Imperial College of Science, Technology and Medicine
Dyson School of Design Engineering

Adaptive Machine Learning Techniques for Edge Computing

Yushan Huang

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy and Diploma of Imperial College London
April 2025

Abstract

Edge computing, by deploying computational capabilities to terminal devices near data sources (*e.g.*, sensors, wearables, microcontrollers), significantly reduces latency and privacy risks, providing critical support for real-time sensing scenarios such as healthcare monitoring and industrial IoT. However, the widespread distribution of IoT devices and their resource constraints pose multidimensional challenges to the adaptation of edge computing: these edge devices are typically deployed in complex environments, resulting in data that may be state-varying and noisy; the local tasks can also be heterogeneous, requiring personalized solutions; edge computing tasks are generally energy-sensitive and often constrained by limited computational capabilities.

This thesis explores to improve adaptation for edge computing from three perspectives: data, model, and system. At the data aspect, we propose an information-theory-based feature extraction framework for multivariate, multi-source, state-varying, and noisy time-series data, achieving highly robust and privacy-preserving edge data representation. At the model aspect, we aim to address data drift issues and achieve efficient on-device personalization. We first utilize mobile devices (such as Raspberry Pi) as the experimental devices, design a lightweight targeted fine-tuning method that dynamically maps data drift types to model blocks. We then utilize more resource-constrained MCUs as the experimental devices, propose MicroT, which employs techniques including self-supervised learning, knowledge distillation, and MCU-designed early-exiting. Finally, at the system aspect, we further explore the MCU with AI accelerators, establish an energy and power characterization for related operations involved in real-world applications.

Our research covers low-to-mid-performance edge devices, including mobile devices (*e.g.*, Raspberry Pi), ultra-resource-constrained MCUs, and MCUs equipped with AI accelerators. The experimental datasets include image and time-series data. The research theoretically and practically validates the necessity of improving the adaptation for edge computing from data, model, and system aspects, establishing a methodological foundation and technical benchmarks for building efficient, robust, and energy-sustainable edge computing.

Acknowledgements

The acknowledgements section of my thesis has always been the last part I write - for my bachelor's, master's, and now this doctoral dissertation. As I compose this final acknowledgements today, I'm once again struck by how time flies and memories fade. It turns out the acknowledgements are actually the hardest chapter to write!

Throughout this doctoral journey, I am deeply grateful to everyone who has supported, guided, and accompanied me. It is your help that enabled me to overcome challenges, grow continuously, and ultimately complete this dissertation.

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Professor Hamed Haddadi. You not only gave me tremendous academic freedom and trust to explore topics I was passionate about, but also provided invaluable guidance on research methodology and career development. Your rigorous scholarship and keen insight into cutting-edge technologies have profoundly influenced me, helping me grow into an independent researcher. If possible, I'd love to continue exploring delicious coffee with you in the future!

I also sincerely thank my secondary supervisor, Professor Payam Barnaghi. Your kindness and patient guidance have benefited me greatly in both research and life. You taught me to maintain an optimistic attitude in research, to face failures bravely, to stay calm in adversity, and most importantly, how to make the right choices and learn when to let go.

I owe a special thanks to Ms. Niu Liu, my elementary school teacher and the one who first opened the door to learning for me. Her patience and encouragement planted the seeds for my love of knowledge, which has carried me all the way to this point. Of course, there are also other teachers who have accompanied me on my journey of learning and growth: Ms. Feng Ma, Ms. Aili Jiang, Mr. Xushan Sun.

I'm grateful to the China Scholarship Council (CSC) and the Dyson School at Imperial College London for giving me the opportunity to pursue my PhD at Imperial.

To my parents - your selfless support and understanding were my greatest motivation to persevere. While you may no longer guide me academically, you remain my best teachers in life's philosophy! To my girlfriend Yongjie Jiang - your companionship, encouragement and patience

helped me maintain a positive mindset under pressure. Our four-year long-distance relationship, sustained through countless video calls and messages, has overcome numerous challenges as we supported and grew with each other! To my roommate Yuankai Ren - you've helped me see research and life's dilemmas from fresh perspectives! And of course, to my cat Xiaobai Jiang - your 'supervision' and 'mischief' added much joy to my research life. Every time I returned home exhausted, you were always the family member waiting for me!

My industrial internships provided crucial inspiration and practical significance for my research. My thanks go to Taesik Gong, SiYoung Jang, Fahim Kawsar, and Chulhong Min at Nokia Bell Labs (Cambridge). You helped me understand industrial research approaches and bridge academic research with practical applications. I still vividly remember our team-building activities and those evenings when I rode my bike home against the sunset - Cambridge remains an unforgettable place for me! Additionally, I thank Kleomenis Katevas, Stefanos Laskaridis, and Ali Shahin Shamsabadi at Brave Software. Our collaboration deepened my understanding of large language models and edge computing. Oh, and I still love those Hawaiian meals with extra salmon from the market best!

The research journey is never walked alone. My sincere thanks to my research companions: Ranya Alouf, Xavier Cadet, Yuchen Zhao, Josh Millar, Capstick Alexander, Francesca Palermo, Mengjia Niu, and Sandra Siby. Whether through stimulating academic discussions or daily encouragement and support, you all made this journey more fulfilling and enjoyable.

So many thanks cannot be fully expressed on these limited pages, yet every detail of these experiences remains vivid in my memory. While my doctoral studies are coming to an end, the knowledge, friendships, and growth from this journey will continue to accompany me moving forward. Once again, my deepest gratitude to all who have helped me along the way!

Oh, and let me thank myself too...

Statement of Originality

This is to certify that this thesis embodies the results of my own course of study and research. All else from other published or unpublished work is appropriately referenced.

Copyright Declaration

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-NonCommercial 4.0 International Licence (CC BY-NC).

Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose.

When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes.

Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Contents

1 Introduction **1**

1.1 Motivation and Objectives 1

1.2 Research Scope and Performance Metrics 3

1.3 Contributions and Outlines 4

1.4 Overview Literature Review 9

1.5 Publications 11

2 Data-Related Adaptations **13**

2.1 Introduction 14

2.2 Background 17

2.2.1 Clinical Decision Making 17

2.2.2 Traditional Approaches 18

2.2.3 Deep Learning Approaches 20

2.2.4 Information-Theory-Based Approaches 21

2.3 Methodology 23

2.3.1 Data Collection and Preprocessing 23

2.3.2 Feature Construction 25

2.4 Implementation & Evaluation Setup 35

2.4.1 Datasets 35

2.4.2 Models and performance metrics 39

2.5 Evaluation Results 42

2.5.1 Minder 42

2.5.2 ESRD 45

2.5.3 PTBDB 46

2.6 Discussions and Limitations 49

2.7 Conclusions 52

3 Model-Related Adaptations for Mobile Devices 54

3.1 Introduction 55

3.2 Background 57

3.2.1 Data Drift 57

3.2.2 Transfer Learning 59

3.2.3 Personalization 59

3.2.4 Efficient Learning 60

3.3 Methodology 61

3.3.1 Overview 61

3.3.2 Adaptive Personalization 62

3.3.3 Definition of Models and Sub-Blocks 67

3.4 Implementation & Evaluation Setup . . . 67
3.4.1 Models and Datasets . . . 68
3.4.2 Prototype and Performance Metrics . . . 69
3.5 Results . . . 70
3.5.1 Motivation Experiments . . . 70
3.5.2 Overall Performance . . . 71
3.5.3 Single-Batch & Multi-Batch Training . . . 72
3.5.4 Model Performance . . . 72
3.5.5 System Cost . . . 76
3.6 Discussions and Limitations . . . 76
3.7 Conclusions . . . 78

4 Model-Related Adaptations for Ultra-Resource-Constrained MCUs 79

4.1 Introduction . . . 80
4.2 Background . . . 84
4.3 Methodology . . . 89
4.3.1 Optimization Problem Formulation . . . 89
4.3.2 Overview . . . 89
4.3.3 Self-Supervised Knowledge Distillation . . . 94
4.3.4 Model Segmentation and Joint Training . . . 96
4.3.5 Classifier Training . . . 99
4.3.6 Stage-Decision . . . 101

4.3.7 Partial Quantization . . . 102
4.4 Implementation & Evaluation Setup . . . 104
4.4.1 MicroT Prototype . . . 105
4.4.2 Models and Datasets . . . 105
4.4.3 Performance Metrics . . . 107
4.4.4 Baselines . . . 108
4.5 Results . . . 109
4.5.1 Overall Performance . . . 109
4.5.2 MicroT Local Training *vs* Cloud-Level Model Training . . . 110
4.5.3 Model Performance . . . 111
4.5.4 Determination of Confidence Thresholds . . . 116
4.5.5 System Cost . . . 117
4.6 Ablation Study . . . 121
4.7 Discussions and Limitations . . . 126
4.8 Conclusions . . . 130

5 System-Related Adaptations **131**

5.1 Introduction . . . 132
5.2 Background . . . 134
5.2.1 Related Works . . . 134
5.2.2 MAX78000 . . . 135
5.2.3 Configuration Description . . . 137

5.3 Implementation & Evaluation Setup . . . 139
5.3.1 Application Pipeline . . . 139
5.3.2 Experimental Models and Datasets . . . 141
5.4 Experiments on Latency . . . 141
5.4.1 Relationship Between Latency and CNN Frequency . . . 142
5.4.2 Relationship Between Latency and System Frequency . . . 144
5.5 Experiments on Energy . . . 144
5.5.1 Micro Benchmark . . . 146
5.5.2 Macro Benchmark . . . 152
5.6 Discussions and Limitations . . . 160
5.7 Conclusions . . . 161

6 Summary and Future Outlook **162**

6.1 Conclusions and Contributions . . . 162
6.2 Discussion and Future Work . . . 165

Bibliography **167**

List of Tables

2.1 The average performance of the models for the Minder. 44

2.2 Comparison of ESRD classification results. 44

2.3 Comparison of PTBDB classification results. 47

2.4 Ablation results of each entropy feature from InTS. '1' means with this entropy feature, '0' means without this entropy feature. SE means Shannon entropy of Markov chains, ER means entropy rate of Markov chains, EP means entropy production of Markov chains, VNE means von Neumann entropy of Markov chains. 49

3.1 Noised block fine-tuning accuracy (best performance highlighted). 70

3.2 Overall results (accuracy, energy cost E, and saving rate ES compared with full tuning). $I-L$, $F-L$, $O-L$ denote input/feature/output-level drifts; $BlockAvg$ shows block-tuning average (excludes full model). Best accuracy for per drift are highlighted. 71

3.3 128 and 1 batch of training accuracy results (on the Living17 dataset with 10% training data). 72

3.4 The accuracy results of TBFT with various training size. Highlighted values indicate best block-tuning performance. 74

3.5 Time and energy costs of TBFT *vs.* full model fine-tuning. The *Energy-SavingRate* compares current energy cost to full-tuning baseline. 75

4.1 The experimental MCU boards. 105

4.2 Results of model performance with varying batch sizes. 110

4.3 Top-1 accuracy results of MicroT and the baselines. MicroT achieves the highest accuracy with two DNN architectures on three cross-domain datasets. 113

4.4 The median, stage-decision ratio, and model accuracy across varying sample sizes. N denotes the number of samples used to compute the median value, $Med.$ indicates the median value, *Ratio* signifies the stage-decision ratio, and *Acc.* reflects the model accuracy (%). 115

4.5 The energy costs (kJ) of the MCU local inference. MicroT achieves energy saving on two boards and two models. 120

4.6 Model performances (%) with various stage-decision ratios. 123

4.7 The energy costs (kJ) of the MCU local inference with various stage-decision ratios. 123

4.8 The impact of quantization on accuracy (%). 124

4.9 Ablation experiments on the local data drifts (Model: MCUNet, Dataset: Pet). 'Source Domain Acc' means the accuracy on the original Pet dataset using the local MCUNet; 'Target Domain Acc' means the accuracy on the Pet dataset with added noise; 'Fine-Tuned Acc' means the accuracy on the noisy Pet dataset after fine-tuning the classifier. 125

5.1 MAX78000 operation modes. 136

5.2 System configuration variables involved in the experiment. 136

5.3 Micro-level power (measured in mW) and energy (measured in mJ) assessment of computation operation. 149

5.4 Micro-level power (in mW) analysis: default idle implementation. 151

5.5 The power (in mW) of various idle implementations. 151

5.6 System configurations of the speech recognition experiments. 153

5.7 Results of the speech recognition experiments. 155

5.8 System configurations of the image segmentation experiments. 158

5.9 Results of the image segmentation experiments. 159

List of Figures

2.1 Overview of the IoT platform for healthcare monitoring. 14

2.2 Methods overview. 23

2.3 Example of Markov chain generated from state transition data. 24

2.4 Markov chain entropy rate calculation example. Rectangular nodes in ① indicate locations (states), with arrows showing transitions between them. Color coding represents starting locations: blue (bathroom), orange (bedroom), green (kitchen), black (lounge), and red (hallway). Transition probabilities (shown beside arrows) correspond to Table ② and Transition Matrix T ③. 27

2.5 For a 5-state Markov chain, von Neumann entropy calculation involves: *Spatial*: (1) Daily route frequencies $a_{i,j}^{d}$ (i, j: states, d: date); (2) Weekly aggregation; (3) Frequency matrix A; (4) Pearson matrix R; (5) Density matrix ρ by $f(\bullet)$; (6) von Neumann entropy by $f'(\bullet)$. And for *Temporal*: Replace frequencies with durations. 32

2.6 Examples of PLWDs with (a) clear routines and (b) fewer routines. The more routine-oriented participant (a) shows consistent daily patterns (red boxes) at fixed times. X-axis represents time of day, Y-axis represents colors indicate house locations. 37

2.7 ESRD visualization with sampling points (X-axis) and EEG signals (μV) (Y-axis). 38

2.8 PTBDB histogram color maps: (a) abnormal and (b) normal cases. X-axis shows sampling points, Y-axis shows normalized heartbeat values. 38

2.9 Evaluation results on the Minder dataset (x-axis: evaluation metrics) 43

2.10 The ROC-AUC curves of the Minder dataset. 43

2.11 Feature selection results for ESRD: (left) Pearson correlation matrices, (right) mutual information with selected (red) and unselected (blue) features. 45

2.12 ESRD results: (a) Evaluation metrics (recall, F1, accuracy, ROC-AUC); (b) Model structure comparison (baseline vs. InTS/entropy). Due to pre-processed low-noise data, all models achieve ideal AUC-ROC performance. 46

2.13 PTBDB feature selection results with Pearson correlation matrices (left) and mutual information (right, red=selected, blue=unselected). 47

2.14 PTBDB results: (a) evaluation metrics, (b) baseline vs. InTS/entropy model structures. With pre-processed low-noise data, all models achieve ideal AUC-ROC performance. 48

3.1 TBFT overview. 61

3.2 The ResNet-26 model. 68

3.3 The accuracy results of TBFT on Cifar10-C datasets with different training sizes. Solid/dashed lines show *BlockAvg*/full-tuning baselines. Specific values are shown in Table 3.4. 73

3.4 The accuracy results of TBFT on Living17 datasets with different training sizes. Specific values are shown in Table 3.4. 73

3.5 The accuracy results of TBFT on Cifar-Flip datasets with different training sizes. Specific values are shown in Table 3.4. 74

4.1 System overview of MicroT. 90

4.2 MicroT prototype overview. 104

4.3 Main structure of ProxylessNAS and MCUNet. 105

4.4 The overall comparison between MicroT (stage-decision 0.5) and the baselines. . 109

4.5 The segmentation fused scores for MCUNet and ProxylessNAS, with the X-axis indicating the part model up to a specific module and the Y-axis showing the fused score. 112

4.6 The performance enhancement achieved through SSKD joint training for the part and full models, with dashed connecting lines highlighting the variations in model performance. 113

4.7 Comparison between stage-decision ratio determined by median or random selection. 117

4.8 Energy cost of local training. 119

4.9 The model accuracy between SSL and SSKD. 121

4.10 The model accuracy between segmentation by global/local points. 122

5.1 Overview of MAX78000. 135

5.2 Relationship between latency and CNN frequency. 143

5.3 Relationship between latency and system frequency. 145

5.4 Micro-level assessment: sensing operation. Bold fonts indicate configurations with greater energy efficiency. 147

5.5 Micro-level evaluation: inference operation. 148

Abbreviations

IoT	Internet of Things
ML	Machine Learning
DL	Deep Learning
TL	Transfer Learning
TTL	Traditional Transfer Learning
DNN	Deep Neural Network
NN	Neural Network
RNN	Recurrent Neural Networks
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
MLP	Multi-Layer Perceptron
LR	Logistic Regression
SVM	Support Vector Machines
pp	Percentage Points
PIR	Passive Infrared
VNE	Von Neumann Entropy
MI	Mutual Information
PCC	Pearson Correlation Coefficient
PLWD	People Living with Dementia
DWT	Discrete Wavelet Transform
EEG	Electroencephalogram
ECG	Electrocardiogram

SSL	Self-Supervised Learning
KD	Knowledge Distillation
KNN	K-Nearest Neighbors
EPR	Entropy Production Rate
NEEP	Neural Estimator for Entropy Production
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
LPM	Low-Power Mode
UPM	Ultra-Low-Power Mode
SIMO	Single-Input Multiple-Output
WUT	Wake-Up Timer
LPT	Low-Power Timer
TinyTL	Tiny Machine Learning
MCU	Microcontroller Unit
LLM	Large Language Model
NAS	Neural Architecture Search
ES	Energy saving rate
EB	Energy cost of block-based fine-tuning
EF	Energy cost of full fine-tuning
E	Energy cost
FC layer	Fully connected layer
SYS_CLK	System clock
SYS_DIV	System divider
CNN_DIV	The divider of MAX78000 CNN quadrant
OPE_Freq	Operating frequency
Acc	Accuracy
Med	Median value
Avg	Average value

Mathematical Symbols

H	Shannon Entropy
x, X	States of a Markov chain
x_{src}, X_{src}	Input of the source model
y_{src}, Y_{src}	Output of the source model
x_{trg}, X_{trg}	Input of the target model
y_{trg}, Y_{trg}	Output of the target model
y, Y	Labels
s, S	Trajectory of a Markov chain
n, N, l, L	Count (context-dependent)
P	Event Probability
p	Probability distribution
P_{src}	Event probability of the source model
P_{trg}	Event probability of the target model
i, j, t, k, m	Index (context-dependent)
T	Transition matrix
π	Stationary distribution
TW	Time window
SP	Start point of a sequence
θ	Model trainable parameters
h	Trainable models/modules/blocks
EPR	Entropy production rate of a Markov chain
VNE	von Neumann entropy of a Markov chain

ξ	Entropy rate of a Markov chain
A	Activity frequency matrix
a	Elements of the activity frequency matrix
R	Pearson correlation matrix
I	Identity matrix
M	Invertible matrix
$\boldsymbol{\rho}$	Density operator
MI	Mutual Information
PCC	Pearson Correlation Coefficient
e	Entropy features
f	Functions
pp	Percentage points
ϵ	Convergence threshold
B	Sub-block/module or selected blocks/modules of a model
$\mathcal{L}$	Loss function
$\mathcal{L}_{SSL}$	Loss function of self-supervised learning
$\mathcal{L}_{JT}$	Loss function of joint training
$\mathcal{L}_{MSE}$	Mean squared error loss function
$\mathcal{L}_{CE}$	Cross entropy loss function
$\mathcal{M}$	A method or algorithm
D_{KL}	Kullback-Leibler divergence
ϕ	Activation function
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution, with mean μ and covariance matrix Σ
E	Energy cost
F_{sys}	System frequency
o	Output logits
o_g	Output logits of the guide model
o_e	Output logits of the exploration model
o_f	Output logits of the full model

o_p	Output logits of the part model
o_t	Output logits of the teacher model
τ	Temperature scalars
α	Weight factor
Z	Model segmentation fused score
$R_A^{\mathbf{norm}}$	Normalized ratio of accuracy loss
$R_M^{\mathbf{norm}}$	Normalized MAC reduction ratio
G	Normalized gain ratio
Acc	Accuracy
Med	Median value
Avg	Average value
$W/$	With ..
W/O	Without ..

Chapter 1

Introduction

1.1 Motivation and Objectives

Edge Computing is a device-centric computing paradigm whose core objective is to deploy data processing, storage, and computational capabilities directly at the data source or end devices, rather than always relying on remote cloud devices [1, 2]. By executing computational tasks locally on devices, edge computing significantly reduces data transmission latency, minimizes dependency on network bandwidth, and enhances system real-time performance, energy efficiency, and privacy protection. Edge computing is particularly well-suited for scenarios with strict requirements for low latency, high reliability, and data localization, such as smartphones [3], smart home [4], industrial Internet of Things (IoTs) [1], autonomous vehicles [5], augmented reality (AR)/virtual reality (VR) [6], and medical devices [7]. Compared to traditional cloud computing, edge computing places greater emphasis on efficient utilization of computational resources and task execution at the device level.

Motivating Examples. The smart healthcare system is a critical application domain of edge computing, integrating IoT devices, machine learning, and edge computing to enhance medical decision support and health monitoring. However, it faces significant adaptation challenges in practical applications: the ability to effectively address the complexity of data, model, and system levels in diverse and dynamic scenarios. Specifically: at the data level, it needs to handle state-varying and noisy data [8, 9]; at the model level, it needs to satisfy various local

tasks or conditions with efficient methods [10, 11]; at the system level, it needs to optimize the system/hardware configuration to achieve lower energy consumption or power usage while maximizing model performance ($e.g., classificationaccuracy$), thereby extending operational lifespan or standby time. Motivations and objectives are detailed in three aspects as follows:

1. Data-Related Adaptations

Due to the current strain on medical resources, mild to moderate patients commonly adopt a conservative home-based treatment model. Decentralized smart home monitoring systems are typically equipped with multiple sensors and primarily rely on two types of signals to achieve early warnings: (1) Discrete state-transition signals, such as events triggered by infrared position sensors, can be used to detect symptoms like wandering or pacing in dementia patients; (2) Continuous physiological signals, such as EEG/ECG collected by wearable devices to detect cardiac and neurological disorders. These data exhibit typical characteristics of being multivariate and multi-source, dynamic, and noisy, specifically manifested as [12, 13, 14]: (1) Environmental interference ($e.g.$, activities of other household members triggering sensors), device noise ($e.g.$, ECG/EEG signal drift due to improper wearing), and physiological noise ($e.g.$, baseline drift, electromyographic interference [15, 16]); (2) Pronounced signal alterations caused by pathological features or abnormal behaviors. These characteristics lead to excessively high false alarm rates in traditional detection methods based on fixed thresholds or averages, urgently necessitating the development of robust, general, and adaptive edge computing methods to achieve high-precision and efficient health monitoring [17].

2. Model-Related Adaptations

Model-related adaptations primarily address data drift and model personalization issues [18]. For instance, smart healthcare systems often involve image data classification to enable disease categorization and diagnosis. Specifically, machine learning models in smart healthcare systems are typically trained centrally and deployed across various local environments. Unlike the data used during central training, local data may have data drifts. For example, image quality changes ($e.g.$, persistent fogging or color temperature discrepancies) caused by hardware differences or variations in imaging environments. Another example is that, variations in feature

distributions for the same disease due to regional differences among patients. These data drifts can lead to a decline in model performance (*e.g.*, accuracy) when the source model is deployed locally. We need to explore how to achieve efficient model personalization on edge devices with limited computational resources.

3. System-Related Adaptations

Smart healthcare systems involve numerous sensors and IoT devices, along with on-device signal processing and computational inference. Although current edge devices can perform inference for some small-scale models, further reducing energy consumption, power, and latency remains a critical challenge. System-related adaptations primarily focus on optimizing system configurations to meet the performance requirements of diverse application scenarios, such as energy and power consumption [19]. For example, reducing the power and energy consumption of wearable devices can significantly extend their operational lifespan and reduce the frequency of recharging or battery replacement. Lower latency can also enable faster auxiliary medical decision-making. This motivates our research to explore optimal system configuration methods, ensuring computational performance (*e.g.*, accuracy) while minimizing energy or power consumption.

Although smart healthcare systems serve as the motivating example, the challenges of data-, model-, and system-related adaptations are generally exist across edge computing applications, including cross-border wildlife monitoring systems [20], wearable devices [21], industrial inspection systems [22]. Therefore, the experimental datasets and scenarios in this thesis are not limited to medical and health-care.

1.2 Research Scope and Performance Metrics

Definition of Edge Computing. In this thesis, edge computing refers to a technical paradigm where machine learning tasks are executed on terminal devices close to the data source. It generally has limited computational capability, such as memory footprint, power, and energy consumption. This thesis focuses on mid-to-low-level edge devices: (1) Mobile devices (*e.g.*, Raspberry Pi with 1-4GB memory, 1-2GHz CPU), representing scenarios requiring moderate

computational complexity; (2) Basic MCUs (*e.g.*, STM32, <1MB memory), representing ultra-resource-constrained scenarios; (3) MCUs with AI accelerators (*e.g.*, MAX78000[23]), optimized for low-power (mW-level) inference.

Evaluation Metrics. The evaluation metrics include both model performance and system cost. Classification accuracy serves as the primary performance metric across all chapters, with Chapter 2 augmenting this with medical-specific evaluations (F1-score, recall, and ROC-AUC) to ensure clinical reliability in neurological diagnosis. System cost includes power consumption, energy cost, and memory footprint. We evaluated our methods on various edge devices, which are carefully selected to best represent each chapter's research focus: Chapter 3 - mobile devices (refer to Raspberry Pi in this chapter); Chapter 4 - MCU; Chapter 5 - MCUs with AI accelerators. For Chapter 2's medical data analysis, where **UK data protection regulations (including GDPR and the Data Protection Act 2018 [24])** precluded edge deployment during initial development, we used model parameter count as a hardware-agnostic proxy for system cost.

1.3 Contributions and Outlines

This thesis explores the adaptation of machine learning in edge computing through four core research chapters (Chapters 2-5). Each chapter focuses on different technical aspects: at the data level (Chapter 2), we explore how to handle state-varying and noisy data; at the model level (Chapters 3-4), we explore how to efficiently address data drift on resource-constrained devices and achieve personalized fine-tuning of local models; at the system level (Chapter 5), we explore how to reduce system and power consumption through system configuration, established a energy/power characterization. Due to the different research perspectives of each chapter, the specific application contexts, experimental devices, and experimental datasets are different, but all are strictly tailored to the corresponding research problems. Detailed explanations, along with the background and contributions of each chapter, are provided below.

1. Data-Related Adaptations: Entropy-Based Feature Analysis for Time-Series Pattern Recognition in Neurological Conditions

Chapter 2 - This chapter focuses on enhancing the adaptation of edge computing at the data level, specifically addressing how to extract more effective features from state-varying and noise data. We utilize medical time-series data as the research context. This healthcare platform primarily collects state-varying and noise EEG/ECG signals and infrared-triggered sensor signals. Notably, this chapter only uses time-series data rather than image data due to the platform's privacy protection requirements—image data risks leaking identity information and residential environment details. We propose InTS, a general framework for time-series analysis based on information theory.

Contributions.

• InTS adopts a unified information-theoretic framework to process neural time-series data, distinguishing it into state-transition signals (*e.g.*, infrared sensor trigger data) and non-state-transition signals (*e.g.*, EEG/ECG) [25].

• For state-transition data, we introduce four new feature extraction methods based on Markov chains and entropy.

• For non-state-transition data, we integrate various existing entropy-based features, and develop a dual-stage feature selection mechanism.

• We validated InTS on a dementia care dataset (collected by the home monitoring platform), the Epileptic Seizure Recognition Dataset, and the PTB Diagnostic ECG Database [26, 27] [1].

• For state-transition data, we use average activity frequency and duration as baseline features and compare performance across logistic regression (LR), support vector machine (SVM), multi-layer perceptron (MLP), and long short-term memory (LSTM) models. Compared to baselines, InTS improves accuracy, recall, and F1 score by an average of 13.08 percentage points (pp), 10.80 pp, and 7.88 pp, respectively.

• For non-state-transition data, we compare against baseline models with automatic feature extraction, including CNN, LSTM, and CNN-LSTM. Across all experiments, compared to

[1]Due to the medical data access restrictions of the platform, we were unable to obtain the collected EEG and ECG signals, so we used two public medical datasets (the last two datasets).

baseline models, InTS improves accuracy, recall, and F1 score by an average of 2.52 pp, 2.42 pp, and 2.32 pp, respectively. InTS also reduces the model parameters by 3.10×.

2. Model-Related Adaptations: Energy-Efficient Personalization for Mobile Devices

Chapter 3 - This chapter focuses on enhancing the adaptation at the model level, primarily addressing data drift to improve local model performance (*e.g.*, accuracy). This chapter utilized a mobile device (Raspberry Pi) as the experimental platform. We explore the difference in model accuracy between fine-tuning the blocks/modules of various depths for different types of data drift. We propose Target Block Fine-Tuning (TBFT), a low-energy adaptive personalization framework specifically designed for mobile devices.

Contributions.

• We categorized data drift into three types: input-, feature-, and output-level, and provided mathematical definitions.

• From a mathematical perspective, we derived that input-, feature-, and output-level drifts are more closely related to the model's front, middle, and rear modules/blocks, respectively.

We validated the effectiveness of TBFT on the CIFAR series and BREED/Living17 datasets, using small-scale training sets (10%, 20%, 30% of full training datasets) to simulate the limited local data.

We used 'independently fine-tuning modules/blocks at different depths of the model and averaging their accuracies' as the baseline (named *BlockAvg*). Compared to this, TBFT improves accuracy by an average of 15.30% across training sets of varying sizes.

• Compared to full-model fine-tuning, TBFT saves an average of 41.57% in energy consumption.

3. Model-Related Adaptations: Energy-Efficient Personalization for Ultra-Resource-Constrained MCUs

This chapter aligns with the core objective of Chapter 3, both focusing on enhancing model-level

adaptation in edge computing environments to address data drift issues. However, it introduces stricter constraints in application scenarios: (1) The experimental platform changes from mobile devices (Raspberry Pi) in Chapter 3 to MCUs with more severe resource constraints, typically having RAM capacities below 1MB; (2) An additional assumption is introduced that 'the cloud cannot access local data', making it challenging to directly leverage cloud computing resources and utilize the personalization methods for mobile devices. We continue to use the public image dataset ImageNet but excludes the CIFAR series due to its low resolution and similar feature distribution with ImageNet. Furthermore, we carefully selected three other datasets with significant feature distribution differences from ImageNet for experiments. We propose MicroT, an efficient and low-energy on-device personalization framework tailored for MCUs.

Contributions.

• MicroT follows the traditional transfer learning (TTL) paradigm, splitting the model into a feature extractor and a classifier, and training only the lightweight classifier on local devices.

• MicroT leverages SSL to train a robust and general feature extractor in the cloud to address the limited learning space caused by fine-tuning only the classifier and reduce the burden of local fine-tuning.

• MicroT adopts a two-stage knowledge distillation to reduce the size of the feature extractor while keep its generality.

• MicroT introduces an MCU-tailored early-exit strategy to reduce the energy of model inference.

• MicroT is evaluated on two models, three datasets, and two MCU development boards. Experimental results show that MicroT outperforms traditional transfer learning (TTL) and two SOTA methods in accuracy, achieving improvements of 7.01-11.60%, 2.29-5.73 and 2.12-3.68%, respectively. During local training, MicroT reduces energy consumption by 2.03-2.28× compared to two SOTA methods; during local inference, it achieves a 14.17% reduction in energy consumption compared to TTL.

5. System-Related Adaptations: Energy Characterization and Optimization for

Tiny AI Accelerators on Microcontrollers

This chapter focuses on optimizing energy cost from the perspective of system configuration. We mainly focus on inference rather than model training, as inference occupies the majority of time in edge computing applications. We shift our experimental devices to MCUs with AI accelerator due to its rapid development. We select the MAX78000 as the typical platform for our study. We conduct an in-depth energy and power characterization for MAX78000 [2].

Contributions.

• We conducted an end-to-end pipeline analysis of latency, power, and energy consumption. The pipeline covers operations including sensing, inference, computation, memory I/O, and idle.

• We explore the relationship between system frequency, CNN accelerator frequency, and latency of model inference.

• We conduct micro-level evaluation, where the energy and power for each operation in the application pipeline is independently evaluated. The evaluation considers system configuration from core configuration control, frequency management, and quadrant-based control, with eight system configuration options tested.

• We treat the application pipeline as a whole and conduct macro-level evaluation, assessing the impact of changing single system configuration on the overall energy cost.

• We find that dynamic configurations for different operations can improve energy efficiency. Compared to the default static configuration of the MAX78000, dynamic configurations reduced total energy consumption by 3.19×.

• We explore and optimize idle implementation methods to reduce system cost during idle. Compared to the default idle implementation, we find that ultra-low-power mode (UPM) reduced idle power consumption from 13.11mW to just 2.02mW.

[2]Models and datasets are available at:
https://github.com/analogdevicesinc/msdk/tree/main/Examples/MAX78000

1.4 Overview Literature Review

In this section, we briefly introduce the related work covered in each research chapter, with more detailed literature reviews provided in the respective chapters.

Data-Related Adaptations. Chapter 2 primarily explores how to achieve efficient and effective feature extraction for state-varying and noisy medical time-series data in decentralized healthcare monitoring platforms. Traditional signal processing techniques like Fourier transforms and autoregressive models have provided foundational approaches for neurological time-series analysis, yet their reliance on stationarity assumptions and linear frameworks limits their effectiveness for non-stationary physiological signals [28, 29]. While wavelet transforms and HMMs offered improved handling of transient features and state transitions [30, 31], these methods often require expert parameter tuning. Deep learning approaches, particularly CNNs and LSTMs, have demonstrated superior capability in capturing spatiotemporal patterns [32, 33], though concerns remain regarding computational demands and interpretability [34]. Information-theoretic methods employing entropy metrics have emerged as a promising alternative. It can measure the uncertainty of information, providing powerful feature extraction and certain interpretability [35, 36].

Model-Related Adaptations. In our research on model-related adaptation, we initially used mobile devices (*e.g.*, Raspberry Pi) as the experimental platform to explore efficient fine-tuning strategies for different types of data drift. The primary techniques related to the block-selected fine-tuning proposed in Chapter 3 include full-model and last-layer fine-tuning. While full-model fine-tuning maximizes performance (*e.g.*, classification accuracy), its high computational cost severely limits its applicability on resource-constrained devices [37]. Conversely, adjusting only the final classifier layer (*e.g.*, fully connected layer) [38] reduces computational overhead but often leads to performance degradation due to limited learning space. In Chapter 4, we shifted to extremely resource-constrained MCUs as the experimental platform to further investigate model personalization. The conflict between the computational resources required for model personalization and the limited capabilities of devices is even more pronounced on MCUs. Federated learning, multi-task learning, and zero-shot learning leverage cloud computing re-

sources, but they either require stable network communication or need some prior knowledge of local data (*e.g.*, data distribution) [39, 40, 41, 42]. Some researchers have proposed purely on-device personalization solutions. For instance, SparseUpdate reduces computation by selectively updating layers [43], and TinyTL utilizes lightweight plug-in modules [44]. However, these methods still face drawbacks in terms of energy consumption, memory, power, and latency due to their extra parameter selection and training added modules (detailed in Section. 4.2). Additionally, the extreme resource constraints of MCUs force researchers to adopt specialized optimization strategies, such as compromising on low-resolution datasets [45] or simplifying model structures to the level of simple models [46]. More critically, the lack of operating system support on MCUs requires models to be deployed directly to bare-metal environments using C/C++, in stark contrast to the flexible Python-based development ecosystem on mobile devices [47]. This makes methods validated on resource-rich devices like smartphones [48] often challenging to directly transfer to true MCU scenarios.

System-Related Adaptations. This chapter primarily investigates how to optimize the system cost (*e.g.*, energy and power) of model inference from the perspective of system configuration. In terms of system configuration optimization, researchers have proposed various dynamic adjustment strategies: Lin et al. achieved power control through real-time frequency adjustments [49], Rotteleuthner et al. developed a modular dynamic voltage and frequency scaling scheme [50], and Qian et al. focused on configuration optimization during device idle operations [51]. These studies demonstrate that system configuration adjustments can reduce energy consumption in edge devices. In the field of energy benchmarking, current research mainly focuses on mobile devices and MCUs. For instance, Laskaridis et al. established methods for quantifying the energy characteristics of large language models on mobile devices [52], and Sah et al. developed benchmarking tools for MCU-based object detection models [53]. For MCUs equipped with AI accelerators, benchmarking efforts are just beginning. Some studies, using the MAX78000 as an example, have established thermodynamic and energy consumption benchmarks for different ML model inferences [54, 55, 56]. However, these benchmarking efforts are typically limited to the model inference operation, overlooking other application pipeline operations, such as sensing and memory I/O. Additionally, they often neglect the importance

of system configuration, focusing solely on differences across ML models.

1.5 Publications

This research accomplished from 2021 to 2025 led to 6 published as the first author, including 1 best paper award. The related research outcomes have been also presented at 7 international and national conferences. In addition, there are 2 other papers related to the topic during PhD (labeled with $*$):

Yushan Huang, Taesik Gong, SiYoung Jang, Fahim Kawsar, Chulhong Min. 'Energy Characterization of Tiny AI Accelerator-Equipped Microcontrollers'. Accepted to The Second International Workshop on Human-centered Sensing, Networking, and Multi-Device Systems HumanSys'24, Co-located with SenSys'24 and BuildSys'24. **Best Paper Award**.

Yushan Huang, Ranya Aloufi, Xavier Cadet, Yuchen Zhao, Payam Barnaghi, Hamed Haddadi. 'Low-Energy and Adaptive Models for MCUs'. Accepted to The Ninth ACM/IEEE Symposium on Edge Computing (SEC 2024).

Yushan Huang, Josh Millar, Yuxuan Long, Yuchen Zhao, Hamed Haddadi. 'Towards Low-Energy Adaptive Personalization for Resource-Constrained Devices'. Accepted to The 4th Workshop on Machine Learning and Systems EuroMLSys 2024, co-located with EuroSys'24.

Yushan Huang, Yuchen Zhao, Capstick Alexander, Francesca Palermo, Hamed Haddadi, Payam Barnaghi. 'Analyzing entropy features in time-series data for pattern recognition in neurological conditions'. Accepted to Artificial Intelligence in Medicine, Elsevier, 2024.

Yushan Huang, Hamed Haddadi. 'Poster: Towards Machine Learning and Inference for Resource-constrained MCUs'. Accepted to The 21st ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2023).

Yushan Huang, Yuchen Zhao, Hamed Haddadi, Payam Barnaghi. 'Using Entropy Measures for Monitoring the Evolution of Activity Patterns'. Accepted to The 2022 IEEE 8th World Forum on Internet of Things (WF-IoT 2022).

Josh Millar, **Yushan Huang**, Anil Madhavapeddy, Sarab Sethi, Hamed Haddadi. 'Benchmarking Ultra-Low-Power μNPUs'. Under reviewing by The 31st Annual International Conference on Mobile Computing and Networking (MobiCom 2025). *

Shuo Zhao, Yvette Sum, Cenxi Tian, Maya Thabel Herron, **Yushan Huang**. 'Loose-Fitting E-Textile Sensing System Design and System Establishment Review'. In submission of Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT/Ubicomp). *

Chapter 2

Data-Related Adaptations

This chapter explores data-related adaptation in edge computing. We take a smart healthcare system as an example, since this study is driven by a decentralized home monitoring platform and the medical field imposes strict requirements on data and features, which is suitable to explore data-related adaptation. The collected data from the platform show multivariate, multi-source, and highly noisy characteristics. Conventional feature extraction methods suffer from limitations when handling these data, while neural network-based approaches introduce computational overhead. To address these challenges, we propose an information theory-driven framework for neurological time-series analysis. We categorize the collected data into state-transition (*e.g.*, infrared-triggered sensor signals) and non-state-transition (*e.g.*, EEG/ECG) types. For state-transition data, we develop four entropy- and Markov chains-based features. For non-state-transition data, we first compute a series of entropy variants (*e.g.*, approximate entropy, dispersion entropy) and then implement a dual-stage feature selection to retain informative, relevant, and low-collinearity features. For state-transition data, we adopt average activity frequency and duration as baseline features, evaluating performance across LR, SVM, MLP, and LSTM models. Compared to these baselines, InTS improves accuracy, recall, and F1 score by an average of 13.08 pp, 10.80 pp, and 7.88 pp, respectively. For non-state-transition data, we utilize features derived by InTS in conjunction with an MLP model. Compared against automatic feature extraction (CNN, LSTM, and CNN-LSTM), InTS improves accuracy, recall, and F1 score by an average of 2.52 pp, 2.42 pp, and 2.32 pp, respectively. Additionally, InTS reduces model parameters by 3.10×.

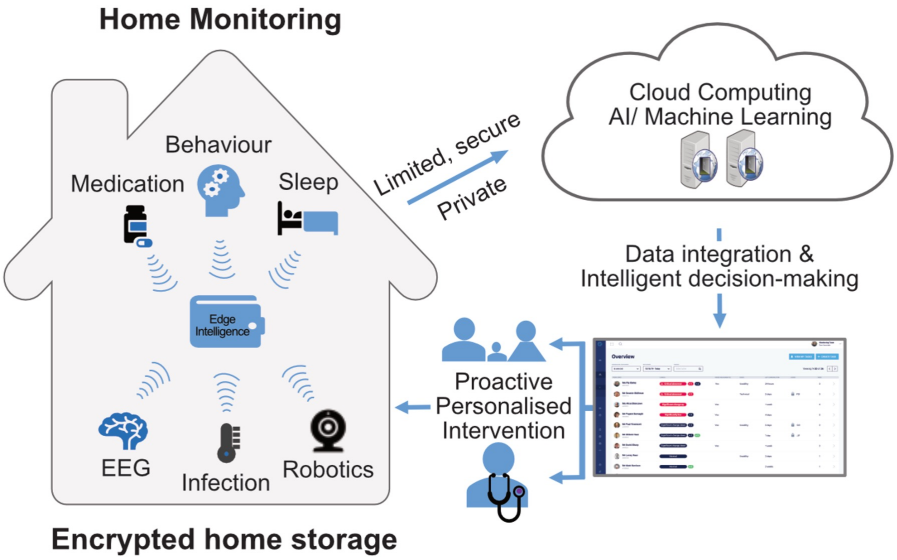


Figure 2.1: Overview of the IoT platform for healthcare monitoring.

2.1 Introduction

With the rapid development of IoT and AI technologies, smart healthcare systems are gaining increasing attention [57, 58]. Many patients with mild to moderate conditions prefer conservative treatment at home to maintain their daily lives. As shown in Fig. 2.1, smart healthcare systems typically integrate various sensors, such as infrared-triggered sensors, EEG, and ECG sensors [59, 60]. Through edge-cloud collaboration, the system can monitor patients' daily activities in real time, issuing alerts or assisting with medical decisions when necessary.

The time-series data in the smart healthcare systems can be commonly classified into two categories: state-transition signals and non-state-transition signals [25]. State-transition signals encapsulate discernible shifts in a participant's condition, such as those observed in home monitoring systems for dementia or traumatic brain injury (TBI) patients, where indoor infrared sensors log pivotal events like entering a room or operating a microwave, marking explicit state changes. Non-state-transition signals, exemplified by EEG and ECG, are typically gathered through continuous sampling at fixed or elevated frequencies, forming uninterrupted waveform datasets. These two types of data typically exhibit multivariate, multi-source, state-varying, and noisy characteristics. For instance, EEG and ECG signals may show significant changes in amplitude, frequency distribution, or waveform morphology under different physiological or pathological conditions, reflecting their multivariate and state-varying nature [61]. Signals col-

lected by infrared sensors are multi-source, as these sensors are typically deployed at various locations within a home. Moreover, when patients exhibit abnormal behaviors (*e.g.*, wandering or pacing), the triggering patterns of infrared sensors display diverse fluctuations and variations, also demonstrating multivariate and state-varying properties. Both types of signals are susceptible to noise, such as measurement biases from improperly worn EEG or ECG sensors, physiological noise in EEG/ECG (*e.g.*, baseline drift, electromyographic interference [15, 16]), and unintended triggering of infrared sensors by visitors or cohabitants [62].

While conventional linear techniques—such as Fourier transforms [63] and autoregressive models [64]—retain theoretical utility and practical relevance under simplified conditions, their efficacy tends to falter when confronted with datasets marked by pronounced multivariate complexity, multi-source heterogeneity, state variability, and noise. Consequently, contemporary research has increasingly explored deep learning methodologies, including RNN and LSTM networks, which excel at discerning intricate temporal dependencies and evolving patterns within such datasets [65, 66]. As well as information theory-driven techniques, such as entropy and mutual information, have gained attention for their capacity to quantify data complexity and uncertainty, yielding relative interpretable features [67, 68].

Despite the notable advances achieved through the application of information-theory-based and neural network-driven methodologies in analyzing time-series data within neurological and medical contexts, these approaches continue to face substantial challenges. Primarily, a significant portion of existing techniques is inherently task-specific, encompassing tailored processes such as data cleaning, signal processing, feature engineering, and model selection. Although effective within their designated domains, this specificity restrict their broader applicability and capacity for systematic deployment across diverse scenarios, hindering their scalability. Additionally, certain methodologies exhibit pronounced sensitivity to the quality of raw data, posing difficulties in sustaining consistent and robust performance (*e.g.*, classification accuracy) when confronted with multivariate, multi-source, state-varying, and noisy datasets. Furthermore, although some information-theory-based strategies have begun to target essential statistical properties—such as the distributional patterns of data points and the intrinsic uncertainties embedded within raw datasets—their implementation often remains constrained. Typically,

these approaches either lean on basic or conventional metrics like Shannon entropy or mutual information, regarding information-theoretic features as supplements rather than harnessing their full potential, or they cannot be directly used to discrete signal (state-transition data).

Contributions. In this chapter, we introduce an information-theory-based framework for neural time-series analysis [1] (InTS). InTS is designed for non-centralized medical decision systems (as shown in Fig. 2.1), to achieve effective feature extraction from multivariate, multi-source, state-varying, and noisy data. The primary contributions are outlined as follows:

(1) We distinguish two types of neural time series data: state-transition and non-state-transition. InTS is applicable to both of them. For both types of data, InTS employs different processing workflows and feature extraction methods, but both are grounded in information theory and share the same platform architecture.

(2) For state-transition data, we proposed four Markov chain-based entropy features to detect abnormal behavioral patterns (*e.g.*, wandering or pacing). These entropy features are carefully designed, fully incorporating the theoretical basis used by clinicians to identify abnormal behaviors, namely by analyzing the complexity and uncertainty of behaviors to recognize anomalous states.

(3) For non-state-transition data, we first compute multiple entropy-based features (*e.g.*, slope entropy and phase entropy). Subsequently, we perform a dual-stage feature selection based on mutual information and Pearson correlation matrices to identify a subset of features with high information content, strong representativeness, and low multicollinearity.

(3) We validated the effectiveness of InTS using three datasets: one dementia care dataset derived from our clinical study on remote healthcare monitoring, and two public datasets, Epileptic Seizure Recognition Dataset and the PTB Diagnostic ECG Database [26, 27]. These three datasets encompass both state-transition and non-state-transition data types.

(4) For state-transition data, we adopt average activity frequency and duration as baseline features, evaluating performance across LR, SVM, MLP, and LSTM models. Relative to these

[1] The code, models, and datasets are available at: https://github.com/yushan-huang/EntropyPipeline.

baselines, InTS enhances accuracy, recall, and F1 score by an average of 13.08 pp, 10.80 pp, and 7.88 pp, respectively.

(5) For non-state-transition data, we utilize features derived by InTS in conjunction with an MLP model, comparing against baseline models employing automatic feature extraction, such as CNN, LSTM, and CNN-LSTM. InTS outperforms baseline models in accuracy, recall, and F1 score by an average of 2.52 pp, 2.42 pp, and 2.32 pp, respectively. Additionally, InTS reduces model parameters by 3.10×.

This chapter is related to the following papers:

Yushan Huang, Yuchen Zhao, Capstick Alexander, Francesca Palermo, Hamed Haddadi, Payam Barnaghi. 'Analyzing entropy features in time-series data for pattern recognition in neurological conditions'. Accepted to Artificial Intelligence in Medicine, Elsevier, 2024.

Yushan Huang, Yuchen Zhao, Hamed Haddadi, Payam Barnaghi. 'Using Entropy Measures for Monitoring the Evolution of Activity Patterns'. Accepted to The 2022 IEEE 8th World Forum on Internet of Things (WF-IoT 2022).

2.2 Background

2.2.1 Clinical Decision Making

This chapter based on a decentralized healthcare monitoring platform designed to provide clinicians with essential decision-making support. In the field of healthcare monitoring, clinicians often need to identify critical patterns from complex physiological and behavioral data when assessing patient conditions [69, 70, 71, 72]. This cognitive process is fundamentally a quantitative judgment of information uncertainty [73, 74, 75]. Information entropy, as a core metric for measuring uncertainty in information theory, naturally serves as a potential tool for feature extraction.

For state-transition data (*e.g.*, triggered infrared sensors), take monitoring patients with Alzheimer's disease as an example. Clinically relevant wandering behavior exhibits three typical charac-

teristics: spatial trajectory irregularity, temporal rhythm disruption, and repetitive behavioral sequences [76, 77]. These manifest in data collected by triggered infrared sensors as irregular patterns in triggered locations. For instance, while the normal nighttime path might be from bedroom to bathroom, frequent deviations such as bathroom to kitchen occur; or bedroom sensors are repeatedly triggered; or a unidirectional loop from bedroom to living room to bathroom and back to bedroom is observed. Our proposed entropy-based method (detailed in Section. 2.3) accounts for spatial aspects (Markov chains), temporal aspects (time window settings, activity duration), and information uncertainty (a series of entropy variants), aligning with real-world clinical judgment criteria. Similarly, for non-state-transition data (*e.g.*, EEG/ECG signals), changes in patterns during abnormal symptom episodes can be effectively captured by entropy variants of nonlinear dynamics indicators [78, 79]. These quantitative entropy metrics correspond clearly to visually interpreted features relied upon by clinicians, such as 'spike-and-slow-wave complexes' [80] or 'irregular RR intervals'[81].

InTS achieves a deep integration of clinical expertise and computational models. It not only provides raw data for direct analysis by clinicians but also offers decision-making support based on InTS results, enabling comprehensive assessments of patient health. Furthermore, feature engineering based on information entropy offers a degree of interpretability [82], providing traceable decision-making evidence: each anomaly judgment can be traced back to specific entropy value changes and their clinical implications. This transparency is critical in medical contexts, allowing clinicians to reference system recommendations while making final decisions based on professional judgment.

2.2.2 Traditional Approaches

The clinical analysis of neurological time-series signals, relies on the precise identification and dynamic monitoring of characteristic patterns over time. Traditional signal processing techniques, supported by solid theoretical foundations and computational efficiency, dominated early research in this field. For example, the Fourier transform, facilitates the decomposition of signals into sinusoidal components across varying frequencies via time-frequency domain conversion, proving instrumental in applications such as heart rate variability analysis and the

estimation of power spectra for EEG rhythm waves, including alpha and beta bands [28, 83, 84]. Despite its widespread utility, the Fourier transform is fundamentally limited by its assumption of signal stationary - that a signal's statistical properties remain constant over time. This assumption makes it unsuitable for analyzing non-stationary physiological phenomena, such as transient ECG arrhythmia or sudden epileptic discharges in EEG recordings during seizures. While traditional methods remain valuable in certain contexts, their inability to handle the complex, non-stationary nature of physiological signals has driven the development of more robust analytical tools.

