensorFlow Lite operations.

G.1.6 MCU Memory Allocation, Compilation, and Resource Requirements Evaluation

The next step involves allocating memory on the MCU (PlatformIO project with *esp32-s3-devkitc-1* environment) and compiling the appropriate inference code along with the model (C header file). We carefully allocate resources to ensure that the model fits within the ESP32-S3's memory limits (320 KiB SRAM). Compilation involves building the application with the necessary TensorFlow Lite Micro runtime (we utilized our developed PlatformIO library [174]) and the optimized model. After the build process, we keep track of Flash size to ensure that the model consumes a minimal amount of Flash memory.

G.1.7 Model Deployment and Metric Collection

Finally, the model is deployed onto the ESP32-S3. After deployment, we run the model on the testing split of the IMU sensor data (see Appendix F for sending testing split to MCU) and collect key performance metrics, including execution time, power consumption, classification accuracy, etc. These metrics are crucial for validating the system's effectiveness in practical applications. If the model performs within

the required constraints, the process concludes; otherwise, further optimization is conducted.

G.2 Experimental Results and Analysis

This section details the experimental methodology used to evaluate the performance of our composite HAR system on the resource-constrained device. The analysis covers model training, optimization techniques, and deployment on the ESP32-S3 MCU. The experiments were conducted to assess not only classification accuracy but also computational efficiency, including memory and Flash usage, execution time, and power consumption. We focused on balancing model performance with the inherent hardware limitations of the ESP32-S3, aiming for an optimized neural network solution suitable for real-world, low-power applications. Lastly, the experimental setup for this investigation is identical to the setup discussed in Chapter 6 Section 6.1.

G.2.1 The Proposed Neural Network Model

The model used in this study is a simple feedforward MLP, mainly designed for efficient inference of 21 activities on ESP32-S3 Dev Kit. The architecture comprises four fully connected layers, each contributing to the classification task based on preprocessed time-series sensor data.

Model Architecture

The MLP architecture is as follows:

1. **Input Layer:** The input layer takes a flattened vector of size 900, which corresponds to the time-series data (100 time steps $\times$ 9 features). This layer processes the sensor readings and prepares them for the following hidden layers.

2. **Dense Layer 1:** The first hidden layer consists of 128 neurons with a ReLU (Rectified Linear Unit) activation function, which introduces non-linearity to help the model learn complex patterns from the input features.

3. **Dense Layer 2:** The second hidden layer contains 64 neurons and also uses the ReLU activation function. This layer continues to refine the learned features, reducing the dimensionality and complexity of the data.

4. **Dense Layer 3:** The third hidden layer, comprising 32 neurons, follows the same ReLU activation, further distilling the information while minimizing the feature space.

5. **Output Layer:** The final layer is a softmax layer with 21 neurons (21 activities to predict). This layer outputs the probability distribution for each class, enabling multi-class classification.

The architecture is lightweight, focusing on maximizing classification performance (21 activities) while maintaining minimal computational and memory overhead, making it suitable for deployment on low-power, low-compute devices.

Table G.1: MLP model summary of trainable parameters and their respective memory usage.

Layer (Type)	Output Shape	Params
Dense (128)	(None, 128)	115,328
Dense_1 (64)	(None, 64)	8,256
Dense_2 (32)	(None, 32)	2,080
Dense_3 (21)	(None, 21)	693
Trainable Params	126,357	493.58 KiB
Non-trainable Params	0	0 B
Optimizer Params	252,716	987.18 KiB
Total Params	**379,073**	**1.45 MiB**

Table G.2: MLP Model testing (offline) and deployment (online) metrics side by side.

Metric	Accuracy	Precision	F1 Score
MLP (Offline)	89.711 ± 0.56%	90.032 ± 0.45%	89.786 ± 0.35
MLP (Deployed)	90%	89.79%	89.786%

FLOPs Calculation

Calculating FLOPs is crucial for understanding the computational load of the neural network model, especially when deploying on low-power devices like MCUs. FLOPs provide a rough estimate of how many operations the model requires to make predictions, helping to assess the energy efficiency and execution speed. Lower FLOP count leads to faster inference, lower memory usage, and reduced power consumption [89], [161], [193], all of which are vital for real-time applications on edge devices. The total number of FLOPs for a dense (fully connected) layer is calculated as:

$$\text{FLOPs}_{\text{Dense}} = 2 \times \text{input units} \times \text{output units} \tag{G.1}$$

