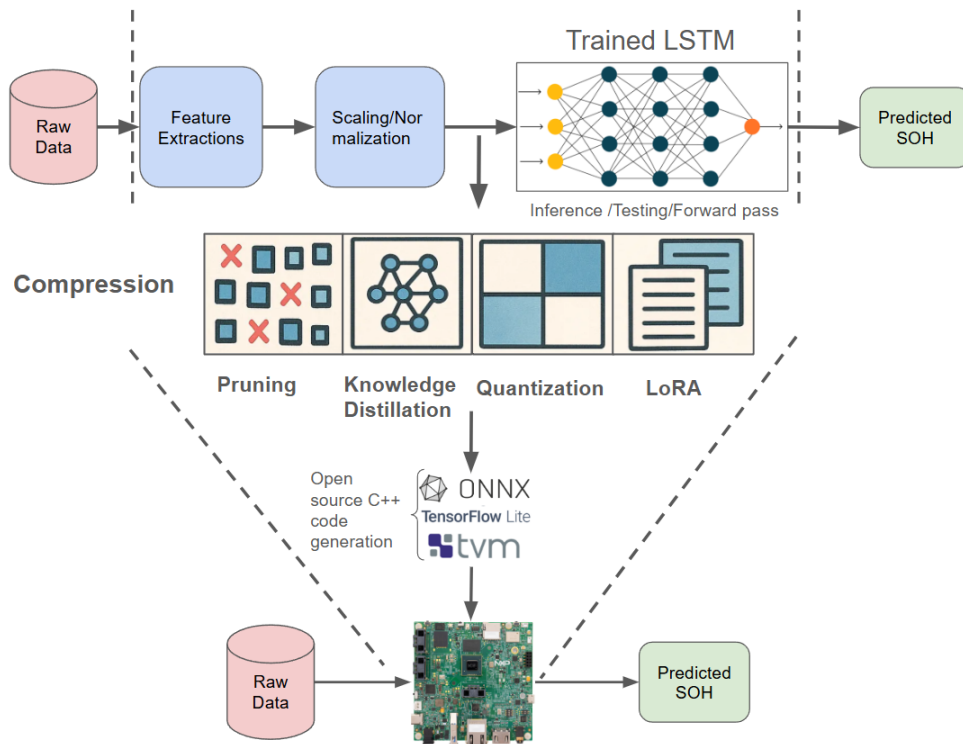




Edge-Optimized Deep Learning for Real-time state of Estimation of Lithium-Ion Batteries Embedded Systems

Master's thesis in Embedded Electronic System Design and System, Controls and Mechatronics

Krishna Hanumaraddi and Xuqi fu

MASTER'S THESIS 2026

Edge-Optimized Deep Learning for Real-Time State of Health Estimation of Lithium-Ion Batteries in Embedded Systems

A Lightweight Data-Driven Approach for Battery Health
Monitoring on Resource-Constrained Embedded Platforms.

Krishna Hanumaraddi and Xuqi fu



Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Edge-Optimized Deep Learning for Real-Time State of Health Estimation of Lithium-Ion Batteries in Embedded Systems
Krishna Hanumaraddi and Xuqi fu

© Krishna Hanumaraddi and Xuqi fu, 2026.

University Supervisor: Torsten Wik, Chalmers
Company Supervisor: Christian, Cognivity AI
Examiner: Torsten Wik, Chalmers

Master's Thesis 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of a deep LSTM does state of health estimation and undergoes compression/optimization and Hardware deployment .

Typeset in L^AT_EX
Gothenburg, Sweden 2026

Edge-Optimized Deep Learning for Real-Time State of Health Estimation of Lithium-Ion Batteries in Embedded Systems.

Krishna Hanumaraddi and Xuqi fu
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Accurate State of Health (SoH) estimation is critical for lithium-ion battery management. While deep learning models like Long Short-Term Memory (LSTM) networks offer superior predictive capabilities, their computational complexity and memory footprints prohibit direct deployment on resource-constrained Battery Management System (BMS) microcontrollers. This thesis addresses this bottleneck by proposing an edge-optimized deep learning framework for real-time SoH estimation. Utilizing features extracted from early-cycle operational data (the first 10 cycles), we develop a Physics-Informed Long Short-Term Memory (PINN-LSTM) architecture as the full-precision teacher model. To achieve embedded compatibility, we introduce a model compression pipeline integrating Physics-Informed Neural Network Knowledge Distillation (PINN-KD), structural pruning, and W8A32 dynamic quantization, where the knowledge of the PINN-LSTM teacher is transferred to a lightweight LSTM student model. The optimization workflow compiles the models from PyTorch into bare-metal C applications using the Apache TVM stack and Relax Virtual Machine, enabling static memory allocation and Flash-aligned weight storage. Software-in-the-Loop (SIL) on Infineon TC4D7 and Processor-in-the-Loop (PIL) evaluations on NXP MCXN947 microcontrollers validate the system’s real-time performance and concurrent asynchronous data handling. Results demonstrate a massive memory reduction, with Flash storage shrinking from 127.13 KB (full-precision PINN-LSTM teacher) to merely 1.25 KB (quantized LSTM student). Crucially, the W8A32 quantized student model maintains exceptional mathematical fidelity, exhibiting a Mean Absolute Error (MAE) of 3.32×10^{-8} against the corresponding full-precision LSTM student model. Finally, the study identifies gradient dominance as a fundamental challenge in developing chemistry-agnostic models across mixed LFP, NMC, and NCA datasets, highlighting avenues for future domain-adaptation research.

Keywords: Lithium-ion Batteries, State of Health (SoH), Deep Learning, Edge Computing, Model Compression, Performance profiling, Bare-Metal Deployment, Apache TVM.

Acknowledgements

We would like to express our sincere gratitude to Torsten Wik, our university supervisor at Chalmers University of Technology, for his valuable guidance, constructive feedback, and continuous support throughout this thesis. His academic expertise and insightful feedback have been instrumental in shaping and improving this work.

We would also like to extend our sincere appreciation to Christian, our company supervisor at Cognivity AI, for his guidance, encouragement, and support throughout the project. His expertise and valuable discussions have greatly contributed to the progress and development of this thesis.

We are especially grateful to Anna, Enderson, and Nlandi at Cognivity AI for their technical support, helpful discussions, and willingness to share their knowledge throughout the project. Their assistance and valuable insights have been greatly appreciated, particularly during the implementation and development of the proposed work.

We would also like to thank Chalmers University of Technology and Cognivity AI for providing the academic and professional environment, resources, and opportunities that made it possible to carry out this thesis.

Finally, we would like to express our heartfelt gratitude to our families and friends for their constant encouragement, patience, and support throughout this journey. Their understanding and motivation have been invaluable during the challenges and successes of this work.

Krishna Hanumaraddi and Xuqi Fu, Gothenburg, October 2026

Declaration of Generative AI

Generative AI tools were used to improve the grammar, vocabulary, clarity, and overall phrasing of the written text, including paraphrasing and rephrasing. They were not used to generate the original academic content, arguments, or conclusions of this thesis. The tools were also used to suggest code improvements, support documentation in this thesis. All generated outputs were reviewed, verified, and adapted by the authors, who take full responsibility for the final content of the thesis.

Krishna Hanumaraddi and Xuqi Fu, Gothenburg, August 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ALUs	Arithmetic Logic Units
AMC	Automatic Model Compression
AOT	Ahead-Of-Time
BMS	Battery Management System
CAT	Compression-Aware Training
CC	Constant Current
CEI	Cathode Electrolyte Interphase
CNN	Convolutional Neural Network
CV	Constant Voltage
DL	Deep Learning
ECM	Equivalent Circuit Model
EIS	Electrochemical Impedance Spectroscopy
EKF	Extended Kalman Filter
ELM	Extreme Learning Machine
EOL	End of Life
FLOPS	Floating-Point Operations Per Second
FP32	32-bit Floating-Point
GP	Grow-and-Prune
INT8	8-bit Integer
KD	Knowledge Distillation
LAM	Loss of Active Material
LE	Lithium Electroplating
LFP	Lithium Iron Phosphate (LiFePO_4)
LLI	Loss of Lithium Inventory
LSTM	Long Short-Term Memory
MACs	Multiply-Accumulate operations
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
NAS	Neural Architecture Search
NCA	Nickel Cobalt Aluminum
NMC	Nickel Manganese Cobalt
OPS	Operations Per Second
P2D (or P@D)	Pseudo-Two-Dimensional

PDE	Partial Differential Equation
PIL	Processor-in-the-Loop
PINN	Physics-Informed Neural Network
QAT	Quantization-Aware Training
R^2	Coefficient of Determination
RMSE	Root Mean Square Error
RSA	Representational Similarity Analysis
RTOS	Real-Time Operating System
RUL	Remaining Useful Life
SEI	Solid Electrolyte Interphase
SIL	Software-in-the-Loop
SOC	State of Charge
SoH	State of Health
SPM	Single Particle Model
SRAM	Static Random-Access Memory
SVD	Singular Value Decomposition
TFLM	TensorFlow Lite for Microcontrollers
t-SNE	t-Distributed Stochastic Neighbor Embedding

Nomenclature

Below is the list of symbols used throughout this thesis, categorized by type:

Indices and Sets

E	Number of epochs
i, j, n	General discrete sequence indices
K	Number of folds in Stratified Group K-Fold cross-validation
K'	Set of indices for surviving hidden units after pruning
k	Index for kernels, hidden units, or window size parameter
l	Layer index in a neural network (1 to L)
M	Number of battery sequences evaluated
m, n, k	Matrix dimensions in Matrix-Vector (MV)
N	Number of samples in a dataset or batch
P	Set of non-overlapping random cycle pairs within a mini-batch
t	Time step or cycle index

Parameters and Constants

$\alpha, \lambda_{response}$	Penalty weighting coefficients balancing the respective loss components
b	Target bit-width (e.g., 8-bit precision)
b_f, b_i, b_c, b_o	Bias vectors for the LSTM gates
$Bandwidth_{peak}$	Maximum memory bus throughput
C_j	Convolution coefficients for the Savitzky-Golay filter
C_{new}	New (initial) capacity
$C_{nominal}$	Nominal capacity
η	Learning rate during backpropagation
$\lambda_{dist}, \lambda_{cos}$	Penalty weighting coefficients balancing the respective loss components
P_{peak}	Peak processor compute capability limit
R_{EOL}	End of Life resistance
R_{new}	New (initial) resistance

s	Symmetric scaling factor for weight quantization
τ	Pruning threshold (for hidden units) or singular value penalty (in SVD)
W, b	Weight matrix and bias vector
$W_{ih}^{(g)}, W_{hh}^{(g)}$	Input-to-hidden and hidden-to-hidden weights for an LSTM gate g
$(W_{ih}^{(g)})', (W_{hh}^{(g)})'$	Pruned input-to-hidden and hidden-to-hidden weights
W_f, W_i, W_c, W_o	Weight matrices for the LSTM gates
W_{linear}, W'_{linear}	Original and pruned weight matrix of the subsequent linear layer
W_{new}	Updated weight matrix after backward pass step

Variables and Metrics

$C_{current}$	Current capacity
$c_t, \tilde{c}_t$	Cell state and candidate cell state at time step t
$D_B^T(i, j), D_B^S(i, j)$	Pairwise squared Euclidean distances for Teacher and Student within a batch
ΔI	Mean Decrease in Impurity (MDI)
E_{window}	Window Energy (integral of absolute power over a window)
f_t, i_t, o_t	Forget, input, and output gates in an LSTM cell at time step t
H	Physics-informed constraint (residual error of the PDE)
h_k^l	Feature map for the k -th kernel in the l -th layer
$h_{S,i}$	Feature representations of the i -th sample by the Student (S)
$h_{T,i}$	Feature representations of the i -th sample by the Teacher (T)
h_t	Hidden state at time step t
I	Arithmetic Intensity (Operations per Byte of memory traffic)
I_{min}, I_{max}	Minimum and maximum current in a sampled window
I_k	Importance score of the k -th hidden unit based on L_2 -norm
L_{dist}, L_{cos}	Distance-preserving loss and Cosine similarity-preserving loss
L_{ground_truth}	Empirical ground truth loss
L_{hard}	Hard label loss (Mean Squared Error against true SoH target)
L_{PDE}, L_{mono}	Physics-informed PDE loss and Monotonicity constraint loss
$L_{response}$	Empirical Student-Teacher soft response loss
L_{soft}	Soft label loss (Mean Squared Error against Teacher predictions)

L_{total}	Total Knowledge Distillation composite loss function
λ	Eigenvalue of the Gram matrix
M	Gram matrix ($W^T W$)
MAE	Mean Absolute Error
MIC	Maximal Information Coefficient
MSE	Mean Squared Error
m_k	Binary masking variable for targeted unit retention
$\nabla W, dW$	Gradient of the weight matrix
P	Roofline attainable performance
Q, K, V	Queries, Keys, and Values matrices in Attention mechanism
Q_{window}	Window Capacity (integral of absolute current over a window)
q	Quantized discrete integer weights
R^2	Coefficient of Determination
$R_{current}$	Current resistance
$RMSE$	Root Mean Square Error
r_{xy}	Pearson correlation coefficient
$S_B^T(i, j), S_B^S(i, j)$	Pairwise cosine similarity matrices for Teacher and Student within a batch
$Size(Activation, l)$	Size of input/output activation tensors at layer l
$SoH_{C, scaled}$	Scaled State of Health based on capacity
SoH_R	State of Health based on internal resistance
$SRAM_{peak}$	Maximum dynamic runtime memory required across layers
σ	Singular value ($\sqrt{\lambda}$)
σ	Standard deviation of a signal (when used in statistics)
σ_{new}	Updated singular value after compression penalty
$T_{data\ movement}$	Latency to move network weights or activations into dynamic memory
$T_{computation}$	Execution latency of arithmetic computations
T_{memory}	Execution latency of memory access operations
T_{min}, T_{max}	Minimum and maximum temperature in a sampled window
U, V	Left and Right singular matrices retained after lowering matrix rank
$u, \hat{u}$	True State of Health (SoH) and Estimated State of Health
u_t, u_x	Partial derivatives of SoH with respect to time (t) and input features (x)
V_{min}, V_{max}	Minimum and maximum voltage in a sampled window
v_1, u_1	Right and Left Singular Vectors
$Var(P)$	Variance of instantaneous power
$\hat{W}$	Dequantized (recovered) approximation of the original weights

$\bar{x}, \bar{y}$	Mean of a signal or target distribution
x, y_i	Input vector and true target/output vector (hard label)
$\hat{y}_{s,i}$	Estimated State of Health by the Student model
$\hat{y}, Y_{pred}$	Predicted output vector
$y_{PINN,i}$	Estimated State of Health by the Teacher/PINN model (soft label)

Functions and Operators

$det(\cdot)$	Determinant operator
$F(\cdot; \Phi)$	Primary solution network parameterized by weights Φ
$G(\cdot; \Theta)$	Secondary network modeling degradation dynamics parameterized by Θ
$g(\cdot)$	Exact non-linear aging function governing internal degradation dynamics
I	Identity matrix (when used in linear algebra/SVD)
$\phi(\cdot)$	Generic non-linear activation function
$\sigma(\cdot)$	Sigmoid activation function
$\star$	Convolution operator
$\odot$	Hadamard (element-wise) product

List of Figures

2.1	Illustration of charging and discharging Process in a Lithium ion battery cell [1]	7
2.2	Illustration of State of Health (SoH) estimation methods [2].	11
2.3	Illustration of Direct method [2].	11
2.4	Illustration of Model-based methods [2]	13
2.5	Illustration of Data-Driven methods [2].	14
2.6	Illustration of Hybrid Model-Data methods [2].	15
2.7	Illustration of MLP architecture [3].	17
2.8	Illustration of CNN architecture [4].	17
2.9	Illustration of LSTM architecture [3].	18
2.10	Illustration of Transformers architecture [3].	19
2.11	Illustration of PINN architecture [5].	21
2.12	Illustration of unstructured pruning [6]	23
2.13	Illustration of structured pruning [6]	23
2.14	Illustration of Quantization	24
2.15	Illustration of LoRA	25
2.16	Illustration of Knowledge Distillation [7]	25
2.17	Illustration of Neural Architecture Search[8]	26
2.18	Illustration of Frameworks workflow [3]	28
3.1	Illustration of Frameworks workflow	30
3.2	Illustration of preprocessed data of MIT Lab (LFP)	33
3.3	Illustration of preprocessed data of HUST Lab(LFP)	33
3.4	Illustration of preprocessed data of ECO Lab(LFP)	33
3.5	Illustration of preprocessed data of LUHBLANK2024cyclic Lab(NMC)	34
3.6	Illustration of preprocessed data of LUHBLANK2024profile Lab(NMC)	34
3.7	Illustration of Raw data and preprocessed data of XJTU Lab(NMC)	34
3.8	Illustration of Raw data and preprocessed data of TONGJI Lab(NCA)	35
3.9	Illustration of Raw data and preprocessed data of NASA Lab(NCA)	35
3.10	Comparison of early-cycle data from different datasets.	36
3.11	Illustration of input data sampling, LAB:MIT	37
3.12	Illustration of input data sampling, LAB:HUST	37
3.13	Illustration of input data sampling, LAB: XJTU	37
3.14	Illustration of input data sampling, LAB: NASA	37
3.15	Illustration of data split strategy	43
3.16	Illustration of GP training [9]	45

3.17	Illustration of GP training Framework [9]	46
3.18	Illustration of Compression Aware Training Framework	48
3.19	Illustration of relation KD	49
3.20	Illustration of relation PINN-KD [5]	53
3.21	Illustration of Post quantization Training [10]	53
3.22	Apache TVM compilation pipeline translating PyTorch models into bare-metal C code and aligned weight structures.	58
4.1	Illustration of Feature importance using random forest	63
4.2	Illustration of Maximal Information Coefficient	64
4.3	Illustration of Feature multicollinearity check	64
4.4	Baseline Model architecture [3]	66
4.5	HUST dataset results	66
4.6	MIT dataset results	66
4.7	Attia dataset results.	66
4.8	luhblank2024 Cyclic dataset results.	67
4.9	luhblank2024 profile dataset results.	67
4.10	RWTH dataset results.	67
4.11	XJTU dataset results.	68
4.12	GESLIN2025 dataset results.	68
4.13	GRAZ2023 dataset results.	68
4.14	STROEBL dataset results.	68
4.15	LFP dataset results specialized LFP domain	69
4.16	LFP dataset results specialized LFP domain	69
4.17	LFP dataset results specialized LFP and NMC domain	70
4.18	LFP dataset results of specialized LFP and NMC domain	71
4.19	LFP dataset results of specialized LFP, NMC and NCA domain	71
4.20	LFP dataset results of specialized LFP, NMC and NCA domain	71
4.21	Illustration of Gradient Dominance in the gradient updates	73
4.22	Illustration of Gradient cosine similarity	73
4.23	Illustration of Representational Similarity Analysis (RSA)	74
4.24	Illustration of Latent/hidden state features in 2d space	75
4.25	Illustration of Pruning ration vs Relative accuracy	77
4.26	HUST dataset results of GP model	77
4.27	HUST dataset results of GP model	78
4.28	MIT dataset results of GP model	79
4.29	HUST dataset results of CAT	82
4.30	MIT dataset results of CAT	83
4.31	HUST dataset results of Relational KD	85
4.32	MIT dataset results of Relational KD	86
4.33	HUST dataset results of PINN-KD.	88
4.34	MIT dataset results of PINN-KD.	89
4.35	Comparison of Flash and SRAM memory footprints across the models. .	93
4.36	Mathematical fidelity validation of the W8A32 Quantized model. . . .	94
4.37	Latency comparison between the Expert Model and Distilled Model across 10 independent datasets executed on the NXP MCXN947. . . .	95

4.38 Cross-platform mathematical fidelity. The physical execution on the NXP MCXN947 closely aligns with the PC simulation, with minor acceptable hardware drift stemming from MCU floating-point rounding mechanisms.	97
--	----

List of Tables

2.1	Comparison of deep learning architectures for battery State of Health (SOH) Prediction	21
3.1	Summary of battery datasets used in this work.	31
4.1	Extracted Feature Set	62
4.2	Selected Features	65
4.3	Performance of the specialized LFP domain LSTM model on Test battery datasets.	70
4.4	Performance of the specialized LFP and NMC domain LSTM model on Test battery datasets.	72
4.5	Performance of the specialized LFP,NMC and NCA domain LSTM model on Test battery datasets	72
4.6	Performance of the model across different folds of LFP cells only. . .	76
4.7	Performance of the Sparsed GP model across different folds of LFP cells only.	80
4.8	Performance of the Sparsed int8 GP model across different folds of LFP cells only.	80
4.9	Performance of the CAT model ($\tau = 0.001$) FP32 model across different folds.	81
4.10	Performance of the R-KD student model across different folds.	84
4.11	Performance of the INT8 model with relational knowledge distillation across different folds.	87
4.12	Performance of the PINN-KD student FP32 model across different folds.	87
4.13	Performance of the PINN-KD student INT8 model across different folds.	87
4.14	Comparison of model compression and deployment metrics.	90
4.15	Comparison of model compression and estimated profiling metrics of Relational-KD.	91
4.16	Comparison of model compression and estimated profiling metrics of PINN-KD	91

Contents

List of Acronyms	ix
Nomenclature	xi
List of Figures	xv
List of Tables	xviii
1 Introduction	1
1.1 Background	1
1.1.1 Deep Learning for SoH Estimation	2
1.1.2 The Edge/Embedded Deployment Challenge	2
1.2 Problem Statement	3
1.3 Aim and Research questions	3
1.4 Main Contributions	4
1.5 Scope	4
1.6 Risk Analysis	5
2 Theory	7
2.1 Lithium-Ion Batteries and State of Health (SoH)	7
2.1.1 Lithium-Ion Battery Fundamentals	7
2.1.2 Battery Degradation Mechanisms	8
2.1.3 State of Health (SoH) definition and methods	10
2.2 Deep Learning Methods for Battery SOH prediction	16
2.2.1 Multi-Layer Perceptrons (MLP)	16
2.2.2 Convolutional Neural Networks (CNNs)	16
2.2.3 Long Short-Term Memory (LSTM) Networks	17
2.2.4 Attention Mechanisms and Transformers	18
2.2.5 Physics-Informed Neural Networks (PINNs)	19
2.3 Model Compression Techniques	22
2.3.1 Network Pruning and Sparsity	22
2.3.2 Quantization	23
2.3.3 Low Rank Approximation (LoRA)	24
2.3.4 Knowledge Distillation (KD)	24
2.3.5 Neural Architecture Search (NAS)	26
2.4 Embedded AI Deployment Frameworks	26
2.5 Real-Time Embedded Systems and Concurrency	27

2.5.1	Bare-Metal Execution and Static Memory Management	27
2.5.2	Asynchronous Data Streams and Double-Buffering	28
2.5.3	Functional Safety and Mathematical Fault Isolation	29
3	Methodology	30
3.1	Data source and Data preparation	31
3.1.1	Data source	31
3.1.2	Data preparation	32
3.2	Feature engineering	38
3.2.1	Feature extraction	38
3.2.2	Feature selection	39
3.3	Model Development	40
3.3.1	Baseline Model Architecture	40
3.3.2	Model Training	41
3.3.3	Model Evaluation	44
3.4	Model compression and Optimization Strategies	45
3.4.1	Grow prune Train	45
3.4.2	Compression aware training	46
3.4.3	Relational Knowledge distillation	48
3.4.4	PINN-Knowledge distillation	51
3.4.5	Post-Training Quantization	52
3.5	Resource Metrics Profiling	54
3.6	Model Compilation and Runtime Engine Selection	56
3.6.1	Intermediate Representation and Compilation Backend	57
3.6.2	Memory Management and Weight Handling	57
3.6.3	Custom W8A32 Dynamic Quantization Pipeline	58
3.7	Hardware Deployment and Validation on NXP MCXN947	59
3.7.1	Isolated Inference Benchmarking via Feature Injection	60
3.7.2	FlexCAN Double-Buffering and Concurrency Control	60
3.7.3	On-Device Feature Engineering and System Safety	60
4	Results and Discussion	62
4.1	Features selection	62
4.1.1	List of Features Extracted	62
4.1.2	Selected Features	62
4.2	Baseline model evaluation	65
4.2.1	Architecture and Hyperparameter Optimization	65
4.2.2	Experimental Design for Condition Agnostic Feasibility	69
4.3	Model Compression, quantization and Compressed model evaluation	76
4.3.1	Grow prune strategy	76
4.3.2	Compression Aware training	80
4.3.3	Knowledge Distillation	81
4.4	Models Performance profiling and Final model selection	90
4.4.1	Hardware Target Constraints (NXP MCXN947)	90
4.4.2	Empirical Profiling and Comparative Analysis	90
4.4.3	Final Model Selection	91
4.5	Model Compression and Resource Optimization	92

4.5.1	Flash Storage and SRAM Footprint Reduction	92
4.5.2	Mathematical Fidelity and Inference Accuracy (PC SIL) . . .	93
4.6	Real-Time Embedded Performance	94
4.6.1	Execution Validation on Infineon TC4D7	94
4.6.2	Inference Performance Benchmarking on MCXN947	95
4.6.3	Cross-Platform Mathematical Fidelity (Silicon vs PC)	95
4.6.4	Concurrency and End-to-End Challenges in System Integration	96
5	Future Work and Conclusion	98
	Bibliography	101
	caption	

1

Introduction

1.1 Background

The Role of Battery Management Systems (BMS): The rapid increase of electric vehicles (EVs) and renewable energy storage systems, as well as with enormous developments in drone technology and mobile robotics, has driven significant advancements in lithium-ion (Li-ion) battery technology. Across all these applications, ensuring the safety, efficiency, and endurance of the battery pack requires a highly reliable Battery Management System (BMS). The BMS manages critical functionalities, including thermal management, cell balancing, fault diagnosis, and continuous state estimation [11].

The Importance of State of Health (SoH) Monitoring: Among the battery states discussed in [12], the State of Health (SoH) is one of the most important because it indicates how much the battery has aged. It reflects the permanent loss of battery capacity and increase in internal resistance that occur during long-term use. By monitoring SoH, the Battery Management System (BMS) can better understand the condition of the battery and adjust charging and discharging processes to reduce stress on aging cells, helping to extend battery life. Accurate SoH information also improves overall battery management by providing more reliable driving-range estimates and enabling power delivery based on the battery's actual condition rather than its original specifications. In addition, SoH estimation supports the reuse of retired electric vehicle batteries in second-life applications, such as stationary energy storage systems, which helps reduce waste and make better use of available energy resources. Furthermore, continuous SoH monitoring is essential for estimating the Remaining Useful Life (RUL) of a battery, ensuring reliable operation and reducing the risk of serious failures, including thermal runaway [13][14].

Limitations of Conventional Estimation Methods: Traditionally, BMS algorithms have relied on model-based techniques, such as Equivalent Circuit Models (ECMs) coupled with Kalman filters, or empirical degradation models to estimate SoH [15]. While these methods are highly interpretable, battery aging is a highly complex, nonlinear, and time-varying electrochemical process that is heavily influenced by operating temperatures, discharge rates, and charging patterns. Consequently, constructing a generalized mathematical model that remains accurate throughout the battery life cycle is difficult, and this requires updating battery parameters [16].

1.1.1 Deep Learning for SoH Estimation

To overcome the limitations of conventional methods of modeling, data-driven approaches using Machine Learning (ML) and Deep Learning (DL) have emerged as highly effective alternatives. Deep learning architectures, particularly Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU), as well as their different combinations like CNN-LSTM, Transformers, and Physics-Informed Neural Networks (PINN), excel at capturing the complex, nonlinear dependencies inherent in battery degradation data. By processing extensive datasets of historical charge-discharge cycles, these algorithms can map multidimensional time-series inputs and statistical features—such as capacity degradation trajectories, and temperature gradients. By processing extensive datasets of historical charge-discharge cycles, these algorithms can map multidimensional time-series inputs and statistical features—such as capacity degradation trajectories, temperature gradients, and localized voltage distribution characteristics—directly to the current SoH of the battery [17]. This approach allows the system to accurately capture degradation dynamics without the requirement of complex parameterization processes.

1.1.2 The Edge/Embedded Deployment Challenge

Despite the data driven methods gives good accuracy and prediction capabilities, the transition of these deep learning models from high resource workstations to real-world vehicles remains a critical bottleneck. Modern deep learning models are computationally intensive and demand high memory footprints. Conversely, a typical automotive BMS relies on resource-constrained embedded microcontrollers that operate under strict limitations regarding SRAM, Flash memory, and power consumption [18].

Many researchers point out that running large, highly detailed ML models on small embedded processors often takes too much time. For a Battery Management System to be safe, it must finish its calculations within strict deadlines. If the ML is too slow, it can disrupt the entire system and make it unstable or unsafe. Additionally, simply copying a standard ML model directly onto a microcontroller usually causes memory crashes. The processor either runs out of space or fails to deliver the complex mathematical operations required. This creates a major roadblock for actually using ML to estimate battery health in real-world applications [19][20][21].

While techniques such as post-training quantization, pruning, quantization-aware training (QAT) and using compression-aware training have been proposed to compress these models, such as utilizing lookup tables to replace floating point operations [22] [23]. There is limited comprehensive research evaluating the trade-off between compression-induced accuracy degradation and embedded performance specifically tailored for battery SoH estimators.

1.2 Problem Statement

As discussed, the main challenge in modern battery management is no longer achieving high model accuracy, but rather making these models work in real-world systems. The core problem this thesis addresses is the lack of a proven process for moving deep learning State of Health estimators onto small microcontrollers without missing strict safety deadlines. Specifically, current research does not clearly show how different model compression methods affect both the accuracy of the predictions and the actual running speed on real hardware. Until these compressed models are rigorously tested directly on embedded processors, AI-driven battery management will remain just a concept rather than a practical tool.

1.3 Aim and Research questions

The aim of this thesis is to develop, optimize, and deploy an efficient deep learning SoH estimation model directly on edge hardware. This work focuses on systematically applying model compression techniques to meet strict embedded resource limits while maintaining high estimation accuracy. By establishing a rigorous Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) verification workflow across multiple platforms, this research aims to provide a deterministic, standalone edge solution that ensures both timing compliance and algorithmic reliability for next-generation BMS applications.

Main research questions:

Q1: Is it feasible to develop a condition-agnostic and generalized Long Short-Term Memory (LSTM) model for battery State of Health (SoH) estimation using diverse publicly available datasets, and what baseline estimation accuracy can be achieved before considering embedded deployment constraints?

Q2: How do advanced deep learning compilation frameworks, specifically Apache TVM utilizing Ahead-of-Time (AOT) and Relax Virtual Machine (VM) architectures, enable the deterministic deployment of PyTorch models onto bare-metal microcontrollers, and what is their impact on inference latency, mathematical fidelity, and static memory footprint?

Q3: How do various model optimization techniques (such as architecture reduction, pruning, post-training quantization, and compression-aware training) impact the trade-off between SoH estimation accuracy and embedded resource usage?

Q4: What is the minimum viable model complexity required to maintain acceptable SoH estimation accuracy on BMS-class microcontrollers, and how does this requirement change across different hardware platforms?

Q5: How do the optimized embedded models perform in terms of timing compliance, memory safety, and system reliability when evaluated through Software-in-the-Loop

(SIL) latency benchmarking and subjected to asynchronous, high-frequency CAN bus data streams within a Process-in-the-Loop (PIL) double-buffered concurrent architecture?

1.4 Main Contributions

This thesis bridges the gap between complex deep learning algorithms and resource-constrained edge hardware for modern Battery Management Systems (BMS). The main contributions of this work are summarized as follows:

- **Physics-Informed Dimensionality Reduction:** A novel Physics-Informed Neural Network Knowledge Distillation (PINN-KD) framework is proposed. This method effectively reduces the dimensionality of complex battery degradation dynamics, transferring critical electrochemical constraints and aging patterns from a high-capacity expert model into a lightweight, edge-compatible Student architecture without sacrificing predictive reliability.
- **Advanced Embedded Compilation and Quantization:** The Apache TVM compiler stack to transform PyTorch models into standalone, deployable C/C++ engineering projects tailored for Microcontroller Units (MCUs). This implementation features a decoupled architecture separating executable inference logic from model weights (which are aligned for direct Flash memory access), and introduces a custom W8A32 dynamic quantization strategy that drastically reduces the physical memory footprint.
- **Real-Time Processor-in-the-Loop (PIL) Validation:** Successfully designed the execution workflow and validated the generated models on real automotive-grade edge hardware, specifically the NXP MCXN947 microcontroller. This deployment includes an isolated inference benchmarking protocol and a robust double-buffering concurrency mechanism to ensure safe, deterministic execution under high-frequency asynchronous sensor data streams.

1.5 Scope

This thesis encompasses the development and deep-embedded deployment of advanced Long Short-Term Memory (LSTM) network architectures for accurate battery State of Health (SoH) estimation. The algorithmic scope focuses on utilizing Physics-Informed Neural Networks (PINN) and Knowledge Distillation (KD) paradigms (PINN-KD) to compress complex, multidimensional degradation dynamics into lightweight, edge-compatible Student models.

Crucially, the deployment scope transitions away from generic middleware interpreters (e.g., standard ONNX or TensorFlow Lite) toward the Apache TVM compiler stack. This transition is driven by the need to eliminate runtime interpretation overhead and dynamic memory allocation, which are unsuited for strict bare-metal environments. The implementation traces PyTorch models into TVM Relay/Relax Intermediate Representations (IR) to generate highly optimized, statically allocated bare-metal C applications targeted directly at resource-constrained microcontrollers.

By compiling ahead-of-time, the pipeline ensures deterministic execution timing and prevents memory fragmentation.

To operate within tight hardware budgets, the optimization scope specifically investigates memory-aligned structural engineering, including W8A32 custom quantization, static memory bump allocators for SRAM management, and Flash-friendly continuous weight header generation. These mechanisms ensure that memory consumption remains strictly bounded and completely predictable over extended operational lifecycles.

Finally, the validation scope transitions from PC-based mathematical verification to real-time edge hardware execution. The deployment is comprehensively evaluated using an NXP MCXN947 for Processor-in-the-Loop (PIL) testing to assess absolute inference latency via direct feature injection, as well as system resilience and concurrency control under live, asynchronous FlexCAN data streams.

1.6 Risk Analysis

This section outlines the potential risks identified for the implementation of edge-optimized deep learning models on embedded platforms, along with the corresponding mitigation strategies.

1. Unsupported Neural Network Operators

- **Risk Source:** A significant technical risk involves the conversion of PyTorch models to C/C++ implementations specifically for embedded inference engines. There is a high probability that specific neural network layers, such as complex LSTM variants or custom attention mechanisms, may not be supported by the target micro-controller's instruction set or the inference library.
- **Action:** To mitigate this, the project will conduct "naive deployment" tests during the early prototyping phase to identify unsupported operators immediately. If incompatibilities arise, the strategy involves replacing complex custom layers with standard supported operations or, if feasible within the timeline, implementing custom kernels.

2. Hardware Resource Overflow

- **Risk Source:** Deploying deep learning models on resource-constrained micro-controllers presents a high risk of exceeding the available SRAM or Flash memory. An overflow in memory usage would prevent the application from loading or executing.
- **Action:** The primary mitigation strategy is the systematic application of model compression techniques, including Post-Training Quantization (PTQ) and pruning. Additionally, iterative architecture reduction will be performed to decrease the number of hidden layers or units, ensuring the model fits within the strict hardware footprint while maintaining acceptable performance.

3. Real-Time Latency Violation

- **Risk Source:** For safety-critical Battery Management Systems (BMS), the inference time of the deep learning model must strictly adhere to the control loop deadlines. There is a risk that the model inference may introduce latency that violates these real-time constraints.
- **Action:** A benchmarking protocol will be established to measure inference latency on the actual target hardware early in the development cycle. Optimization efforts will focus on code generation workflows that utilize hardware-specific acceleration instructions, and trade-offs between minor accuracy reductions and execution speed will be evaluated quantitatively.

4. Accuracy Degradation from Quantization

- **Risk Source:** The transition from 32-bit floating-point models to 8-bit integer formats poses a risk of significant accuracy degradation in State of Health (SoH) and State of Charge (SoC) estimation. If the accuracy drops below the safety threshold, the model becomes unusable for BMS applications.
- **Action:** Quantization-Aware Training (QAT) will be implemented to allow the model to adapt to quantization noise during the training process. Furthermore, functional equivalence between the high-level Python model and the converted C++ model will be rigorously validated against full-precision baselines.

5. Toolchain Integration Complexity

- **Risk Source:** Integrating the generated C/C++ deep learning code into the existing Simulink-based BMS framework creates a risk of interface mismatches, compilation errors, or signal compatibility issues. This complexity can lead to significant delays in the validation phase.
- **Action:** The project will employ a Processor-in-the-Loop (PIL) verification strategy to validate the workflow incrementally. By testing the integration with a "minimum viable model" early in the process, potential integration failures can be resolved before the final, complex models are deployed.

2

Theory

2.1 Lithium-Ion Batteries and State of Health (SoH)

2.1.1 Lithium-Ion Battery Fundamentals

Lithium-ion (Li-ion) batteries operate as electrochemical energy storage devices through a reversible process known as intercalation[24]. The basic architecture comprises three primary components: a negative electrode (anode, typically composed of graphite), a positive electrode (cathode, typically a lithium metal oxide such as $LiCoO_2$ or $LiFePO_4$), and an ionically conductive, electronically insulating electrolyte solution containing dissolved lithium salts[25].

During the discharge phase, lithium atoms at the anode undergo oxidation, simultaneously releasing electrons into the external circuit and liberating lithium ions into the electrolyte. The flow of electrons through the external load is driven by the electrochemical potential difference between the electrodes, while the concurrent migration of lithium ions through the liquid electrolyte maintains internal charge neutrality as they intercalate into the cathode [24] [25]. The charging process reverses this mechanism; an external voltage source forces lithium ions out of the cathode crystalline structure, driving them back across the electrolyte to be re-intercalated into the anode[26].

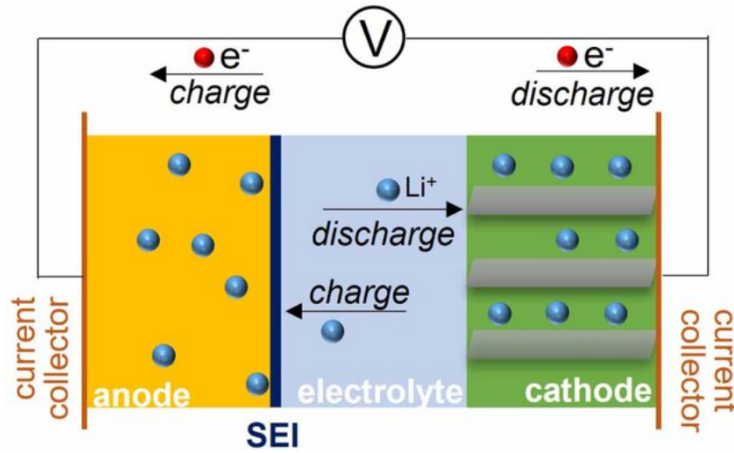


Figure 2.1: Illustration of charging and discharging Process in a Lithium ion battery cell [1]

2.1.2 Battery Degradation Mechanisms

The performance of lithium-ion batteries degrades over time due to a complex network of parasitic chemical and mechanical degradation mechanisms [27]. These degradation pathways are typically categorized into three main modes:

Loss of Lithium Inventory (LLI): This is primarily driven by the continuous growth of the Solid Electrolyte Interphase (SEI) layer on the graphite anode. During initial cycles, the electrolyte decomposes upon contact with the highly reducing anode surface, creating a protective passivating layer. However, due to volumetric changes during cycling, this layer cracks and reforms continuously, consuming active lithium ions and electrolyte solvents, permanently reducing the available capacity [28][29].

Loss of Active Material (LAM) : Mechanical are induced by the repeated insertion and extraction of lithium ions that cause microscopic volume changes in the host interstitial lattices. Over time, this leads to particle cracking, phase transitions, and electrical isolation of the electrode material, decreasing the number of available sites for lithium intercalation [28][29].

Loss of Electrolyte (LE): The electrolyte is lost in SEI formation and drying out of the liquid electrolyte. This impedes the physical transport of lithium ions between the electrodes, severely increasing the internal ohmic resistance of the cell.

These three macro-level modes are driven by a variety of distinct, micro-level physical and chemical mechanisms occurring in the different components inside the battery

Anode (Negative Electrode) Mechanisms:

Solid Electrolyte Interphase (SEI): it is normally the most dominant cause of aging during normal operation. The electrolyte continuously reduces on the surface of the graphite anode, forming a passivating layer. While necessary for protection, this layer continuously cracks and reforms due to volume changes during cycling, perpetually consuming active lithium and electrolyte solvents (driving LLI and LE) and thickening the resistive barrier [30].