To mitigate the shortcomings of traditional approaches in handling non-stationary signals, autoregressive (AR) models have been employed, forecasting future signal states through linear combinations of past observations. This technique has demonstrated notable success in applications such as long-term blood pressure trend forecasting and spectral analysis of EEG signals [29, 85, 86]. However, AR model performance critically depends on appropriate model order selection. Suboptimal choices compromise their ability to capture dynamic changes, particularly when analyzing nonlinear phenomena like the abrupt ST-segment deviations in myocardial ischemia - patterns that linear frameworks cannot adequately represent. Wavelet transforms have consequently emerged as a powerful alternative, employing multiresolution time-frequency analysis to effectively capture transient features. This approach enables precise identification of physiological signatures, including QRS complexes in ECG and epileptiform discharges in EEG, through simultaneous temporal and spectral characterization. [30, 87]. However, wavelet transform performance critically depends on proper basis function selection, which typically requires domain-specific expertise and thus limits practical application. Similarly, Hidden Markov Models (HMMs) have emerged as a powerful tool for capturing latent state transitions in physiological signals. Their ability to model sequential patterns has proven particularly valuable for sleep stage classification and cardiac rhythm analysis, providing important insights into the temporal dynamics of signal behavior [31, 88]. Concurrently, techniques that directly derive statistical features from the time domain—such as peaks, variance, and skewness—continue to serve as foundational components in real-time systems, prized for their computational simplicity and responsiveness, particularly in resource-constrained settings [89, 90].

While traditional signal processing techniques have established a foundational framework for pattern recognition, providing an invaluable starting point for subsequent advancements, their limitations have grown increasingly evident as data complexity escalates. When applied to multivariate, multi-source time-series data with state transitions and significant noise contamination, these methods demonstrate both reduced local feature sensitivity and impaired global pattern generalization capability. For instance, in the context of multi-channel physiological signals such as EEG or ECG, traditional approaches often struggle to precisely detect brief yet critical local anomalies—like epileptic spikes or cardiac arrhythmia—owing to their reliance on global statistical measures or frequency-domain decomposition, which lack the granularity to respond effectively to transient shifts. Moreover, anchored in rigid assumptions such as signal stationarity or linear relationships, these techniques frequently falter in adapting to heterogeneous data sources or dynamically evolving states, resulting in limited generalization across broader patterns.

2.2.3 Deep Learning Approaches

Deep neural networks (DNNs) have emerged as a powerful tool in tackling the nonlinear modeling and pattern recognition challenges inherent in medical time-series data, owing to their remarkable capacity for autonomous feature extraction, thereby offering a alternative to the constraints of traditional methodologies. For example, CNNs harness localized receptive fields and weight-sharing strategies to adeptly discern spatial topological features within multi-lead physiological signals, such as ECG or EEG, capturing nuances like local waveform variations or inter-channel spatial correlations with notable efficacy in real-time signal analysis [32]. Concurrently, LSTMs excel at modeling extended temporal dependencies through their gated recurrent units, preserving historical context while filtering out irrelevant noise. This capability has led to significant outcomes in applications like dynamic blood pressure forecasting and early epileptic seizure detection, markedly enhancing prediction accuracy and reliability [91, 92]. As the need for increasingly intricate and precise models intensified within the medical domain, researchers have ventured into integrating diverse architectures to amplify performance. Notably, the CNN-LSTM hybrid amalgamates CNN's prowess in spatial feature extraction with LSTM's

temporal sequence modeling strengths, yielding substantial advancements in two-dimensional time-series analysis and automated diagnostics [33, 93]. Furthermore, innovative configurations pairing Transformer encoders with generative adversarial networks (GANs) have gained momentum, capitalizing on the Transformer's self-attention mechanism to address global dependencies while employing GANs to generate high-fidelity synthetic data, thereby bolstering model adaptability and robustness for clinical tasks [94].

Despite significant advances in applying DNNs to medical time-series analysis, their clinical implementation continues to face substantial challenges. Foremost among these is the stark disparity between model efficiency and resource demands. For example, while Transformer-based models effectively capture long-range dependencies in physiological signals, their quadratic computational complexity relative to sequence length makes them unsuitable for real-time clinical applications like bedside monitoring, where low-latency processing is essential. A second critical issue lies in the absence of robust statistical semantics, which undermines the trustworthiness of decision-making. Predominantly, current DNN approaches prioritize point predictions, such as probability outputs in classification tasks, while sidelining probabilistic modeling of signal generation processes—such as quantifying uncertainty in ECG waveform predictions—or the extraction of high-dimensional statistical features tied to potential pathological correlations, like the mutual information linking heart rate variability to autonomic nervous system activity. This shortfall fails to align with the rigorous demands of precision medicine for uncertainty-aware reasoning and interpretable outcomes [34]. Another issues is the inherent 'black-box' nature of neural networks, which obscures the traceability of their decision-making processes. This opacity poses a significant barrier to their integration into high-stakes clinical contexts, such as diagnostic support or therapeutic decision-making, where healthcare professionals and patients require a clear understanding of the rationale behind model outputs.

2.2.4 Information-Theory-Based Approaches

In neurological time-series analysis, information theory methods are becoming more popular for feature extraction. These approaches combine mathematical theory with machine learning to uncover complex patterns in signals. Specifically, these approaches employ mathematical met-

rics such as entropy and mutual information to model the inherent randomness, complexity, and dynamic interdependencies within signals, thereby uncovering informational patterns. The concept of entropy, pioneered by Shannon to quantify informational uncertainty, established the bedrock of modern information theory [35]. Building upon this foundation, a variety of entropy variants have emerged, including spectral entropy for frequency-domain insights [95] and sample entropy for assessing time-series regularity [96]. For instance, Nurwulan et al. illustrated that multi-scale entropy features derived from tri-axial accelerometer data surpassed conventional statistical features in classification tasks using k-nearest neighbors (KNN) and random forest (RF) algorithms, underscoring the superior capacity of entropy-based features to capture dynamic complexity [36]. Similarly, Bao et al. enhanced the precision of activity prediction models by integrating frequency-domain entropy features with traditional statistical measures such as mean, energy, and correlation, highlighting the versatility of entropy in multidimensional analysis [97].

In summary, information-theory-based approaches offer a multifaceted advantage in neurological time-series analysis, markedly enhancing the performance of learning models, while frequently surpassing the capabilities of traditional DNN models. This enhancement is pivotal for facilitating data-driven, informed clinical decision-making, as these methods leverage metrics like entropy and mutual information to capture signal complexity and uncertainty, yielding more robust and interpretable features for medical diagnostics. Nevertheless, several shortcomings in current research temper the full realization of their potential. Notably, many studies either narrowly concentrate on a single entropy feature—such as Shannon Entropy or Sample Entropy—or employ entropy features in conjunction with conventional statistical measures without customizing their selection to align with specific data attributes, such as the Markovian properties of a system or stochastic state transitions. This lack of tailored feature selection is often unaccompanied by rigorous justification explaining why a particular entropy metric is better suited to the task at hand. Furthermore, these information-theoretic methods are predominantly task-specific, tailored to isolated applications like epilepsy detection or heart rate variability analysis, and fall short of providing a unified, scalable entropy-based pipeline capable of accommodating a broad spectrum of tasks.

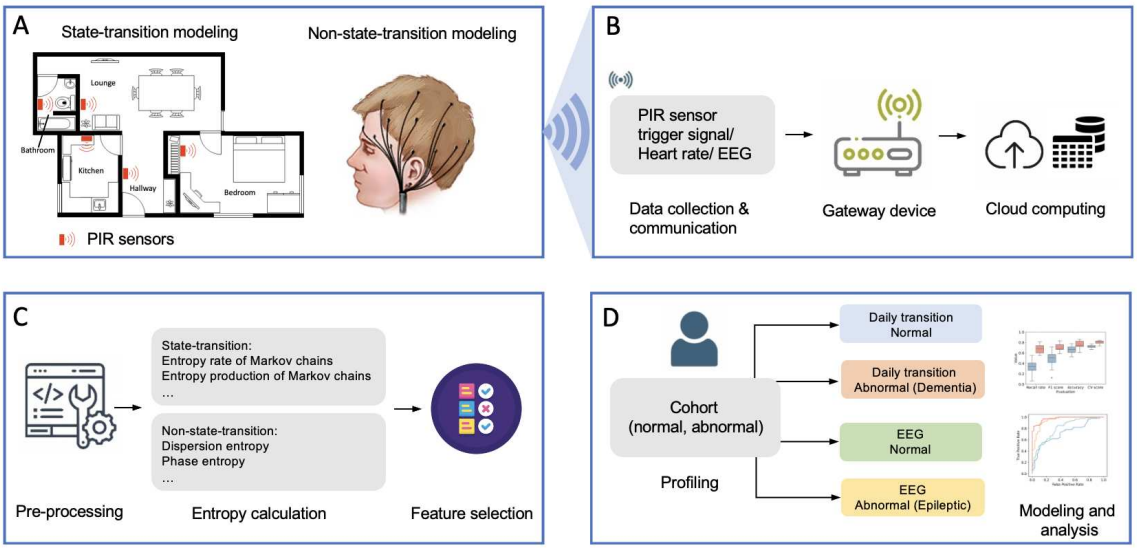


Figure 2.2: Methods overview.

2.3 Methodology

The operational workflow of InTS comprises four principal components: data collection, data preprocessing, feature construction, and modeling, as depicted in Figure 2.2. Within this structure, the initial three phases—data collection, preprocessing, and feature construction—are executed locally on edge devices, such as IoT gateways or wearable units, whereas the modeling phase is relegated to cloud servers. It should be noted that, although InTS supports the analysis of both state-transition data and non-state-transition data within a single healthcare monitoring platform, the two data types are processed through independent parallel processing workflows. This design aligns with clinical decision-making logic, as the two data types correspond to distinct diagnostic paradigms: behavioral pattern analysis (*e.g.*, wandering behavior detection) relies on uncertainty among discrete events, whereas physiological signal analysis (*e.g.*, seizure identification) focuses on uncertainty in continuous waveforms.

2.3.1 Data Collection and Preprocessing

To enable systematic analysis, we classify the collected data into two distinct categories: state-transition signals and non-state-transition signals. State-transition signals record the dynamic state-changing processes of participants. For instance, passive infrared (PIR) sensors, as depicted in Figure 2.2.A, are strategically positioned across various rooms within a household to monitor occupant movement. These sensors activate upon detecting motion, logging the pre-

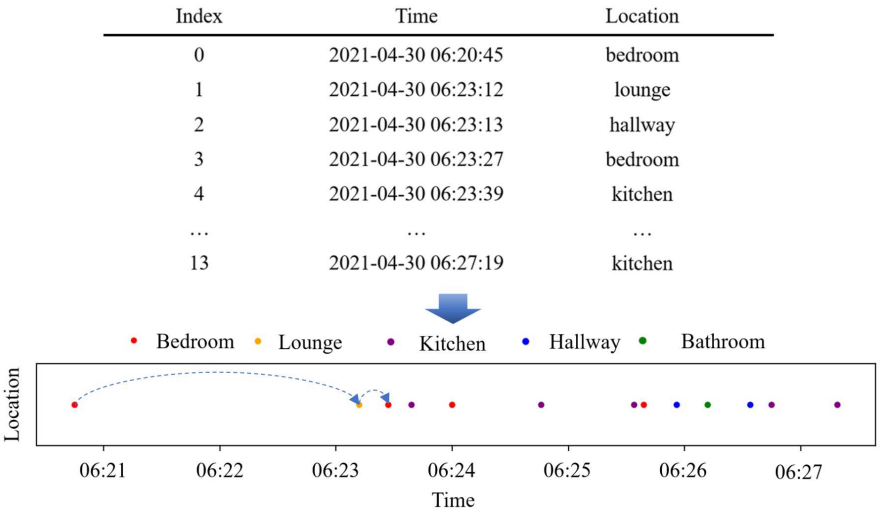


Figure 2.3: Example of Markov chain generated from state transition data.

cise location and timestamp of each event. The raw data amassed by PIR sensors, illustrated in Fig. 2.3, manifest as discrete points—individual instances of time and location—that, in isolation, do not vividly portray the continuum of state transitions. However, through a data preprocessing phase, we synthesize these disparate points by linking them temporally to recast them into a first-order Markov chain, thereby achieving a more coherent representation of the state transition process, such as sequential room-to-room movements, as showcased in the lower segment of Figure 2.3. While the non-state-transition signals, exemplified by EEG signals. These are acquired through continuous sampling at a fixed frequency over regular time intervals, generating smooth time-series datasets that reflect ongoing brain activity rather than discrete state shifts.

In the preprocessing phase, we implement a suite of procedures encompassing denoising, Z-score normalization, and the management of missing values to ensure data quality and consistency. Specifically, Z-scores transform variables to a uniform scale (mean = 0, SD = 1). For handling missing values, our default approach is forward-fill, whereby missing data points are imputed using the preceding observation. While for missing values occurring at the outset of a dataset, where no prior observations are available, we employ backward-fill, utilizing the nearest subsequent observation to maintain data integrity. Furthermore, for non-state-transition signals, we undertake a numerical transformation of location labels. For instance, descriptive labels such as ‘bedroom’ are assigned a fixed value of 1, and ‘kitchen’ is designated as 2.

2.3.2 Feature Construction

For state-transition data, we leverage the constructed first-order Markov chain and develop four entropy features. For non-state-transition data, we initially generate an assortment of entropy-based features, then employ a two-stage feature selection. In the subsequent sections, we provide an in-depth exposition of the entropy feature construction process for state-transition data, detailing how entropy metrics are derived from the Markov chain, alongside the dual-stage feature selection.

State-Transition Data and Markov Chains

The state-transition data encompass trigger timestamps and corresponding location details, offering a wealth of spatial and temporal insights. In the spatial dimension, we conceptualize each recorded location—such as living room or kitchen within a household—as a discrete node. It enables the computation of activation frequencies and counts for specific locations or pathways (*e.g.*, transitions from bedroom to kitchen). In the temporal dimension, we can further measure both occupancy times (*e.g.*, duration in the living room) and transition intervals between locations.

Shannon Entropy of Markov Chains

We hypothesize that human activity patterns, like location visitation sequences, can be accurately represented by Markov chains. This approach treats activities as stochastic events, enabling statistical analysis of their frequencies and patterns. We first regard each activity/location as independent points, and adopt Shannon's entropy. The selection of Shannon's entropy is motivated by its ability to quantify the uncertainty and intricacy of information in a data-driven manner. Suppose $X = \{x_1, x_2, \ldots, x_n\}$ denote n states (*e.g.*, a participant whose daily activities span n distinct locations), and $S = \{s_1, s_2, \ldots, s_L\}$ ($s \in X$) denote a Markov chain trajectory with length L. Within this context, the Shannon entropy $H(X)$ serves to measure the informational uncertainty of the Markov chain, with its mathematical formulation reflecting the probability distribution of location occurrences:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i) \tag{2.1}$$

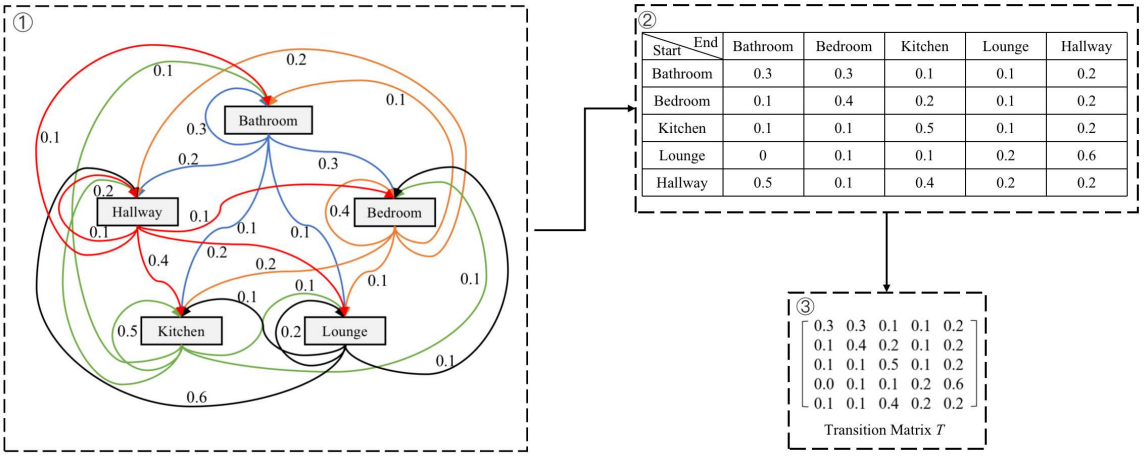
Where $P(x_i)$ denotes the probability of a participant engaging in a specific state x_i, such as visiting a particular location. As the frequency of a participant's activities fluctuates—for instance, if an individual begins frequenting the kitchen more often while reducing time spent in the living room—$H(X)$ will change. Specifically, an elevated $H(X)$ value signifies heightened uncertainty and complexity, indicative of a more diverse and less predictable behavioral repertoire, such as frequent transitions among multiple locations without a discernible routine. Conversely, a diminished $H(X)$ value reflects a reduction in uncertainty, pointing to a more consistent and predictable behavioral pattern, such as prolonged stays at a single location or adherence to a fixed daily trajectory.

Entropy Rate of Markov Chains

We propose the entropy rate of a first-order Markov chain, which is prompted by the shortcomings of Shannon entropy: while Shannon entropy excels at quantifying the randomness or uncertainty associated with isolated events—such as the probability distribution of visiting a specific location—its design does not account for the sequential dependencies and interrelationships that are hallmarks of human activities. For instance, in daily routines, the transition from kitchen to dining room is often directly influenced by the preceding state, underscoring the interconnected nature of behavioral sequences. In contexts where activities are interdependent and events exert mutual influence, traditional Shannon entropy provides only a static, isolated measure of information, falling short of capturing the full complexity and temporal evolution inherent in such data. To address this gap, we employ the entropy rate of Markov chains, integrating the probabilistic dependencies between successive states [98, 99].
Similarly, Suppose $X = \{x_1, x_2, \ldots, x_n\}$ denote n states and $S = \{s_1, s_2, \ldots, s_L\}$ $(s \in X)$ denote a Markov chain trajectory with length L. Suppose the previous state is x_i, the current state is x_j, then the transition probability is given by:

$$P_{ij} = P(x_j \mid x_i) \tag{2.2}$$

For states $x_i, x_j \in X$ in an n-state Markov chain, the transition probabilities form an $n \times n$ matrix $T = (P_{ij})_{i,j \in X}$ (see Fig. 2.4). The stationary distribution π satisfies:



Start \ End	Bathroom	Bedroom	Kitchen	Lounge	Hallway
Bathroom	0.3	0.3	0.1	0.1	0.2
Bedroom	0.1	0.4	0.2	0.1	0.2
Kitchen	0.1	0.1	0.5	0.1	0.2
Lounge	0	0.1	0.1	0.2	0.6
Hallway	0.5	0.1	0.4	0.2	0.2

Figure 2.4: Markov chain entropy rate calculation example. Rectangular nodes in ① indicate locations (states), with arrows showing transitions between them. Color coding represents starting locations: blue (bathroom), orange (bedroom), green (kitchen), black (lounge), and red (hallway). Transition probabilities (shown beside arrows) correspond to Table ② and Transition Matrix T ③.

$$\pi = \pi T \tag{2.3}$$

The stationary distribution π (an n-dimensional vector for an n-state Markov chain) determines the entropy rate:

$$\xi = -\sum_{ij}^{n} \pi_i P_{ij} \log P_{ij} \tag{2.4}$$

Where π_i represents the probability in the stationary distribution associated with activity $x_i \in X$, which reflects the long-term equilibrium state of the Markov chain. The entropy rate calculation requires two time windows: one for transition probabilities P_{ij} and another for the stationary distribution π_i to characterize the time-series data. The time window for the transition probability is typically determined by the specific task objectives, while the time window for the stationary distribution requires special attention, as it should capture the stationary behavior patterns of the participant. For instance, a participant's daily activities may be influenced by seasonal variations, so it is essential to set the time window to avoid the effects of seasonal fluctuations, such as using a long time window across seasons to eliminate seasonal effects. The dual time-window design captures both short-term transitions (P_{ij}) and

Algorithm 1 Entropy Rate of Markov Chains.

Input: Trajectory $s = \{s_1, \ldots, s_L\}$, states $X = \{x_1, \ldots, x_n\}$, windows $2 \leq TW_1, TW_2 \leq L$
Output: Stationary distribution π, entropy rates ξ
/* 1. Transition Matrix T */
for $i = 1$ to n **do**
  $N_i \leftarrow$ count of x_i in $s[1 : TW_1{-}1]$
  for $j = 1$ to n **do**
    $T_{ij} \leftarrow$ count of $x_i \to x_j$ divided by N_i if $N_i > 0$ else $1/n$
  end for
end for
/* 2. Stationary Distribution π */
$\pi^{(0)} \leftarrow (1/n, \ldots, 1/n)$; $k \leftarrow 0$
repeat
  $\pi^{(k+1)} \leftarrow \pi^{(k)} T$; normalize $\pi^{(k+1)}$
  $k \leftarrow k + 1$
until $\|\pi^{(k)} - \pi^{(k-1)}\|_1 < \epsilon$
$\pi \leftarrow \pi^{(k)}$
/* 3. Entropy Rates ξ */
$\xi \leftarrow$ empty list
for $t = 1$ to $L{-}TW_2{+}1$ by TW_2 **do**
  for $i = 1$ to n **do**
    $N_i \leftarrow$ count of x_i in $s[t : t{+}TW_2{-}2]$
    for $j = 1$ to n **do**
      $P_{ij} \leftarrow$ count of $x_i \to x_j$ divided by N_i if $N_i > 0$ else 0
    end for
  end for
  $\xi_t \leftarrow 0$
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      if $P_{ij} > 0$ **then**
        $\xi_t \leftarrow \xi_t - \pi_i P_{ij} \log_2 P_{ij}$
      end if
    end for
  end for
  Append ξ_t to ξ
end for
return π, ξ

long-term trends (π_i). The full computation process is detailed in Algorithm 1.

Entropy Production Rate of Markov Chains

Shannon entropy and the entropy rate exhibit a critical shortfall when addressing temporal irreversibility—they cannot describe the probabilistic asymmetry distinguishing forward evolutionary trajectories (*e.g.*, from living room to kitchen) from their reverse counterparts (*e.g.*, from kitchen to living room). This limitation becomes evident in scenarios where directional differences carry significant behavioral meaning. For example, consider an individual continuously moving in a circular pattern within a home (*e.g.*, from living room to kitchen, then to dining room, and back to living room), resulting in a unidirectional sequence of transitions

with no reverse movements. Even if this unidirectional loop exhibits randomness akin to a disordered wandering sequence (*e.g.*, lacking a fixed direction) in terms of entropy value, the stark contrast in their internal directional dynamics eludes capture by Shannon entropy or entropy rate alone. To bridge this gap, we incorporate the entropy production rate (EPR), derived from nonequilibrium thermodynamics, into our analytical repertoire. EPR quantifies the inherent directional bias within the state transition matrix of a Markov chain by assessing the asymmetry between forward and reverse transition probabilities, thereby offering a direct measure of the intrinsic irreversibility embedded in behavioral dynamics.

Taking indoor activities as an illustrative case, we can elucidate the application and significance of the EPR in analyzing behavioral dynamics. For healthy individuals, daily activity patterns typically adhere to the detailed balance condition, whereby the transition probability (P_{ij}) from one location, such as a bedroom (x_i), to another, like a living room (x_j), and its reverse (P_{ji}) manifest symmetric distributions at steady state, satisfying the equation $\pi_i P_{i\to j} = \pi_j P_{j\to i}$. Here, π_i and π_j denote the stationary probabilities of the bedroom and living room, respectively. Under these circumstances, the forward and reverse pathways exhibit statistical equilibrium, resulting in an EPR that approaches zero, indicative of time-reversal symmetry where the behavioral sequence displays no pronounced directional preference. In stark contrast, individuals in pathological states—such as those with Alzheimer's disease—may exhibit persistent directional biases stemming from neurological dysregulation, such as dysfunction in the basal ganglia, leading to scenarios where $P_{ij} >> P_{ji}$.

From the perspective of non-equilibrium thermodynamics, EPR quantifies the minimal energy dissipation rate required to sustain a specific behavioral pattern. EPR can be defined as:

$$\mathrm{EPR} = \sum_{i,j} \pi_i P_{ij} \ln \frac{P_{ij}}{P_{ji}} \tag{2.5}$$

For any state pair x_i and x_j, if the forward transition probability P_{ij} consistently exceeds the reverse probability P_{ji}, the logarithmic term $\ln \frac{P_{i\to j}}{P_{j\to i}}$ contributes a positive entropy production. By cumulatively integrating these asymmetric contributions, the EPR macroscopically

quantifies the global irreversibility intensity of behavioral dynamics.

The EPR can be estimated using the Neural Estimator for Entropy Production (NEEP) [100]:

$$\hat{EPR}(\theta) = \sum_{t=1}^{L-1} \left[h_\theta(s_t, s_{t+1}) - e^{-h_\theta(s_t, s_{t+1})} \right] \tag{2.6}$$

where h_θ is a trainable model with θ as trainable parameters, operating on consecutive states of trajectory $S = \{s_1, \ldots, s_L\}$ [100]. Here, ΔS_θ quantifies the directional disparity in state transitions and is expressed as:

$$\Delta S_\theta \left(s_t, s_{t+1} \right) \equiv h_\theta \left(s_t, s_{t+1} \right) - h_\theta \left(s_{t+1}, s_t \right) \tag{2.7}$$

Within the NEEP framework, an embedding layer transforms discrete states (*e.g.*, living room, kitchen) into continuous, trainable vector representations [100]. These vectors are then processed by a hidden MLP layer, which employs nonlinear transformations to derive directional features of state transitions. Notably, the length of the time-series data is a critical factor in the training process of the NEEP model. A sufficiently prolonged temporal span ensures an adequate volume of training samples to optimize the neural network parameters θ, while also accurately reflecting the participant's behavioral patterns, such as habitual pathways or directional biases. An insufficiently short time window may fail to encompass long-term behavioral trends, potentially undermining the reliability of the EPR estimate.

Von Neumann Entropy of Markov Chains

The von Neumann entropy (VNE) traces its origins to quantum mechanics, where it was initially developed as a metric to quantify the uncertainty or informational disorder inherent in quantum states, with its theoretical underpinnings extensively elaborated in quantum information theory [101]. Although rooted in quantum mechanics, the foundational concept of computing VNE via density matrices exhibits remarkable versatility beyond its original domain. At its core, a density matrix serves as a mathematical construct capable of representing the state distributions and evolutionary interactions within any complex system in a probabilistic frame-

work, extending its utility far beyond quantum phenomena [102]. This adaptability renders VNE a potent tool for characterizing complexity and uncertainty across diverse contexts, such as biological systems, social networks, or time-series data analysis, where it can illuminate state distribution patterns or interactions, like those in human behavioral sequences or neurological signals.

The limitation of a first-order Markov chain is that it only considers the relationship between the current state and the previous state, potentially losing important information when handling long time-series data. To address this, we propose a von Neumann Entropy-based approach built on Markov chains. This method takes a week as the unit of analysis, comparing activities and path differences between days, which to some extent mitigates the shortcomings of first-order Markov chains in processing long time-series data. Specifically, we structure the information from seven days into a matrix, with each column representing the activity state distribution for a single day (*e.g.*, frequencies or durations of visits to different locations), thereby constructing a density matrix that encapsulates the behavioral pattern. Features from frequency and duration are first fused via a single-layer MLP with a negligible number of parameters before being used in subsequent processing. Leveraging this framework, we employ the VNE to quantify the complexity and uncertainty inherent in this matrix. Unlike traditional entropy measures like Shannon entropy, which typically computes the probability distribution of a single variable—such as the frequency of visiting the living room on a given day—and struggles to capture inter-day interactions (*e.g.*, whether a transition from bedroom to living room on Monday correlates with a similar transition on Tuesday or reflects a weekly cyclical trend), VNE's matrix-based approach offers a distinct advantage. By treating the seven-day state distributions as an integrated whole, the density matrix enables VNE to simultaneously characterize both the individual behavioral features of each day (*e.g.*, the visitation profile on a specific day) and the dynamic interdependencies across days (*e.g.*, transition tendencies from Monday through Sunday). VNE also remains robust against noise and incomplete data, providing stable system-state approximations despite minor perturbations or missing values.

The VNE is defined based on a density operator ρ, quantifying the complexity and uncertainty of the pathway matrix through the trace of the logarithmic density operator:

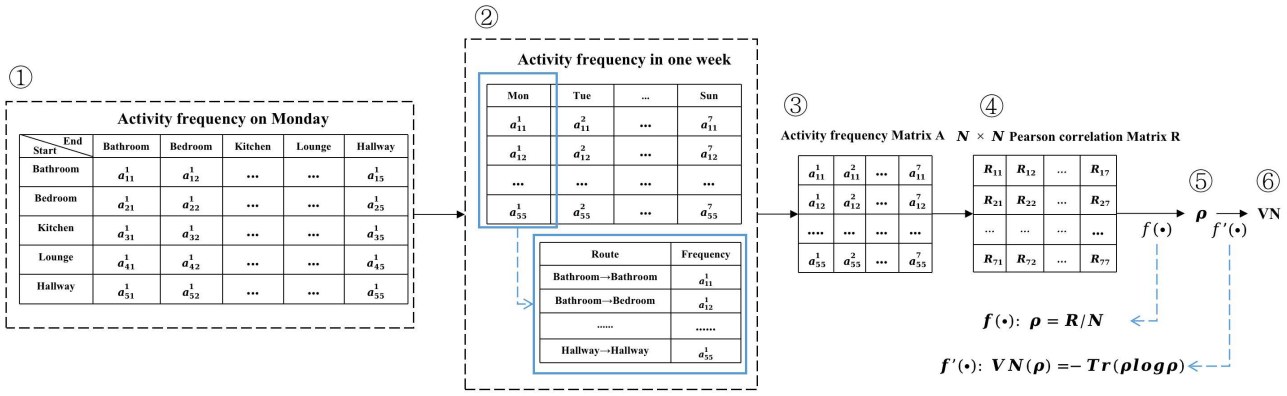


Figure 2.5: For a 5-state Markov chain, von Neumann entropy calculation involves: *Spatial*: (1) Daily route frequencies $a^d_{i,j}$ (i, j: states, d: date); (2) Weekly aggregation; (3) Frequency matrix A; (4) Pearson matrix R; (5) Density matrix ρ by $f(\bullet)$; (6) von Neumann entropy by $f'(\bullet)$. And for *Temporal*: Replace frequencies with durations.

$$VNE(\boldsymbol{\rho}) = -\operatorname{tr}(\boldsymbol{\rho}\log\boldsymbol{\rho}) \tag{2.8}$$

A pivotal aspect of computing VNE is constructing the density operator ρ, which must fulfill three conditions: (1) Be Hermitian (self-adjoint); (2) Possess a unit trace (normalized); (3) Be positive semi-definite (non-negative eigenvalues). Take a 5-state Markov chain and the frequency of weekly activities (paths) as an example, as shown in Fig. 2.5, we first calculate the frequency of different paths for each day (Fig. 2.5, step 1), obtaining the daily activity frequency matrix. Then, we flatten it into a single column and concatenate the frequencies of all seven days into matrix $\boldsymbol{A}$ (Fig. 2.5, steps 2 and 3). Next, we treat each column as a variable and compute the Pearson correlation matrix between columns, resulting in an $N \times N$ matrix $\boldsymbol{R}$ (Fig. 2.5, step 4), where $N = 7$ in this example. Subsequently, we calculate $\boldsymbol{\rho} = \boldsymbol{R}/N$, ensuring that the density matrix $\boldsymbol{\rho}$ satisfies the three aforementioned conditions (Fig. 2.5, step 5). Finally, we compute the VNE according to Equation 2.8 (Fig. 2.5, step 6).

Next, we analyze the reasons why the density matrix $\boldsymbol{\rho}$ satisfies the three aforementioned conditions: (1) $\boldsymbol{R}$, as a Pearson correlation matrix, is symmetric and thus Hermitian; (2) The diagonal elements of $\boldsymbol{R}$ are 1, yielding a trace of N, and division by N ensures $\operatorname{tr}(\boldsymbol{\rho}) = 1$, satisfying the unit-trace condition; (3) Positive semi-definiteness requires non-negative eigenvalues for $\boldsymbol{R}$, which typically holds for complete correlation matrices, though sparse data (*e.g.*, untraveled

Algorithm 2 Feature Selection with PCC and MI.

Require: Feature set $e_{all} = \{e_1, \ldots, e_m\}$; label y_{trg}; count N; PCC threshold $threshold$; max retries max_time
Ensure: Selected feature set e_{sel}

Compute $MI(e, y_{trg})$ for all $e \in e_{all}$
Sort e_{all} by descending $|MI(e, y_{trg})|$; set $trial \leftarrow 0$
while $trial < max_time$ **do**
  $e_{sel} \leftarrow$ first N features in sorted e_{all}
  $i \leftarrow N$
  repeat
    $changed \leftarrow$ FALSE
    for all $e, e' \in e_{sel}, e \neq e'$ **do**
      if $|PCC(e, e')| > threshold$ **then**
        Remove the one with smaller $|MI(\cdot, y_{trg})|$ from e_{sel}
        $changed \leftarrow$ TRUE; BREAK
      end if
    end for
  until NOT $changed$
  while $|e_{sel}| < N$ AND $i < |e_{all}|$ **do**
    $f \leftarrow e_{all}[i]$; $i \leftarrow i + 1$
    Add f to e_{sel}
    repeat
      $changed \leftarrow$ FALSE
      for all $e, e' \in e_{sel}, e \neq e'$ **do**
        if $|PCC(e, e')| > threshold$ **then**
          Remove lower $|MI|$ one; $changed \leftarrow$ TRUE; BREAK
        end if
      end for
    until NOT $changed$
  end while
  if $|e_{sel}| \geq N$ **then**
    BREAK
  else
    $threshold \leftarrow 0.9 \times threshold$
    $trial \leftarrow trial + 1$
  end if
end while
return e_{sel}

pathways) may yield zero eigenvalues, still meeting the semi-definite criterion ($\lambda_j \geq 0$).

While ρ meets these criteria, real-world PIR sensor data may render it sparse, with many zero entries due to infrequent pathways, posing challenges for standard logarithmic computations of $\log \rho$ (*e.g.*, anomalies from logging zeros). To circumvent this, we employ the Mercator Series for matrix logarithm calculation [103]:

$$\log(\boldsymbol{\rho}) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(\boldsymbol{\rho} - I)^k}{k} \tag{2.9}$$

where k is the order, and I is the identity matrix.

Feature Selection

Feature selection enhances model efficiency by filtering relevant variables that significantly contribute to predictive power, while reducing data dimensionality and computational complexity. For datasets suitable for state-transition modeling, we focus on entropy features associated with Markov chains, including Shannon entropy, entropy rate, entropy production rate, and VNE. In cases where state-transition modeling is infeasible, we apply a methodology based on the Pearson correlation coefficient (PCC) and mutual information (MI) to select the most appropriate features from various entropy methods. The detailed procedure is described in Algorithm 2.

Initially, features are ranked by calculating the absolute MI between each feature e and the target label y_{trg}. This approach captures the non-linear dependencies between variables, providing an efficient first-pass selection. The MI is calculated as follows:

$$MI(e; y_{trg}) = \sum_{y_{trg}} \sum_{e} p(e, y_{trg}) \log \left(\frac{p(e, y_{trg})}{p(e)p(y_{trg})} \right), \tag{2.10}$$

with summation over all possible values of e and y_{trg}.

After ranking features by their absolute MI values, we select the top N candidates. To address potential feature redundancy, we then calculate pairwise PCC between distinct features e and e' in this candidate set. When any pair exceeds a predefined PCC threshold, indicating significant linear dependence, we retain only the feature with higher MI to y_{trg} and exclude the other. This process continues iteratively until we obtain N features that are both strongly predictive and minimally redundant. The PCC is calculated as:

$$PCC(e, e') = \frac{\sum (e - \bar{e})(e' - \bar{e'})}{\sqrt{\sum (e - \bar{e})^2} \sqrt{\sum (e' - \bar{e'})^2}} \tag{2.11}$$

where $\bar{e}$ and $\bar{e'}$ denote the mean values of feature e and e', respectively.

In our experiments, we adopt default thresholds of 0.8 for PCC (indicating high linear correlation). This value is based on conventions from prior research [104, 105], typically regarded as

an empirical boundary for characterizing strong linear correlation. We also reserve a dynamic threshold adjustment mechanism (although not triggered in our experiments and datasets), which allows the algorithm to automatically explore looser thresholds to obtain sufficient features that meet the low redundancy requirement when the initial threshold is inappropriate.

2.4 Implementation & Evaluation Setup

We applied InTS to three distinct datasets from different domains: (1) A dementia care dataset from the ongoing Minder study by the UK Dementia Research Institute; (2) A publicly available epileptic seizure dataset (ESRD) focusing on EEG signals; and (3) A heart disease dataset (PTBDB) focusing on ECG signals. To build and evaluate our models, we employed a range of machine learning approaches for comparative analysis, including LR, SVM, MLP, CNN, LSTM, and a combined CNN-LSTM architecture. To ensure the proper structure of the experiments, all datasets were divided into training, validation, and test sets in a 70%, 15%, 15% ratio, respectively. Furthermore, to validate the stability and reliability of our findings, all experiments were repeated 30 times, with both the mean and standard deviation of the results calculated.

2.4.1 Datasets

We evaluated InTS using three datasets. One is the Minder dataset [2], which meticulously captures daily activity information of individuals with dementia (PLWD) [99] from the medical monitoring platform. Due to the medical data access restrictions of the platform, we were limited to using triggered sensor data and could not access EEG and ECG signals. Therefore, we utilized two publicly available medical datasets: the Epileptic Seizure Recognition Dataset (ESRD), a collection of EEG signals used to study epilepsy patient conditions [106]; and the PTB Diagnostic ECG Database (PTBDB), a repository of ECG recordings employed for cardiac condition research [26, 27]. Although both datasets underwent basic filtering, resulting in relatively low overall noise levels, they still retained physiological noise and the experimental results still demonstrate the effectiveness of InTS.

[2]Due to the platform's compliance restrictions on the use of medical data, the Minder dataset is only applicable to research within the currently authorized scope. Consequently, subsequent research chapters of this thesis do not utilize this dataset.

Minder Dataset

This research leverages data sourced from the Minder in-home monitoring initiative (as depicted in Fig. 2.1), a program meticulously crafted to people living with dementia (PLWD) [99]. The Minder platform incorporates a sophisticated array of cutting-edge sensors and smart devices, enabling the systematic acquisition of diverse digital metrics, such as records of daily activities, usage patterns of household appliances, and relevant clinical health insights, thereby providing a rich and real-time dataset for analysis. Furthermore, the project's deployment prioritizes user privacy and data security (no image data), ensuring the trustworthiness and dependability of the monitoring process. The study's design and execution adhered rigorously to ethical principles, having earned formal endorsement from the London-Surrey Borders Research Ethics Committee and the South West London Ethics Committee following a thorough review and evaluation process [3]. To safeguard ethical integrity and protect participants' rights, we secured fully informed written consent from all individuals involved prior to the study's initiation, with the consent process documented and archived for transparency.

The Minder dataset is annotated using alerts produced by the Minder platform, which functions continuously around the clock to monitor users' health conditions in real time. These alerts are meticulously validated by a dedicated monitoring team through direct communication with PLWD or their caregivers, ensuring a high degree of precision and reliability in data pertaining to potential health incidents, such as falls, unusual behavioral patterns, hospital stays, urinary tract infections, and manifestations of anxiety, depression, agitation, confusion, and sleep disruptions. Data linked to participants who have encountered these adverse health occurrences are distinctly tagged, facilitating ongoing monitoring and providing structured support for in-depth analysis of these health transitions. The primary emphasis of this investigation centers on examining the activity records captured by the Minder platform, classified as state-transition data, which reflect the dynamic shifts in participants' health statuses. Spanning from December 2020 to March 2022, the dataset comprises a comprehensive total of 3,762 person-weeks of detailed records, offering a robust temporal scope and strong representativeness. The mean

[3]Approval information: https://www.hra.nhs.uk/about-us/committees-and-services/res-and-recs/search-research-ethics-committees/london-surrey-borders/

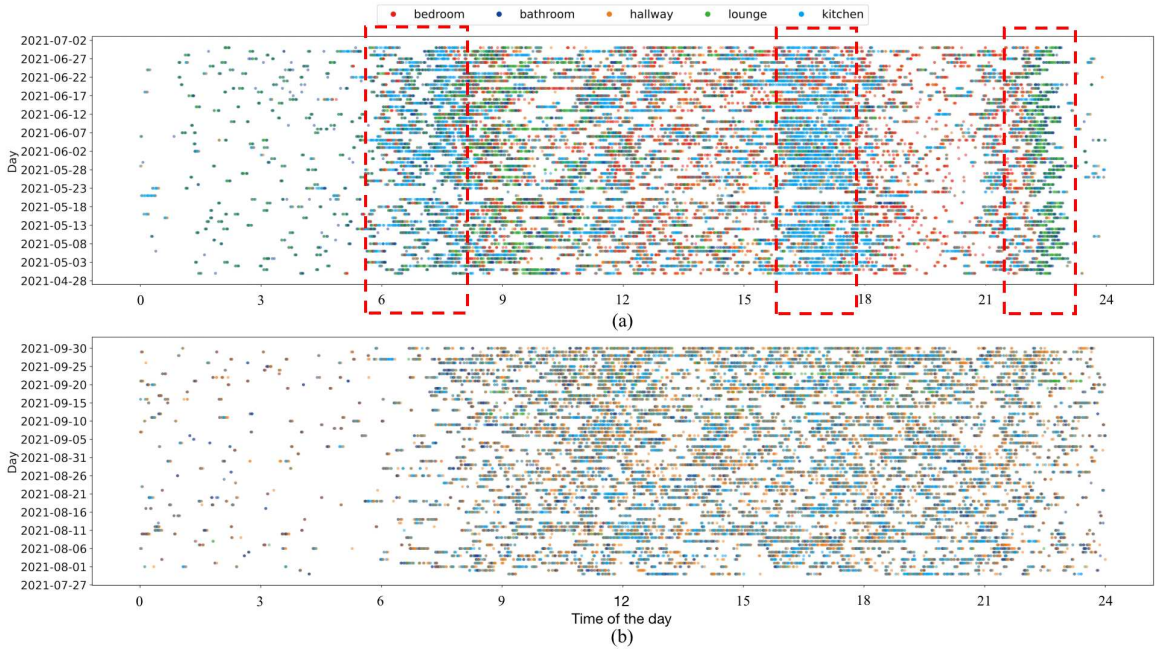


Figure 2.6: Examples of PLWDs with (a) clear routines and (b) fewer routines. The more routine-oriented participant (a) shows consistent daily patterns (red boxes) at fixed times. X-axis represents time of day, Y-axis represents colors indicate house locations.

age of participants stands at 79 years, underscoring that the study predominantly involves an elderly cohort. Moreover, to safeguard participant confidentiality and align with ethical standards, all data utilized in this study have undergone rigorous anonymization procedures prior to analysis, ensuring that individual identities remain untraceable.

The activity data is gathered through PIR sensors strategically placed across various key locations within the residence, such as the kitchen, bathroom, bedroom, living room, and hallway, as depicted in Fig. 2.2.A. These sensors capture an event with second-by-second accuracy whenever an individual moves past them. An example of raw data, highlighting a comparison between the daily routines of two PLWDs, is presented in Fig. 2.6.

Epileptic Seizure Recognition Dataset

The ESRD constitutes a non-state-transition dataset, meticulously assembled from EEG recordings of 500 clinical participants, encompassing a total of 11,500 time-series samples tailored to investigate the EEG signatures associated with epileptic phases [106]. Each sample is precisely divided into 23 temporal segments, with each segment comprising 178 data points, achieving a temporal resolution of 1 second, thereby ensuring continuity and uniformity across the time dimension. The University of California, Irvine (UCI) applied sophisticated data reconstruc-

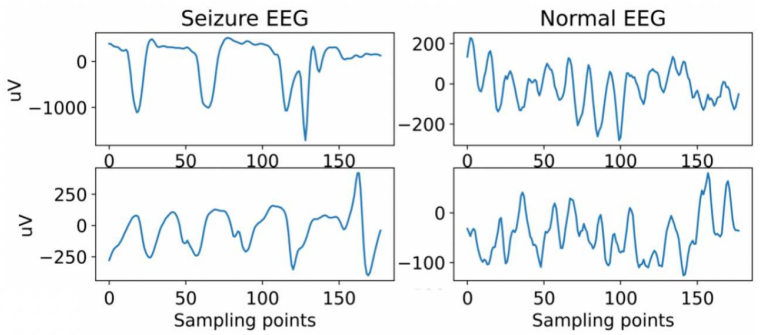


Figure 2.7: ESRD visualization with sampling points (X-axis) and EEG signals (μV) (Y-axis).

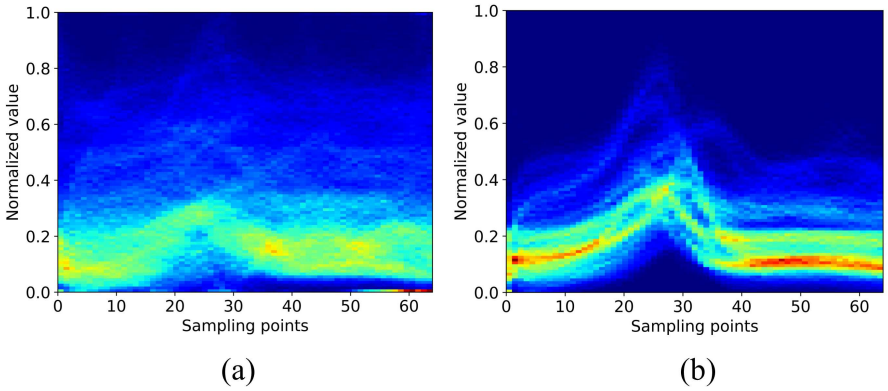


Figure 2.8: PTBDB histogram color maps: (a) abnormal and (b) normal cases. X-axis shows sampling points, Y-axis shows normalized heartbeat values.

tion methodologies, utilizing randomized segment permutations to produce this standardized dataset, with its architectural design clearly outlined in Fig. 2.7, offering researchers a well-structured analytical foundation. The ESRD classifies physiological conditions into five distinct groups: a singular category is reserved for documenting epileptic seizure instances, while the other four categories span healthy control conditions ranging from wakefulness through various sleep stages, devoid of epileptic symptoms. Although the dataset inherently accommodates multiclass classification modeling, prevailing research trends favor binary classification approaches, predominantly aiming to differentiate the epileptic cohort (Category 1) from the non-epileptic control groups, a strategy that balances analytical simplicity with the critical need for seizure detection. Consistent with this focus, the present study targets binary differentiation between epilepsy-positive and epilepsy-negative states.

PTB Diagnostic ECG Database

This dataset aggregates 549 entries from 290 participants (209 males and 81 females), categorized as non-state-transition data [26, 27]. The participants' ages span a wide range, from 17 to 87 years, with a mean age of 57.2 years, providing a diverse and representative sample pool for analysis. The recordings were captured at a sampling rate of 125 Hz, ensuring high-resolution and precise signal acquisition. The diagnostic classifications within PTBDB encompass an array of cardiac conditions, including myocardial infarction, cardiomyopathy or heart failure, bundle branch block, dysrhythmia, myocardial hypertrophy, valvular heart disease, and myocarditis, alongside a healthy control cohort, establishing a comprehensive baseline for comparative studies. In this investigation, we specifically isolated heartbeat signals from ECG lead 2 [107], narrowing our focus to the myocardial infarction subgroup and the healthy control group to examine signal characteristics under specific cardiac conditions. The dataset comprises a robust total of 14,552 samples. Fig. 2.8 employs color-mapped histograms to visually depict the distribution of abnormal and normal ECG signals within the PTB data.

2.4.2 Models and performance metrics

This section presents the experimental models and performance metrics for both data types.

Models for State-Transition Data

For state-transition data, after feature extraction, we selected LR, SVM, MLP, and LSTM as models, which have been widely utilized in medical time-series data [108, 109, 110].

We designated LR and linear-kernel SVM as representatives of linear models, while selecting MLP and LSTM to exemplify non-linear models, thereby encompassing a broad range of machine learning methodologies. For the parameter tuning of LR and SVM, we leveraged the $GridSearchCV$ utility from Python's $scikit-learn$ library, employing a grid search approach to systematically explore and evaluate parameter combinations, ensuring optimal model performance across various configurations. Specifically, the parameter space for LR was defined, including the regularization parameter C with values (0.001, 0.01, 0.1, 1, 10, 100), regularization techniques (L1 or L2), and a suite of solvers (liblinear, newton-cg, lbfgs, sag, saga), allowing

for an extensive investigation of its behavior under diverse optimization scenarios. For SVM, the tuning process concentrated on refining the C values (0.001, 0.01, 0.1, 1, 10, 100).

To enhance the performance and training efficiency of MLP and LSTM, we merge learning rate warm-up with cosine annealing, enabling dynamic adaptation of the learning rate to suit the needs of model convergence. To maintain fairness and consistency in model comparisons, we standardized the architectures of both MLP and LSTM to include two hidden layers, with the number of hidden units adjusted proportionally to the input dimensions to align model capacity with task complexity. Specifically, for baseline (input dim a, hidden units b) and entropy-based (input dim c, hidden units d) methods, we maintained $a/c = b/d$. Training used Binary Cross-Entropy loss with SGD optimization. To capture periodic patterns and temporal/monthly influences, we set the LSTM time step to 4 weeks, enhancing sequential regularity and long-term trend modeling. The dataset is split 70:15:15 into training, test, and validation sets.

Baseline. For state-transition data represented by the Minder dataset, investigations into feature extraction from PIR sensors remain relatively limited, with sparse coverage in existing literature. Drawing upon the foundational work of Chimamiwa et al. [111], we adopted the average frequency and duration of various activities as baseline features. To further discern behavioral variations across the temporal dimension, we segmented the day into two distinct periods: daytime (06:00 to 18:00) and nighttime (18:00 to 24:00 and 00:00 to 06:00), a partitioning that facilitates the identification of potential shifts in diurnal and nocturnal rhythms. These extracted and structured features were subsequently incorporated into a range of machine learning models for analysis, encompassing LR, SVM, MLP, and LSTM.

InTS. For state-transition data, InTS prioritizes Markov chain entropy features for state-transition data: Shannon entropy, entropy rate, entropy production rate, von Neumann entropy (frequency/duration-based).

Models for Non-State-Transition Data

When tackling non-state-transition data (ESRD and PTBDB datasets), we selected CNN, LSTM, and CNN-LSTM as baseline models. To safeguard the meticulousness of the exper-

imental design while upholding the fairness and reliability of comparisons among different models, we opted to incorporate MLP, a model also rooted in deep learning, as the model of the InTS, paired with entropy-based features as input data. To enhance training efficiency and convergence quality, all models adopted a strategy blending temperature warm-up with cosine annealing.