Based on the architecture of our MLP model, the FLOPs for each layer are computed as follows:

- **First Dense Layer**: The input has 900 units, and the output has 128 units:

$$\text{FLOPs} = 2 \times 900 \times 128 = 230,400$$

- **Second Dense Layer**: The input has 128 units, and the output has 64 units:

$$\text{FLOPs} = 2 \times 128 \times 64 = 16,384$$

- **Third Dense Layer**: The input has 64 units, and the output has 32 units:

$$\text{FLOPs} = 2 \times 64 \times 32 = 4,096$$

- **Output Layer**: The input has 32 units, and the output has 21 units (the number of classes):

$$\text{FLOPs} = 2 \times 32 \times 21 = 1,344$$

Summing the FLOPs across all layers gives the total FLOPs for the MLP model:

$$\text{Total FLOPs} = 230,400 + 16,384 + 4,096 + 1,344 = 252,224$$

Thus, the total number of floating-point operations for the MLP model is **252,224**.

G.2.2 Model Testing and Deployment

After training, the model was tested to evaluate its generalization performance. The results indicated a respectable classification performance across various classes. The precision, recall, and F1 Score for the model are 90.032 ± 0.45%, 89.79 ± 0.25% and 89.786 ± 0.35% respectively, with the model achieving an overall accuracy of 89.711 ± 0.56% on the test split. Next, Table G.1 shows the parameter summary of our trained MLP model.

Figure G.2 shows the confusion matrix of our MLP model. The average classification accuracy for all activities was 89.8%. The activities classified with the highest accuracy were washing windows (99%), writing (98%), and eating (97%), which is understandable as these activities all involve repetitive motions with the dominant hand. The activities that caused the most confusion were folding clothes (80%), mopping (82%), and using the toilet (82%). Folding clothes was primarily misclassified as hanging out clothes (10%), which is reasonable as these two are similar activities. Consequently, mopping and using the toilet were misclassified as a wider range of activities, and we suspect taking a larger window would improve the performance.

Once trained and evaluated on a standard machine-learning platform, the model was deployed to an ESP32-S3 Dev Kit (target MCU). The deployment required conversion to the TensorFlow Lite format and then to TFLite Micro. Details regarding this are briefly discussed in Section G.1.5. Post-deployment, the model's performance was measured on the MCU, and it achieved 90% accuracy with an average precision of 89.79%, which is closely aligned with the performance achieved during offline testing. For ease of comparison, Table G.2 shows the metrics side by side for offline and online testing. In terms of hardware performance, the model exhibited an average execution time of 20.4 ms per inference cycle, which is critical for real-time applications. The memory footprint of the MLP model is just 5.8 KB with a total memory consumption of 27.988 KB. The Flash storage consumption is around 509 KB out of a total Flash of 819.953 KB. The per-layer and per-operation execution time, along with input/output sizes, are detailed in Table G.3.