Lithium Plating: Under harsh operating conditions—such as low temperatures or excessively high charging currents (high C-rates)—the rate of lithium diffusion into the graphite anode is slower than the deposition rate. Consequently, metallic lithium plates onto the surface of the anode rather than intercalating into the lattice. This causes severe LLI and can lead to the formation of dendrites, which pose a critical risk of penetrating the separator and causing internal short circuits [31].

Particle Cracking and Fracture: The intercalation and extraction of lithium cause microscopic volume expansion and contraction (mechanical stress). Over time, this stress cracks the graphite particles, exposing fresh, unpassivated graphite to the electrolyte and triggering further SEI formation [32].

Graphite Exfoliation: Solvent molecules from the electrolyte can occasionally co-intercalate (enter the graphite layers alongside lithium ions). Once inside, they decompose and generate gas, which physically pushes the graphite layers apart, destroying the crystal structure and causing localized LAM [27].

Loss of Electrical Contact: Extreme mechanical volume changes or binder degradation can cause active graphite particles to physically detach from the conductive carbon matrix or the copper current collector. These isolated particles become electrochemically inactive, resulting in [33].

Cathode (Positive Electrode) Mechanisms:

Transition Metal Dissolution: Metals inherent to the cathode lattice (such as manganese, nickel, or cobalt) can dissolve into the acidic electrolyte. This structural collapse directly causes LAM. Furthermore, the dissolved metal ions migrate through the electrolyte to the anode, where they act as catalysts to accelerate continuous SEI growth [34].

Structural Phase Transitions: Operating at extreme states of charge or high voltages can permanently alter the crystalline structure of the cathode. This structural degradation permanently reduces the cathode's ability to host lithium ions (LAM) [35].

Cathode Electrolyte Interphase (CEI) Growth: Similar to the anode, the electrolyte can oxidize on the high-voltage surface of the cathode. This forms a resistive surface film (CEI) that increases internal resistance (power fade) and consumes active lithium [35].

Particle Cracking: Cathode materials also experience mechanical stress from volume changes during cycling. Repeated stress causes primary and secondary cathode particles to fracture, isolating active material and exposing new surfaces to continuous CEI growth [33].

Electrolyte and System-Level Mechanisms:

Electrolyte Decomposition and Gas Generation: Exposure to high temperatures or high operating voltages causes the liquid electrolyte solvents to break down, generating gases such as CO_2 , H_2 , and C_2H_4 . This gas generation can cause cell swelling and significantly contributes to Loss of Electrolyte [28].

Separator Clogging: Degradation byproducts, such as dissolved transition metals or excessive SEI components, can clog the microscopic pores of the polymer separator. This restricts the flow of lithium ions, dramatically increasing internal resistance and limiting the cell's power capability [28].

Current Collector Corrosion: The copper current collector on the anode side can corrode if the cell is severely over-discharged. Conversely, the aluminum current

collector on the cathode side can corrode under excessively high voltages. Both scenarios increase the cell's internal ohmic resistance [36].

2.1.3 State of Health (SoH) definition and methods

Definition:

The State of Health (SoH) is a dimensionless, indirectly measurable metric that quantifies a battery's irreversible physical and electrochemical degradation relative to its fresh state. Unlike the State of Charge (SoC), which reflects transient energy levels, SoH represents the cumulative macroscopic impact of microscopic aging mechanisms over time such as Solid Electrolyte Interphase (SEI) growth and the loss of active lithium [37]. Because SoH cannot be read directly by standard voltage or current sensors, the Battery Management System (BMS) must continuously estimate it using advanced algorithms. Accurate SoH tracking is strictly required for the BMS to dynamically adjust safe operating limits, ensure reliable driving range predictions, and determine the system's Remaining Useful Life (RUL) before a critical failure threshold is reached [38].

Classification SOH Estimation method:

To accurately monitor battery degradation, a wide variety of State of Health (SoH) estimation techniques have been developed in recent literature. These methodologies are broadly classified into four primary categories based on their underlying principles: direct measurement, model-based, data-driven, and hybrid estimation approaches. The following sections summarize the theoretical frameworks of these categories and compare their distinct methodologies for extracting SoH from battery operating data.

Direct method:

Direct measurement methods evaluate battery degradation through dedicated experimental procedures, such as complete charge-discharge cycling or pulse current characterization. While these empirical techniques are fundamentally straight forward, they offer the highest degree of accuracy in quantifying the State of Health (SoH). Consequently, the SoH values derived from direct experimental measurements are universally utilized as the benchmark or ground-truth reference to validate the performance of more complex estimation algorithms [39].

State of Health (SoH) by Capacity Fade:

The standard definition compares the cell's current maximum capacity to the manufacturer's nominal rating:

$$\text{SoH}_C = \frac{C_{\text{current}}}{C_{\text{nominal}}} \times 100\%, \quad (2.1)$$

where C_{current} is the aged battery's maximum capacity, and C_{nominal} is the initial rated capacity.

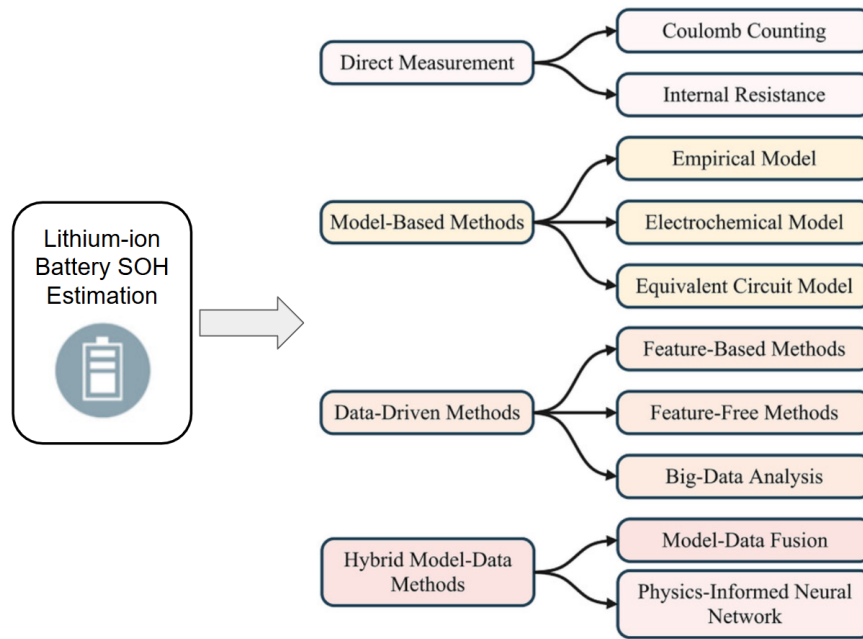


Figure 2.2: Illustration of State of Health (SoH) estimation methods [2].

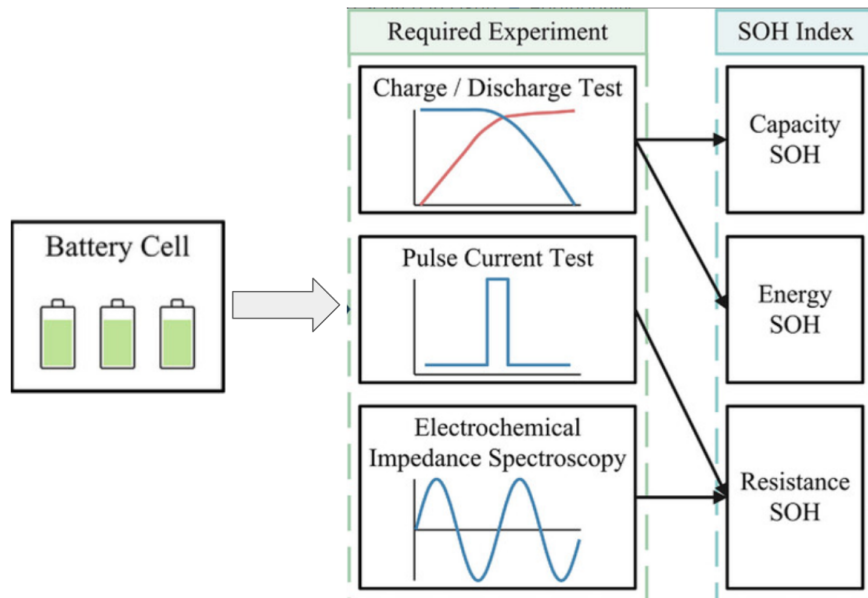


Figure 2.3: Illustration of Direct method [2].

In automotive applications, End-of-Life (EOL) is typically defined as 80% of the initial capacity due to safety and performance risks. Consequently, a scaled SoH definition maps this 80% absolute capacity threshold to 0% usable SoH [38]:

$$\text{SoH}_{C,\text{scaled}} = \left(1 - \frac{C_{\text{new}} - C_{\text{current}}}{0.2 \times C_{\text{new}}}\right) \times 100\%, \quad (2.2)$$

where C_{new} is the initial capacity of the fresh battery.

State of Health (SoH) by Internal Resistance:

Rising internal (ohmic) resistance limits instantaneous power delivery and increases thermal losses. Therefore, SoH can also be defined based on resistance growth:

$$\text{SoH}_R = \frac{R_{\text{EOL}} - R_{\text{current}}}{R_{\text{EOL}} - R_{\text{new}}} \times 100\%, \quad (2.3)$$

where R_{current} is the present internal resistance, R_{new} is the fresh cell resistance, and R_{EOL} is the predefined End-of-Life (EOL) resistance threshold (often corresponding to a 100% increase over R_{new}).

Because internal resistance varies nonlinearly with the State of Charge (SoC), temperature, and load current, SoH_R is typically evaluated under standardized operating conditions. In practice, it is often calculated as a weighted average over a complete charge–discharge cycle. In high-power applications, such as hybrid electric vehicles, this power-fade metric (SoH_R) may determine battery usability before the capacity limit (SoH_C) is reached [37].

Model-based methods:

Model-based methods utilize theoretical principles or historical knowledge to construct mathematical representations of the battery. These models dynamically estimate SoH by applying parameter optimization or adaptive filtering algorithms to real-time operational data. They are typically divided into three subcategories

Empirical Models: These rely on historical cycle data and mathematical curve-fitting techniques to quantify degradation. While simple to develop, they often exhibit lower accuracy in practice due to individual cell manufacturing variances and unpredictable or uncertain operating conditions [40].

Electrochemical Models: These utilize complex systems of partial differential equations, such as Pseudo-Two-Dimensional (P2D) or Single-Particle (SP) models, to simulate the exact internal chemical kinetics of the cell. Although highly accurate, their severe computational cost and strict reliance on proprietary internal cell parameters make them highly challenging to deploy on embedded hardware [41][42].

Equivalent Circuit Models (ECM): These emulate battery dynamics using traditional electrical networks comprising voltage sources, resistors, and capacitors. When paired with adaptive online identification algorithms like the Extended Kalman Filter (EKF) or Recursive Least Squares (RLS) to update the parameters

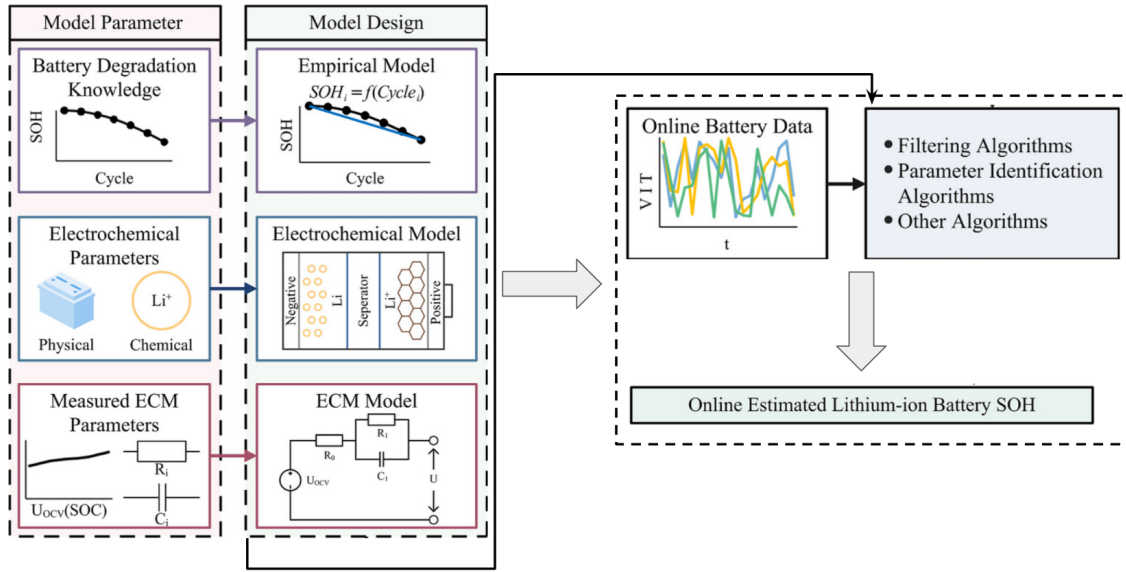


Figure 2.4: Illustration of Model-based methods [2]

in real time, ECMs provide a strong balance between computational efficiency and estimation accuracy [43][44].

Data-Driven Methods:

Data-driven approaches map raw historical operating data to battery degradation using machine learning (ML) and deep learning (DL) models, completely bypassing the need to solve complex internal electrochemical equations. These approaches are broadly divided into two methodologies:

Feature-Based Methods: These rely on the manual design and extraction of explicit Health Indicators from raw operating data, such as incremental capacity (IC) curves, voltage variances, or Electrochemical Impedance Spectroscopy (EIS) features. These engineered features are then fed into ML algorithms such as Random Forests, Gaussian Process Regression, or Extreme Learning Machines (ELM) to estimate SoH. While these models are highly interpretable, their success heavily depends on the quality of the manual feature engineering. Furthermore, they often require domain adaptation or transfer learning to maintain accuracy across different battery chemistries or varying load profiles [45][46].

Feature-Free (End-to-End) Methods: To eliminate the burden of manual feature extraction, feature-free methods feed raw time-series sensor data directly into advanced deep neural networks, such as Long Short-Term Memory (LSTM) networks or Convolutional Neural Networks (CNNs). The network automatically learns and extracts hidden temporal features or even explicitly the features can be extracted from the raw data by feature engineering approaches. While this is highly adaptive, this approach requires massive datasets for model training and inherently creates black-box models, sacrificing physical interpretability for estimation performance [47][48].

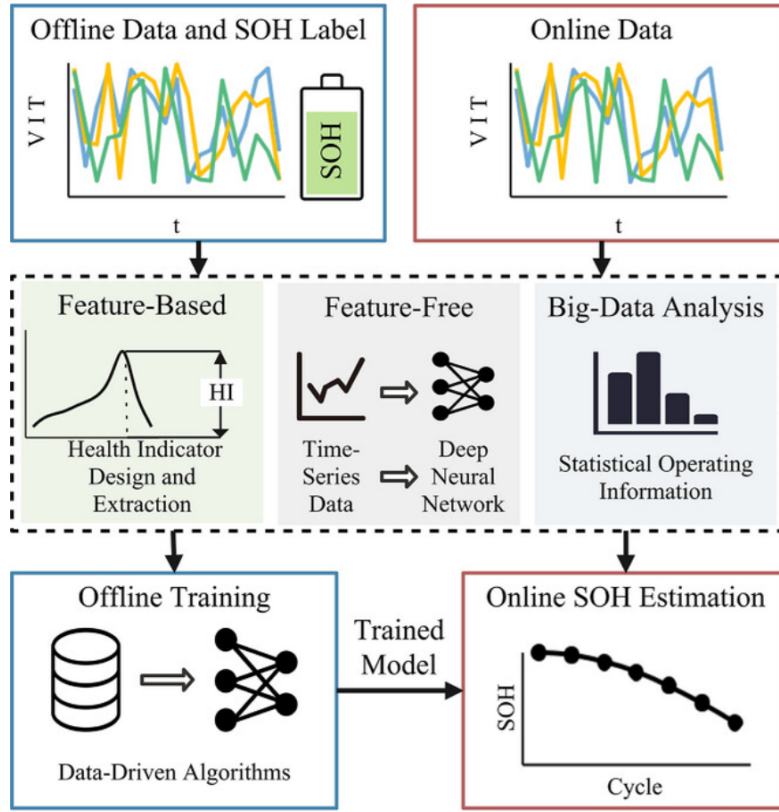


Figure 2.5: Illustration of Data-Driven methods [2].

Hybrid Model-Data Methods:

Hybrid methodologies bridge the gap between the physical interpretability of model-based approaches and the high accuracy of data-driven techniques. By integrating physical battery models with machine learning, these approaches mitigate the black-box limitations of pure AI and the computational fragility of pure mathematical models. They are primarily classified into two categories:

Model-Data Fusion (Feature-Level Integration): This approach utilizes traditional physical models (such as Equivalent Circuit Models or Open-Circuit Voltage models) as advanced feature extractors. Instead of feeding raw voltage and current into a neural network, the physical models estimate internal parameters (like RC time constants or incremental capacity peaks) [49]. These mathematically derived parameters are then used as high quality Health Indicators and fed into data-driven models (for instance CNNs or RNNs or Transformers) to estimate SoH. This significantly enhances the interpretability and transferability of the machine learning model [50][51]

Physics-Informed Neural Networks (Algorithm-Level Integration): PINNs integrate physical battery models with deep learning by embedding domain knowledge directly into the network's training process, typically through specialized physical loss functions. By constraining the network's output to obey established physical laws, PINNs significantly enhance both estimation accuracy and physical interpretability [52]. Recent literature [53][54] demonstrates this versatility through

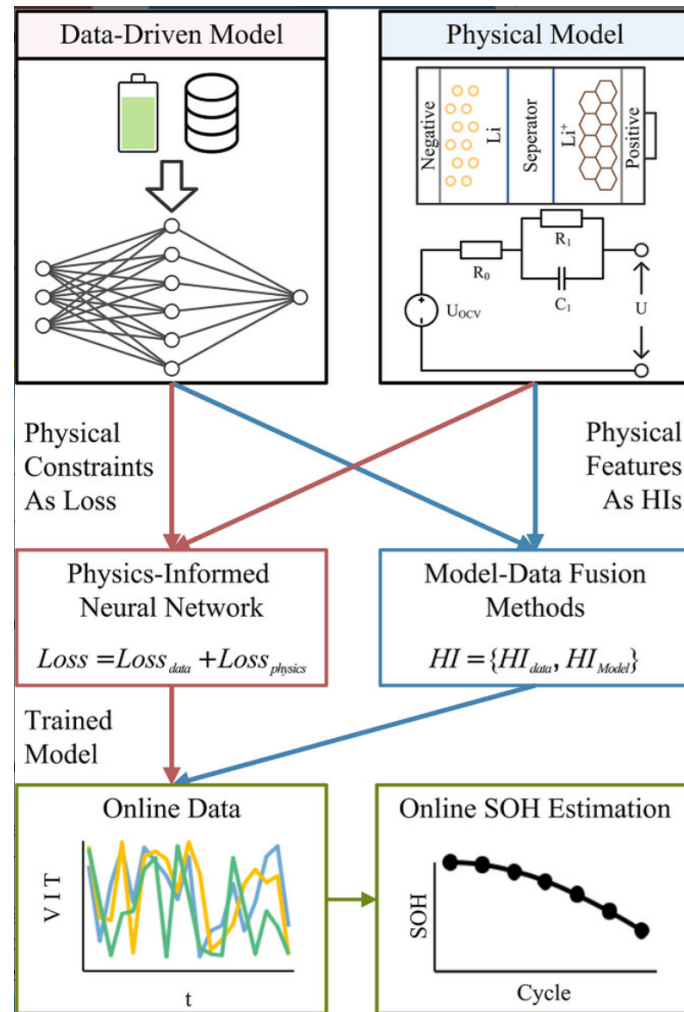


Figure 2.6: Illustration of Hybrid Model-Data methods [2].

various implementations, such as enforcing monotonic aging constraints on Recurrent Neural Networks (RNNs), embedding degradation-based partial differential equations (PDEs) into the loss function, or integrating temperature-corrected Equivalent Circuit Models (ECMs). By bridging empirical data with theoretical constraints—such as halfcell Open Circuit Voltage (OCV), PINNs extend beyond simple capacity prediction to successfully isolate, internal electrochemical states like Loss of Lithium Inventory (LLI) and Loss of Active Material (LAM) [55].

2.2 Deep Learning Methods for Battery SOH prediction

Deep learning (DL) has emerged as the dominant framework for data-driven State of Health (SoH) estimation due to its capacity to model highly non-linear, multidimensional degradation dynamics without requiring explicit electrochemical equations. Unlike traditional machine learning algorithms that rely on manually engineered health indicators, DL architectures can automatically extract latent spatio-temporal features directly from raw operational data (voltage, current, and temperature profiles). Recent literature benchmarks several distinct Deep Learning architectures and its combinations of temporal and physics-informed neural networks, each presenting unique trade-offs between estimation accuracy, computational complexity, and sequence-handling capability [3][5][56].

2.2.1 Multi-Layer Perceptrons (MLP)

The Multi-Layer Perceptron is the foundational architecture of artificial neural networks, consisting of an input layer, several fully connected hidden layers, and an output layer. The output y of a given layer is calculated as

$$y = \phi(Wx + b), \quad (2.4)$$

where $\phi(\cdot)$ represents the non-linear activation function, W is the weight matrix, b is the bias vector, and x is the input vector.

While MLPs are effective for dynamic static prediction tasks, their strictly feed-forward nature means they possess no internal memory. Consequently, they are poorly suited for sequential time-series problems like battery degradation, as they cannot capture the temporal dependencies between consecutive charging cycles without incurring an excessively heavy computational load [3].

2.2.2 Convolutional Neural Networks (CNNs)

Originally designed for spatial image processing, CNNs have been effectively adapted for 1D time-series battery data [3]. A CNN utilizes convolutional layers to apply local filters across the input sequence, extracting critical spatial features such as voltage plateaus or incremental capacity peaks. The feature map h_k^l for the k -th kernel in the l -th layer is defined as:

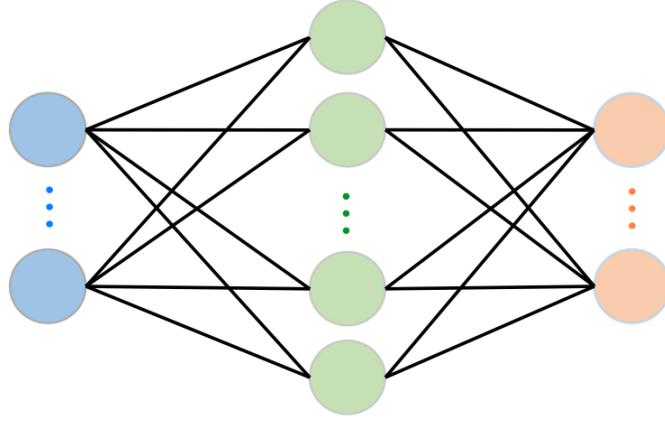


Figure 2.7: Illustration of MLP architecture [3].

$$h_k^l = \phi \left(w_k^l \star x + b_k^l \right), \quad (2.5)$$

where $\star$ denotes the convolution operator. To introduce non-linearity, activation functions like ReLU (Rectified Linear Unit) are applied, followed by pooling layers to down-sample the feature map, thereby reducing computational complexity.

While CNNs excel at isolating local variations within a single charging profile, standard 1D-CNNs lack the recurrent connections necessary to map long-term, multi-cycle degradation trends [3].

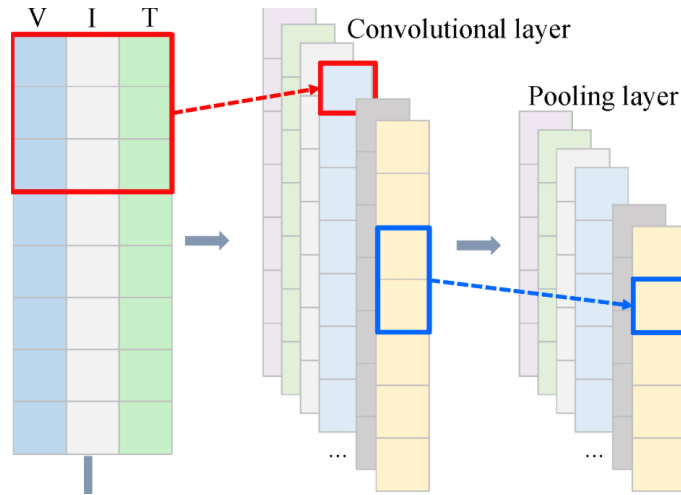


Figure 2.8: Illustration of CNN architecture [4].

2.2.3 Long Short-Term Memory (LSTM) Networks

The LSTM network, which introduces a sophisticated internal memory cell governed by a gating mechanism. An LSTM cell regulates the flow of information through three primary gates: the forget gate (f_t), the input gate (i_t), and the output gate

(o_t) according to

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \quad (2.6)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \quad (2.7)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c), \quad (2.8)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad (2.9)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \quad (2.10)$$

$$h_t = o_t \odot \tanh(c_t), \quad (2.11)$$

where $\sigma(\cdot)$ is the Sigmoid activation function, $\odot$ represents the Hadamard (element-wise) product, c_t is the cell state, and h_t is the hidden state. This architecture allows the LSTM to retain critical historical degradation data (such as early-life capacity drops) over thousands of cycles, making it industry standard for battery SoH estimation [3].

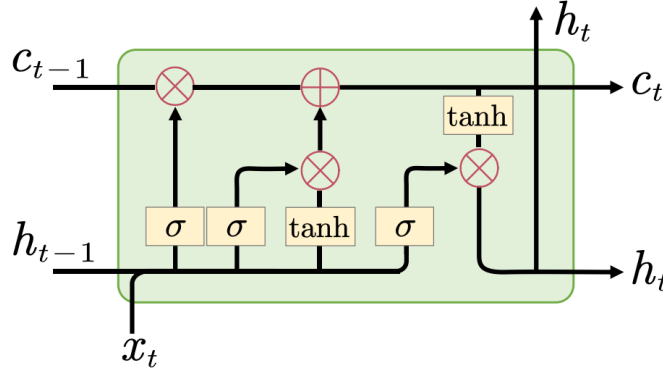


Figure 2.9: Illustration of LSTM architecture [3].

2.2.4 Attention Mechanisms and Transformers

Attention mechanisms allow neural networks to dynamically weigh the importance of different segments of an input sequence, rather than processing them strictly sequentially. The self-attention mechanism, foundational to Transformer architectures, calculates a weighted sum of values based on the relationship between Queries (Q), Keys (K), and Values (V):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.12)$$

where d_k represents the dimension of the hidden layer and softmax is generalization of the logistic function to multiple dimensions.

In battery prognostics, attention mechanisms allow the model to focus specifically on highly degraded knee point cycles or anomalous temperature events, effectively ignoring irrelevant operational noise. While Transformers offer state-of-the-art predictive accuracy, their quadratic memory complexity presents severe barriers for real-time edge deployment [3].

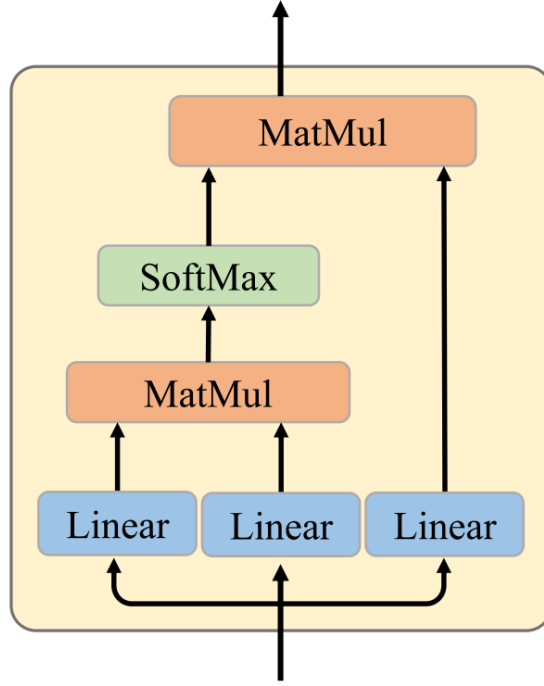


Figure 2.10: Illustration of Transformers architecture [3].

2.2.5 Physics-Informed Neural Networks (PINNs)

As highlighted in recent literature[5], traditional empirical models oversimplify battery aging as a univariate function of time. In reality, degradation is a multivariate process heavily influenced by conditions like temperature, State of Charge (SOC), Depth of discharge (DOD) and charging rates. To capture this complexity, the State of Health (SoH) is modeled as a multivariate function, $u = f(t, \mathbf{x})$, where t is time (or Equivalent full cycles) and $\mathbf{x}$ represents a vector of operational features. The internal degradation dynamics are governed by a partial differential equation (PDE):

$$\frac{\partial u}{\partial t} = g(t, \mathbf{x}, u; \theta), \quad (2.13)$$

Because the exact non-linear aging function $g(\cdot)$ is fundamentally unknown, Physics-Informed Neural Networks (PINNs) utilize deep learning to approximate these dynamics while strictly adhering to thermodynamic constraints. As illustrated in Figure 2.11, the PINN framework employs two interacting neural networks. Crucially, this framework is architecturally agnostic. Depending on the sequence-handling requirements of the data, the two sub-networks can be constructed using various combinations of deep learning models, such as MLP-MLP, LSTM-MLP, or LSTM-LSTM configurations.

1. The Solution Network $F(\cdot)$ Parameterized by learnable weights Φ , this primary network maps the temporal sequence t and feature vector $\mathbf{x}$ to output the estimated SoH ($\hat{u}$). Using automatic differentiation, it simultaneously extracts the partial derivatives of the SoH with respect to time (u_t) and input features (u_x).

2. The Dynamic Approximator Network $G(\cdot)$: Instead of relying on a rigid mathematical formula, a secondary network parameterized by Θ models the unknown degradation dynamics by taking the outputs of the solution network as inputs, approximating $u_t \approx G(t, \mathbf{x}, u, u_t, u_x; \Theta)$. By linking these networks, the physics-informed constraint $\mathcal{H}$ is defined as the residual error between the calculated temporal derivative and the output of the dynamic network:

$$\mathcal{H} = \frac{\partial F(t, \mathbf{x}; \Phi)}{\partial t} - G(t, \mathbf{x}, u, u_t, u_x; \Theta), \quad (2.14)$$

3. The Composite Loss Function: To ensure the discrete neural network training satisfies continuous physical PDEs, backpropagation is forcibly constrained by a composite loss function ($\mathcal{L}$), governed by trade-off hyperparameters α and β :

$$L = L_{\text{data}} + \alpha L_{\text{PDE}} + \beta L_{\text{mono}}. \quad (2.15)$$

Empirical Data Loss ($\mathcal{L}_{\text{data}}$): Mean Squared Error between the true SoH (u^i) and the predicted SoH ($\hat{u}^i$):

$$L_{\text{data}} = \sum_{i=1}^N (u_i - \hat{u}_i)^2. \quad (2.16)$$

PDE Physics Loss ($\mathcal{L}_{\text{PDE}}$): Strictly enforces the dynamic degradation equation, penalizing the network if the residual $\mathcal{H}$ deviates from zero:

$$L_{\text{PDE}} = \sum_{i=1}^N |H(t_i, \mathbf{x}_i)|^2. \quad (2.17)$$

Monotonicity Constraint ($\mathcal{L}_{\text{mono}}$): A Rectified Linear Unit (ReLU) penalty reflecting thermodynamic reality, ensuring predicted capacity does not unphysically increase over consecutive cycles:

$$L_{\text{mono}} = \sum_{j=1}^M \sum_{k=1}^{N_j} \text{ReLU}(\hat{u}_{k+1} - \hat{u}_k). \quad (2.18)$$

Performance Metrics and Accuracy Benchmarking:

Here Table 2.1 provides a comprehensive performance benchmark of the reviewed deep learning architectures across standardized laboratory data sources, sequence-handling capabilities, and expected error thresholds. This comparative framework maps the evolutionary trajectory of SoH prediction from basic empirical profiles to complex, physics-informed time-series models. It explicitly correlates architectural complexity with predictive precision using data from CC-CV protocols, pulse testing, and large-scale data sources such as the XJUT/MIT/Stanford dataset [3].

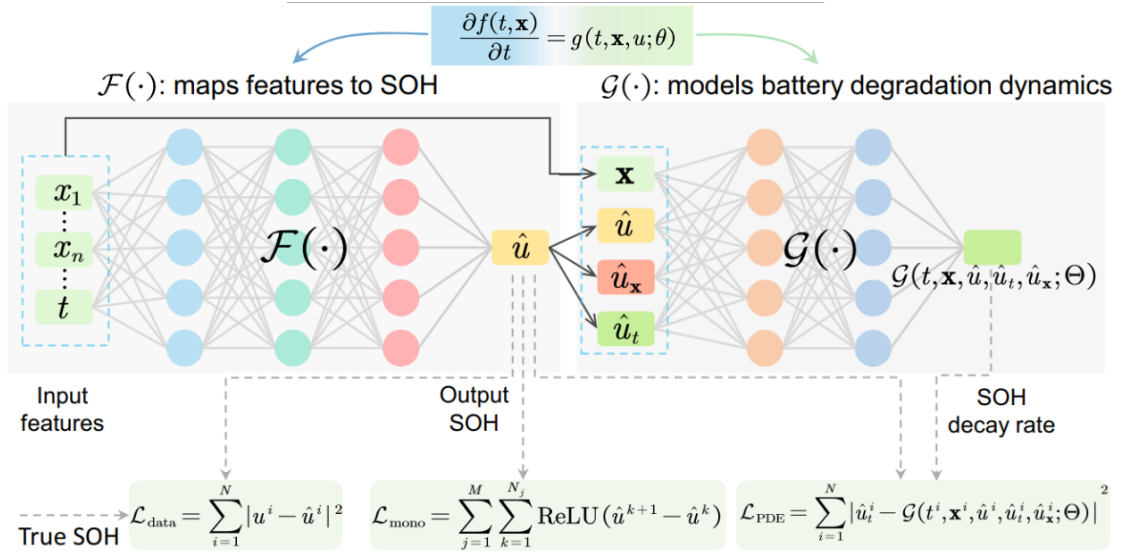


Figure 2.11: Illustration of PINN architecture [5].

Table 2.1: Comparison of deep learning architectures for battery State of Health (SOH) Prediction

Deep Learning Architecture	Data Type	MSE	R ²	Key Literature
MLP	(CC–CV) cycling/ pulse tests/Drive cycle.	0.000139	0.9331	[3]
CNN	(CC–CV) cycling/ pulse tests/Drive cycle.	0.000161	0.9187	[3]
LSTM	(CC–CV) cycling/ pulse tests/Drive cycle.	0.00011	0.9407	[3]
GRU	(CC–CV) cycling/ pulse tests/Drive cycle.	0.0000983	0.9503	[3]
Attention / Transformer	(CC–CV) cycling/ pulse tests/Drive cycle.	0.000135	0.9317	[3]
PINN	(CC–CV) cycling/ pulse tests/Drive cycle.	0.00012	-	[5]

2.3 Model Compression Techniques

The main goal of model compression is to simplify a neural network’s design. Reduce its size and its inference latency without much affecting the predictive accuracy of the model. Size reduction minimizes the total parameter count and memory footprint, directly conserving Random Access Memory (RAM) for other critical system functions. Concurrently, latency reduction accelerates the inference process, which fundamentally lowers energy consumption. Because model size and memory access frequency are closely related, these optimizations go hand-in-hand. Ultimately, achieving both reductions is a kind of mandatory prerequisite for deploying complex deep learning models onto power and memory constrained edge hardware, such as embedded microcontrollers [57].

Deploying advanced deep learning architectures on resource-constrained Battery Management Systems (BMS) requires bridging the gap between high predictive accuracy and strict hardware limitations [57]. Model compression resolves this computational bottleneck through three core paradigms:

Structural Simplification: Reduces the physical size and operation count of the network. Network Pruning induces sparsity by removing redundant weights or neurons, while Low-Rank Approximation decomposes large, dense matrices into computationally efficient sub-matrices [57][58].

Precision Scaling: Lowers arithmetic complexity without altering the network geometry. Model Quantization converts high-precision floating-point weights into low precision integers, drastically shrinking the memory footprint [57][58].

Architectural Transfer and Search: Generates inherently lightweight models. Knowledge distillation transfers the learned degradation dynamics from a massive Teacher model to a compact Student [59], while Neural Architecture Search (NAS) automates the discovery of custom hardware-optimized network topologies [8][60].

2.3.1 Network Pruning and Sparsity

Deep neural networks are fundamentally over-parameterized, often containing extra numbers of redundant weights that encode identical information. Network pruning systematically removes these unwanted connections to reduce the model’s physical memory footprint and floating-point operations (FLOPs). This process is classified into two distinct paradigms:

Unstructured Pruning: This approach evaluates and eliminates individual weights independently. Using magnitude based criteria such as the iterative thresholding method established by Han et al [61]. computationally insignificant weights are explicitly set to zero, followed by network retraining to compensate for the removed connections. While this achieves extreme compression rates with minimal accuracy loss, it generates highly irregular sparse matrices. Because standard embedded microcontrollers within a Battery Management System (BMS) cannot efficiently pro-

cess irregular memory accesses, unstructured pruning rarely yields actual latency or energy reductions without specialized hardware accelerators.

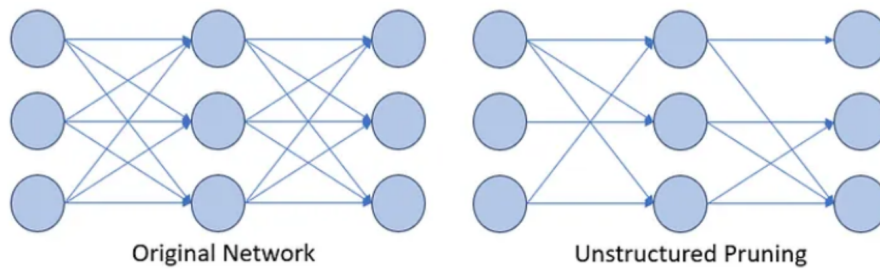


Figure 2.12: Illustration of unstructured pruning [6]

Structured Pruning Instead of picking off isolated weights, it takes a more holistic approach: it removes entire architectural building blocks, such as complete neurons, channels, or convolutional filters [62]. This is the key to unlocking actual inference speed on standard edge hardware. Because this method systematically slices out entire rows or columns, the network’s weight matrices shrink but remain perfectly dense and organized [63]. This elegantly avoids the messy, sparse connectivity bottlenecks that standard microcontrollers typically struggle to process.

Modern implementations have made this optimization process highly intelligent. For instance, Huang et al. [64] introduced data-driven pruning agents algorithms that iteratively test and remove different filter combinations to reduce the computational heavy lifting. Crucially, these agents operate under a strict drop bound, which acts as a safety net to ensure the model’s predictive accuracy never falls below a predefined limit. Ultimately, for real-time State of Health (SoH) estimation, this structured methodology is not just optional; it ensures the neural network is custom-tailored to physically fit within the strict memory and latency budgets of the target microcontroller [65].

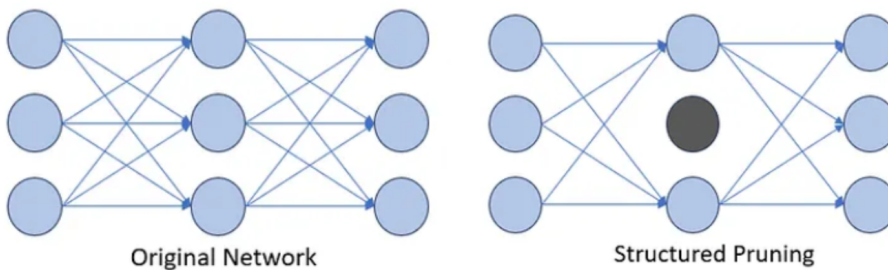


Figure 2.13: Illustration of structured pruning [6]

2.3.2 Quantization

While pruning reduces the number of parameters, quantization decreases the physical size of the weights that remain. Exploiting a neural network’s inherent resilience to numerical noise, quantization maps high-precision 32-bit floating-point (FP32)

weights and activations to lower-bit discrete formats, most commonly 8-bit integers (INT8). By binning or clustering weights based on their layer-wise distributions, techniques like those pioneered by Han et al. [58] can instantly compress a model’s physical memory footprint by a factor of four.

To achieve true hardware acceleration, the network’s mathematical operators must also be quantized. This requires careful calibration of intermediate activation ranges to prevent bit-overflows during inference[66]. Ultimately, mapping the model to INT8 precision strictly aligns it with the integer-optimized Arithmetic Logic Units (ALUs) of standard microcontrollers. By completely bypassing memory-heavy floating point operations, quantization allows models to execute fast, power-efficient State of Health (SoH) predictions directly on an embedded Battery Management System (BMS)[22].