Baseline. For non-state-transition data, we implemented three deep learning models as baselines to assess their performance across different approaches. First, we adapted a 1D-CNN model [112] for raw signal input, modifying its structure to fit our data characteristics. Next, for LSTM feature extraction, we adapted Farisi et al.'s model [113] by tuning parameters and architecture to fit the data dimension. Lastly, for CNN-LSTM analysis, we employed Hussain et al.'s hybrid model [114], leveraging CNN for local features and LSTM for long-term dependencies. For the input of the CNN-LSTM model, we utilized 5-level wavelet decomposition with single-level approximate DWT, transforming raw signals to time-frequency representations. To maintain experimental consistency and comparability, the loss functions and optimizers of these models were synchronized with those applied to the state-transition models.

InST. For non-state-transition data, we initiate the process by calculating a set of entropy-related features, such as increment entropy, approximate entropy, and slope entropy, each quantifying distinct aspects of data variation patterns. Subsequently, these features undergo a screening process using MI and PCC to eliminate redundancy and retain the most informative attributes, with the specifics of this procedure elaborated in Section 2.3.2. After feature extraction, an MLP model is integrated as the primary classification head due to its simplicity. Moreover, to facilitate a consistent and objective comparison with baseline models, the loss function and optimizer settings of the MLP are aligned with those employed by the baseline models.

Performance Metrics

To systematically and comprehensively evaluate the performance of these models, we designated accuracy, F1 score, recall, and the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) as the core evaluation metrics. The experiments are independently repeated ten

times with 'Mean ± Std'. Due to UK data protection regulations (including GDPR and the Data Protection Act 2018 [24]), which prohibited edge deployment during initial development, we used model parameter count as a hardware-agnostic proxy for system cost in experiments involving non-state-transition data. This is achieved by calculating the total parameter count for each model by iterating through the model layers in *PyTorch*.

2.5 Evaluation Results

InTS was tested on Minder (state-transition data) and ESRD/PTBDB (non-state-transition data).

2.5.1 Minder

The Minder dataset is state-transition data, where each distinct location is modeled as a node or state, and movements between these locations are interpreted as transitions. To account for the potential effects of sundowning and circadian rhythms in PLWD [115], we divided each day into two well-defined segments: daytime (06:00 to 18:00) and nighttime (18:00 to 24:00 and 00:00 to 06:00), enabling the detection of critical behavioral differences across these periods. Consequently, the baseline features consist of the average frequency and duration of various activities per week, calculated separately for both daytime and nighttime periods [111]. The model produces a binary classification output, indicating whether healthcare-related events have occurred (True or False), thereby facilitating the monitoring and evaluation of health conditions.

The entropy features extracted in InTS encompass several key measures: Shannon's entropy of Markov chains, entropy rate of Markov chains, entropy production rate of Markov chains, and von Neumann entropy of Markov chains derived from activity frequency and duration, each designed to quantify distinct facets of state-transition complexity and dynamics. Given that the dataset's label frequency is weekly, we compute these entropy metrics on a weekly cycle to align with the labeling structure. Specifically, Shannon entropy is first calculated daily and then aggregated by averaging across each week to obtain the weekly value. The entropy production rate is determined through a machine learning approach (as detailed in a Section. 2.3.2): for

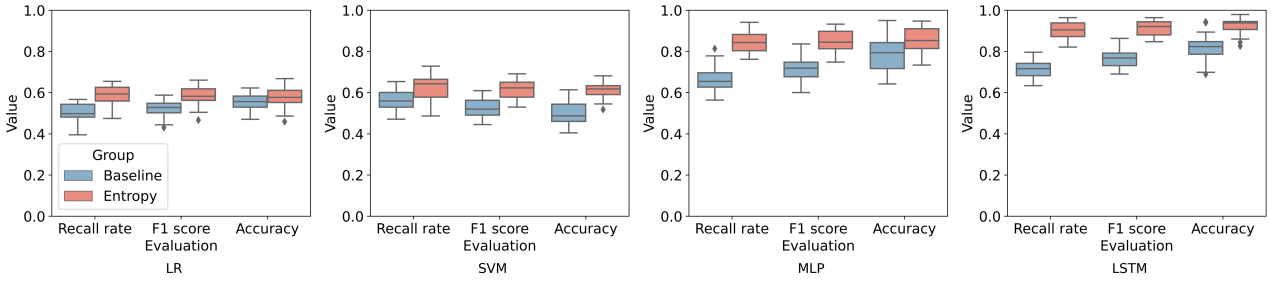


Figure 2.9: Evaluation results on the Minder dataset (x-axis: evaluation metrics)

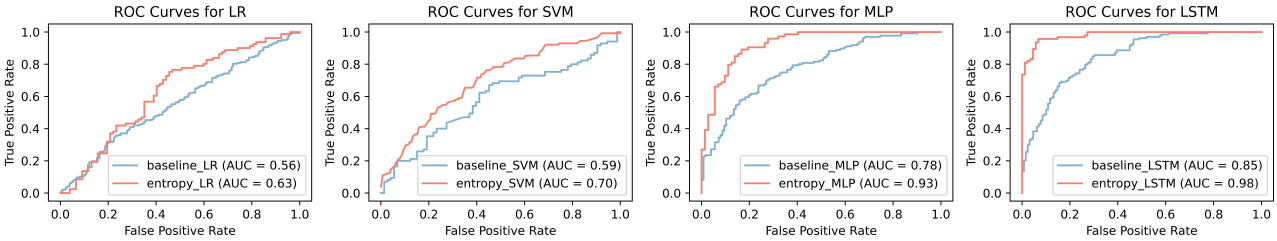


Figure 2.10: The ROC-AUC curves of the Minder dataset.

each participant, the model is trained on the training dataset, after which the weekly entropy production rate is computed based on the trained model's output. For the entropy rate, to capture stationary distributions and minimize the impact of seasonal fluctuations, we establish a four-month time window as the reference period, with the target analysis window set to a single day, followed by averaging to derive the weekly value. The von Neumann entropy is assessed by examining the frequency or duration of various activities within a day, subsequently consolidated into a weekly metric.

The evaluation outcomes for the Minder dataset are detailed in Fig.2.9, Fig.2.10, and Table 2.1. These findings illustrate that the features derived by InTS exhibit pronounced effectiveness and utility across a diverse set of models. The implementation of InTS markedly bolsters the predictive capability of linear models, such as LR and Linear Kernel SVM, enhancing their performance on state-transition data, while also delivering considerable advancements in non-linear models, including MLP and LSTM, underscoring its versatility across different modeling paradigms. The experimental analysis reveals that InTS contributes to a mean improvement of 13.08 percentage points (pp) in recall, 10.80 pp in F1 score, and 7.88 pp in accuracy.

InTS reveals more benefits within non-linear architectures such as LSTM and MLP. Specifically,

Table 2.1: The average performance of the models for the Minder.

	Evaluation	Baseline	Entropy	Improvement
LR	Recall rate	50.13±4.68%	58.97±4.61%	8.84pp
	F1 score	52.63±4.11%	58.46±4.42%	5.83pp
	Accuracy	55.52±4.13%	58.03±4.71%	2.51pp
SVM	Recall rate	56.05±4.60%	62.72±6.56%	6.67pp
	F1 score	52.82±4.75%	61.63±4.30%	8.81pp
	Accuracy	50.12±5.75%	60.94±4.03%	10.82pp
MLP	Recall rate	66.14±5.70%	84.16±5.21%	18.02pp
	F1 score	71.55±5.61%	84.97±4.97%	13.42pp
	Accuracy	78.37±8.11%	85.88±5.51%	7.51pp
LSTM	Recall rate	71.51±4.04%	90.29±4.41%	18.78pp
	F1 score	76.17±4.27%	91.29±3.72%	15.12pp
	Accuracy	81.72±6.41%	92.41±4.18%	10.69pp
Average	Recall rate	-	-	**13.08**pp
	F1 score	-	-	**10.80**pp
	Accuracy	-	-	**7.88**pp

Table 2.2: Comparison of ESRD classification results.

	Recall rate	F1 score	Accuracy
Baseline-CNN	91.18±1.04%	92.51±0.98%	93.90±1.07%
Baseline-LSTM	95.74±1.09%	95.12±0.54%	94.52±0.93%
Baseline-CNN-LSTM	96.30±1.35%	96.08±0.77%	95.88±0.53%
Entropy-MLP	97.53±0.74%	97.82±0.43%	98.12±0.67%
Avg Improvement	**3.12pp**	**3.25pp**	**3.35pp**

the LSTM model is improved in recall by 18.78 pp, F1 score by 15.12 pp, and accuracy by 10.69 pp. The MLP model is improved by 18.02, 13.42, and 7.51 pp in recall, F1 score, and accuracy, respectively. This improvement is attributable to the multi-layer processing capabilities and memory retention mechanisms intrinsic to non-linear models, which strengthen their proficiency in deciphering complex patterns and leveraging high-dimensional feature sets.

We also utilized ROC-AUC curves as an evaluation mechanism, as shown in Fig.2.10. The incorporation of entropy-based features resulted in improvements in AUC values across all models. In particular, the AUC for the LR model rose from 0.56 to 0.63, and the SVM model saw an increase from 0.59 to 0.70. The MLP model exhibited a leap, with its AUC value climbing from 0.78 to 0.93. Similarly, the LSTM model, upon integrating entropy features, attained an AUC of 0.98. These outcomes collectively verify the effectiveness of InTS.

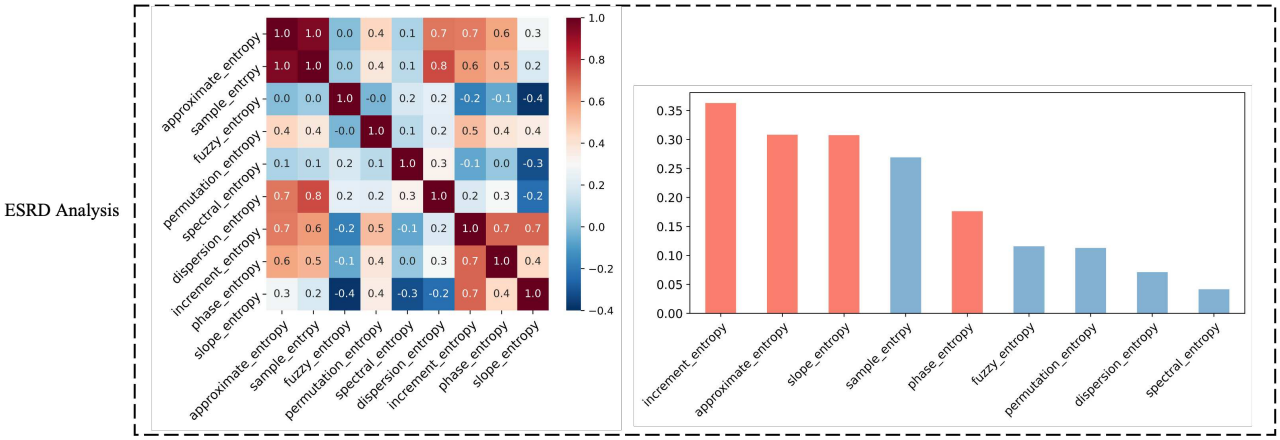
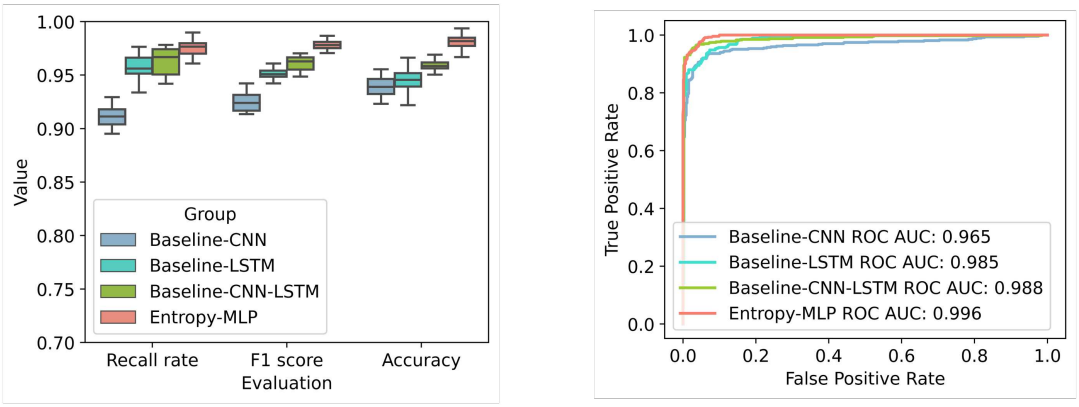


Figure 2.11: Feature selection results for ESRD: (left) Pearson correlation matrices, (right) mutual information with selected (red) and unselected (blue) features.

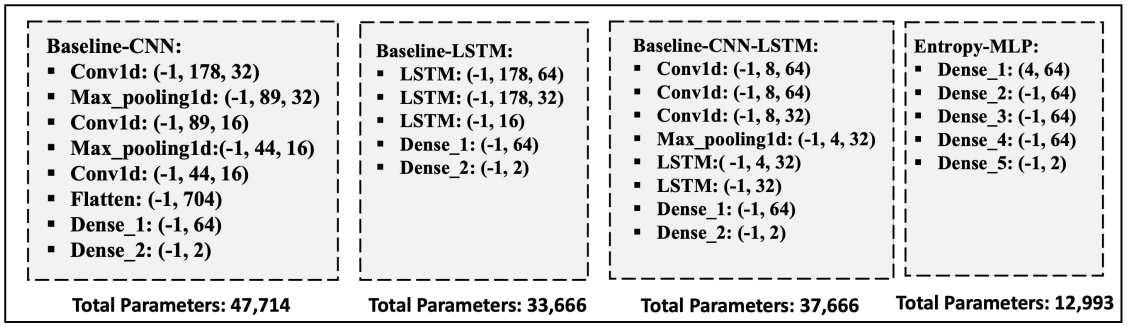
2.5.2 ESRD

ESRD constitutes a non-state-transition dataset. We designated CNN, LSTM, and CNN-LSTM models as baselines. For the CNN and LSTM models, the preprocessed data serves as the direct input to preserve essential signal information; in contrast, for the CNN-LSTM model, we augmented the process by employing discrete wavelet transform (DWT) to convert the signals into a time-frequency domain representation, enhancing feature extraction, and used these transformed signals as inputs. The ultimate output of the models is a binary classification, determining whether a participant has undergone an epileptic seizure.

For InTS, we commenced by calculating an array of entropy-related features, followed by a feature selection process employing MI and PCC to refine the feature set. As shown in Fig. 2.11, the initial MI assessment pinpointed increment entropy, approximate entropy, slope entropy, and sample entropy as the top four features. To address potential issues of feature collinearity and redundancy that could impact model performance, we performed a subsequent selection using PCC. The analysis indicated a high PCC value (exceeding 0.8) between sample entropy and approximate entropy, suggesting considerable correlation, with approximate entropy outperforming in MI scores; thus, we opted to retain approximate entropy. Following the MI ranking, we then incorporated phase entropy as an additional feature. A subsequent PCC check confirmed that the selected features exhibited no substantial collinearity or redundancy while preserving sufficiently high MI scores. Consequently, the final entropy-based feature set



(a) Evaluation results



(b) Model Structure

Figure 2.12: ESRD results: (a) Evaluation metrics (recall, F1, accuracy, ROC-AUC); (b) Model structure comparison (baseline vs. InTS/entropy). Due to pre-processed low-noise data, all models achieve ideal AUC-ROC performance.

for ESRD comprised increment entropy, approximate entropy, slope entropy, and phase entropy.

The experimental outcomes for the ESRD dataset, as presented in Fig. 2.12 and Table 2.2. Compared to the baseline models, InTS improved 3.12 pp in recall, 3.25 pp in F1 score, and 3.35 pp in accuracy. Furthermore, analysis of the ROC-AUC curve provides additional confirmation of InTS's classification performance. The parameter count of the InTS model was reduced by an average factor of 3.05, highlighting the efficiency InTS.

2.5.3 PTBDB

PTBDB constitutes a non-state-transition dataset. Consistent with the experimental framework established for the ESRD dataset, we designated CNN, LSTM, and CNN-LSTM models as baselines for performance evaluation. To maintain comparability and uniformity across experiments, the preprocessing procedures for PTBDB mirrored those applied in the ESRD experiments, encompassing data standardization and alignment. The ultimate output of these models is a binary classification, determining whether a participant exhibits heart disease.

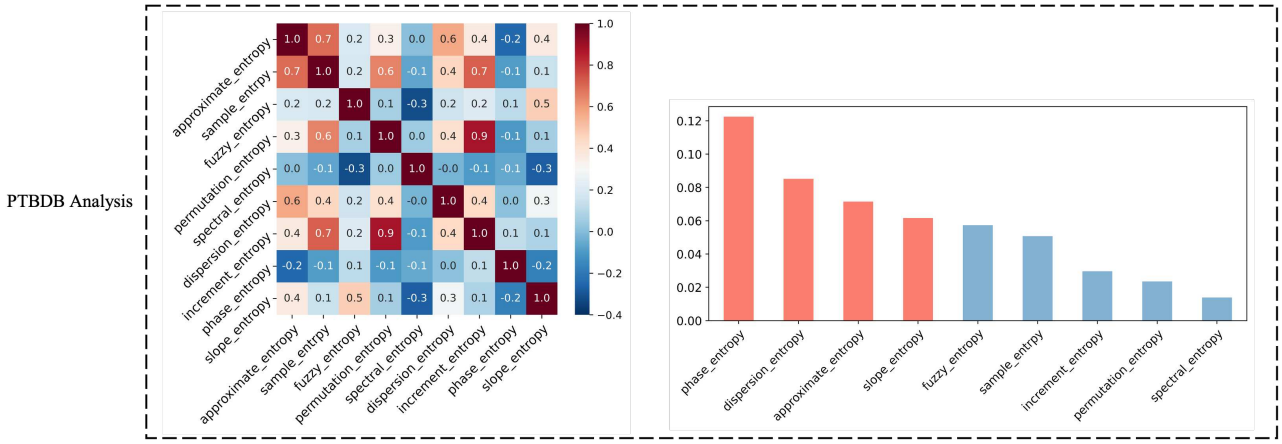


Figure 2.13: PTBDB feature selection results with Pearson correlation matrices (left) and mutual information (right, red=selected, blue=unselected).

Table 2.3: Comparison of PTBDB classification results.

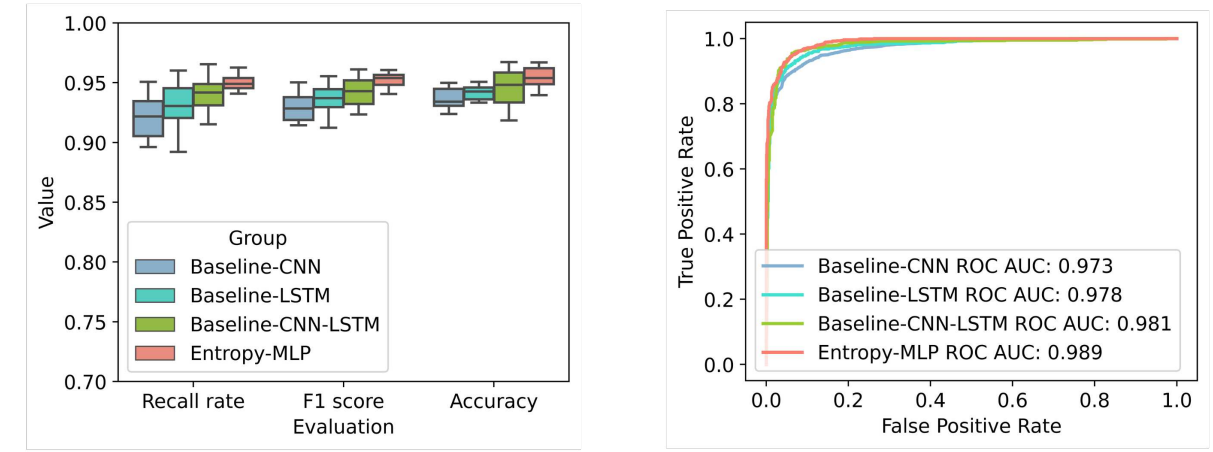
	Recall rate	F1 score	Accuracy
Baseline-CNN	92.16±1.83%	92.92±1.22%	93.70±0.88%
Baseline-LSTM	93.11±2.02%	93.61±1.27%	94.14±0.60%
Baseline-CNN-LSTM	94.07±1.57%	94.32±1.28%	94.59±1.58%
Entropy-MLP	95.02±0.76%	95.21±0.64%	95.42±0.99%
Avg Improvement	**1.91pp**	**1.59pp**	**1.28pp**

The results of the feature selection are shown in Fig.2.13. By initial selection through MI, we pinpointed phase entropy, dispersion entropy, approximate entropy, and slope entropy as the top four features. To further ensure the independence and utility of these features, a subsequent selection using the PCC was conducted, revealing that all chosen features maintained low correlation coefficients (below 0.8), indicating an absence of notable collinearity. As a result, we confirmed the aforementioned entropy features as the finalized entropy-based feature set for subsequent analysis.

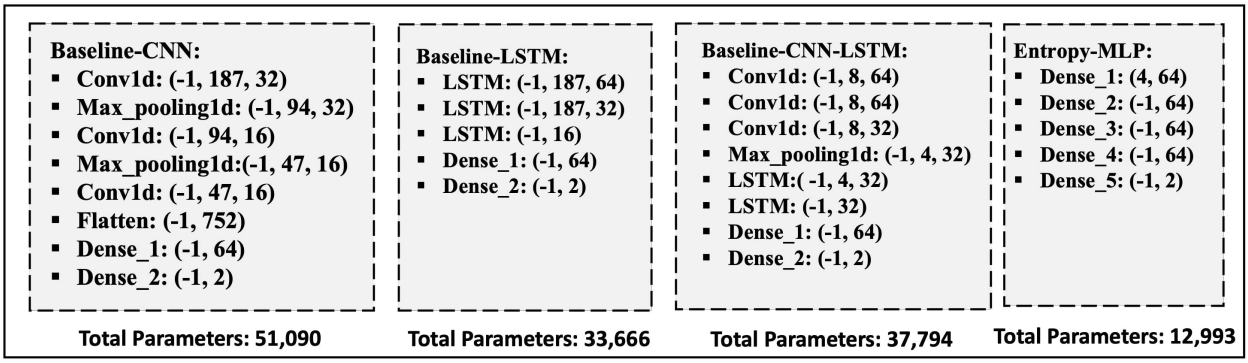
The experimental results are shown in Fig.2.14 and Table.2.3. InTS improved recall rate by an average of 1.91 pp, F1 score by 1.59 pp, and accuracy by 1.28 pp. Analysis of the ROC-AUC curves further corroborates its classification capability, aligning closely with the outcomes of SOTA approaches. Moreover, in comparison to baseline models, InTS reduced the parameter count by an average 3.14×, boosting computational efficiency.

Ablation Study

We conducted an ablation study on four Markov chain-based entropy features: Shannon entropy of Markov chains (SE), entropy rate of Markov chains (ER), entropy production of Markov



(a) Evaluation results



(b) Model Structure

Figure 2.14: PTBDB results: (a) evaluation metrics, (b) baseline vs. InTS/entropy model structures. With pre-processed low-noise data, all models achieve ideal AUC-ROC performance.

chains (EP), and von Neumann entropy of Markov chains (VNE). We take the LSTM model as an example because it performed best in previous experiments. Using a controlled variable approach, we utilize different groups of features (named G01-G14), then compared the accuracy against the full-feature model (G00). The results are shown in Table 2.4.

First, analyzing the three-feature groups (G01–G04), we observe that removing VNE (G01) resulted in the highest accuracy drop (to 87.69%), followed by EP (G02, 88.55%), ER (G03, 89.23%), while SE removal (G04) had the smallest impact (89.77%). This trend is further showed by the two-feature groups (G05–G10): The most severe performance degradation occurred when VNE was removed alongside features like EP (G05: EP+VNE absent, 82.51%), whereas combinations excluding SE (*e.g.*, G10: SE+ER absent, 86.12%) showed relatively minimal effects. These findings align with expectations: (1) VNE integrates both activity frequency and temporal information, and its weekly-scale comparison of daily variations miti-

Table 2.4: Ablation results of each entropy feature from InTS. '1' means with this entropy feature, '0' means without this entropy feature. SE means Shannon entropy of Markov chains, ER means entropy rate of Markov chains, EP means entropy production of Markov chains, VNE means von Neumann entropy of Markov chains.

Groups	SE	ER	EP	VNE	Acc	Groups	SE	ER	EP	VNE	Acc
G00	1	1	1	1	92.41±4.18%	G08	0	1	1	0	84.46±4.85%
G01	1	1	1	0	87.69±4.52%	G09	0	1	0	1	85.63±4.53%
G02	1	1	0	1	88.55±4.31%	G10	0	0	1	1	86.12±4.68%
G03	1	0	1	1	89.23±4.35%	G11	1	0	0	0	79.45±5.52%
G04	0	1	1	1	89.77±4.29%	G12	0	1	0	0	81.51±5.25%
G05	1	1	0	0	82.51±5.20%	G13	0	0	1	0	82.17±4.97%
G06	1	0	1	0	83.68±4.85%	G14	0	0	0	1	83.53±4.88%
G07	1	0	0	1	84.78±4.81%						

gates first-order Markov chains' inherent limitations in modeling long-range dependencies; (2) SE treats activities as independent events in its calculation, failing to capture path-dependent state transitions; (3) Both ER and EP leverage first-order Markov chains to model transition paths, while EP further accounts for path irreversibility. Single-feature validation (G11–G14) corroborates these conclusions: VNE alone achieved the highest accuracy (83.53%), while SE performed weakest (79.45%), exactly matching the importance ranking from the combinatorial experiments. Notably, the full-feature model (92.41%) outperformed all other feature groups, demonstrating that these features collectively quantify system uncertainty from complementary perspectives.

2.6 Discussions and Limitations

To validate the efficacy of our approach, we conducted experiments on three datasets. The results indicate that, compared to baseline models, InTS demonstrates performance improvements across multiple evaluation metrics and diverse dataset types. We attribute these improvements primarily to the following factors:

• **Effective High-Level Entropy Features.** The InTS system designs and refines a suite of entropy features to encapsulate essential statistical information and dynamic properties, thereby facilitating the derivation of highly expressive advanced features from raw data, enhancing the comprehension and analysis of intricate data patterns. For state-transition data, InTS preprocesses the raw data into the form of a first-order Markov chain to precisely delineate

state transition relationships, subsequently employing metrics such as entropy rate, entropy production rate, and von Neumann entropy, which collectively quantify the randomness and information generation processes of state transitions from diverse perspectives. For non-state-transition data, InTS leverages variant features like increment entropy and slope entropy, which are particularly adept at capturing local variations and trends within non-dynamic sequences, thus accommodating the analytical requirements of different data types.

• **Multi-Dimensional Entropy Feature Analysis.** The InTS system amalgamates a variety of entropy features, each meticulously crafted to uncover the intrinsic complexity and latent patterns of data from distinct yet complementary perspectives [116, 117]. By embracing this multi-dimensional analytical strategy, InTS achieves a deeper and more holistic insight into the dynamics and underlying structural properties of neural time-series data, capturing both temporal fluctuations and embedded characteristics. In contrast, conventional analytical techniques often concentrate on a solitary data feature or statistical measure, risking the omission of other vital informational dimensions and thus constraining the breadth of analysis. However, InTS's multi-dimensional entropy feature analysis, guarantees a comprehensive encapsulation of the data, yielding a more precise representation that lays a foundation for subsequent modeling and decision-making processes.

• **Efficient Feature Selection.** To proactively circumvent the detrimental effects of redundant and highly collinear features on model performance—such as overfitting or diminished predictive capacity—we devised and implemented a dual-feature-selection mechanism grounded in information theory. This mechanism synergistically integrates MI and the PCC matrix to systematically identify the optimal entropy features for subsequent model construction. MI evaluates the strength of the informational relationship between features and the target variable, ensuring the selection of features with substantial informational content, while the PCC matrix further examines inter-feature correlations to eliminate redundant or strongly collinear features, thereby enhancing the independence and utility of the resulting feature set.

• **Enhanced Model Simplicity through Efficient Entropy Features.** Conventional deep learning models typically demand extensive computational resources and prolonged training

durations to proficiently dissect intricate time-series data, ensuring the capture of deep-seated patterns and properties. In contrast, by thoughtfully designing and employing efficient entropy features, our approach substantially alleviates these requirements, enabling commendable analytical outcomes even with streamlined architectures such as LR and MLP. This strategy extracts pivotal statistical and dynamic information from the data, diminishing the reliance on elaborate model frameworks, thereby preserving performance while concurrently augmenting model simplicity and computational efficiency.

However, InTS also has certain limitations:

• This chapter employs a first-order Markov process to model patient behavior, a choice that can reduce computational complexity but inevitably comes with several limitations. First, a first-order Markov chain relies solely on the previous state, which may lead to the loss of critical information when handling behavioral patterns with long-range dependencies [118]. Additionally, first-order Markov chains are more susceptible to interference from cohabitants. Although von Neumann entropy based on Markov chains can partially mitigate these issues by measuring differences and uncertainties in activities across days, the limitations of first-order Markov chains persist. More complex higher-order models could improve accuracy but would significantly increase computational overhead. Future research could focus on developing adaptive order selection algorithms based on information criteria to dynamically balance computational cost and model complexity [119, 120].

• InTS employed the PCC threshold (default set at 0.8) in the feature selection algorithm to identify and remove highly linearly correlated redundant features. This value was chosen based on widely accepted practices in the feature selection domain, typically regarded as an empirical boundary for characterizing strong linear correlation[104, 105]. However, we fully recognize that fixing the PCC threshold at 0.8 as the default may not always be optimal or universally applicable when applied to a wide variety of tasks and datasets. Significant differences may exist in feature distributions, scales, noise levels, and the structure and strength of inter-feature correlations across different datasets. A fixed threshold that performs well on one dataset may lead to over-filtering or under-filtering on another dataset where inter-feature

correlations are generally stronger or weaker. In high-dimensional small-sample datasets, the likelihood of features exhibiting high PCC values by chance increases. A fixed 0.8 threshold may be more prone to removing non-redundant features (false positives), thereby affecting the quality of the final feature set. Although threshold adjustment was not triggered in our experiments, we reserved a dynamic threshold adjustment mechanism that allows the algorithm to automatically explore more relaxed thresholds to obtain sufficient features meeting low-redundancy requirements when the initial threshold is unsuitable. However, there may be better methods for setting the initial threshold that do not require dynamic adjustment, thereby improving efficiency.

• Integration of Multimodal Datasets. Currently, InTS employs parallel but independent processing workflows for state-transition and non-state-transition signals. While this design ensures system modularity, it also leaves space for research into multimodal fusion. Cross-modal correlations may contain critical diagnostic information. For instance, in Alzheimer's disease monitoring, studies have found that sleep disruptions in patients may be linked to abnormal health events (*e.g.*, wandering) [121], suggesting the potential for multimodal modeling of EEG/ECG signals and triggered sensor data. Similarly, in epilepsy monitoring, abnormal activity patterns during the prodromal phase (*e.g.*, repetitive activity) may correlate with changes in brain rhythms (*e.g.*, change on EEG signal) [122]. Multimodal analysis could provide earlier warning signals compared to single-modal analysis.

2.7 Conclusions

We propose InTS, an information-theoretic pipeline that extracts high-level features from neurological time-series data while improving model accuracy and efficiency. Validated across dementia care, epilepsy, and heart disease datasets, InTS demonstrates effectiveness and scalability. For state-transition data, InTS employs Shannon entropy, entropy rates, entropy production rate, and von Neumann entropy of Markov chains. For non-state-transition data, InTS utilizes various entropy variants. Additionally, InTS introduces a feature selection method based on the MI and PCC matrix. Compared to baselines, InTS improves recall rate, F1 score, and accuracy

by an average of 13.08 pp, 10.80 pp, and 7.88 pp, respectively. Evaluated against deep learning baselines on ESRD and PTBDB, InTS outperforms CNN, LSTM, and CNN-LSTM, enhancing accuracy, recall, and F1 score by an average of 2.52 pp, 2.42 pp, and 2.32 pp, respectively, and reduces model parameters by 3.10× on average. InTS enhances the adaptation of edge computing at the data level, enabling effective and efficient feature extraction when faced with state-varying and noisy time-series data.

Chapter 3

Model-Related Adaptations for Mobile Devices

This chapter addresses the challenge of model personalization under data drift in resource-constrained IoT edge environments. We utilize mobile devices (refer to Raspberry Pi in this chapter) as the experimental platform, since it has moderate computational resources and mature operating systems. Conventional approaches, such as full-model fine-tuning, incur excessive energy costs, while partial adaptation of only the final layers leads to insufficient personalization performance. To resolve this trade-off, we propose Target Block Fine-Tuning (TBFT), a lightweight adaptation method that selectively fine-tunes specific model blocks based on drift type. Our framework categorizes data drift into three types: input-level drift (front-block fine-tuning), feature-level drift (mid-block fine-tuning), and output-level drift (rear-block fine-tuning), enabling targeted optimization of critical modules for energy-efficient personalization. We provide a theoretical analysis of TBFT's effectiveness and validate it through experiments on ResNet architectures, three datasets, three training scales, and Raspberry Pi edge devices. We utilize public image datasets, including the CIFAR series and BREEDS/Living17 (a subset of ImageNet), since the medical datasets from Chapter 2 are subject to access restrictions for other research and image data can express the above three types of drift more simply and clearly. Compared to full-model fine-tuning, TBFT reduces energy consumption by 41.57% on average, while outperforming the averaged block-wise fine-tuning baseline ($BlockAvg$) by 15.30% in average accuracy.

3.1 Introduction

Data drift is one of the significant challenges for on-device ML [123, 124]. as data drift between source (cloud) and target (local) domains severely degrades local model performance (*e.g.*, accuracy). IoT/mobile devices' limited computational resources, storage, and battery life impose strict requirements on on-device ML for memory usage, power control, and energy efficiency [125, 126]. Therefore, how to achieve efficient and stable personalized ML models on resource-constrained devices has become an important direction in current research.

To address these challenges, researchers have proposed several solutions. One approach improves the base model's domain generalization on source data, enhancing adaptation to diverse distributions [127, 128]. Another approach involves fine-tuning the base model on labeled target data, which significantly improves the model's performance in practical applications [129, 130, 131]. Studies have shown that, in terms of performance metrics such as accuracy, fine-tuning methods often outperform domain generalization and unsupervised adaptation approaches, as fine-tuning can more directly leverage the characteristics of the target domain data to optimize the model [132, 133].

Currently, the main strategy in fine-tuning methods focuses on optimizing either the entire model or its final layers [37, 134, 135]. For example, large models typically use a general front-end feature extractor [38], adapting only the classifier (*e.g.*, FC layer) for new tasks. However, full fine-tuning increases energy and hardware demands, while last-layer tuning often underperforms. Some methods employ search strategies to identify and fine-tune only the most important model parameters, but if such search strategies are run locally on resource-constrained devices, they result in significant energy consumption; if run in the cloud, they introduce additional privacy risks [136, 137, 47]. Data drift can be categorized into three types: input-level, feature-level, and output-level [138, 139]. The ICM principle [140, 141] states that causal generative processes comprise independent, non-interfering modules. Therefore, different data drift types reflect localized causal process changes, suggesting full-model or last-layer fine-tuning often suboptimal for diverse drift scenarios. This prompts us to consider whether targeted fine-tuning of modules associated with specific types of data drift on local

devices can keep satisfied model performance (*e.g.*, accuracy) while reduce energy cost.

Contributions. We propose Target Block Fine-Tuning (TBFT) [1], a low-energy adaptive personalization framework specifically designed for resource-constrained devices, aiming to address data drift issues and achieve model personalization. Specifically, we categorize data drift into three types: input-level, feature-level, and output-level, and TBFT adaptively fine-tunes the front blocks, middle blocks, and rear blocks of the model based on the type of data drift. Here, 'data drift' refers to the significant differences in statistical properties between the source data and the target data [142], while a 'block' represents a fundamental building unit of the model, which may encompass multiple layers (*e.g.*,a combination of convolutional and pooling layers) or a single layer (*e.g.*, a fully connected layer).

(1) TBFT adaptively fine-tunes various model blocks/modules (front, middle, and rear) depending on the nature of data drift (input-, feature-, or output-level).

(2) We establish the block-drift correspondence through: (i) mathematical analysis proving that each drift type (input/feature/output) primarily affects blocks at specific depths (Section. 3.3), and (ii) experiments where fine-tuning only the corresponding blocks (*e.g.*, front blocks for input-level drift) achieves the best accuracy (the brief summary of the experimental results is given below).

(3) To comprehensively validate the effectiveness and robustness of TBFT, we conducted experiments on a ResNet model using three different datasets and deployed it on a Raspberry Pi for testing. Additionally, considering the efficiency of local fine-tuning and the scale limitations of local datasets, we tested TBFT under three different training sizes (10%, 20%, and 30% of the full training dataset) to verify its stability across varying data volumes.

(4) TBFT outperforms *BlockAvg* (individual block fine-tuning with averaged improvements) by 15.30% accuracy.

(5) Compared to full model fine-tuning, TBFT saves an average of 41.57% in energy consumption, reducing resource usage.

[1]The code, models, and datasets are available at:
https://github.com/yushan-huang/AdaptivePersonalization

This chapter is related to the following paper:

Yushan Huang, Josh Millar, Yuxuan Long, Yuchen Zhao, Hamed Haddadi. 'Towards Low-Energy Adaptive Personalization for Resource-Constrained Devices'. Accepted to The 4th Workshop on Machine Learning and Systems EuroMLSys 2024, co-located with EuroSys'24.

3.2 Background

3.2.1 Data Drift

We introduce the definition of three drifts, including input-, feature-, and output-level [138, 139].

Input-Level Drift

Input-level drift is a typical form of covariate shift, where the target domain input distribution $P_{\text{trg}}(X)$ is generated by applying deterministic linear transformations to the source input X_{src}, while maintaining label consistency $Y_{\text{trg}} = Y_{\text{src}}$. Specifically, given a source input $x_{\text{src}} \in \mathbb{R}^n$, the target input can be expressed as $x_{\text{trg}} = Mx_{\text{src}}$, where the invertible matrix $M \in \mathbb{R}^{n \times n}$ encodes linear perturbations. These transformations can be uniformly modeled as linear projections in pixel space, encompassing common image modifications. For example, color channel mixing adjusts an image's RGB composition by applying a fixed linear transformation to the color space. Similarly, global brightness or contrast scaling multiplies all pixel intensities by a constant factor, which is also a linear transformation in pixel space.

Feature-Level Drift

Feature-level drift is a fundamental challenge in domain adaptation, characterized by preserved class semantics but divergent feature distributions within each class. Specifically, while source and target domains share identical label spaces, each class comprises distinct subpopulations across domains. For instance, in the Living17 dataset [143], the 'vegetables' class in the source domain may include carrots and broccoli, whereas the target domain contains onions and spinach. This shift arises from intra-class heterogeneity rather than global input/label transformations.

Suppose $Y_{src} = Y_{trg}$, feature-level drift satisfies:

$$D_{\mathrm{KL}}(P_{\mathrm{src}}(X_{src}|Y_{src})\|P_{\mathrm{trg}}(X_{trg}|Y_{trg})) > 0 \tag{3.1}$$

where D_{KL} is the Kullback-Leibler divergence.

Compared with input-level drift, feature-level drift exhibit fundamental differences in origin and mathematical characterization. Input-level drift stems from transformations in the **raw input space**, where target samples x_{trg} are generated through linear operations ($x_{\mathrm{trg}} = Mx_{\mathrm{src}}$) that preserve class-discriminative features but alter low-level signal properties. In contrast, feature-level drift arises from shifts in the **semantic space**, where conditional distributions $P(X|Y)$ diverge between domains despite identical label spaces Y. This distinction is quantifiable through their perturbation scopes: input-level drift applies content-agnostic transformations (e.g., brightness changes affect pixels), while feature-level drift introduces content-dependent variations (e.g., intra-class diversity in subcategories). The linearity of input-level drift ensures its invertibility ($\exists M^{-1}$), whereas feature-level drift requires non-linear alignment due to heterogeneous semantic manifestations.

Output-Level Drift

Output-level drift is a typical form of concept drift, where the target domain input distribution remains consistent with the source domain ($P_{trg}(X_{trg}) = P_{src}(X_{src})$), but the label conditional distribution undergoes systematic shifts ($P_{trg}(Y_{trg}|X_{trg}) \neq P_{src}(Y_{src}|X_{src})$). Specifically, given a source input-label pair (x_{src}, y_{src}), the target label can be modeled as a deterministic transformation $y_{\mathrm{trg}} = M(y_{\mathrm{src}}; \theta)$, where M is linear perturbations. Typical examples include linear scaling, classification threshold adjustments (*e.g.*, changes in positive criteria for medical diagnosis), or ordinal label remapping (*e.g.*, revisions to user rating scales).

Unlike input-level shifts, correcting output-level shifts does not require intervention in the feature extraction process. When M is a linear invertible transformation, adjusting only the weights of the final classifier $\theta^{(L)}$ is sufficient to fully compensate for the label shift, whereas modifying the weights of the first layer $\theta^{(1)}$ provides no explicit improvement to the target loss

$\mathcal{L}_{\text{trg}}$ [139]. This phenomenon arises because label perturbations are independent of the input feature space, and their correction signals can be directly propagated through the output layer.

Research in this area holds universal value for multi-domain applications. For instance, in medical imaging analysis, differences in pathological staging criteria across hospitals need to be modeled as label shifts [144]; in natural language processing, discrepancies in sentiment polarity annotations between social media and formal texts also fall into this category [145]. By analyzing the mathematical mechanisms of output-level drift, a theoretical foundation can be established for designing adaptive classification algorithms.

3.2.2 Transfer Learning

Transfer Learning (TL) leverages pre-trained models to accelerate new task learning, reducing training time and data needs through source-to-target knowledge transfer. Research has shown that freezing certain layers or parameters during fine-tuning can effectively mitigate overfitting [146, 147, 148, 149] and enhance adaptation efficiency [150, 151, 152], thereby significantly improving TL performance across various application scenarios. However, despite extensive efforts to analyze the adaptability of model layers to different types of data drift [153, 154], most existing methods pre-select fixed layers (usually the last few layers) for fine-tuning across all types of data drift.

Unlike these traditional methods, TBFT proposes a more flexible and adaptive model fine-tuning strategy. Instead of using fixed layers for all types of data drift, TBFT selects different model blocks for fine-tuning based on the specific type of data drift. For example, fine-tuning intermediate layers for feature-level drift and front layers for input-level drift.

3.2.3 Personalization

A key deployment challenge is personalizing pre-trained models to target domain distribution shifts. This need for personalization arises from the diversity and dynamics of real-world data distributions, such as behavioral differences among user groups, systematic biases in device sensors, or variations in environmental conditions. To address this challenge, both academia and industry have proposed numerous strategies, including but not limited to domain adaptation,

transfer learning, and online learning.

Research has shown that for certain types of domain shifts, particularly output-level shifts (*i.e.*, changes in label distribution), fine-tuning only the final layer of the model is often sufficient to achieve good performance [155]. For example, in tasks with spurious correlations (*e.g.*, differences in annotation standards across hospitals in medical diagnosis), adjusting the weights of the classifier's output layer can effectively mitigate inconsistencies in label distribution. However, for more complex shift types (*e.g.*, input-level or feature-level shifts), fine-tuning only the final layer may not be sufficient to achieve optimal performance. Lee et al. further discovered that different model blocks exhibit significant differences in sensitivity to these types of shifts [139]. For instance, shallow networks are more sensitive to input-level shifts, while deeper networks are more sensitive to feature-level and output-level shifts. These findings motivated our development of block-specific fine-tuning for different data drifts. We further validated its stability on small local datasets, crucial for edge computing and privacy-sensitive scenarios with limited target-domain data.

3.2.4 Efficient Learning

Developing efficient machine learning models and reducing their size and inference energy consumption has become a critical research focus for resource-constrained IoT and mobile devices with limited energy availability [156, 157]. These devices often face strict computational and energy constraints, necessitating a balance between model performance and resource consumption. To address this, researchers have developed efficiency optimization techniques including pruning, quantization, knowledge distillation, and energy-aware architecture search [158, 159, 160]. These methods significantly enhance the applicability of models in resource-constrained environments by reducing model parameters, lowering computational complexity, or optimizing hardware utilization.

In this chapter, we explore efficient learning from a novel perspective: TBFT achieves sufficient performance by adjusting only a sub-block of the model, thereby significantly reducing the energy cost required for personalization. Unlike traditional global fine-tuning approaches, according to different data drift types, TBFT optimizes only related modules, ensuring model

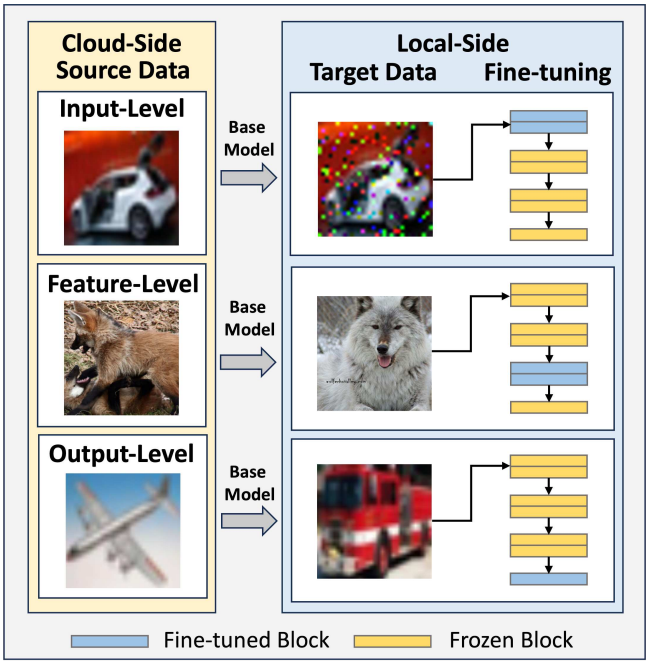


Figure 3.1: TBFT overview.

accuracy while drastically reducing computational overhead. This approach excels in resource-constrained scenarios like edge computing and mobile real-time inference.

Furthermore, TBFT can be seamlessly integrated with various existing efficient learning techniques. For example, combining TBFT with quantization can further compress model size and reduce energy consumption; integrating it with knowledge distillation allows lightweight student models to approximate the behavior of complex teacher models; and incorporating pruning techniques can eliminate redundant parameters, thereby reducing computational load.

3.3 Methodology

3.3.1 Overview

Assumptions. We focus on image data with two key scenarios: (1) Cloud-trained base models face local data drift (input-/feature-/output-level), with the drift types being independent; (2) Energy-sensitive devices (*e.g.*, [161]); (3) The drift types are known, we leave the drift detection methods as an open research question, which is discussed in Section. 3.6.

TBFT (as shown in Fig. 3.1), which initially trains a base model on cloud source data and then fine-tunes targeted sub-blocks on local devices to address specific data drift. The model h

is structured as a stacked architecture of modular blocks, expressed as $h = h_n \circ \ldots \circ h_1$, where each block h_i incorporates a parameter set θ_i and multiple neural layers. The TBFT framework fine-tunes specific blocks B and their parameters $\{\theta_i | i \in B\}$ corresponding to local drifts to minimize the task-specific loss $\hat{\mathcal{L}}_{tgt}$. The optimization objective is:

$$\min_{\theta_i, i \in S} \hat{\mathcal{L}}_{tgt}(h(\theta_1, \ldots, \theta_n)), \tag{3.2}$$

where non-selected blocks ($i \notin B$) remain frozen. For instance:

- Front blocks ($B = \{1\}$) adapt to input-level variations
- Final block ($B = \{n\}$) handles output-level drift

When $B = \{1\}$, the target model becomes [139]:

$$h^{tgt} = h_n \circ \cdots \circ h_2 \circ h_1^{tgt} \tag{3.3}$$

3.3.2 Adaptive Personalization

We provide mathematical theoretical analysis that each drift type is more closely associated with specific depth blocks (front/middle/rear) in the model, and explain the definitions of sub-blocks.

Input-Level

Input-level drift occurs when target inputs x_{trg} are linear transforms of source inputs x_{src}. For invertible $M \in \mathbb{R}^{n \times n}$:

$$x_{trg} = Mx_{src}, \quad y_{trg} = y_{src} \tag{3.4}$$

where $x_{src} \in \mathbb{R}^n$ and $x_{trg} \in \mathbb{R}^n$ are input vectors, y_{src}, y_{trg} denote labels, and M represents linear transformation matrix.

Consider a source model $h_{src} = h_{\neq 1,src} \circ h_{1,src}$ with first-module $h_{1,src}(x_{src}) = \theta_{1,src}x_{src}$ and the rest of modules $h_{\neq 1,src}$. Source optimality requires:

$$h_{\neq 1,src}(\theta_{1,src}x_{src}) = y_{src}, \quad \forall (x_{src}, y_{src}) \sim P_{src} \tag{3.5}$$

For target domain adaptation, if we only fine-tune the first-module and keep the rest of modules frozen, which means $h_{\neq 1,src} = h_{\neq 1,trg}$, so we abbreviate them as $h_{\neq 1}$. We enforce output equivalence through:

$$\begin{aligned} h_{\neq 1}(\theta_{1,trg}x_{trg}) &= h_{\neq 1}(\theta_{1,trg}Mx_{src}) \\ &= h_{\neq 1}(\theta_{1,src}x_{src}) = y_{src} = y_{trg} \end{aligned} \tag{3.6}$$

The critical weight adaptation follows from:

$$\theta_{1,trg}Mx_{src} = \theta_{1,src}x_{src} \quad \forall x_{src} \implies \theta_{1,trg} = \theta_{1,src}M^{-1} \tag{3.7}$$

This demonstrates that input-level drifts can be *fully compensated* through first-layer linear inversion, with zero target loss ($\mathcal{L}_{\text{trg}} = 0$) guaranteed by matrix invertibility.

However, without fine-tuning the first module, the error can not be compensated. Consider the counterexample where the transformation matrix M is set to the negative identity matrix $-I$, with the source-domain first-module activations $\theta_{1,src}x_{src}$ strictly positive for all inputs x_{src}:

$$M = -I, \quad \text{with } \theta_{1,src} x_{src} > 0 \tag{3.8}$$

Under this transformation, the target-domain features exhibit sign reversal through the linear projection $\theta_{1,src} x_{trg} = -\theta_{1,src} x_{src}$, followed by complete activation collapse when passing through the ReLU nonlinearity $\phi(z) = \max(z, 0)$. This occurs because all negative pre-activations are zeroed out:

$$\theta_{1,src} x_{trg} = -\theta_{1,src} x_{src}, \tag{3.9}$$

$$\phi(\theta_{1,src} x_{trg}) = \max(-\theta_{1,src} x_{src}, 0) = 0, \tag{3.10}$$

The information loss propagates through subsequent layers $\theta_{\neq 1}$, resulting in degenerate model predictions $h(x_{trg}; \theta_{1,src}, \theta_{\neq 1}) = 0$ that mismatch the target labels y_{trg} (which equal y_{src} by our input-drift assumption). This demonstrates that deeper module adaptation cannot recover from first-module mismatches.