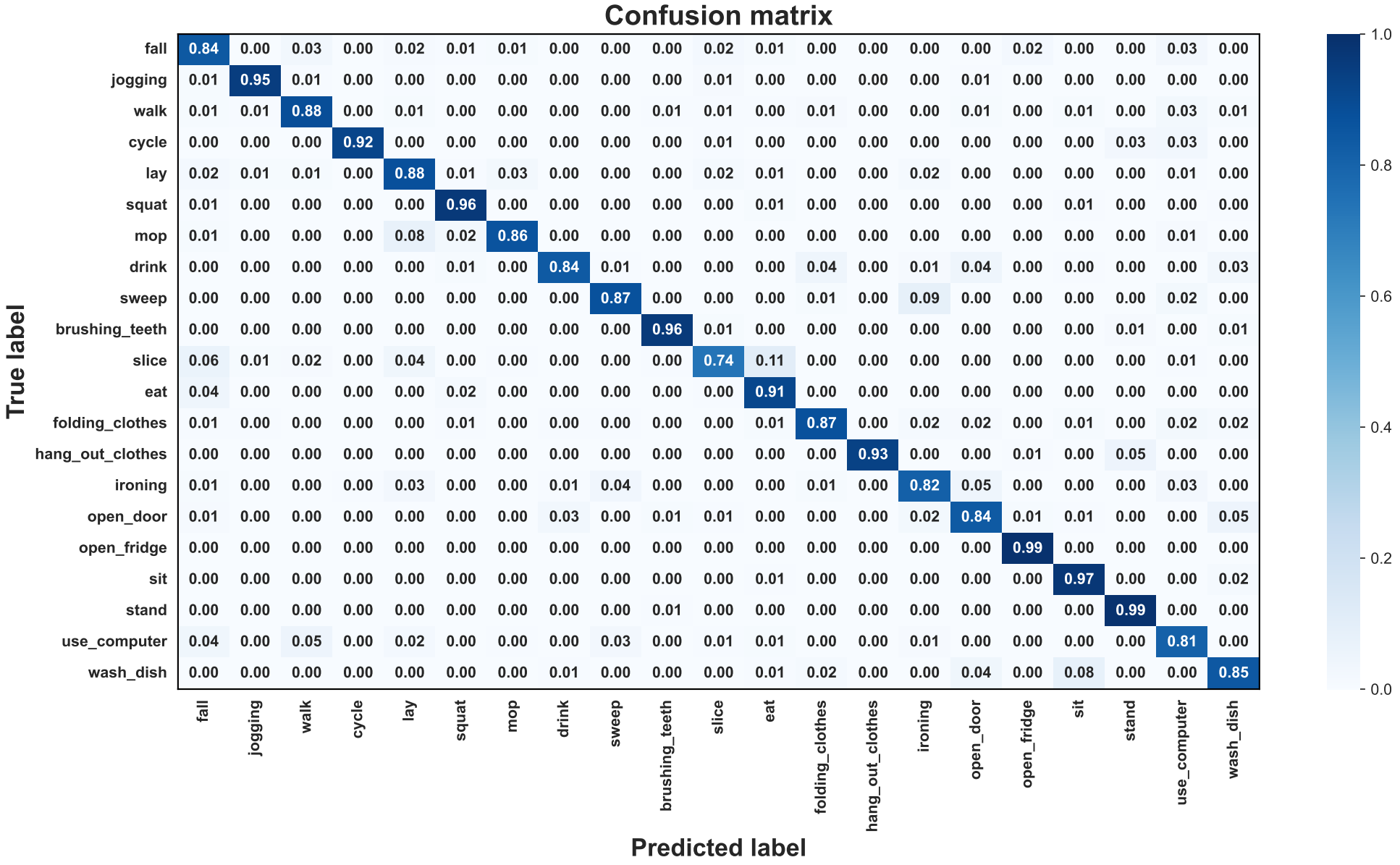


Figure G.2: Confusion matrix showing classification results on 21 activities using our MLP model with the IMU sensor positioned on the right wrist.

Table G.3: MLP model's per-layer and per-operation execution time on ESP32-S3 at constant 240 MHz respectively.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Dense (128 units)	FULLY_CONNECTED	18546	18546
Dense (64 units)	FULLY_CONNECTED	1346	1346
Dense (32 units)	FULLY_CONNECTED	347	347
Output (Softmax)	FULLY_CONNECTED	121	170
	SOFTMAX	49	
Total	**5.8 KB (SRAM)**		**20409 µs**

These figures remain well within acceptable limits, leaving an abundant amount of resources for developers to add other features on top of the existing MLP model. Additionally, the average power consumption was 235 mW at an operating frequency of 240 MHz. These results indicate that the model is well-suited for deployment in low-power, resource-constrained environments like wearable devices or embedded systems.

Table G.4: Comparison between non-quantized and 8-bit integer-quantized MLP models.

Method	Accuracy	F1 Score (macro)	Precision (macro)	MLP Size (KB)	Deployable
None[a]	90.15%	90.00%	89.79%	509.012	✔
Integer-8	90.01%	89.97%	89.92%	130.384	✔

[a] Here *None* represents the non-quantized default 32-bit floating point MLP model.

While the MLP model delivered inference times more than orders of magnitude faster than both CNN-S1 and CNN-S2, we pursued further optimization via post-training quantization to reduce latency and memory footprint. In this study we applied only 8-bit integer quantization, since TensorFlow Lite Micro and the ESP32-S3 hardware provide native support for 8-bit kernels. Mixed precision and 16-bit floating point quantization methods, although potentially beneficial for dynamic range, require custom operators that are not supported by TFLM and the ESP32-S3. Detailed justification for this design choice is already established in Section 6.2.