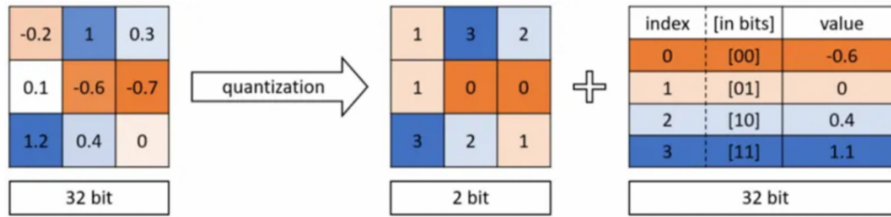


Figure 2.14: Illustration of Quantization

2.3.3 Low Rank Approximation (LoRA)

Low-rank approximation tackles over-parameterization by exploiting the inherent cross-channel and cross-filter redundancy within deep neural networks. It mathematically decomposes massive, full-rank weight tensors into a linear combination of smaller, computationally efficient sub-matrices. For standard 2D weights (in our case fully connected or LSTM layers), this is typically achieved via Singular Value Decomposition (SVD), while multi-dimensional convolutional filters utilize tensor factorizations like CP or Tucker decomposition. For instance, Kim et al. utilized Tucker decomposition to drastically reduce floating-point operations (FLOPs) and energy consumption for mobile processors [67]. Similarly, Jaderberg et al. demonstrated that factorizing full-rank filters into rank-1 vectors can yield a $4.5\times$ computational speedup with minimal accuracy loss [68].

Crucially for deployment on an embedded Battery Management System (BMS), low-rank approximation yields an architecture composed entirely of smaller, dense matrices. Because it avoids the irregular sparsity patterns generated by unstructured pruning, it requires no specialized hardware accelerators. Standard automotive microcontrollers can natively execute these factorized models, achieving immediate reductions in memory footprint and inference latency [67][68][69].

2.3.4 Knowledge Distillation (KD)

Knowledge Distillation bridges the performance gap between high-capacity neural networks and resource-constrained edge hardware. While achieving high pre-

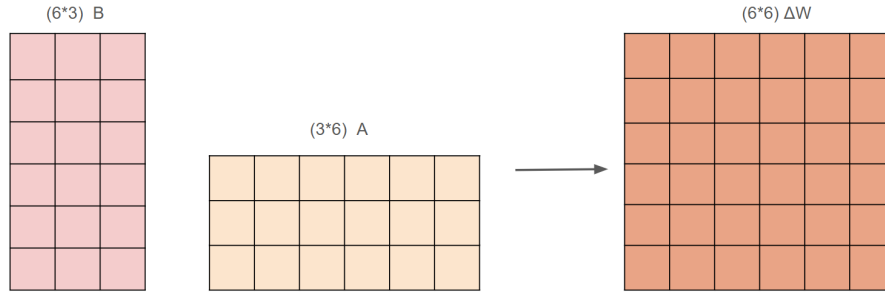


Figure 2.15: Illustration of LoRA

dictive accuracy for complex battery degradation dynamics requires large, over-parameterized models, microcontrollers are severely limited by strict Flash, SRAM, and latency budgets. Knowledge Distillation resolves this conflict by allowing a massive, highly accurate Teacher model to transfer its generalized representations and rich output distributions into a compact, structurally independent Student model [21]. To achieve this, Hinton et al. introduced a temperature parameter (T) to the Teacher's softmax output, generating a softened probability distribution. This reveals latent relational patterns within the incorrect predictions, providing the Student with a much richer gradient signal than standard ground-truth targets alone [70].

Knowledge transfer also extends to the network's internal structure; Romero et al. demonstrated that a Student can minimize the mathematical distance to the Teacher's intermediate layers [71]. Because the Student does not have to share the Teacher's topology, KD provides immense architectural flexibility. For State of Health (SoH) estimation, complex degradation trajectories can be mapped offline by massive network ensembles, and then seamlessly distilled into a lightweight architecture custom-built for the computational limits of a Battery Management System (BMS).

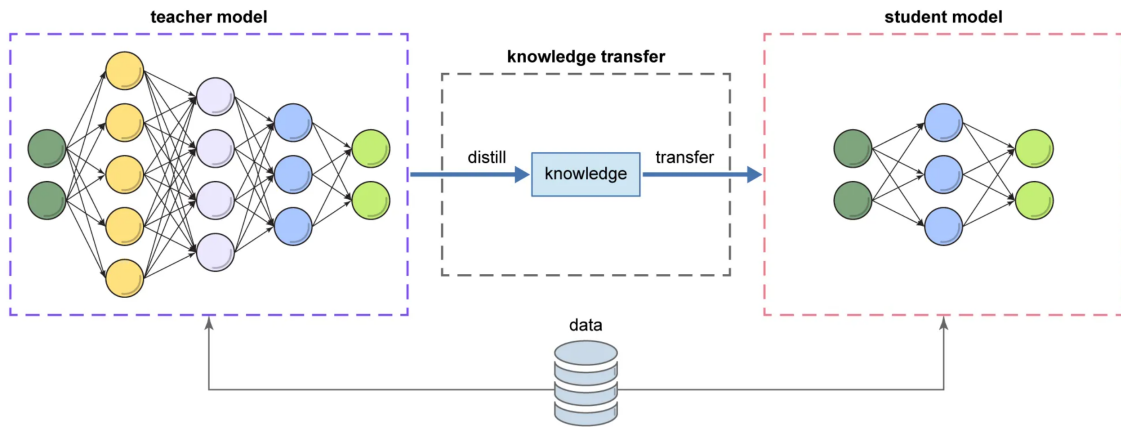


Figure 2.16: Illustration of Knowledge Distillation [7]

2.3.5 Neural Architecture Search (NAS)

Combining multiple compression techniques creates a design space too vast for manual trial-and-error. Neural Architecture Search (NAS) automates this optimization by using machine learning to systematically discover architectures that outperform human-designed models. As defined by Elsken et al., NAS relies on a search space of possible operations, a search strategy (Reinforcement Learning) to explore it, and a performance estimation strategy to evaluate candidates [8]. When applied to compression, NAS dynamically tailors networks to exact hardware budgets. For example, He et al. developed AutoML for Model Compression (AMC), which uses an agent parameterized by target latency and memory limits to automatically output bespoke, layer-by-layer pruning ratios [72].

To bypass the immense computational cost of training every candidate from scratch, advanced NAS employs predictive modeling such as Bayesian Optimization to estimate the accuracy of randomly shrunk architectures, ensuring only the most promising models undergo full evaluation [73]. Ultimately, for real-time State of Health (SoH) estimation, NAS eliminates edge-deployment guesswork. By encoding the exact RAM and processing constraints of a Battery Management System (BMS)/Embedded systems directly into the search space, NAS automatically engineers a custom neural network that balances accurate degradation tracking with the physical limits of automotive hardware.

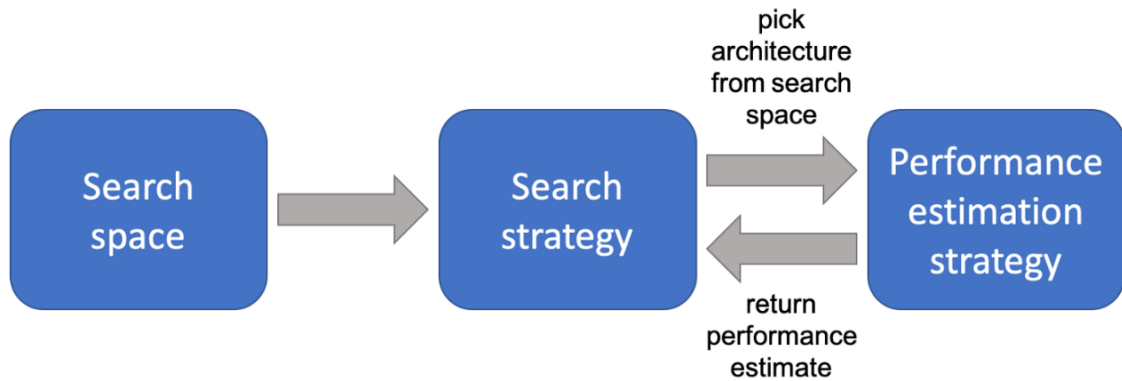


Figure 2.17: Illustration of Neural Architecture Search[8]

2.4 Embedded AI Deployment Frameworks

Deploying compressed models onto edge hardware requires specialized software eco systems to bridge the gap between high-level training environments and bare-metal microcontrollers. These toolchains convert pre-trained models into statically allocated C/C++ code, bypassing the need for dynamic memory allocation or an underlying operating system.

For broad hardware compatibility, models are often exported to intermediate representations like ONNX and optimized via compilers like Apache TVM. On ultra-

low-power devices, TensorFlow Lite for Microcontrollers (TFLM) serves as the industry standard. TFLM executes models on microcontrollers with mere kilobytes of memory, minimizing runtime binary size while natively supporting INT8 quantized operations [74]. However, TFLM relies on an interpreter-based architecture where a micro-runtime parses a serialized model flatbuffer during execution. While highly compatible, the interpreter introduces inherent parsing overhead and limits the deep fusion of sequential mathematical operations.

For automotive Battery Management Systems (BMS), engineers rely on proprietary frameworks—such as STMicroelectronics STM32Cube.AI or NXP’s eIQ—to explicitly exploit specific microcontroller architectures. These tools automatically map network layers to internal DSP instructions and memory hierarchies [75]. Crucially for real-time State of Health (SoH) estimation, they enable precise pre-deployment validation of flash memory footprints and execution latencies, ensuring the AI safely coexists with critical battery control protocols on the same chip.

To achieve maximum optimization and bypass the limitations of interpreter-based runtimes, modern compiler-based frameworks like Apache TVM fundamentally lower the neural network into an intermediate representation (IR), subsequently generating heavily optimized, hardware-specific source code before deployment. Modern deployments utilizing Apache TVM leverage the latest **Relax** framework, which supersedes the legacy Relay architecture. Relax introduces first-class support for dynamic shapes and highly customized computational graphs, which is particularly beneficial when handling specialized W8A32 quantization strategies where weights and activations operate in mixed precisions.

For bare-metal microcontrollers, TVM utilizes an Ahead-of-Time (AOT) compilation strategy combined with the Relax Virtual Machine (VM) execution engine to generate a self-contained, static C application. A critical advancement in this runtime is the modernized `tvm.nd.array` Application Programming Interface (API). By mapping input features and long-term LSTM hidden states directly through these ND arrays, the framework drastically reduces data movement overhead and ensures deterministic memory access during real-time inference.

2.5 Real-Time Embedded Systems and Concurrency

The translation of a deep learning model into C code is only a partial solution for automotive Battery Management Systems (BMS). The generated AI engine must be integrated into a rigorous real-time embedded ecosystem where computational determinism, strict latency deadlines, and fault tolerance are mandatory.

2.5.1 Bare-Metal Execution and Static Memory Management

In critical automotive edge applications, software is often deployed in a "bare-metal" configuration executing directly on the MCU without the scheduling overhead or

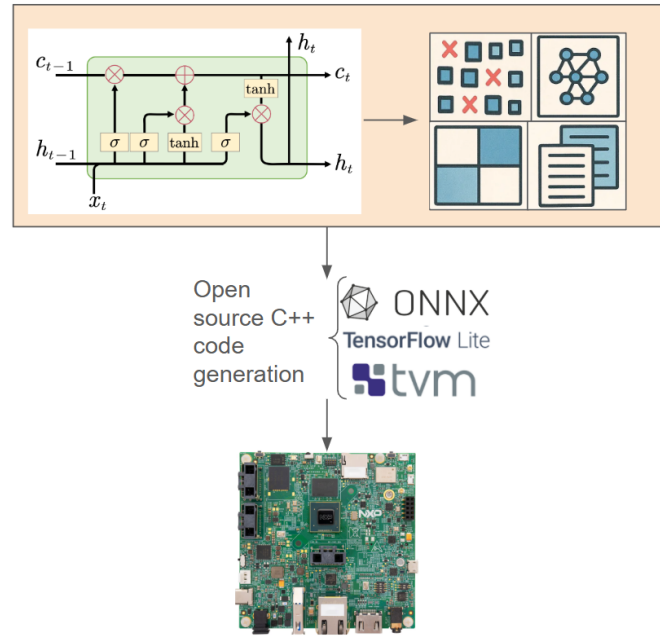


Figure 2.18: Illustration of Frameworks workflow [3]

context-switching jitter of a Real-Time Operating System (RTOS). This ensures that control loops and inference tasks execute with absolute temporal predictability.

A fundamental constraint in bare-metal safety-critical systems is the prohibition of dynamic memory allocation. Dynamic allocation introduces unpredictable execution times and highly risks heap fragmentation, which can cause the system to unexpectedly run out of memory (Out-Of-Memory fault) during continuous operation. To satisfy this requirement, neural network deployments must employ static memory management. Techniques such as a *Static Bump Allocator* reserve a contiguous, highly aligned block of SRAM at compile time. During inference, temporary activation tensors are sequentially allocated from this static pool, and the entire pool is instantly reclaimed by resetting a single offset pointer at the end of the execution step, guaranteeing mathematical safety and absolute memory stability.

2.5.2 Asynchronous Data Streams and Double-Buffering

In a real-world BMS, operational data (voltage, current, temperature) is acquired asynchronously via high-speed vehicular networks such as the Controller Area Network (CAN) bus. These data packets arrive as hardware interrupts. A critical architectural challenge arises because the AI inference process—which requires millions of clock cycles—cannot block the rapid execution of the Interrupt Service Routine (ISR) capturing the sensor data. Conversely, the ISR cannot be allowed to overwrite the sensor buffers while the AI model is actively reading them, which would cause severe data corruption (Race Conditions).

To decouple asynchronous data acquisition from synchronous AI computation, a *Double-Buffering* (or Ping-Pong buffering) architecture is utilized. This concurrency control mechanism employs two identical memory arrays. The ISR writes incoming

sensor data exclusively into the "active" buffer. Simultaneously, the main application loop reads from the "ready" buffer to execute feature extraction and LSTM inference. Once the active buffer is filled and the inference on the ready buffer is complete, the roles of the two buffers are atomically swapped using strict concurrency locks. This ensures zero data loss during high bus loads and isolates the computational bottlenecks from the communication interfaces.

2.5.3 Functional Safety and Mathematical Fault Isolation

Edge AI algorithms are inherently brittle to corrupted inputs. In automotive environments, electromagnetic interference (EMI), sensor degradation, or communication glitches can easily introduce invalid numerical values—such as Not-a-Number (NaN) or Infinity (Inf)—into the data stream.

For recurrent architectures like LSTMs, this vulnerability is amplified. Because the network's hidden state and cell state are recursively fed back into the model at every time step, a single NaN value injected by a faulty sensor will mathematically poison the entire long-term memory state. This corruption inevitably cascades, causing the State of Health (SOH) predictions to collapse entirely. Consequently, robust embedded implementations must enforce strict boundary enforcement and mathematical fault isolation. All raw features must be sanitized through limit-bounding (e.g., Min-Max clamping) and explicit floating-point validation functions prior to inference. Furthermore, the generated output states must be verified for mathematical finiteness before being authorized to overwrite the LSTM's feedback buffers for the subsequent sequence step.

3

Methodology

Figure 3.1 illustrates the end-to-end experimental workflow. The pipeline begins with data partitioning into training and test sets, followed by battery-specific feature extraction and selection. Subsequently, a compressed deep learning model is trained to ensure high State of Health (SoH) prediction accuracy. Finally, the optimized model is deployed onto target microcontroller hardware to jointly assess predictive performance and resource efficiency.

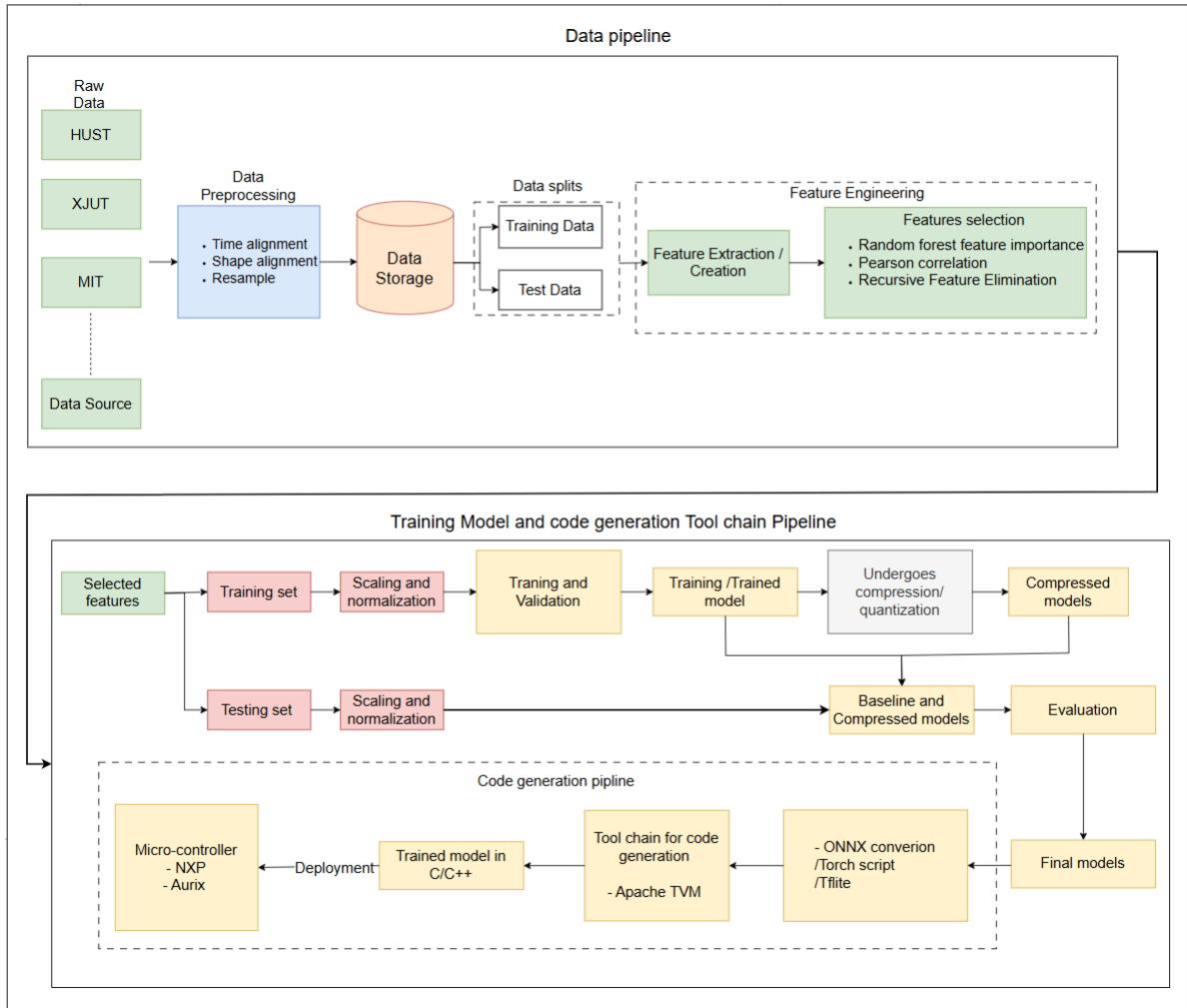


Figure 3.1: Illustration of Frameworks workflow

3.1 Data source and Data preparation

Developing a robust and generalizable model relies heavily on data quality. This section details the dataset sources and data preparation process, including the pre-processing and feature engineering pipelines.

3.1.1 Data source

As the core aim of this thesis is to develop a highly generalized model that remains completely chemistry-agnostic, condition-agnostic with respect to varying operating temperatures and current profiles. A comprehensive data collection strategy was required. The gathered data consist of cyclic aging diverse range of operational profiles, including constant-current constant-voltage (CCCV) charge and discharge cycles, pulse tests and standard dynamic drive cycles. These datasets are collected from various openly available public repositories and institutions, which include three primary lithium-ion battery chemistries: Lithium Nickel Manganese Cobalt Oxide (NMC), Lithium Nickel Cobalt Aluminum Oxide (NCA), and Lithium Iron Phosphate (LFP).

A comprehensive breakdown of these sources detailing the specific chemistries, the total number of batteries acquired from each institution, and the final count of viable and non-damaged batteries validated for training and profile information which is presented in Table 3.1.

Table 3.1: Summary of battery datasets used in this work.

Dataset	Chem.	Cells	Profile
ECOLECENTRALELYON2024_WLTP LFP		13	WLTP driving charge and discharge cycles
ECOLECENTRALELYON2024_CC	LFP	3	Constant current (CC)
ATTIA2020TESTSC	LFP	188	Fast charge + CC discharge
ATTIA2020	LFP	44	Fast charge + CC discharge
MIT	LFP	138	–
HUST	LFP	77	Constant-current cycling
LUHBLANK2024_CYCLIC	NMC	96	Constant-current cycling
LUHBLANK2024_PROFILE	NMC	31	Dynamic WLTP
XJTU	NMC	44	Fast charge + CC discharge
SCHMITT2024	NMC	10	WLTP driving profile
RWTH	NMC	24	–
GESLIN2025	NCA	92	Driving cycle, pulse discharge, CC charging
NASA	NCA	16	–
GRAZ2023	NCA	87	Dynamic load profile
STROEBL	NCA	147	CC cycling

3.1.2 Data preparation

The objective of the proposed methodology is to establish an early-life forecasting approach for State of Health (SoH) prediction[76]. The raw input data consists of the three primary parameters directly sensed from the battery voltage, current, and temperature while the target variable is SoH. To develop a robust deep learning model that operates independently of explicit cycle aging information, the model must rely strictly on these three physical signals. Therefore, transforming the raw data into a unified, mathematically consistent format is a prerequisite for effective data-driven training [77].

To achieve this, the data undergoes a process to reshape, realign, and resample the raw measurements. Because the datasets are from various independent laboratories, they are sampled at different frequencies. To resolve this inconsistency, all datasets are resampled with an identical number of sample points [78]. This resampling aligns batteries with vastly different lifespans onto a uniform sample point grid, ensuring perfectly consistent feature dimensions across the entire training process.

Additionally, raw sensor data usually picks up noise and errors from the hardware. A two-step filtering process is used to clean the time-based data before extracting key features. First, a median filter is applied to catch and remove sudden, sharp spikes in the readings [79].

For a given time-series sequence x , the median-filtered output y at time step n utilizing a sliding window of size $2k + 1$ is defined mathematically as:

$$y[n] = \text{median} (x[n - k], \dots, x[n], \dots, x[n + k]) \quad (3.1)$$

Next, a Savitzky-Golay filter is used to smooth out any remaining background noise [80]. Its equation is given as:

$$y_i = \sum_{j=-m}^m C_j x_{i+j} \quad (3.2)$$

This specific filter is highly valuable because it smooths the signal while keeping the exact shape, height, and width of the data's peaks and waves. Preserving these physical details is necessary to accurately calculate advanced statistics features for early life prediction [81].

Target side:

To ensure consistency across all datasets, every battery's entire lifespan was resampled to exactly 200 cycle points. Regardless of how many physical cycles a battery actually completed during testing, its full degradation history is compressed into this fixed 200-point window. This standardization gives the neural network a uniform input size, ensuring the model learns the true physical signs of battery aging rather than simply counting the number of cycles. And the series of figures demonstrates some examples of raw data to preprocessed data.

Input side:

The input data of the model to the network and mapping the input to the target

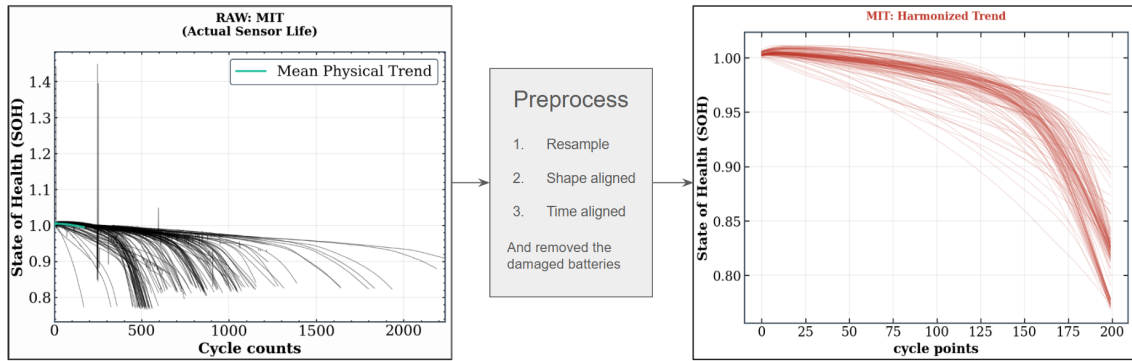


Figure 3.2: Illustration of preprocessed data of MIT Lab (LFP)

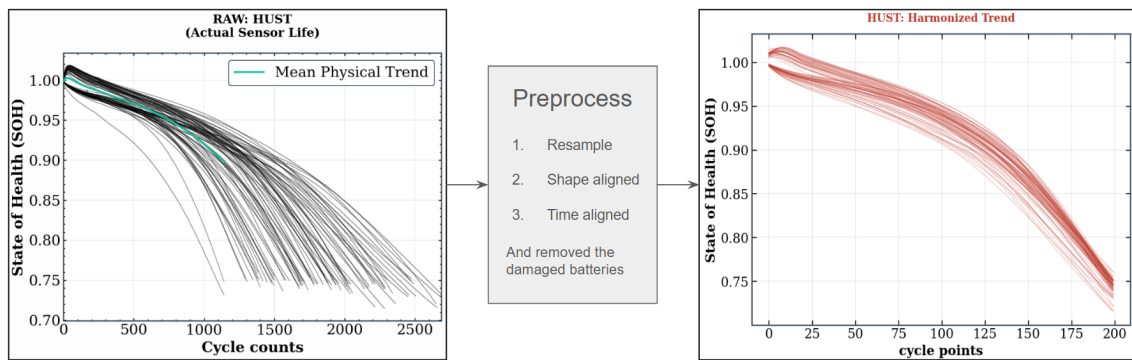


Figure 3.3: Illustration of preprocessed data of HUST Lab(LFP)

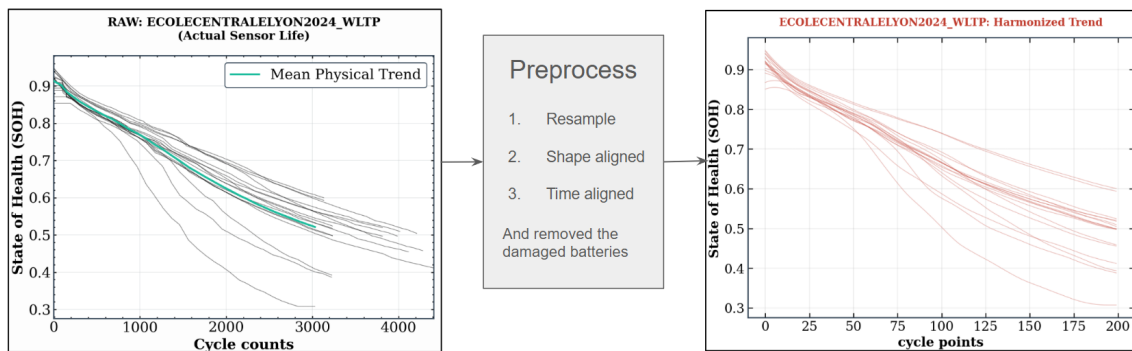


Figure 3.4: Illustration of preprocessed data of ECO Lab(LFP)

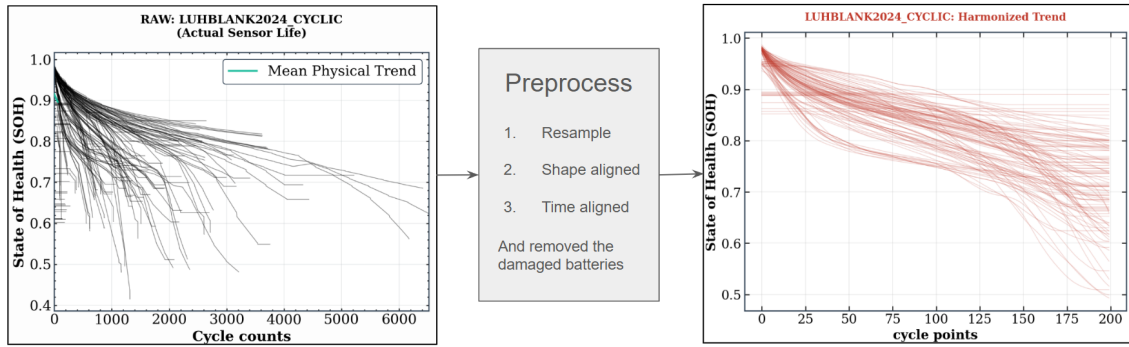


Figure 3.5: Illustration of preprocessed data of LUHBLANK2024cyclic Lab(NMC)

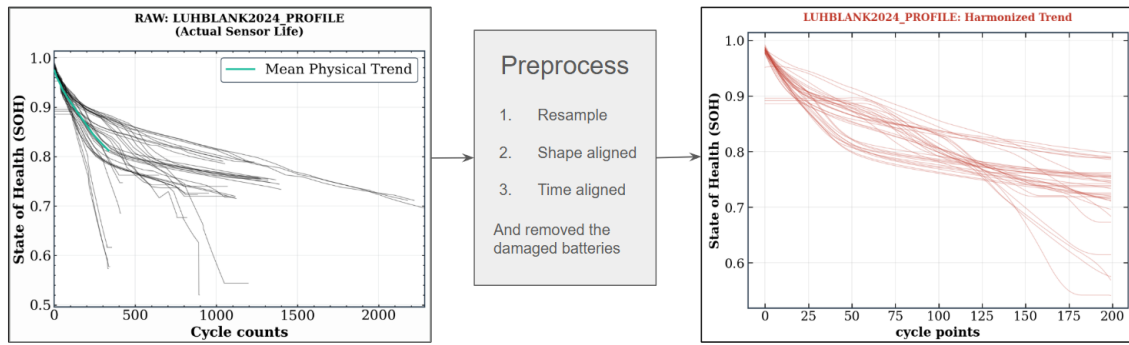


Figure 3.6: Illustration of preprocessed data of LUHBLANK2024profile Lab(NMC)

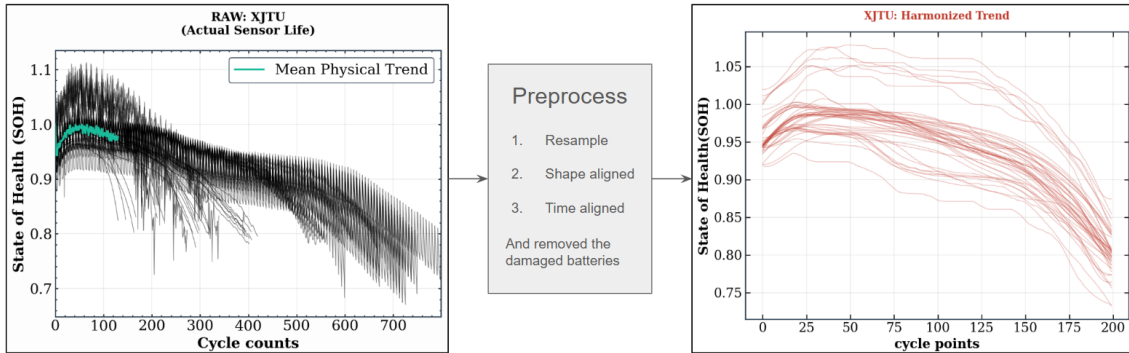


Figure 3.7: Illustration of Raw data and preprocessed data of XJTU Lab(NMC)

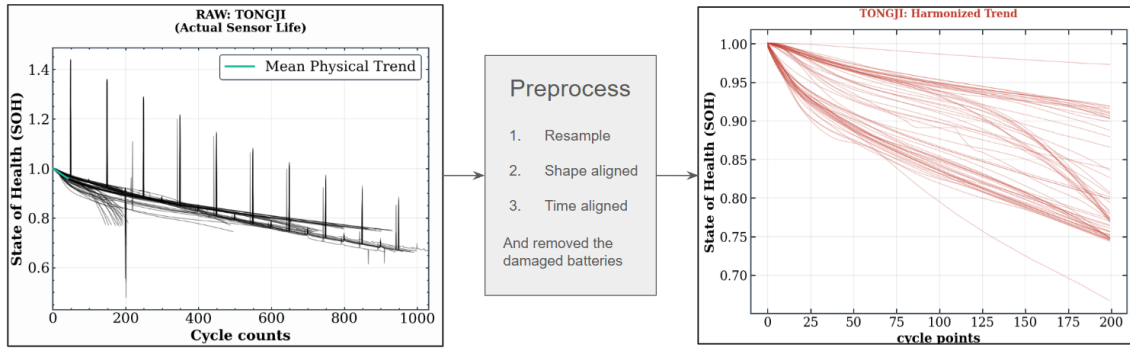


Figure 3.8: Illustration of Raw data and preprocessed data of TONGJI Lab(NCA)

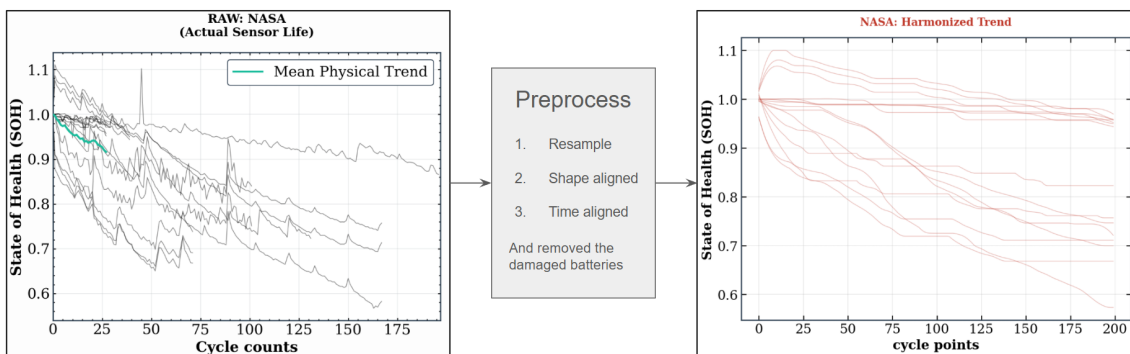


Figure 3.9: Illustration of Raw data and preprocessed data of NASA Lab(NCA)

is built specifically around the earliest cycles of the battery. The network takes the voltage (V), current (I), and temperature (T) readings strictly from the first 10 cycles of the battery's life. Within this standardized preprocessing setup, the data for every individual cycle is sampled to 200 points. Therefore, combining the first 10 cycles creates a continuous sequence of exactly 2,000 data points for each physical parameter (V, I, and T). This establishes a clear mapping strategy: a single block of 10 cycles represented by these 2,000 sample points of V, I, and T serves as the complete input used to predict one corresponding State of Health (SoH) target value.

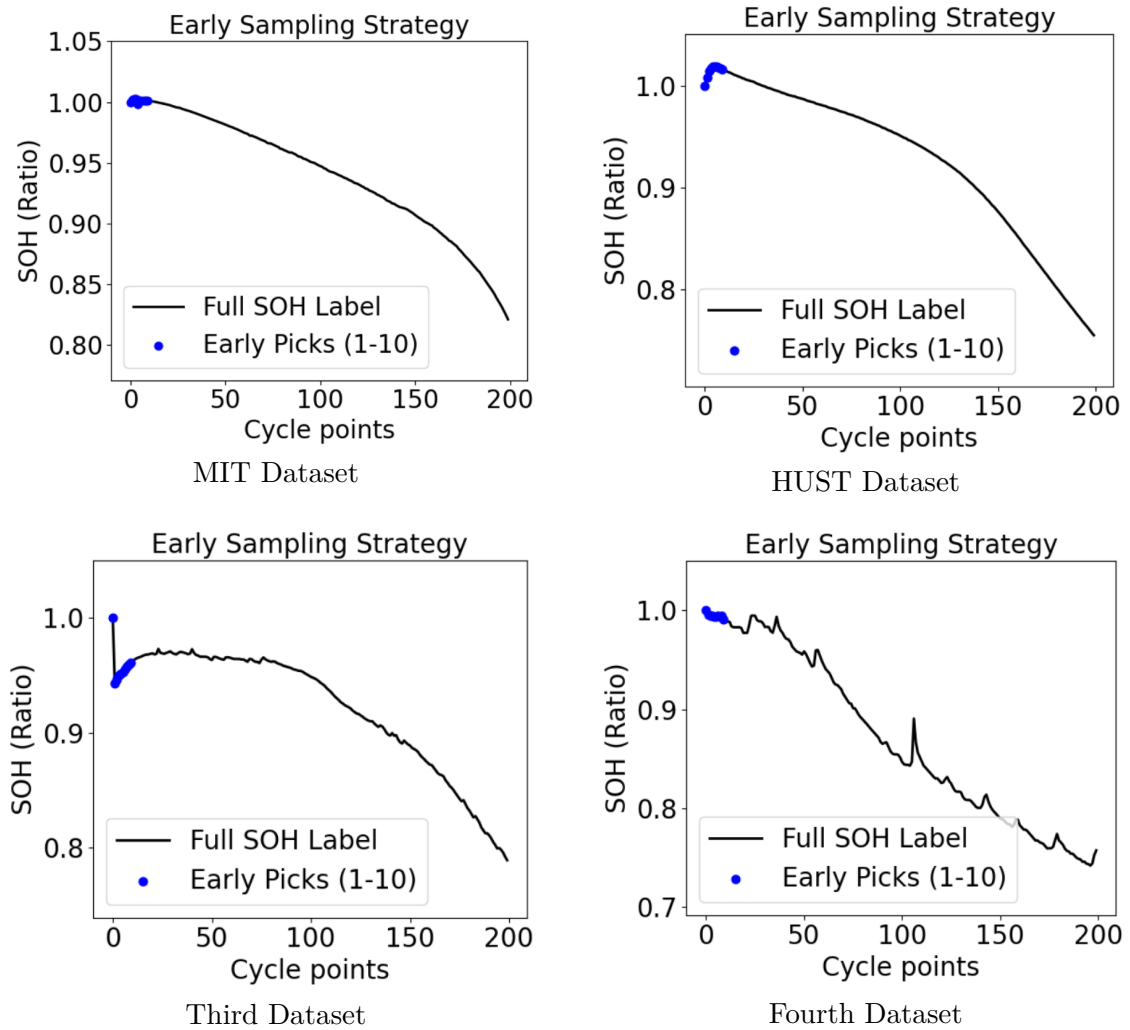


Figure 3.10: Comparison of early-cycle data from different datasets.

The specific data extracted from the initial 10 cycles is highlighted in blue in Figure 3.10. And figures 3.11 through 3.14 provide further examples of the extracted input data profiles across different batteries.