Output-Level

Output-level drift occurs when target outputs y_{trg} are linear transforms of source outputs y_{src} with identical inputs. For invertible matrix M:

$$x_{trg} = x_{src}, \quad y_{trg} = M y_{src} \tag{3.11}$$

Consider an L-layer model $h = h_{\neq L} \circ h_L$, where $h_{L,src}(x_{src}) = \theta_{L,src} x_{src}$ is the last module, and $h_{\neq L}$ is the rest modules. Source optimality requires:

$$h_{\neq L,src}(\theta_{L,src} x_{src}) = y_{src}, \quad \forall (x_{src}, y_{src}) \sim P_{src} \tag{3.12}$$

For target adaptation, construct $\theta_{L,trg} = \theta_{L,src}M^{-1}$ while freezing $h_{\neq L}(h_{\neq L,src} = h_{\neq L,trg} = h_{\neq L})$:

$$\begin{aligned} h(x_{trg}) &= h_{\neq L}(\theta_{L,trg}x_{src}) \\ &= h_{\neq L}(\theta_{L,src}M^{-1}x_{src}) \\ &= My_{src} = y_{trg} \end{aligned} \tag{3.13}$$

This shows that output-level drift can be fully compensated through last-layer fine-tuning, achieving $\mathcal{L}_{trg} = 0$ when M is invertible.

However, adapting front-end modules ($\theta_{\neq L}$ denoting parameters before the last modules) faces intrinsic limitations. Consider the sign-flipping case with $M = -I$ (I being identity matrix), where:

$$y_{trg} = -y_{src}, \quad \text{with } \phi(\theta_{L,src}x_{src}) > 0 \tag{3.14}$$

The non-negativity of ReLU-activated ϕ features imposes:

$$\phi(\theta_{\neq L}x_{trg}) \geq 0, \tag{3.15}$$

$$\theta_{L,src}\phi(\theta_{\neq L}x_{trg}) \geq 0 \neq y_{trg} = -y_{src} \tag{3.16}$$

regardless of front-end adjustments. This leads to irreducible error:

$$\min_{\theta_{\neq L}} \mathcal{L}_{trg} \geq \epsilon > 0 \tag{3.17}$$

This demonstrates that for output-level drift, last-layer adaptation (θ_L) is strictly necessary.

Feature-Level

Consider a model $h = h_L \circ h_m \circ h_1$, where h_1 is the first module, h_m represents the middle modules (from module 2 to $L-1$, has nonlinear capabilities), h_L is the final module.

To address feature-level drift, we align the feature distributions by minimizing the maximum mean discrepancy (MMD):

$$\min_{\theta_m} D_{MMD}\left(h_{m,src}(x_{src}), h_{m,trg}(x_{trg})|y\right) \tag{3.18}$$

When the output of $h_{m,src}(x_{src})$ and $h_{m,trg}(x_{trg})$ becomes domain-invariant, the model maintains consistent performance (*e.g.*, accuracy) across domains, achieving $\mathcal{L}_{\text{trg}} \to 0$.

However, adapting only the first or last layer is insufficient. Consider a single class y with different subpopulations in source and target domains:

$$X_{\text{src}}|y \sim \tfrac{1}{2}\mathcal{N}(\mu_1, \Sigma) + \tfrac{1}{2}\mathcal{N}(\mu_2, \Sigma), \quad X_{\text{trg}}|y \sim \tfrac{1}{2}\mathcal{N}(\mu_3, \Sigma) + \tfrac{1}{2}\mathcal{N}(\mu_4, \Sigma), \quad \{\mu_1, \mu_2\} \cap \{\mu_3, \mu_4\} = \emptyset. \tag{3.19}$$

This models a feature-level drift where the same class label is associated with distinct subclusters across domains (*e.g.*, 'vegetables' contain carrots and broccoli in source but onions and spinach in target).

First-Module Failure. If the first module h_1 is a linear or affine transformation shared across classes, it cannot simultaneously align multiple subclusters that differ between source and target. A single global mapping can at best align means or global statistics, but fails to resolve the mismatch of multiple disjoint modes within each class. Therefore, without deeper non-linear modules to perform cluster-wise matching, the subclusters remain misaligned in the shared feature space.

Last-Module Failure. Adapting only the final classifier h_L is insufficient because the upstream features $h_m(h_1(X_{\text{trg}}))$ still preserve the subcluster discrepancy. Since the target subclusters occupy distinct regions that do not overlap with the source subclusters, a single linear decision boundary cannot correctly classify both domains simultaneously. Consequently, even

an optimal classifier h_L yields systematic prediction errors for misaligned target subclusters:

$$\mathbb{E}_{X_{\text{trg}}|y}\big[h_L(h_m(h_1(X_{\text{trg}})))\big] \neq y. \tag{3.20}$$

3.3.3 Definition of Models and Sub-Blocks

We consider neural architectures with a modular, hierarchical structure, such as ResNet or SqueezeNet, where the network is composed of sequential blocks with increasing representational capacity. To enable targeted adaptation, we segment the model into three functional regions: **front**, **middle**, and **rear** blocks.

The *front blocks* include the initial convolutional stem (*e.g.*, conv1) and the first basic block. These layers primarily extract low-level visual patterns such as edges and textures. The *rear blocks* primarily consist of the classifier head, which aggregate semantic representations. The front and rear blocks exhibiting primarily linear behavior with limited nonlinear capacity [**?**].

The *middle blocks* refer to intermediate blocks between the front and rear blocks. These blocks represent a gradual semantic transition—from local patterns to high-level structures—and are crucial for capturing intra-class variations. Notably, the number of middle blocks varies with model depth: in shallow networks (*e.g.*, 4 blocks in total), they may only include the 2nd and 3rd blocks, while in deeper architectures (*e.g.*, 10 blocks in total), they span a wider range, each encoding progressively abstract features. We did not delve deeper into the specific division and determination of middle blocks in this chapter. We leave this as a potential direction for future research and discuss it in Section. 3.6.

3.4 Implementation & Evaluation Setup

This section covers the experimental model, datasets, implementation, and performance metrics.

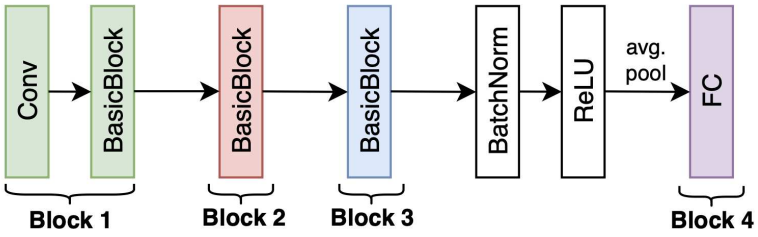


Figure 3.2: The ResNet-26 model.

3.4.1 Models and Datasets

We utilize the ResNet-26 architecture [162], which is visually represented in Figure 3.2. The decision to adopt this model stems from its distinctive residual block configuration, allowing the model to be seamlessly integrated with the TBFT methodology without necessitating any modifications to its foundational design, thereby simplifying the experimental setup. Additionally, the model can be simply divided into four blocks, facilitating intuitive validation of TBFT's effectiveness while avoiding the impact of having too many middle blocks (as mentioned in Section. 3.3.2). We train the model from scratch on source data, ensuring independent and reproducible results. To achieve effective optimization, we employ the Adam optimizer. Additionally, we implement an early-stopping technique guided by the model's accuracy on a separate validation set, a strategy designed to curb overfitting and bolster generalization. During target-domain fine-tuning, we set the learning rate to 0.01 for the fully connected layer (enabling rapid task adaptation) and 0.001 for other modules. Remaining hyperparameters vary by experiment, as detailed in Section 3.5.

We evaluate TBFT across three types of datasets, each aligned with a specific type of data shift, to explore its effectiveness under varying conditions. To mimic scenarios with constrained local data, we configure the training portions to represent 10%, 20%, and 30% of the full training dataset, enabling an analysis of TBFT's performance with limited resources. The validation and testing sets are consistently set to 10% of the total data, ensuring a standardized evaluation framework.

Input-Level: Cifar10/Cifar10-C. The source data is derived from the Cifar10 dataset, featuring 32×32 color images categorized into 10 distinct classes. The target data, sourced from Cifar10-C, comprises degraded versions of images from these same classes, with degradation

severity spanning five levels, indicative of an input-level shift. This dataset encompasses 14 unique corruption forms, such as frost effects, representing diverse real-world image deterioration conditions. For experimental purposes, we conduct training and evaluation separately on each corruption type and present the aggregated performance across all types. Following the data preparation methodology outlined in [163], we restrict our training and assessment to samples exhibiting the maximum severity, such as corruption level 5, to emphasize model behavior under the most challenging circumstances.

Feature-Level: Living17. This dataset emerges from the BREEDS benchmark (from ImageNet), designed to investigate subpopulation shifts as introduced in [143]. Within the BREEDS framework, subpopulations from ImageNet are categorized using the hierarchical structure of WordNet, enabling the creation of diverse subpopulation-specific datasets. The Living17 dataset, in particular, encompasses images of living entities organized into 17 primary classes, with each class subdivided into four distinct subclasses. The source and target distributions are constructed from different subsets of these subclasses, resulting in a notable distinction between them. This distinction manifests as a feature-level shift, characterized by variations in feature distributions across the source and target domains.

Output-Level: Cifar10/Cifar-Flip. We systematically transform the class labels of the Cifar10 dataset such that each original label y is reassigned to $9 - y$ [139]. For instance, a label of 0 is converted to 9, a label of 1 becomes 8, and this pattern continues accordingly, fundamentally altering the label distribution. The source data remains the unaltered Cifar10 training set.

3.4.2 Prototype and Performance Metrics

We assess TBFT using a Raspberry Pi 4 Model B equipped with 2GB RAM, a device frequently selected as a representative of resource-constrained systems due to its modest hardware capabilities, ideal for testing in low-power contexts. The Pi 4 features a Broadcom BCM2711 processor, a 64-bit quad-core Cortex-A72 (ARM v8) system-on-chip running at 1.5GHz, which strikes an effective balance between processing power and energy conservation, enhancing its suitability for practical deployments. The system is supported by 2GB of LPDDR4-3200 SDRAM.

Table 3.1: Noised block fine-tuning accuracy (best performance highlighted).

Tuned \ Added Noise	Block 1	Block 2	Block 3	FC
Block 1	84.88±0.23	69.24±0.40	43.44±0.38	36.30±0.41
Block 2	72.38±0.37	83.74±0.21	70.54±0.32	67.86±0.35
Block 3	56.86±0.37	74.95±0.27	83.04±0.29	83.85±0.39
FC	30.58±0.24	46.94±0.23	51.82±0.41	84.28±0.33
Block Avg	61.18±0.30	68.72±0.28	62.21±0.35	68.07±0.37
Full	83.56±0.28	82.87±0.34	82.14±0.34	83.58±0.42

We assess TBFT's accuracy and operational efficiency under different training data scales. To measure system cost, we utilize the Power Monitoring HAT for Raspberry Pi, focusing on three key metrics: (1) Runtime (measured in seconds), captured via the device's internal clock; (2) Memory consumption (in GB), tracked internally by the system; (3) Energy cost (in joules), computed by power and time. Additionally, we assess energy savings (ES) with the equation $ES = (EB - EF)/EF$, where EB signifies the energy consumed during block-based fine-tuning and EF corresponds to full model fine-tuning, providing a comparative measure of energy efficiency. Our evaluations cover both 32×32 (Cifar10-C, Cifar-Flip) and 128×128 (Living17) resolutions. To accommodate the memory limitations of the Raspberry Pi, we configure the batch size to 1 during system cost assessments, ensuring experimental feasibility.

3.5 Results

We conducted a series of experiments to validate the relationship between different drift types and various blocks, specifically examining whether fine-tuning the corresponding blocks yields optimal classification accuracy for each drift type. Additionally, we measured the memory, power, and energy consumption required for fine-tuning.

3.5.1 Motivation Experiments

To determine whether fine-tune distinct model blocks/modules can adequately tailor the model to various data drifts, we conduct a motivation experiment by injecting normally distributed noise into several model sections, including the final FC layer. This approach aims to probe the model's responsiveness to localized drifts. We implement this on a ResNet-26 framework [162], which is trained from scratch on the Cifar-10 dataset [164]. Subsequently, we fine-tune selected blocks of the network using Cifar10 target data, while immobilizing the remaining parameters

Table 3.2: Overall results (accuracy, energy cost E, and saving rate ES compared with full tuning). $I-L$, $F-L$, $O-L$ denote input/feature/output-level drifts; $BlockAvg$ shows block-tuning average (excludes full model). Best accuracy for per drift are highlighted.

	Block1	Block2	Block3	FC	Block Avg	Full
I-L Acc (%)	75.20±0.36	70.98±0.33	70.62±0.21	68.74±0.37	71.40±0.26	75.09±0.28
F-L Acc (%)	48.63±0.37	57.61±0.42	68.31±0.43	60.06±0.29	58.65±0.35	67.31±0.41
O-L Acc (%)	21.25±0.29	38.69±0.38	64.27±0.30	84.74±0.35	52.29±0.29	61.11±0.41
E (J)	0.898	0.708	0.564	0.325	0.624	1.068
ES (%)	15.92	33.71	47.19	69.57	**41.57**	-

by freezing their weights, thus isolating the impact of targeted block/module adjustments and minimizing interference from broader network alterations.

Table 3.1 presents the outcomes of our motivation experiments, offering evidence to substantiate our hypothesis. The findings highlight a significant trend: selectively fine-tuning only the block perturbed with noise yields superior performance compared to fine-tuning other blocks, and it either matches or surpasses the results of full model fine-tuning. TBFT yields 18.94% accuracy increase versus $BlockAvg$.

3.5.2 Overall Performance

Table 3.2 offers a comprehensive overview of our experimental outcomes, encapsulating average model accuracy across three drift categories under various training data volumes, alongside average system costs for all tested configurations. Our analysis reveals that fine-tuning specific components, namely Block 1, Block 3, and the fully connected layer, which align with input-, feature-, and output-level drifts respectively, delivers performance that either approximates or exceeds that of full model fine-tuning. Specifically, TBFT enhances model accuracy by an average of 15.30% compared to the $BlockAvg$ baseline, pointing to a meaningful correlation between drift types and the model blocks most impacted, where tuning the pertinent block substantially boosts efficacy. Furthermore, in terms of energy efficiency, TBFT demonstrates a notable reduction in consumption, achieving an average energy cost saving of 41.53% relative to full model fine-tuning. These findings are concisely presented in Table 3.2, with additional details provided in the subsequent sections.

Table 3.3: 128 and 1 batch of training accuracy results (on the Living17 dataset with 10% training data).

Batch Size	Block 1	Block 2	Block 3	FC	Full
128	47.30±1.05	55.57±0.46	65.83±0.35	59.01±0.41	64.69±0.29
1	45.17±0.98	54.12±0.62	64.23±0.31	57.92±0.52	62.97±0.32

3.5.3 Single-Batch & Multi-Batch Training

For resource-constrained training, batch size is generally set to 1 to accommodate hardware limits. However, this configuration, which does not leverage data parallelism, tends to substantially diminish experimental efficiency. We compare single-batch (size=1, Raspberry Pi) versus multi-batch (size=128, GPU) training to evaluate performance gaps and substitution feasibility. To ensure a fair assessment, all training parameters, aside from batch size, are held constant across both setups. Given single-batch inefficiency, we evaluate on 10% of Living17 [143], the results are shown in Table 3.3).

Table 3.3 reveals that, under equivalent conditions, single-batch and multi-batch training exhibit closely aligned accuracy levels. This finding justifies the adoption of GPU-based multi-batch training outcomes for performance analysis in subsequent experiments, enhancing overall efficiency. Accordingly, unless otherwise noted, we utilize GPU-derived training results as the standard measure of model performance throughout this work.

3.5.4 Model Performance

We evaluate personalization accuracy across drift types using three datasets: Cifar10-C [165] (input-level), Living17 [143] (feature-level), and Cifar-Flip [166] (output-level) with ResNet-26 [162]. All experiments use early stopping and 10 repetitions. Testing includes 10%, 20%, and 30% training sizes for small dataset accuracy.

Input-Level: Cifar10-C. We initialize and train ResNet-26 [162] from scratch on Cifar10 (10-class, 32×32 images). Following this, we implement TBFT on the Cifar10-C dataset, activating only one block at a time—such as Block 1, Block 2, Block 3, or the FC layer—while keeping all other blocks frozen, to evaluate their individual contributions to handling input-level drift. The findings are illustrated in Fig. 3.3 and Table. 3.4. Our results demonstrate that, for input-

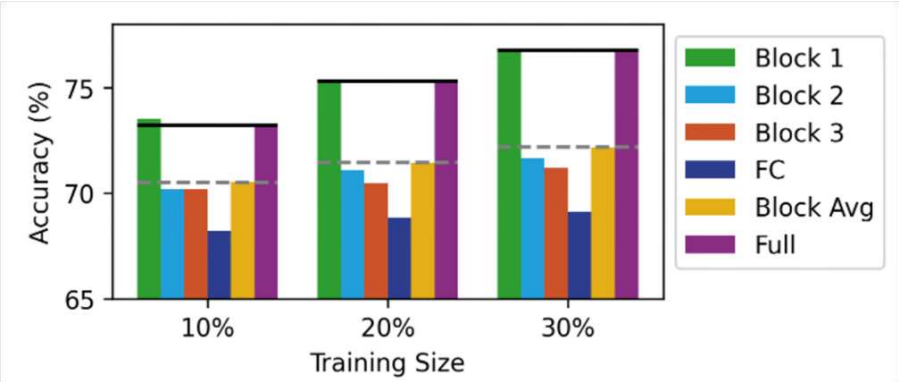


Figure 3.3: The accuracy results of TBFT on Cifar10-C datasets with different training sizes. Solid/dashed lines show *BlockAvg*/full-tuning baselines. Specific values are shown in Table 3.4.

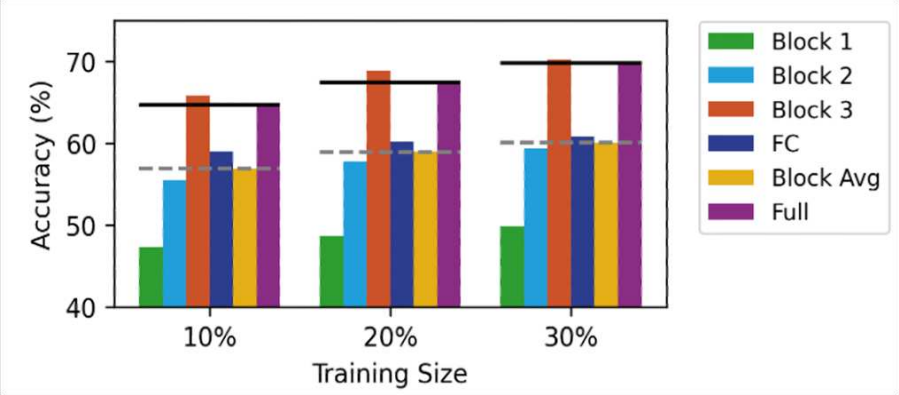


Figure 3.4: The accuracy results of TBFT on Living17 datasets with different training sizes. Specific values are shown in Table 3.4.

level drift, Block 1 fine-tuning achieves optimal performance, matching or exceeding full-model tuning. Specifically, compared to *BlockAvg*, Block 1 tuning at 10%, 20%, and 30% training sizes improves accuracy by 3.00%, 3.88%, and 4.56%, respectively. Moreover, we observe a clear positional effect: blocks closer to the model's front, particularly Block 1, exhibit superior performance, while fine-tuning the FC layer results in the poorest outcomes. Intermediate blocks (2-3) yield moderate fine-tuning results. These experimental observations are consistent with our theoretical analysis in Section 3.3, suggesting that, in the context of input-level drift, the front block of the model is more responsible to input-level data drift.

Feature-Level: Living17. We train ResNet-26 from scratch on Living17 (17-class, 128×128 images). Our data preprocessing and augmentation procedures closely follow the methods specified in BREEDS [143] for the Living17 dataset, ensuring experimental consistency. The

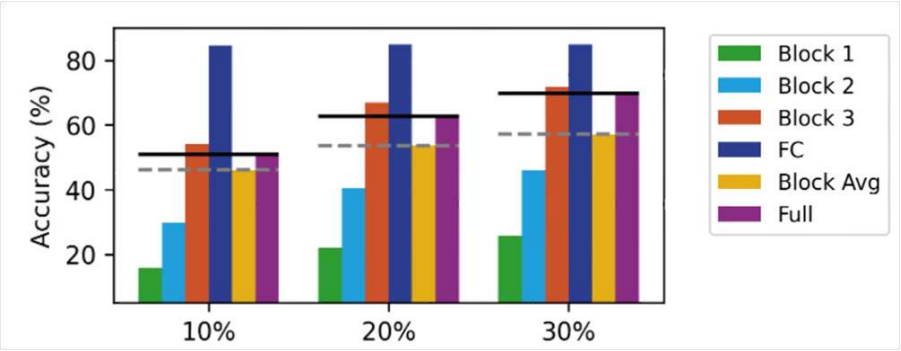


Figure 3.5: The accuracy results of TBFT on Cifar-Flip datasets with different training sizes. Specific values are shown in Table 3.4.

Table 3.4: The accuracy results of TBFT with various training size. Highlighted values indicate best block-tuning performance.

Drift Type & Dataset	Training Size	Block 1	Block 2	Block 3	FC	Block Avg	Full
Input-L Cifar10-C	10%	73.53±0.22	70.18±0.02	70.17±0.06	68.24±0.08	70.53±0.10	73.21±0.37
	20%	75.32±0.10	71.08±0.07	70.48±0.08	68.86±0.09	71.44±0.09	75.29±0.14
	30%	76.75±0.17	71.68±0.34	71.20±0.05	69.11±0.06	72.19±0.16	76.78±0.31
Feature-L Living17	10%	47.30±1.05	55.57±0.46	65.83±0.35	59.06±0.41	56.94±0.57	64.69±0.29
	20%	48.72±0.27	57.85±0.35	68.88±0.42	60.27±0.52	58.93±0.39	67.47±0.48
	30%	49.88±0.82	59.40±0.46	70.22±0.65	60.86±0.78	60.09±0.68	69.76±0.58
Output-L Cifar-Flip	10%	15.94±0.98	29.78±0.34	54.20±0.28	84.43±0.10	46.08±0.24	50.86±0.35
	20%	21.95±1.05	40.32±0.59	66.84±0.38	84.87±0.17	53.65±0.41	62.65±0.27
	30%	25.87±0.68	45.98±0.40	71.78±0.43	84.91±0.20	57.14±0.16	69.81±0.45

corresponding experimental results are presented in Fig. 3.4 and Table. 3.4. For feature-level drift, Block 3 tuning outperforms full-model tuning. Compared to *BlockAvg*, it improves accuracy by 8.89% (10%), 9.95% (20%), and 10.13% (30% training sizes). Consistent with our findings on the Cifar10-C dataset, the positional effect of model blocks significantly influences performance: blocks closer to the middle of the model, such as Block 3, demonstrate superior fine-tuning outcomes. For example, at a training size of 30%, Block 3 as a reference point (best performance) has an accuracy of 70.22%, while fine-tuning its neighboring blocks - Block 2 and the FC layer - has an accuracy of 59.40% and 60.86% respectively. However, fine-tuning Block 1, which is furthest from Block 3, has an accuracy of only 49.88%.

Output-Level: Cifar-Flip. The model configuration, training and fine-tuning process is the same as that used in the Cifar10-C experiments, with the only difference being that the Cifar10-C dataset was replaced with the Cifar-Flip dataset to further explore the effects of output-level drift. The corresponding experimental results are presented in Fig. 3.5 and Table.

Table 3.5: Time and energy costs of TBFT *vs.* full model fine-tuning. The *Energy-SavingRate* compares current energy cost to full-tuning baseline.

	Block 1	Block 2	Block 3	FC	Block Avg	All
Resolution 32, Output 10						
Time (s)	0.45	0.33	0.29	0.17	0.31	0.53
Energy Cost (J)	0.090	0.066	0.058	0.034	0.062	0.106
Energy-Saving Rate (%)	15.09	37.74	45.28	67.92	41.51	-
Resolution 128, Output 17						
Time (s)	6.82	5.40	4.28	2.46	4.74	8.12
Energy Cost (J)	1.705	1.350	1.070	0.615	1.185	2.030
Energy-Saving Rate (%)	16.01	33.50	47.29	69.70	41.63	-

3.4. Consistent with prior findings, we observe that for output-level drift, fine-tuning rear blocks (particularly the FC layer) achieves optimal accuracy. Specifically, compared to the *BlockAvg* baseline, FC layer tuning yields accuracy improvements of 38.35%, 31.22%, and 27.77% at 10%, 20%, and 30% training sizes respectively, surpassing the corresponding gains of 33.57%, 22.22%, and 15.10% from full-model fine-tuning. Additionally, these results reveal the same positional effect: blocks closer to the model's rear, such as the FC layer, exhibit superior accuracy when fine-tuned. For example, at a training size of 30%, fine-tuning blocks from the front to the rear—Block 1 through the FC layer—results in accuracies of 25.87%, 45.98%, 71.78%, and 84.91%, respectively, highlighting the pivotal role of rear blocks in output-level drift. Notably, fine-tuning only the FC layer with just 10% of the training data achieves performance nearly comparable to that with 30% of the data, suggesting that, for output-level data drift, adjusting the FC layer with a minimal local dataset size can suffice to obtain satisfactory model accuracy.

Summary. Our analysis demonstrates that the optimal fine-tuning block varies with drift type: front blocks (*e.g.*, Block 1) prove most effective for input-level drift, middle blocks (*e.g.*, Block 3) for feature-level drift, and rear blocks (*e.g.*, FC layer) for output-level drift. In scenarios where only one aspect of the distribution shifts while others remain stable, fine-tuning the corresponding block alone is sufficient to achieve satisfactory performance. For input-level drift (*e.g.*, image interference), the pixel-level features of the images change, but the fundamental data structure remains consistent between the source and target datasets; thus, adjusting only the front block is adequate to address this image perturbation. For output-level drift (*e.g.*, label shift), the input-level features of the images remain unchanged, with only the output labels drifting, necessitating fine-tuning of the rear block alone. For feature-level drift, precise

fine-tuning to the model's intermediate blocks are required to achieve domain-invariant feature learning, and the experimental results demonstrate that fine-tuning the model's middle block(s) optimally adapts to feature-level drift.

3.5.5 System Cost

We quantify the system cost on a Raspberry Pi using the Power Monitoring HAT. The device operates with an input voltage of 5V and exhibits a standby memory usage of 0.75GB. During both TBFT and full model training, we measure the runtime memory consumption under varying experimental conditions. Specifically, with an image resolution of 32 pixels and an output size of 10 classes, the operational memory usage remains approximately 0.11GB. Likewise, at a resolution of 128 pixels and an output size of 17 classes, the runtime memory usage is about 0.28GB. These memory usage figures indicate that, across all tested resolutions and output sizes, the system resource demands stay well within the hardware constraints of the Raspberry Pi, fully meeting the requirements of the experimental setup.

We first measure the power consumption of a Raspberry Pi, determining that the standby power usage is approximately 4W. During operation, the power consumption varies with different resolutions and output sizes: it is about 4.20W for an image resolution of 32×32 and an output size of 10 classes, and approximately 4.25W for a resolution of 128×128 and an output size of 17 classes. The corresponding energy costs and runtime, are detailed in Table 3.5. We can find that, compared to full fine-tuning, TBFT demonstrates energy savings ranging from 15.09% (minimum) to 69.70% (maximum), with 41.57% average savings. Moreover, TBFT consistently demonstrates stable energy savings across different training configurations.

3.6 Discussions and Limitations

Our results suggest that fine-tuning the blocks corresponding to specific types of data drift—namely, through TBFT—can deliver superior model performance while substantially reducing energy consumption. However, TBFT also has some limitations:

• **Requires Prior Knowledge of Drift Types.** Currently, TBFT relies on prior knowl-

edge of the data drift type as a prerequisite for its implementation. However, real-world drift type identification remains challenging. Existing solutions like Auto-RGN and Auto-SNR [139] leverage training gradient dynamics for parameter/block selection, but their integration into the full training pipeline increases computational/memory overhead on resource-constrained devices, potentially compromising performance. Perhaps a lightweight discriminator head is a possible solution, which can identify the type of data drift based on the distribution of a small amount of local data.

• **Module-Based Model Structure.** Since TBFT requires selecting the front, middle, or rear modules of a fine-tuned model based on the drift type, we chose ResNet-26 for experimental convenience. This is because ResNet-26 has a module-based structure, making it easy to divide into front, middle, and rear modules. Our front/middle/rear segmentation is based on functional semantics rather than fixed depth, which ensures generality across architectures. However, the more specific and detailed division of the middle block still requires further exploration, especially for deeper models. Future work may explore dynamic or data-driven segmentation, such as identifying functional boundaries via layer-wise representational similarity or gradient-based sensitivity analysis [139].

• **Applicability of TBFT.** TBFT was primarily developed and validated on block-based convolutional image models (*e.g.*, ResNet, SqueezeNet [167]) and image inputs. These architectures exhibit clear modularity and spatial hierarchy, aligning naturally with our proposed front/middle/rear block segmentation. Extending TBFT to other architectures—such as transformer-based models or those handling non-visual modalities (*e.g.*, language or multimodal inputs)—remains to be validated. In such cases, the hierarchical separation of features may not map directly to simple block divisions, and fine-tuning specific blocks may suppress critical information pathways. Future work could explore developing more general adaptation mechanisms across different architectures or introducing module division methods based on feature distribution analysis to enhance the strategy's applicability breadth.

• **Mixed Data Drift.** In practical applications, data drift often manifests as a complex mixture of multiple types rather than a single form, such as simultaneous input-level and feature-level

drifts: local data may be affected by input-level noise while also including unseen samples from the same category as the source domain. To address this challenge, a lightweight discriminator head can analyze the distribution of a small amount of local data to identify primary drift types and their importance, enabling a stepwise fine-tuning strategy that sequentially adjusts different model blocks based on drift priority, thereby reducing computational costs and enhancing model robustness to mixed drifts.

3.7 Conclusions

This chapter presents TBFT, a low-energy framework designed to address data drift and personalization challenges on resource-constrained devices. It categorizes drift into input-, feature-, and output-level types, and introduces targeted fine-tuning strategies by adapting the corresponding front, middle, or rear blocks of modular convolutional neural networks. We tested TBFT on ResNet, and compared to fine-tuning non-corresponding blocks, TBFT improves accuracy by 15.30% on average; compared to full-model fine-tuning, TBFT reduces energy use by 41.57% on average. These results highlight the effectiveness of TBFT in visual domains with modular, hierarchical architectures.

Chapter 4

Model-Related Adaptations for Ultra-Resource-Constrained MCUs

This chapter continues to explore the model personalization problem in edge computing. Unlike Chapter 3, in this chapter, we focus on nonlinear feature-level data drift, and shift the experimental platform to ultra-resource-constrained MCUs, with RAM typically <1MB. Similar to Chapter 3, we still used public image data. We did not use the CIFAR dataset due to its low resolution, making experimental result comparisons less distinct. Instead, we used ImageNet (224 resolution) and carefully selected three datasets with significant feature distribution differences from ImageNet. Existing MCU personalization methods either rely on stable cloud connectivity, simplistic model architectures, or demand complex and energy-intensive local training, thereby contradicting the inherent energy advantages of MCUs. To address these limitations, we propose MicroT, a low-energy on-device personalization framework for MCUs. MicroT constructs a general yet lightweight feature extractor through self-supervised knowledge distillation, enabling localized adaptation via lightweight fine-tuning of task-specific heads. MicroT has an MCU-designed early-exit mechanism that allows users to customize exit criteria to dynamically balance energy cost and model performance (*e.g.*, accuracy). Evaluated on two models, three datasets, and two MCUs, MicroT achieves accuracy improvements of 2.12–11.60% compared to traditional transfer learning (TTL) and two SOTA methods. It reduces on-device training energy by 2.28× while maintaining SRAM usage below 1MB. During inference, MicroT reduces energy cost by 14.17% compared to TTL.

4.1 Introduction

MCUs stand out due to their cost-effectiveness and energy efficiency, establishing them as a significant hardware platform for Tiny Machine Learning (TinyML) [156]. MCUs lend themselves to versatile integration across various practical contexts, such as health monitoring, smart industry, and smart home [168, 169, 170]. Due to the limited computational capacity of the MCU, its ML model is typically trained in the cloud. However, the training data in the cloud may experience data drift compared to local data, leading to a decline in model performance (*e.g.*, accuracy).

Data drift [171], defined as a substantial divergence in the statistical distributions of source and target data, exerts a deleterious influence on a model's predictive prowess, precipitating a decline in performance. To solve this challenge, researchers have developed several personalization techniques, including multi-task learning [172, 173], zero-shot learning [42], and cloud-edge collaborative paradigms like federated learning [174, 175, 176]. However, these methodologies still have limitations: multi-task learning, which enhances performance by concurrently optimizing multiple related tasks, tends to be confined to predefined local task scenarios, potentially faltering when confronted with unanticipated tasks or data distributions; zero-shot learning hinges on transferring pre-trained knowledge to unseen data categories, but its generalization capacity may wane amid highly heterogeneous local data distributions, further compounded by the prerequisite of possessing prior knowledge about local data characteristics—a condition often unmet in practice; cloud-edge approaches like federated learning, which aggregate multi-party data resources via distributed training to refine models, predicate their efficacy on frequent and unwavering communication between cloud infrastructure and local devices, imposing rigorous demands on network bandwidth, latency, and connection reliability, while also necessitating robust local computational capabilities to execute training, resulting in considerable resource expenditure that may prove infeasible in environments with unstable networks or limited device performance. Consequently, directly personalizing and tuning models at the local devices is a significant strategy for upholding their effectiveness and stability.

On-device personalization, executed mainly by the local device without necessitating cloud-

based communication, markedly enhances operational efficiency and curtails latency stemming from data transmission [177]. Techniques such as TinyTL [44] and SparseUpdate [43] have successfully implemented personalization on MCUs, and have been deployed and rigorously evaluated in conjunction with sophisticated models, affirming their feasibility within resource-constrained environments. However, our experimental findings reveal that, these approaches incur considerable energy costs, as elaborated in Section 4.5. Specifically, $TinyTL$ integrates lightweight residual modules into the backbone network and only trains these added modules, it still demands significant energy cost. SparseUpdate adopts an alternative tactic by training only pre-selected layers or channels. However, the layer/channel selection process requires access to a complete local dataset, and the selection process entails thousands of computationally intensive search operations [43] or pruning steps [178], leading to training durations that can extend up to ten days [47]. Compounding these issues, the above methods predominantly concentrate on refining local training processes while overlooking the energy of local inference, a critical phase that significantly influences the MCU's energy cost during long-time operation.

Challenges.

(1) **Ultra-Resource-Constrained Nature.** MCUs have severely limited hardware specifications, typically featuring less than 1MB of RAM. This memory capacity is approximately three orders of magnitude smaller than that of mobile devices, such as smartphones, and five to six orders of magnitude below the capabilities of cloud-based GPUs [179]. The extremely limited resources also make the approach proposed in Chapter 3, 'fine-tuning the middle blocks of the model for feature-level data drift' difficult to implement on MCUs. This is because fine-tuning middle blocks requires more RAM and can lead to discontinuous quantization during deployment, resulting in device support errors.

(2) **Hardware-Aware Software Design for MCU Personalization.** Personalization on MCUs requires integration of algorithms and model design with the constraints of the underlying hardware platform. Unlike conventional computing systems, MCUs typically lack an operating system, necessitating direct implementation of model deployment and training in low-level languages like C/C++. To effectively manage memory footprint and energy consump-

tion in this constrained environment, models must be carefully designed to leverage existing MCU AI toolkits, balancing model complexity, computational efficiency, and hardware-specific capabilities to ensure the feasibility and effectiveness of personalization on these platforms.

(3) **Energy Sensitivity.** MCUs are engineered with energy efficiency as a paramount design principle, a trait that renders them indispensable in cost-sensitive and power-conscious applications, such as wearable devices (*e.g.*, health-monitoring wristbands) and batteryless systems reliant on energy harvesting (*e.g.*, sensor nodes) [126, 125]. These use cases impose stringent requirements on device longevity and power conservation. Consequently, implementing model personalization on MCUs must preserve their hallmark low-energy profile.

These challenges have prompted researchers to pivot toward experimenting with more robust computational platforms, such as mobile devices with 8GB of RAM—like smartphones—as exemplified in studies like [180, 181, 182]. These platforms, in stark contrast to MCUs with a mere 1MB of RAM, boast ample computational resources and memory capacity, enabling them to more readily accommodate the demands of complex model training and inference tasks. Nonetheless, this shift underscores the persistent technical conundrum of achieving personalization on ultra-low-resource devices, a frontier that remains ripe for innovative solutions.

Contributions. In this chapter, we introduce MicroT, a low-energy model personalization framework for ultra-resource-constrained MCUs [1]. The goal of MicroT is to sustain personalization performance (*e.g.*, accuracy) while reducing its energy consumption. We consider that model personalization includes both training and inference and therefore explored both of these stages. Our contributions are summarized as follows:

(1) MicroT follows the traditional transfer learning (TTL) paradigm, splitting the model into a feature extractor and a classifier, and training only the lightweight classifier on local devices. This configuration maximizes the utilization of the MCU's scarce resources, and facilitates model quantization (quantizing only the feature extractor).

(2) MicroT leverages SSL to train a robust general feature extractor in the cloud by public

[1]The code, models, and datasets are available at: https://github.com/yushan-huang/MicroT

dataset, which can address the limited learning space caused by fine-tuning only the classifier, and avoid using local dataset.

(3) MicroT adopts a two-stage KD approach. Experiments reveal that applying SSL directly to small-scale models yields limited performance improvements [183] (detailed analysis available in Section 4.6). Therefore, we first apply SSL to a computationally powerful large model and generate an auxiliary dataset from the high-quality embedding features extracted after training; then, a compact feature extractor is trained by simultaneously learning from both this generated dataset and the original dataset, effectively balancing performance and model size reduction.

(4) MicroT employs an MCU-optimized early-exit strategy that leverages a model segmentation fused score—a metric balancing performance and efficiency—to divide the feature extractor into partial or full model configurations (see Section 4.3.4 for details).

(5) We evaluated MicroT across two models, three datasets, and two MCU development boards. The results show that MicroT achieves superior accuracy, surpassing TTL by 7.01 - 11.60% and outperforming two SOTA methods (TinyTL and SparseUpdate) by 2.29 - 5.73% and 2.12 - 3.68% across two models and three datasets, respectively. During the MCU training phase, MicroT achieves energy savings of 2.03 - 2.28X compared to SOTA methods across two models and two datasets. Additionally, in the MCU inference phase, MicroT reduces energy consumption by 14.17% relative to TTL across two models and two datasets.

(6) MicroT provides a customizable stage-decision ratio, allowing users to dynamically adjust the trade-off between model accuracy and energy cost.

This chapter is related to the following papers:

Yushan Huang, Ranya Aloufi, Xavier Cadet, Yuchen Zhao, Payam Barnaghi, Hamed Haddadi. 'Low-Energy and Adaptive Models for MCUs'. Accepted to The Ninth ACM/IEEE Symposium on Edge Computing (SEC 2024).

Yushan Huang, Hamed Haddadi. 'Poster: Towards Machine Learning and Inference for Resource-constrained MCUs'. Accepted to The 21st ACM International Conference on Mobile Systems, Applications, and Services (MobiSys 2023).

4.2 Background

MCU-AI is Pervasive. With the rapid advancement of artificial intelligence technologies, a significant trend in AI deployment is the progressive shift of models from conventional cloud-based systems to edge devices. This shift offers key advantages, such as significantly reducing the latency associated with transmitting data to the cloud while avoiding the upload of sensitive data to remote servers, thereby enhancing user privacy and data security [184, 185]. MCU-based edge devices play a crucial role in this trend due to their affordability, extremely low power consumption, and ability to operate efficiently in resource-constrained environments, making them highly suitable for large-scale deployments in IoT and embedded systems. For example, MCUs are capable of handling signals from diverse sensors, facilitating a broad practical uses. Specifically, they can analyze audio data captured by microphones to achieve acoustic detection, identifying specific sound patterns or anomalous noises in the environment [186]; they can also process data from accelerometers and gyroscopes for predictive maintenance, detecting potential mechanical failures in advance [187]; additionally, when paired with low-power cameras, MCUs can support image detection tasks, such as simple object recognition in battery-powered devices [125]; and they can even analyze sensor data for human activity recognition, aiding in health monitoring or optimizing smart home functionalities [188]. These examples highlight the versatility and flexibility of MCUs in edge AI applications. Compared to high-performance GPUs reliant on cloud infrastructure, the low cost and energy efficiency of MCUs enable their widespread adoption in resource-limited scenarios, driving the penetration of AI technologies into everyday life and industrial production on a broader scale.

MCUs are Resource-Constrained. MCUs have limited memory (SRAM) and storage (Flash) capacities, which are closely tied to their design goals of operating in low-power, small-sized, and cost-effective scenarios. For instance, the STM32L4R5ZI, a commonly used MCU model, is equipped with an ARM Cortex-M4 core, but its resource allocation is quite restricted, offering only 2 MB of Flash storage for firmware code and data, and 640 KB of RAM for runtime data processing. This resource limitation implies that developers must optimize code and manage data when designing MCU applications to ensure the system can operate efficiently within such a confined space. In contrast, other computing platforms possess resources far

exceeding those of MCUs. For example, cloud GPUs like the NVIDIA V100 come with 16 GB of memory and storage capacities reaching terabytes or even petabytes, tailored for handling large-scale parallel computing tasks such as artificial intelligence model training and big data analytics. Similarly, modern mobile devices like the iPhone 11, while not as extreme as cloud GPUs, are equipped with 4 GB of memory and 64 GB of storage, sufficient to support complex operating systems, multitasking, and user data storage. The memory and storage capacities of these devices surpass those of MCUs by approximately 3 to 6 orders of magnitude, reflecting their entirely different application domains: MCUs are geared toward embedded systems and real-time control, whereas GPUs and mobile devices cater to high-performance computing and user interaction experiences. Thus, the resource constraints of MCUs are both an advantage—enabling low power consumption and efficiency—and a limitation, requiring developers to carefully balance design choices based on specific needs.

Cloud-Edge Collaboration Personalization. In the rapid development of artificial intelligence, cloud-edge collaboration has emerged as a critical direction for achieving personalized applications. Federated learning demonstrates a cooperative strategy between cloud and local resources, where a global model is trained in the cloud and subsequently fully or partially deployed to local devices for refinement to suit particular tasks [39]. This method allows devices to participate in model training without uploading raw data, thereby preserving data privacy to some extent [184]. However, federated learning relies heavily on frequent and stable communication between the cloud and local resources. In scenarios with poor network conditions, such as remote mountainous areas or regions with underdeveloped infrastructure, this communication may become unreliable or disrupted, resulting in reduced training efficiency or even failure. Although recent research efforts have sought to mitigate this issue by optimizing communication protocols or reducing communication frequency, these challenges remain significant in real-world deployments. Moreover, frequent data interactions not only increase energy consumption—posing a burden especially for battery-powered edge devices—but also introduce privacy risks due to potential data leakage during communication [40, 189].

Meanwhile, multi-task learning offers another approach to enhance the efficiency of cloud-edge collaboration. It leverages domain information related to the source task, embedded within

the training signals of the target task, to simultaneously optimize the performance of multiple tasks [41, 190]. The essence of this approach rests on the commonality among tasks, which facilitates the sharing of features and the transfer of knowledge. For example, in a smart home scenario, a single model might simultaneously recognize voice commands and detect ambient temperature, with multi-task learning extracting shared features to improve efficiency. Nevertheless, the effectiveness of multi-task learning depends on the relationship between the tasks involved. If prior knowledge about the target task is lacking, or if significant disparities are present between the distributions of the source and target data, multi-task learning may struggle to deliver effective improvements and could even result in negative transfer, degrading performance.

Zero-shot learning further extends the personalization capabilities of cloud-edge collaboration by enabling models to perform tasks without training data for the target task. This approach depends on rich semantic information to facilitate knowledge transfer across categories, which may include manually defined attribute vectors (*e.g.*, labels describing object characteristics), automatically extracted word vectors (based on natural language processing techniques), context-based embeddings, or combinations thereof [42]. For instance, in image classification, zero-shot learning can infer features of new categories based on semantic descriptions of known categories [191]. However, its application on edge devices faces challenges, as generating and processing semantic information demands significant computational resources, which are often limited on edge devices. Furthermore, if the semantic information is inaccurate or incomplete, the effectiveness of knowledge transfer diminishes [192].

On-device Personalization. Currently, several on-device training methods have been developed to minimize reliance on the cloud and optimize local training processes. For example, SparseUpdate is a technique that reduces computational overhead by selectively updating specific layers or channels of a model [43]. However, this method requires access to the entire local dataset and demands nearly ten days to search for and determine which parts of the model to update, followed by the actual training process. Such a time-intensive approach is clearly impractical for applications requiring rapid responses. On the other hand, TinyTL seeks to achieve personalization by training additional lightweight modules locally [44]. While this

approach theoretically reduces modifications to the original model, it necessitates the simultaneous training of numerous extra modules, which increases computational complexity and renders the training process inefficient. Although both methods contribute meaningfully to improving model performance, they fall short in addressing energy consumption and practical feasibility. For instance, the prolonged search process of SparseUpdate and the multi-module training of TinyTL can deplete the limited power reserves of edge devices, while also failing to meet real-time requirements effectively.

Hardware-Aware Software Design. Cloud and mobile devices typically operate with full-fledged operating systems that support high-level programming languages like Python, making the development and deployment of machine learning models relatively flexible and straightforward. In contrast, MCUs are bare-metal devices without operating systems, imposing strict limitations on both hardware resources and software environments. As a result, they cannot directly execute complex code written in high-level languages. Instead, the inference and training routines of on-device models must be implemented in low-level languages such as $C/C++$. This language constraint requires developers to build the operational logic of models from the ground up, significantly increasing development complexity and necessitating a tight alignment between the model design and the MCU's hardware characteristics. This underscores the importance of hardware-aware software design: model development cannot focus solely on the algorithm itself but must be integrated with existing MCU AI libraries (*e.g.*, lightweight frameworks optimized for embedded systems) while meticulously managing memory, storage, and energy consumption to ensure efficient operation within the severely constrained environment.

However, current research still faces significant challenges in achieving effective hardware-aware software design. For instance, some works opt to use low-resolution datasets, such as CIFAR10 or CIFAR100, to test model performance. These datasets are smaller in scale and have lower computational demands, making them marginally compatible with the limited resources of MCUs [45]. Other studies lean toward deploying simple models, such as small convolutional neural networks or linear models, to reduce demands on computational power and memory [46]. Additionally, some research sidesteps the extreme constraints of MCUs altogether by validating their methods on more resource-rich mobile devices, such as smartphones or development

boards, rather than true MCU platforms [47, 48]. While these approaches can demonstrate the effectiveness of algorithms to some extent, they fail to address the core issue of integration between hardware and software in the MCU context. For example, low-resolution datasets do not reflect the demands of complex real-world tasks, and simple architectures may sacrifice expressive power, leading to suboptimal performance. Moreover, results obtained from mobile device testing are difficult to generalize to MCUs due to significant differences in hardware capabilities and runtime environments.

Model Inference is Crucial. When implementing artificial intelligence models on edge devices, notably MCUs, the efficiency of the inference stage is as vital as the training process, as it directly impacts the model's response time and energy usage in practical scenarios. At present, methods facilitating model inference on MCUs primarily emphasize model compression strategies, including quantization and pruning [193, 194]. Quantization decreases storage demands and computational complexity by transforming model parameters from high-precision formats (*e.g.*, 32-bit floating-point) to lower-precision representations (*e.g.*, 8-bit integers) [194, 193], while pruning simplifies network structures by removing redundant weights or neurons [195, 196]. Executed on cloud devices and then deployed to MCUs, these techniques significantly lower the resource demands during inference. However, a key limitation unites these methods: they are static. After the model is compressed and implemented on the local device, its architecture and parameters are set and no longer adjustable, unable to dynamically adjust inference energy consumption or performance based on real-time task requirements.

In the realm of large language models (LLMs), early-exit mechanisms have emerged as a popular technique to enhance inference efficiency by reducing unnecessary computations [197, 198, 199]. Traditional early-exit mechanisms incorporate multiple potential exit layers within the model, allowing the inference process to terminate prematurely when predefined exit criteria (*e.g.*, a confidence threshold) are met, rather than running the entire network. This approach proves effective on resource-rich devices, as it balances accuracy with reduced computational load. However, applying these methods directly to resource-constrained MCUs presents significant challenges. First, supporting multiple exit layers requires reserving additional computational and storage space for each layer, which strains the limited memory and processing capabilities

of MCUs. Second, complex or frequent exit criteria—such as real-time evaluation of output confidence at each layer—demand extra logical judgments and computations, further increasing energy consumption and latency. For battery-powered MCU devices that need to operate over extended periods, these additional overheads may offset the efficiency gains of early exiting, potentially degrading overall performance.

4.3 Methodology

In this section, we first formulate our optimization problem, then provide an overview of the MicroT. Following this, we detail the methods utilized in MicroT. The system overview of MicroT is illustrated in Fig. 4.1.

4.3.1 Optimization Problem Formulation

We focus on two conflicting objectives: maximizing model accuracy (Acc) and minimizing local training energy cost (TE). For a given algorithm $\mathcal{M}$, the problem can be formulated as a bi-objective Pareto optimization:

$$\begin{aligned} &\max\ Acc(\mathcal{M}) \\ &\min\ TE(\mathcal{M}). \end{aligned} \tag{4.1}$$

4.3.2 Overview

Assumptions. (1) The availability of locally acquired data through standard sensor modalities such as cameras or other embedded sensors commonly integrated in MCU platforms; (2) Labeled data are available locally, either via prior annotation or supervised data collection methods; (3) Cloud side does not have access to local datasets but has abundant publicly available datasets; (4) There exists feature distribution shift between local data and cloud data; (5) Local devices are resource-constrained and energy-sensitive, such as MCUs.