Table G.5: Quantized (8-bit integer) MLP model's per-layer and per-operation execution time on ESP32-S3 at constant 240 MHz respectively.

Layer	Operation	Operation Execution Time (µs)	Layer Total Execution Time (µs)
Dense (128 units)	FULLY_CONNECTED	10247	10247
Dense (64 units)	FULLY_CONNECTED	774	774
Dense (32 units)	FULLY_CONNECTED	207	207
Output (Softmax)	FULLY_CONNECTED	80	367
	SOFTMAX	287	
Total	**1.98 KB (SRAM)**		**11595 µs**

Following 8-bit integer quantization, the MLP model retained virtually the same predictive performance: it achieved 90.00% accuracy and an F1 score of 89.97%, corresponding to less than a 0.03% absolute drop. Model size was reduced by 75%, from 507.884 KB to 130.384 KB. On-device per-operation inference latency (see Table G.5) decreased by approximately 50%. SRAM usage fell from 5.8 KB to 1.98 KB, while Flash consumption dropped from 507.884 KB to 130.384 KB. Power draw remained constant at 235 mW, indicating that quantization did not increase dynamic energy requirements. These results confirm that 8-bit integer quantization is feasible for our MLP without compromising accuracy. Table G.4 presents the full comparison between the 32-bit floating point baseline and the 8-bit quantized model.

Table G.6: Comparison between our optimized MLP model and CAPPIMU author trained state-of-the-art models. The column *W/O* represents the comparison between *Window* and *Overlap* size in seconds utilized by the respective studies.

Study	Model	IMU Position	MFLOPs	Compute	F1 Score	W/O (s)
[132]	Deep-ConvLSTM	Right Wrist	606.03	N/A	82.8%	6/3
	FCN	Right Wrist	17.16		98.1%	
	DCNN	Right Wrist	507.14		96.6%	
	TPN	Right Arm	46.47		88.6%	
	Shallow-LSTMs	Left Tibia	531.80		79.1%	
	CNN-TA	Right Wrist	66.96		97.8%	
Ours	**MLP**	**Right Wrist**	**0.25**	**ESP32-S3**	**90.0%**	**2/1**

Table G.7: Quantized and non-quantized MLP models' total system resource consumption on ESP32-S3 and comparison with CNN-S1 and CNN-S2.

Model	**MFLOPs**	**Inference Latency (µs)**	**Total Memory (KB)**	**Total Flash (KB)**	**Power Consumption (mW)**
MLP	0.253	20,409	27.988	819.953	235
MLP-INT8		11,595[a]	21.468	449.405	
CNN-S1	6.14	1,836,271	91.636	552.649	210
CNN-S2	2.19	639,543	111.676	476.457	215

[a] Quantization latency excluded. If included, total latency = inference latency + quantization latency = 11 595 µs + 680 µs = 12 275 µs.

Table G.7 presents a detailed breakdown of end-to-end resource utilization on the ESP32-S3. It reports inference latency in microseconds for each model, peak SRAM usage, Flash memory footprint, and average power consumption. The 8-bit quantized MLP completes inference in 11,595 µs, which is more than 158 times faster than the 1,836,271 µs required by CNN-S1 and nearly fifty-five times faster than the 639,543 µs of CNN-S2. SRAM consumption for the quantized MLP is 1.98 KB compared with 91.636 KB for CNN-S1 and 111.676 KB for CNN-S2. Flash memory requirements shrink from 507.884 KB in the full-precision MLP to 130.384 KB after quantization, while CNN-S1 and CNN-S2 occupy 552.649 KB and 476.457 KB respectively. Power consumption remains at 235 mW for the quantized MLP and measures 210 mW and 215 mW for CNN-S1 and CNN-S2. These results demonstrate that our quantized MLP delivers superior resource efficiency in latency, memory, flash storage, and energy without sacrificing predictive performance.

Lastly, Table G.6 compares our optimized MLP against author-trained state-of-the-art models. Key metrics include the number of parameters, total MFLOPs, IMU sensor position, utilized window size, consequent window overlap size, and F1 Score. A deeper discussion of these trade-offs and additional per-operation metrics is provided in Chapter 8.