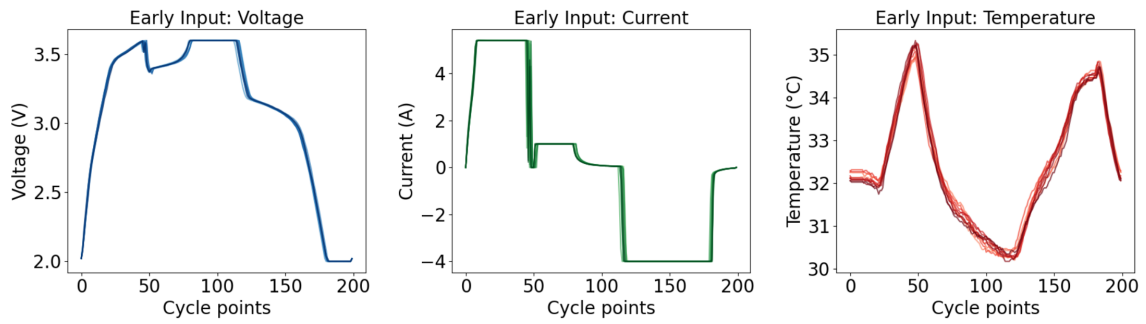


Figure 3.11: Illustration of input data sampling, LAB:MIT

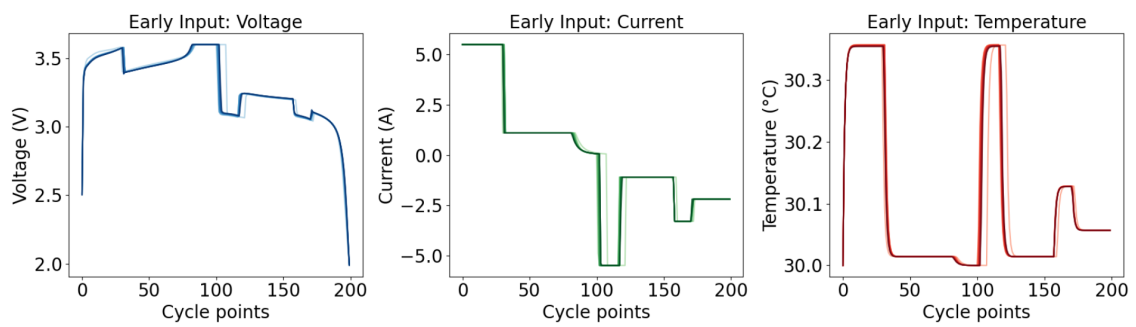


Figure 3.12: Illustration of input data sampling, LAB:HUST

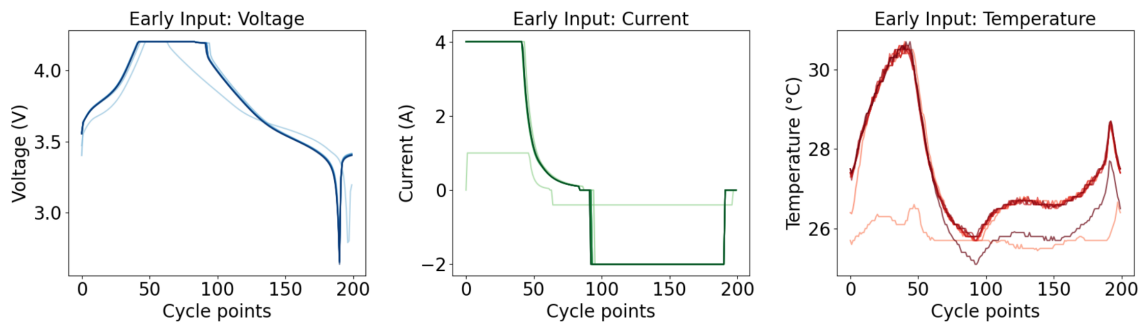


Figure 3.13: Illustration of input data sampling, LAB: XJTU

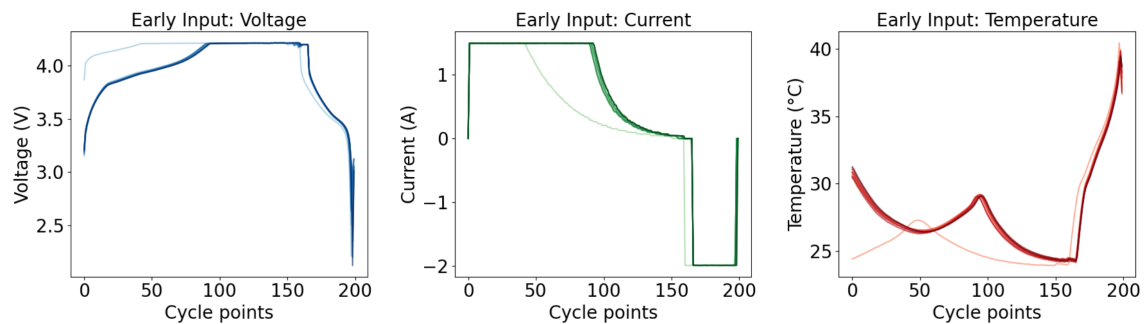


Figure 3.14: Illustration of input data sampling, LAB: NASA

3.2 Feature engineering

This process transforms the preprocessed data (shape and time aligned) of charging and discharging profiles into a condensed set of numerical descriptors and statistical features. Rather than relying on raw high-dimensional inputs, this step extracts over 28 specific features that effectively capture the underlying mechanisms of health degradation [3].

3.2.1 Feature extraction

This step computes specific physical, statistical, and informational metrics from the temporal data profiles[3][5]. The extraction processes for the primary feature categories are detailed below:

Energy, Capacity, and Power Metrics:

These metrics quantify the fundamental energy transfer characteristics and power dynamics within the sampled window.

Window Capacity (Q_{window}): Calculated by integrating the absolute current over the selected window (scaled for Ah conversion):

$$Q_{\text{window}} = \sum \frac{|I_t| \cdot \Delta t}{3600}, \quad (3.3)$$

Window Energy (E_{window}): Calculated by integrating the product of voltage and absolute current over the selected window:

$$E_{\text{window}} = \sum \frac{|V_t \cdot I_t| \cdot \Delta t}{3600}, \quad (3.4)$$

Power Variance: Measures the fluctuation in power delivery, defined as the variance of the instantaneous power:

$$\text{Var}(P) = \text{Var}(V \cdot I), \quad (3.5)$$

Statistical Indicators: Four primary statistical features are derived to quantify the signal distributions from a mathematical perspective. These include the signal mean ($\bar{x}$), standard deviation (σ), skewness, and kurtosis

Mean:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x^{(i)} \quad (3.6)$$

Standard Deviation:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x^{(i)} - \bar{x})^2} \quad (3.7)$$

Skewness:

$$\text{Skewness} = \frac{\sum_{i=1}^n (x^{(i)} - \bar{x})^3}{(n-1)\sigma^3} \quad (3.8)$$

Kurtosis:

$$\text{Kurtosis} = \frac{\sum_{i=1}^n (x^{(i)} - \bar{x})^4}{(n-1)\sigma^4} \quad (3.9)$$

voltage entropy:

$$H(V) = - \sum_{i=1}^n P(v_i) \log_2 P(v_i) \quad (3.10)$$

Minimum and Maximum Parameters (V, I, T):

To capture the operational boundaries within each sampled window, the peak and minimum values are extracted for the voltage, current, and temperature profiles.

Voltage:

$$V_{\min} = \min(V), \quad V_{\max} = \max(V) \quad (3.11)$$

Current:

$$I_{\min} = \min(I), \quad I_{\max} = \max(|I|) \quad (3.12)$$

where the absolute magnitude is used to capture the peak current flow.

Temperature:

$$T_{\min} = \min(T), \quad T_{\max} = \max(T) \quad (3.13)$$

3.2.2 Feature selection

While the feature extraction process yields over 28 distinct features, not all of them contribute meaningfully to the final prediction. Considering the strict computational limitations of the target Microcontroller Unit (MCU) as well as the critical need to prevent overfitting and eliminate redundant information, a three step feature selection pipeline is implemented to isolate the most robust predictors for State of Health (SoH) forecasting.[82]:

As mentioned, this selection pipeline isolates the optimal variables through a combination of three complementary techniques:

Pearson Correlation Analysis: This initial step identifies and eliminates multicollinearity among the extracted features. Features exhibiting a high linear correlation with one another are filtered out to ensure the final set inputs contains unique and independent markers, which reduces unnecessary clutter[83]. Mathematically, the Pearson correlation coefficient r_{xy} evaluates the linear relationship between two feature vectors, x and y , of length n

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \quad (3.14)$$

In the equation, $\bar{x}$ and $\bar{y}$ represent the average values of those features. If the correlation score between any two features goes above a specific limit, they are considered too similar. To keep the data completely independent, one of these overlapping features is dropped.

Maximal Information Coefficient (MIC): Pearson correlation is well suited to linear dependencies, while battery degradation is governed by highly non-linear electrochemical dynamics. To capture such complex relationships, the Maximal Information Coefficient (MIC) is incorporated into the pipeline [84]. MIC is a powerful statistical tool to detect and measure the strength of linear and non-linear associations between a single feature and the target variable (SoH). It does this by overlaying a grid on the scatterplot of the two variables and computing the mutual information at multiple resolutions of the grid. The MIC score is the maximal mutual information, normalized to a scale between 0 (no relationship) and 1 (deterministic relationship):

$$\text{MIC}(x, y) = \max_{a \times b < B(n)} \frac{I^*(x, y, a, b)}{\log_2(\min(a, b))} \quad (3.15)$$

Here, $I^*(x, y, a, b)$ is the maximum mutual information achieved for a grid of a columns and b rows, and $B(n)$ represents the maximum allowed grid size as a function of the sample size n (typically $B(n) = n^{0.6}$). Features that exhibit a low MIC score with the SoH are discarded, ensuring that only variables possessing a strong underlying physical or statistical relationship to battery degradation are retained [85].

Random Forest Feature Importance: Following the correlation and non-linear filtering, a Random Forest algorithm scores the remaining features based on their predictive value for the battery's State of Health (SoH) [86]. It measures how well each feature reduces uncertainty (variance) across the model's decision tree. This reduction is called the Mean Decrease in Impurity (MDI) and is calculated at any splitting node τ as:

$$\Delta I = p(\tau)I(\tau) - p(\tau_L)I(\tau_L) - p(\tau_R)I(\tau_R), \quad (3.16)$$

Here, $p(\tau)$ is the fraction of data reaching the node, and $I(\tau)$ represents the current impurity. The L and R subscripts denote the data proportions passed to the left and right branches. A feature's final importance score is simply the sum of these impurity reductions across the entire forest.

3.3 Model Development

This section details the development, training, and evaluation of the predictive model used to estimate the battery's State of Health (SoH).

3.3.1 Baseline Model Architecture

A Long Short-Term Memory (LSTM) network was selected as the baseline architecture. LSTMs were explicitly chosen to address the primary research question due

to its success and robustness in battery state estimation tasks [87]. Unlike standard feedforward neural networks, LSTMs utilize internal memory mechanisms, specifically input, forget, and output gating units, that make them exceptionally capable of capturing the longterm temporal dependencies and sequential degradation patterns inherent in lithium ion battery cycling.

Although LSTMs are very accurate at predicting trends over time, their complex design usually requires a lot of memory and processing [88]. Since the main goal of this research is to shrink and optimize the model so it can run on small devices, this standard LSTM acts as our starting point for comparison. It is carefully built to find a balance between function of model and the limited hardware resources of the target Microcontroller Unit.

Architecturally, the baseline network begins with an input layer specifically structured to accept the time-series sequences of the final optimized feature subset. This structured data is then passed into one or more LSTM layers designed to process the temporal dynamics of the battery’s electrochemical behavior over the recorded cycles. Finally, a fully connected (dense) output layer maps these extracted temporal features to a single, continuous numerical State of Health (SoH) prediction [89].

3.3.2 Model Training

This section outlines the methodological approach utilized to train the LSTM network, focusing on the dataset Splitting Strategy, selected loss function, the hyperparameter optimization strategy and the implementation of early stopping to ensure generalization.

Dataset Splitting Strategy: Before optimizing the network weights, the battery dataset needs to be divided carefully. Since State of Health (SoH) forecasting involves interconnected time-series data, a standard random split can easily lead to data leakage. If individual measurements are randomly shuffled, data from the same physical battery may end up in both the training and testing sets. In that case, the model could simply memorize the degradation pattern of a particular battery instead of learning aging patterns that can generalize to unseen batteries [21].

To avoid this issue, a two-stage splitting strategy is used at the Battery ID level.

First, the dataset is split into a 95% training and validation pool and a fixed 5% test set. Importantly, the ID’s are selected in test set in such a way that it contains battery IDs from every laboratory represented in the dataset. These batteries are kept completely separate from the training and validation data. This ensures that the final evaluation is performed on batteries the model has never seen before.

The remaining 95% of the batteries from all sources are combined into a single pool. To minimize the risk of the model learning patterns specific to a particular

laboratory or chronological order, the battery IDs in this pool are thoroughly shuffled. This creates a more mixed training sequence, where batteries from different sources are interleaved rather than being processed laboratory by laboratory. For example, a battery from one facility may be followed by batteries from several other laboratories.

After shuffling, a Stratified Group K-Fold cross-validation strategy is applied to the 95% pool for model training and hyperparameter tuning, including the early stopping process. This strategy is designed to satisfy three important requirements: group isolation, laboratory-level stratification, and rotation between folds.

First, strict group isolation is maintained by using the unique Battery ID as the grouping variable. This ensures that all measurements from a given battery remain entirely within either the training or validation set and are never split between the two [90]. Second, the laboratory information is used to stratify the folds so that each fold contains a balanced and representative proportion of batteries from the different laboratories. This prevents any individual fold from being dominated by data from a particular source. Finally, the training and validation sets are rotated across the different folds. For example, in one fold, a specific subset of batteries is used for training while the remaining batteries are used for validation. In the next fold, these groups are reassigned. This process allows every battery in the 95% pool to contribute to both training and validation at different stages, while still maintaining strict separation within each fold.

Overall, this combined strategy helps prevent data leakage, supports reliable hyperparameter tuning, and provides a more balanced and robust assessment of the model's ability to generalize to unseen batteries [91].

Loss Function and Mini-Batch Training: During the training phase, the machine learning model relies on an objective loss function to continuously evaluate its predictions against the actual expected output. The network is trained over multiple iterations, known as epochs, where one epoch represents a complete pass through the entire training dataset. After each pass, the network's internal parameters are updated based on the calculated loss to incrementally improve prediction accuracy for the next iteration. For this forecasting task, the error between the target State of Health (SoH) and the model's output is measured using the Mean Squared Error (MSE) [92].

The objective function is defined using the Mean Squared Error (MSE), which measures the average squared difference between the predicted and actual State of Health (SoH) values:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3.17)$$

where n is the total number of samples, y_i denotes the actual SoH value of the i th sample, and $\hat{y}_i$ represents the corresponding SoH value predicted by the model.

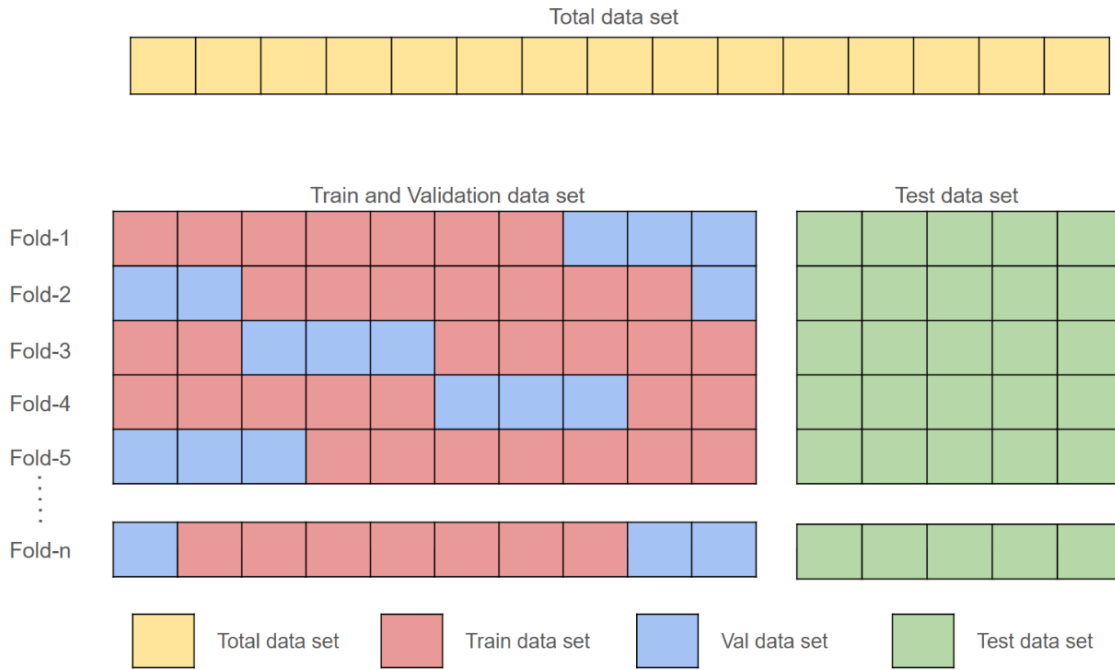


Figure 3.15: Illustration of data split strategy

However, instead of calculating the MSE once per epoch over the entire dataset, the data is randomly divided into smaller subsets called mini-batches. The size of these subsets is dictated by the batch-size hyper-parameter. Calculating the loss and updating the network weights per mini-batch rather than after the full dataset is processed, allows for more frequent and granular adjustments. This approach mimics the stabilizing effect of a smaller learning rate, ultimately improving the robustness and efficiency of the model's convergence [93].

Hyper-parameter Optimization: To maximize the model's predictive performance, it is crucial to determine the optimal configuration of its hyper-parameters. These parameters govern both the physical architecture of the network, such as the number of hidden layers and the amount of LSTM units per layer and the training mechanics, including the learning rate and batch size [94]. Because the ideal configuration is rarely known in advance, a defined range of potential values must be investigated for each variable. To accomplish this, a grid-search algorithm is employed. This method systematically iterates through a defined grid of parameter ranges, and does evaluate the model's performance for every possible combination to isolate the perfect parameter that achieves the best overall evaluation score.

Early Stopping Mechanism: A common challenge when training data-driven models over extended periods is the risk of overfitting, where the network begins to memorize the training data rather than learning generalizable degradation patterns [95]. To mitigate this effect and simultaneously reduce unnecessary training duration, an early stopping mechanism is integrated into the training routine [96]. Instead of forcing the model to complete a fixed, pre-defined number of epochs, the MSE loss on the dataset is continuously monitored.

Because the primary objective during training is to minimize this loss, the process can be safely halted when the model ceases to make meaningful improvements. Specifically, if the absolute total loss error difference falls below a minimal threshold of 0.001 for 3 consecutive epochs, the training is automatically stopped. This ensures the model halts and prevents it from overfitting to the training data [92].

3.3.3 Model Evaluation

Following the training and hyper-parameter optimization phases, the generalized performance of the LSTM model is ultimately tested using the entirely unseen fixed testing set. Evaluating the model strictly on this isolated subset ensures an objective measurement of how accurately the algorithm can predict the State of Health (SoH) of batteries it has never encountered [97],[98]. To provide a comprehensive assessment of the predictive accuracy and regression performance of the model, three standard statistical metrics are used: the root mean square error (RMSE), the mean absolute error (MAE), and the Coefficient of Determination (R^2) [99],[100].

The Root Mean Square Error (RMSE) is utilized as a primary evaluation metric because it measures the standard deviation of the prediction errors. By squaring the errors before averaging them, RMSE gives disproportionately higher weight to larger deviations. In the context of battery health monitoring, heavily penalizing large estimation errors is critical, as a significant overestimation of the SoH could lead to unexpected system failures in real-world applications [101], [102]. The RMSE is mathematically defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (3.18)$$

where n represents the total number of samples in the testing set, y_i is the actual expected SoH value, and $\hat{y}_i$ is the model's predicted SoH.

To complement the RMSE, the Mean Absolute Error (MAE) is also calculated. Unlike RMSE, MAE calculates the average magnitude of the errors without considering their direction, treating all individual differences equally. This provides a straightforward and highly interpretable measure of the average absolute discrepancy between the model's predictions and the true battery degradation state [103]. The formula for MAE is given by:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (3.19)$$

Furthermore, the Coefficient of Determination (R^2) is used to evaluate the proportion of the variance in the target SoH that is successfully explained by the model's input features. An R^2 score approaching 1.0 indicates that the model fits the degradation trends perfectly, whereas a lower score suggests that the model fails to capture the underlying variance of the aging process [104]. The R^2 metric is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (3.20)$$

where $\bar{y}$ represents the mean of the actual expected SoH values.

Finally, because the overarching objective of this research is to develop a deployable embedded model, predictive accuracy is not the sole criterion for success. The model's evaluation also encompasses computational metrics critical for Microcontroller Unit (MCU) integration. Alongside RMSE, MAE, and R^2 , the final evaluation framework measures the model's total parameter count, memory footprint (in kilobytes), and inference latency. This ensures a holistic assessment of how effectively the optimized LSTM balances high-fidelity SoH forecasting with the strict hardware limitations of edge devices [105][106].

3.4 Model compression and Optimization Strategies

3.4.1 Grow prune Train

Training standard neural networks usually are way too big and heavy for embedded systems. To fix this and find a lean, sparse setup that runs well on embedded hardware, we use a dynamic Grow-and-Prune (GP) training method see the figure 3.16. Everything starts by setting up the network, either from a sparse starting point or a fully trained dense model and running a short warm-up period. During this time, standard forward and backward passes happen normally to settle the weights, ensuring that initial gradient flow behaves reliably before any structural changes take place [9]. Following warm-up, network enters GP-Active Phase, causing the

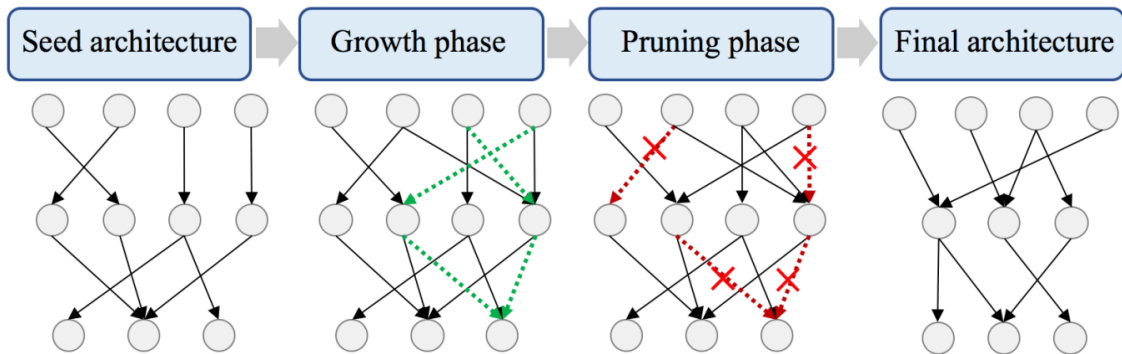


Figure 3.16: Illustration of GP training [9]

architecture to evolve continuously through automated alternating cycles. First, gradient-based growth policy locates effective dormant connections reducing loss

function L . Dormant connection possessing weight w activates strictly when gradient magnitude ($|w.\text{grad}| = |\partial L / \partial w|$) surpasses (100α) -th percentile of gradient magnitudes of the corresponding weight matrix. Targeted growth successfully minimizes loss while assisting algorithm in escaping local minima.

Immediately afterward, magnitude-based pruning policy removes redundancy. Active connections featuring absolute magnitude ($|w|$) falling beneath (100β) -th percentile within layer mask back to zero, and entire neurons undergo pruning upon losing all input or output connections [107].

Varying optimal architectural constraints require active phase governance through multi-dimensional hyper-parameter grid search rather than static variables. Grid systematically iterates across varying target sparsity loops (0.1 to 0.9), sensitivity percentile thresholds determining α and β , plus operational frequency loops. Finally, network transitions into Refinement Phase. Structural growth and pruning operations pause permanently, allowing finalized sparse model to undergo uninterrupted retraining for performance degradation recovery, yielding compact, highly accurate inference architecture [6].

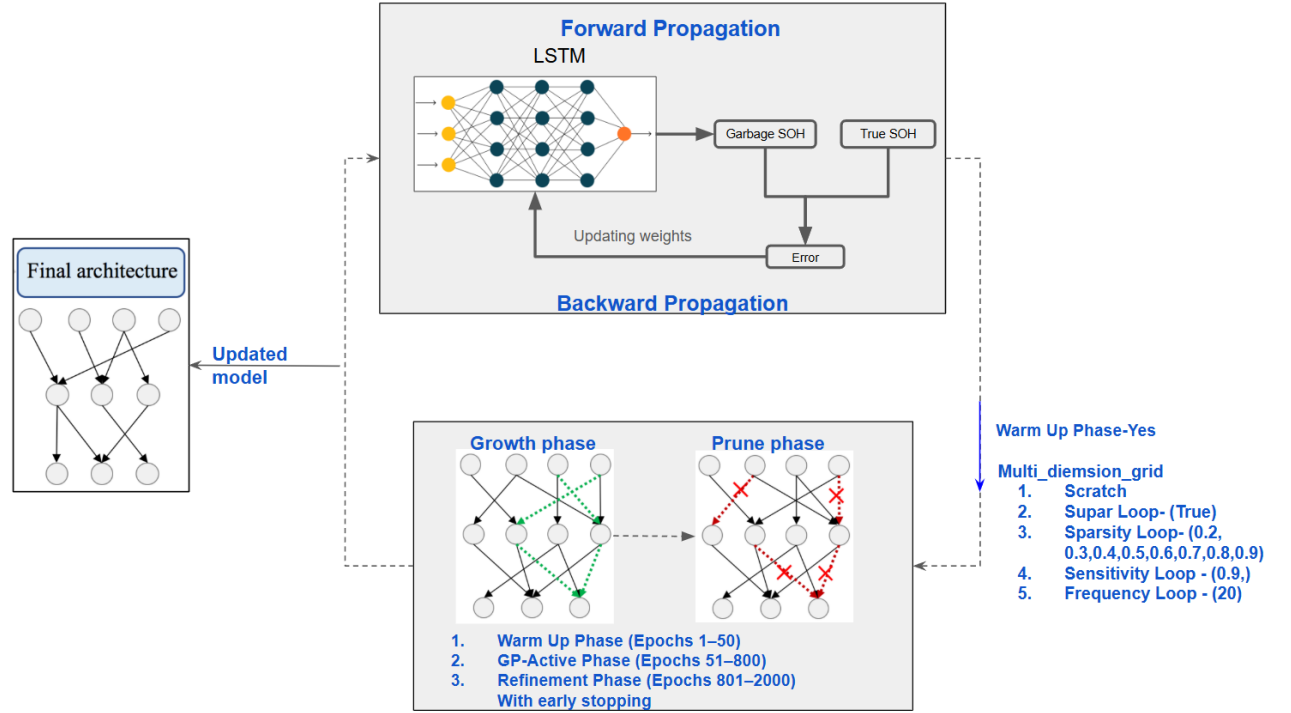


Figure 3.17: Illustration of GP training Framework [9]

3.4.2 Compression aware training

Instead of relying completely on post-training compression, which often leads to severe accuracy degradation, Compression Aware Training explicitly encourages structural compactness during the active optimization process [69]. To ensure the network is inherently compressible, the learning process is formulated as a regularized

minimization problem:

$$\min_{\Theta} \left[\frac{1}{N} \sum_{i=1}^N \mathcal{L}(y_i, f(x_i; \Theta)) + r(\Theta) \right], \quad (3.21)$$

where Θ encompasses all network parameters, $\mathcal{L}(\cdot, \cdot)$ is the supervised loss function, and $r(\Theta)$ is a specialized regularizer designed to enforce both sparsity and low-rank properties within the parameter matrices. To maximize the eventual compression rate, the training algorithm combines two distinct regularization strategies. First, a sparse group Lasso regularizer is utilized to explicitly zero out and remove entire units (such as redundant neurons or gates), which reduces the absolute number of input and output channels [108]. Second, to ensure the surviving parameter matrices can be efficiently compressed, a low-rank regularizer is applied. Because directly minimizing the rank of a matrix is NP-hard, a convex relaxation using the nuclear norm is employed [69], [109]. For a network with L layers, the combined regularizer is mathematically defined as

$$r(\Theta) = \sum_{l=1}^L \left[(1 - \alpha) \lambda_l \sqrt{P_l} \sum_{n=1}^{K_l} \|\theta_l^n\|_2 + \alpha \lambda_l \|\theta_l\|_1 \right] + \tau \sum_{l=1}^L \|\hat{\theta}_l\|_*, \quad (3.22)$$

where α balances the influence of sparsity on entire groups versus individual parameters, λ_l is a layer-wise penalty hyper-parameter, K_l represents the number of units in layer l , and τ dictates the influence of the nuclear norm $\|\hat{\theta}_l\|_*$. The model is optimized using proximal stochastic gradient descent, applying a soft-thresholding operator to the singular values of the weight matrices during training [69].

By mathematically forcing the parameter matrices into a low-rank state during training, the network is perfectly ready for a post-processing Singular Value Decomposition (SVD) step [110]. Once trained, the parameter matrix of each layer is decomposed ($\hat{\theta}_l \approx U_l M_l$). The algorithm truncates the matrices, keeping only the top r_l singular values based on a defined threshold.

This decomposition fundamentally changes how inference is executed on hardware. Instead of a direct, heavy multiplication of weights and inputs ($Y_{pred} = W \cdot X$) that demands a large block of temporary memory, the computation is evaluated sequentially:

$$Y_{pred} = U \cdot (\Sigma \cdot (V^T \cdot X)),$$

By multiplying the input vector through the compressed factors step-by-step ($V^T \cdot X$ first), this completely avoids generating or storing large matrices. This sequential evaluation drastically cuts down temporary workspace requirements in SRAM, lowers active computational latency, and reduces Flash storage.

This dual approach explicitly removing zeroed-out units via group sparsity and decomposing the remaining low-rank matrices via SVD results in massive memory

and computational savings, enabling the complex model to execute efficiently on resource-constrained microcontrollers [69],[110],[69].

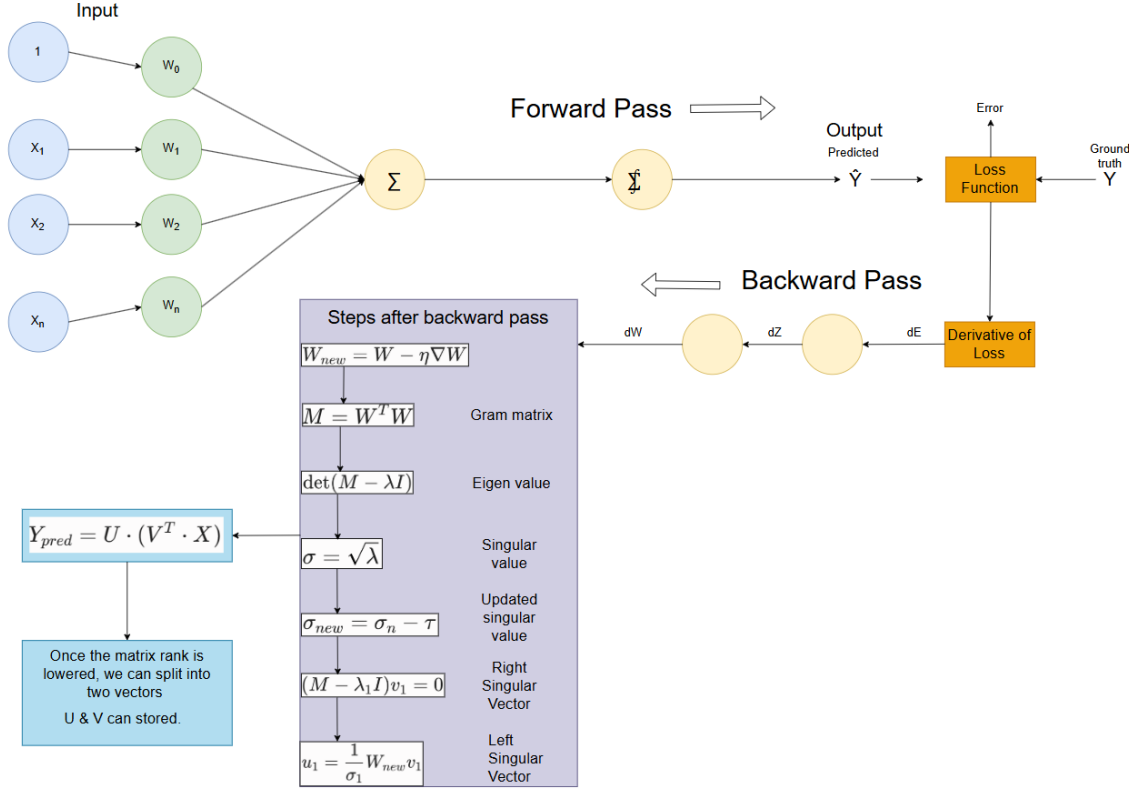


Figure 3.18: Illustration of Compression Aware Training Framework

3.4.3 Relational Knowledge distillation

To compress the model without losing critical predictive capabilities, a hybrid Knowledge Distillation (KD) framework is employed [70].

This strategy transfers knowledge from a highly accurate, high-capacity Teacher model to a lightweight Student model suitable for edge Microcontroller Units (MCUs). Unlike standard point-wise KD, this framework explicitly focuses on transferring structural relationships between different data samples, which allows it to handle latent representations of completely different dimensionalities [111].

Let M denote the batch size. For a given input $\mathbf{X}_i$, the latent representations extracted from the final hidden layer of the Teacher and Student models are denoted as $\mathbf{h}_{T,i}$ and $\mathbf{h}_{S,i}$, respectively. To transfer relational knowledge, the framework computes pairwise squared Euclidean distances and cosine similarities among hidden features within each batch:

$$D_B^T(i, j) = \|\mathbf{h}_{T,i} - \mathbf{h}_{T,j}\|_2^2, \quad D_B^S(i, j) = \|\mathbf{h}_{S,i} - \mathbf{h}_{S,j}\|_2^2, \quad (3.23)$$

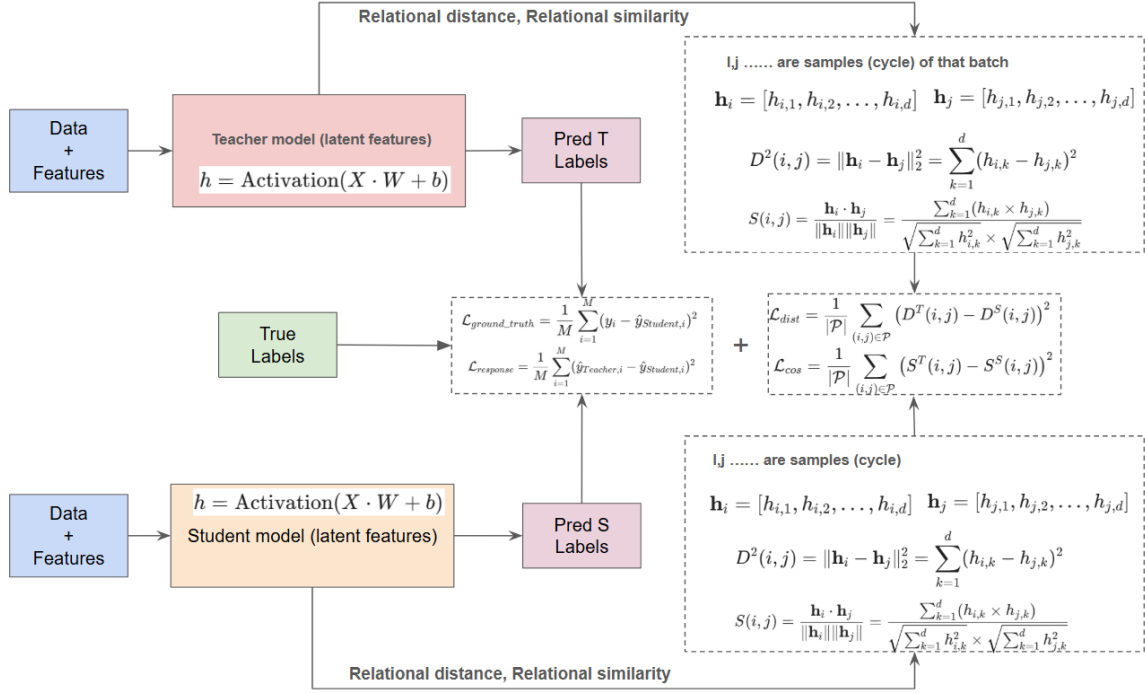


Figure 3.19: Illustration of relation KD

where $\mathbf{h}_{T,i}$ and $\mathbf{h}_{S,i}$ denote the feature representations of the i th sample produced by the teacher and student networks, respectively.

Similarly, the cosine similarity matrices are computed as

$$S_B^T(i, j) = \frac{\mathbf{h}_{T,i}^\top \mathbf{h}_{T,j}}{\|\mathbf{h}_{T,i}\|_2 \|\mathbf{h}_{T,j}\|_2}, \quad S_B^S(i, j) = \frac{\mathbf{h}_{S,i}^\top \mathbf{h}_{S,j}}{\|\mathbf{h}_{S,i}\|_2 \|\mathbf{h}_{S,j}\|_2}, \quad (3.24)$$

where (i, j) denotes the pairwise relationship between the i th and j th samples within a mini-batch.

Because these relationships are transformed into scalars, the original dimensions of the feature vectors become irrelevant, allowing direct comparison. Distillation losses are calculated over a set of non-overlapping random cycle pairs, $\mathcal{P}$, selected from the mini-batch [111]:

The distance-preserving loss is defined as

$$\mathcal{L}_{\text{dist}} = \frac{1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} (D_T(i, j) - D_S(i, j))^2, \quad (3.25)$$

where $\mathcal{P}$ denotes the set of all sample pairs in the mini-batch.

Similarly, the cosine similarity preserving loss is given by

$$\mathcal{L}_{\text{cos}} = \frac{1}{|\mathcal{P}|} \sum_{(i,j) \in \mathcal{P}} (S_T(i, j) - S_S(i, j))^2, \quad (3.26)$$

where $D_T(i, j)$ and $D_S(i, j)$ represent the pairwise Euclidean distances between the teacher and student feature representations, respectively, while $S_T(i, j)$ and $S_S(i, j)$ denote the corresponding pairwise cosine similarities.

The framework also uses the Teacher’s output as a soft target alongside the actual ground truth [70]. The total loss function optimized during training is a weighted combination of these objectives:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{ground_truth}} + \lambda_{\text{response}}\mathcal{L}_{\text{response}} + \lambda_{\text{dist}}\mathcal{L}_{\text{dist}} + \lambda_{\text{cos}}\mathcal{L}_{\text{cos}} \quad (3.27)$$

To find optimal, hardware-constrained the Student model, a nested hyperparameter sweep is conducted during this distillation phase. The Student architecture maintains the same number of LSTM layers as the Teacher, while systematically sweeping the hidden layer capacities to maximize model fidelity within the MCU’s strict memory limits.

Concurrently, the sweep evaluates the optimal penalty weights for the loss function. Iterating through a primary coefficient ($\lambda_{\text{response}}$), the remaining weight ratio is deterministically split to establish the relational penalty parameters: the distance loss weight is defined as $\lambda_{\text{dist}} = (1.0 - \lambda_{\text{response}}) \times 0.6$, and the cosine loss weight is defined as $\lambda_{\text{cos}} = (1.0 - \lambda_{\text{response}}) \times 0.4$. Candidate configurations are evaluated using raw sampled time-series data and derived statistical features to capture specific voltage degradation profiles. This targeted sweep efficiently identifies the optimal combination of hidden size, λ_{dist} , and λ_{cos} that minimizes validation error without exceeding embedded hardware constraints [112].

Following distillation, the Student model is further compressed via structured pruning. This process targets the LSTM layer by evaluating the importance of each hidden unit based on the L_2 -norm of its corresponding weights in the subsequent linear layer connecting the hidden state to the output [63]:

$$I_k = \|\mathbf{W}_{\text{linear}}(:, k)\|_2, \quad (3.28)$$

where I_k is the importance score of the k -th hidden unit, and $\mathbf{W}_{\text{linear}}$ represents the weight matrix of the linear layer. A binary mask, m_k , is generated to identify which units to retain:

$$m_k = \begin{cases} 1, & \text{if } I_k \geq \tau, \\ 0, & \text{otherwise} \end{cases} \quad (3.29)$$

The threshold τ is dynamically determined by a target pruning ratio [61]. Letting K' represent the set of indices for surviving hidden units ($m_k = 1$), the model parameters are systematically reduced. The linear layer is compressed by selecting only columns associated with K' :

$$\mathbf{W}'_{\text{linear}} = \mathbf{W}_{\text{linear}}(:, K') \quad (3.30)$$

For each gate g in the LSTM layer, the weight matrices and bias vectors are strictly pruned to match the dimensions of the surviving units [4]. Specifically, the input-to-hidden weights are reduced to $(\mathbf{W}_{ih}^{(g)})' = \mathbf{W}_{ih}^{(g)}(K', :)$, and the hidden-to-hidden weights are compressed to $(\mathbf{W}_{hh}^{(g)})' = \mathbf{W}_{hh}^{(g)}(K', K')$. Furthermore, the corresponding bias vectors are pruned to match these dimensions, resulting in $(\mathbf{b}_{ih}^{(g)})' = \mathbf{b}_{ih}^{(g)}(K')$ and $(\mathbf{b}_{hh}^{(g)})' = \mathbf{b}_{hh}^{(g)}(K')$.

3.4.4 PINN-Knowledge distillation

As detailed in Section 2.2.6, the Physics-Informed Neural Network (PINN) models battery aging via a dual-network [113]. The primary high-capacity LSTM, $\mathcal{F}(\cdot)$, maps input features and time to the predicted State of Health (SoH), $\hat{u}$. Concurrently, a secondary network, $\mathcal{G}(\cdot)$, captures the physical dynamics of the SoH decay rate. The architecture is jointly optimized using a composite loss function comprising empirical data fidelity ($\mathcal{L}_{\text{data}}$), a monotonicity constraint for irreversible capacity fade ($\mathcal{L}_{\text{mono}}$), and a partial differential equation residual ($\mathcal{L}_{\text{PDE}}$) [53].

Once this dual-network PINN is fully trained offline, the secondary network ($\mathcal{G}(\cdot)$) is discarded. And the remaining primary network responsible for SOH prediction, physics-guided LSTM ($\mathcal{F}(\cdot)$) is considered as the Teacher model. However, even after removing the secondary network, this Teacher LSTM remains highly parameterized and is still far too computationally complex and memory-hungry for direct inference on a resource-constrained Microcontroller Unit (MCU).

To bridge this hardware gap without sacrificing physical consistency, PINN-Knowledge Distillation (PINN-KD) is integrated into the compression framework [114]. The objective of the KD phase is to perform a soft-penalty transfer of the complex Teacher’s physics-informed latent representations to a purely data-driven, heavily compressed LSTM Student model designed specifically for small device inference.