Distinguish from traditional cloud-edge framework, MicroT only requires a single cloud interaction to enable prolonged local operation. This design reduces communication overhead and enhances system autonomy. MicroT is built upon five essential components, represented in

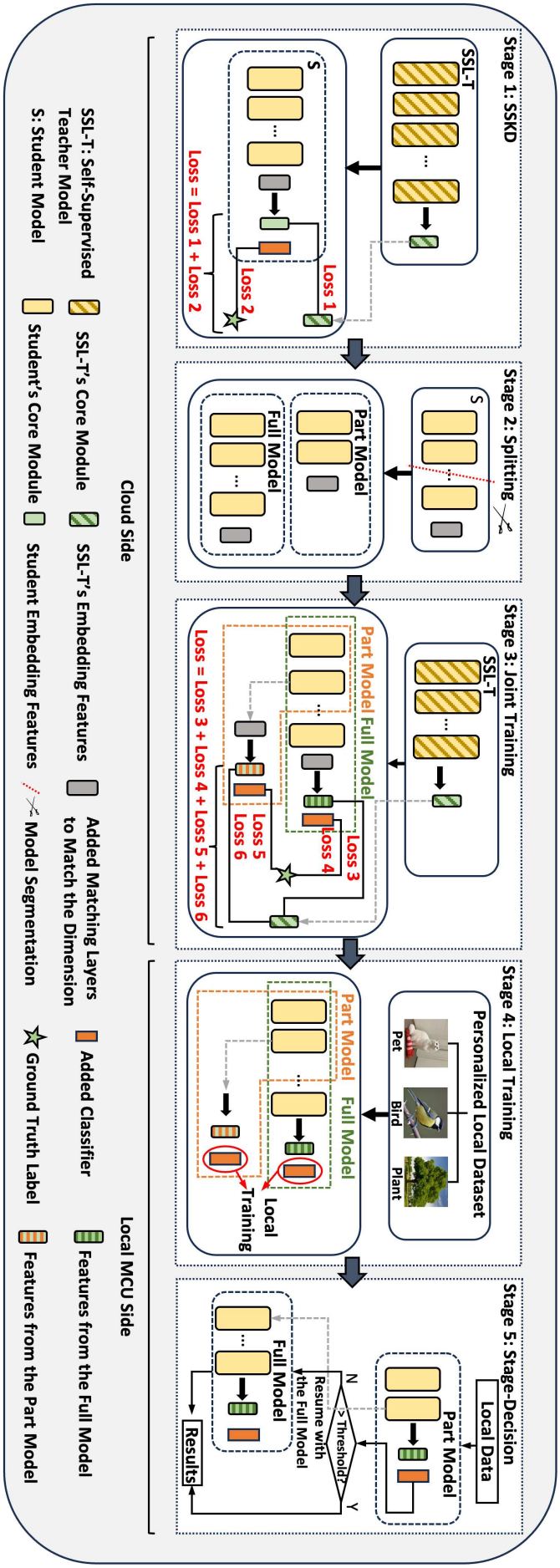


Figure 4.1: System overview of MicroT.

Fig. 4.1. (1) At the cloud level, MicroT employs self-supervised knowledge distillation (SSKD) to enhance the feature extractor's ability to generalize; (2) The model segmentation process divides the original model into a part model and a full model based on an optimal splitting point, preparing the groundwork for an early-exit mechanism on the MCU while ensuring effective resource utilization; (3) Joint training refines the coordination between the part model and the full model, optimizing their collaborative performance and overall accuracy; (4) On the local MCU side, MicroT enables local training capabilities, empowering devices to independently update and enhance models within the constraints of limited computational resources; (5) The stage-decision mechanism can tailor the model inference to the evolving demands of practical application scenarios (focus more on model accuracy or energy cost).

The following paragraphs provide a thorough introduction of MicroT's five core components.

The Design Philosophy of MicroT is Based on Transfer Learning. TTL offers distinct benefits for MCU personalization. It leverages a pre-trained model's generalized knowledge, which is subsequently fine-tuned on local devices. This adaptability renders transfer learning particularly well-suited to resource-constrained MCU environments, where it facilitates efficient model optimization despite limited computational capacity. Techniques such as TinyTL and SparseUpdate, both grounded in TTL, exemplify this principle. Each method begins with a pre-trained source model or backbone network and employs local training or fine-tuning to tailor the model to specific tasks.

Feature Extractor + Classifier Paradigm. MicroT partitions the model into two distinct components—a feature extractor and a classifier—with the division informed by considerations of energy efficiency and computational constraints. Existing TTL-based approaches tailored for MCUs, such as TinyTL and SparseUpdate, achieve enhanced predictive capabilities, yet these gains come at the expense of increased energy consumption and computational complexity, as elaborated in Section 4.5.5. This trade-off diminishes the inherent benefits of MCUs, which are prized for their efficiency in resource-limited settings. In contrast, the 'feature extractor + classifier' paradigm, while yielding limited learning space due to fine-tuning only the classifier, excels in minimizing both energy usage and computational overhead. From a pragmatic per-

spective, we contend that the most effective path to optimization lies not in abandoning this paradigm but in refining its to optimize performance while preserving its strengths in energy efficiency and resource utilization.

MicroT Supports a Powerful but Tiny Feature Extractor. The suboptimal model accuracy of the 'feature extractor + classifier' paradigm largely stems from the constrained capacity of the classifier, which struggles to compensate and learning variations within the feature space when trained on a MCUs. Given the resource limitations imposed on the classifier training, we propose that a more feasible enhancement lies in strengthening the feature extractor's capability to learn complex feature patterns. The objective of this feature extractor is to achieve a high degree of generalization and adaptability, thereby enabling the classifier to deliver satisfied performance with less training and computation. To this end, we employ SSL, which increase the extractor's generalization through methods such as data augmentation. Moreover, since cloud systems lack access to local data, training relies on a non-local, publicly available dataset featuring a broad array of categories and samples—such as ImageNet—ensuring that the feature extractor acquires sufficiently expansive knowledge applicable across varied domains. The constrained memory capacity of MCUs necessitates the deployment of lightweight models, such as MCUNet [179], tailored for resource-constrained devices. Our experimental evaluations revealed that applying SSL directly to these lightweight models yielded only marginal accuracy gains, with the restricted model capacity emerging as a critical bottleneck [183], as detailed in Section 4.6. Therefore, we integrated SSL with KD, termed SSKD, with the process outlined in Stage 1 of Fig. 4.1.

MicroT Offers Low Energy Model Training on Local Devices. Upon receiving the cloud-trained feature extractor, the MCU initializes a local classifier tailored to the specific requirements of the local task. This classifier is subsequently trained on-device using local dataset, as depicted in Stage 4 of Fig. 4.1. Owing to the general and powerful feature extractor, the classifier requires only a minimal task head (*e.g.*, a straightforward classifier) and simple fine-tuning to achieve personalization enhancements. In addition, due to the split design of 'Feature Extractor + Classifier', local data can be stored locally in the form of features, for example, an original 128×128 size image can be stored with only 512 size features. Finally,

since the part model and full model are embedded in each other, the task head training of the full model can also continue to use the intermediate values of the part model, thus saving memory, time, and energy.

MicroT Offers Low Energy Model Inference on Local Devices. The system cost of model inference will determine the long-term or even life-long energy cost of the MCU. Inspired by early-exit mechanisms, we find that full model inference is not universally necessary; leveraging only the initial segment of the model often suffices to produce outcomes comparable to those of complete inference, thereby save the MCU's resource and energy cost. Conventional early-exit strategies typically embed multiple prospective exit points across various model layers, yet this approach proves suboptimal for MCUs due to the elevated computational load and energy overhead stemming from recurrent exit evaluations and the maintenance of multiple classifier heads. In response, we design an alternative by splitting the model into two distinct parts—the part model and the full model—as illustrated in Stage 2 of Fig. 4.1. The part model, constituting a subset of the full model, substantially reduces memory requirements during deployment. To ensure an efficient partitioning strategy, we introduce a model split fused score that balances model performance against system consumption. Alignment between the part and full models is achieved through joint training, as depicted in Stage 3 of Fig. 4.1. On the local MCU, part and full models collaborate via a stage-decision process (Stage 5, Fig. 4.1), wherein data is first processed by the part model, with the full model selectively engaged based on a confidence threshold. To establish this threshold effectively, we propose a median-quartile-based technique that determines a localized confidence level using merely five local data points, with its efficacy substantiated through experimentation (see Section 4.5.4 for details).

MicroT is Flexible and Adaptable. Unlike systems tied to specific hardware, MicroT can be deployed across a range of commercial MCUs, such as the STM32 series and the Espressif ESP series. This broad compatibility enhances its practical utility in varied settings. Furthermore, MicroT accommodates variety of model architectures, provided that the models and their associated operators are supported by the target MCU platform and adhere to its resource constraints, such as memory and computational capacity. This inclusive design ensures that MicroT can effectively integrate with diverse modeling approaches tailored to specific application

needs. During the local stage-decision phase, MicroT offers users the ability to dynamically adjust the trade-off between model accuracy and energy expenditure. By tuning the stage-decision ratio, users can prioritize energy efficiency by modestly compromising accuracy or, conversely, allocate additional energy costs to increase model accuracy.

4.3.3 Self-Supervised Knowledge Distillation

As described in Section 4.3.2, to reduce the system costs and memory requirements for local model training on MCUs, we partition the model into two components: a feature extractor and a classifier. The design of the feature extractor was guided by two primary considerations: (i) the extracted features must exhibit sufficient generalizability to accommodate personalized local data, thereby enabling the appended local task head to be trained efficiently under constrained system resources; (ii) the lack of access to local data in the cloud for training purposes, which precludes the direct application of conventional cloud-based model optimization techniques. To address these challenges, we first employ SSL to enhance the generalizability of the feature extractor, allowing it to adapt to diverse local data distributions. Subsequently, we apply KD to transfer knowledge from a larger model into a smaller feature extractor.

Firstly, we apply SSL to the teacher model rather than directly to the student model. This decision is based on the fact that SSL typically provides more significant performance improvements for larger and more complex models, whereas the student model, due to its simpler architecture, may experience only limited gains from direct SSL application [183] (see Section 4.6 for details). To develop a teacher model with strong generalization capabilities, we follow the SSL paradigm outlined in [38], designing two complementary models: a guide model and an exploration model, which collaborate to enable efficient unsupervised feature learning. These models process distinct variants of the same image, produced using a variety of data augmentation methods, including random cropping, color perturbation, and flipping, thereby producing diverse input representations to support comprehensive feature learning. The guide model is designed to employ an exponential moving average (EMA) approach, smoothing its parameter updates during training to maintain output stability and ensure the reliability of the extracted features. In contrast, the exploration model updates its parameters directly via

back-propagation, learning features from the varied perspectives provided by augmentation to align its outputs with those of the guide model. The loss function is expressed as follows:

$$\mathcal{L}_{\text{SSL}} = -\sum \text{softmax}\left(\frac{o_g}{\tau_g}\right) \log \text{softmax}\left(\frac{o_e}{\tau_e}\right) \tag{4.2}$$

where g_i and e_i denote the output logits of the guide model and the exploration model, respectively, reflecting their predictive capabilities for the given input features. The parameters τ_g and τ_e are temperature scalars that modulate the sensitivity of the $softmax$ function by scaling the logits, thereby influencing the smoothness of the output probability distributions. This loss function is crafted to prompt the exploration model to learn and mimic the behavior of the guide model by handling various augmented perspectives of the same data, promoting the alignment of their feature representations. Specifically, the optimization objective is to minimize this cross-entropy loss, enabling the exploration model to effectively mimic the predictive distribution of the guide model. Meanwhile, the guide model employs an EMA strategy, which smooths its parameter updates during training to mitigate abrupt fluctuations caused by variations in data views, thereby allowing it to progressively adapt to the diversity of the input data and enhance its ability to capture robust data representations.

After obtaining a performance-optimized guide model trained through SSL, this model transitions into the teacher model for the KD phase, serving as the foundation for guiding subsequent model training. In the KD process, to ensure the student model effectively gains knowledge from the teacher model, we extract the embedding features of the teacher model to serve as a learning reference. Specifically, we pass a public dataset through the trained teacher model and extract the embedding outputs from its feature extractor (*i.e.*, the layers preceding the task head), thereby generating a dataset enriched with informative feature representations. This step leverages the generalization capabilities acquired by the teacher model during the SSL phase. To guarantee that the output dimensions of the student model's feature extractor match those of the teacher model, we add a fully connected layer, known as the matching layer, following the student model's feature extractor. This layer performs a linear transformation to adjust the feature dimensions of the source model to match those of the teacher model [200].

The distillation loss function $\mathcal{L}_{\text{distill}}$, crafted to enhance the student model's learning process, is defined as follows:

$$\mathcal{L}_{\text{distill}} = \sum [\alpha * \mathcal{L}_{\text{MSE}}(o_s, o_t) + \mathcal{L}_{\text{CE}}] \tag{4.3}$$

$\mathcal{L}_{\text{MSE}}$ calculates the mean squared error between the output logits of the student model o_s and the teacher model o_t, $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss between the ground truth label and student model's output, and α serves as a weighting factor to balance the influence of the two loss components. This loss function combines supervision at both the feature and prediction levels, ensuring that the source model thoroughly absorbs the knowledge contained within the teacher model.

We employ a non-local public dataset stored in the cloud for training the model, encompassing both SSL and KD phases. To ensure that the teacher model and the feature extractor achieve robust generalization capabilities, we deliberately select a dataset that includes a diverse range of categories and abundant learning samples. This choice is motivated by the fact that, in the cloud environment, there is no access to specific information about local data, resulting in a lack of prior knowledge regarding its distribution. Under these circumstances, employing a varied public dataset enables the teacher model to encounter a broader spectrum of data patterns during training, thereby enhancing its adaptability to unknown local data.

4.3.4 Model Segmentation and Joint Training

To minimize the energy consumption of local inference on MCUs, MicroT adopts an approach that relies on model segmentation and joint training. Specifically, it divides the feature extractor into two components: a part model and a full model, enabling stage-wise decision-making and joint inference on the MCU to minimize unnecessary computational overhead. To achieve a rational segmentation of the feature extractor, we introduce a model split fused score, denoted as Z, which serves to evaluate and determine the effectiveness of the segmentation point. Subsequently, we adopt a joint training approach to simultaneously optimize the parameters of both the part model and the full model, further enhancing their performance. Given the

absence of local data in the cloud, we rely on a non-local public dataset as the basis for training and evaluation, analyzing its feature distribution and model behavior to identify the optimal segmentation point, thereby ensuring the general applicability of the segmentation strategy. The definition of Z is given as follows:

$$Z = \frac{3 \times (1 - R_{Acc}^{\text{norm}}) \times G \times R_{MAC}^{\text{norm}}}{(1 - R_{Acc}^{\text{norm}}) + G + R_{MAC}^{\text{norm}}} \tag{4.4}$$

Z is a composite score employed to assess the rationality and effectiveness of the segmentation point within the feature extractor, guiding the decision-making process for model segmentation. R_{Acc}^{norm} represents the normalized ratio of accuracy loss, which quantifies the extent of performance degradation due to segmentation by comparing the predictive accuracy of the part model to that of the full model, thereby reflecting the impact of the split on model accuracy. R_{MAC}^{norm} is the normalized MAC reduction ratio, which measures the reduction in computational complexity by calculating the decrease in the number of MAC operations in the part model relative to the full model, a metric particularly critical for resource-constrained devices. G denotes the normalized gain ratio, capturing the trade-off between accuracy improvement and the increase in MAC operations within the part model; specifically, it evaluates whether the accuracy enhancement achieved by allocating additional computational resources to the part model justifies the cost. This composite score is designed to balance accuracy and computational cost, providing a quantitative foundation for selecting an optimal segmentation point.

In determining the optimal model segmentation point, we jointly consider the predictive performance and computational efficiency of the model, fully recognizing the influence of computational load on energy consumption, as evidenced by prior studies [201, 202]. Accordingly, we employ accuracy (Acc) as the primary metric for evaluating model performance and the number of MACs (denoted as MAC) as the standard for assessing computational load. The procedure is detailed in Algorithm 3: we begin by dividing the complete model into several part models based on its modular structure, and then measure the Acc and MAC for each part model to obtain baseline data on performance and computational cost. Next, we compute the accuracy improvement (ΔAcc) and the variation in MACs (ΔMAC) for each part model relative to its

Algorithm 3 Optimal Model Segmentation Point.

Require: Full model
Ensure: Optimal segmentation index OptimalSplit
$PM \leftarrow$ Split full model into part models by module
$Acc_{\text{full}} \leftarrow$ Accuracy of the full model
$MAC_{\text{full}} \leftarrow$ MAC of the full model
$Z \leftarrow 0$
OptimalSplit $\leftarrow 1$
for $i = 1$ to length of PM **do**
  $Acc_i \leftarrow$ Accuracy of $PM[i]$
  $MAC_i \leftarrow$ MAC of $PM[i]$
  if $i > 1$ **then**
    $\Delta Acc \leftarrow Acc_i - Acc_{i-1}$
    $\Delta MAC \leftarrow MAC_i - MAC_{i-1}$
    $G \leftarrow \text{Normalize}(\Delta Acc / \Delta MAC)$
  end if
  $R_{Acc} \leftarrow (Acc_{\text{full}} - Acc_i)/Acc_{\text{full}}$
  $R_{Acc}^{\text{norm}} \leftarrow \text{Normalize}(R_{Acc})$
  $R_{MAC} \leftarrow (MAC_{\text{full}} - MAC_i)/MAC_{\text{full}}$
  $R_{MAC}^{\text{norm}} \leftarrow \text{Normalize}(R_{MAC})$
  $Z_{\text{score}} \leftarrow \frac{3(1-R_{Acc}^{\text{norm}})GR_{MAC}^{\text{norm}}}{(1-R_{Acc}^{\text{norm}})+G+R_{MAC}^{\text{norm}}}$
  if $Z_{\text{score}} > Z$ **then**
    $Z \leftarrow Z_{\text{score}}$
    OptimalSplit $\leftarrow i$
  end if
end for
return OptimalSplit

predecessor, using these values to derive the normalized gain ratio (G), which reflects the trade-off between performance enhancement and computational overhead. Additionally, Algorithm 3 assesses the accuracy reduction rate (R_{Acc}) and MAC reduction rate (R_{MAC}) of each part model compared to the complete model, normalizing these rates to maintain a consistent evaluation standard. By combining these metrics, we calculate a composite score (Z_{score}) for each part model, which serves as the basis for selecting the segmentation point. After computing Z_{score} for all part models, we identify the split point with the highest score Z as the optimal segmentation point, thereby determining the part model for embedding and the complete model, as illustrated in Stage 2 of Fig. 4.1.

Following the model segmentation, we conduct joint training building upon the pre-trained feature extractor derived from the SSKD approach, as depicted in Stage 3 of Fig. 4.1. To ensure that the feature representations of the part model align with those of the teacher model, we introduce a matching layer that adjusts the feature dimensions of the part model accordingly. Subsequently, we refine both the part model and the full model using a composite loss function,

L_{JT}:

$$\mathcal{L}_{\mathrm{JT}} = \sum_{k} [\alpha_1 * \mathcal{L}_{\mathrm{MSE}}(o_f, o_t) + \alpha_2 * \mathcal{L}_{\mathrm{MSE}}(o_p, o_t) + \mathcal{L}_{\mathrm{CE},f} + \mathcal{L}_{\mathrm{CE},p}] \tag{4.5}$$

Here, α_1 and α_2 are weighting parameters that adjust the impact of each loss term, o_f indicates the output logits from the full model, o_p signifies the output logits from the part model, and o_t corresponds to the output logits from the teacher model. $\mathcal{L}_{\mathrm{MSE}}$ represents the mean squared error loss between the two variables behind, $\mathcal{L}_{\mathrm{CE},f}$ is the cross-entropy loss between the full model's predicted outputs and the true labels, and $\mathcal{L}_{\mathrm{CE},p}$ is the cross-entropy loss between the part model's predicted outputs and the true labels. Throughout the training phase, as both the part model and the full model are optimized to mirror the embedding features of the teacher model, their feature representations become well-aligned, maintaining consistency across the feature space.

4.3.5 Classifier Training

The feature extractor, trained in the cloud with `int8` quantization and exhibiting only approximately 0.5% accuracy degradation (as elaborated in Section 4.6), is deployed onto the MCU. The MCU then locally constructs and optimizes classifiers to handle personalized local data, as illustrated in Stage 4 of Fig. 4.1. In contrast to cloud servers or mobile devices, MCUs lack an operating system and are incapable of supporting high-level programming languages like Python for machine learning model deployment. Consequently, the deployment, inference, and training of classifiers on the local MCU are entirely implemented using C/C++. To achieve this, we develop the necessary code leveraging AI development toolkits provided by commercial MCUs, such as STM32 X-CUBE-AI. The essential steps are:

(1) Define the classifier's structural design, encompassing the number of layers, the quantity of neurons in each layer, the input data and desired outputs of the neural network, the arrangement of layers (excluding the input layer), and the learning rate parameter used during training; (2) construct the parameter-related matrix frameworks, including the weight matrix, bias vector, output vector, and error term (expressed as the partial derivative of the total error relative

to the weighted sum); (3) designate the type of activation function to modulate the network's nonlinear expressive capacity; (4) craft a function to load datasets and pre-trained models, ensuring accurate input of data and model parameters; (5) design a model construction function to manage memory allocation and initialize the network structure; (6) implement a training function that incorporates the computational processes of forward and backward propagation to refine model parameters; (7) establish a function to store the trained model, enabling the preservation of model parameters post-training; (8) provide a function to de-allocate memory, freeing resources after task completion to prevent memory retention.

To alleviate memory constraints during classifier training, the MCU frees up the memory used by the feature extractor before starting the training process, retaining only the feature data it has produced to enhance memory efficiency. Owing to the constrained memory resources of the MCU, the batch size for training is fixed at 1 to prevent memory overflow caused by data accumulation. To further conserve energy, we employ a staged training strategy that enables collaborative optimization of the classifiers for both the part model and the full model. The specific workflow is as follows: when input data enters the model, it is initially processed by the part model, which updates the parameters of the part model's classifier. During this step, we capture and store the intermediate outputs from the layer immediately preceding the part model's classifier. These stored intermediate results are then utilized as the starting point for subsequent inference, facilitating the continued training of the full model's classifier.

Given that local data acquisition may occur intermittently, we incorporate considerations for data storage into the design to ensure seamless classifier training. MicroT does not require retention of raw data; instead, it stores only the feature information derived from the raw data by the feature extractor. Once the raw data is processed through the feature extractor, an embedding feature vector with a dimension of 512 is produced. Storing this feature in `float32` format within Flash memory requires just 2KB per feature. This storage demand is well within the capabilities of the Flash memory found in commercial MCUs. For instance, deploying MCUNet [179] on the STM32L4R5ZI, which is equipped with 2MB of Flash memory, occupies approximately 0.91MB. Under these conditions, approximately 1.09MB of Flash memory remains available, sufficient to accommodate about 500 additional local sample features.

Algorithm 4 Stage-Decision.

Require: Training dataset $\mathcal{D}_{\text{train}}$ (size N), Inference dataset $\mathcal{D}_{\text{infer}}$ (size M), Part model function f_p, Full model function f_f, Adjustment factor AdjustFactor
Ensure: Stage-decision outputs $\mathcal{O}$
$\mathcal{C} \leftarrow []$ {Confidence scores}
$\mathcal{O} \leftarrow []$ {Final outputs}
Determine median confidence threshold:
for $i = 1$ to N **do**
    $o_{i,p}, c_{i,p} \leftarrow f_p(\mathcal{D}_{\text{train}}[i])$
    Append $c_{i,p}$ to $\mathcal{C}$
end for
Sort $\mathcal{C}$ in ascending order
if $N \bmod 2 = 1$ **then**
    $c_{\text{med}} \leftarrow \mathcal{C}\left[\frac{N+1}{2}\right]$
else
    $c_{\text{med}} \leftarrow \frac{\mathcal{C}\left[\frac{N}{2}\right]+\mathcal{C}\left[\frac{N}{2}+1\right]}{2}$
end if
$Threshold \leftarrow c_{\text{med}} \times \text{AdjustFactor}$
Stage-decision on inference samples:
for $i = 1$ to M **do**
    $o_{i,p}, c_{i,p} \leftarrow f_p(\mathcal{D}_{\text{infer}}[i])$
    if $c_{i,p} < Threshold$ **then**
        $o_{i,f} \leftarrow f_f(o_{i,p})$
        $\mathcal{O}[i] \leftarrow Softmax(o_{i,f})$
    else
        $\mathcal{O}[i] \leftarrow Softmax(o_{i,p})$
    end if
end for
return $\mathcal{O}$

4.3.6 Stage-Decision

Upon completion of local classifier training, the MicroT system employs a stage-decision mechanism to further minimize energy consumption during the inference phase, as illustrated in Stage 5 of Fig. 4.1. Stage-decision operates as an efficient early-exit strategy, wherein the part model initially processes the input data and computes corresponding confidence scores to determine the necessity of additional computation. Specifically, if the confidence score does not meet a pre-established threshold, the full model takes over, resuming inference from the intermediate results immediately preceding the part model's output layer to ensure prediction accuracy. Data preprocessing during local training and stage-decision should align with the preprocessing methods used in the cloud during the training of the feature extractor.

Establishing an appropriate threshold plays a pivotal role in the system. To achieve this, we calculate and set the threshold using a median-quartile approach, leveraging the sequence of confidence scores produced by the part model. This method analyzes the quartile value of

the confidence score sequence to ascertain the proportion of input samples that can be processed solely by the part model without necessitating the involvement of the full model—for instance, a quartile value at the median yields a proportion of 0.5. We designate this proportion as the stage-decision ratio and implement a dynamic adjustment mechanism through the $AdjustFactor$ function, as detailed in Algorithm 4. The $AdjustFactor$ function empowers users to flexibly balance prediction accuracy and energy consumption based on the requirements of their specific application scenarios. Increasing the value of $AdjustFactor$ reduces the stage-decision ratio, meaning a greater number of samples are directed to the full model for processing, thereby enhancing the overall prediction accuracy of the model. Conversely, decreasing the $AdjustFactor$ value elevates the stage-decision ratio, allowing more samples to be handled exclusively by the part model, which in turn reduces the system's energy consumption. Our experimental findings (presented in detail in Section 4.5) demonstrate that the median and the corresponding threshold can be precisely established using only a small number of samples, thereby preventing excessive resource utilization. To evaluate the effectiveness of stage-decision, we designate the median confidence score—where exactly 50% of the input samples are processed solely by the part model without invoking the full model—as the 'standard stage-decision ratio'.

4.3.7 Partial Quantization

For MCUs, two primary quantization strategies are commonly employed. The first strategy, known as post-quantization, involves initially training the model on a high-performance platform before converting its weight parameters to the int8 format to accommodate the resource constraints of the MCU. However, models processed through post-quantization exhibit a fixed nature, meaning their parameters cannot be further adjusted or refined on the MCU, which substantially hinders the feasibility of local fine-tuning. An alternative approach is quantization-aware training, which incorporates quantization constraints during the training phase to prepare the model for low-precision computation environments. Yet, given the inherent limitations of MCUs in terms of memory capacity and computational power, implementing this method directly on the device poses considerable challenges and is impractical for real-time training needs.

Consequently, we seek to devise a balanced approach that harnesses the benefits of quantization—such as improved energy efficiency and reduced resource demands—while preserving the capability for low-energy, personalized training on the MCU.

Given that we partition the model into feature extractors and classifiers, with the expectation that the feature extractors require only static inference while the classifiers are trained locally on the MCU, this structural paradigm organically directs us toward a 'partial quantization' strategy. Specifically, this approach entails applying post-quantization exclusively to the feature extractors, converting their parameters to the `int8` format to enhance resource efficiency, while leaving the classifiers in their original floating-point format without quantization constraints. The motivations for this design are as follows:

• In most NN architectures, the feature extractor typically accounts for the predominant share of parameters and computational workload. Quantization to the feature extractor can diminish memory usage and energy consumption while expediting the processing speed during inference, thereby achieving greater resource utilization efficiency on the MCU.

• The feature extraction process generally demonstrates robust resilience, capable of withstanding a certain degree of precision degradation or approximation, whereas classifiers exhibit heightened sensitivity to parameter adjustments, especially in scenarios involving multi-class tasks or subtle inter-class distinctions [203]. Consequently, quantizing solely the feature extractor enables a balance between preserving model prediction capabilities and enhancing operational efficiency, establishing a practical trade-off between performance and resource demands. Moreover, the precision loss in the feature extractor resulting from quantization can be mitigated through subsequent local fine-tuning of the classifier.

• Personalized training typically relies on floating-point operations to ensure computational precision, yet quantization-aware training is challenging to implement efficiently on MCUs due to limitations in memory capacity and processing power. To address this, we quantize the feature extractor to the `int8` format to reduce resource requirements, while retaining the classifier in the `float32` format to facilitate flexible adjustments. This hybrid quantization strategy not only supports localized personalization of the classifier on the MCU but also

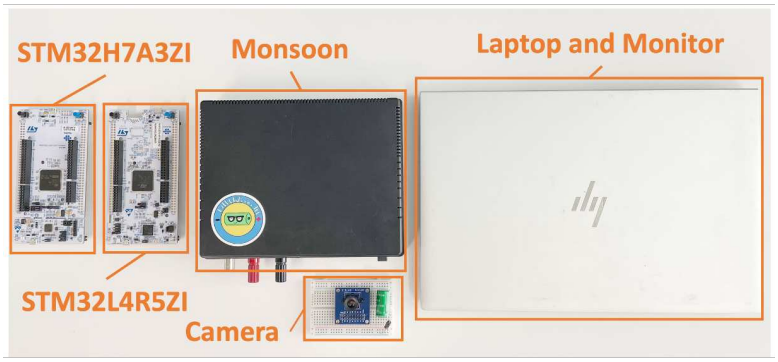


Figure 4.2: MicroT prototype overview.

aligns with low-power operational needs.

The primary procedure of quantization involves mapping continuous floating-point data into discrete 8-bit integer representations, thereby adapting it to low-precision computational environments. This process is mathematically expressed as follows [193]:

$$value_q = \text{round}\left(\frac{value_o - \min}{\text{scale}}\right) + \text{zero_point} \tag{4.6}$$

Where $value_q$ is the values after quantization, $value_o$ denotes the original floating-point value, representing the unprocessed input data prior to quantization; min and max signify the minimum and maximum values within the tensor to be quantized, delineating the dynamic range of the data; $scale$ is a fixed scaling factor, calculated as $scale = (max - min)/255$, which maps the floating-point range onto the 255 discrete levels available in an 8-bit integer. $zero_point$ is an offset to position the zero value of the floating-point range as close as possible to the midpoint of the 8-bit integer range, ensuring a balanced representation of both positive and negative values within the constrained scope.

4.4 Implementation & Evaluation Setup

Here, we introduce the implementation, models, datasets, performance metrics, and baselines.

Table 4.1: The experimental MCU boards.

MCU Board	Processor	Flash	RAM	Max Freq.
STM32H7A3ZI	Arm Cortex-M7	2MB	1.18MB	280MHz
STM32L4R5ZI	Arm Cortex-M4	2MB	640KB	120MHz



Figure 4.3: Main structure of ProxylessNAS and MCUNet.

4.4.1 MicroT Prototype

We perform experiments on the STM32 series of MCUs, selected for their status as representative and well-established platforms in the commercial domain. To deploy MicroT, we leverage STM32's ML development toolkit, X-CUBE-AI [204], which integrates essential functionalities for implementing machine learning models on commercial MCUs, such as model inference capabilities and support for `INT8` quantization to optimize computation and storage requirements. Building upon the code framework provided by X-CUBE-AI, we customize its generated $C/C++$ code to enable data preprocessing, model training and inference processes, dynamic threshold tuning, and the execution of the stage-decision mechanism. Specifically, we execute MicroT on the STM32H7A3ZI and STM32L4R5ZI development boards, chosen as they exemplify high-performance and low-power MCUs, respectively, with their detailed specifications outlined in Table 4.1. To maintain controlled experimental conditions, we employ the Monsoon High Voltage Power Monitor [205] to stabilize the MCU supply voltage at 1.9V, a configuration informed by prior work [206]. Additionally, we configure the operating frequency of both MCU boards to 120MHz to meet the performance demands of on-device inference and training. An overview of the MicroT prototype system is presented in Fig. 4.2. While a camera is an indispensable component in the fully operational system, we streamline the experimental setup and focus on core functionality validation by simulating image data acquisition via serial communication, thereby reducing reliance on external hardware.

4.4.2 Models and Datasets

Similar to Chapter 3, this chapter also uses image data for experiments. This consideration is twofold: first, it inherits and extends the research on model personalization from Chapter

3; second, the image domain offers abundant open-source models and datasets, facilitating the validation of MicroT. Additionally, we did not use the Cifar series, as its feature distribution is similar to ImageNet. We carefully selected three other datasets with feature distributions significantly different from ImageNet. In terms of model selection, since the MCU used in this chapter is more resource-constrained than the mobile device (Raspberry Pi) used in Chapter 3, we opted for more lightweight models (details below).

We designate a Vision Transformer (ViT)-based model architecture, termed ViT-S [38], as the teacher model to guide subsequent knowledge transfer. For the student models, intended for deployment on micro-controller units (MCUs), we opt for ProxylessNAS_w0.3 [207] and MCUNet_int3 [179]. These architectures represent widely adopted lightweight CNNs and modular designs, renowned for their ability to strike an effective balance among prediction accuracy, memory usage, and energy consumption [208, 209]. Specifically, both MCUNet and ProxylessNAS are fundamentally built upon Mobile Inverted Bottleneck Convolutions (MBConv), a structure that optimizes convolutional operations to significantly reduce computational complexity and resource demands, with its design details illustrated in Fig. 4.3.

The datasets are classified into two distinct categories: cloud datasets and local datasets. For the cloud datasets, we employ ImageNet as the primary resource, facilitating various stages including SSL teacher model training, SSKD, model segmentation, and joint training. ImageNet, with its diverse and abundant sample content, provides a robust foundation for the feature extractor to acquire generalizable features from the teacher model. Regarding the local datasets, we incorporate several representative datasets to emulate the characteristics of personalized local data, namely The Oxford-IIIT Pet (referred to as Pet [210]), CUB-200-2011 Bird (referred to as Bird [211]), and PlantCLEF 2017 (referred to as Plant [212]). These three datasets exhibit no overlap with ImageNet and possess different data distributions. Throughout the experiments, we standardize the input image resolution to 224×224.

The Pet dataset [210] includes 3,680 images covering 37 different pet species. The Bird dataset [211] consists of 11,788 images representing 200 distinct bird categories, showcasing significant categorical variety. As for the Plant dataset [212], we carefully curated a subset

based on the number of images per category, resulting in 20 plant species with a total of 11,660 training images. We randomly partition these datasets into training, testing, and validation sets at a ratio of 8:1:1.

On the cloud side, we conduct SSKD training from scratch on the feature extractor, spanning 50 epochs with a learning rate of 0.01, employing the stochastic gradient descent (SGD) optimizer to ensure effective parameter updates. Subsequently, we partition the pre-trained feature extractor into part and full models, followed by an additional joint training phase lasting 10 epochs, utilizing a reduced learning rate of 0.005 while continuing to rely on the SGD optimizer for parameter refinement. It is noteworthy that, to facilitate the implementation of SSKD and joint training, we incorporate a matching layer prior to the task head of both the part and full models. This layer adjusts the feature dimensions to align with those of the teacher model's embedding features. Upon completion of training, this matching layer is discarded, preserving only the original feature extraction components of the part and full models. On the local MCU side, we deploy these trained models and equip each—the part model and the full model—with a single-layer task head tailored to local tasks. Given the constrained memory capacity of the MCU, we fix the batch size at 1, adopting the SGD optimizer with a learning rate of 0.1 [43] to achieve efficient local optimization under resource-limited conditions.

4.4.3 Performance Metrics

Model Performance. We employ prediction accuracy as the metric to gauge the performance of MicroT when processing personalized local data.

System Cost. Our primary focus lies in assessing the energy consumption associated with local training and inference tasks on MCUs. To this end, we utilize the Monsoon High Voltage Power Monitor [205] with a sampling frequency of $50Hz$ to precisely measure and analyze energy usage. During the experiments, we configure the Monsoon device [205] to maintain a stable supply voltage of 1.9V, a setting informed by prior work [206], while recording the operational time and the average power. Using these values, we compute the average energy consumption by the product of time and power, to quantify the system's energy profile. To ensure the reliability and stability of the measurements, we exclusively use current data collected after the

system has been running for 1 minutes, thereby mitigating potential fluctuations during the initial phase. All reported results represent the average of ten independent experimental runs.

4.4.4 Baselines

In evaluating model performance, we conduct a comparative analysis of the MicroT system against four baseline approaches to examine its effectiveness:

(1) *None* refrains from any on-device training, initializing the local classifier with random parameters;

(2) *TTL* employs a conventional transfer learning approach, fine-tuning solely the task head without optimization on the feature extractor [203];

(3) *TinyTL* enhances the model backbone by incorporating lightweight residual modules, fixing the backbone parameters during training and adjusting only the added components [44];

(4) *SparseUpdate* pre-selects layers and channels for updates through multiple gradient computations, subsequently performing online optimization of these designated portions on the device [43].

To clearly illustrate the performance enhancements introduced by MicroT, we standardize the feature extractor across MicroT and all baseline methods, utilizing both MCUNet and ProxylessNAS models. Additionally, to guarantee equitable experimental conditions, we keep the number of training epochs uniform across all methods and methodically identify the optimal learning rate for each using a grid search technique. To thoroughly evaluate the maximum performance of these approaches, we utilize the full dataset, dividing it into training, testing, and validation sets in an 8:1:1 ratio. Although the stage-decision ratio in MicroT is adjustable, allowing for flexible configuration based on specific needs, we adopt the standard stage-decision ratio of 0.5 in these experiments for direct comparison with the baseline methods.

In assessing the system cost associated with local training, we initially selected TinyTL, TTL, and SparseUpdate as baseline methods for comparison. However, our experimental analysis revealed that SparseUpdate's layer and channel selection process necessitates full access to the

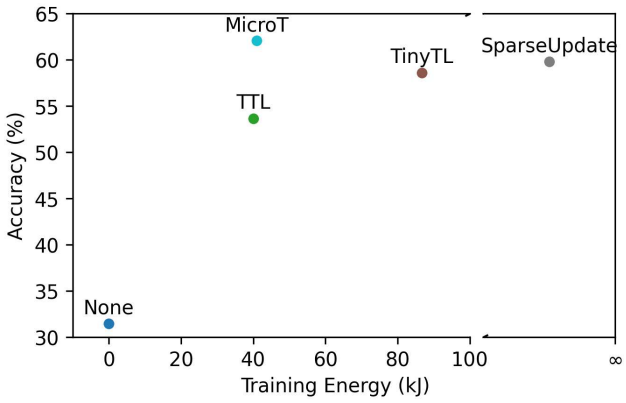


Figure 4.4: The overall comparison between MicroT (stage-decision 0.5) and the baselines.

entire local dataset and involves thousands of computationally demanding search operations [43] or pruning procedures [178], significantly prolonging the training duration, with reports indicating a training time of up to ten days [47]. Moreover, due to the static nature of SparseUpdate's optimization approach, the introduction of a new task head requires repeating this intricate and resource-intensive search process from scratch, substantially diminishing its flexibility and efficiency in practical applications. Consequently, we opted to exclude SparseUpdate from the system cost comparison, focusing instead on evaluating MicroT against TinyTL and TTL.

When evaluating the system cost of local inference, we observed that TinyTL and SparseUpdate predominantly rely on conventional full model inference techniques during the inference phase, closely resembling the operational approach of TTL. This similarity implies a lack of substantial differentiation in terms of energy consumption and computational resource demands among these methods. Given this insight, to more effectively highlight the distinctive advantages of MicroT, we opted to compare only MicroT—which employs an early-exit inference strategy—with TTL, which adheres to traditional full model inference.

4.5 Results

4.5.1 Overall Performance

We first present an overall comparison between MicroT (stage-decision ratio 0.5) and baselines. To provide an intuitive display of results across all datasets and models, we use the **average** values of accuracy and local training energy cost from all experimental datasets and models.

Table 4.2: Results of model performance with varying batch sizes.

Batch Size	Feature Extractor	Accuracy
128 (GPU)	MCUNet	80.24±0.23
	ProxylessNAS	77.45±0.27
1 (MicroT)	MCUNet	79.31±0.31
	ProxylessNAS±0.26	76.28±0.33

Additionally, since SparseUpdate requires a complex parameter search process on the MCU, its local training energy cost far exceeds that of other methods involved in the experiments. Therefore, we did not measure its specific value and instead represent it with a value in the range $[100, +\infty]$ in the figure. The results are shown in the Fig. 4.4.

We observe that 'None' requires no local fine-tuning, resulting in zero energy consumption, but achieves only 31.47% accuracy. In contrast, MicroT attains an average accuracy of 62.05%, surpassing TinyTL (58.60%) and SparseUpdate (59.79%) by 3.45 and 2.26 pp, respectively, while reducing energy consumption significantly from TinyTL's 86.73 kJ to 40.98 kJ. TTL's energy consumption (40.00kJ) is close to MicroT's, but its accuracy is only 53.65%, far below MicroT. Overall, MicroT achieves the optimal balance between energy consumption and accuracy. In subsequent sections, we will provide a detailed analysis and presentation of energy consumption and accuracy.

4.5.2 MicroT Local Training *vs* Cloud-Level Model Training

Given the memory limitations, we set the batch size to 1 during classifier training on the MCU, allowing the training process to function effectively within the constrained resource setting. However, this configuration markedly diminishes the experimental efficiency, as a single-batch approach fails to leverage the parallel processing capabilities of the hardware. To tackle this constraint and optimize the experimental process, we conduct a comparative analysis at the start of the experiments, investigating the performance gap between classifier training on the MCU with a batch size of 1 and training on a GPU with a batch size of 128. This investigation aims to determine whether the latter can serve as a viable approximation of the former, thereby enhancing the overall efficiency of the experiments. The training configurations on both the MCU and the cloud are maintained consistently, as outlined in Section 4.4.2. We exclude the use of momentum, as it offers no performance advantage in a single-batch training scenario

and may, in fact, exacerbate memory usage due to additional computational requirements [43]. The classifier training on the MCU is implemented using $C/C++$ to suit the embedded environment, whereas the GPU-based training leverages *Python* to capitalize on its efficient computational framework. Given the reduced efficiency inherent in single-batch training, we sampled 40 images randomly from each class in the Pet dataset, resulting in a total of 1480 images.

The results are presented in Table 4.2. Through analysis, we observe that the model accuracy achieved via single-batch training on the MCU closely aligns with that obtained from training on the GPU with a batch size of 128. This finding enables us to leverage the results from batch training on the GPU for evaluation, thereby enhancing experimental efficiency. Consequently, unless explicitly specified otherwise, we adopt the classifier training results derived from the GPU as the default reporting standard, grounded in the classifier training configuration described earlier.

4.5.3 Model Performance

In these experiments, we evaluate the accuracy of the *MicroT* model, comparing it against the baseline methods outlined in Section 4.4.4. To provide a comprehensive overview, we begin by detailing the deployment process of MicroT, encompassing the critical steps from model preparation to implementation on the target device. Subsequently, we conduct a comparison of the model performance between MicroT and the baselines.

Upon completing the training of the SSL teacher model and applying SSKD to the feature extractor, we proceed with the model segmentation. The optimal segmentation point is identified using the model segmentation fused score, with its computation and application thoroughly described in Section 4.3.4. It is important to emphasize that, during this process, we aim to prevent the part model from being excessively small, which could significantly constrain its performance, while also avoiding the model depth too similar to the full model, as this would yield negligible improvements in system cost optimization. Accordingly, for the MCUNet model, we restrict our analysis to modules 4 through 14, and for the ProxylessNAS model, we examine modules 6 through 17, ensuring that the segmentation point strikes a reasonable balance be-

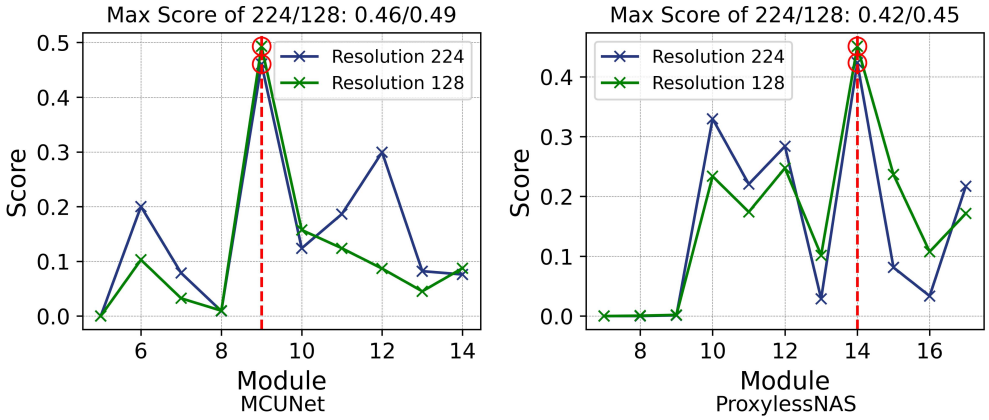


Figure 4.5: The segmentation fused scores for MCUNet and ProxylessNAS, with the X-axis indicating the part model up to a specific module and the Y-axis showing the fused score.

tween performance and efficiency. The resulting model segmentation fused scores for MCUNet and ProxylessNAS are presented in Fig. 4.5.

Through experimental analysis, we ascertain that the optimal model segmentation points, as determined by Algorithm 3, occur at the 9th module for MCUNet and the 14th module for ProxylessNAS. In both model architectures, the number MACs typically exhibits a consistent upward trend as the network depth increases. However, at these optimal segmentation points, we observe a marked enhancement in prediction accuracy, suggesting that the models effectively capture essential features at these specific depths. We believe that this depth represents an optimal trade-off between computational cost and performance efficacy. To further substantiate this finding, we also conduct a model segmentation evaluation at an input resolution of 128, revealing that the optimal segmentation points remain consistent at the 9th and 14th modules for MCUNet and ProxylessNAS, respectively. We also further explore this phenomenon in ablation experiments (Section. 4.6).

Based on the foregoing analysis, we execute model segmentation at the 9th module of MCUNet and the 14th module of ProxylessNAS, thereby generating the corresponding part and full models. These models are subsequently subjected to further joint training on the ImageNet dataset to enhance their performance. The performance of each model under conditions with (W/) and without (W/O) joint training are illustrated in Fig. 4.6. We can find that joint training impacts the performance of the part and full models to varying extents. Notably, the

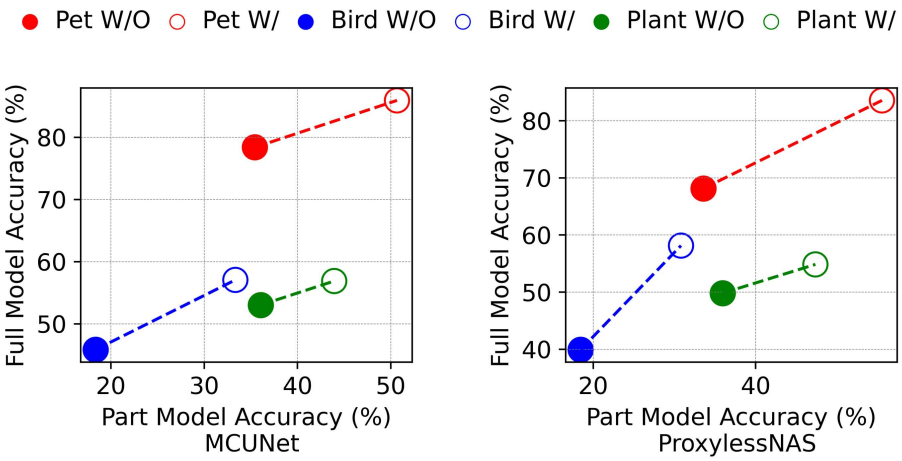


Figure 4.6: The performance enhancement achieved through SSKD joint training for the part and full models, with dashed connecting lines highlighting the variations in model performance.

Table 4.3: Top-1 accuracy results of MicroT and the baselines. MicroT achieves the highest accuracy with two DNN architectures on three cross-domain datasets.

Model	Method	Pet	Bird	Plant	Avg.
MCUNet	None	42.53±0.12	20.14±0.85	35.31±0.38	32.66
	TTL	74.20±0.37	43.84±0.27	47.01±0.45	55.02
	TinyTL	76.21±0.31	47.12±0.69	**53.81±0.23**	59.05
	SparseUpdate	79.09±0.49	49.77±0.25	50.73±0.58	59.86
	MicroT (Ours)	**81.21±0.35**	**52.85±0.42**	52.92±0.51	**62.33**
ProxylessNAS	None	37.41±0.62	19.44±0.53	33.97±0.77	30.27
	TTL	70.12±0.73	39.91±0.28	46.80±0.55	52.28
	TinyTL	78.04±0.39	48.12±0.47	48.28±0.61	58.15
	SparseUpdate	**81.21±0.37**	47.83±0.68	50.13±0.33	59.72
	MicroT (Ours)	80.33±0.51	**51.51±0.57**	**53.45±0.42**	**61.76**

accuracy improvement observed in the full model is marginally less pronounced than that in the part model. We hypothesize that this phenomenon arises from a confluence of factors: (1) During joint training, the introduction of a segmentation point and the inclusion of the part model may lead to a certain degree of information loss, potentially compromising the feature representation capacity of the full model; (2) The accuracy gains achieved by the part model can positively influence the full model through due to the feedback mechanism, propagating beneficial effects to its performance. However, though the presence of information loss in the full model, this trade-off is still acceptable when weighed against the substantial performance enhancements observed in both the part and full models, coupled with the energy savings

realized during local inference, as elaborated in Section 4.5.5.

To comparison with the baseline methods, we configure MicroT with a standard stage-decision rate of 0.5, and the experimental results are presented in Table 4.3. Analysis of these results reveals that, the None method consistently exhibits the lowest prediction accuracy across all experiments. This observation underscores the critical need for personalization when deploying models locally on MCUs. In contrast to None, the TTL demonstrate a substantial performance enhancement, affirming the efficacy of the transfer learning paradigm in addressing local personalization challenges. However, when TTL compares with TinyTL and SparseUpdate, TTL reveals a performance gap, suggesting that training only the classifier can not reach the satisfied performance to the local personalized data. Nevertheless, we do not entirely discard the 'training only the classifier' approach, as it demands the least computational resources and energy consumption on MCUs (as detailed in Section 4.5.5), retaining its merit in resource-constrained environments. Instead, MicroT shift focus toward enhancing the quality of the feature extractor, aiming to overcome the limitations of solely training the classifier and elevate the overall system performance.

According to Table. 4.3, the experimental results demonstrate that $MicroT$ attains the highest prediction accuracy across the majority of evaluated datasets and models, also securing the top position in overall average accuracy. Specifically, MicroT surpasses TTL, TinyTL, and SparseUpdate by margins ranging from 7.01 to 11.60 pp, 2.29 to 5.73 pp, and 2.12 to 3.68 pp, respectively. This advantage underscores that MicroT delivers robust model performance sufficient to support on-device personalization on MCUs, effectively meeting the demands of local adaptation. Furthermore, the most compelling advantage of MicroT lies in its substantially reduced energy consumption (as elaborated in Section 4.5.5), coupled with superior performance when compared to baseline methods such as TTL, TinyTL, and SparseUpdate. Unlike these approaches, MicroT does not need parameter search processes (as required by SparseUpdate) or the incorporation of numerous additional lightweight residual modules (as required by TinyTL). By leveraging a straightforward single-layer classifier, MicroT not only achieves performance that is comparable to or exceeds these baseline methods but also exhibits greater efficiency in terms of computational complexity and energy cost.

Table 4.4: The median, stage-decision ratio, and model accuracy across varying sample sizes. N denotes the number of samples used to compute the median value, *Med.* indicates the median value, *Ratio* signifies the stage-decision ratio, and *Acc.* reflects the model accuracy (%).