In this distillation process, the Student model implicitly inherits the critical battery physics and electrochemical knowledge during the PINN’s dual-network training phase. This acts as a final safeguard; it showcases that even under extreme sparsity and dynamic quantization, the student model remains stable to irregular sensor noise and is mathematically constrained from predicting impossible SoH trajectories during real world MCU operation [114][115].

To achieve this, the final distillation loss ensures the student model learns from both the correct data and the physics-based predictions of the PINN through specific loss functions:

Hard Loss: Hard labels represent the actual ground-truth SoH measured at each cycle. The hard loss calculates the Mean Squared Error (MSE) between the student model’s predictions and these labels. Its primary purpose is to ensure training stability and anchor the student with independent, data-driven predictive capabilities

[70]:

$$L_{\text{hard}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_{s,i} - y_i)^2 \quad (3.31)$$

where N represents the batch size, $\hat{y}_{s,i}$ denotes the SoH predicted by the student model at index i , and y_i is the corresponding hard label.

Soft Loss: Soft labels are the predictions generated by the highly complex, physics-guided PINN teacher, $\mathcal{F}(\cdot)$. The soft loss calculates the Mean Squared Error (MSE) between the outputs of the compressed student model and this teacher. Its primary purpose is to force the student to adopt the physics-informed reasoning and predictive patterns inherent to the teacher [70]:

$$L_{\text{soft}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_{s,i} - y_{\text{PINN},i})^2 \quad (3.32)$$

where $y_{\text{PINN},i}$ represents the SoH predicted by the PINN teacher model at index i .

Total Distillation Loss: The final loss function allows the lightweight student model to simultaneously learn from empirical ground truth and teacher predictions. While the hard loss provides direct guidance from real-world data, the soft loss transfers the teacher’s higher-level physical representations. Weighting these two components defines the total distillation loss:

$$L_{\text{total}} = \alpha L_{\text{soft}} + (1 - \alpha) L_{\text{hard}} \quad (3.33)$$

where $\alpha \in [0, 1]$ is a hyperparameter that controls the weighting between the soft loss and the hard loss.

Building upon the hardware-constrained sweep methodology established during the Relational KD phase, the PINN-KD process executes a dedicated search to optimize the soft-penalty transfer of physical boundaries from the PINN Teacher. While the Student architecture remains strictly constrained to the Teacher’s LSTM layer count, this phase systematically evaluates discrete hidden layer capacities alongside specific loss-balancing and regularization coefficients ($\alpha \in \{0.01, 0.05, 0.1\}$ and $(1 - \alpha) = \beta \in \{0.01, 0.05, 0.1\}$). By evaluating these candidate configurations against the previously defined time-series data and statistical features, the sweep efficiently identifies the optimal balance of physical consistency and predictive accuracy without exceeding the embedded hardware constraints.

3.4.5 Post-Training Quantization

As a complementary compression step, Static Post-Training Quantization (PTQ) is employed to convert the remaining high-precision floating-point weights and activations of all resulting models into low-precision integer values [10]. This drastically reduces the physical memory footprint and, depending on the model’s structure, accelerates inference speed on the MCU.

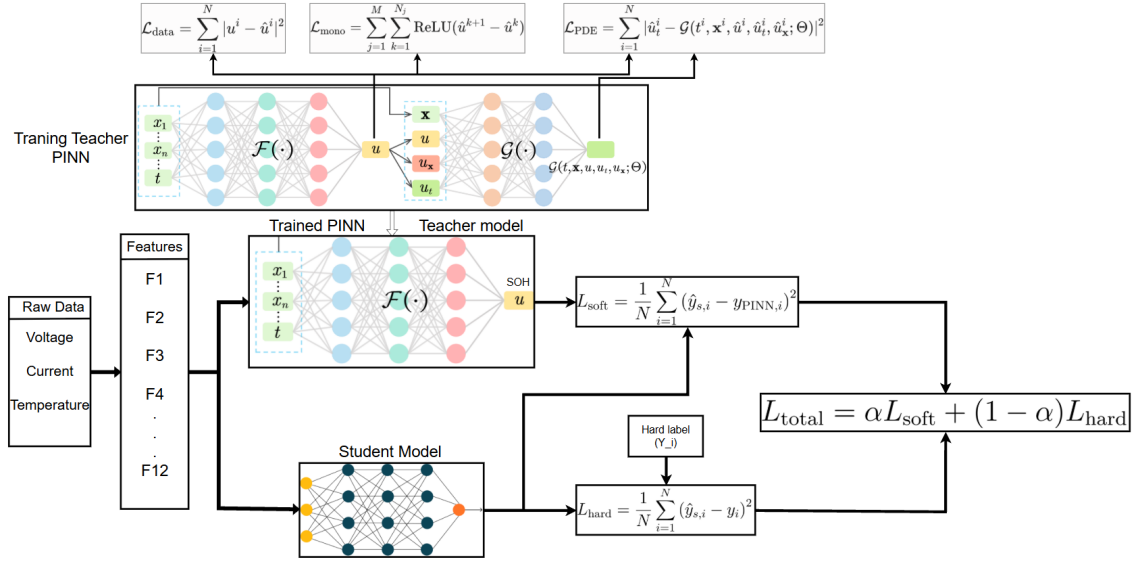


Figure 3.20: Illustration of relation PINN-KD [5]

For a given weight tensor W , a symmetric scaling factor s is computed based on the target bit-width b

$$s = \frac{\max(|W|)}{2^{b-1} - 1} \quad (3.34)$$

The weights are mapped to discrete integer values q during the quantization process[116]

$$q = \text{round}\left(\frac{W}{s}\right) \quad (3.35)$$

During real-time inference, the pre-calculated static scale factors are utilized, and an approximation of the original weights is recovered for output computation via dequantization ($\hat{W} = s \cdot q$). This the model achieves memory efficiency with degradation in State of Health estimation accuracy [10].

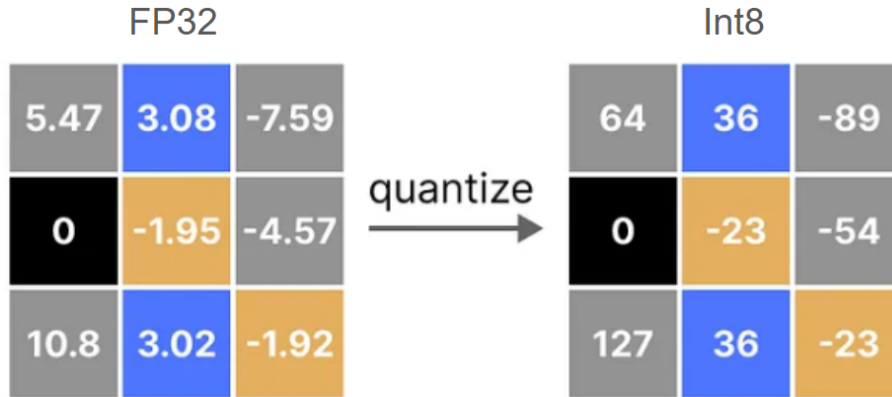


Figure 3.21: Illustration of Post quantization Training [10]

3.5 Resource Metrics Profiling

To systematically evaluate the compression framework on resource-constrained Microcontroller Units (MCUs), an analytical baseline of inference latency, memory footprint, and computational complexity must be established prior to applying compression [65]

When data transfer and computation execute concurrently on edge hardware, total inference latency is governed by the bottleneck between execution time and memory transfer time [105]:

$$\text{Latency} \approx \max(T_{\text{computation}}, T_{\text{memory}},) \quad (3.36)$$

Computation Time ($T_{\text{computation}}$):

$$T_{\text{computation}} \approx \frac{\text{Total Operations in Neural Network Model}}{\text{Processor Execution Capacity (OPS)}}, \quad (3.37)$$

Memory Transfer Time (T_{memory}): The time required to fetch the static network weights depends heavily on the model size and the processor’s memory bandwidth [117]:

$$T_{\text{memory}} \approx T_{\text{data movement of activations}} + T_{\text{data movement of weights}}, \quad (3.38)$$

Similarly, the time required to read input features and write output activation tensors to dynamic memory across all layers is defined as:

$$T_{\text{data movement of weights}} \approx \frac{\text{Neural Network Model Size (Bytes)}}{\text{Memory Bandwidth of Processor (Bytes/s)}}, \quad (3.39)$$

$$T_{\text{data movement of activations}} \approx \frac{\text{Input Activation Size} + \text{Output Activation Size}}{\text{Memory Bandwidth of Processor (Bytes/s)}}, \quad (3.40)$$

Architectural Parameter Formulation:

The total model size is directly proportional to the network’s topological parameters. For the time-series models utilized in this study, the parameter count is dominated by the Long Short-Term Memory (LSTM) layers and the fully connected linear projections. For an LSTM layer with an input feature dimension d_{in} and a hidden unit size h , the parameter count accounting for the four internal gates, recurrent weights, and biases is:

$$N_{\text{LSTM}} = 4 \times (d_{in}h + h^2 + 2h), \quad (3.41)$$

The parameter count for a subsequent dense linear layer mapping a hidden state to an output dimension d_{out} :

$$N_{\text{Linear}} = (h \times d_{out}) + d_{out}, \quad (3.42)$$

Memory Footprint (Flash):

Model size measures the required non-volatile storage (Flash memory) for storing all network weights and biases. Assuming uniform data precision across all layers, the total model size in bits is defined as the product of the total parameter count and the numerical precision of those:

$$\text{Model Size} = \# \text{Parameters} \times \text{Bit Width}, \quad (3.43)$$

For the uncompressed baseline, this bit width is typically 32 (FP32), whereas the quantized student model utilizes an 8-bit width (INT8). To evaluate compatibility with target edge hardware, this metric is converted into Bytes and Kilobytes (KB):

$$\text{Model Size (Bytes)} = \frac{\# \text{Parameters} \times \text{Bit Width}}{8} \quad (3.44)$$

$$\text{Model Size (KB)} = \frac{\# \text{Parameters} \times \text{Bit Width}}{8 \times 1024} \quad (3.45)$$

Dynamic Memory (Peak Activations): Maximum runtime SRAM required across layers during execution [65]:

$$\text{SRAM}_{\text{peak}} = \max_{l \in \text{Layers}} (\text{Size}(\text{Activation}_{\text{input}, l}) + \text{Size}(\text{Activation}_{\text{output}, l})) \quad (3.46)$$

Computational Workload and Operations: The core computational complexity of the neural network is quantified by the Multiply-Accumulate (MAC) metric, which forms the basis of the tensor algebra executed within the network’s layers (such as LSTM gates and dense projections). Because each MAC operation consists of two fundamental mathematical steps—one addition and one multiplication the absolute operation count (OPs or FLOPs) is generally double the total MAC count Multiply-Accumulate (MAC) Operation [118][105]:

$$a \leftarrow a + b \cdot c$$

Matrix Operations:

Matrix-Vector Multiplication (MV):

$$\text{MACs} = m \cdot n \quad (m \times n \text{ matrix by } n \times 1 \text{ vector})$$

General Matrix-Matrix Multiplication (GEMM):

$$\text{MACs} = m \cdot n \cdot k \quad (m \times k \text{ matrix by } k \times n \text{ matrix})$$

Operations Count:

$$\text{OPs} \approx 2 \times \text{MACs}$$

$$\text{FLOPs} \approx 2 \times \text{MACs} \quad (32\text{-Bit Baseline Floating-Point Operations})$$

Throughput and Roofline Constraints: To evaluate the actual hardware performance during inference, the total operational volume is measured as an execution rate over time (OPS or FLOPS). However, a processor’s theoretical peak execution speed is often bottlenecked by how quickly it can fetch weights and activations from memory. The Roofline Performance Model visualizes this bottleneck by calculating the arithmetic intensity (operations per byte of memory traffic) and bounding the attainable performance by either the processor’s peak compute limit or its memory bus bandwidth [119].

Throughput Rates:

$$\text{OPS} = \frac{\text{OPs}}{\text{Second}} \quad (\text{Quantized Integer Operations Per Second})$$

$$\text{FLOPS} = \frac{\text{FLOPs}}{\text{Second}} \quad (\text{Floating-Point Operations Per Second})$$

Arithmetic Intensity (I):

$$I = \frac{\text{Total Operations (OPs)}}{\text{Total Memory Traffic (Bytes)}}$$

Roofline Attainable Performance (P):

$$P = \min(P_{\text{peak}}, I \times \text{Bandwidth}_{\text{peak}})$$

where P_{peak} represents the peak processor compute capability (OPS) and $\text{Bandwidth}_{\text{peak}}$ represents the maximum memory bus throughput (Bytes/s) [119].

Power and Energy Consumption:

For embedded systems, the energy efficiency of the neural network inference is a critical metric. The total energy consumed per inference (E) is determined by the active power drawn by the microcontroller (P_{active}) and the total inference latency (T_{latency}):

$$E = P_{\text{active}} \times T_{\text{latency}} \quad (3.47)$$

The baseline active power is a hardware-specific constant dependent on the core clock frequency (f_{core}), the active current density of the processor (I_{density}), and the supply voltage (V_{supply}) [120]:

$$P_{\text{active}} = f_{\text{core}} \times I_{\text{density}} \times V_{\text{supply}} \quad (3.48)$$

3.6 Model Compilation and Runtime Engine Selection

To bridge the gap between high-level deep learning frameworks and resource constrained embedded environments, Apache TVM is employed as the core compilation

toolchain. The deployment pipeline translates the trained PyTorch models (both full-precision and quantized variants) into statically allocated C/C++ code, bypassing the need for a traditional operating system or dynamic memory allocation.

3.6.1 Intermediate Representation and Compilation Backend

The conversion process begins by tracing the PyTorch model into TVM’s Intermediate Representation (IR). To accommodate the evolution of the model architecture, the compilation backend leverages the modern TVM Relax framework. The PyTorch models are converted to generate the Relax IR, which natively supports advanced dynamic shape handling and custom quantized operators.

For the deployment of the models, a bare-metal Ahead-of-Time (AOT) compilation strategy is utilized alongside the Relax Virtual Machine (VM) engine. Because this study targets two fundamentally different microcontroller architectures, the compilation pipeline is distinctly bifurcated:

For the NXP MCXN947, which is based on the ARM Cortex-M33 architecture, the TVM compiler leverages the LLVM backend. The target string is explicitly configured to enable hardware-specific code generation and vectorization optimizations natively supported by the ARM instruction set.

Conversely, the Infineon TC4D7 utilizes the proprietary TriCore architecture. Since TriCore is not natively supported by the standard LLVM deployment pipelines in TVM, the model is compiled using the generic C backend. This approach generates platform-agnostic, pure ANSI C source code representing the network graph and operators. The generated C code is subsequently compiled and linked into a bare-metal binary using a dedicated TriCore cross-compilation toolchain.

During both compilation phases, patches are applied to the TVM C Run-Time (CRT) templates to prevent structural signature conflicts, ensuring a seamless integration into the respective standalone bare-metal applications [121].

3.6.2 Memory Management and Weight Handling

To bridge the gap between high-level deep learning frameworks and resource constrained embedded environments, an automated compilation pipeline is employed. Figure 3.22 illustrates this end-to-end transformation process, translating the trained PyTorch models into statically allocated bare-metal C code while decoupling the network’s mathematical operators from its static parameters [122].

A critical challenge in edge AI deployment is accommodating the strict volatile memory constraints of the MCU. The compilation pipeline addresses this bottleneck by explicitly decoupling the model’s static weights from its executable logic. Rather than initializing massive parameter arrays within the limited SRAM, the weights are exported as independent data structures. Depending on the precision stage, these weights are either exported as external binary structures or encoded as constant arrays within an auto-generated C header file. By applying specific hardware memory

alignment attributes, the computational engine maps its input pointers directly to these parameters stored in the Read-Only Flash memory. This architectural design enables highly efficient bus access and drastically minimizes the runtime memory footprint without consuming volatile SRAM [123].

Furthermore, dynamic memory allocation operations are strictly prohibited during bare-metal execution to prevent unpredictable latency and heap fragmentation. To manage the temporary memory required for intermediate activation tensors, the pipeline implements a custom static bump allocator via the TVM backend API (`TVMBackendAllocWorkspace`). A structurally aligned, contiguous block of SRAM is reserved at compile time, and the allocator safely cycles through this bounded pool by resetting its offset pointer at the beginning of each inference step. This deterministic memory architecture guarantees mathematical safety and completely eliminates the overhead associated with dynamic memory management under continuous operation [123][124].

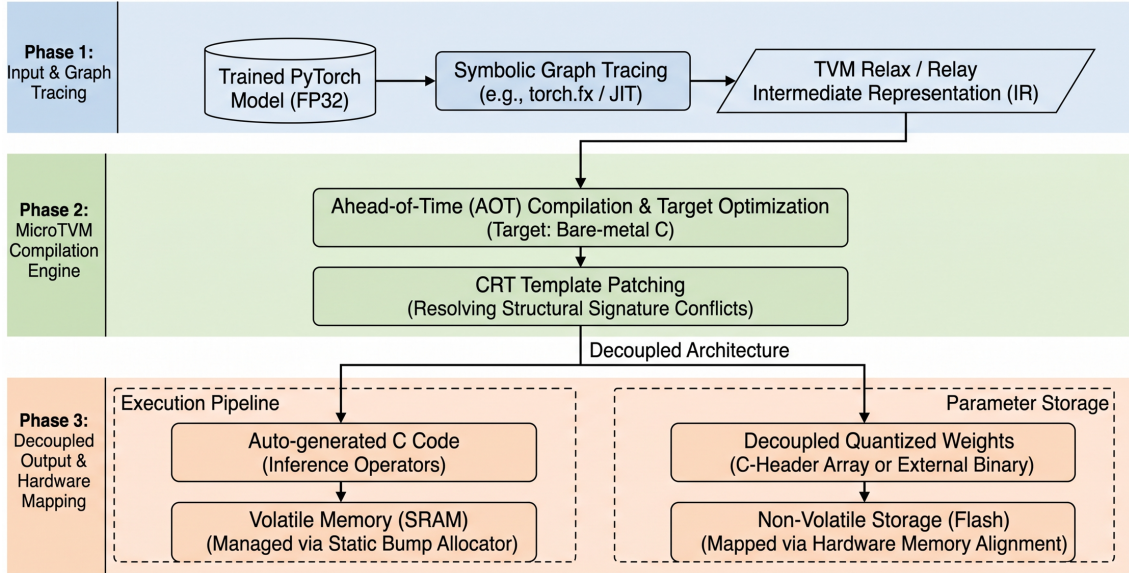


Figure 3.22: Apache TVM compilation pipeline translating PyTorch models into bare-metal C code and aligned weight structures.

3.6.3 Custom W8A32 Dynamic Quantization Pipeline

Standard full-integer (INT8) quantization frameworks often struggle to deploy recurrent architectures like LSTMs onto bare-metal microcontrollers. The recursive nature of LSTM cell states means that minor quantization errors in intermediate activations accumulate rapidly over time, frequently leading to vanishing representations or catastrophic accuracy degradation. Furthermore, automated mixed-precision compilation passes can encounter operator support failures on highly constrained proprietary hardware targets.

To overcome these deployment bottlenecks, a custom W8A32 (Weight-8 bit, Activation 32 bit) dynamic quantization strategy was engineered. The W8A32 paradigm leverages the statistical robustness of neural network weights by compressing the

static parameters, while preserving the dynamic activations and temporal accumulations at full precision [125].

Explicit Graph Tracing and Quantization Mechanics

Rather than relying on automated post-training quantization compilers, the W8A32 semantics were explicitly integrated into the model’s computational graph prior to compilation.

First, symmetric quantization is applied to the static weights of the fully connected layers and LSTM gates offline. The 32-bit floating-point weights are scaled and clamped into 8-bit integers. To completely bypass limitations in the compiler’s automated parameter handling, these compressed weights are decoupled from standard neural network parameters and registered as static constant buffers.

Crucially, dynamic dequantization logic is explicitly embedded directly into the forward pass of the model. The network is subsequently traced into the TVM Relax IR using symbolic graph tracing (`torch.fx`). By capturing the explicit dequantization nodes within the IR, the pipeline intentionally bypasses default mixed-precision translation passes that might fail during C code generation[126].

Runtime Execution and Hardware Benefits

During real-time inference on the edge device, the execution follows a strict W8A32 flow:

1. **Memory-Efficient Fetching:** The 8-bit weights are fetched directly from the read-only Flash memory, achieving a theoretical 75% reduction in non-volatile storage requirements compared to a standard FP32 model.
2. **Dynamic Dequantization:** Immediately before the core matrix multiplication operations, the fetched 8-bit weights are dynamically cast back to 32-bit floating-point values using their respective scaling factors.
3. **High-Fidelity Accumulation:** All subsequent operations—including input matrix multiplications, gating activations, and long-term cell state updates—are executed at full FP32 precision utilizing the microcontroller’s hardware Floating-Point Unit (FPU).

This custom compilation pipeline successfully bridges the gap between deep compression and mathematical stability. It guarantees that the generated bare-metal C application benefits from the drastic storage reductions of INT8 quantization, while entirely averting the temporal precision loss typically associated with integer-only recurrent [124].

3.7 Hardware Deployment and Validation on NXP MCXN947

Following the successful TVM compilation and static resource profiling, the deployment transitioned to a physical execution architecture. While the Software-in-the-

Loop (SIL) architecture was qualitatively validated on the Infineon TC4D7, the comprehensive real-time benchmarking—including both deterministic latency profiling and asynchronous Processor-in-the-Loop (PIL) stability testing—was consolidated on the NXP MCXN947 platform.

3.7.1 Isolated Inference Benchmarking via Feature Injection

To rigorously benchmark the pure computational performance of the AI models on the MCU without compounding errors from dynamic sensor noise or real-time feature scaling limits, an isolated testing protocol was established. For performance profiling, pre-computed and normalized features from independent test datasets were directly injected into the static SRAM allocated for the TVM engine.

The MCXN947's Cortex-M33 Data Watchpoint and Trace (DWT) cycle counter was utilized to profile the exact clock cycles and millisecond latency consumed by the inference functions. This decoupled approach ensures an objective measurement of the algorithmic efficiency and absolute execution latency for both the Expert and Distilled [127].

3.7.2 FlexCAN Double-Buffering and Concurrency Control

To handle high-frequency sensor data without halting the AI inference engine in a real-world scenario, a non-blocking double-buffering (ping-pong) architecture was implemented over the FlexCAN interface. The system captures raw voltage, current, and temperature packets inside a high-priority Interrupt Service Routine (ISR) and pushes them into an active sub-window array.

To prevent data corruption and race conditions without introducing blocking RTOS primitives, a lock-free buffer swapping mechanism is enforced. Buffer pointers are exchanged atomically within a minimal critical section using global interrupt masking, ensuring that the ISR and the background TVM inference loop never access the same memory buffer simultaneously. Under standard operation, this guarantees reliable data ingestion. However, if the AI engine is still processing a window when the next window fills up during extreme bus congestion, an `overflow_counter` logs the dropped frame. This graceful degradation allows the system to isolate transient timing overruns without triggering a hard [128].

3.7.3 On-Device Feature Engineering and System Safety

To enable a real-time SOH estimator, the MCU continuously captures raw voltage, current, and temperature streams from the CAN bus into active sub-window buffers. Rather than relying on external preprocessing, a dedicated on-device routine dynamically extracts degradation-relevant features directly from these raw buffers.

Because edge AI models are highly sensitive to corrupted sensor data or transient bus errors, multiple architectural safety barriers were established:

- **Mathematical Fault Isolation:** The system implements real-time validation checks on all normalized input features, intermediate recurrent states, and final

predictions. If non-finite numerical values are detected, the system intercepts the error and resets the LSTM hidden and cell states. This mechanism prevents mathematical corruption from propagating across subsequent sequence steps.

- **Workspace Overflow Protection:** To avoid heap fragmentation and unpredictable latency, dynamic memory allocation is strictly prohibited. Intermediate tensor computations are managed by a static bump allocator. This allocator enforces rigorous boundary checks, ensuring that any computation exceeding the statically reserved SRAM pool is safely trapped before causing adjacent memory region overwrites.
- **Direct Flash Weight Mapping and Hardware Alignment:** To strictly conserve volatile SRAM, the network’s quantized parameters are not loaded into RAM. Instead, the computational engine maps its pointers directly to the read-only Flash memory. Furthermore, to support the MCU’s FPU safely, all memory buffers and state tensors are aligned to meet specific hardware requirements, enabling safe vectorized execution [129].

4

Results and Discussion

4.1 Features selection

4.1.1 List of Features Extracted

Applying the theory and equations detailed in Section 3.2.1, a total of 28 statistical and physics-based features are derived from the raw time-series sampled data. The complete list of extracted features is summarized in the Table 4.1 below:

Table 4.1: Extracted Feature Set

No.	Feature Name	No.	Feature Name
1	global_CC_Q	15	current_std
2	global_CV_Q	16	current_min
3	global_CC_Time	17	current_max
4	global_CV_Time	18	current_last
5	voltage_mean	19	current_skew
6	voltage_std	20	current_kurt
7	voltage_min	21	temperature_mean
8	voltage_max	22	temperature_std
9	voltage_last	23	temperature_min
10	voltage_slope	24	temperature_max
11	voltage_skew	25	temperature_last
12	voltage_kurt	26	capacity_window
13	voltage_entropy	27	energy_window
14	current_mean	28	power_var

4.1.2 Selected Features

Following the methodology detailed in Section 3.2.2, a multi-evaluation strategy is employed to identify the most useful variables from the total extracted features.

The bar plot in Figure 4.1 presents the results of the random forest feature importance analysis, showcasing the relative contribution of each variable in predicting the State of Health (SoH). From this evaluation, the features are ranked by their predictive strength, allowing for the direct identification of the most influential variables that dictate capacity fade. Furthermore, the circular bar plot in Figure 4.3 details the individual predictive strength of each feature against the target SoH, capturing

complex, non-linear relationships via Maximal Information Coefficient (MIC) scores. Because battery degradation dynamics are highly non-linear, relying solely on linear metrics is insufficient; the MIC scores ensure that features with intricate but highly predictive patterns are properly recognized.

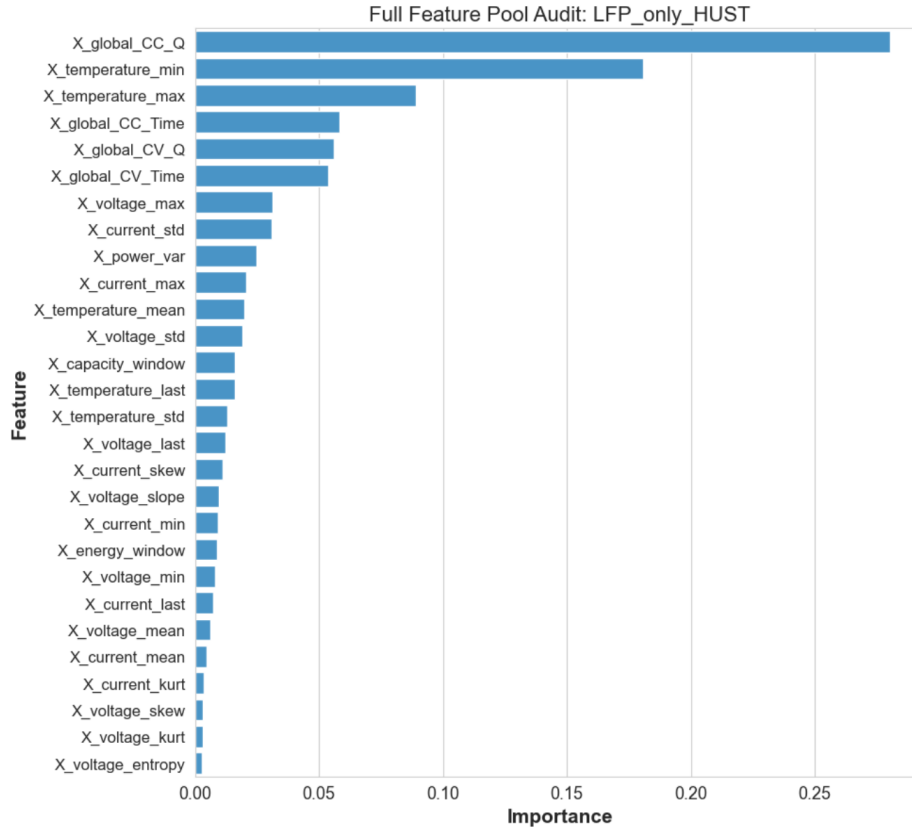


Figure 4.1: Illustration of Feature importance using random forest

To further support this selection and optimize the computational efficiency of the model, a Pearson correlation heat map (Figure 4.2) is utilized to illustrate internal collinearity and eliminate redundant data. By analyzing the Pearson collinearity results, overlapping features carrying the same statistical information are systematically filtered out to prevent overfitting and reduce noise. Specifically, when the collinearity value between two or more features exceeds a threshold of 0.85, their predictive rankings from the feature importance and MIC evaluations are cross-referenced. The higher-ranked feature is retained as the primary information carrier, while the less influential, redundant features are removed from the final dataset.

Through this rigorous elimination process, the initial feature pool was successfully reduced to a final optimized subset of 12 highly predictive features. These variables, which serve as the final inputs for the model, are summarized in Table 4.2.

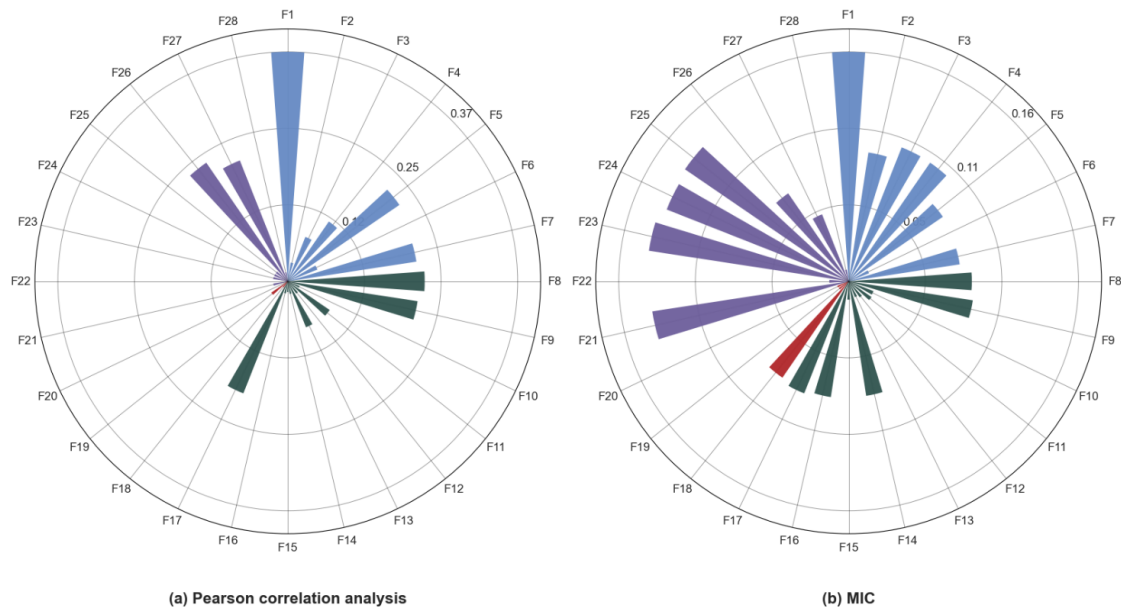


Figure 4.2: Illustration of Maximal Information Coefficient

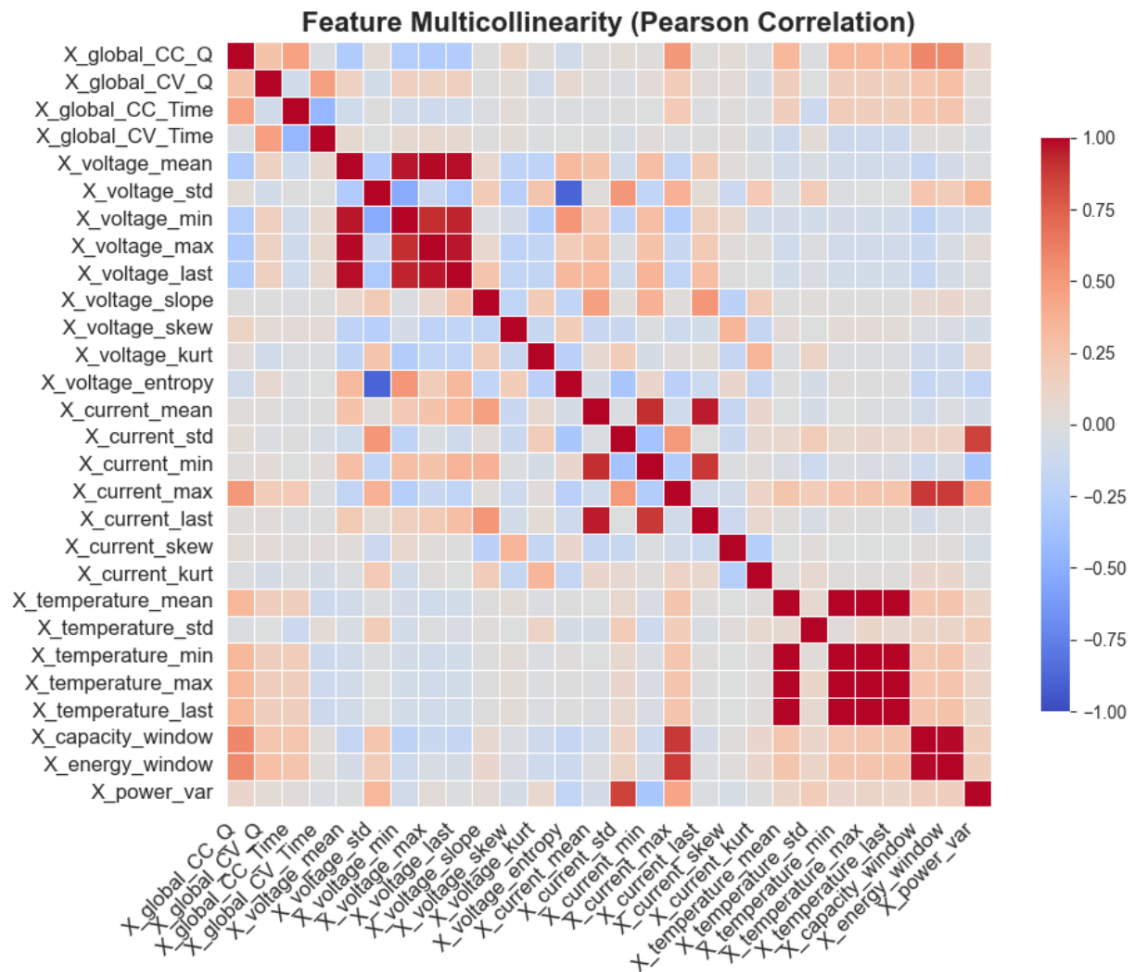


Figure 4.3: Illustration of Feature multicollinearity check

Table 4.2: Selected Features

No.	Feature Name	No.	Feature Name
1	global_CC_Q	7	voltage_slope
2	global_CV_Q	8	current_max
3	global_CC_Time	9	temperature_std
4	global_CV_Time	10	temperature_max
5	voltage_std	11	capacity_window
6	voltage_max	12	voltage_kurt

4.2 Baseline model evaluation

The default baseline model is the Long Short-Term Memory (LSTM) network introduced above in Section 2.2.3, in line with the key research questions that motivate this study. The baseline assessment is important as it provides a common performance reference point for the rest of the study.

4.2.1 Architecture and Hyperparameter Optimization

The macro-architecture of the model to two sequential LSTM layers followed by a single fully connected (dense) output layer is fixed, to ensure a strict and controlled evaluation. A thorough hyperparameter sweep with a fixed structural depth was performed to improve the predictive capability of the model. Specifically, the search considered batch sizes of 16, 32 and 64 and four different capacity settings for the dual LSTM layers: [32, 32], [64, 32], [64, 64] and [128, 64] hidden units. The systematic tuning process was used to find the best trade-off between model complexity and generalization in describing the degradation dynamics of the battery.

Throughout the evaluation, key training hyperparameters were held constant to maintain a consistent baseline. The maximum number of training epochs was set to 2500, supplemented by an early stopping mechanism with a patience of 150 epochs to mitigate overfitting. Additionally, the initial learning rate was fixed at 1×10^{-3} , utilizing a scheduled learning rate adaptation strategy. This allowed the model to dynamically adjust the step size as training progressed, ensuring stable and efficient convergence.

From the results of the hyperparameter sweep, the most robust performance was obtained from the configuration with a batch size of 32 and a layer capacity of [64, 32] hidden units. Hence, this architecture is chosen as the final baseline model for all the subsequent evaluations.

The first evaluation is performed on the Lithium Iron Phosphate (LFP) dataset after fixing the architecture. The model was trained and tested only on LFP data to provide a clear baseline. In this step we show the LSTM’s ability to capture the specific aging patterns of LFP cells before introducing other battery chemistries.

The following figures visualize this baseline performance, comparing the model’s predicted State of Health (SoH) against the actual experimental degradation data.

Next, we evaluated the baseline model on the NMC dataset, training and testing it

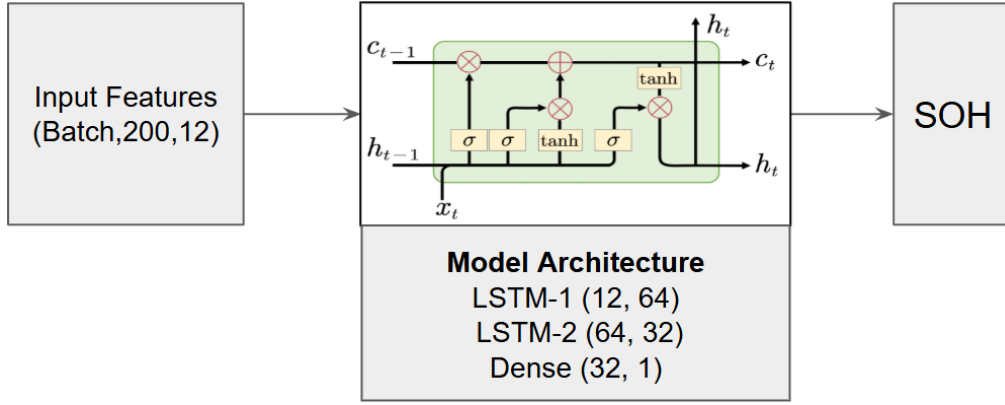


Figure 4.4: Baseline Model architecture [3]

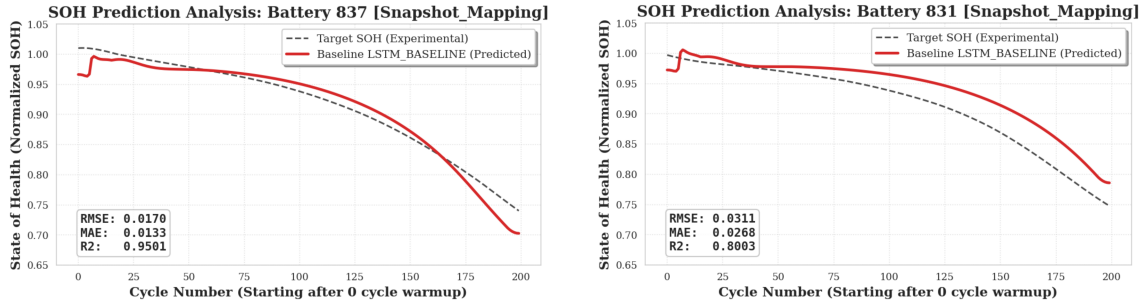


Figure 4.5: HUST dataset results

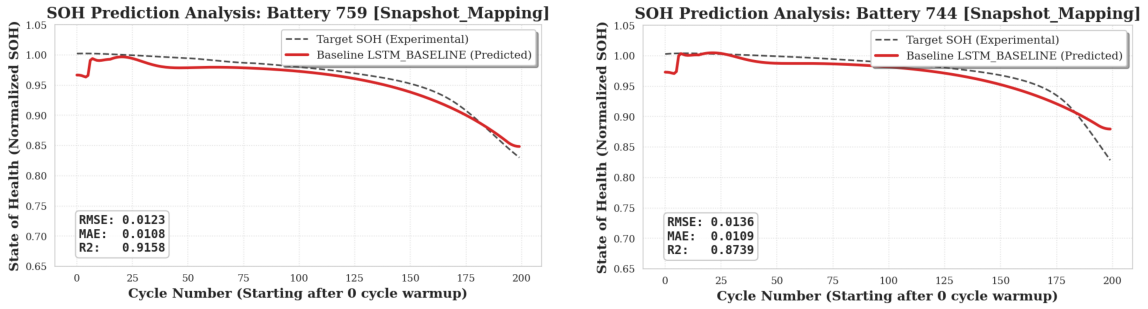


Figure 4.6: MIT dataset results

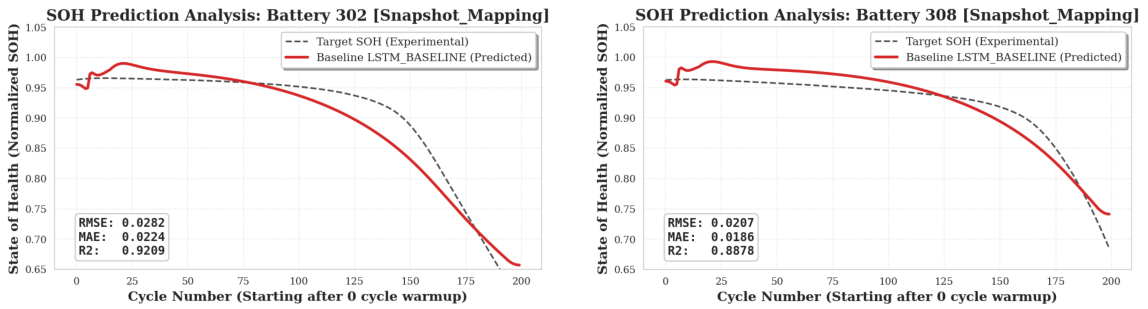


Figure 4.7: Attia dataset results.

exclusively on NMC data. This provides a direct baseline for the model's ability to learn NMC-specific aging dynamics.

The following figures The Figures 4.8, 4.9, 4.10 and 4.14 display the predicted SoH trajectories compared against the actual experimental NMC degradation data.

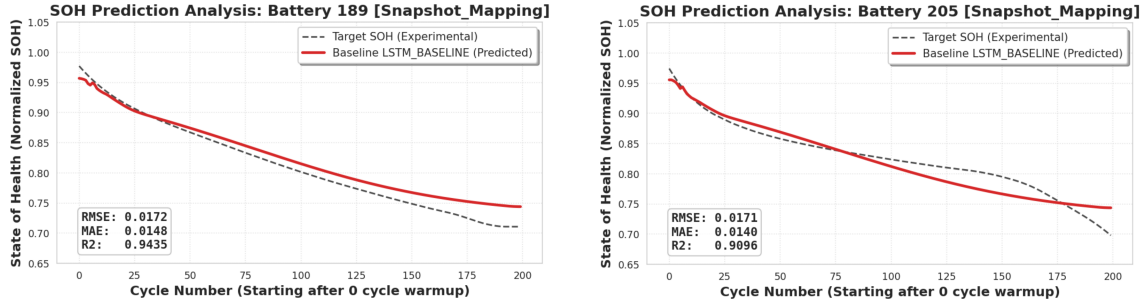


Figure 4.8: luhblank2024 Cyclic dataset results.

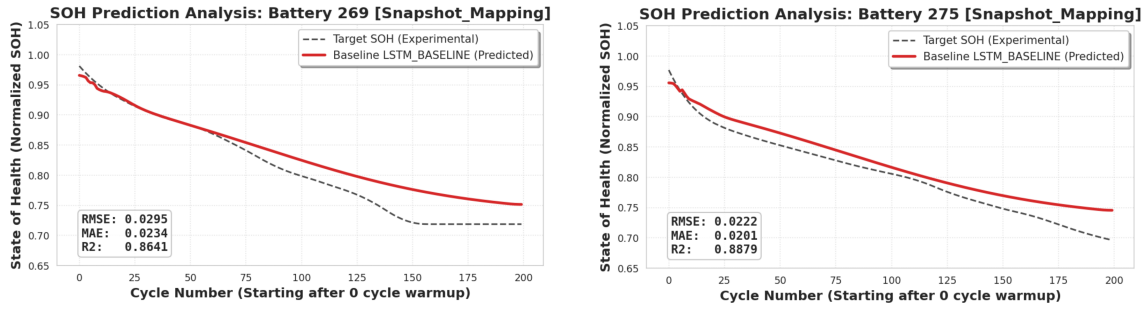


Figure 4.9: luhblank2024 profile dataset results.

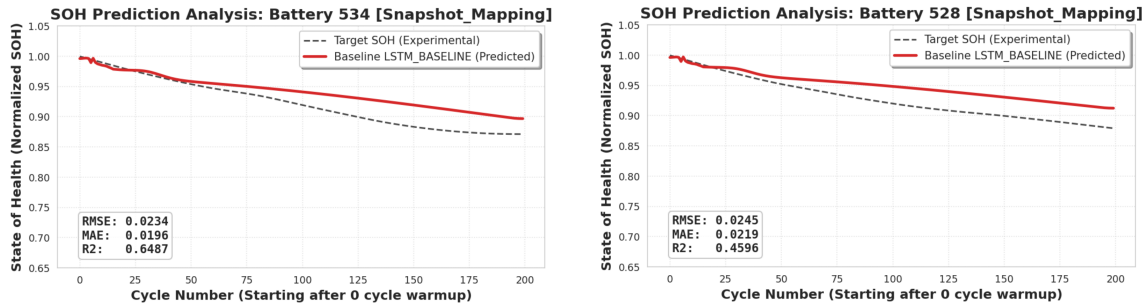


Figure 4.10: RWTH dataset results.

Finally, we evaluated the baseline model on the NCA dataset. By training and testing exclusively on NCA data.

The Figures 4.11, 4.12, 4.13 and 4.14 below compare the predicted SoH with the actual experimental data for the NCA cells.

4. Results and Discussion

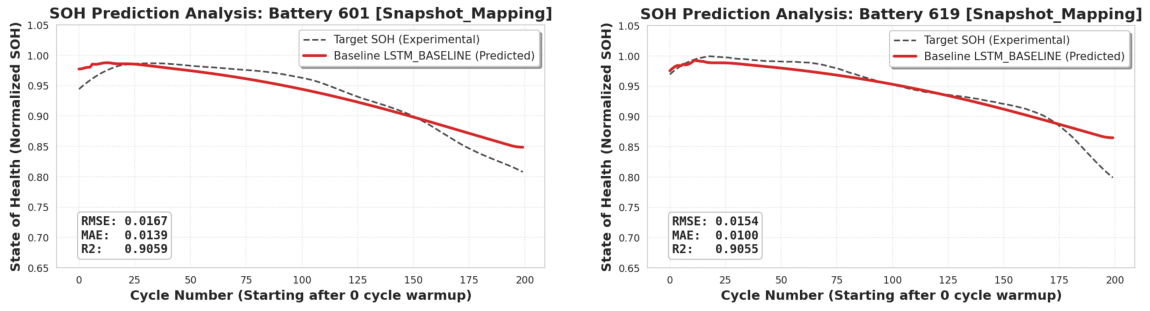


Figure 4.11: XJTU dataset results.

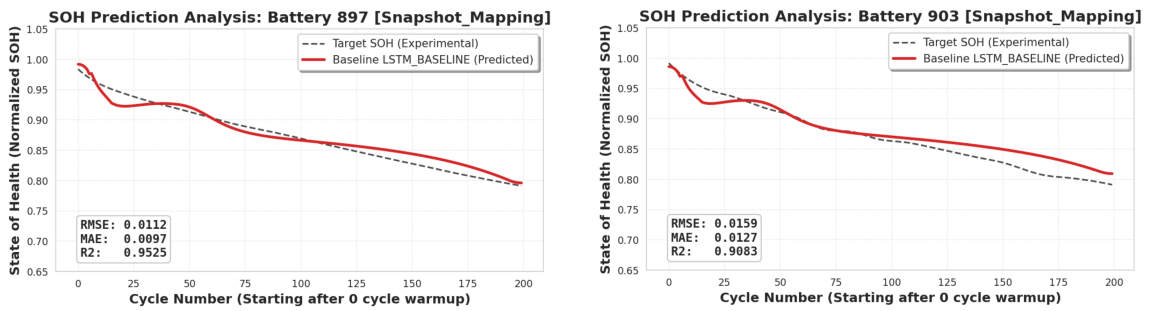


Figure 4.12: GESLIN2025 dataset results.

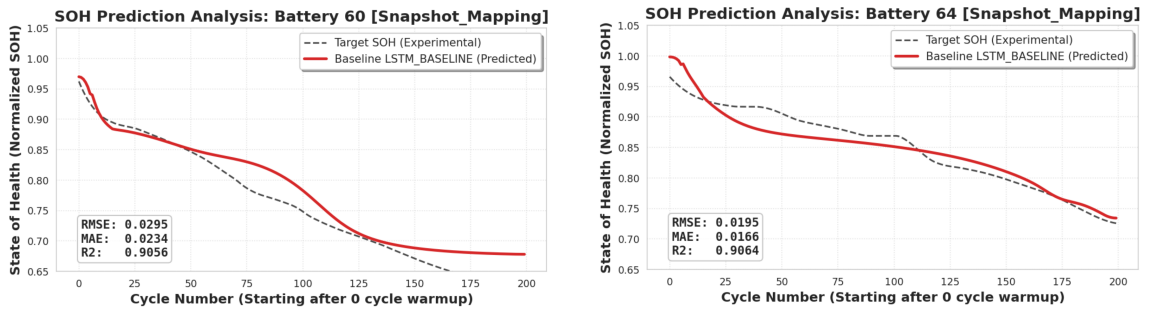


Figure 4.13: GRAZ2023 dataset results.

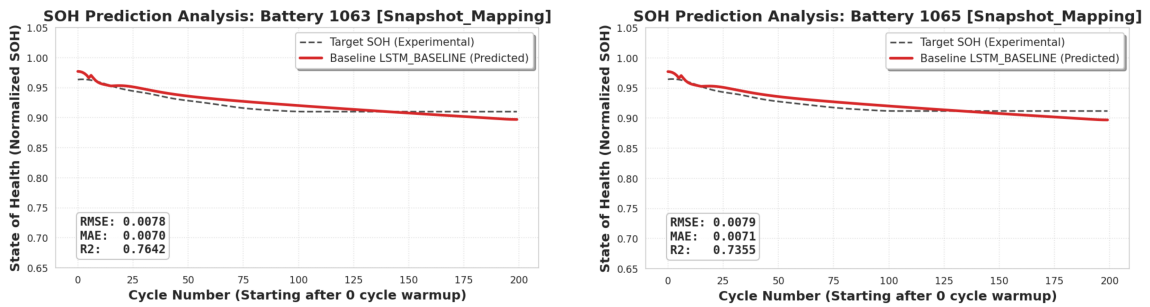


Figure 4.14: STROEBL dataset results.

4.2.2 Experimental Design for Condition Agnostic Feasibility

Following the initial experimentation, specifically the training and testing on individual battery chemistries and evaluating their isolated performance. The next objective of core research question: Is it feasible to develop a condition-agnostic, generalized LSTM Architecture for SoH estimation?

To test this, we designed a carefully controlled experiment. The main approach was to keep the test data completely separate while gradually adding more variety to the training data. For testing, we only used a specific group of LFP batteries that the model had never seen, having set them aside during the initial random data split.

As the training data expanded from a specialized LFP domain to a universal, condition-agnostic domain, a clear trend emerged in the model's predictive accuracy on the isolated LFP test set.

Training and Testing only on LFP data:

The baseline model, which was trained and evaluated solely on LFP datasets, is described in this section. This domain-specific model accurately tracked the real SoH trajectories, as shown in Figures 4.15 and 4.16. The forecasts relatively close match to the ground-truth data.

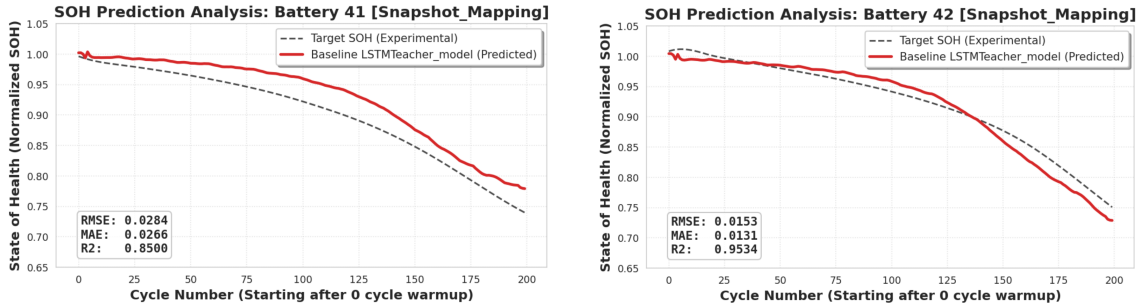


Figure 4.15: LFP dataset results specialized LFP domain

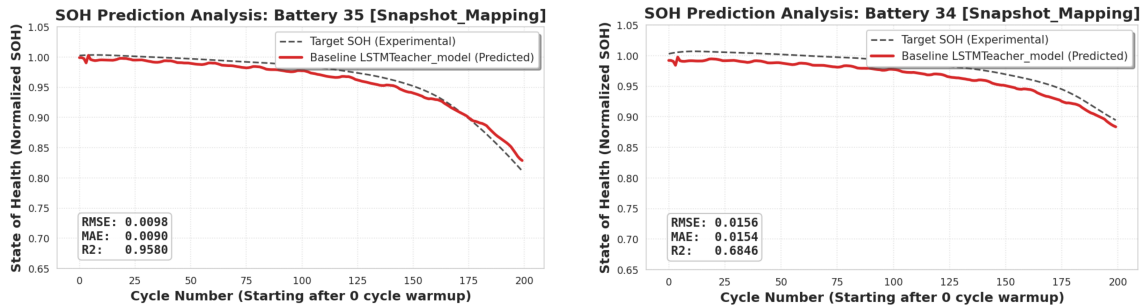


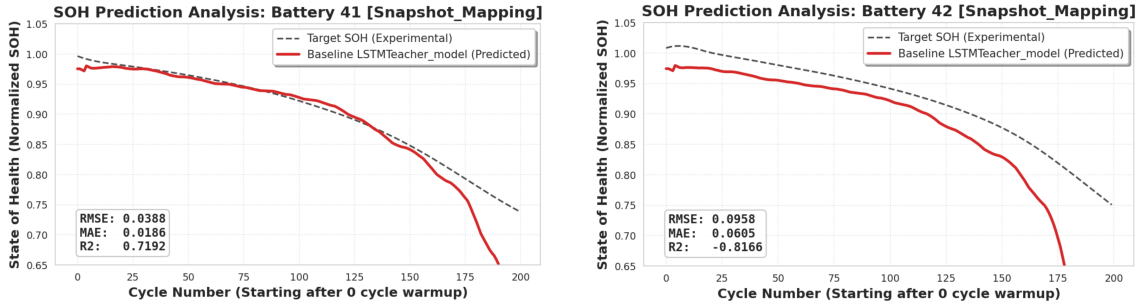
Figure 4.16: LFP dataset results specialized LFP domain

Table 4.3: Performance of the specialized LFP domain LSTM model on Test battery datasets.

Chemistry	Battery ID	RMSE	MAE	R ²
LFP	HUST-41	0.0284	0.0266	0.8500
LFP	HUST-42	0.0153	0.0131	0.9534
LFP	HUST-43	0.0205	0.0158	0.9103
LFP	HUST-44	0.0192	0.0169	0.9318
LFP	MIT-34	0.0156	0.0154	0.6846
LFP	MIT-35	0.0098	0.0090	0.9580
LFP	MIT-38	0.0232	0.0219	0.8514
LFP	MIT-39	0.0318	0.0305	0.7132

Training and Testing on mix of LFP and NMC data:

In the second experimental phase, NMC datasets were introduced into the training pool while the test set remained strictly isolated to the same LFP batteries. Forcing the network to simultaneously learn the degradation mechanics of both LFP and NMC chemistries resulted in a visible drop in local accuracy. The predicted SoH curves began to deviate from the ground truth, reflecting a decrease in precision compared to the specialized baseline results shown in Figures 4.17 and 4.18.

**Figure 4.17:** LFP dataset results specialized LFP and NMC domain

Training and Testing on mix of LFP, NMC and NCA data:

In the final phase of the experiment, the training pool was expanded to its maximum diversity by integrating NCA datasets alongside LFP and NMC. The model was then evaluated on the exact same isolated LFP test batteries. Here forcing the network to simultaneously learn the degradation mechanics of both LFP, NMC and NCA chemistries resulted in a considerable drop shown in Figures 4.19 and 4.20 .

The results show a major limitation of standard LSTMs: they struggle to generalize across different battery types. While an LSTM can spot general capacity-fade trends, its architecture cannot perfectly handle the conflicting voltage profiles of LFP, NMC, and NCA batteries all at once. To understand exactly why building a

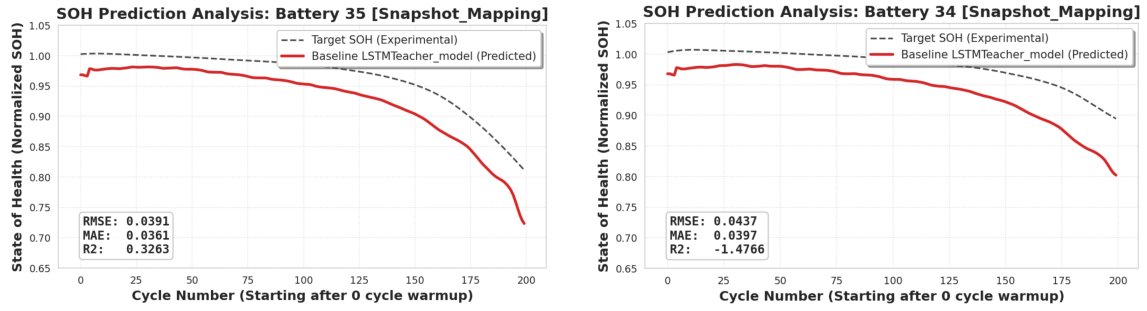


Figure 4.18: LFP dataset results of specialized LFP and NMC domain

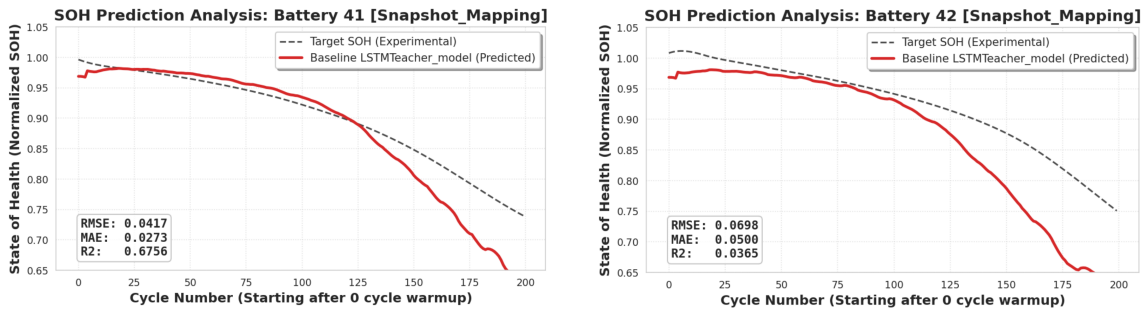


Figure 4.19: LFP dataset results of specialized LFP, NMC and NCA domain

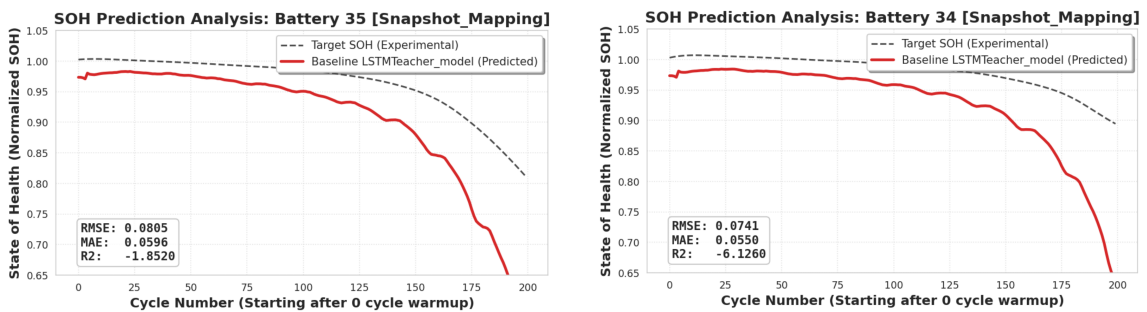


Figure 4.20: LFP dataset results of specialized LFP, NMC and NCA domain

Table 4.4: Performance of the specialized LFP and NMC domain LSTM model on Test battery datasets.

Chemistry	Battery ID	RMSE	MAE	R ²
LFP	HUST-41	0.0388	0.0186	0.7192
LFP	HUST-42	0.0958	0.0605	-0.8166
LFP	HUST-43	0.0522	0.0438	0.4169
LFP	HUST-44	0.1147	0.0657	-1.4355
LFP	MIT-34	0.0437	0.0397	-1.4766
LFP	MIT-35	0.0391	0.0361	0.3263
LFP	MIT-38	0.0578	0.0505	0.0762
LFP	MIT-39	0.0998	0.0821	-1.8330

Table 4.5: Performance of the specialized LFP,NMC and NCA domain LSTM model on Test battery datasets

Chemistry	Battery ID	RMSE	MAE	R ²
LFP	HUST-41	0.0417	0.0273	0.6756
LFP	HUST-42	0.0698	0.0500	0.0365
LFP	HUST-43	0.0644	0.0501	0.1100
LFP	HUST-44	0.0507	0.0337	0.5237
LFP	MIT-34	0.0741	0.0550	-6.1260
LFP	MIT-35	0.0805	0.0596	-1.8520
LFP	MIT-38	0.0730	0.0590	-2.9433
LFP	MIT-39	0.1219	0.0960	-3.2245

chemistry-agnostic model is so difficult for chosen Network, we analyzed the internal weights of the LSTM's.

Gradient dominance

A diagnostic analysis is conducted on the model weights. When an LSTM model attempts to learn multiple distinct behaviors simultaneously, a specific battery chemistry that dominates the training process will induce gradients with relatively large magnitudes. This phenomenon is gradient dominance[130].

As illustrated in Figure 4.21, an analysis of the backpropagated gradient magnitudes (measured via the L2 Norm) reveals a severe imbalance between the three chemistries:

NCA acts as the dominant domain, producing the largest gradient magnitude (0.0790). LFP follows with a heavily influential gradient magnitude (0.0518). NMC produces a nearly negligible gradient magnitude (0.0074).

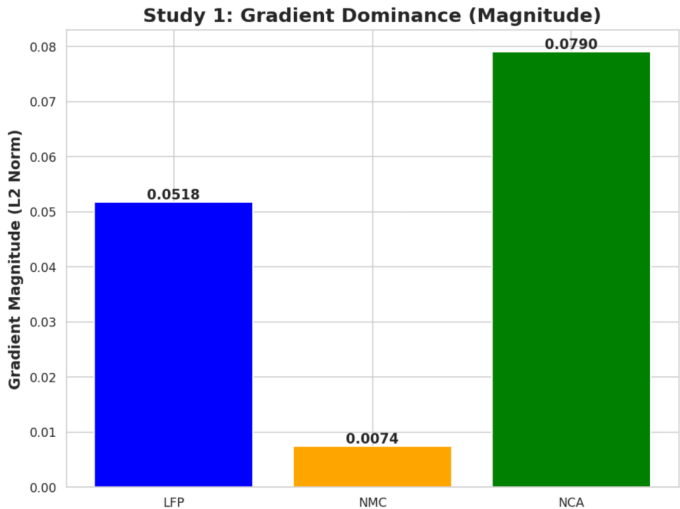


Figure 4.21: Illustration of Gradient Dominance in the gradient updates

Gradient cosine

The second analysis of the gradients focuses on the direction of the gradient updates. In multi-domain learning, as demonstrated by Yu et al [131], it is critical that gradients from different datasets align to learn a shared, universal representation. When gradients fail to align, the model struggles to find common latent features which leads to poor generalization.

As shown in Figure 4.22, the absolute cosine similarities between the backpropagated gradients are severely low across all chemistry pairings. Because these values are close to zero, it indicates that the gradient updates are nearly orthogonal.

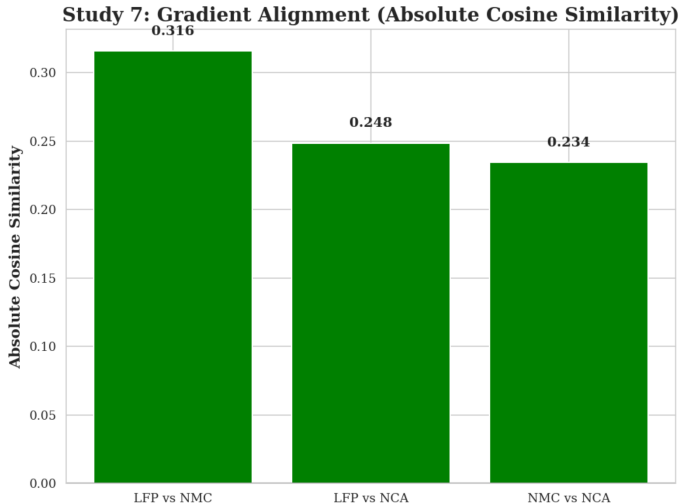


Figure 4.22: Illustration of Gradient cosine similarity

Specifically, the alignment between LFP and NMC gradients peaks at just 0.316, while the similarities between LFP vs. NCA (0.248) and NMC vs. NCA (0.234) are even lower.

Consequently, the network’s updates do not align. Instead of working together to learn a common aging pattern, the different battery chemistries pull the model’s weights in other directions. When this misalignment is combined with the magnitude imbalances observed earlier, it actively disrupts the network’s ability to converge on a unified, chemistry-agnostic model.

Representational dissimilarity

To understand how the network internally perceives the different battery chemistries, we analyzed its hidden states (latent features) using Representational Similarity Analysis (RSA). Dissimilarity is measured as $1 - r$ (where r is the Pearson correlation); higher values indicate completely distinct internal representations[132].

As shown in Figure 4.23, a clear representational divide emerges within the network’s hidden layers:

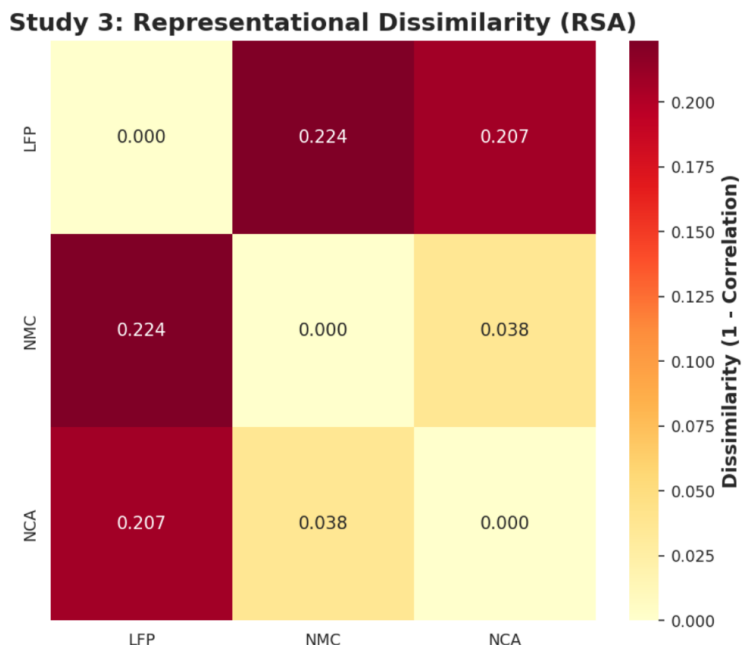


Figure 4.23: Illustration of Representational Similarity Analysis (RSA)

The model processes NMC and NCA almost identically (scoring 0.038), reflecting their similar, sloping voltage profiles. In contrast, LFP is processed entirely differently, showing a massive representational gap when compared to NMC (0.224) and NCA (0.207).

To visually confirm this, we applied t-Distributed Stochastic Neighbor Embedding (t-SNE) to the LSTM’s hidden states. As illustrated in Figure 4.24, basically presenting the high-dimensional feature space into two dimensions reveals a clear clustering of latent features. The LFP chemistry forms a distinctly isolated cluster, whereas the NMC and NCA chemistries exhibit partial overlap. The presentation of these features highlights the difficulty of building a single, unified model across these domains. The distinct voltage profiles of the different chemistries force the network

into conflicting internal representations, preventing the model from converging on a truly chemistry-agnostic feature space[133].

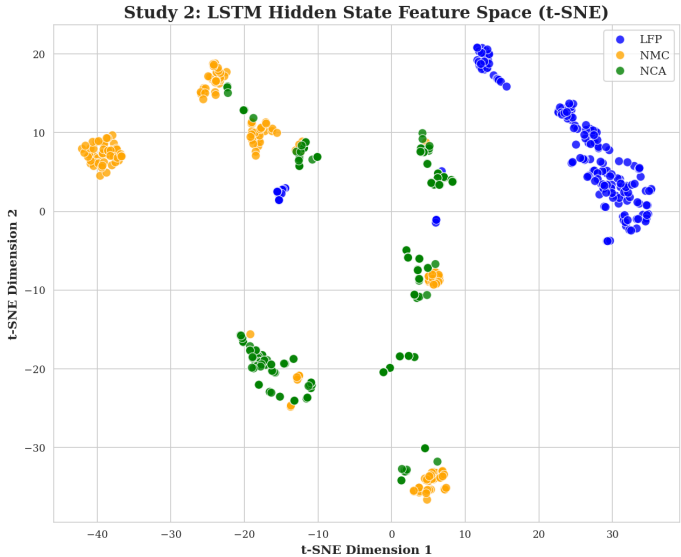


Figure 4.24: Illustration of Latent/hidden state features in 2d space

Ultimately, these backpropagated gradient conflicts, combined with the fragmented latent space, prove that using only LSTM architecture across multiple battery types causes bad interference. The model simply struggles to balance the distinct characteristics of all chemistry simultaneously. Therefore, developing generalized models within a single chemistry domain is a much more reliable approach for accurate State of Health (SoH) estimation. By focusing on one battery chemistry at a time, the network can learn a consistent set of physical relationships without having to continually reconcile conflicting patterns.

Stratified groupfold results for baseline model

However, we need to make sure that this chemistry-specific model truly learns the aging patterns across different conditions, rather than just memorizing the training batteries. To test this, we used the Stratified Group K-Fold cross-validation method (described in Section 3.3.2). Since this approach keeps unique Battery IDs completely separate during testing, it proves the model works reliably on new, unseen data. The results of this cross-validation are presented in Table 4.6.

Cross-validation results show lower accuracy compared to other published research papers. This is because the primary goal was complete generalization for a single battery chemistry under many different conditions.

The dataset combines several profiles such as CC-CV, constant charge and discharge, pulse tests, HPPC tests, and standard drive cycles into a single chunk. Furthermore, this data comes from different labs and uses cells with varying basic features, like capacity. Despite these challenges, the results remain promising, making it possible to continue with this approach.

Table 4.6: Performance of the model across different folds of LFP cells only.

Folds	Chemistry	Number of Batteries (IDs)	RMSE	MAE	R^2
1	LFP	44	0.0250	0.0202	0.8684
2	LFP	44	0.0247	0.0201	0.8441
3	LFP	44	0.0221	0.0180	0.8680
4	LFP	44	0.0242	0.0193	0.8668
5	LFP	44	0.0283	0.0238	0.8242
6	LFP	44	0.0269	0.0230	0.8460

4.3 Model Compression, quantization and Compressed model evaluation

4.3.1 Grow prune strategy

Grow prune framework

Building upon the theoretical framework in Section 3.4.1, the Grow-and-Prune (GP) strategy was executed to compress the baseline LSTM’s memory footprint without much compromising model accuracy.

To identify the optimal sparse architecture, the GP-Active Phase was governed by a comprehensive hyperparameter tournament. This grid search swept target sparsity thresholds from 0.1 to 0.99, while keeping the growth threshold fixed at 0.95 to reactivate only the top 5% of dormant connections with the highest gradient magnitudes. Furthermore, the tournament evaluated structural update frequencies of 10, 20, and 30 epochs to balance training stability with compression speed, along with sparse structural updates and dynamic weight scaling.

As illustrated in Figure 4.25, SOH prediction accuracy across varying pruning ratios showcases three distinct operational regimes for both the Sparse FP32 baseline and the dynamic INT8 Quantization (PTQ) models. In the mild-to-moderate pruning regime (10% to 60% parameter removal), the sparse FP32 model consistently achieves positive relative accuracy gains over the unpruned baseline, peaking at a +7.5% improvement near 40% sparsity. This demonstrates that targeted parameter pruning acts as an effective regularizer, reducing network over-parameterization without loss of predictive capacity.

Furthermore, dynamic INT8 quantization demonstrates remarkable resilience, closely tracking the FP32 trajectory up to 99% parameter removal while staying above or near baseline performance. However, pushing parameter reduction beyond 95% triggers a sharp performance degradation the sparsity cliff where both FP32 and INT8 models collapse to approximately -20% relative accuracy change near 95% pruning ratio. This sharp threshold identifies the absolute structural limit of the LSTM, beyond which the network loses the capacity to retain core battery aging features. Based on these findings, a pruning ratio of 90% to 95% provides the op-

timal trade-off, achieving maximum memory compression while maintaining high accuracy relative to the full-precision baseline. Figures 4.26, 4.27 and 4.28 shows

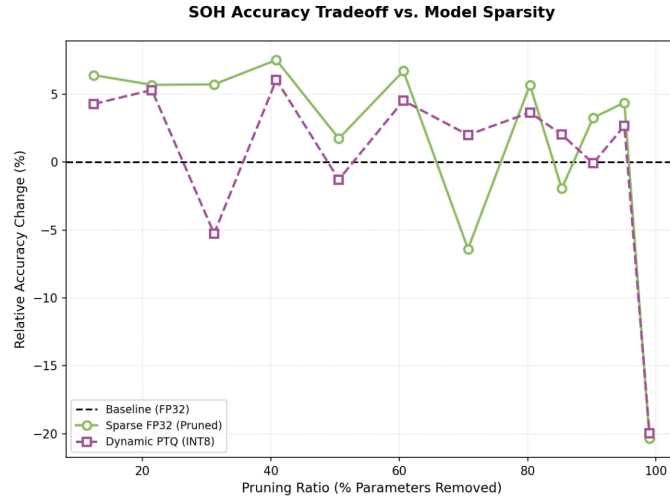


Figure 4.25: Illustration of Pruning ration vs Relative accuracy

examples of comparison results of Dense model, sparsed model and Int8 model :

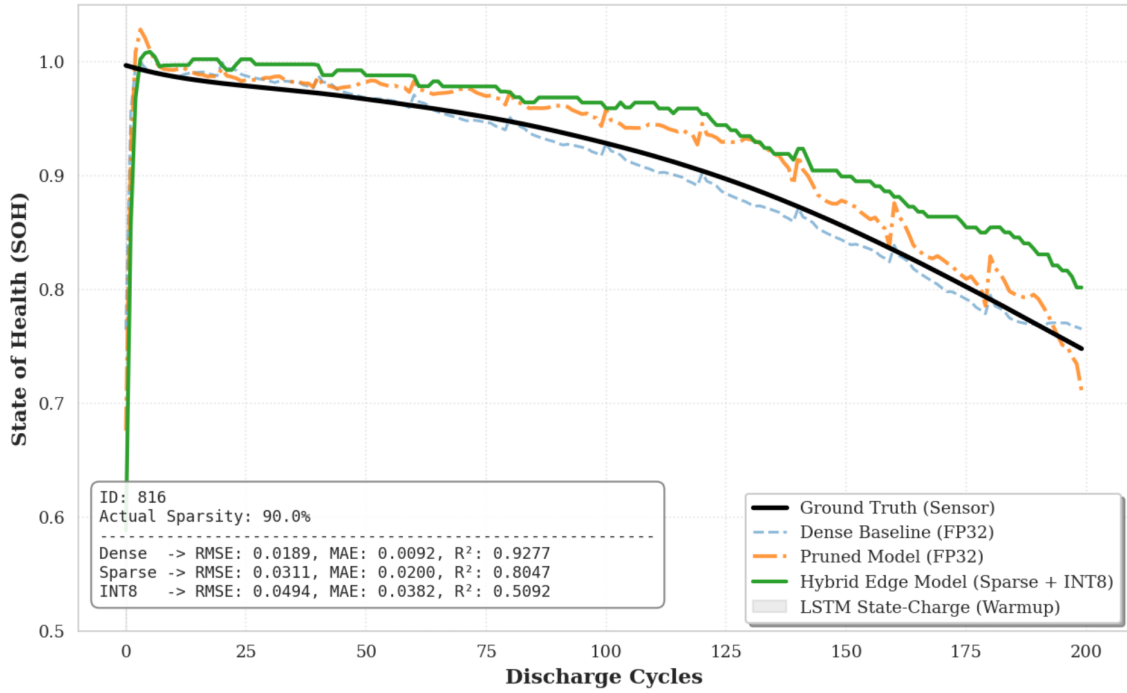


Figure 4.26: HUST dataset results of GP model

Stratified groupfold results for Sparsed GP(F32) model and Sparsed INT8 GP model

Tables 4.7 and 4.8 present the Stratified Group K-Fold cross-validation results for the Grow-and-Prune (GP) models.