N	MCUNet									ProxylessNAS								
	Pet			Bird			Plant			Pet			Bird			Plant		
	Med.	Ratio	Acc.	Med.	Ratio	Acc.	Med.	Ratio	Acc.	Med.	Ratio	Acc.	Med.	Ratio	Acc.	Med.	Ratio	Acc.
5	0.688	0.486	81.35	0.408	0.456	53.35	0.467	0.447	52.72	0.558	0.495	79.97	0.389	0.471	49.99	0.448	0.471	52.89
10	0.674	0.504	81.12	0.400	0.465	52.45	0.439	0.484	52.15	0.560	0.493	79.34	0.378	0.489	50.84	0.440	0.481	51.95
20	0.676	0.501	81.17	0.384	0.491	52.76	0.439	0.484	52.15	0.552	0.503	79.79	0.374	0.495	51.49	0.432	0.496	53.29
40	0.679	0.497	81.26	0.386	0.488	52.74	0.436	0.491	52.49	0.557	0.495	79.88	0.369	0.502	51.35	0.429	0.499	53.38
All	0.677	0.500	81.21	0.378	0.500	52.85	0.430	0.500	52.92	0.554	0.500	80.33	0.371	0.500	51.51	0.428	0.500	53.45

4.5.4 Determination of Confidence Thresholds

Confidence thresholds directly affect the stage-decision ratio, and thus the proportion of samples that are fully processed by the part model. The threshold is calculated by the confidence score sequence produced by the part model. Specifically, the part model first evaluates N samples to generate a sequence of confidence scores, from which the median is extracted as the reference value to establish the standard stage-decision ratio of 0.5. Theoretically, under ideal conditions, computing the median by the entire test sample set would precisely calibrate the stage-decision ratio to 0.5, ensuring that 50% of the samples are handled exclusively by the part model. However, in practical real-world contexts, expending substantial time and utilizing the entire local samples to iteratively refine and pinpoint this threshold is infeasible. To address this and explore the practicality of efficiently setting the threshold locally, we conduct an experiment by randomly extracting sample sets of varying sizes (independent of category) from the full dataset, calculating the median confidence scores for each subset, and assessing the resultant stage-decision ratios alongside their effects on model performance. The findings of these experiments are presented in Table 4.4.

Table 4.4 reveals that as the number of samples increases, the median of the confidence score sequence, the stage-decision ratio, and the model's performance progressively converge toward the values derived from utilizing the entire sample set. This trend suggests that a larger sample size facilitates a more precise approximation of the real median value. Notably, however, even when the sample size is limited to just five samples, the deviation from the full-sample results remains minimal and within an acceptable range for practical purposes. Specifically, the average deviation in the median is only 0.020, and the average difference in model accuracy stands at 0.55%. This finding indicates that processing as few as five samples on MCUs is sufficient to efficiently establish the standard stage-decision ratio of 0.5 within a short time.

We attribute this phenomenon to the high distributional consistency of the confidence scores generated by the part model, which reflects both the effectiveness of the feature extractor and the structure of the input data. If the feature space follows an approximately normal or symmetric unimodal distribution, the median—being a robust statistic—can reliably represent

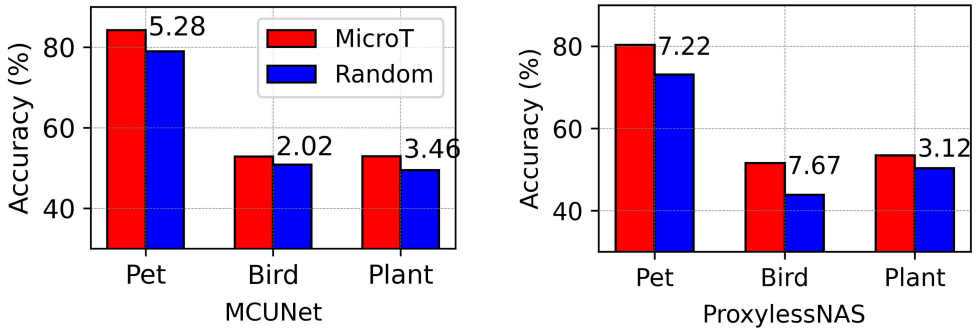


Figure 4.7: Comparison between stage-decision ratio determined by median or random selection.

the central tendency of the confidence distribution, even when estimated from a small number of samples. Notably, such a distributional shape often implies that the feature extractor has learned generalizable and well-clustered representations [213, 214], which is a core design goal of MicroT's cloud-side training using SSKD and joint learning. Furthermore, this structural regularity in the confidence scores reduces the variability of decision outcomes, thereby enhancing the reliability of stage-decision under limited calibration data. Nevertheless, in scenarios involving complex or multi-modal data distributions, a larger number of local samples may be required to achieve the same level of threshold accuracy. We also further discuss this phenomenon in Section. 4.7.

Beyond establishing the threshold via the confidence median, randomly selecting 50% of the samples is also an alternative approach to directly achieve a stage-decision ratio of 0.5. To evaluate the superiority of MicroT's confidence-based threshold strategy, we employ the random selection of 50% samples as a baseline and conduct a comparison experiment. The results, as illustrated in Fig. 4.7, demonstrate that MicroT outperforms the strategy of random sampling, enhancing the average prediction accuracy by 4.80%.

4.5.5 System Cost

MicroT operates in two distinct phases on the MCU: classifier training and model inference. To assess the energy cost associated with the training phase, we conduct a comparative analysis of MicroT against the baseline methods TTL and TinyTL. We exclude SparseUpdate from this comparison, as it demands approximately 10 days to search for the layers and channels to update

and subsequently train the model—a process deemed excessively energy-intensive and impractical for real-world applications [43]. For evaluating the energy cost of the inference phase, we designate TTL as the baseline approach (*i.e.*, full model inference), reflecting the inference strategy commonly employed in most on-device personalization studies. To comprehensively examine energy consumption, we select the Plant dataset [212] and the Bird dataset [211] for testing, as these datasets represent the extremes of class diversity, with the fewest and most numerous categories, respectively, thereby capturing a range of data complexities. To streamline the energy cost analysis while ensuring representative findings, we compute the average energy consumption across these two datasets.

Memory Footprint

We measure the memory footprint of MicroT utilizing the STM32-CUBE-IDE tool [215] to evaluate its resource consumption on MCUs. To demonstrate that MicroT satisfies the memory constraints of commercial MCUs, we adopt the following analytical approach: (1) We utilize the memory footprint of the local training phase as the maximum memory footprint, given that this stage imposes a higher memory demand compared to the inference phase; (2) We focus on the full model, as its memory requirements exceed those of the part model, thereby representing the upper bound of memory consumption. In our experiments, conducted at an input resolution of 224, the maximum RAM usage for MCUNet is 614.40 KB, while ProxylessNAS is 624.64 KB. Regarding flash memory usage, MCUNet consumes approximately 0.91 MB, and ProxylessNAS approximately 0.70 MB. These findings substantiate that MicroT's memory footprint remains well within the capacity of commercial MCUs.

Local Training

We utilize the Monsoon Power Monitor [205] to measure the latency and average power consumption during the local training process, subsequently calculating the corresponding energy consumption values, as described in Section 4.4.3. To ensure clarity and consistency in comparing and quantifying energy usage, we adopt a representative training scenario: training 100 local data samples over 40 epochs, amounting to a total of 4,000 iterations. This configuration is designed to emulate a typical local training task. Throughout the measurement, we exclude

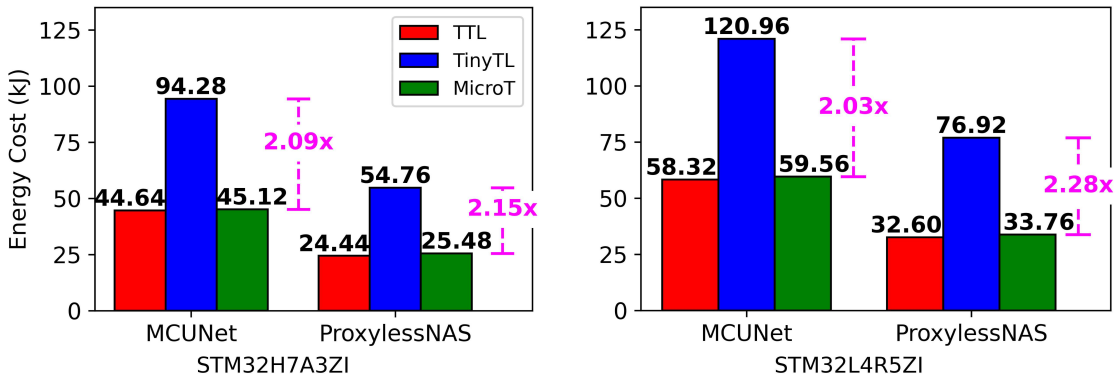


Figure 4.8: Energy cost of local training.

the energy consumption attributable to other MCU components or non-training operations, concentrating solely on the core elements of the local training phase—namely, data preprocessing, inference by the feature extractor, and parameter optimization of the classifier. The results are shown in Fig. 4.8.

We can find that MicroT exhibits no substantial deviation from TTL in terms of energy cost. Although MicroT necessitates the training of two classifiers—one for the part model and one for the full model—potentially increasing the computational burden, we mitigate this through the adoption of a stage-training strategy designed to conserve energy. Specifically, during training, each sample is first processed by the part model to train the part model's classifier, with the intermediate outputs from the layer immediately preceding this classifier being preserved. These pre-stored intermediate values are subsequently reused, allowing the full model's classifier training to proceed without redundant feature extraction, thereby reducing additional energy cost. Consequently, the primary distinctions in energy consumption between MicroT and TTL stem from the training of the part model classifier and the modest overhead associated with storing and transmitting a limited number of data. Furthermore, MicroT employs a simple single-layer classifier, which further curbs energy cost increase. In contrast, TinyTL incurs significantly higher energy costs, attributable to the need to train multiple lightweight residual modules, which substantially increase computational resource demands. Compared with TinyTL, MicroT achieves energy savings ranging from 2.03 to 2.28× for model training, an improvement credited to its simplified model architecture and the efficient training strategy.

Table 4.5: The energy costs (kJ) of the MCU local inference. MicroT achieves energy saving on two boards and two models.

Model	Method	STM32H7A3ZI	STM32L4R5ZI	Avg.
MCUNet	TTL	9.52	12.42	10.97
	MicroT	**8.22**	**10.54**	**9.38**
ProxylessNAS	TTL	4.64	6.19	5.42
	MicroT	**4.01**	**5.31**	**4.66**

Local Inference

Following the analysis of energy consumption in the local training, we proceed to evaluate the energy cost of the inference stage. We designate TTL (*i.e.*, full model inference) as our baseline for comparison, a decision driven by its prevalent use across most research on on-device personalization. Similarly with the experiments for the local training, we solely focus on the energy cost during model inference, deliberately excluding the energy consumption attributable to other MCU components or operations unrelated to inference. The MCU performs 1,000 model inference tasks, a scale intended to emulate high-frequency inference scenarios encountered in real-world applications. The results are presented in Table 4.5.

According to Table. 4.5, we initially note that MicroT achieves a substantial reduction in energy consumption during model inference across both development boards and model architectures. When compared with the baseline TTL, MicroT demonstrates an average energy reduction of 14.17%, underscoring its potential to reduce energy cost over prolonged operational periods—a particularly valuable attribute for resource-constrained MCU applications. Additionally, we find that MCUNet exhibits higher energy consumption compared to ProxylessNAS, a disparity attributable to the more complex computational operations in MCUNet, which elevate its energy demands. Furthermore, at the hardware level, we find that for the same model, the STM32L4R5ZI MCU incurs higher energy consumption than the STM32H7A3ZI, despite the former being engineered as a lower-power device. This counterintuitive outcome stems from the extended runtime of the STM32L4R5ZI, which negates the energy-saving benefits of its low-power design. This observation highlights a trade-off between runtime and power consumption in determining overall energy usage, suggesting that optimizing inference energy efficiency necessitates a balanced consideration of computational efficiency and execution duration.

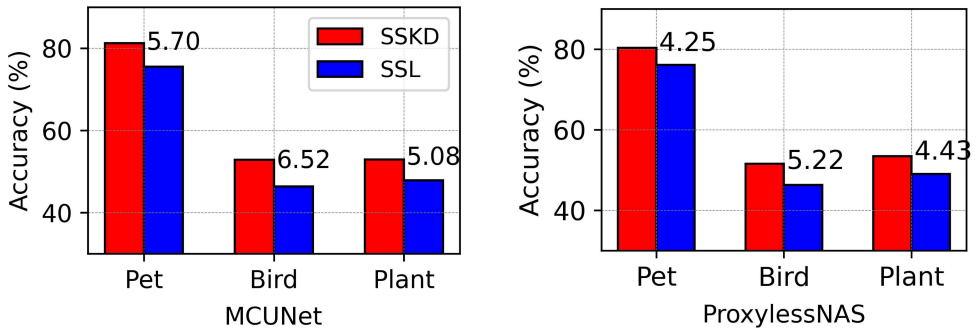


Figure 4.9: The model accuracy between SSL and SSKD.

4.6 Ablation Study

SSL vesus SSKD

We conducted a comparative experiment to assess the disparity in model performance between directly applying SSL to the feature extractor and employing SSKD, with the results shown in Fig. 4.9. We can find that directly implementing SSL on the feature extractor yields a comparatively less enhancement in model performance, whereas SSKD delivers a more pronounced improvement. This difference can primarily be attributed to the inherent characteristics of the models deployed on MCUs, which are typically constrained by their smaller scale, limited parameter capacity, and computational resources. As a result, the performance gains achievable through direct SSL application are curtailed by these structural limitations. In contrast, SSKD adopts a more effective training strategy: it first leverages SSL on a large-scale teacher model to capture a rich set of feature representations, then facilitates knowledge transfer to a smaller student model by enabling it to learn concurrently from both ground-truth labels and the embedded features distilled from the teacher. This approach substantially alleviates the challenge of knowledge acquisition for the small model, allowing it to enhance performance more effectively within its constrained capacity.

Universality of Model Segmentation Points

MicroT uses public datasets (*e.g.*, ImageNet) in the cloud to compute the global model segmentation point, which is then applied to local models and dataset splitting. The main considerations are: (1) Computing the model split point locally on an MCU is too computationally

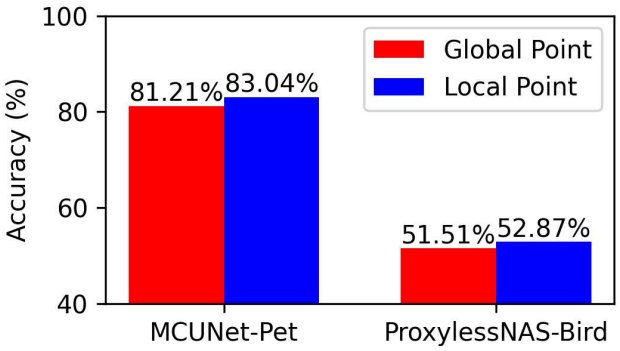


Figure 4.10: The model accuracy between segmentation by global/local points.

expensive, and joint training for fine-tuning after model segmentation is difficult to perform locally; (2) The cloud cannot access local datasets; (3) MicroT's feature extractor is universal, and we hypothesize that the model segmentation point may also have some universality. To verify the universality of the model segmentation point, we computed more precise model segmentation points based on local datasets (Pet, Bird, Plant) and compared them with the global model segmentation point.

We find that for the MCUNet model, the local points for the Bird and Plant datasets align with the global point, all at the 9th Module, while the local point for the Pet dataset shifts to the 10th Module. For the ProxylessNAS model, the local points for the Pet and Plant datasets align with the global point at the 14th Module, but the local point for the Bird dataset moves earlier to the 12th Module. Therefore, we re-split and fine-tuned the MCUNet-Pet and ProxylessNAS-Bird using their local points and compared the model accuracy differences between using the local points and the global points. The results are shown in Fig. 4.10.

We observed that, although global and local points differ for some local datasets, the model accuracy differences remain within an acceptable range. The main reasons for this phenomenon are: (1) The universality of the feature extractor ensures that local and global split points do not deviate significantly, remaining consistent in most cases with minor differences in a few cases; (2) After model splitting, joint training is performed, and even using the global split point, fine-tuning can partially compensate for the differences with the local point. Additionally, we speculate that global and local points do not significantly impact energy cost during local

Table 4.6: Model performances (%) with various stage-decision ratios.

Model	Method	Pet	Bird	Plant	Avg.
MCUNet	MicroT_0.25	84.34	56.05	53.87	64.75
	MicroT_0.75	76.68	47.87	51.65	58.73
ProxylessNAS	MicroT_0.25	82.35	51.97	54.04	62.79
	MicroT_0.75	76.44	46.59	50.36	57.80

Table 4.7: The energy costs (kJ) of the MCU local inference with various stage-decision ratios.

Model	Method	STM32H7A3ZI	STM32L4R5ZI	Avg.
MCUNet	TTL	9.52	12.42	10.97
	MicroT_0.25	8.67	11.28	9.98
	MicroT_0.75	7.75	9.61	8.68
Proxy	TTL	4.64	6.19	5.42
	MicroT_0.25	4.23	5.65	4.94
	MicroT_0.75	3.71	4.87	4.29

training and inference (*e.g.*, stage decision ratio of 0.5) because: (1) Experiments show that global and local points are not significantly different in terms of position; (2) The part and full models are inherently integrated, so a energy reduction/increase in the part model corresponds to an increase/reduction in the full model, offsetting each other.

Model Performance with Various Stage-Decision Ratio

In Section 4.5, we have introduced the model performance of MicroT under a standard stage-decision ratio of 0.5. To show the performance trends associated with varying stage-decision ratios more thoroughly, we extend our analysis by examining the model performance at ratios of 0.25 and 0.75, with the findings presented in Table 4.6. We can find that, as the proportion of samples processed by the full model increases (*i.e.*, as the stage-decision ratio decreases), the prediction accuracy of the model correspondingly rises. This trend suggests that greater involvement of the full model in the inference tasks significantly enhances MicroT's performance, particularly in scenarios demanding higher accuracy.

Energy Saving with Various Stage-Decision Ratio

Following our analysis of MicroT's performance trends, we proceed to evaluate the energy costs of local inference under varying stage-decision ratios, specifically encompassing ratios of 0.25 and 0.75. It should be noted that stage-decision ratios of 0 or 1 imply that inference is conducted entirely by the full model or the part model, respectively; these extreme scenarios, lacking the

Table 4.8: The impact of quantization on accuracy (%).

Model	Quant.	Pet	Bird	Bird	Avg.
MCUNet	W/O	81.64	53.13	53.19	62.65
	W/	81.21	52.85	52.92	62.33
ProxylessNAS	W/O	80.82	52.21	54.26	62.43
	W	80.33	51.51	53.45	61.76

benefits of stage-wise collaboration, are excluded from this evaluation. We maintain TTL as the baseline method for comparison with MicroT, and the results are presented in Table 4.7. The experimental results reveal that MicroT achieves diverse levels of energy savings across different stage-decision ratios. Specifically, when compared with TTL's full model inference, MicroT at a ratio of 0.25 reduces energy consumption by an average of 8.94%, while at a ratio of 0.75, it realizes a more substantial average reduction of 20.81%.

The Impact of Quantization

MicroT implements a partial quantization strategy, wherein the feature extractor is quantized to the `int8` format, while the classifier, locally generated on the MCUs, retains the `float32` format. This approach capitalizes on the strengths of both components: the feature extractor, constituting a substantial portion of the model's parameters, benefits from quantization by reducing memory usage, accelerating inference execution, and diminishing energy consumption. Concurrently, the classifier, trained locally with floating-point precision, can offset the accuracy degradation induced by the quantized feature extractor through fine-tuning. This hybrid quantization technique establishes a balanced solution tailored to the resource-constrained environment of MCUs. To assess its efficacy, we conducted experiments comparing performance with (W/) and without (W/O) partial quantization, with the results shown in Table 4.8. The results reveal an average accuracy difference of approximately 0.5% between the two configurations. This modest difference is acceptable within the context of the trade-off between prediction accuracy and system costs.

The Impact of Local Drifts

Although MicroT's research focuses on feature space drift (*e.g.*, the feature distributions of Pet, Plant, and Bird datasets differ from cloud-based ImageNet), to test MicroT's robustness, we introduced additional data drift on local datasets and conducted ablation experiments.

Table 4.9: Ablation experiments on the local data drifts (Model: MCUNet, Dataset: Pet). 'Source Domain Acc' means the accuracy on the original Pet dataset using the local MCUNet; 'Target Domain Acc' means the accuracy on the Pet dataset with added noise; 'Fine-Tuned Acc' means the accuracy on the noisy Pet dataset after fine-tuning the classifier.

	Source Domain Acc	**Target Domain Acc**	**Fine-Tuned Acc**
Gaussian Noise	81.21%	76.94%	80.15%
Motion Blur	81.21%	72.42%	78.74%
Partial Occlusion	81.21%	64.18%	72.56%

Specifically, using MCUNet and the Pet dataset as examples, we selected three typical types of image data noise—Gaussian noise, motion blur, and partial occlusion—and tested the accuracy on the source MCUNet-Pet model after adding noise, as well as the accuracy recovery after fine-tuning (only the classifier).

As shown in the Table. 4.8, for undisturbed source domain data, MicroT achieves an accuracy of 81.21%. When typical noise perturbations are introduced, the target domain accuracy decreases to varying degrees. Specifically, for Gaussian noise, the accuracy drops from 81.21% to 76.94%, the smallest decrease, and nearly fully recovers to 80.15% after fine-tuning. Gaussian noise is a pixel-level local perturbation, and MicroT employs DINOv2 techniques during the SSL phase, which includes multi-view augmentation (such as random noise and color jitter). The subsequent SSKD process enables the student model to learn these representations, making it robust to such perturbations. For motion blur, the accuracy drops to 72.42%, a larger decrease than with Gaussian noise, but recovers to 78.74% after fine-tuning. Blur is a structural distortion that damages edges and local details. Although MicroT's SSL techniques mitigate low-frequency damage, information degradation persists during inference. Fine-tuning the classifier can partially compensate for the accuracy loss due to information degradation. For partial occlusion, the accuracy drops to 64.18%, the most decrease, and recovers to 72.56% after fine-tuning. Occlusion results in direct information loss, forcing the feature extractor to rely on contextual inference in affected areas, with limited information recovery capabilities. Fine-tuning the classifier alone struggles to fully compensate for the semantic loss, resulting in the smallest recovery margin.

Notably, while all three typical perturbations lead to a decrease in target domain accuracy, comparative results show that MicroT remains somewhat robust under pixel noise (Gaussian

noise) and slight structural distortion (motion blur). This is because MicroT, through SSKD, has learned semantic information from the teacher model (DinoV2), which provides some resistance to such drifts. For semantic loss caused by partial occlusion, despite SSKD learning the teacher model's semantic information, MCUNet itself lacks a block-level self-attention mechanism, and the information loss due to occlusion is irreversible, thus rely solely on fine-tuning the classifier has limited effectiveness.

4.7 Discussions and Limitations

In this section, we discuss some of the analyses and limitations of this study.

• **Limited Experimental Models, Datasets, and Scenarios.** This chapter uses image data as an example and focuses on classification tasks to validate MicroT. Other interesting application scenarios in edge computing remain worthy of exploration. For example, battery-free underwater devices powered by acoustic energy utilize low-power cameras to detect and identify marine species [125]; solar-powered low-power sensor networks enable precise monitoring of wildlife populations [216]; environmental monitoring systems driven by wind or water energy require long-term self-sustainability [217]; and implantable medical devices fueled by bioenergy support health monitoring [218]. Future work could explore applying the core concepts of MicroT (*e.g.*, SSKD and stage-decision) to other fascinating domains, as well as extending its application to other models (*e.g.*, SqueezeNet [167], MobileNet [219]) and other data modalities (*e.g.*, IMU and audio).

• **Cloud Dependency of MicroT.** MicroT offloads the training of the feature extractor to a cloud environment, and ideally, it only requires a one-time interaction between the cloud and the local device for model deployment. This paradigm significantly reduces the need for frequent cloud-device communication and aligns well with most typical deployment scenarios. However, this design inevitably introduces a limited dependency on cloud infrastructure. In ultra-constrained or security-critical scenarios—such as air-gapped systems or privacy-sensitive edge deployments—any reliance on cloud infrastructure, even for a one-time model transfer, may be unacceptable. Addressing this limitation remains a key direction for future work. One

potential solution is to migrate the SSKD and joint training processes to more capable edge devices [220, 221] (*e.g.*, NVIDIA Jetson), which would require optimizing SSKD to reduce its computational demands. Alternatively, decentralized learning strategies—such as peer-to-peer model transfer or collaborative training across a network of local devices—could further minimize or eliminate the need for cloud-based training while preserving the benefits of on-device personalization [222, 223].

• **Stability of Median-Quartile-Based Threshold.** Although our experiments demonstrate that the median-quartile-based threshold estimation method works well even with a small number of local samples, this phenomenon is largely attributed to the high consistency in the distribution of confidence scores generated by the part model. This consistency reflects both the generalization capability of the feature extractor and the intrinsic structural characteristics of the input data. When the feature space exhibits an approximately normal distribution or a symmetric unimodal structure, the median, as a robust statistic, effectively captures the central tendency of the confidence score distribution, even with limited samples. Existing studies suggest that such distribution patterns typically indicate that the feature extractor has learned generalizable and well-clustered representations [213, 214], which is one of the core objectives of MicroT's adoption of SSKD and joint training strategies for cloud-based feature extractor optimization. However, we also recognize that these assumptions may not hold in certain scenarios. When dealing with highly heterogeneous input data (*e.g.*, cross-domain samples, non-standard sensor modalities, or edge data heavily affected by noise), the confidence score distribution may exhibit significant skewness or multimodal characteristics. In such cases, relying solely on median estimation with small-scale samples may lead to inaccurate threshold settings, thereby impacting the system's decision-making efficiency and accuracy. To mitigate these issues, potential solutions include increasing the size of local samples used for threshold estimation, employing lightweight distribution fitting methods (*e.g.*, kernel density estimation [224] or histogram smoothing [225]), or introducing unsupervised clustering techniques to identify structural subgroups within the data. Exploring these methods will be a key direction for enhancing MicroT's adaptability in non-ideal data environments in the future.

• **Stability of Model Segmentation.** One of the principal objectives in designing the fea-

ture extractor is to enhance its generalization capability. Consequently, we believe that the globally optimal segmentation point, determined using large public datasets such as ImageNet, may also serve as the optimal—or near-optimal—segmentation point for local datasets. However, in practical settings, the distribution of local datasets may diverge from those of global data, potentially resulting in a discrepancy between the local and global optimal segmentation points. To address this possibility, we adopt a conservative strategy by prioritizing the global optimal segmentation point. This decision is grounded in its greater stability and broader applicability, offering a relative stable segmentation point. Our experiments indicate that, even when the optimal segmentation point on a local dataset exhibits slight variations, the overall performance of MicroT remains within an acceptable range, sufficiently meeting the demands of most application scenarios (Section. 4.6). This prudent approach is particularly advantageous in resource-constrained MCU environments, as it avoids the additional computational complexity and energy overhead that would arise from frequently adjusting the segmentation point to suit local conditions.

• **The Paradigm of 'Feature Extractor + Classifier'.** The design paradigm employed by MicroT exhibits pronounced advantages in energy efficiency. However, to further minimize resource utilization, we confine local training exclusively to the classifier, leaving the feature extractor static. While this strategy alleviates the computational burden on the MCU, it imposes stringent demands on the generalization capability of the feature extractor. This generalization assurance stems from two pivotal factors: firstly, the rapid proliferation and advancement of smart devices have yielded an abundance of high-quality, publicly accessible training datasets, ensuring ample resources for pre-training the feature extractor; secondly, the integration of advanced data augmentation techniques such as SSL enhances the feature extractor's adaptability to unseen data.

• **Re-Training Issues.** Under ideal conditions, once the classifier is locally trained and satisfies the performance requirements for a given task, it could theoretically stop further optimization and remain functional over an extended period. In practice, however, local data is often subject to a degree of drift over time—such as shifts in lighting conditions induced by seasonal changes or other environmental variations. While the generalized feature extractor in MicroT,

having been optimized for robustness, is capable of accommodating minor local data drifts and maintaining performance stability, there exist extreme scenarios where substantial deviations in data distribution may necessitate additional fine-tuning of the classifier. We believe that such subsequent fine-tuning can typically be executed efficiently and within a short time, as changes in local data distributions tend to be progressive rather than abrupt, requiring only modest adjustments to the classifier's existing parameters to adapt to the evolving data characteristics.

Concerning the retraining of the feature extractor, under most conditions, variations in local data do not need a entire re-training for the feature extractor, which stems from our optimization of the feature extractor's generalization capacity. Nevertheless, in exceptional cases—such as when significant and persistent local data drift pushes the distribution beyond anticipated bounds—the generalization capability of the feature extractor may reach its limits, rendering it inadequate for maintaining optimal task performance. To address these cases, we plan to utilize partial retraining rather than a full retraining. Specifically, given that the existing feature extractor retains a degree of effectiveness, we can selectively unfreeze the final few layers and fine-tuning their parameters using new local data.

• **Robustness of MicroT.** Results show MicroT demonstrates good overall robustness when facing typical perturbations such as sensor noise, optical blur, and partial occlusion. However, due to the simplicity of the lightweight model's structure—lacking explicit semantic modeling mechanisms like block-level self-attention—its ability to compensate for local semantic loss (*e.g.*, occlusion) relies solely on feature transfer during the distillation phase. Consequently, its capacity to recover performance in the face of irreversible information loss remains limited. This indicates that relying solely on classifier fine-tuning or representations acquired through distillation encounters bottlenecks when addressing severe structural defects. For future work, two directions can be considered: first, while maintaining model lightweightness, design pluggable contextual inference modules or local attention mechanisms to enhance the recovery of locally missing information; second, explore partially unfreezing the feature extractor or adopting adaptive parameter adjustment methods to further improve robustness and generalization in complex local perturbation scenarios.

• **Data Acquisition and Labeling.** Currently, we assume that locally acquired data are available from sensors such as cameras commonly embedded in MCU platforms. Moreover, we assume that labeled data exist locally. These assumptions simplify the scope of our work, allowing us to focus on efficient model personalization on MCUs. However, we acknowledge that these assumptions also introduce limitations in terms of full automation and scalability of on-device personalization. Specifically, the acquisition of labeled data often requires manual annotation or carefully designed supervised data collection, which may not always be feasible in real-world deployments. Additionally, our current system does not explicitly address local drift detection, or automatic retraining triggers to handle evolving data distributions over time. Future work should explore integrating lightweight on-device data labeling techniques, unsupervised or self-supervised learning methods to reduce dependence on labeled data, and mechanisms for drift detection and adaptive retraining to maintain model robustness in dynamic environments.

4.8 Conclusions

Existing methods for personalizing ML models on MCUs typically depend on cloud-based communication or involve resource-intensive pre-training and training procedures directly on the MCUs. These approaches demand substantial energy, which contradicts the energy-efficient nature of MCUs. To tackle this challenge, we introduce MicroT, a lightweight framework that leverages self-supervised knowledge distillation combined with an early-exit strategy tailored for MCUs. MicroT achieves superior accuracy, surpassing TTL and two SOTA methods by 7.01 - 11.60%, 2.29 - 5.73%, and 2.12 - 3.68%, respectively. Furthermore, MicroT demonstrates energy efficiency, reducing energy usage during the MCU training phase by 2.03 - 2.28× compared to SOTA techniques. In the inference phase, it reduces energy consumption by 14.17% compared to TTL.

Chapter 5

System-Related Adaptations

This chapter focuses on optimizing energy/power in edge computing from a system configuration perspective. In previous chapters, we have used Raspberry Pi and ultra-resource-constrained MCUs as experimental platforms. However, with the rapid development of AI accelerators, such as the Analog MAX78000, ARM Ethos-U65, and GreenWaves GAP9 [23, 226, 227], these accelerators are increasingly applied in local model inference and computation. Therefore, this chapter takes MCUs equipped with AI accelerators as the research subject. While compact AI accelerators provide hardware-level support for on-device inference, their energy characterization under varying system configurations remain underexplored. We conducted the study using the MAX78000 as an example, selecting models and datasets from its official repository to eliminate potential confounding factors and avoid interference from improper operations. To keep the generality of the research, we included both image data and time-series data. We first analyze the relationship between accelerator frequency and the latency of model inference-related operations. We then conduct comprehensive energy/power benchmarking at micro- and macro-levels: micro-level evaluation quantifies the power and energy profiles of individual operations (sensing, model inference, computation, memory I/O, idle operations) under diverse system configurations; macro-level evaluation investigates system-wide configurations on overall energy consumption of end-to-end application pipelines. Our experiments reveal that dynamic system configuration in single application operation can enhance efficiency, reducing total energy consumption by up to 38%. This study provides practical guidelines for system-level optimization in resource-constrained, energy-sensitive edge AI applications.

5.1 Introduction

Compact MCU with AI accelerators (AI-MCU), such as the Analog MAX78000 [23], ARM Ethos-U65 [226], and GreenWaves GAP9 [227], have redefined the physical scope of AI by shrinking its hardware demands, bringing its potential into daily life. Tailored for MCUs, these devices feature a highly streamlined design—such as the MAX78000 with dimensions of just 8mm×8mm—allowing effortless incorporation into wearable technologies like fitness bands or health monitors. This downsizing facilitates the on-device execution of AI tasks [228], reducing response times and bolstering privacy safeguards through localized data handling.

The integration of micro-scale AI accelerators into wearable technology platforms demonstrates significant potential for advancing human-centered physiological monitoring applications. These devices integrate a diverse array of on-body sensors—including accelerometers, heart rate monitors, and ambient light detectors—to capture real-time physiological signals and environmental data, enabling intelligent wearable to deliver tailored contextual services without relying on smartphones. For example, an attention alert system could employ smart glasses to assess a user's visual focus and trigger haptic feedback through a worn ring, aiding in timely responses to distractions or hazards [229]. However, this enhanced computational capacity introduces substantial hurdles, particularly in terms of *energy efficiency*, a critical limitation given the constrained battery life typical of wearable devices.

Prior research has laid groundwork by examining the energy profiles of mobile and embedded accelerators [230, 231], offering initial insights into optimization strategies. Nonetheless, while some benchmark studies have explored micro AI accelerators integrated into MCUs [55], these efforts primarily focus on evaluating energy usage across various AI models, neglecting a systematic analysis of their energy behavior under diverse operational conditions. This gap in understanding hinders a comprehensive assessment of accelerator performance in dynamic environments.

Contributions. We conducted an in-depth energy/power characterization of MCUs integrated with tiny AI accelerators, with the goal of supporting the development of efficient, human-centered sensing systems and edge computing.

(1) We conducted a systematic power/energy characterization on an ultra-low-power AI accelerators, MAX78000. To facilitate the analysis, we considered an end-to-end pipeline based on real-world applications, consisting of sensing, inference, computation, memory operation, and idle operations.

(2) We explored the relationship between system frequency, CNN accelerator frequency, and model inference latency.

(3) We performed a micro-level analysis by independently evaluating each operation of the pipeline. For each operation, we partially or fully considered core configuration control, frequency management, and quadrant-based control. These three categories were further divided into smaller subcategories, resulting in a total of eight system configuration options. We considered two cases/scenarios for the experiments of each application operation.

(4) We conducted a macro-level analysis, treating the pipeline as a whole and changing system configurations one by one to assess their impact on overall energy consumption. We considered two application scenarios: speech recognition and image segmentation.

(4) We found that dynamic system configurations are more energy-efficient. In other words, the most energy-efficient configuration varies for each operation of the application pipeline. Based on this observation, we introduced the 'Best' dynamic configurations with the lowest working energy in the macro-level analysis. Compared to the default static system configuration of the MAX78000, the 'Best' configuration reduces total energy (energy consumption including the idle phase) by 3.19×, and working energy by $1.38X$ on average.

(5) We explored different idle implementation methods to optimize the MAX78000's default busy-waiting. We ultimately found that placing the system in ultra-low-power mode (UPM) resulted in the lowest power consumption with just $2.02mW$, while the default idle implementation is $13.11mW$.

This chapter is related to the following paper:

Yushan Huang, Taesik Gong, SiYoung Jang, Fahim Kawsar, Chulhong Min. 'Energy Characterization of Tiny AI Accelerator-Equipped Microcontrollers'. Accepted to The Second In-

ternational Workshop on Human-centered Sensing, Networking, and Multi-Device Systems HumanSys'24, Co-located with SenSys'24 and BuildSys'24. **Best Paper Award**.

5.2 Background

In this section, we first introduce related works, then introduce MAX78000, application pipeline, and related system configurations.

5.2.1 Related Works

Currently, some studies are exploring energy optimization in edge computing from the perspective of system configuratio, for instance, certain studies reduce energy consumption and power by dynamically adjusting system frequency [49, 232]. Rotteleuthner et al. modularized clock configuration to lower energy consumption through dynamic voltage and frequency scaling [50], while Qian et al. focused on optimizing configurations during the idle-waiting phase, thereby reducing overall power consumption [51]. These studies collectively demonstrate the effectiveness of system configuration in reducing energy consumption in edge devices.

In the field of energy benchmarking, existing research primarily focuses on mobile devices and traditional MCUs. Laskaridis et al. quantified the energy characteristics of large language models on mobile devices [52], Sah et al. established energy benchmarks for MCU-based object detection models [53], and Pesovic et al. compared the energy efficiency metrics of ARM Cortex-M series processors with WSN processors [233]. For MCUs equipped with AI accelerators, some efforts have begun to establish benchmarks. For example, Jang et al. used the MAX78000 to develop thermodynamic characteristics [54], while Moss et al. characterized the energy consumption of different ML model inferences on the MAX78000, including latency, power, and memory [55]. Dabbous et al. further compared the energy efficiency of the MAX78000 with STM32 and Raspberry Pi during ML model inference [56]. However, existing AI-MCU benchmark studies have clear limitations: first, they focus on single-dimensional testing, concentrating solely on the model inference phase while neglecting other pipeline components such as sensing and memory I/O; second, their perspective only emphasize energy

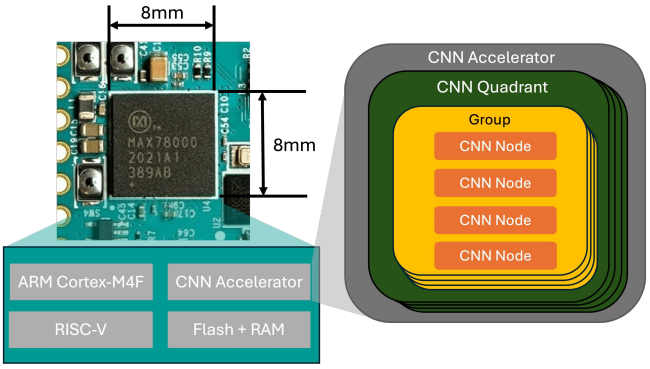


Figure 5.1: Overview of MAX78000.

differences across ML models while lacking analysis of the energy efficiency impact of system configuration parameters.

5.2.2 MAX78000

In recent years, the landscape of tiny AI accelerators has expanded rapidly, with the emergence of devices such as the Analog MAX78000 [23], ARM Ethos-U65 [226], and GreenWaves GAP9 [227]. Nevertheless, the majority of these accelerators are either not yet commercially available or offer limited control over their underlying operations, posing challenges for researchers aiming to analyze energy consumption in resource-constrained environments. To address this, we opted for the Analog MAX78000 as our experimental platform, a decision driven by its extensive suite of open-source tools and comprehensive documentation. These resources facilitate a deep understanding of the hardware architecture and provide a robust foundation for systematically investigating the energy consumption characteristics.

The MAX78000 features two processors—the ARM Cortex-M4F and RISC-V, both are capable of operating as the primary processor, supported by 512 KB of flash memory and 128 KB of SRAM to ensure robust storage and execution capabilities. The CNN accelerator is architecturally distinctive, comprising 4 nodes per group and 4 groups per quadrant, totaling 64 nodes, with each node dedicated to performing convolution operations and utilizing shared memory to manage input data and activation values efficiently. Additionally, the accelerator is equipped with dedicated memory resources: 512 KB for input data, 442 KB for weights, and 2 KB for biases, enhancing data access and computational efficiency. Fig. 5.1 offers a detailed overview

Table 5.1: MAX78000 operation modes.

Processor Mode	ARM	RISC-V	Oscillators	Memory	CNN Quadrants	CNN RAM	Peripherals
ACTIVE	On	On	All Available	Available	Active, Configurable	Active, Configurable	Available
SLEEP	Retention	On/Retention	All Available	Available	Active, Configurable	Active, Configurable	Available
LPM	Retention	On/Retention	ISO, IBRO ERTCO, INRO	0,1: Retention 2,3: Available	Active, Configurable	Active, Configurable	Available
UPM	Retention	Retention	IBRO, ERTCO INRO	Retention	Optionally off	Selectable Retention	Retention

Table 5.2: System configuration variables involved in the experiment.

	Core Configuration Control		Frequency Management			CNN Accelerator Quadrant Control		
	Processor	Operation Mode	System Clock	System Divider	CNN Divider	Number of Activated Nodes	CNN Node Contiguity	CNN Boost
Sensing	✓	✓	✓	✓				
Inference	✓	✓	✓	✓	✓	✓	✓	✓
Computation	✓	✓	✓	✓				
Memory I/O	✓	✓	✓	✓	✓	✓	✓	✓
Idle	✓		✓	✓				

of the MAX78000 architecture.

5.2.3 Configuration Description

The MAX78000 incorporates a range of adjustable system parameters that significantly influence its energy performance during application execution, offering flexibility for optimizing resource utilization. We place particular emphasis on three primary control categories: core configuration control, frequency control, and CNN accelerator quadrant control. Each category is subdivided into several subcategories, collectively yielding a total of eight system configuration variables, as detailed in Table 5.2. Additionally, the table highlights the availability of these control options during various stages of the application pipeline—such as sensing, inference, and computation.

Processor. The MAX78000 integrates both an ARM Cortex-M4F and a RISC-V processor, each capable of serving as the primary processing unit and adaptable to varying task requirements. However, the architectural distinctions between these processors—such as differences in instruction sets, clock management, or caching mechanisms—may lead to variations in power and energy consumption under identical workloads. To gain deeper insights into these differences and their impact on system performance, we selected a specific processor type for each operation of the application pipeline—such as sensing, inference, and computation—allowing us to compare the power and energy characteristics of the ARM Cortex-M4F and RISC-V.

Operation Mode. The MAX78000 integrates multiple operational modes (as shown in Table 5.1). For instance, in ACTIVE mode, all system components, including the processor, memory, and accelerator, are fully activated to ensure maximum computational capacity, while SLEEP mode transitions the ARM or RISC-V processor into a retention state, suspending non-essential operations to substantially lower power usage during brief idle periods. Modes such as Low-Power Mode (LPM) and Ultra-Low-Power Mode (UPM) introduce complexities by influencing other configurations, including memory access strategies and clock frequencies, potentially complicating energy consumption analysis. Consequently, we concentrate solely on ACTIVE and SLEEP modes (excluding the ‘idle’ operation).

System Clock. The operating frequency (SYS_Freq) of a processor serves as a pivotal factor influencing the energy efficiency of AI-MCU, directly shaping the system's computational speed and power usage. This frequency is governed by two essential parameters: the system clock (SYS_CLK) and the clock divider (SYS_DIV), with their relationship expressed as:

$$SYS_Freq = \frac{\text{SYS_CLK}}{2^{\text{SYS_DIV}}} \tag{5.1}$$

Here, the system clock acts as a central timing signal, synchronizing the processor, CNN accelerator, peripherals, and other system components to ensure coordinated execution at precise speeds and timings. Although the MAX78000 supports a variety of clock sources for the system clock, we opted to utilize the most prevalent options—the Internal Primary Oscillator (IPO, 100 MHz) and the Internal Secondary Oscillator (ISO, 60 MHz)—across all experimental operations, with specific details documented in Table 5.2.

System Divider. The clock divider regulates system speed by applying a divider value to the base clock frequency, enabling precise control over components like processors and accelerators. The MAX78000's system divider governs the global clock frequency, while a dedicated CNN divider fine-tunes the accelerator's frequency for varied computational needs. For these experiments, we selected divider values of 1, 2, and 4 from a total of eight available options, since the significantly increased latency associated with higher values. This choice aligns with the demands of real-world applications.

Number of Activated Nodes. The MAX78000 allows developers to flexibly set active CNN accelerator nodes. By adjusting register parameters, developers can tailor the number of activated nodes to match the task complexity or resource constraints. In our experiments, we implemented two distinct configurations for comparison: one where the accelerator operates with the minimum required number of CNN nodes to assess energy efficiency under resource-conserving conditions, and another where all 64 nodes are activated to evaluate energy characteristics under maximum computational capacity.

CNN Node Contiguity. The distinctive architectural framework of the CNN accelerator.

Given that data is collectively utilized within each group, a scenario where node activation becomes discontinuous—some nodes being active (represented as **f**) and others inactive (represented as **0**)—may trigger supplementary data exchanges across memory domains. Such additional operations could elevate the resource demands of the system, thereby exerting a potentially effect on energy efficiency. To thoroughly assess this dynamic, our experimental approach incorporated an analysis of two representative activation patterns: a continuous mode, exemplified by **ff00**, where activation remains uniform across certain nodes, and a discontinuous mode, illustrated by **f0f0**, where active and inactive states alternate. By examining these contrasting patterns, our study aims to elucidate the precise impact of discontinuous node activation on memory operations and energy utilization.

CNN Boost. The MAX78000 incorporates an external CNN boost circuit engineered to augment the power supply for the CNN accelerator. Under typical conditions, the internal Single-Input Multiple-Output (SIMO) power system adequately supports moderate computational demands. However, during scenarios requiring maximum processing capacity, such as transient over-current events, the SIMO supply may encounter voltage instability, manifesting as drops or brown-outs, which could disrupt accelerator performance. To mitigate this, the CNN boost circuit serves as an auxiliary power source, reinforcing the SIMO system to maintain consistent functionality and avert power-induced disruptions in the accelerator. To investigate the practical impact of this feature, we conducted a series of experiments measuring power and energy consumption under two conditions: with the CNN boost activated and without it, aiming to assess its contribution to effective power management.

5.3 Implementation & Evaluation Setup

We first introduce related operations, then introduce our experimental models and datasets.

5.3.1 Application Pipeline

The application pipeline within wearable devices encompasses a series of critical operations. This pipeline typically operates in a continuous manner, punctuated by periodic intervals to enable the delivery of situational services, such as health monitoring or environmental adapta-

tions. We selected a set of common operations:

Sensing denotes the process of gathering data from the physical environment via sensors, serving as the foundational step for the intelligent functionalities of wearable devices. This operation may include, for example, utilizing a camera to capture real-time images for analyzing the user's visual surroundings or employing a microphone to record audio data for detecting voice commands or ambient noise patterns.

Inference represents a critical operation that leverages the CNN accelerator to run a convolutional neural network model, addressing complex tasks such as image classification or object detection. This process involves feeding sensor-collected data into the model to extract high-level features and produce predictive outcomes, serving as the cornerstone for intelligent decision-making in wearable devices. For instance, inference can enable smart glasses to identify objects within a user's field of view or analyze audio data to detect specific environmental events, thereby facilitating real-time contextual services.

Computation (COMP) encompasses a range of operations executed exclusively by the processor, including preprocessing raw sensor data—such as filtering or format conversion—and performing intricate mathematical computations like the Softmax function to derive classification probabilities.

Memory I/O (MIO) refers to the data exchange operations between the processor and the CNN accelerator. This operation include retrieving model parameters from SRAM and transferring them to the CNN accelerator to facilitate inference tasks, as well as returning the inference outcomes from the accelerator to SRAM for subsequent processing or storage.

Idle denotes the waiting periods during which the AI-MCU is not actively engaged in task execution, a state particularly prevalent in resource-constrained devices. Typically, during these idle phases, the MCU transitions into a low-power mode to minimize unnecessary energy consumption while remaining responsive to the next operation or external input. For instance, in wearable devices, the idle state may occur between sensor data collection intervals or following the completion of an inference task.

5.3.2 Experimental Models and Datasets

In the micro-level evaluation, 'Inference' operation involves model selection. To avoid interference from unknown factors or improper operations, we selected two publicly available models provided in the MAX78000 official examples: a simple 4-layer CNN model and a more complex 18-layer UNet model. These two architectures differ significantly in depth and complexity, enabling a more comprehensive assessment of the MAX78000's energy consumption performance under different model conditions. For dataset selection, we strictly adhered to the official datasets used by selected models to ensure the reproducibility and representativeness of the evaluation results, establishing an accurate energy consumption baseline for the MAX78000.

Macro-level evaluation also involve model selection, considering this chapter's focus on optimizing system configuration, we covered two typical modalities—image data and time-series data—to keep the generalizability of the benchmark. Specifically, we continued to use application scenarios, models, and corresponding datasets provided by the MAX78000 official resources, including the KWS20 model for speech recognition and the 18-layer UNet model for image segmentation tasks.

All the aforementioned models and datasets are available at the MAX78000 resources [1].

5.4 Experiments on Latency

The MAX78000 is equipped with a dedicated CNN accelerator, the CNN frequency CNN_Freq is not only controlled by the system frequency SYS_Freq (determined by system clock and system divider), but can also be adjusted via an independent CNN divider. Specifically, the CNN frequency is a division of the system frequency by 2^n, where n is the division coefficient, ranging from 0 to 4. This relationship can be expressed as:

$$CNN_Freq = \frac{SYS_Freq}{2^n}, \quad n = 0, 1, 2, 3, 4. \tag{5.2}$$

[1] Models and datasets are available at:
https://github.com/analogdevicesinc/msdk/tree/main/Examples/MAX78000

For example, when the system frequency is 100 MHz, the supported CNN frequencies are 100 MHz ($n = 0$), 50 MHz ($n = 1$), 25 MHz ($n = 2$), 12.5 MHz ($n = 3$), and 6.25 MHz ($n = 4$).

Conversely, given a CNN frequency F_{cnn}, the supported system frequency F_{sys} can be derived through a multiplication relationship. Specifically, the system frequency is 2^n times the CNN frequency, where m is the multiplication coefficient, ranging from 0 to 4, and the system frequency does not exceed the maximum system frequency (*e.g.*, 100 MHz). This relationship can be expressed as:

$$SYS_Freq = CNN_Freq \times 2^n, \quad n = 0, 1, 2, 3, 4. \tag{5.3}$$

For example, when the CNN frequency is 6.25 MHz, the supported system frequencies are 6.25 MHz ($n = 0$), 12.5 MHz ($n = 1$), 25 MHz ($n = 2$), 50 MHz ($n = 3$), and 100 MHz ($n = 4$).

To gain a deeper understanding of the latency characteristics of the CNN accelerator, we focused on studying key operations related to the CNN accelerator, including CNN initialization, weight/bias loading, CNN configuration, input loading, model inference, and post-processing (*e.g.*, softmax). By analyzing the relationship between the frequency and latency of these operations, we aim to reveal how the system frequency and CNN frequency affect their latency characteristics.