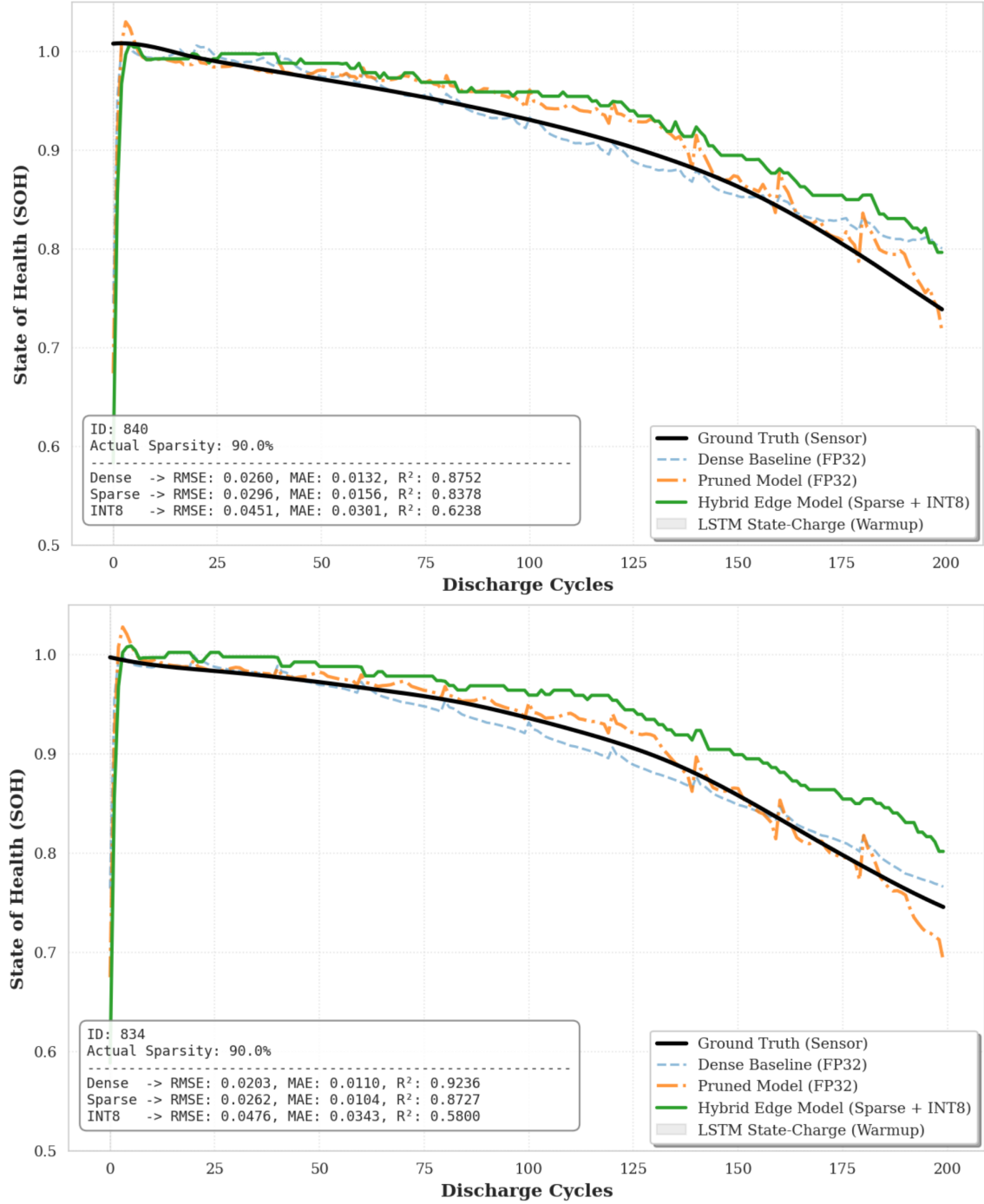


Figure 4.27: HUST dataset results of GP model

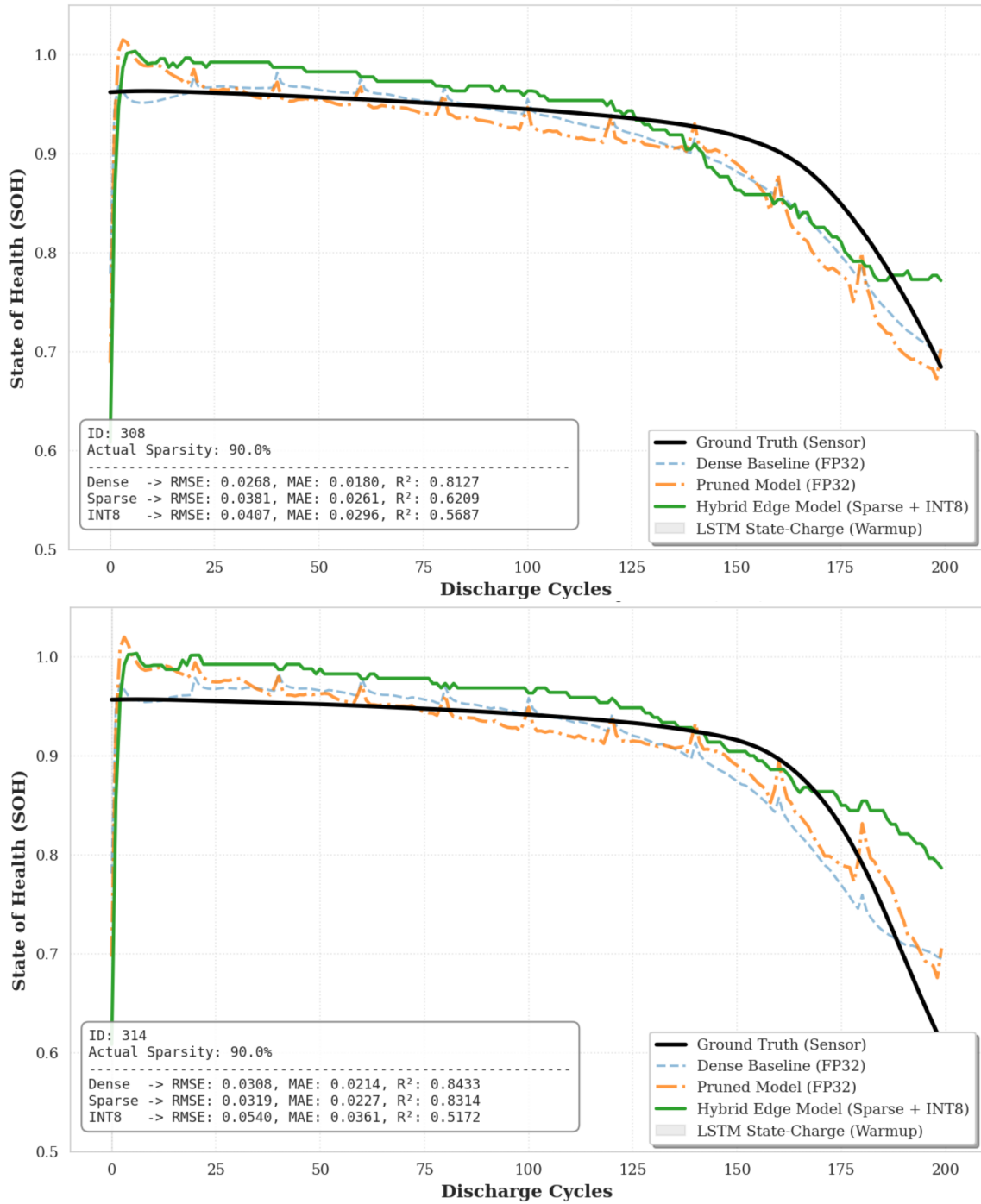


Figure 4.28: MIT dataset results of GP model

As shown in Table 4.7, the standard sparsed GP model shows predictive performance across all six folds, confirming that the pruned network generalizes well to unseen batteries. Table 4.8 displays the results after converting this sparse model into an 8 bit integer (INT8) format for microcontroller deployment. While the INT8 quantization naturally causes an increase in prediction error (RMSE) and a drop in the R^2 score due to the loss of numerical precision, the quantized model successfully preserves overall stability and remains viable for resource-constrained execution.

Table 4.7: Performance of the Sparsed GP model across different folds of LFP cells only.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.01369	0.01089	0.8856
2	44	0.03649	0.02519	0.7368
3	44	0.02136	0.01510	0.8337
4	44	0.03510	0.02233	0.7264
5	44	0.02139	0.01660	0.7938
6	44	0.03123	0.01932	0.7666

Table 4.8: Performance of the Sparsed int8 GP model across different folds of LFP cells only.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.0438	0.0280	0.5552
2	44	0.0422	0.0264	0.5814
3	44	0.0457	0.0303	0.5331
4	44	0.0418	0.0257	0.5904
5	44	0.0442	0.0284	0.5501
6	44	0.0431	0.0275	0.5636

Profiling metrics of Grow Prune

In the pruned model, redundant weights are simply set to zero, but the overall structure of the network remains exactly the same. Because the matrix dimensions do not change, the hardware still exerts the same computational effort, resulting in RAM consumption that is nearly identical to the baseline model.

4.3.2 Compression Aware training

Following the theory from Section 3.4.2, we applied Compression Aware Training (CAT) to the baseline model to shrink its size while maintaining predictive accuracy. To find the best configuration, we controlled the level of compression using the regularization factor (τ). We tested τ values of [0.0009, 0.001, 0.005, 0.009, 0.01]. This parameter acts as a penalty during training, dictating how aggressively the network

is forced to reduce its internal capacity and adapt to the compression.

We evaluated each τ value using the Stratified Group K-Fold cross-validation method to observe its direct impact on the final model size and the predictive error (RMSE). The results demonstrate a clear trade-off between compression and accuracy.

As the τ penalty increases, the network is forced to discard more parameters from its original baseline of 32,545. For example, a lower penalty of $\tau = 0.0009$ reduced the parameters by 43.52% (leaving 18,382 parameters), maintaining a low RMSE of 0.0303. A slight increase to $\tau = 0.001$ resulted in a 46.96% reduction (17,262 parameters) with an RMSE of 0.0332. Pushing τ to 0.005 led to a much more aggressive 89.23% reduction (leaving only 3,506 parameters), but this loss of capacity naturally increased the RMSE to 0.0520. The results also reveal a physical limit to this compression technique. At $\tau = 0.009$, the model reaches a maximum parameter reduction of 93.66% (2,062 parameters) with an RMSE of 0.0589. Increasing τ further to 0.01 does not compress the model any further, as the parameter count remains exactly 2,062, but the prediction error continues to degrade, reaching an RMSE of 0.0663. This indicates that pushing the regularization penalty too high eventually damages the network’s predictive ability without providing any additional memory savings.

Based on these findings, we selected $\tau = 0.001$ as the final configuration, as it provided the best balance of stable predictions and a significantly reduced weight count. Some example of results plot of compression aware training shown in figure 4.29 and 4.30.

As shown in Table 4.9, the Compression-Aware Training (CAT) student model demonstrates strong predictive performance across all six folds.

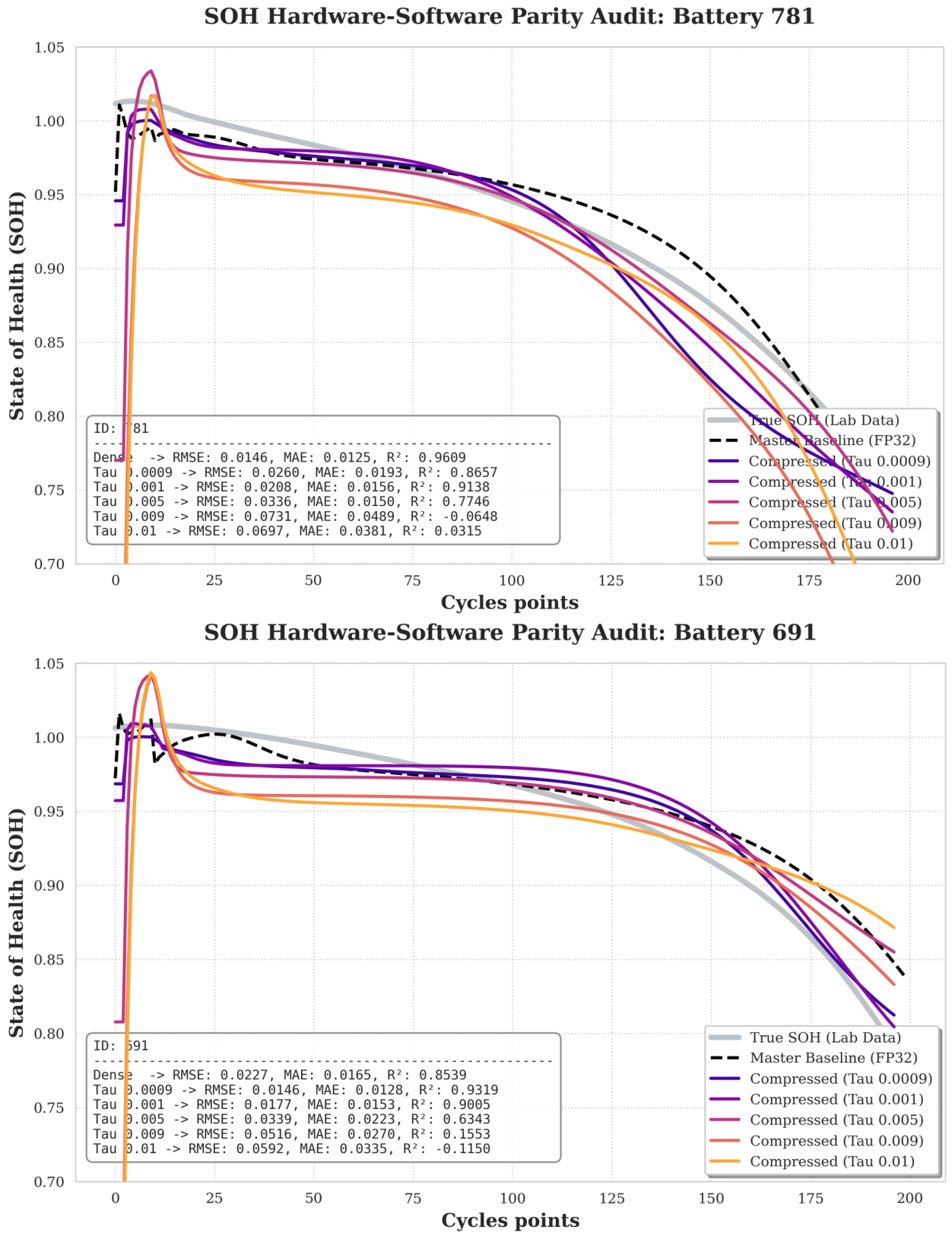
Table 4.9: Performance of the CAT model ($\tau = 0.001$) FP32 model across different folds.

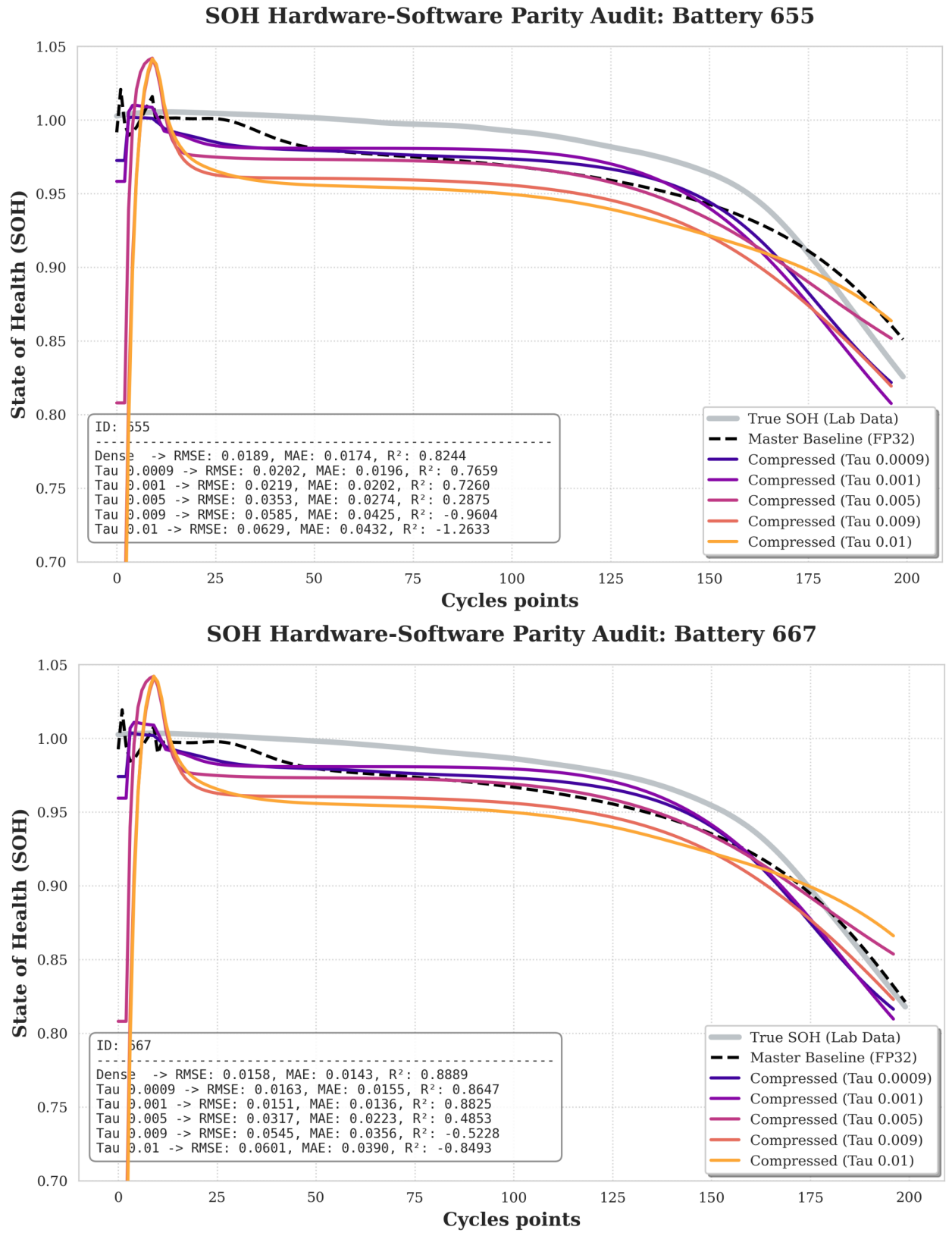
Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.01804	0.01508	0.86400
2	44	0.02101	0.01759	0.85457
3	44	0.02484	0.02027	0.82313
4	44	0.02323	0.01840	0.84509
5	44	0.02055	0.01691	0.87810
6	44	0.02415	0.01892	0.81455

4.3.3 Knowledge Distillation

4.3.3.1 Relational Knowledge distillation

To identify the optimal configuration for Relational Knowledge Distillation (RKD), an experimental grid search was conducted to balance the transfer of latent feature relationships from the dense teacher model, which utilizes a two-layer [64, 32] LSTM





architecture, to a compressed student model. While there were no strict capacity restrictions imposed during this selection, the primary objective was to discover the minimal architecture that successfully maintained predictive performance while preserving the fundamental two-layer LSTM structure.

Consequently, the search evaluated the student’s hidden dimension capacity (sweeping across 4,8,16 and 32 hidden size) alongside a combination of distillation penalties. The response loss weight ($\lambda_{response}$) was iteratively swept from 0.1 to 0.8 in increments of 0.1 to determine exactly how heavily the student is penalized for diverging from the teacher’s output distributions versus learning from the raw training data. Concurrently, the relational distance ($\lambda_{distance}$) and angle (λ_{cosine}) loss weights were tuned to govern the structural mapping of the latent feature space.

From the results of the grid search, we identified the most effective and minimal configuration of student architecture is [8,8] hidden units, paired with the specific loss weighting formulation of $\lambda_{response} = 0.8$, $\lambda_{distance} = 0.12$, and $\lambda_{cosine} = 0.08$. Examples in figures 4.31 and 4.32 shows the comparison results of Teacher baseline results, Student F32 model results and Student INT8 results

Stratified groupfold results for Relational-KD student model

As shown in Table 4.11, the Relational KD student model shows predictive performance across all six folds, confirming that the distilled network effectively captures the teacher model’s representations and generalizes well to unseen batteries. Table 4.12 displays the results after converting this distilled model into an 8-bit integer (INT8) format for microcontroller deployment. While the INT8 quantization naturally causes a slight increase in prediction error (RMSE) and a drop in the R^2 score due to the loss of numerical precision, Still the still remains highly stable across the different folds.

Table 4.10: Performance of the R-KD student model across different folds.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.019882	0.015360	0.891133
2	44	0.020501	0.016156	0.879280
3	44	0.022715	0.018783	0.819519
4	44	0.018705	0.015103	0.878988
5	44	0.023404	0.018317	0.849594
6	44	0.023599	0.017760	0.817788

4.3.3.2 PINN Knowledge distillation

To find the best setup for the Physics-Informed Knowledge Distillation (PINN KD) model, a grid search was conducted where the model is tested with a range of hidden sizes of [4,8,16,32]. At the same time, tested with different penalty weights for the physics-based loss function, specifically trying α and β values of [0.001, 0.005, 0.01,

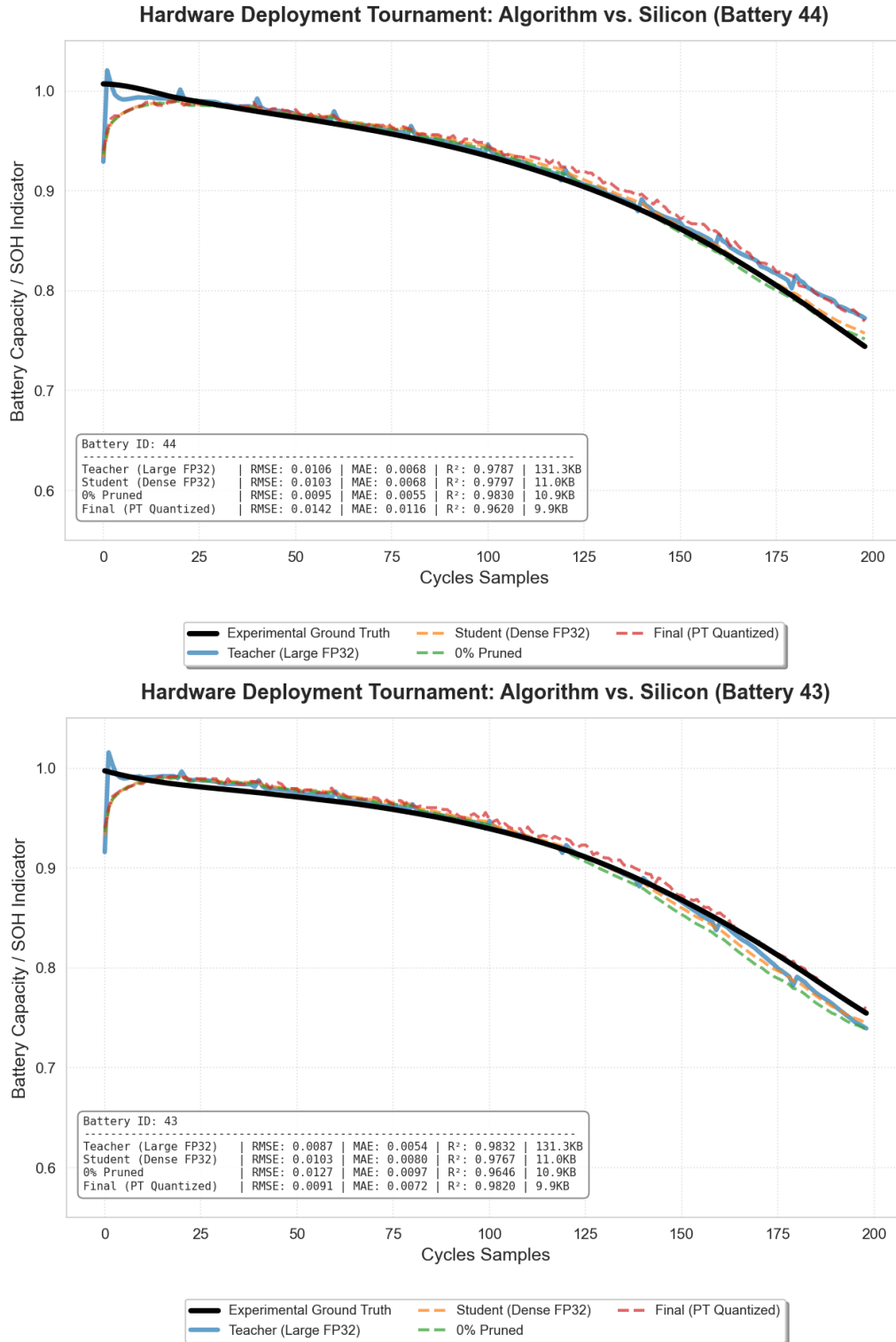


Figure 4.31: HUST dataset results of Relational KD

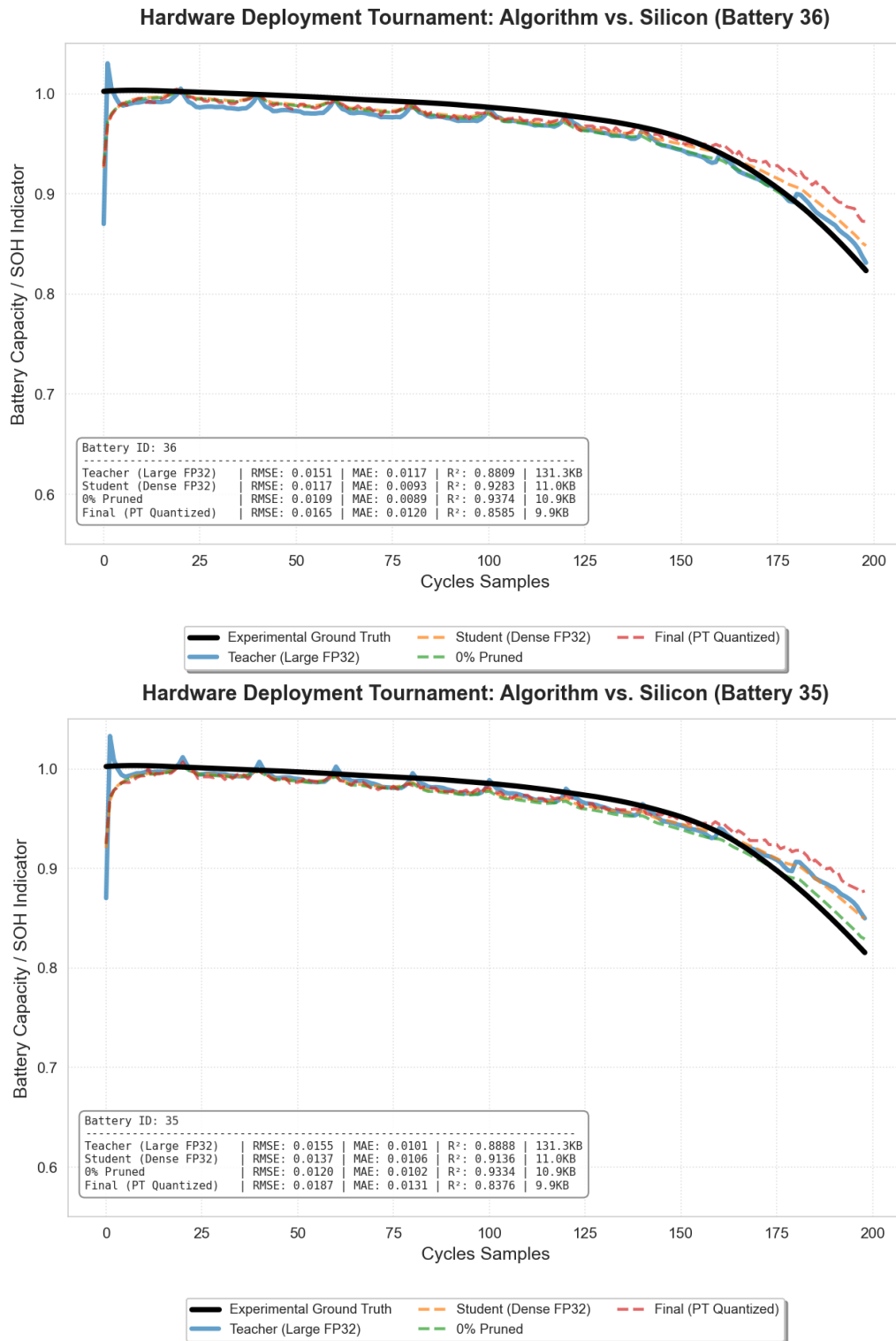


Table 4.11: Performance of the INT8 model with relational knowledge distillation across different folds.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.020504	0.015669	0.889392
2	44	0.023794	0.018913	0.803239
3	44	0.023384	0.018820	0.820828
4	44	0.020250	0.016433	0.853731
5	44	0.025257	0.020293	0.794764
6	44	0.023480	0.017783	0.816934

0.05]. These α and β weights control how much hardloss and softloss need to be considered. The search showed that the most effective configuration is a student model with 8 hidden units, using physical constraint penalties of $\alpha = 0.05$ and $\beta = 0.05$.

Stratified groupfold results for PINN-KD student model

Tables 4.15 and 4.16 showcase the cross-validation results for PINN-KD model F32 and PINN-KD model INT8.

Table 4.12: Performance of the PINN-KD student FP32 model across different folds.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.02326	0.01728	0.89347
2	44	0.01957	0.01525	0.92351
3	44	0.02368	0.01747	0.85229
4	44	0.01958	0.01496	0.89507
5	44	0.01668	0.01306	0.92474
6	44	0.01191	0.00917	0.95177

Table 4.13: Performance of the PINN-KD student INT8 model across different folds.

Fold	Number of Batteries (IDs)	Average RMSE	Average MAE	Average R^2
1	44	0.02305	0.01658	0.88854
2	44	0.02085	0.01635	0.91075
3	44	0.02760	0.02167	0.81893
4	44	0.02215	0.01647	0.84451
5	44	0.01811	0.01465	0.90502
6	44	0.01498	0.01182	0.91736

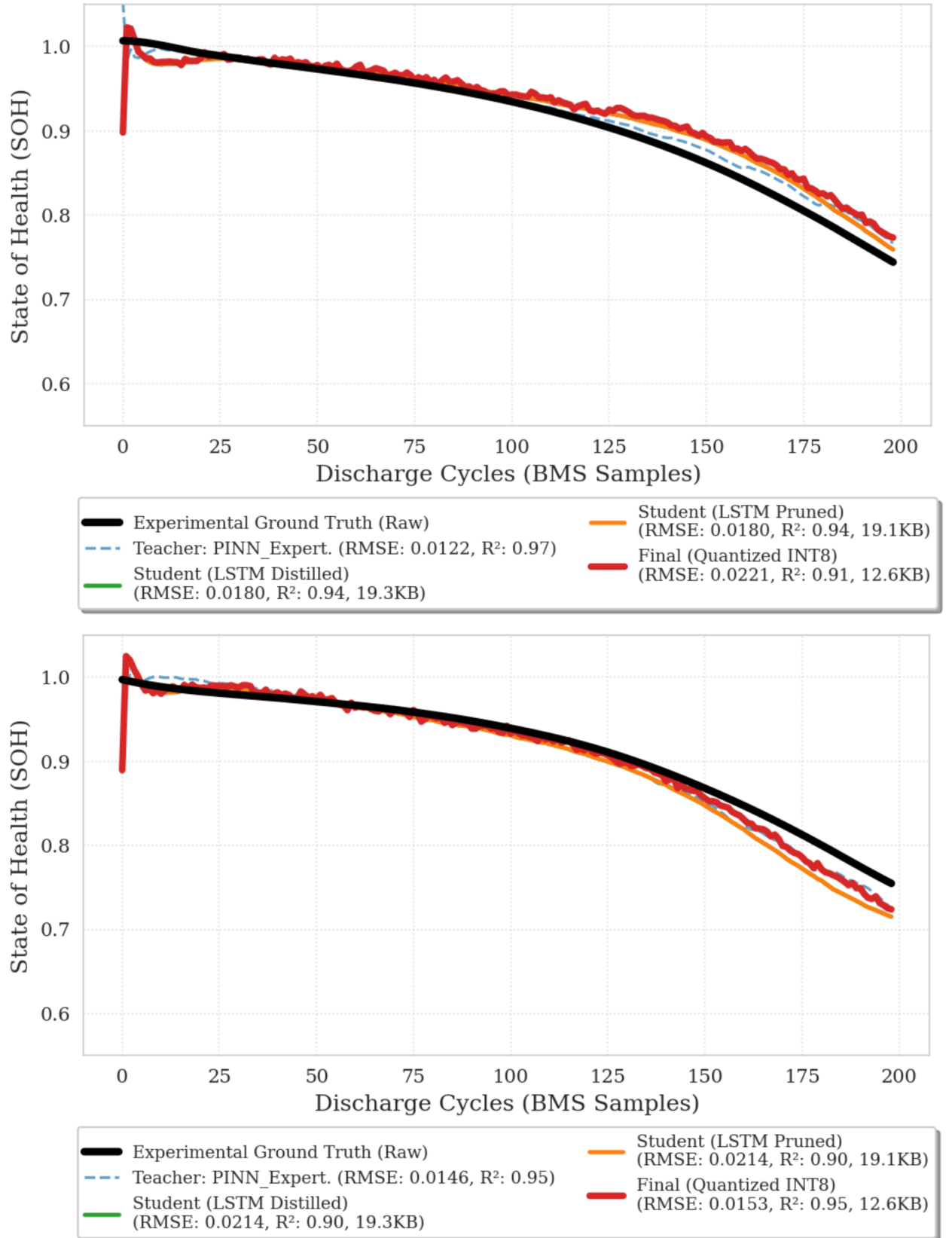


Figure 4.33: HUST dataset results of PINN-KD.

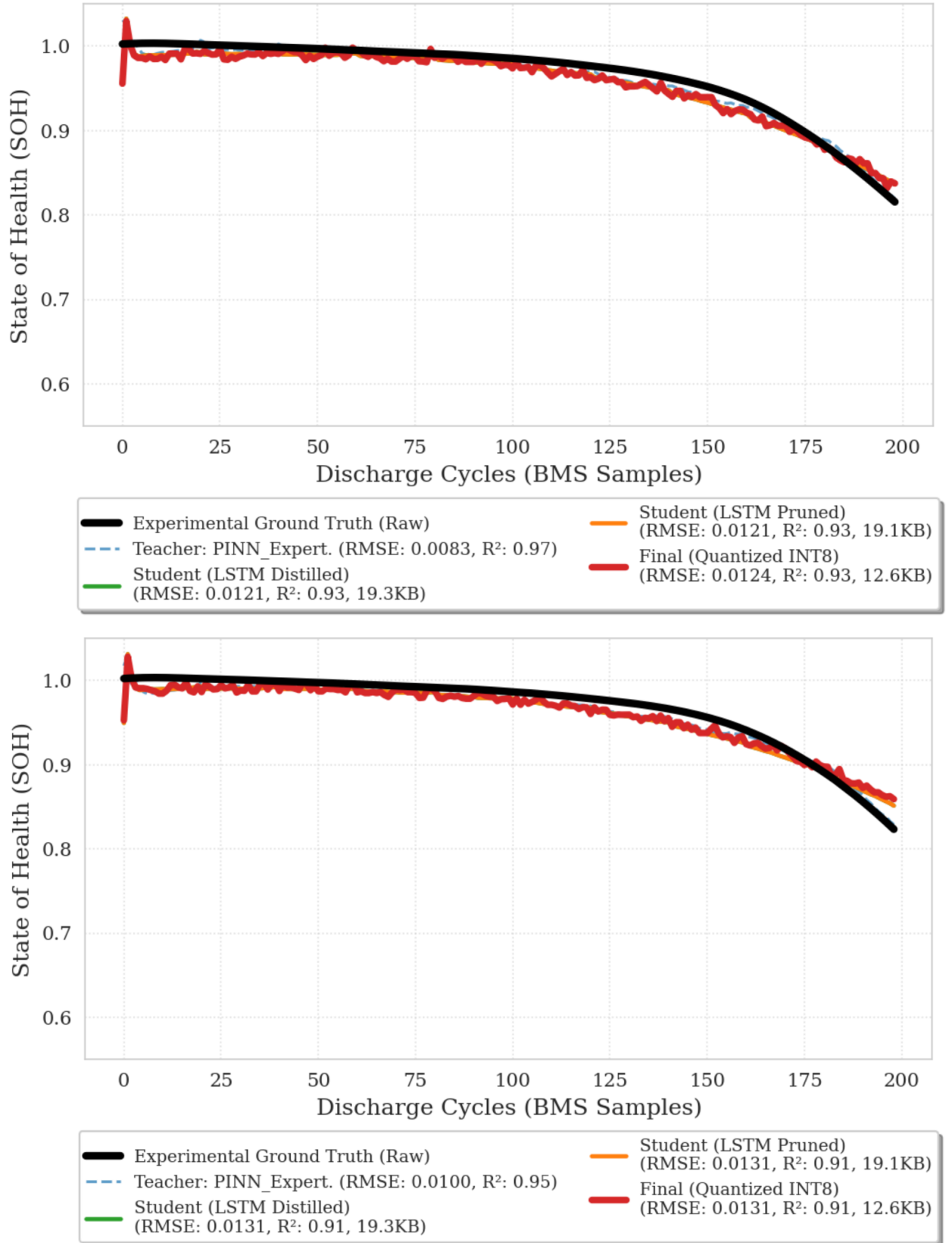


Figure 4.34: MIT dataset results of PINN-KD.

4.4 Models Performance profiling and Final model selection

Building upon the analytical framework and resource metric equations established in Section 3.5, this section presents the empirical hardware profiling results for the State of Health (SoH) estimation models. While baseline deep learning architectures achieve high predictive fidelity in unconstrained environments, successful deployment on an embedded Battery Management System (BMS) requires strict adherence to the target microcontroller’s physical limitations.

4.4.1 Hardware Target Constraints (NXP MCXN947)

The deployment platform for this evaluation is the NXP MCXN947 microcontroller, powered by a dual-core ARM Cortex-M33 architecture. During standard inference operations, the core runs at 150 MHz, drawing an active current density of 0.47 mA/MHz under a nominal supply voltage of 3.3 V. To accommodate resource-constrained edge applications, the physical memory boundaries provide a maximum of 2048 KB of non-volatile Flash storage for static model parameters and 512 KB of volatile SRAM for dynamic runtime activations.

4.4.2 Empirical Profiling and Comparative Analysis

Testing candidate designs against hardware limits means converting math formulas for power usage, memory (Flash and SRAM), and processing speed into real-world measurements. This analysis shows the actual effects of model compression. Specifically, it compares structural weight pruning (using the Grow-Prune framework) with smaller architectures built through Knowledge Distillation (Relational-KD and PINN-KD), as well as post-training INT8 quantization.

Grow-Prune (GP) Framework Profiling

Table 4.14 summarizes the hardware metrics for the Grow-Prune framework. While the uncompressed baseline maintains 0.0% sparsity with 32,545 parameters, introducing 90.2% sparsity reduces active parameters to 3,181. This structural reduction shrinks the static Flash footprint from 131.30 KB down to 14.91 KB, drops total latency from 61.11 ms to 6.25 ms, and lowers inference energy from 14.22 mJ to 1.45 mJ. Applying INT8 quantization to the pruned architecture further optimizes the footprint, compressing Flash size to 3.73 KB and peak SRAM to 47.84 KB.

Table 4.14: Comparison of model compression and deployment metrics.

Stage	Format	Units	Sparsity	Params	Flash Size	Peak SRAM	Lat (Sum)	Energy
Baseline	FP32	[64, 32]	0.0%	32,545	131.30 KB	167.38 KB	61.11 ms	14.2178 mJ
Pruned	FP32	[64, 32]	90.2%	3,181	14.91 KB	167.38 KB	6.25 ms	1.4546 mJ
INT8	INT8	[64, 32]	90.2%	3,181	3.73 KB	47.84 KB	6.18 ms	1.4376 mJ

Knowledge Distillation Frameworks (Relational-KD and PINN-KD) Profiling

In contrast to sparse training, the Knowledge Distillation framework alters network topology by scaling hidden layer dimensions down from [64, 32] to [8, 8]. Because hardware profiling metrics are strictly a function of structural parameter counts, network topology, and data precision rather than the underlying training loss function, the physical execution footprint remains identical across different distillation strategies. Consequently, the profiling results for Relational-KD and PINN-KD yield matching performance metrics, as presented in Table 4.15 and Table 4.16. Through structural reorganization, the distilled student network reduces the total parameter count to 1,289. Combining knowledge distillation with INT8 quantization achieves the most aggressive footprint reduction, scaling static Flash memory down to 1.29 KB, reducing peak SRAM to 15.03 KB, dropping total latency to 1.61 ms, and lowering inference energy consumption to 0.3751 mJ.

Table 4.15: Comparison of model compression and estimated profiling metrics of Relational-KD.

Stage	Format	Units	Params	Flash Size	Peak SRAM	Total Latency (Sum)	Energy
Teacher Baseline	FP32	[64, 32]	32,545	131.3 KB	167.38 KB	60.40 ms	14.05 mJ
Dense Distilled Student	FP32	[8, 8]	1,289	5.16 KB	36.13 KB	4.61 ms	1.0726 mJ
Structural Pruned (0%)	FP32	[8, 8]	1,289	5.16 KB	36.13 KB	4.61 ms	1.0726 mJ
Final (KD + INT8)	INT8	[8, 8]	1,289	1.29 KB	15.03 KB	1.61 ms	0.3751 mJ

Table 4.16: Comparison of model compression and estimated profiling metrics of PINN-KD

Stage	Format	Units	Params	Flash Size	Peak SRAM	Total Latency (Sum)	Energy
PINN Teacher Baseline	FP32	[64, 32]	32,545	131.3 KB	167.38 KB	60.40 ms	14.05 mJ
PINN Distilled Student	FP32	[8, 8]	1,289	5.16 KB	36.13 KB	4.61 ms	1.0726 mJ
Structural Pruned (0%)	FP32	[8, 8]	1,289	5.16 KB	36.13 KB	4.61 ms	1.0726 mJ
PINN Student (KD + INT8)	INT8	[8, 8]	1,289	1.29 KB	15.03 KB	1.61 ms	0.3751 mJ

4.4.3 Final Model Selection

Testing four compression strategies—Grow-and-Prune (GP), Compression-Aware Training (CAT), Relational Knowledge Distillation (RKD), and Physics-Informed Knowledge Distillation (PINN-KD)—showed that the student model from the PINN-KD framework is reliable for embedded deployment.

While heavy 8-bit integer (INT8) quantization usually hurts accuracy because of noise, the PINN-KD student model stayed remarkably resilient. Furthermore, the PINN-KD student model achieved greater and promising results across all six-fold comparisons compared to the other approaches. Because the distillation process regularizes the model using fundamental laws of battery physics, the architecture gains extra structural robustness.

Consequently, the PINN-KD student model successfully maintained its baseline prediction accuracy even after conversion to the heavily compressed INT8 format. This makes it the ideal choice for real-time edge execution: it achieves the extreme memory and energy reductions required for resource-constrained microcontrollers without sacrificing predictive performance.

4.5 Model Compression and Resource Optimization

This section evaluates the effect of the applied optimization pipeline, quantifying the reduction in memory utilization and the mathematical fidelity of the compiled TVM C-engine against the original PyTorch environment.

4.5.1 Flash Storage and SRAM Footprint Reduction

The physical memory limits of automotive microcontrollers necessitate a severe reduction in both non-volatile storage (Flash) for model weights and volatile memory (SRAM) for intermediate tensor workspaces. The profiling data collected during the TVM compilation phase demonstrates a dramatic decrease in resource consumption across the three model iterations.

In the Figure 4.35 illustrates the uncompressed, full-precision Physical-Informed Neural Network (PINN) expert model requires 130,180 bytes (approximately 127.13 KB) of Flash storage. Furthermore, the static tensor workspace required by the Ahead-of-Time (AOT) executor for the expert model peaks at 156,256 bytes (152.59 KB) in SRAM. This memory requirement exceeds the capacity of many low-tier BMS microcontrollers.

Following the knowledge distillation phase, the compressed FP32 model exhibits a massive reduction in size. The distilled model’s weights consume only 5,156 bytes (5.04 KB) of Flash memory. Similarly, the SRAM workspace requirement drops significantly to 5,280 bytes (5.16 KB).

As detailed in the custom W8A32 pipeline(Section 3.6.3), by keeping activations in FP32 while compressing weights to INT8, the model achieved a massive reduction in Flash storage without suffering the temporal precision collapse typically seen in fully quantized LSTM networks. The final optimization stage, employing a W8A32 quantization strategy, further shrinks the physical storage footprint. The quantized model’s weights, exported directly as a C-header array, occupy merely 1,289 bytes (1.26 KB) of Flash memory. For this variant, the SRAM is managed dynamically by the TVM Relax Virtual Machine Memory Arena, replacing the static workspace allocation.