5.4.1 Relationship Between Latency and CNN Frequency

To investigate the performance characteristics of the MAX78000 platform, we conducted experiments by fixing the system frequency (SYS_Freq) and varying the CNN accelerator frequency (CNN_Freq) under two system clock configurations: IPO and ISO. The experimental results are presented in Figure 5.2, which illustrates the trends of latency (Time) as a function of CNN Frequency.

The experimental results demonstrate a significant consistency in latency trends across the IPO and ISO system clock configurations. In both setups, latency decreases notably with increasing CNN Freq, particularly within the 0-40 MHz range, where it drops from approximately 10^3 us

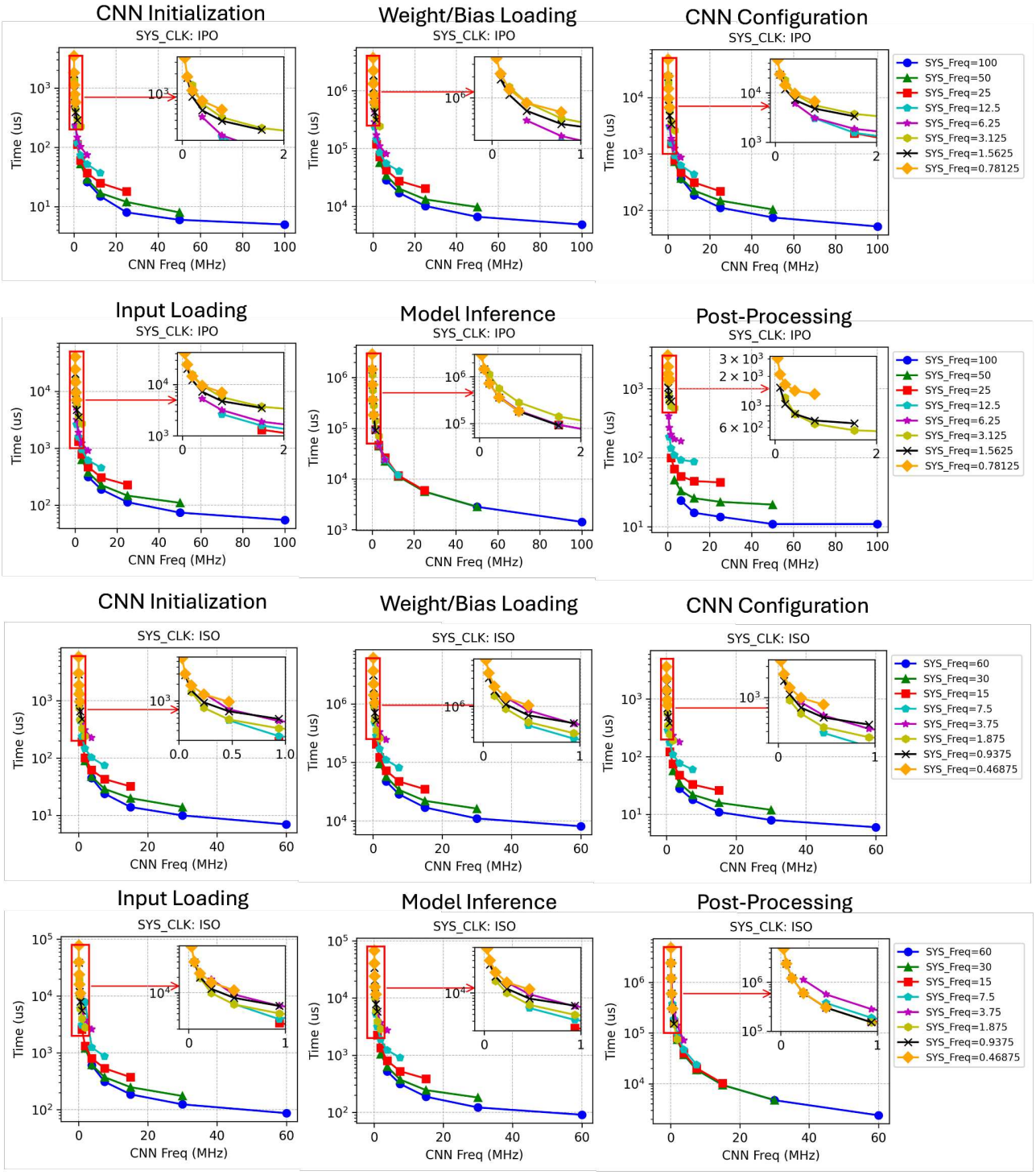


Figure 5.2: Relationship between latency and CNN frequency.

to 10^1 us, a reduction of nearly two orders of magnitude. This consistency suggests that the MAX78000 platform's performance exhibits low dependence on the clock configuration.

Furthermore, we observed a heightened sensitivity of latency to changes in the low CNN frequency range. Across all SYS_Freq settings and both clock configurations, latency exhibits

a steeper decline within this interval, decreasing from 10^3 us to 10^1 us, indicating that low-frequency optimization is critical for performance enhancement.

The experimental results further indicate that, under fixed system frequency, CNN Freq and latency exhibit an inverse relationship, which can be approximated as:

$$Time \propto \frac{1}{CNN_Freq} \tag{5.4}$$

where k is a constant reflecting the speed of latency reduction with increasing CNN Frequency. This exponential relationship is supported by the decline observed on the logarithmic Y-axis, most prominently in the 0-40 MHz low-frequency range, where latency decreases rapidly, consistent with the observed high sensitivity in this regime.

5.4.2 Relationship Between Latency and System Frequency

Similarly, under the two system clock configurations (IPO and ISO), we fixed the CNN accelerator frequency and varied the system frequency to explore the relationship between latency and system frequency, as illustrated in the Fig. 5.3.

The experimental results indicate that for most operations (except for model inference), latency and system frequency also exhibit an inverse relationship, with latency showing heightened sensitivity to changes in the low system frequency range. However, it is noteworthy that for the model inference operation, there is no significant relationship between latency and system frequency. This suggests that when the CNN accelerator frequency is fixed, the system frequency does not affect the latency of model inference. This further demonstrates that on the MAX78000, model inference is solely related to the CNN accelerator and runs exclusively on the CNN accelerator.

5.5 Experiments on Energy

We executed experiments at both micro and macro levels to comprehensively evaluate system performance. The micro-level experiments were designed to meticulously explore the energy

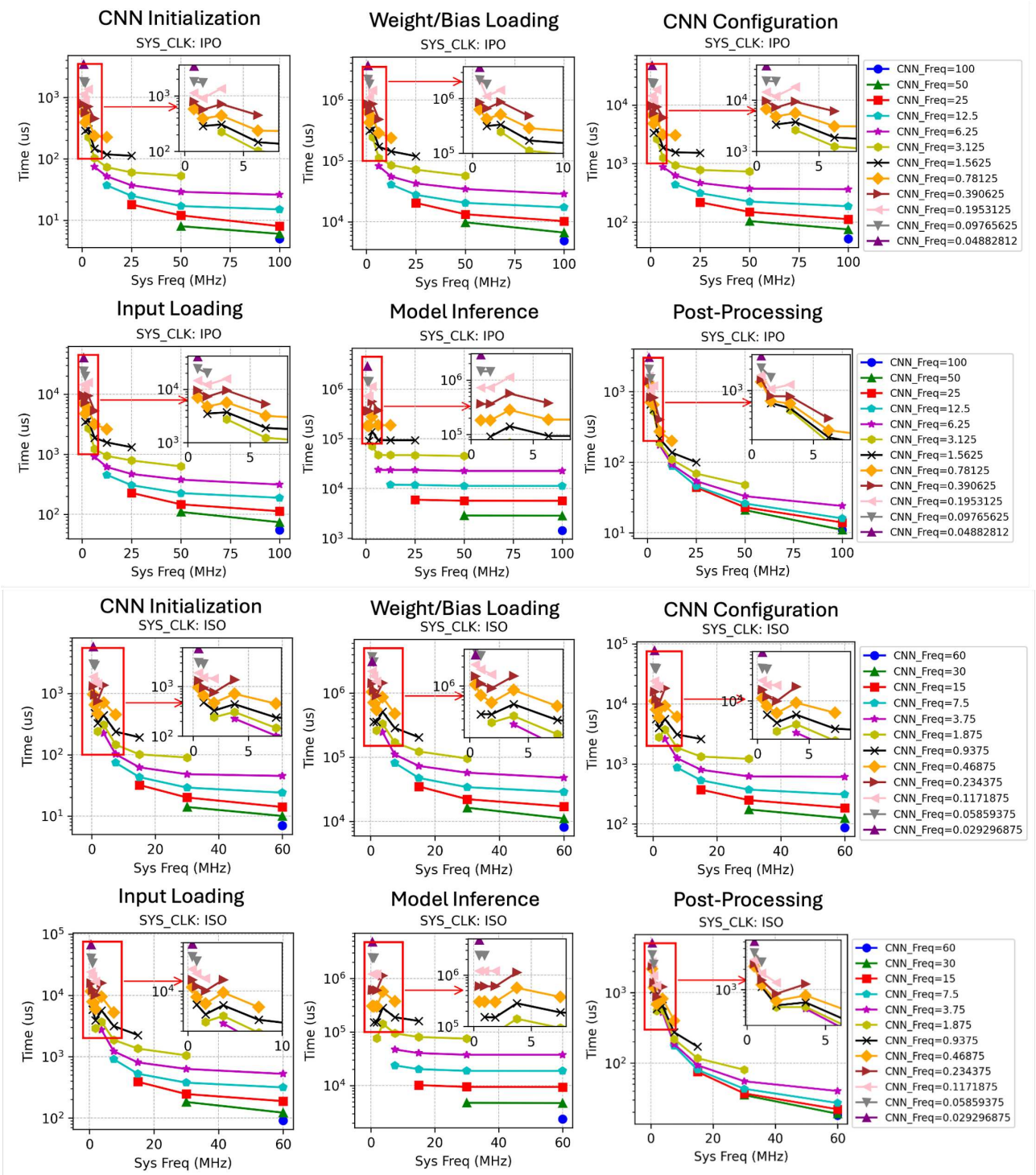


Figure 5.3: Relationship between latency and system frequency.

characteristics associated with each individual operation. Through detailed measurements and analysis, these experiments enabled us to discern the energy behavior under varying system configurations, thus facilitating the determination of the most efficient system configuration tailored to each operation. The macro-level experiments adopted a broader perspective, focusing on assessing the cumulative impact of these optimized configurations when integrated

across the entire processing pipeline.

In our experiments, we employed the Monsoon High Voltage Power Monitor [205] to precisely determine the **average power usage**, utilizing a sampling frequency of $50Hz$ and maintaining a constant input voltage of $3.0V$ to standardize measurement conditions. To calculate the **energy consumption**, we initially recorded the average power using this setup, followed by capturing the latency via the internal clock integrated within the MAX78000. The energy consumption was then derived using the product of time and power. To ensure the consistency and reliability of the measurements, we exclusively utilized data collected after a stabilization period of 1 minute of system operation, thereby mitigating potential fluctuations during the initial phase. All presented results represent the mean values obtained from ten independent experimental trials.

5.5.1 Micro Benchmark

In the micro-level evaluations, as detailed in Table 5.2, we conducted a comprehensive evaluation of each operation by taking into account core configuration control, frequency regulation, and CNN accelerator quadrant control, encompassing either partial or full adjustments. To maintain experimental precision and isolate the effects of a specific system configuration variable under investigation, all non-experimental system configurations are fixed, minimizing interference from extraneous factors. The non-experimental system configurations are: the processor is configured to ARM, the operation mode is set to ACTIVE, the system clock is IPO, both the system divider and CNN divider are fixed at 1, activate part CNN nodes, active CNN node contiguity, inactive CNN Boost, and idle implementation is default busy-waiting.

Sensing

We explored two practical scenarios: image capture via a camera, and audio recording via a microphone, both facilitated by integrated sensors embedded within the system. The sensing operation predominantly hinges on the strategic control of core configurations and the effective regulation of operational frequency, with outcomes documented in Fig. 5.4. Energy consumption is influenced by both power usage and latency. By implementing (a) ARM as the primary

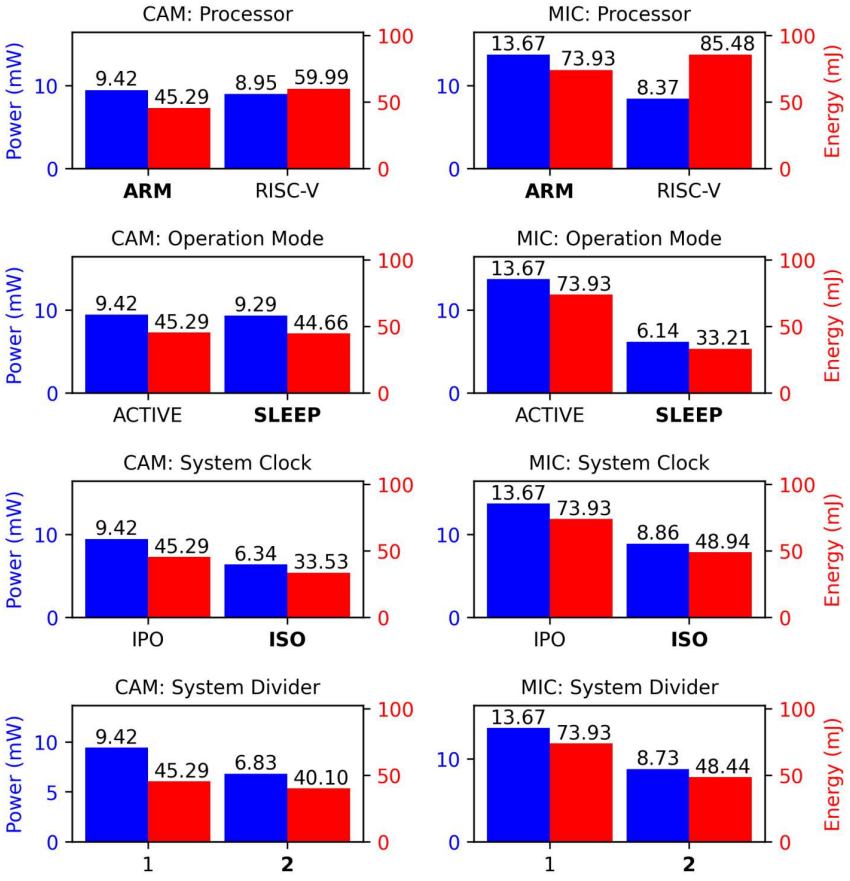


Figure 5.4: Micro-level assessment: sensing operation. Bold fonts indicate configurations with greater energy efficiency.

processor, (b) SLEEP operational mode, (c) ISO clock settings, and (d) a system divider set to 2, we achieved a reduction in energy cost. Compared against the configuration comprising RISC-V as the primary processor, ACTIVE mode, IPO clock, and a system divider of 1. The optimized configurations lowered energy consumption by average factors of 1.24×, 1.62×, 1.43×, and 1.33×, respectively.

Inference

We selected two representative neural network models for analysis: a simple 4-layer CNN network and a more complex 18-layer Unet [234], the latter incorporating encoder and decoder components, widely utilized in tasks requiring fine-grained feature extraction. To ensure the statistical reliability of the experimental results, we repeated the inference process of the simple CNN model 1,000 times and that of the more computationally demanding Unet model 100

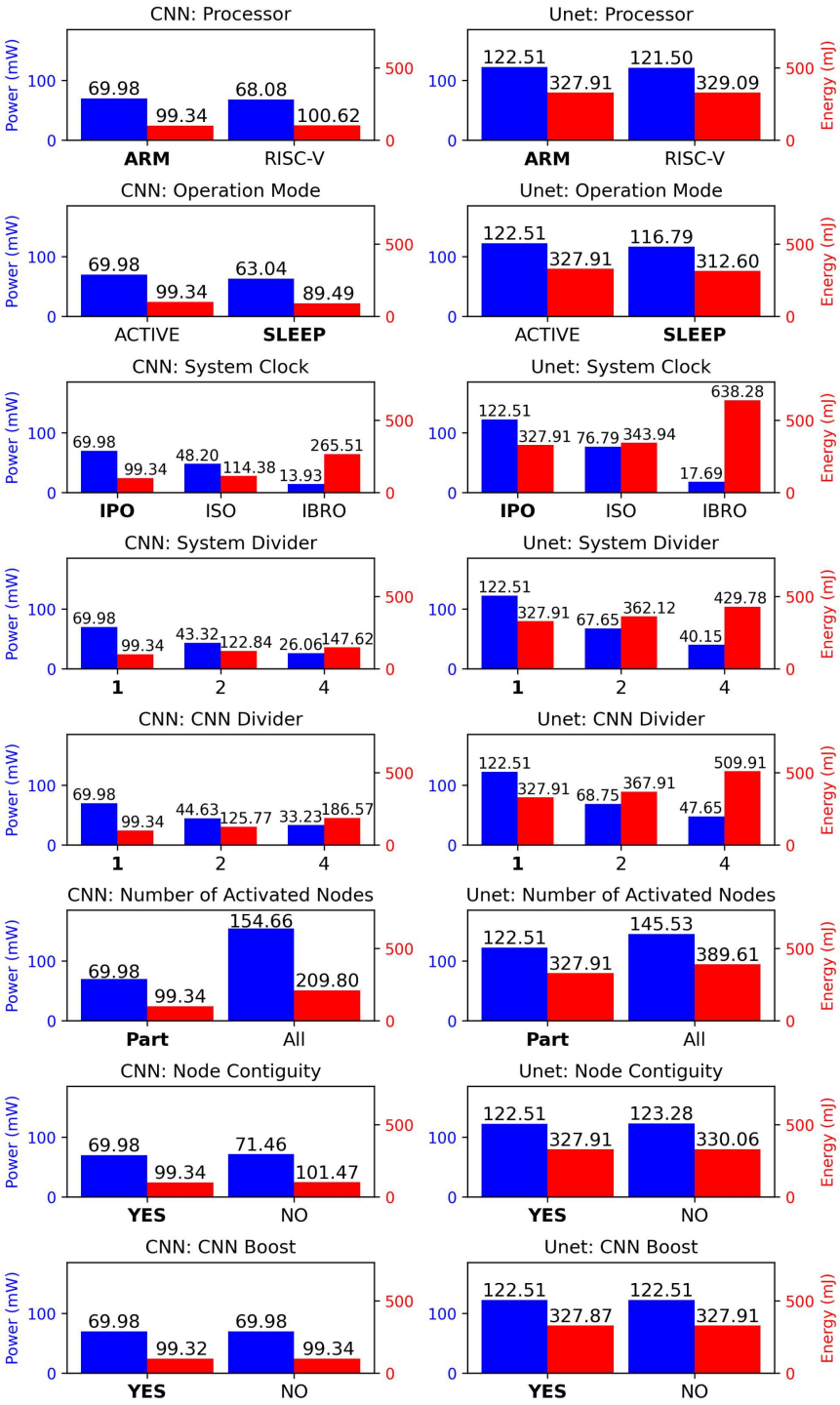


Figure 5.5: Micro-level evaluation: inference operation.

times. During the experiments, we systematically adjusted various parameters for the inference operation based on the control configurations listed in Table 5.2, presenting the results in Fig. 5.5.

The results indicated that the type of main processor (ARM versus RISC-V) had a negligible

Table 5.3: Micro-level power (measured in mW) and energy (measured in mJ) assessment of computation operation.

		Fibonacci		Softmax	
		Power	Energy	Power	Energy
Processor	**ARM**	13.96	37.76	14.29	30.83
	RISC-V	8.65	56.06	9.53	64.28
Operation Mode (Main: ARM)	**SLEEP**	13.96	37.76	14.29	30.83
	ACTIVE	14.72	39.82	14.80	31.93
Processor Mode (Main: RISC-V)	**SLEEP**	8.65	56.06	9.53	64.28
	ACTIVE	12.89	83.54	14.06	94.85
System Clock	**IPO**	13.96	37.76	14.29	30.83
	ISO	9.02	40.66	9.09	32.82
System Divider	**1**	13.96	37.76	14.29	30.83
	2	8.83	47.77	8.91	38.45
	4	6.38	68.78	6.42	55.40

effect on power and energy consumption, with an average difference of less than 0.1% between the two, suggesting that, for inference operation, the processor architecture itself is not a primary factor in energy optimization, and that energy consumption in inference operations is predominantly tied to the CNN accelerator. We can also find that transitioning the primary processor to SLEEP mode during inference was more effective than maintaining it in ACTIVE mode, resulting in a 7.29% decrease in both average power and energy usage.

In terms of CNN accelerator quadrant control, activating only the minimum number of nodes required for the task substantially reduced power/energy consumption, achieving an average reduction of 1.68 times, highlighting the importance of on-demand computational resource allocation for energy savings. In contrast, the optimization effects of node contiguity and CNN Boost were relatively limited, yielding only minor reductions in energy use. These slight improvements may stem from enhanced data transfer smoothness due to continuously active nodes, as well as the stable voltage provided by the CNN Boost circuit, which mitigates performance instability caused by voltage fluctuations and, to some extent, shortens processing time.

Computation

We selected two computational tasks as the subjects of our study: the calculation of the 34th Fibonacci number to validate basic arithmetic performance, and the Softmax computation based on a model with 100 network nodes, repeated 50,000 times, providing a representative

scenario for simulating common operations in neural networks. As a computation-intensive task, the COMP operation's performance is primarily influenced by core configuration control and frequency management; thus, we focused our experiments on these two aspects. The experimental results are presented in Table 5.3.

Across parameters such as processor type, system clock, and system divider, the results for the COMP operation exhibited patterns consistent with those observed in the inference operation, reflecting a uniform impact of these factors across task types. However, a distinct consideration emerged with respect to operation mode. Unlike the inference scenario, where the main processor could be shifted to SLEEP mode, the COMP operation relies directly on the main processor to execute computational logic, precluding its transition to a low-power state during task execution. Consequently, we redirected our focus to evaluate the influence of the secondary processor's operational mode on overall power and energy consumption while the main processor remained active in task processing. This approach sought to uncover potential energy efficiency gains in a multi-processor environment, where the secondary processor's configuration could complement the primary computational workload.

The experimental findings, as detailed in Table 5.3, highlight notable variations in power and energy consumption across different processor pairing configurations. When ARM is established as the primary processor, the additional activation of RISC-V in a supporting role leads to a modest upsurge in the system's overall power and energy usage, with an increase limited to 4.31%, indicating that this dual-processor arrangement imposes a relatively minor energy overhead. In stark contrast, when RISC-V assumes the primary role and ARM is brought online as the secondary processor, the disparity in energy consumption becomes markedly pronounced, soaring by 32.56%, underscoring the pivotal role of processor hierarchy in energy efficiency outcomes. These results yield valuable strategic implications: (1) For tasks necessitating the operation of a single processor, the most effective approach is to rely exclusively on the main processor, eschewing the activation of a secondary unit. Here, ARM, despite demanding greater instantaneous power, excels in total energy efficiency due to its swift execution and consequent reduction in task latency; RISC-V, on the other hand, consumes less power momentarily but incurs higher energy costs overall, likely attributable to slower processing speeds

Table 5.4: Micro-level power (in mW) analysis: default idle implementation.

Processor		System Clock		System Divider		
ARM	RISC-V	IPO	ISO	1	2	4
13.11	10.76	13.11	8.60	13.11	8.42	6.19

Table 5.5: The power (in mW) of various idle implementations.

	Default	Wake-Up Source	SLEEP	LPM	**UPM**
ARM	13.11	WUT	6.11	5.33	2.02
		LPT	6.11	5.33	2.02
RISC-V	10.76	WUT	6.41	5.42	2.02
		LPT	6.41	5.42	2.02

that prolong task completion. (2) In more intricate scenarios where dual-processor collaboration is required—such as cooperative processing or parallel computation aimed at amplifying performance—the configuration pairing ARM as the lead processor with RISC-V in a subordinate capacity demonstrates superior energy efficiency. This arrangement harnesses ARM's robust computational prowess while augmenting it with RISC-V's supplementary contribution, achieving an enhanced processing capability with a restrained escalation in energy expenditure.

Idle

We first undertook a meticulous evaluation of the power consumption exhibited by the default busy-waiting implementation, aiming to scrutinize its performance and energy characteristics under standard configurations. As documented in Table 5.4, the adoption of RISC-V as the principal processor, in conjunction with reduced system clock frequencies and minimized divider values, leads to a pronounced reduction in the system's power draw. This evidence suggests that, for the Idle operation, tempering the operational frequency and designating RISC-V as the main processor constitute a viable approach to mitigating instantaneous power requirements at the hardware level. Nonetheless, the intrinsic operational paradigm of busy-waiting imposes distinct constraints: to maintain real-time sensitivity to external signals or events, all associated components are compelled to sustain uninterrupted activity. While this continuous engagement ensures swift reactivity, it prolongs the system's exposure to heightened power states, consequently amplifying the cumulative energy consumption over time.

Due to the energy efficiency constraints identified with the busy-waiting approach, we proceeded to investigate a range of alternative idle implementation strategies aimed at substantially cur-

tailing energy consumption during inactive periods. Table 5.1 enumerates the power-saving modes available on the MAX78000, encompassing SLEEP, Low-Power Mode (LPM), and Ultra-Low-Power Mode (UPM), each offering progressively deeper sleep states that correspondingly diminish the count of active system components, thereby reducing overall power usage. To enable efficient reactivation from these non-ACTIVE states, we examined two wake-up mechanisms: the Wake-Up Timer (WUT) and the Low-Power Timer (LPT), both engineered to reliably transition the MAX78000 back to operational status from power-saving conditions. The power consumption profiles of these diverse idle strategies are cataloged in Table 5.5 for comparative analysis. The findings reveal that UPM mode delivers the most substantial reduction in power consumption across all tested implementations, registering a mere 2.02mW, a result attributable to its minimal activation of system components, optimizing energy utilization during idle phases. Intriguingly, the data further indicate that this power consumption level of approximately 2.02mW remains consistent, irrespective of whether UPM mode is initiated via the ARM or RISC-V processor, or whether reactivation is triggered by the WUT or LPT mechanism, underscoring the mode's uniformity across varying configurations and wake-up pathways.

5.5.2 Macro Benchmark

After conducting a micro-level analysis of each operation, we proceeded to a macro-level analysis two cases: speech recognition (using KWS20 model [235]) and image segmentation (using 18-layer Unet). Drawing from the outcomes of micro-level experiments, we identified the system configuration variables with notable effects: processor type (ARM or RISC-V), operating mode (ACTIVE or SLEEP), system clock source (IPO or ISO), system divider setting (1 or 2), CNN divider value (1 or 2), number of active nodes (partial or full), and idle strategy.

Speech Recognition Experiment

We employed a speech recognition experiment. This experiment utilizes an 8-layer CNN network architecture with an input size of 128×128. We presume that every pipeline execution cycle takes 2.5 seconds (overall duration), with the idle period influenced by the total time required for the operations within an application. This ratio of working time to idle time is

Table 5.6: System configurations of the speech recognition experiments.

	Configure 1 (Default)							Configure 2 (Change SYS_CLK)						
	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	ISO	ISO	ISO	ISO	ISO	ISO	ISO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	-	1	1	1	-	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay							Active Part CNN Nodes, Idle Method = MXC_Delay						
	Configure 3 (Change SYS_DIV)							Configure 4 (Change Processor Mode)						
	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	SLEEP	ACTIVE	ACTIVE	SLEEP	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	2	2	2	2	2	2	2	1	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	-	1	1	1	-	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay							Active Part CNN Nodes, Idle Method = MXC_Delay						
	Configure 5 (Change CNN_DIV)							Configure 6 (Change Number of Activated CNN Nodes)						
	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	2	2	2	-	-	-	-	1	1	1	-	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay							Active All CNN Nodes, Idle Method = MXC_Delay						
	Configure 7 (Change Idle Strategy							Configure 8 (Change Processor)						
	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	RISCV	RISCV	RISCV	RISCV	RISCV	RISCV	RISCV
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	-	1	1	1	-	-
Others	Active Part CNN Nodes, Idle Method = WUT_UPM							Active Part CNN Nodes, Idle Method = MXC_Delay						
	Configure 9 (Best)													
	Sensing	COMP_1	MIO_1	Inference	MIO_2	COMP_2	Idle							
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM							
Mode	SLEEP	ACTIVE	ACTIVE	SLEEP	ACTIVE	ACTIVE	SLEEP							
SYS_CLK	ISO	IPO	IPO	IPO	IPO	IPO	ISO							
SYS_DIV	2	1	1	1	1	1	2							
CNN_DIV	-	-	1	1	1	-	-							
Others	Active Part CNN Nodes, Idle Method = WUT_UPM													

approximately 1:3, which aligns with real-world scenarios.

We formulated a total of nine distinct experimental configurations, as shown in Table 5.6 to facilitate systematic analysis and reference. We adopted the pre-established default configuration of the MAX78000 as the baseline benchmark (denoted as Configuration 1). To delve into the effects of individual system configurations on the system cost throughout the entire application pipeline, we introduced seven additional experimental setups (Configurations 2 through 8). These were crafted by altering a single system configuration from the configuration set while preserving the alignment of all other parameters with the default configuration. For example, to examine the influence of processor selection, we only reconfigured the processor to RISC-V, while retaining the other configurations same with the default settings, isolating the processor type as the variable under scrutiny. Furthermore, we designed a configuration (Configuration 9), named 'Best' tailored to minimize working energy consumption during the execution of the complete application pipeline.

Overall Analysis. The experimental results are illustrated in the Table 5.7. We observe that changing a specific system configuration affects power consumption, energy consumption, and latency differently across various operations. These effects also manifest differently when considering the overall pipeline's power, energy, and latency. For instance, comparing **Configure 2 (SYS_CLK=ISO)** with **Configure 1 (SYS_CLK=IPO)**, the power consumption for all operational stages (excluding Idle) decreases. This is expected, as reducing the system frequency leads to lower power consumption. In terms of latency, all operations (except for 'Sensing', which shows a slight increase) slow down by **1.57× to 1.68×**. Energy consumption, influenced by both latency and power, shows that only the 'sensing' operation decreases by **1.57×**, while the energy consumption of other operations increases to varying degrees.

When examining the **Working Energy** (excluding idle), despite most operations in **Configure 2** consuming more energy compared to **Configure 1**, the overall working energy of **Configure 2** is lower. This is primarily because the 'sensing' operation, which dominates the energy consumption, experiences a more significant reduction in energy in **Configure 2**. Considering the **total energy consumption** (including idle), **Configure 1**, with its higher IPO clock

Table 5.7: Results of the speech recognition experiments.

Configure 1	Time (us)	Power (mw)	Energy (mJ)	**Configure 2**	Time (us)	Power (mw)	Energy (mJ)	**Configure 3**	Time (us)	Power (mw)	Energy (mJ)
Sensing	398759	13.69	5.459	Sensing	402734	8.64	3.480	Sensing	417268	8.42	3.513
COMP_1	46018	12.83	0.590	COMP_1	77092	8.31	0.641	COMP_1	92054	8.12	0.747
MIO_1	132435	25.27	3.347	MIO_1	221615	16.88	3.741	MIO_1	264745	16.22	4.294
Inf	14631	106.73	1.562	Inf	24459	66.58	1.628	Inf	29197	57.97	1.693
MIO_2	68	10.77	0.001	MIO_2	107	9.04	0.001	MIO_2	135	8.53	0.001
COMP_2	1292	16.73	0.022	COMP_2	2177	12.53	0.027	COMP_2	2594	12.03	0.031
Idle	1906797	15.60	29.746	Idle	1771816	10.26	18.179	Idle	1694007	15.60	26.427
Working	593203	18.51	10.980	Working	728184	13.07	9.518	Working	805993	12.75	10.280
Total	2500000	16.29	40.726	Total	2500000	11.08	27.697	Total	2500000	14.68	36.706
Configure 4	Time (us)	Power (mw)	Energy (mJ)	**Configure 5**	Time (us)	Power (mw)	Energy (mJ)	**Configure 6**	Time (us)	Power (mw)	Energy (mJ)
Sensing	398759	6.25	2.492	Sensing	398759	13.69	5.459	Sensing	398759	13.69	5.459
COMP_1	46018	12.83	0.590	COMP_1	46018	12.83	0.590	COMP_1	46018	12.83	0.590
MO_1	132435	25.27	3.347	MO_1	179897	18.86	3.393	MO_1	132435	25.27	3.347
Inf	14631	98.81	1.446	Inf	28925	62.28	1.801	Inf	14631	156.67	2.292
MO_2	68	10.77	0.001	MO_2	84	10.09	0.001	MO_2	68	10.77	0.001
COMP_2	1292	16.73	0.022	COMP_2	1292	16.73	0.022	COMP_2	1292	16.73	0.022
Idle	1906797	15.6	29.746	Idle	1845025	15.6	28.782	Idle	1906797	15.6	29.746
Working	593203	13.31	7.897	Working	654975	17.20	11.266	Working	593203	19.74	11.711
Total	2500000	15.06	37.643	Total	2500000	16.02	40.049	Total	2500000	16.58	41.457
Configure 7	Time (us)	Power (mw)	Energy (mJ)	**Configure 8**	Time (us)	Power (mw)	Energy (mJ)	**Configure 9**	Time (us)	Power (mw)	Energy (mJ)
Sensing	398759	13.69	5.459	Sensing	737704	8.35	6.160	Sensing	429413	3.68	1.580
COMP_1	46018	12.83	0.590	COMP_1	128852	8.12	1.046	COMP_1	46018	12.83	0.590
MIO_1	132435	25.27	3.347	MIO_1	214544	22.79	4.889	MIO_1	132435	25.27	3.347
Inf	14631	106.73	1.562	Inf	14942	105.36	1.574	Inf	14631	98.81	1.446
MIO_2	68	10.77	0.001	MIO_2	111	9.74	0.001	MIO_2	68	10.77	0.001
COMP_2	1292	16.73	0.022	COMP_2	3812	10.61	0.040	COMP_2	1292	16.73	0.022
Idle	1906797	2.02	3.852	Idle	1400035	15.6	21.841	Idle	1876143	2.02	3.790
Working	593203	18.51	10.980	Working	1099965	12.47	13.711	Working	623857	11.20	6.985
Total	2500000	5.93	14.832	Total	2500000	14.22	35.552	Total	2500000	4.31	10.775

frequency, completes tasks faster, leading to increased 'idle' time and power, thus higher 'idle' energy consumption compared to **Configure 2**. This demonstrates that a faster clock does not necessarily result in better energy efficiency.

Additionally, we observe that during idle operations, the default busy-waiting introduces unnecessary power consumption, sometimes even exceeding the power required for certain active operations. This highlights the importance of optimizing idle strategies to minimize energy waste.

Default *vs.* **Best.** A comparative analysis of the 'Best' and 'Default' configurations unveils several pivotal distinctions, primarily centered on the system clock and divider settings during sensing and idle operations, the operational modes employed across sensing, inference, and idle phases, and the idle implementation strategy. These disparities underscore a critical insight: the default configuration of the MAX78000 is tailored to enhance efficiency in only select segments of the operational sequence—namely COMP_1, MIO_1, inference, MIO_2, and COMP_2—while overlooking the comprehensive energy demands of the end-to-end pipeline, particularly neglecting the sensing and idle operations.

Moreover, the study reveals that uniformly imposing a static system configuration across the entire pipeline falls short of achieving optimal energy efficiency, given that the most energy-efficient settings vary distinctly across different operations, rendering a one-size-fits-all approach suboptimal. In contrast, the 'Best' configuration adopts a holistic and adaptive methodology, meticulously accounting for the unique requirements of each operation within the pipeline and dynamically tailoring system parameters—such as clock frequency, divider values, and mode selections—to minimize energy usage at every stage. Empirical evidence substantiates the efficacy of this approach: relative to the 'Default' configuration, the 'Best' configuration achieves a remarkable reduction in **total energy** by **3.78**× and in **working energy** by **1.57**×, demonstrating substantial energy savings. For power, the 'Best' configuration reduces **3.78**× total power and **1.65**× working power. Regarding latency, the 'Best' configuration exhibits a marginal slowdown compared to the default setup solely during the sensing operation. However, in application contexts characterized by waiting intervals—such as those with intermittent

task execution—this slight prolongation of sensing latency exerts a negligible effect on overall performance.

Image Segmentation Experiment

Similarly, we employed an image segmentation experiment. This experiment utilizes an 18-layer UNet network architecture with an input size of $48 \times 48 \times 3$. We assume that each pipeline execution cycle lasts for 3 seconds. Similar to the system configuration in the Speech Recognition experiment, we also defined a default configuration (**Configure 1**), seven experimental configurations (**Configure 2** to **Configure 8**), and an optimal working energy configuration (**Configure 9**). Note that since the on-board camera has a minimum requirement for frequency, in configure 3 (change SYS_DIV) we keep SYS_DIV=1 in the 'sensing' operation. The detailed experimental configurations are shown in Table. 5.8.

Overall Analysis. The experimental results are shown in Table 5.9. Similar to the results of the speech recognition experiments, changing specific system configurations has varying impacts on the power consumption, energy consumption, and latency of different operations. When considering the overall pipeline's power, energy, and latency, the extent of the impact caused by changes in a single operation also varies.

For example, comparing **Configure 9 (Processor=RISC-V)** with **Configure 1 (Processor=ARM)**, the power consumption for all operational stages (excluding 'idle') decreases, while the latency increases. This is consistent with the micro-level experimental results. As for energy consumption, the difference in the 'inference' stage is minimal, with both configurations consuming approximately 3.509mJ. This is a combined effect of latency and power, and since the 'inference' operation is primarily related to the CNN accelerator, the change of the main processor does not significantly impact it.

Overall, the difference in energy consumption of the 'sensing' operation contributes the most to the difference in **working energy**, accounting for approximately **85.36%**. For the **total energy** (including idle), the 'idle' operation dominates the energy consumption, further highlighting the importance of optimizing idle strategies.

Table 5.8: System configurations of the image segmentation experiments.

	Configure 1						Configure 2					
	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	ISO	ISO	ISO	ISO	ISO	ISO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	1	1	1	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay						Active Part CNN Nodes, Idle Method = MXC_Delay					
	Configure 3						Configure 4					
	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	SLEEP	ACTIVE	ACTIVE	SLEEP	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	1	2	2	2	2	2	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	1	1	1	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay						Active Part CNN Nodes, Idle Method = MXC_Delay					
	Configure 5						Configure 6					
	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM	ARM
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	2	2	2	-	-	-	1	1	1	-
Others	Active Part CNN Nodes, Idle Method = MXC_Delay						Active All CNN Nodes, Idle Method = MXC_Delay					
	Configure 7						Configure 8					
	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle
Processor	ARM	ARM	ARM	ARM	ARM	ARM	RISCV	RISCV	RISCV	RISCV	RISCV	RISCV
Mode	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE	ACTIVE
SYS_CLK	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO	IPO
SYS_DIV	1	1	1	1	1	1	1	1	1	1	1	1
CNN_DIV	-	-	1	1	1	-	-	-	1	1	1	-
Others	Active Part CNN Nodes, Idle Method = WUT_UPM						Active Part CNN Nodes, Idle Method = MXC_Delay					
	Configure 9											
	Sensing	COMP_1	MIO_1	Inference	MIO_2	Idle						
Processor	ARM	ARM	ARM	ARM	ARM	ARM						
Mode	SLEEP	ACTIVE	ACTIVE	SLEEP	ACTIVE	SLEEP						
SYS_CLK	ISO	IPO	IPO	IPO	IPO	ISO						
SYS_DIV	2	1	1	1	1	2						
CNN_DIV	-	-	1	1	1	-						
Others	Active Part CNN Nodes, Idle Method = WUT_UPM											

Table 5.9: Results of the image segmentation experiments.

Configure 1	Time (us)	Power (mw)	Energy (mJ)	**Configure 2**	Time (us)	Power (mw)	Energy (mJ)	**Configure 3**	Time (us)	Power (mw)	Energy (mJ)
Sensing	660342	9.27	6.121	Sensing	753073	6.24	4.699	Sensing	660342	9.27	6.121
COMP_1	1207	13.47	0.016	COMP_1	2421	8.83	0.021	COMP_1	2709	8.43	0.023
MO_1	27368	25.74	0.704	MO_1	45803	17.83	0.817	MO_1	54747	15.92	0.872
Inf	28246	124.23	3.509	Inf	49267	80.91	3.986	Inf	56480	65.94	3.724
MO_2	3206	24.24	0.078	MO_2	5366	17.03	0.091	MO_2	6413	15.42	0.099
Idle	2279631	14.76	33.647	Idle	2144070	9.23	19.790	Idle	2219309	9.02	20.018
Working	720369	14.48	10.429	Working	855930	11.23	9.615	Working	780691	13.88	10.839
Total	3000000	14.69	44.076	Total	3000000	9.80	29.405	Total	3000000	10.29	30.857
Configure 4	Time (us)	Power (mw)	Energy (mJ)	**Configure 5**	Time (us)	Power (mw)	Energy (mJ)	**Configure 6**	Time (us)	Power (mw)	Energy (mJ)
Sensing	660342	9.05	5.976	Sensing	660342	9.27	6.121	Sensing	660342	9.27	6.121
COMP_1	1207	13.47	0.016	COMP_1	1207	13.47	0.016	COMP_1	1207	13.47	0.016
MO_1	27368	25.74	0.704	MO_1	37187	18.9	0.703	MO_1	27368	25.74	0.704
Inf	28246	117.13	3.308	Inf	56477	70.98	4.009	Inf	28246	143.79	4.061
MO_2	3206	24.24	0.078	MO_2	4002	18.27	0.073	MO_2	3206	24.24	0.078
Idle	2279631	14.76	33.647	Idle	2240785	14.76	33.074	Idle	2279631	14.76	33.647
Working	720369	14.00	10.083	Working	759215	14.39	10.922	Working	720369	15.24	10.981
Total	3000000	14.58	43.730	Total	3000000	14.67	43.996	Total	3000000	14.88	44.629
Configure 7	Time (us)	Power (mw)	Energy (mJ)	**Configure 8**	Time (us)	Power (mw)	Energy (mJ)	**Configure 9**	Time (us)	Power (mw)	Energy (mJ)
Sensing	660342	9.27	6.121	Sensing	924478	8.81	8.145	Sensing	753073	6.24	4.699
COMP_1	1207	13.47	0.016	COMP_1	3742	8.69	0.033	COMP_1	1207	13.47	0.016
MO_1	27368	25.74	0.704	MO_1	43801	22.98	1.007	MO_1	27368	25.74	0.704
Inf	28246	124.23	3.509	Inf	28794	121.85	3.509	Inf	28246	124.23	3.509
MO_2	3206	24.24	0.078	MO_2	4969	21.65	0.108	MO_2	3206	24.24	0.078
Idle	2279631	2.02	4.605	Idle	1994216	12.09	24.110	Idle	2186900	2.02	4.418
Working	720369	14.48	10.429	Working	1005784	12.73	12.800	Working	813100	11.08	9.007
Total	3000000	5.01	15.034	Total	3000000	12.30	36.910	Total	3000000	4.47	13.424

Default *vs.* **Best.** We implemented the 'Best' system configuration using a dynamic configuration approach. Compared to the 'Default' configuration, the 'Best' configuration achieves significant reductions in both **total energy** (3.28×) and **working energy** (1.16×), demonstrating considerable energy savings. In terms of power, the 'Best' configuration also achieves significant reductions in **total power** (3.29×) and **working power** (1.31×). Regarding latency, compared to the 'Default' configuration, the 'Best' configuration exhibits only a slight slowdown during the sensing operation.

5.6 Discussions and Limitations

In this section, we discuss certain constraints of this research.

• **Limited Experimental Scenarios/Cases.** Within the micro-level experiments, we devised two distinct scenarios for each operation to evaluate their energy consumption, whereas in the macro-level experiments, we employed speech recognition and image segmentation as representative cases to examine the system's overall costs. Despite the valuable insights yielded by these experimental setups, it is imperative to acknowledge that our investigation does not encompass the full spectrum of potential scenarios and sensor modalities, such as accelerometers, among others. Future research can focus on incorporating a wider array of sensor types—such as accelerometers and gyroscopes—and a more extensive set of application contexts, including motion recognition and environmental monitoring, to conduct a deeper exploration of the energy characteristics of AI accelerators.

• **Generalizability of Conclusions.** This chapter uses the MAX78000 as an example of AI-MCU. However, extending the conclusions of this chapter to other MCUs equipped with AI accelerators, such as ARM Ethos-U65 [226] and GreenWaves GAP9 [227], requires more cautious validation and experimentation. The main reasons include: (1) the MAX78000's hybrid architecture (ARM, RISC-V, and CNN accelerator) may differ from other AI-MCUs, for instance, in mechanisms for frequency control, sensor interaction, and memory I/O; (2) our CNN quadrant-based control optimizations are hardware-specific, relying on the MAX78000's CNN accelerator and its unique structure, which may not exist in other devices; (3) due to differences

in semiconductor processes, power, energy consumption, and latency curves may vary (we confirmed this in a subsequent research [236]); (4) our idle implementation optimization depends on the MAX78000's specific mode implementation, while other AI-MCUs may offer different mode state hierarchies or wakeup overheads. These architecture- and implementation-specific factors necessitate careful re-validation when applying our conclusions to other platforms.

• **Dynamic Configuration Overhead.** We empirically measured the latency and energy cost associated with dynamic configuration switches, such as frequency scaling or power mode transitions. Results show that a single switch incurs only approximately 1 us of latency and consumes power comparable to the default idle state (around 15 mW). This suggests that the per-switch energy overhead is negligible relative to the total energy consumption of typical applications. However, frequent reconfiguration can still introduce measurable latency accumulation, which may be non-negligible for ultra-low-latency applications. Moreover, excessive switching may trigger power management artifacts, potentially disturbing sensor analog front-ends in precision-critical scenarios [237, 238]. For systems with stringent real-time requirements, we recommend using hardware-assisted solutions such as pre-configured register sets or state machines to eliminate runtime switching latency [239].

5.7 Conclusions

In this chapter, we explored the energy characteristics of the ultra-low-power DNN accelerator MAX78000. The system configurations we investigated cover three domains—core configuration control, frequency management, and quadrant-based control, total eight system configurations. Our approach commenced at the micro level, where we evaluate the energy and power characteristic of five operations from application pipeline. We then conducted macro-level evaluation, regarded the pipeline as a whole, and evaluated the overall impacts from each single system configuration. We find that, the configuration yielding optimal energy efficiency is not static to each operation. Compared to the default static system configuration, the dynamic configuration diminished total energy by 3.53×, working energy by 1.37×, total power by 3.54×, and working power by 1.48×. Beyond the concrete data and conclusions presented, this benchmark endeavor extends significant implications for broader technological spheres.

Chapter 6

Summary and Future Outlook

6.1 Conclusions and Contributions

Artificial intelligence is demonstrating a trend of shifting from the cloud to the edge, driven by its advantages in enhancing privacy protection and reducing latency. In this thesis, I have developed and implemented methods/algorithms/systems to improve the adaptation of edge computing from the perspectives of data, models, and systems. The main contributions are summarized as follows:

• This thesis first explores methods to enhance edge computing adaptation from a data perspective. Using a decentralized healthcare monitoring platform as a case study, we propose InTS to better handle multivariate, multi-source, state-varying, and noisy medical time-series data. For state-transition data, we developed a series of entropy-based methods grounded in first-order Markov chains, including Shannon entropy, entropy rate, entropy production, and von Neumann entropy. For non-state-transition data, we first computed a series of entropy variants, followed by a dual feature selection mechanism using PCC and MI to identify features with low collinearity and high relevance to labels. For state-transition data, we used average activity frequency and duration as baseline features, evaluating the performance of baseline and InTS-extracted features across LR, SVM, MLP, and LSTM models. Results show that InTS improved accuracy, recall, and F1 score by an average of 13.08 pp, 10.80 pp, and 7.88 pp, re-

spectively. For non-state-transition data, we combined InTS-extracted features with the MLP model and compared them against CNN, LSTM, and CNN-LSTM models. InTS improved accuracy, recall, and F1 score by an average of 2.52 pp, 2.42 pp, and 2.32 pp, respectively. Additionally, InTS reduced model parameters by 3.10×. InTS demonstrates effectiveness and efficiency on medical time-series data. However, this work still has limitations, including the inherent shortcomings of first-order Markov chains in long sequences, the applicability of fixed thresholds in the dual selection mechanism, and the lack of validation on other data modalities (*e.g.*, images) due to constraints of the healthcare platform and data access.

• Subsequently, this thesis explores enhancing edge computing adaptation from a model perspective. Using mobile devices with moderate computational capabilities (specifically Raspberry Pi in this chapter) as the experimental platform, we investigated efficient model personalization to address data drift. We propose TBFT, a low-energy personalization framework that tackles various data drifts by selectively fine-tuning different model blocks. Specifically, we categorize data drift into three types—input-, feature-, and output-level—corresponding to fine-tuning the front, middle, and rear modules/blocks of the model, respectively. We provide mathematical definitions for these three drift types and validate TBFT's rationality and effectiveness through theoretical analysis and experimental results. Public image datasets were used as experimental data due to access restrictions on the medical dataset from Chapter 2, and image data more easily reflects the three drift types. Experimental results show that, compared to full-model fine-tuning, TBFT reduces energy consumption by 41.57% on average while outperforming the averaged block-wise fine-tuning baseline (*BlockAvg*) by 15.30% in average accuracy. However, TBFT still has limitations, including the need for prior knowledge of local drift types, limited dataset/model evaluation (tested only on module-based models), and the need for more precise middle block partitioning.

• Subsequently, I shifted the experimental platform from mobile devices to ultra-resource-constrained MCUs with RAM typically less than 1MB, further explored personalization challenge for edge computing. Such extremely constrained resources make TBFT difficult to apply on MCUs, as it introduces additional memory overhead or errors in quantization format support. Therefore, I proposed MicroT, an efficient and low-energy MCU personalization ap-

proach. MicroT includes a robust, general, yet lightweight feature extractor developed through self-supervised knowledge distillation, which trains a task-specific head to enable independent on-device personalization with minimal energy and computational requirements. Additionally, MicroT implements an MCU-optimized early-exit inference mechanism, termed stage-decision, which allows user-configurable exit criteria (stage-decision ratio) to adaptively balance energy cost with model performance. I evaluated MicroT using two models, three datasets, and two MCU boards. The results show that MicroT outperforms traditional transfer learning and two state-of-the-art approaches in model performance, local training efficiency, and local inference efficiency. Targeting widely used energy-aware edge devices, MicroT's on-device training requires no additional complex operations while keeping SRAM usage below 1MB, demonstrating its suitability for long-term use on energy-aware, resource-constrained MCUs. However, MicroT also has limitations, including the continued need for cloud-based training of the feature extractor, potential instability in model split points and stage-decision thresholds, and the assumption that local data labels have already been collected.

• Finally, I explored how to optimize energy consumption and power in edge computing from a system configuration perspective. Employing Analog Devices' MAX78000 ultra-low-power MCU, I conducted an in-depth evaluation of its energy characteristics. I outlined the hardware components and their compatible power settings, followed by thorough benchmarking at both micro- and macro-levels. Specifically, I divided the complete application pipeline into six operations: sensing, initialization, memory I/O, model inference, CPU computation, and idle. I also explored three system configuration dimensions—core configuration control, frequency management, and CNN accelerator quadrant control—incorporating eight adjustable variables. At the micro-level, I evaluated the power/energy consumption of different system configurations across individual operations. At the macro-level, I assessed how specific system configurations influence the total energy consumption of end-to-end application pipelines. Beyond the benchmarks, I discovered that dynamically adjusting system configurations during different operations achieves optimal energy efficiency. Compared to the default static configuration, dynamic configuration reduced total energy by 3.53×, working energy by 1.37×, total power by 3.54×, and working power by 1.48×. However, this work still has limitations, includ-

ing limited experimental scenarios/cases, the applicability of MAX78000 findings to other AI accelerators, and potential impacts of dynamic configuration.

6.2 Discussion and Future Work

With the rapid development of the IoT, edge computing is increasingly becoming an integral part of our daily lives. Efficient and adaptive edge computing remains a key research focus both now and in the future. Notably, as the variety of sensors continues to grow, models/algorithms/systems relying solely on single signal or data sources may face performance limitations. Leveraging and processing multimodal data can significantly enhance the adaptability of edge computing. Furthermore, the advent of big data has led to rising costs for data annotation, and the proportion of unlabeled data is also increasing. Therefore, how to better utilize unlabeled data and achieve unsupervised edge computing is significant for the practical application of edge computing.

• **Efficient and Adaptive Edge Computing.** Although this thesis investigates efficient and adaptive edge computing from the perspectives of data, models, and systems, achieving even more efficient and adaptive edge computing remains a critical research focus both now and in the future. For instance, at the data level, more efficient feature extraction and data mining techniques can be explored to further improve the efficiency and accuracy of data processing [240, 241]. At the model level, on-device personalization technologies need further optimization, including developing more lightweight model architectures, more efficient fine-tuning strategies, and smarter model compression methods. For example, dynamic model compression techniques [242] and sparse inference techniques [243, 244] can significantly reduce model complexity, thereby achieving lower energy consumption and higher performance on resource-constrained devices. Additionally, novel neural network architectures, such as binary neural networks [245], may further enhance the efficiency of edge computing. At the system level, optimization techniques such as memory management, parallel computing, and task scheduling require in-depth research. For instance, designing more efficient memory allocation algorithms, exploring the potential of hardware accelerators, and optimizing resource allocation strategies for multi-task concurrent execution [246] can significantly improve the overall efficiency and adaptability of

edge devices. These new technologies can be flexibly integrated with the methods proposed in this thesis. For example, new feature extraction techniques can be combined with TBFT and MicroT to achieve more efficient model personalization; new memory optimization and parallel computing techniques can also be integrated with InTS, TBFT, and MicroT to further reduce energy consumption, power, and latency.

• **Multimodal Edge Computing.** With the proliferation of IoT devices and advancements in sensor technologies, multimodal data (*e.g.*, visual, audio, text, and sensor data) is increasingly being utilized in edge computing. Multimodal edge computing enhances the performance and adaptability of edge intelligence systems by integrating and analyzing data from diverse modalities, providing richer and more accurate contextual information. For instance, in smart home scenarios, combining visual and audio data can improve the accuracy of human activity recognition; in industrial IoT, fusing sensor data with equipment logs enables more efficient fault prediction and maintenance. However, challenges remain in how to better and more efficiently leverage multimodal features to improve model performance and robustness, such as the heterogeneity of multimodal data [247, 248], the complexity of cross-modal alignment [249, 250], and the demand for efficient computation on resource-constrained devices. The techniques developed in this thesis—InTS, TBFT, system configuration optimization, and MicroT—provide a strong foundation for advancing multimodal edge computing. InTS's information-theoretic framework could be extended to handle multimodal time-series data, such as synchronized biosignals and motion sensor streams, by developing modality-specific feature extraction methods while retaining its efficiency for edge deployment. TBFT's targeted fine-tuning approach could address modality-specific drift, where different data types (e.g., visual vs. sensor data) experience shifts at varying rates, enabling selective adaptation of relevant model components rather than full retraining. Our system-level energy optimization techniques could be adapted to multimodal workloads by analyzing the power consumption patterns of cross-modal operations (*e.g.*, audio-visual fusion) and dynamically adjusting hardware configurations to balance performance and efficiency. The effectiveness of techniques such as SSKD and stage-decision in MicroT can be further validated on other modalities (e.g., audio data), enabling better application in multimodal contexts. For instance, multimodal features can be fused and then

processed with SSKD to enhance the generality and robustness of the feature extractor.

• **Unsupervised Edge Computing.** With the proliferation of IoT devices and enhanced data collection capabilities, the proportion of unlabeled data in edge computing has significantly increased. Unsupervised Edge Computing leverages unlabeled data and unsupervised learning techniques to effectively reduce data annotation costs and enhance the adaptability of edge intelligence systems. For instance, in smart surveillance scenarios, unsupervised learning can automatically detect anomalous behaviors [251]; in industrial equipment monitoring, unsupervised methods can extract key features from unlabeled sensor data for fault detection [252]. InTS's information-theoretic framework could be extended to quantify and filter noisy patterns in unlabeled sensor data, while its efficient feature extraction capability makes it particularly suitable for processing high-dimensional unlabeled data on resource-constrained devices. TBFT's targeted adaptation approach presents a promising direction for unsupervised model personalization, where only specific components of unsupervised models (*e.g.*, the encoder in an autoencoder) could be selectively fine-tuned to adapt to new data patterns, significantly improving convergence efficiency. MicroT's SSKD is an unsupervised technique itself, providing robust feature representations. The edge-side implementation could further employ unsupervised learning. For example, replace traditional supervised classifiers with distance-based alternatives (*e.g.*, k-means or Gaussian mixture models [203]). This modification would allow end-to-end unsupervised operation - SSL extracts features from unlabeled edge data, which are then clustered by the distance-based classifier

Bibliography

[1] Megha Sharma, Abhinav Tomar, and Abhishek Hazra. Edge computing for industry 5.0: Fundamental, applications and research challenges. *IEEE Internet of Things Journal*, 2024.

[2] Linghe Kong, Jinlin Tan, Junqin Huang, Guihai Chen, Shuaitian Wang, Xi Jin, Peng Zeng, Muhammad Khan, and Sajal K Das. Edge-computing-driven internet of things: A survey. *ACM Computing Surveys*, 55(8):1–41, 2022.

[3] Tahmina Zebin, Patricia J Scully, Niels Peek, Alexander J Casson, and Krikor B Ozanyan. Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition. *IEEE Access*, 7:133509–133520, 2019.

[4] Noor Ul Huda, Ijaz Ahmed, Muhammad Adnan, Mansoor Ali, and Faisal Naeem. Experts and intelligent systems for smart homes' transformation to sustainable smart cities: A comprehensive review. *Expert Systems with Applications*, 238:122380, 2024.

[5] Mohammad Kamrul Hasan, Nusrat Jahan, Mohd Zakree Ahmad Nazri, Shayla Islam, Muhammad Attique Khan, Ahmed Ibrahim Alzahrani, Nasser Alalwan, and Yunyoung Nam. Federated learning for computational offloading and resource management of vehicular edge computing in 6g-v2x network. *IEEE Transactions on Consumer Electronics*, 70(1):3827–3847, 2024.

[6] Jinke Ren, Yinghui He, Guan Huang, Guanding Yu, Yunlong Cai, and Zhaoyang Zhang. An edge-computing based architecture for mobile augmented reality. *IEEE Network*, 33(4):162–169, 2019.

[7] Md Abdur Rahman and M Shamim Hossain. An internet-of-medical-things-enabled edge computing framework for tackling covid-19. *IEEE Internet of Things Journal*, 8(21):15847–15854, 2021.

[8] Marc Jayson Baucas and Petros Spachos. Edge-based data sensing and processing platform for urban noise classification. *IEEE Sensors Letters*, 2024.

[9] Lv Chen, Kang An, Dali Huang, Xiaoxian Wang, Min Xia, and Siliang Lu. Noise-boosted convolutional neural network for edge-based motor fault diagnosis with limited samples. *IEEE Transactions on Industrial Informatics*, 19(9):9491–9502, 2022.

[10] Xishun Liao, Zhouqiao Zhao, Matthew J Barth, Amr Abdelraouf, Rohit Gupta, Kyungtae Han, Jiaqi Ma, and Guoyuan Wu. A review of personalization in driving behavior: Dataset, modeling, and validation. *IEEE Transactions on Intelligent Vehicles*, 2024.

[11] Manuele Rusci, Francesco Paci, Marco Fariselli, Eric Flamand, and Tinne Tuytelaars. Self-learning for personalized keyword spotting on ultra-low-power audio sensors. *IEEE Internet of Things Journal*, 2024.

[12] Mahit Kumar Paul, Md Rabiul Islam, and AHM Sarowar Sattar. An efficient perturbation approach for multivariate data in sensitive and reliable data mining. *Journal of Information Security and Applications*, 62:102954, 2021.

[13] Matias Mendieta, Taojiannan Yang, Pu Wang, Minwoo Lee, Zhengming Ding, and Chen Chen. Local learning matters: Rethinking data heterogeneity in federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8397–8406, 2022.

[14] Mi Luo, Fei Chen, Dapeng Hu, Yifan Zhang, Jian Liang, and Jiashi Feng. No fear of heterogeneity: Classifier calibration for federated learning with non-iid data. *Advances in Neural Information Processing Systems*, 34:5972–5984, 2021.

[15] Pei-Chen Lo and Jenq-Shiun Leu. Adaptive baseline correction of meditation eeg. *American journal of electroneurodiagnostic technology*, 41(2):142–155, 2001.

[16] Yang Zheng and Xiaogang Hu. Interference removal from electromyography based on independent component analysis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(5):887–894, 2019.

[17] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. Unsupervised deep anomaly detection for multi-sensor time-series signals. *IEEE Transactions on Knowledge and Data Engineering*, 35(2):2118–2132, 2021.

[18] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. *Proceedings of Machine Learning and Systems*, 4:77–94, 2022.

[19] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2022.

[20] Beatriz Cardoso, Ignacio García-Bocanegra, Pelayo Acevedo, Germán Cáceres, Paulo C Alves, and Christian Gortázar. Stepping up from wildlife disease surveillance to integrated wildlife monitoring in europe. *Research in Veterinary Science*, 144:149–156, 2022.

[21] Suranga Seneviratne, Yining Hu, Tham Nguyen, Guohao Lan, Sara Khalifa, Kanchana Thilakarathna, Mahbub Hassan, and Aruna Seneviratne. A survey of wearable devices and challenges. *IEEE Communications Surveys & Tutorials*, 19(4):2573–2620, 2017.

[22] Xiaoqing Zheng, Song Zheng, Yaguang Kong, and Jie Chen. Recent advances in surface defect inspection of industrial products using deep learning techniques. *The International Journal of Advanced Manufacturing Technology*, 113:35–58, 2021.

[23] Maxim Integrated. Max78000. 2024. `https://www.analog.com/en`.

[24] UK Public General Acts. Data protection act 2018. 2025. `https://www.legislation.gov.uk/ukpga/2018/12/contents`.

[25] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[26] Ralf Bousseljot, Dieter Kreiseler, and Allard Schnabel. Nutzung der ekg-signaldatenbank cardiodat der ptb über das internet. 1995.

[27] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[28] Virender Kumar Mehla, Amit Singhal, Pushpendra Singh, and Ram Bilas Pachori. An efficient method for identification of epileptic seizures from eeg signals using fourier analysis. *Physical and Engineering Sciences in Medicine*, 44(2):443–456, 2021.

[29] Michael P Ward, Rachel M Iglesias, and Victoria J Brookes. Autoregressive models applied to time-series data in veterinary science. *Frontiers in veterinary science*, 7:604, 2020.

[30] Edmund Fosu Agyemang. A gaussian process regression and wavelet transform time series approaches to modeling influenza a. *Computers in Biology and Medicine*, 184:109367, 2025.

[31] Hojat Ghimatgar, Kamran Kazemi, Mohammad Sadegh Helfroush, and Ardalan Aarabi. An automatic single-channel eeg-based sleep stage scoring method based on hidden markov model. *Journal of neuroscience methods*, 324:108320, 2019.

[32] Wenhan Liu, Mengxin Zhang, Yidan Zhang, Yuan Liao, Qijun Huang, Sheng Chang, Hao Wang, and Jin He. Real-time multilead convolutional neural network for myocardial infarction detection. *IEEE journal of biomedical and health informatics*, 22(5):1434–1444, 2017.

[33] Jialing Wang, Shiwei Cheng, Jieming Tian, and Yuefan Gao. A 2d cnn-lstm hybrid algorithm using time series segments of eeg data for motor imagery classification. *Biomedical Signal Processing and Control*, 83:104627, 2023.

[34] Shifei Ding, Hong Zhu, Weikuan Jia, and Chunyang Su. A survey on feature extraction for pattern recognition. *Artificial Intelligence Review*, 37:169–180, 2012.

[35] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[36] Nurul Retno Nurwulan and Bernard C Jiang. Multiscale entropy for physical activity recognition. In *Proceedings of the 2020 2nd Asia Pacific Information Technology Conference*, pages 73–77, 2020.

[37] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7959–7971, 2022.

[38] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[39] Jiaming Pei, Wenxuan Liu, Jinhai Li, Lukun Wang, and Chao Liu. A review of federated learning methods in heterogeneous scenarios. *IEEE Transactions on Consumer Electronics*, 2024.

[40] Bing Luo, Xiang Li, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. Cost-effective federated learning design. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[41] Yi Xin, Junlong Du, Qiang Wang, Ke Yan, and Shouhong Ding. Mmap: Multi-modal alignment prompt for cross-domain multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16076–16084, 2024.

[42] Farhad Pourpanah, Moloud Abdar, Yuxuan Luo, Xinlei Zhou, Ran Wang, Chee Peng Lim, Xi-Zhao Wang, and QM Jonathan Wu. A review of generalized zero-shot learning methods. *IEEE transactions on pattern analysis and machine intelligence*, 45(4):4051–4070, 2022.

[43] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. On-device training under 256kb memory. *Advances in Neural Information Processing Systems*, 35:22941–22954, 2022.

[44] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems*, 33:11285–11297, 2020.

[45] Jiesong Liu, Feng Zhang, Jiawei Guan, Hsin-Hsuan Sung, Xiaoguang Guo, Xiaoyong Du, and Xipeng Shen. Space-efficient trec for enabling deep learning on microcontrollers. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 644–659, 2023.

[46] Yuxuan Zhang, Veysi Adin, Sebastian Bader, and Bengt Oelmann. Leveraging acoustic emission and machine learning for concrete materials damage classification on embedded devices. *IEEE Transactions on Instrumentation and Measurement*, 2023.

[47] Young D Kwon, Rui Li, Stylianos I Venieris, Jagmohan Chauhan, Nicholas D Lane, and Cecilia Mascolo. Tinytrain: Deep neural network training at the extreme edge. *arXiv preprint arXiv:2307.09988*, 4, 2023.

[48] Tingting Wang, Kai Fang, Wei Wei, Jinyu Tian, Yuanyuan Pan, and Jianqing Li. Microcontroller unit chip temperature fingerprint informed machine learning for iiot intrusion detection. *IEEE Transactions on Industrial Informatics*, 19(2):2219–2227, 2022.

[49] Ling Lin, Zhong Tang, Nianxiong Tan, and Xiaohui Xiao. Power management in low-power mcus for energy iot applications. *Journal of Sensors*, 2020(1):8819236, 2020.

[50] Michel Rottleuthner, Thomas C Schmidt, and Matthias Wählisch. Dynamic clock reconfiguration for the constrained iot and its application to energy-efficient networking. *arXiv preprint arXiv:2102.10353*, 2021.

[51] Chao Qian, Tianheng Ling, Christopher Cichiwskyj, and Gregor Schiele. Configuration-aware approaches for enhancing energy efficiency in fpga-based deep learning accelerators. *Journal of Systems Architecture*, 163:103410, 2025.

[52] Stefanos Laskaridis, Kleomenis Katevas, Lorenzo Minto, and Hamed Haddadi. Melting point: Mobile evaluation of language transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 890–907, 2024.

[53] Sudhakar Sah, Darshan C Ganji, Matteo Grimaldi, Ravish Kumar, Alexander Hoffman, Honnesh Rohmetra, and Ehsan Saboori. Mcubench: A benchmark of tiny object detectors on mcus. In *European Conference on Computer Vision*, pages 295–311. Springer, 2025.

[54] SiYoung Jang, Fahim Kawsar, and Chulhong Min. Thermal characterization of ai applications on ai accelerators-equipped microcontrollers. In *Proceedings of the Workshop on Body-Centric Computing Systems*, pages 17–22, 2024.

[55] Arthur Moss, Hyunjong Lee, Lei Xun, Chulhong Min, Fahim Kawsar, and Alessandro Montanari. Ultra-low power dnn accelerators for iot: Resource characterization of the max78000. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 934–940, 2022.

[56] Ali Dabbous, Luca Lazzaroni, Francesco Bellotti, Sara Muñoz Presentación, Alessandro Pighetti, and Riccardo Berta. Tinyml acceleration with max78000. In *Annual Meeting of the Italian Electronics Society*, pages 468–474. Springer, 2024.

[57] Luis R Soenksen, Yu Ma, Cynthia Zeng, Leonard Boussioux, Kimberly Villalobos Carballo, Liangyuan Na, Holly M Wiberg, Michael L Li, Ignacio Fuentes, and Dimitris Bertsimas. Integrated multimodal artificial intelligence framework for healthcare applications. *NPJ digital medicine*, 5(1):149, 2022.

[58] Flavio Di Martino and Franca Delmastro. Explainable ai for clinical and remote health applications: a survey on tabular and time series data. *Artificial Intelligence Review*, 56(6):5261–5315, 2023.

[59] Francesca Palermo, Yu Chen, Alexander Capstick, Nan Fletcher-Loyd, Chloe Walsh, Samaneh Kouchaki, Jessica True, Olga Balazikova, Eyal Soreq, Gregory Scott, et al. Tihm: An open dataset for remote healthcare monitoring in dementia. *Scientific data*, 10(1):606, 2023.

[60] Mohammad Amin Morid, Olivia R Liu Sheng, and Joseph Dunbar. Time series prediction using deep learning methods in healthcare. *ACM Transactions on Management Information Systems*, 14(1):1–29, 2023.

[61] Behrooz Zali-Vargahan, Asghar Charmin, Hashem Kalbkhani, and Saeed Barghandan. Deep time-frequency features and semi-supervised dimension reduction for subject-independent emotion recognition from multi-channel eeg signals. *Biomedical Signal Processing and Control*, 85:104806, 2023.

[62] Alexander Capstick, Francesca Palermo, Kimberley Zakka, Nan Fletcher-Lloyd, Chloe Walsh, Tianyu Cui, Samaneh Kouchaki, Raphaella Jackson, Martin Tran, Michael Crone, et al. Digital remote monitoring for screening and early detection of urinary tract infections. *NPJ Digital Medicine*, 7(1):11, 2024.

[63] Nur Shahirah Md Nor, Nurul Hashimah Ahamed Hassain Malim, Nur Aqilah Paskhal Rostam, J Joshua Thomas, Mohamad Azmeer Effendy, and Zurina Hassan. Automated classification of eight different electroencephalogram (eeg) bands using hybrid of fast fourier transform (fft) with machine learning methods. *Neuroscience Research Notes*, 5(1):116–116, 2022.

[64] Jonas F Haderlein, Andre DH Peterson, Anthony N Burkitt, Iven MY Mareels, and David B Grayden. Autoregressive models for biomedical signal processing. In *2023 45th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 1–6. IEEE, 2023.

[65] Shixiao Xu, Zhihua Wang, Jutao Sun, Zhiqiang Zhang, Zhaoyun Wu, Tiezhao Yang, Gang Xue, and Chuance Cheng. Using a deep recurrent neural network with eeg signal to detect parkinson's disease. *Annals of translational medicine*, 8(14), 2020.

[66] Nicola Michielli, U Rajendra Acharya, and Filippo Molinari. Cascaded lstm recurrent neural network for automated sleep stage classification using single-channel eeg signals. *Computers in biology and medicine*, 106:71–81, 2019.

[67] Snigdha Madhab Ghosh, Sharba Bandyopadhyay, and Debjani Mitra. Nonlinear classification of emotion from eeg signal based on maximized mutual information. *Expert Systems with Applications*, 185:115605, 2021.

[68] Palani Thanaraj Krishnan, Alex Noel Joseph Raj, Parvathavarthini Balasubramanian, and Yuanzhu Chen. Schizophrenia detection using multivariateempirical mode decomposition and entropy measures from multichannel eeg signal. *Biocybernetics and Biomedical Engineering*, 40(3):1124–1139, 2020.

[69] Joaquim Cerejeira, Luísa Lagarto, and Elizabeta Blagoja Mukaetova-Ladinska. Behavioral and psychological symptoms of dementia. *Frontiers in neurology*, 3:73, 2012.

[70] Jiska Cohen-Mansfield. Theoretical frameworks for behavioral problems in dementia. *Alzheimer's Care Today*, 1(4):8–21, 2000.

[71] Dirk De Bacquer, Guy De Backer, Marcel Kornitzer, and Henry Blackburn. Prognostic value of ecg findings for total, cardiovascular disease, and coronary heart disease death in men and women. *Heart*, 80(6):570–577, 1998.

[72] Reto Auer, Douglas C Bauer, Pedro Marques-Vidal, Javed Butler, Lauren J Min, Jacques Cornuz, Suzanne Satterfield, Anne B Newman, Eric Vittinghoff, Nicolas Rodondi, et al. Association of major and minor ecg abnormalities with coronary heart disease events. *Jama*, 307(14):1497–1505, 2012.

[73] Jeroen F Vranken, Rutger R van de Leur, Deepak K Gupta, Luis E Juarez Orozco, Rutger J Hassink, Pim van der Harst, Pieter A Doevendans, Sadaf Gulshad, and René van Es. Uncertainty estimation for deep learning-based automated analysis of 12-lead electrocardiograms. *European Heart Journal-Digital Health*, 2(3):401–415, 2021.

[74] Lei Lu, Tingting Zhu, Davide Morelli, Andrew Creagh, Zhangdaihong Liu, Jenny Yang, Fenglin Liu, Yuan-Ting Zhang, and David A Clifton. Uncertainties in the analysis of heart rate variability: a systematic review. *IEEE Reviews in Biomedical Engineering*, 17:180–196, 2023.

[75] Christopher G Goetz, Murat Emre, and Bruno Dubois. Parkinson's disease dementia: definitions, guidelines, and research perspectives in diagnosis. *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society*, 64(S2):S81–S92, 2008.

[76] Nina M Silverstein, Gerald Flaherty, and Terri Salmons Tobin. *Dementia and wandering behavior: Concern for the lost elder*. Springer Publishing Company, 2006.

[77] Tony Hope, Janet Keene, Rupert H McShane, Christopher G Fairburn, Kathy Gedling, and Robin Jacoby. Wandering in dementia: a longitudinal study. *International Psychogeriatrics*, 13(2):137–147, 2001.

[78] Zhenhu Liang, Yinghua Wang, Xue Sun, Duan Li, Logan J Voss, Jamie W Sleigh, Satoshi Hagihira, and Xiaoli Li. Eeg entropy measures in anesthesia. *Frontiers in computational neuroscience*, 9:16, 2015.

[79] Alessia Cacciotti, Chiara Pappalettera, Francesca Miraglia, Paolo Maria Rossini, and Fabrizio Vecchio. Eeg entropy insights in the context of physiological aging and alzheimer's and parkinson's diseases: A comprehensive review. *GeroScience*, 46(6):5537–5557, 2024.

[80] Cigdem Inan Akman. Nonconvulsive status epilepticus and continuous spike and slow wave of sleep in children. In *Seminars in Pediatric Neurology*, volume 17, pages 155–162. Elsevier, 2010.

[81] David E Krummen, Gregory K Feld, and Sanjiv M Narayan. Diagnostic accuracy of irregularly irregular rr intervals in separating atrial fibrillation from atrial flutter. *The American journal of cardiology*, 98(2):209–214, 2006.

[82] Pietro Barbiero, Gabriele Ciravegna, Francesco Giannini, Pietro Lió, Marco Gori, and Stefano Melacci. Entropy-based logic explanations of neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6046–6054, 2022.

[83] Wlodzimierz Klonowski. Fractal analysis of electroencephalographic time series (eeg signals). *The fractal geometry of the brain*, pages 413–429, 2016.

[84] Nicole L Molin, Clifford Molin, Rohan J Dalpatadu, and Ashok K Singh. Prediction of obstructive sleep apnea using fast fourier transform of overnight breath recordings. *Machine Learning with Applications*, 4:100022, 2021.

[85] Scott L Zeger, Rafael Irizarry, and Roger D Peng. On time series analysis of public health and biomedical data. *Annu. Rev. Public Health*, 27(1):57–79, 2006.

[86] Richard John M Buendia and Geoffrey A Solano. A disease outbreak detection system using autoregressive moving average in time series analysis. In *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–5. IEEE, 2015.

[87] Hojjat Adeli, Ziqin Zhou, and Nahid Dadmehr. Analysis of eeg records in an epileptic patient using wavelet transform. *Journal of neuroscience methods*, 123(1):69–87, 2003.

[88] Miguel Altuve and Nelson F Monroy. Hidden markov model-based heartbeat detector using electrocardiogram and arterial pressure signals. *Biomedical Engineering Letters*, 11(3):249–261, 2021.

[89] Zahra Zamanzadeh Darban, Geoffrey I Webb, Shirui Pan, Charu Aggarwal, and Mahsa Salehi. Deep learning for time series anomaly detection: A survey. *ACM Computing Surveys*, 57(1):1–42, 2024.

[90] Zhenyu Liu, Zhengtong Zhu, Jing Gao, and Cheng Xu. Forecast methods for time series data: a survey. *Ieee Access*, 9:91896–91912, 2021.

[91] Muhammad U Abbasi, Anum Rashad, Anas Basalamah, and Muhammad Tariq. Detection of epilepsy seizures in neo-natal eeg using lstm architecture. *IEEE access*, 7:179074–179085, 2019.

[92] Latifa Nabila Harfiya, Ching-Chun Chang, and Yung-Hui Li. Continuous blood pressure estimation using exclusively photopletysmography by lstm-based signal-to-signal translation. *Sensors*, 21(9):2952, 2021.

[93] Xiashuang Wang, Yinglei Wang, Dunwei Liu, Ying Wang, and Zhengjun Wang. Automated recognition of epilepsy from eeg signals using a combining space–time algorithm of cnn-lstm. *Scientific Reports*, 13(1):14876, 2023.

[94] Vignesh Shankar, Elnaz Yousefi, Alireza Manashty, Dayne Blair, and Deepika Teegapuram. Clinical-gan: trajectory forecasting of clinical events using transformer and generative adversarial networks. *Artificial Intelligence in Medicine*, 138:102507, 2023.

[95] GE Powell and IC Percival. A spectral entropy method for distinguishing regular and irregular motion of hamiltonian systems. *Journal of Physics A: Mathematical and General*, 12(11):2053, 1979.

[96] Joshua S Richman, Douglas E Lake, and J Randall Moorman. Sample entropy. In *Methods in enzymology*, volume 384, pages 172–184. Elsevier, 2004.

[97] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*, pages 1–17. Springer, 2004.

[98] DG247689 Lampard. A stochastic process whose successive intervals between events form a first order markov chain—i. *Journal of Applied Probability*, 5(3):648–668, 1968.

[99] Shirin Enshaeifar, Ahmed Zoha, Andreas Markides, Severin Skillman, Sahr Thomas Acton, Tarek Elsaleh, Masoud Hassanpour, Alireza Ahrabian, Mark Kenny, Stuart Klein, et al. Health management and pattern analysis of daily living activities of people with dementia using in-home sensors and machine learning techniques. *PloS one*, 13(5):e0195605, 2018.

[100] Dong-Kyum Kim, Youngkyoung Bae, Sangyun Lee, and Hawoong Jeong. Learning entropy production via neural networks. *Physical Review Letters*, 125(14):140604, 2020.

[101] Ingemar Bengtsson and Karol Życzkowski. *Geometry of quantum states: an introduction to quantum entanglement*. Cambridge university press, 2017.

[102] Karl Blum. *Density matrix theory and applications*, volume 64. Springer Science & Business Media, 2012.

[103] Cyrus Colton MacDuffee. *The theory of matrices*, volume 5. Springer Science & Business Media, 2012.

[104] Kai Mei, Meifang Tan, Zhihui Yang, and Shaoyue Shi. Modeling of feature selection based on random forest algorithm and pearson correlation coefficient. In *Journal of Physics: Conference Series*, volume 2219, page 012046. IOP Publishing, 2022.

[105] Yaqing Liu, Yong Mu, Keyu Chen, Yiming Li, and Jinghuan Guo. Daily activity feature selection in smart homes based on pearson correlation coefficient. *Neural processing letters*, 51:1771–1787, 2020.

[106] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907, 2001.

[107] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE international conference on healthcare informatics (ICHI)*, pages 443–444. IEEE, 2018.

[108] Shuting Sun, Huayu Chen, Gang Luo, Chang Yan, Qunxi Dong, Xuexiao Shao, Xiaowei Li, and Bin Hu. Clustering-fusion feature selection method in identifying major depressive disorder based on resting state eeg signals. *IEEE Journal of Biomedical and Health Informatics*, 27(7):3152–3163, 2023.

[109] Essam Abdellatef, Heba M Emara, Mohamed R Shoaib, Fatma E Ibrahim, Mohamed Elwekeil, Walid El-Shafai, Taha E Taha, Adel S El-Fishawy, El-Sayed M El-Rabaie, Ibrahim M Eldokany, et al. Automated diagnosis of eeg abnormalities with different classification techniques. *Medical & Biological Engineering & Computing*, 61(12):3363–3385, 2023.

[110] Parikha Chawla, Shashi B Rana, Hardeep Kaur, Kuldeep Singh, Rajamanickam Yuvaraj, and M Murugappan. A decision support system for automated diagnosis of parkinson's

disease from eeg using fawt and entropy features. *Biomedical Signal Processing and Control*, 79:104116, 2023.

[111] Gibson Chimamiwa, Marjan Alirezaie, Hadi Banaee, Uwe Köckemann, and Amy Loutfi. Towards habit recognition in smart homes for people with dementia. In *Ambient Intelligence: 15th European Conference, AmI 2019, Rome, Italy, November 13–15, 2019, Proceedings 15*, pages 363–369. Springer, 2019.

[112] Simin Khalilpour, Amin Ranjbar, Mohammad Bagher Menhaj, and Afshin Sandoogh-dar. Application of 1-d cnn to predict epileptic seizures using eeg records. In *2020 6th international conference on web research (ICWR)*, pages 314–318. IEEE, 2020.

[113] Leila Farsi, Siuly Siuly, Enamul Kabir, and Hua Wang. Classification of alcoholic eeg signals using a deep learning method. *IEEE Sensors Journal*, 21(3):3552–3560, 2020.

[114] Waqar Hussain, Muhammad Tariq Sadiq, Siuly Siuly, and Ateeq Ur Rehman. Epileptic seizure detection using 1 d-convolutional long short-term memory neural networks. *Applied Acoustics*, 177:107941, 2021.

[115] Ladislav Volicer, David G Harper, Barbara C Manning, Rachel Goldstein, and Andrew Satlin. Sundowning and circadian rhythms in alzheimer's disease. *American Journal of Psychiatry*, 158(5):704–711, 2001.

[116] S Janifer Jabin Jui, Ravinesh C Deo, Prabal Datta Barua, Aruna Devi, Jeffrey Soar, and U Rajendra Acharya. Application of entropy for automated detection of neurological disorders with electroencephalogram signals: a review of the last decade (2012-2022). *IEEE Access*, 2023.

[117] Weishu Li, Yuxiu Zhao, Qi Wang, and Jian Zhou. Twenty years of entropy research: A bibliometric overview. *Entropy*, 21(7):694, 2019.

[118] Prashant G Mehta, Umesh Vaidya, and Andrzej Banaszuk. Markov chains, entropy, and fundamental limitations in nonlinear stabilization. *IEEE Transactions on Automatic Control*, 53(3):784–791, 2008.

[119] Yiqi Lin and Xinyuan Song. Order selection for regression-based hidden markov model. *Journal of Multivariate Analysis*, 192:105061, 2022.

[120] Maria Papapetrou and Dimitris Kugiumtzis. Markov chain order estimation with conditional mutual information. *Physica A: Statistical Mechanics and its Applications*, 392(7):1593–1601, 2013.

[121] Alexandra MV Wennberg, Mark N Wu, Paul B Rosenberg, and Adam P Spira. Sleep disturbance, cognitive decline, and dementia: a review. In *Seminars in neurology*, volume 37, pages 395–406. Thieme Medical Publishers, 2017.

[122] Paul B Bernard, Anna M Castano, Christy S Beitzel, Vivian B Carlson, and Tim A Benke. Behavioral changes following a single episode of early-life seizures support the latent development of an autistic phenotype. *Epilepsy & Behavior*, 44:78–85, 2015.

[123] Ozlem Durmaz Incel and Sevda Özge Bursa. On-device deep learning for mobile and wearable sensing applications: A review. *IEEE Sensors Journal*, 23(6):5501–5512, 2023.

[124] Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things*, 2(3):1–49, 2021.

[125] Sayed Saad Afzal, Waleed Akbar, Osvy Rodriguez, Mario Doumet, Unsoo Ha, Reza Ghaffarivardavagh, and Fadel Adib. Battery-free wireless imaging of underwater environments. *Nature communications*, 13(1):5546, 2022.

[126] Yuchen Zhao, Sayed Saad Afzal, Waleed Akbar, Osvy Rodriguez, Fan Mo, David Boyle, Fadel Adib, and Hamed Haddadi. Towards battery-free machine learning and inference in underwater environments. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*, pages 29–34, 2022.

[127] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

[128] Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. *Advances in Neural Information Processing Systems*, 34:22405–22418, 2021.

[129] Nattaya Mairittha, Tittaya Mairittha, and Sozo Inoue. Improving activity data collection with on-device personalization using fine-tuning. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pages 255–260, 2020.

[130] Hong-You Chen, Yandong Li, Yin Cui, Mingda Zhang, Wei-Lun Chao, and Li Zhang. Train-once-for-all personalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11818–11827, 2023.

[131] Zhixin Lai, Xuesheng Zhang, and Suiyao Chen. Adaptive ensembles of fine-tuned transformers for llm-generated text detection. *arXiv preprint arXiv:2403.13335*, 2024.

[132] Elan Rosenfeld, Pradeep Kumar Ravikumar, and Andrej Risteski. Domain-adjusted regression or: Erm may already learn features sufficient for out-of-distribution generalization. In *NeurIPS 2022 Workshop on Distribution Shifts: Connecting Methods and Applications*.

[133] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations. In *The Eleventh International Conference on Learning Representations*.

[134] Zhi Wang, Wei Bi, Yan Wang, and Xiaojiang Liu. Better fine-tuning via instance weighting for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7241–7248, 2019.

[135] Yushan Huang, Ranya Aloufi, Xavier Cadet, Yuchen Zhao, Payam Barnaghi, and Hamed Haddadi. Low-energy on-device personalization for mcus. In *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 45–58. IEEE, 2024.

[136] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[137] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[138] Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

[139] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. *arXiv preprint arXiv:2210.11466*, 2022.

[140] Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. *arXiv preprint arXiv:1206.6471*, 2012.

[141] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[142] Adriana Sayuri Iwashita and Joao Paulo Papa. An overview on concept drift learning. *IEEE access*, 7:1532–1547, 2018.

[143] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. Breeds: Benchmarks for subpopulation shift. *arXiv preprint arXiv:2008.04859*, 2020.

[144] David S Klimstra, Irvin R Modlin, Domenico Coppola, Ricardo V Lloyd, and Saul Suster. The pathologic classification of neuroendocrine tumors: a review of nomenclature, grading, and staging systems. *Pancreas*, 39(6):707–712, 2010.

[145] Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, 50:723–762, 2014.

[146] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[147] Jaejun Lee, Raphael Tang, and Jimmy Lin. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*, 2019.

[148] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4805–4814, 2019.

[149] Vinay V Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*, 2020.

[150] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. Autofreeze: Automatically freezing model blocks to accelerate fine-tuning. *arXiv preprint arXiv:2102.01386*, 2021.

[151] Amélie Royer and Christoph Lampert. A flexible selection scheme for minimum-effort transfer learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2191–2200, 2020.

[152] Cian Eastwood, Ian Mason, and Christopher KI Williams. Unit-level surprise in neural networks. In *I (Still) Can't Believe It's Not Better! Workshop at NeurIPS 2021*, pages 33–40. PMLR, 2022.

[153] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

[154] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *Journal of Machine Learning Research*, 23(67):1–28, 2022.

[155] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations. *arXiv preprint arXiv:2204.02937*, 2022.

[156] Lachit Dutta and Swapna Bharali. Tinyml meets iot: A comprehensive survey. *Internet of Things*, 16:100461, 2021.

[157] Ramon Sanchez-Iborra and Antonio F Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine*, 20(3):4–18, 2020.

[158] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5687–5695, 2017.

[159] Chengyue Gong, Zixuan Jiang, Dilin Wang, Yibo Lin, Qiang Liu, and David Z Pan. Mixed precision neural architecture search for energy efficient deep learning. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2019.

[160] Daniel T Speckhard, Karolis Misiunas, Sagi Perel, Tenghui Zhu, Simon Carlile, and Malcolm Slaney. Neural architecture search for energy efficient always-on audio models. *arXiv preprint arXiv:2202.05397*, 2022.

[161] Seunghyeok Jeon, Yonghun Choi, Yeonwoo Cho, and Hojung Cha. Harvnet: resource-optimized operation of multi-exit deep neural networks on energy harvesting devices. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, pages 42–55, 2023.

[162] Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. *Advances in neural information processing systems*, 35:38629–38642, 2022.

[163] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

[164] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[165] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[166] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019.

[167] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[168] Mostafa Haghi, Kerstin Thurow, and Regina Stoll. Wearable devices in medical internet of things: scientific research and commercially available devices. *Healthcare informatics research*, 23(1):4, 2017.

[169] Youssef Abadade, Anas Temouden, Hatim Bamoumen, Nabil Benamar, Yousra Chtouki, and Abdelhakim Senhaji Hafid. A comprehensive survey on tinyml. *IEEE Access*, 2023.

[170] Ameeth Kanawaday and Aditya Sane. Machine learning for predictive maintenance of industrial machines using iot sensor data. In *2017 8th IEEE international conference on software engineering and service science (ICSESS)*, pages 87–90. IEEE, 2017.

[171] Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *arXiv preprint arXiv:2303.15361*, 2023.

[172] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[173] Ronghang Hu and Amanpreet Singh. Unit: Multimodal multitask learning with a unified transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1439–1449, 2021.

[174] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[175] Jiangshan Hao, Yanchao Zhao, and Jiale Zhang. Time efficient federated learning with semi-asynchronous communication. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 156–163. IEEE, 2020.

[176] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2021.

[177] Dmitri Goldenberg, Kostia Kofman, Javier Albert, Sarai Mizrachi, Adam Horowitz, and Irene Teinemaa. Personalization in practice: Methods and applications. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 1123–1126, 2021.

[178] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. Minilearn: On-device learning for low-power iot devices. In *EWSN*, pages 1–11, 2022.

[179] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33, 2020.

[180] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 450–463, 2022.

[181] Dongqi Cai, Qipeng Wang, Yuanqiang Liu, Yunxin Liu, Shangguang Wang, and Mengwei Xu. Towards ubiquitous learning: A first measurement of on-device training performance. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pages 31–36, 2021.

[182] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. Mandheling: Mixed-precision on-device dnn training with dsp offloading. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 214–227, 2022.

[183] Shekoofeh Azizi, Basil Mustafa, Fiona Ryan, Zachary Beaver, Jan Freyberg, Jonathan Deaton, Aaron Loh, Alan Karthikesalingam, Simon Kornblith, Ting Chen, et al. Big self-supervised models advance medical image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3478–3488, 2021.

[184] Xingyu Chen, Yufeng Liu, Yajiao Dong, Xiong Zhang, Chongyang Ma, Yanmin Xiong, Yuan Zhang, and Xiaoyan Guo. Mobrecon: Mobile-friendly hand mesh reconstruction from monocular image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20544–20554, 2022.

[185] Yinpeng Chen, Xiyang Dai, Dongdong Chen, Mengchen Liu, Xiaoyi Dong, Lu Yuan, and Zicheng Liu. Mobile-former: Bridging mobilenet and transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5270–5279, 2022.

[186] Philipp Mayer, Michele Magno, and Luca Benini. Self-sustaining acoustic sensor with programmable pattern recognition for underwater monitoring. *IEEE transactions on instrumentation and measurement*, 68(7):2346–2355, 2019.

[187] Zijie Chen, Yiming Gao, and Junrui Liang. Lopdm: A low-power on-device predictive maintenance system based on self-powered sensing and tinyml. *IEEE Transactions on Instrumentation and Measurement*, 2023.

[188] Yushan Huang, Yuchen Zhao, Alexander Capstick, Francesca Palermo, Hamed Haddadi, and Payam Barnaghi. Analyzing entropy features in time-series data for pattern recognition in neurological conditions. *Artificial Intelligence in Medicine*, page 102821, 2024.

[189] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, 2021.

[190] Ahmed Agiza, Marina Neseem, and Sherief Reda. Mtlora: Low-rank adaptation approach for efficient multi-task learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16196–16205, 2024.

[191] Dwarikanath Mahapatra, Behzad Bozorgtabar, and Zongyuan Ge. Medical image classification using generalized zero shot learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3344–3353, 2021.

[192] Xi Chen, Lianghua Huang, Yu Liu, Yujun Shen, Deli Zhao, and Hengshuang Zhao. Anydoor: Zero-shot object-level image customization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6593–6602, 2024.

[193] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[194] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[195] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[196] Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. *arXiv preprint arXiv:2005.04275*, 2020.

[197] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

[198] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. Tabi: An efficient multi-level inference system for large language models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 233–248, 2023.

[199] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.

[200] Jacob Peplinski, Joel Shor, Sachin Joglekar, Jake Garrison, and Shwetak Patel. Frill: A non-semantic speech embedding for mobile devices. *arXiv preprint arXiv:2011.04609*, 2020.

[201] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[202] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

[203] Hao Wu, Jinghao Feng, Xuejin Tian, Edward Sun, Yunxin Liu, Bo Dong, Fengyuan Xu, and Sheng Zhong. Emo: Real-time emotion recognition from single-eye images for resource-constrained eyewear devices. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 448–461, 2020.

[204] STMicroelectronics. X-cube-ai. 2023.

[205] Monsoon Solutions Inc. Monsoon high voltage power monitor. 2024. `https://www.msoon.com/`.

[206] Tuochao Chen, Justin Chan, and Shyamnath Gollakota. Underwater messaging using mobile devices. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 545–559, 2022.

[207] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

[208] Sulaiman Sadiq, Jonathon Hare, Simon Craske, Partha Maji, and Geoff Merrett. Enabling imagenet-scale deep learning on mcus for accurate and efficient inference. *IEEE Internet of Things Journal*, 2023.

[209] Zhenhong Sun, Ce Ge, Junyan Wang, Ming Lin, Hesen Chen, Hao Li, and Xiuyu Sun. Entropy-driven mixed-precision quantization for deep network design. *Advances in Neural Information Processing Systems*, 35:21508–21520, 2022.

[210] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. The oxford-iiit pet dataset.

[211] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[212] Herve Goeau, Pierre Bonnet, and Alexis Joly. Plant identification based on noisy web data: the amazing performance of deep learning (lifeclef 2017). CEUR Workshop Proceedings, 2017.

[213] Weitao Wan, Yuanyi Zhong, Tianpeng Li, and Jiansheng Chen. Rethinking feature distribution for loss functions in image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9117–9126, 2018.

[214] Jacob Zavatone-Veth and Cengiz Pehlevan. Learning curves for deep structured gaussian feature models. *Advances in Neural Information Processing Systems*, 36:42866–42897, 2023.

[215] STMicroelectronics. Stm32cubeide. 2023.

[216] Emma E Hart, Julian Fennessy, Henrik B Rasmussen, Michael Butler-Brown, Arthur B Muneza, and Simone Ciuti. Precision and performance of an 180g solar-powered gps device for tracking medium to large-bodied terrestrial mammals. *Wildlife Biology*, 2020(3):1–8, 2020.

[217] Octarina Nur Samijayani, Tyan Permana Saputra, Hamzah Firdaus, and Anwar Mujadin. Solar powered wireless sensor network for water quality monitoring and classification. *Green Intelligent Systems and Applications*, 3(1):14–21, 2023.

[218] Dongjie Jiang, Bojing Shi, Han Ouyang, Yubo Fan, Zhong Lin Wang, and Zhou Li. Emerging implantable energy harvesters and self-powered implantable medical electronics. *ACS nano*, 14(6):6436–6448, 2020.

[219] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[220] I Fang, Hsiao-Chieh Wen, Chia-Lun Hsu, Po-Chung Jen, Ping-Yang Chen, Yong-Sheng Chen, et al. Es3net: accurate and efficient edge-based self-supervised stereo matching network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4472–4481, 2023.

[221] Jesse J Hagenaars, Yilun Wu, Federico Paredes-Vallés, Stein Stroobants, and Guido CHE de Croon. On-device self-supervised learning of low-latency monocular depth from only events. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 17114–17123, 2025.

[222] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. Ml-mcu: A framework to train ml classifiers on mcu-based iot edge devices. *IEEE Internet of Things Journal*, 9(16):15007–15017, 2021.

[223] GP Rimoli, B Boi, P Fusco, C Esposito, and M Ficco. Tiny federated learning with blockchain for privacy and security preservation of mcu-based iot applications. In *2024 6th International Conference on Blockchain Computing and Applications (BCCA)*, pages 124–131. IEEE, 2024.

[224] Yen-Chi Chen. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1(1):161–187, 2017.

[225] Jeffrey S Simonoff. *Smoothing methods in statistics*. Springer Science & Business Media, 2012.

[226] ARM. Ethos-u65. 2024. `https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u65`.

[227] Greenwaves. Ultra low power gap processors. 2024. `https://greenwaves-technologies.com/gap9_processor/`.

[228] Taesik Gong, Si Young Jang, Utku Günay Acer, Fahim Kawsar, and Chulhong Min. Synergy: Towards on-body ai via tiny ai accelerator collaboration on wearables. *arXiv preprint arXiv:2401.08637*, 2023.

[229] Lik-Hang Lee and Pan Hui. Interaction methods for smart glasses: A survey. *IEEE access*, 6:28712–28732, 2018.

[230] Mattia Antonini, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. Resource characterisation of personal-scale sensing models on edge accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pages 49–55, 2019.

[231] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 international workshop on internet of things towards applications*, pages 7–12, 2015.

[232] Ferran Reverter and Manel Gasulla. Experimental characterization of the energy consumption of adc embedded into microcontrollers operating in low power. In *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–5. IEEE, 2019.

[233] Uros Pesovic, Zeljko Jovanovic, Sinisa Randjic, and Dusan Markovic. Benchmarking performance and energy efficiency of microprocessors for wireless sensor network applications. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 743–747. IEEE, 2012.

[234] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[235] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[236] Josh Millar, Yushan Huang, Sarab Sethi, Hamed Haddadi, and Anil Madhavapeddy. Benchmarking ultra-low-power unpus. *arXiv preprint arXiv:2503.22567*, 2025.

[237] Zhenying Cheng, Xin Liu, Yingjun Lei, Qiaosheng Pan, and Ruijun Li. Development of a high-precision optical force sensor with μn-level resolution. *IEEE Sensors Journal*, 22(13):12738–12745, 2022.

[238] YanJin Zhao, NingFang Song, Fuyu Gao, XiaoBin Xu, JiaQi Liu, and Chenchen Liu. High-precision photonic crystal fiber-based pressure sensor with low-temperature sensitivity. *Optics Express*, 29(20):32453–32463, 2021.

[239] Yu Chen, Ren-Fa Li, and Qiang Wu. Automatic reconfigurable system-on-chip design with run-time hardware/software partitioning. In *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 484–491. IEEE, 2009.

[240] Denghuang Zhao, Zhixin Qiu, Yujie Jiang, Xincheng Zhu, Xiaojun Zhang, and Zhi Tao. A depthwise separable cnn-based interpretable feature extraction network for automatic pathological voice detection. *Biomedical Signal Processing and Control*, 88:105624, 2024.

[241] Gianmarco Cerutti, Lukas Cavigelli, Renzo Andri, Michele Magno, Elisabetta Farella, and Luca Benini. Sub-mw keyword spotting on an mcu: Analog binary feature extraction and binary neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(5):2002–2012, 2022.

[242] Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

[243] Renyuan Liu, Yuyang Leng, Shilei Tian, Shaohan Hu, Chun-Fu Chen, and Shuochao Yao. Dynaspa: Exploiting spatial sparsity for efficient dynamic dnn inference on devices. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 422–435, 2024.

[244] Puhan Luo, Jiahui Hou, Mu Yuan, Guangyu Wu, Yunhao Yao, and Xiang-Yang Li. F2zip: Finetuning-free model compression for scenario-adaptive embedded vision. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 15–27, 2024.

[245] Huming Qiu, Hua Ma, Zhi Zhang, Yansong Gao, Yifeng Zheng, Anmin Fu, Pan Zhou, Derek Abbott, and Said F Al-Sarawi. Rbnn: memory-efficient reconfigurable deep binary neural network with ip protection for internet of things. *IEEE transactions on computer-aided design of integrated circuits and systems*, 42(4):1185–1198, 2022.

[246] Taesik Gong, Fahim Kawsar, and Chulhong Min. Dex: Data channel extension for efficient cnn inference on tiny ai accelerators. *Advances in Neural Information Processing Systems*, 37:43925–43951, 2024.

[247] Konstantin Hemker, Nikola Simidjievski, and Mateja Jamnik. Healnet: Multimodal fusion for heterogeneous biomedical data. *Advances in Neural Information Processing Systems*, 37:64479–64498, 2024.

[248] Yasin Yilmaz, Mehmet Aktukmak, and Alfred O Hero. Multimodal data fusion in high-dimensional heterogeneous datasets via generative models. *IEEE Transactions on Signal Processing*, 69:5175–5188, 2021.

[249] Yifan Liu, Kangning Zhang, Xiangyuan Ren, Yanhua Huang, Jiarui Jin, Yingjie Qin, Ruilong Su, Ruiwen Xu, Yong Yu, and Weinan Zhang. Alignrec: Aligning and training in multimodal recommendations. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1503–1512, 2024.

[250] Min Tan, Yinfu Feng, Lingqiang Chu, Jingcheng Shi, Rong Xiao, Haihong Tang, and Jun Yu. Fedsea: Federated learning via selective feature alignment for non-iid multimodal data. *IEEE Transactions on Multimedia*, 26:5807–5822, 2023.

[251] Kamal Kant Verma, Brij Mohan Singh, and Amit Dixit. A review of supervised and unsupervised machine learning techniques for suspicious behavior recognition in intelligent surveillance system. *International Journal of Information Technology*, 14(1):397–410, 2022.

[252] Abdalhamid Rahoma, Syed Imtiaz, Salim Ahmed, and Faisal Khan. Detection and diagnosis of process fault using unsupervised learning methods and unlabeled data. *International journal of advances in engineering sciences and applied mathematics*, 15(1):24–36, 2023.