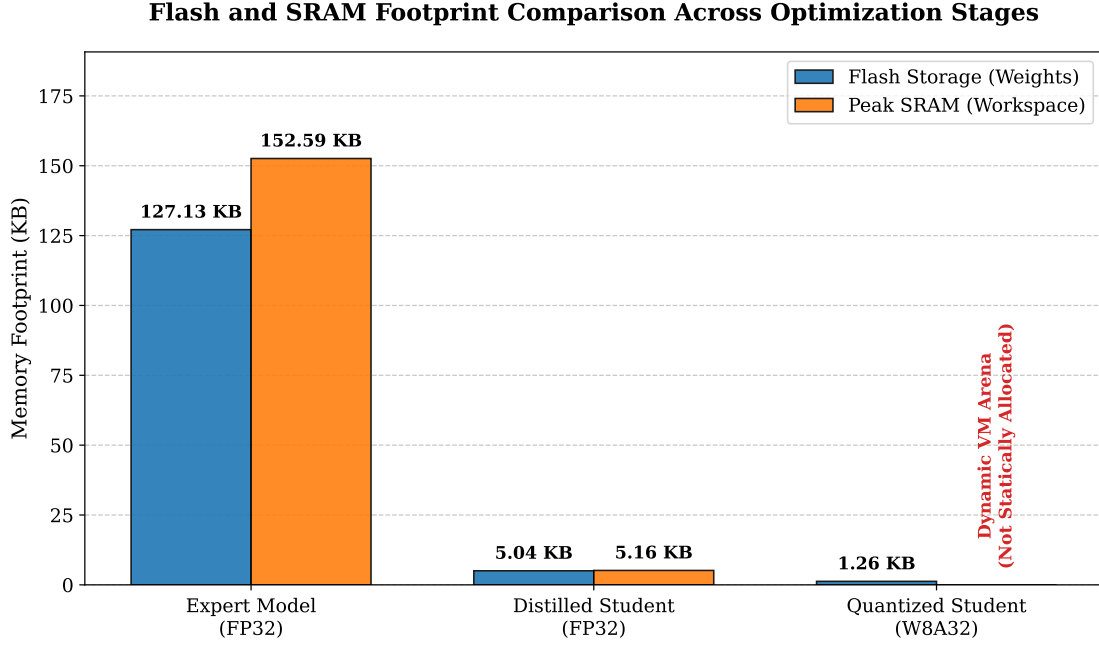


Figure 4.35: Comparison of Flash and SRAM memory footprints across the models.

4.5.2 Mathematical Fidelity and Inference Accuracy (PC SIL)

Translating a high-level PyTorch model into a bare-metal C application introduces the risk of numerical degradation, particularly when managing the long-term temporal states of an LSTM. To validate the mathematical fidelity of the TVM-generated engine prior to physical deployment, a batch execution diagnostic was performed across the 200 time-step feature sequences in a PC environment.

The AOT-compiled FP32 PINN Expert model achieved a remarkably low Mean Absolute Error (MAE) of approximately 2.83×10^{-7} against the native PyTorch predictions, with a maximum absolute error bounded at 8.33×10^{-7} . The Distilled FP32 model maintained a similar level of precision, demonstrating an MAE of 2.63×10^{-7} and a maximum error of 5.49×10^{-7} .

Crucially, the W8A32 Quantized model, running within the Relax VM engine via symmetric quantization and dynamic dequantization strategies, exhibited an exceptional MAE of approximately 3.32×10^{-8} against the PyTorch baseline. These negligible error margins confirm that the TVM compiler completely preserved the mathematical integrity of the SoH estimations, clearing the path for real-world edge execution.

We can see that the quantized model has a smaller MAE. For full-precision (FP32) models, the translation from PyTorch to bare-metal C introduces slight floating-point drifts due to platform-specific rounding mechanisms and operation reordering. Conversely, the W8A32 quantized model relies on discrete integer formats for its core weights, which drastically minimizes these continuous floating-point alignment

discrepancies. Ultimately, this discrete mathematical structure enforces stricter execution determinism between the TVM engine and the native PyTorch baseline, resulting in superior cross-platform mathematical alignment. In any case, they all fall within the range of negligible error.

As visually demonstrated in Figure 4.36, the output trajectories of the native PyTorch prediction and the TVM Relax VM engine are virtually indistinguishable across multiple testing datasets, further confirming the robustness of the applied symmetric quantization strategy. The distilled results are relatively closer to the ground truth, but the numerical differences between the two are very small.

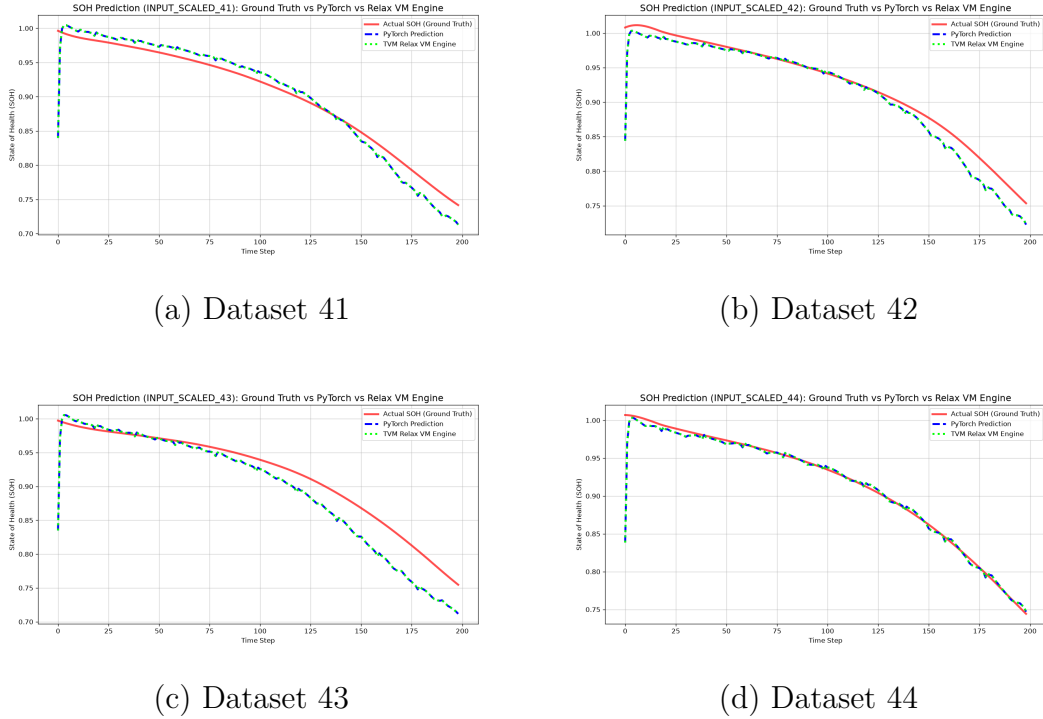


Figure 4.36: Mathematical fidelity validation of the W8A32 Quantized model.

4.6 Real-Time Embedded Performance

Having mathematically validated the compiled AI engines, the deployment transitioned to real-time physical execution on the target automotive edge hardware.

4.6.1 Execution Validation on Infineon TC4D7

The Software-in-the-Loop (SIL) deployment on the Infineon TC4D7 successfully validated the deterministic execution of the compiled TVM models in a bare-metal environment. Utilizing a similar feature-injection protocol, the inference loop was verified to execute smoothly without dynamic memory allocation or RTOS context-switching overhead. This qualitative validation confirmed the fundamental compatibility of the generated C-engine with strict automotive MCU constraints. Following

this architectural validation, the comprehensive quantitative latency benchmarking and concurrent PIL execution testing were consolidated on the NXP MCXN947 platform.

4.6.2 Inference Performance Benchmarking on MCXN947

By isolating the TVM inference engine and feeding pre-computed features directly into the network, the absolute execution latency was measured across 10 independent battery datasets on the MCXN947. This methodology explicitly controls for CAN communication overhead and dynamic feature scaling floating-point variations, focusing purely on the computational limits of the models.

As illustrated in Figure 4.37, the PINN-KD Distilled model achieved a dramatic reduction in inference time compared to the heavily parameterized Expert model. While the Expert model successfully executed without memory overflow, its complex matrix multiplications consumed a significant number of CPU cycles, increasing the risk of violating hard real-time deadlines under high bus loads. In contrast, the Distilled model completed execution in strictly deterministic timeframes, ensuring highly responsive BMS control loop integration.

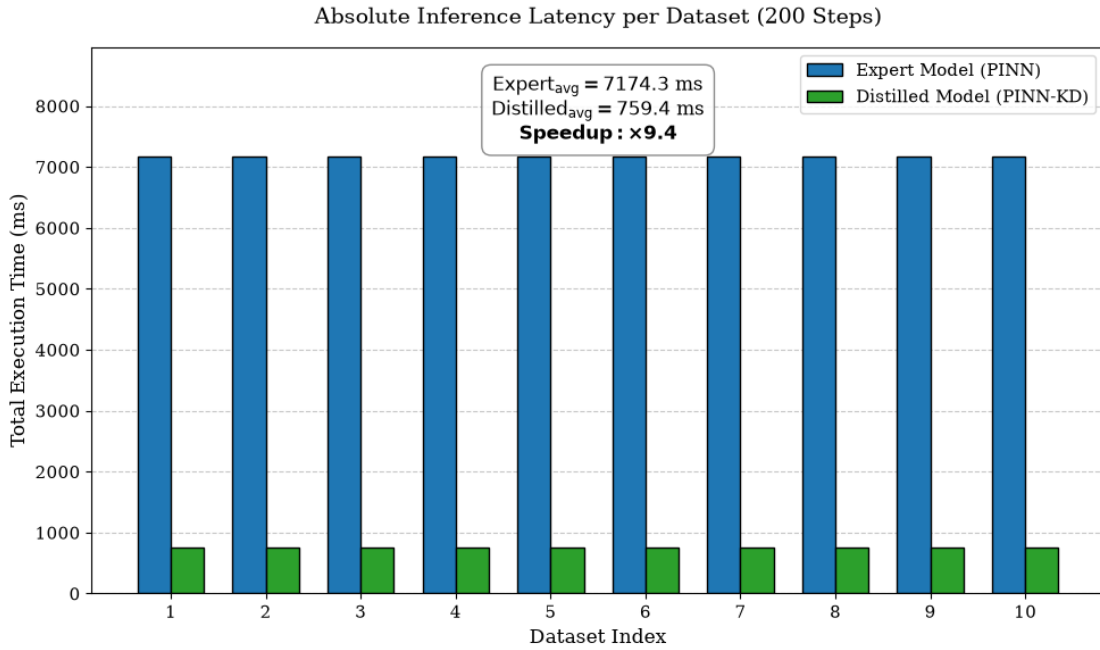


Figure 4.37: Latency comparison between the Expert Model and Distilled Model across 10 independent datasets executed on the NXP MCXN947.

4.6.3 Cross-Platform Mathematical Fidelity (Silicon vs PC)

To further validate the physical edge deployment, it is critical to ensure that the execution on the ARM Cortex-M33 silicon matches the mathematical outputs generated by the PC-based TVM simulation. Using the same isolated feature injec-

tion protocol, the SoH prediction trajectories were recorded directly from the NXP MCXN947.

As illustrated in Figure 4.38, the physical execution on the microcontroller establishes exceptional mathematical fidelity when compared to both the PC-based TVM simulation and the experimental Ground Truth. The PC SIL predictions (dashed lines) and the physical MCU outputs (dotted lines) are nearly indistinguishable. This confirms that the custom W8A32 dynamic quantization pipeline successfully mitigates hardware-induced floating-point drift and preserves computational determinism across distinct architectures.

Quantitatively, the deployed model shows highly robust predictive performance. A critical indicator of this stability is the tight bounding of the RMSE to the MAE. For instance, Dataset 41 yields an RMSE of 0.0168 alongside an MAE of 0.0131, while Dataset 44 achieves an exceptional RMSE of 0.0127 and an MAE of 0.0055. Because RMSE strictly penalizes extreme deviations, this minimal discrepancy crucially indicates the absence of severe outlier predictions. It proves that the model maintains absolute mathematical stability under edge hardware execution, averting any catastrophic miscalculations that could compromise the safety of the Battery Management System.

Furthermore, the model’s capacity to capture complex, long-term degradation trajectories remains completely intact. Across the evaluated datasets, the Coefficient of Determination (R^2) consistently exceeds 0.93 in scenarios, peaking at 0.9696 for Dataset 44, and maintains a solid 0.8776 even in the most challenging profile (Dataset 43). Ultimately, this comprehensive evaluation proves that the physical edge deployment faithfully captures the entire physical aging variance, validating the algorithmic robustness of the PINN-KD Distilled model on bare-metal silicon.

4.6.4 Concurrency and End-to-End Challenges in System Integration

Since the isolated inference engine was proven to be both mathematically accurate (Section 4.5.2) and computationally deterministic (Section 4.5.1), the end-to-end data acquisition pipeline was subsequently validated utilizing the FlexCAN double-buffering architecture.

The system demonstrated exceptional software-hardware stability. The high-priority ISR successfully captured high-frequency asynchronous sensor streams without blocking the AI execution loop, and the `overflow_counter` confirmed zero frame drops. Additionally, the `vector_is_finite` boundary checks effectively prevented NaN-induced cascade failures within the LSTM.

However, despite this architectural stability, the real-time on-device feature extraction yielded sub-optimal numerical predictions in the final SoH estimation. With the neural network itself ruled out as the source of error, this degradation is definitively attributed to the amplification of transient bus noise during on-device statistical calculations, coupled with floating-point accumulation errors during real-time capacity integration. Static normalization boundaries derived from clean laboratory

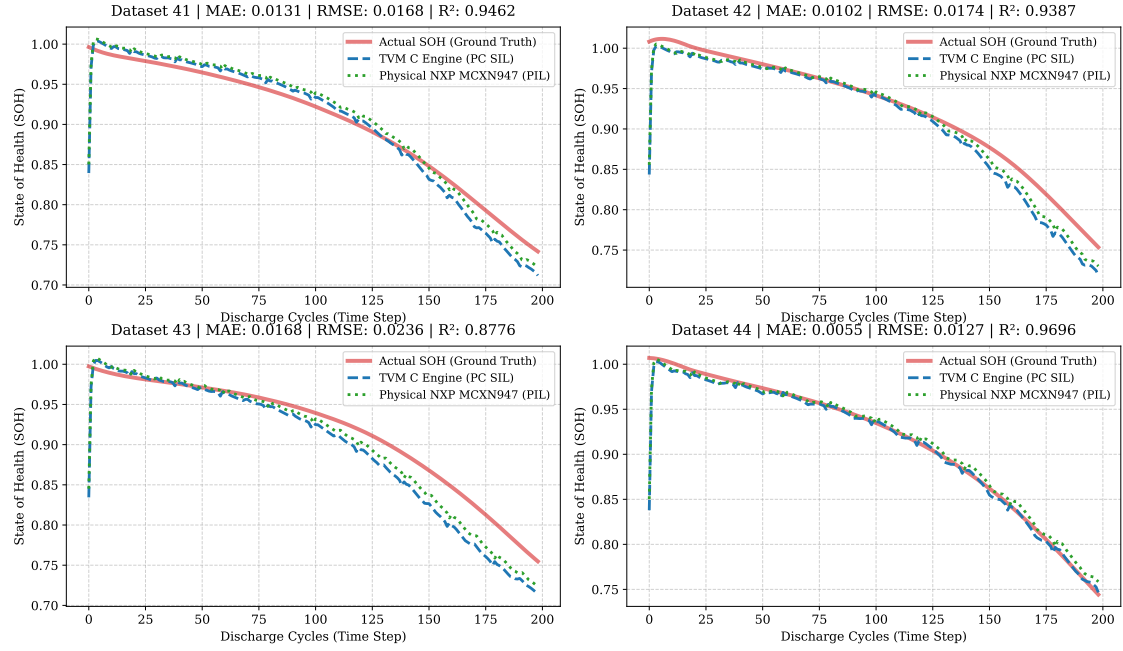


Figure 4.38: Cross-platform mathematical fidelity. The physical execution on the NXP MCXN947 closely aligns with the PC simulation, with minor acceptable hardware drift stemming from MCU floating-point rounding mechanisms.

data struggled to adapt to the dynamic noise inherent in live CAN traffic. This highlights a critical boundary between mathematically sterile validation environments and dynamic physical edge deployments.

5

Future Work and Conclusion

In conclusion, this research successfully established an end-to-end framework for deploying highly compressed, physics-informed State of Health (SoH) estimation models onto resource-constrained embedded microcontrollers. The primary findings and contributions of this work are summarized as follows:

- **Multi-Chemistry Limitations & Specialization:** The investigation proved that standard LSTM architectures fail to build a unified, chemistry-agnostic model across mixed chemistries (LFP, NMC, and NCA) due to severe gradient dominance and representational misalignment, confirming that specialized single-chemistry modeling remains the most reliable path.
- **Compression Strategy Benchmarking:** Out of four benchmarked compression frameworks, the Physics-Informed Knowledge Distillation (PINN-KD) approach emerged as the most resilient, successfully preserving high predictive performance even after aggressive 8-bit integer (INT8) quantization.
- **Deterministic Edge Compilation:** By transitioning the deployment pipeline to the Apache TVM compiler stack for ahead-of-time (AOT) bare-metal C generation, the optimized model achieved extreme memory reduction and deterministic execution timing.
- **Rigorous Hardware Validation:** The complete edge deployment workflow was successfully validated through Processor-in-the-Loop (PIL) testing, proving that physics-informed compressed models can reliably meet the strict real-time constraints of resource-constrained automotive and industrial microcontrollers.

While this thesis successfully established a robust framework for deploying highly compressed, physics-informed LSTM models onto embedded microcontrollers, several avenues remain open for future research and optimization:

- **Advanced Architectures and Feature Engineering for Chemistry-Agnostic Generalization:** While this study utilized a compact LSTM baseline, training a universal model across mixed battery chemistries (LFP, NMC, and NCA) introduced severe gradient dominance—particularly driven by NCA data—that degraded generalized accuracy. Future research should explore advanced modeling paradigms, such as attention-based Transformers or enhanced physics-informed architectures, which are better equipped to capture complex, multi-domain electrochemical dynamics. Furthermore, incorporating explicit auxiliary inputs such as chemistry-specific one-hot encodings, learned categorical embeddings, and dynamic operational indicators (e.g., charging versus

discharging states or fast-charging profiles) can provide the network with vital contextual information. Combining these richer feature spaces and advanced architectures with gradient balancing techniques (such as GradNorm) or Un-supervised Domain Adaptation (UDA) will be critical to achieving a robust, high-performance, chemistry-agnostic edge estimator.

- **Hardware-in-the-Loop (HIL) and Real-Vehicle Validation:** The current verification framework relies on Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) architectures using static operational data fed via CAN bus simulators. The logical next step is transitioning to a full Hardware-in-the-Loop (HIL) setup connected to a physical battery pack and a complete BMS prototype. This will allow for the evaluation of the quantized model’s resilience against real-world electromagnetic interference (EMI), dynamic temperature fluctuations, and highly erratic driving profiles.
- **Exploration of Advanced Lightweight Architectures:** While LSTMs are effective for time-series forecasting, they suffer from sequential bottlenecks. Future work should explore the deployment of emerging lightweight sequence models, such as State Space Models (e.g., Mamba) or efficient Transformers with linear attention mechanisms. Investigating the compilation and bare-metal deployment behavior of these modern architectures via Apache TVM could yield even lower inference latencies without sacrificing the ability to capture long-term degradation dependencies.
- **On-Device Continual Learning:** The compiled TVM models in this study are statically allocated Ahead-of-Time (AOT), meaning their weights are frozen post-deployment. Given that battery aging mechanisms can shift unpredictably due to user habits or extreme environmental exposure over years of operation, future iterations of this framework should explore lightweight, on-device continual learning. Implementing partial weight updating or bias correction mechanisms directly on the MCU would allow the SoH estimator to dynamically adapt to unforeseen degradation trajectories during the vehicle’s lifespan.
- **Robust On-Device Feature Engineering:** While the neural network inference proved highly efficient and deterministic on the MCXN947, the end-to-end pipeline revealed a high sensitivity to raw sensor noise during live CAN execution. Future work must focus on optimizing the on-device feature extraction algorithms. Implementing dynamic moving-average filters, advanced outlier rejection directly within the CAN ISR, and fixed-point safe accumulation logic will be strictly required to bridge the numerical gap between isolated feature injection and live vehicular data streams.

Bibliography

- [1] H. Wang, S.-H. Lu, X. Wang, S. Xia, and H. Chew, “A review of the multiscale mechanics of silicon electrodes in high-capacity lithium-ion batteries,” *Journal of Physics D Applied Physics*, vol. 55, p. 063001, 10 2021.
- [2] L. Su, Y. Xu, and Z. Dong, “State-of-health estimation of lithium-ion batteries: A comprehensive literature review from cell to pack levels,” *Energy Conversion and Economics*, vol. 5, no. 4, pp. 224–242, 2024. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/enc2.12125>
- [3] F. Wang, Z. Zhai, B. Liu, S. Zheng, Z. Zhao, and X. Chen, “Open access dataset, code library and benchmarking deep learning approaches for state-of-health estimation of lithium-ion batteries,” *Journal of Energy Storage*, vol. 77, p. 109884, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352152X23032826>
- [4] H. Ma, X. Bao, A. Lopes, L. Chen, G. Liu, and M. Zhu, “State-of-charge estimation of lithium-ion battery based on convolutional neural network combined with unscented kalman filter,” *Batteries*, vol. 10, no. 6, 2024. [Online]. Available: <https://www.mdpi.com/2313-0105/10/6/198>
- [5] F. Wang, Z. Zhai, Z. Zhao, Y. Di, and X. Chen, “Physics-informed neural network for lithium-ion battery degradation stable modeling and prognosis,” *Nature Communications*, vol. 15, no. 1, p. 4332, 2024. [Online]. Available: <https://doi.org/10.1038/s41467-024-48779-z>
- [6] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Gutttag, “What is the state of neural network pruning?” 2020. [Online]. Available: <https://arxiv.org/abs/2003.03033>
- [7] A. Volle, “Knowledge distillation,” <https://www.britannica.com/technology/knowledge-distillation>, 2025, encyclopedia Britannica. Accessed: July 12, 2026.
- [8] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” 2019. [Online]. Available: <https://arxiv.org/abs/1808.05377>
- [9] X. Dai, H. Yin, and N. K. Jha, “Grow and prune compact, fast, and accurate lstms,” 2018. [Online]. Available: <https://arxiv.org/abs/1805.11797>
- [10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” 2017. [Online]. Available: <https://arxiv.org/abs/1712.05877>

- [11] A. K. M. A. Habib, M. K. Hasan, G. F. Issa, D. Singh, S. Islam, and T. M. Ghazal, "Lithium-ion battery management system for electric vehicles: Constraints, challenges, and recommendations," *Batteries*, vol. 9, no. 3, 2023. [Online]. Available: <https://www.mdpi.com/2313-0105/9/3/152>
- [12] L. Zhou, X. Lai, B. Li, Y. Yao, M. Yuan, J. Weng, and Y. Zheng, "State estimation models of lithium-ion batteries for battery management system: Status, challenges, and future trends," *Batteries*, vol. 9, no. 2, 2023. [Online]. Available: <https://www.mdpi.com/2313-0105/9/2/131>
- [13] M. Li, W. Xu, S. Zhang, L. Liu, A. Hussain, E. Hu, J. Zhang, Z. Mao, and Z. Chen, "State of health estimation and battery management: A review of health indicators, models and machine learning," *Materials*, vol. 18, no. 1, 2025. [Online]. Available: <https://www.mdpi.com/1996-1944/18/1/145>
- [14] K. Liu, J. Fang, S. Zhao, Y. Liu, H. Dai, L. Ye, and Q. Peng, "Battery state-of-health estimation: An ultrasonic detection method with explainable ai," *Energy*, vol. 319, p. 134923, 2025. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360544225005651>
- [15] H. Lee, M. Y. Yoo, J.-H. Choi, W. Sung, and J. S. Heo, "State-of-charge and state-of-health estimation for li-ion batteries of hybrid electric vehicles under deep degradation," *PHME Conference*, vol. 8, no. 1, p. 10, Jun. 2024.
- [16] B. D. Soyoye, I. Bhattacharya, M. V. Anthony Dhason, and T. Banik, "State of charge and state of health estimation in electric vehicles: Challenges, approaches and future directions," *Batteries*, vol. 11, no. 1, 2025. [Online]. Available: <https://www.mdpi.com/2313-0105/11/1/32>
- [17] W. Tang, X. Liu, X. Li, J. Shen, Z. Liao, and M. Gong, "State of health estimation for energy storage batteries based on multi-condition feature extraction," *Batteries*, vol. 12, no. 1, 2026. [Online]. Available: <https://www.mdpi.com/2313-0105/12/1/34>
- [18] Shruthi, G. S. Sheshadri, P. K. T., and K. N., "Analysis of interleaved dc-dc converters for electric vehicle battery charging using anfis control: A comprehensive review," *International Journal of Scientific Research and Technology*, vol. 3, no. 6, pp. 301–311, 2026. [Online]. Available: <https://doi.org/10.5281/zenodo.20541184>
- [19] H. Hu, C. Liang, X. Huang *et al.*, "Onet: Operator network for randomized and robust battery health estimation using operation condition and cycling data matching," *Journal of Power Sources*, vol. 672, p. 239592, 2026. [Online]. Available: <http://dx.doi.org/10.1016/j.jpowsour.2026.239592>
- [20] J. A. Cinco-Ahumada and J. A. Flores-Triana, "Deployment of machine learning algorithm to predict battery behavior," Trabajo de obtención de grado, Especialidad en Sistemas Embebidos, ITESO, Tlaquepaque, Jalisco, 2023.
- [21] F. Svoboda, J. Fernandez-Marques, E. Liberis, and N. Lane, "Deep learning on microcontrollers: a study on deployment costs and challenges," 04 2022, pp. 54–63.

-
- [22] M. H. Al-Meer, "A deep learning method for the health state prediction of lithium-ion batteries based on lut-memory and quantization," *World Electric Vehicle Journal*, vol. 15, no. 2, 2024. [Online]. Available: <https://www.mdpi.com/2032-6653/15/2/38>
- [23] X. Gu, M. Liu, and J. Tian, "State of health estimation for batteries based on a dynamic graph pruning neural network with a self-attention mechanism," *Energies*, vol. 18, no. 20, 2025. [Online]. Available: <https://www.mdpi.com/1996-1073/18/20/5333>
- [24] A. K. Koech, G. Mwandila, F. Mulolani, and P. Mwaanga, "Lithium-ion battery fundamentals and exploration of cathode materials: A review," *South African Journal of Chemical Engineering*, vol. 50, pp. 321–339, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1026918524001100>
- [25] L. Zhang, C. Wei, L. Gao, M.-F. Lin, A. L.-S. Eh, J. Chen, and S. Li, "Recent advances in electrospun nanostructured electrodes in zinc-ion batteries," *Batteries*, vol. 10, no. 1, 2024. [Online]. Available: <https://www.mdpi.com/2313-0105/10/1/22>
- [26] F. Ghani, K. An, and D. Lee, "A review on design parameters for the full-cell lithium-ion batteries," *Batteries*, vol. 10, no. 10, 2024. [Online]. Available: <https://www.mdpi.com/2313-0105/10/10/340>
- [27] J. S. Edge, S. O’Kane, R. Prosser, N. D. Kirkaldy, A. N. Patel, A. Hales, A. Ghosh, W. Ai, J. Chen, J. Yang, S. Li, M.-C. Pang, L. Bravo Diaz, A. Tomaszewska, M. W. Marzook, K. N. Radhakrishnan, H. Wang, Y. Patel, B. Wu, and G. J. Offer, "Lithium ion battery degradation: what you need to know," *Physical Chemistry Chemical Physics*, vol. 23, no. 14, pp. 8200–8221, 04 2021. [Online]. Available: <https://doi.org/10.1039/d1cp00359c>
- [28] J. S. Menye, M.-B. Camara, and B. Dakyo, "Lithium battery degradation and failure mechanisms: A state-of-the-art review," *Energies*, vol. 18, no. 2, 2025. [Online]. Available: <https://www.mdpi.com/1996-1073/18/2/342>
- [29] S. E. J. O’Kane, W. Ai, G. Madabattula, D. Alonso-Alvarez, R. Timms, V. Sulzer, J. S. Edge, B. Wu, G. J. Offer, and M. Marinescu, "Lithium-ion battery degradation: how to model it," *Physical Chemistry Chemical Physics*, vol. 24, no. 13, pp. 7909–7922, 04 2022. [Online]. Available: <https://doi.org/10.1039/d2cp00417h>
- [30] N. A. Arif, S. Mekhilef, M. Seyedmahmoudian, and A. Stojcevski, "Degradation in li-ion batteries and capacity decline under different cycling conditions: A comprehensive review," *Journal of Energy Storage*, vol. 154, p. 121270, 2026. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352152X26009345>
- [31] Z. Yang, T. R. Tanim, H. Liu, I. Bloom, E. J. Dufek, B. Key, and B. J. Ingram, "Quantitative analysis of origin of lithium inventory loss and interface evolution over extended fast charge aging in li ion batteries," *ACS Applied Materials & Interfaces*, vol. 15, no. 31, pp. 37 410–37 421, 2023. [Online]. Available: <https://doi.org/10.1021/acsami.3c06084>

- [32] J. Cao and Y. Zhang, “Analysis and investigation of diffusion-induced stress in lithium-ion particle through elastic-viscoplastic model of binder,” *Batteries*, vol. 11, no. 4, 2025. [Online]. Available: <https://www.mdpi.com/2313-0105/11/4/132>
- [33] A. Mukhopadhyay and B. W. Sheldon, “Deformation and stress in electrode materials for li-ion batteries,” *Progress in Materials Science*, vol. 63, pp. 58–116, 2014. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0079642514000139>
- [34] T. Rahman and T. Alharbi, “Exploring lithium-ion battery degradation: A concise review of critical factors, impacts, data-driven degradation estimation techniques, and sustainable directions for energy storage systems,” *Batteries*, vol. 10, no. 7, 2024. [Online]. Available: <https://www.mdpi.com/2313-0105/10/7/220>
- [35] T. Li, X.-Z. Yuan, L. Zhang, D. Song, K. Shi, and C. Bock, “Degradation mechanisms and mitigation strategies of nickel-rich nmc-based lithium-ion batteries,” *Electrochemical Energy Reviews*, vol. 3, no. 1, pp. 43–80, 2020. [Online]. Available: <https://doi.org/10.1007/s41918-019-00053-3>
- [36] J. S. Edge, S. O’Kane, R. Prosser, N. D. Kirkaldy, A. N. Patel, A. Hales, A. Ghosh, W. Ai, J. Chen, J. Yang, S. Li, M. C. Pang, L. Bravo Diaz, A. Tomaszewska, M. W. Marzook, K. N. Radhakrishnan, H. Wang, Y. Patel, B. Wu, and G. J. Offer, “Lithium ion battery degradation: What you need to know,” *Physical Chemistry Chemical Physics*, vol. 23, no. 14, pp. 8200–8221, 2021. [Online]. Available: <https://doi.org/10.1039/D1CP00359C>
- [37] M. Bercibar, I. Gandiaga, I. Villarreal, N. Omar, J. Van Mierlo, and P. Van den Bossche, “Critical review of state of health estimation methods of li-ion batteries for real applications,” *Renewable and Sustainable Energy Reviews*, vol. 56, pp. 572–587, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1364032115013076>
- [38] M. H. Lipu, M. Hannan, A. Hussain, M. Hoque, P. J. Ker, M. Saad, and A. Ayob, “A review of state of health and remaining useful life estimation methods for lithium-ion battery in electric vehicles: Challenges and recommendations,” *Journal of Cleaner Production*, vol. 205, pp. 115–133, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0959652618327793>
- [39] L. Su, Y. Xu, and Z. Dong, “State-of-health estimation of lithium-ion batteries: A comprehensive literature review from cell to pack levels,” *Energy Conversion and Economics*, vol. 5, no. 4, pp. 224–242, 2024. [Online]. Available: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/enc2.12125>
- [40] J. Zhao, Y. Zhu, B. Zhang, M. Liu, J. Wang, C. Liu, and X. Hao, “Review of state estimation and remaining useful life prediction methods for lithium-ion batteries,” *Sustainability*, vol. 15, no. 6, 2023. [Online]. Available: <https://www.mdpi.com/2071-1050/15/6/5014>
- [41] G. Zhu, C. Kong, J. V. Wang, W. Chen, Q. Wang, and J. Kang, “A simplified electrochemical model for lithium-ion batteries based on ensemble

- learning,” *iScience*, vol. 27, no. 5, p. 109685, 2024. [Online]. Available: <https://doi.org/10.1016/j.isci.2024.109685>
- [42] J. Zhou, B. Xing, and C. Wang, “A review of lithium ion batteries electrochemical models for electric vehicles,” *E3S Web of Conferences*, vol. 185, p. 04001, 01 2020.
- [43] F. Ahmed, K. Abulsaud, and A. Massoud, “On equivalent circuit model-based state-of-charge estimation for lithium-ion batteries in electric vehicles,” *IEEE Access*, vol. PP, pp. 1–1, 01 2025.
- [44] X. Sun, L. Zuo, M. Zhang, Y. Su, Q. Fu, and J. Jiang, “Equivalent circuit models for lithium-ion batteries: A comprehensive review,” *Electronics*, vol. 15, no. 9, 2026. [Online]. Available: <https://www.mdpi.com/2079-9292/15/9/1968>
- [45] K. Severson, P. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M. Chen, M. Aykol, P. Herring, D. Fraggadakis, M. Bazant, S. Harris, W. Chueh, and R. Braatz, “Data-driven prediction of battery cycle life before capacity degradation,” *Nature Energy*, vol. 4, pp. 383–391, 03 2019.
- [46] X. Hu, J. Jiang, D. Cao, and B. Egardt, “Battery health prognosis for electric vehicles using sample entropy and sparse bayesian predictive modeling,” *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 1–1, 01 2015.
- [47] E. Chemali, P. Kollmeyer, M. Preindl, and A. Emadi, “State-of-charge estimation of li-ion batteries using deep neural networks: A machine learning approach,” *Journal of Power Sources*, vol. 400, pp. 242–255, 10 2018.
- [48] N. Noura, L. Boulon, and S. Jemeï, “A review of battery state of health estimation methods: Hybrid electric vehicle challenges,” *World Electric Vehicle Journal*, vol. 11, no. 4, 2020. [Online]. Available: <https://www.mdpi.com/2032-6653/11/4/66>
- [49] Z. Ma, R. Yang, and Z. Wang, “A novel data-model fusion state-of-health estimation approach for lithium-ion batteries,” *Applied Energy*, vol. 237, pp. 836–847, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306261918318907>
- [50] R. Wang, C. Song, M. Gao, J. Zhao, and Z. Xu, “Model-data fusion domain adaptation for battery state of health estimation with fewer data and simplified feature extractor,” *Journal of Energy Storage*, vol. 60, p. 106686, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352152X2300083X>
- [51] X. Bian, Z. Wei, W. Li, J. Pou, D. U. Sauer, and L. Liu, “State-of-health estimation of lithium-ion batteries by fusing an open circuit voltage model and incremental capacity analysis,” *IEEE Transactions on Power Electronics*, vol. 37, no. 2, pp. 2226–2236, 2022.
- [52] T. Hofmann, J. Hamar, M. Rogge, C. Zoerr, S. Erhard, and J. Philipp Schmidt, “Physics-informed neural networks for state of health estimation in lithium-ion batteries,” *Journal of The Electrochemical Society*, vol. 170, no. 9, p. 090524, sep 2023. [Online]. Available: <https://doi.org/10.1149/1945-7111/acf0ef>

- [53] P. Wen, Z.-S. Ye, Y. Li, S. Chen, P. Xie, and S. Zhao, “Physics-informed neural networks for prognostics and health management of lithium-ion batteries,” 2023. [Online]. Available: <https://arxiv.org/abs/2301.00776>
- [54] *Physics-Informed Neural Networks for Degradation Diagnostics of Lithium-Ion Batteries*, ser. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. Volume 3A: 49th Design Automation Conference (DAC), 08 2023. [Online]. Available: <https://doi.org/10.1115/DETC2023-116940>
- [55] Z. Xu, Y. Guo, and J. Saleh, “A physics-informed dynamic deep autoencoder for accurate state-of-health prediction of lithium-ion battery,” *Neural Computing and Applications*, vol. 34, 05 2022.
- [56] Y. Ding, J. Zhu, Y. Liu, D. Ning, and M. Qin, “Integrated framework of lstm and physical-informed neural network for lithium-ion battery degradation modeling and prediction,” *AI*, vol. 6, no. 7, 2025. [Online]. Available: <https://www.mdpi.com/2673-2688/6/7/149>
- [57] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A survey of model compression and acceleration for deep neural networks,” *arXiv preprint arXiv:1710.09282*, 2020. [Online]. Available: <https://arxiv.org/abs/1710.09282>
- [58] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” 2016. [Online]. Available: <https://arxiv.org/abs/1510.00149>
- [59] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” 2015. [Online]. Available: <https://arxiv.org/abs/1503.02531>
- [60] H. Cai, L. Zhu, and S. Han, “Proxylessnas: Direct neural architecture search on target task and hardware,” 2019. [Online]. Available: <https://arxiv.org/abs/1812.00332>
- [61] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” 2015. [Online]. Available: <https://arxiv.org/abs/1506.02626>
- [62] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” 2017. [Online]. Available: <https://arxiv.org/abs/1608.08710>
- [63] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” 2016. [Online]. Available: <https://arxiv.org/abs/1608.03665>
- [64] Q. Huang, K. Zhou, S. You, and U. Neumann, “Learning to prune filters in convolutional neural networks,” 2018. [Online]. Available: <https://arxiv.org/abs/1801.07365>
- [65] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “Mcnunet: Tiny deep learning on iot devices,” 2020. [Online]. Available: <https://arxiv.org/abs/2007.10319>

-
- [66] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.
- [67] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2016. [Online]. Available: <https://arxiv.org/abs/1511.06530>
- [68] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014. [Online]. Available: <https://arxiv.org/abs/1405.3866>
- [69] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," 2017. [Online]. Available: <https://arxiv.org/abs/1711.02638>
- [70] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: <https://arxiv.org/abs/1503.02531>
- [71] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," 2015. [Online]. Available: <https://arxiv.org/abs/1412.6550>
- [72] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, *AMC: AutoML for Model Compression and Acceleration on Mobile Devices*. Springer International Publishing, 2018, p. 815–832. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01234-2_48
- [73] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," 2017. [Online]. Available: <https://arxiv.org/abs/1705.08665>
- [74] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "Tensorflow lite micro: Embedded machine learning on tinymml systems," 2021. [Online]. Available: <https://arxiv.org/abs/2010.08678>
- [75] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for ai-enabled iot devices: A review," *Sensors*, vol. 20, no. 9, 2020. [Online]. Available: <https://www.mdpi.com/1424-8220/20/9/2533>
- [76] K. A. Severson, P. M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M. H. Chen, M. Aykol, P. K. Herring, D. Fraggedakis, M. Z. Bazant, S. J. Harris, W. C. Chueh, and R. D. Braatz, "Data-driven prediction of battery cycle life before capacity degradation," *Nature Energy*, vol. 4, no. 5, pp. 383–391, May 2019. [Online]. Available: <https://doi.org/10.1038/s41560-019-0356-8>
- [77] D. Roman, S. Saxena, V. Robu, M. Pecht, and D. Flynn, "Machine learning pipeline for battery state-of-health estimation," *Nature Machine Intelligence*, vol. 3, no. 5, pp. 447–456, May 2021. [Online]. Available: <https://doi.org/10.1038/s42256-021-00312-3>
- [78] V. Borja-Jaimes, J. S. Valdez-Martínez, M. Beltrán-Escobar, A. Cruz-Rojas, A. Gil-Velasco, and A. Coronel-Escamilla, "A review of fractional order calculus applications in electric vehicle energy storage and management

- systems,” *Mathematics*, vol. 13, no. 18, 2025. [Online]. Available: <https://www.mdpi.com/2227-7390/13/18/2920>
- [79] J. W. Tukey, “Nonlinear (nonsuperposable) methods for smoothing data,” 1974. [Online]. Available: <https://api.semanticscholar.org/CorpusID:118989976>
- [80] A. Savitzky and M. J. E. Golay, “Smoothing and differentiation of data by simplified least squares procedures,” *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964. [Online]. Available: <https://doi.org/10.1021/ac60214a047>
- [81] X. Feng, C. Weng, X. He, X. Han, L. Lu, D. Ren, and M. Ouyang, “Online state-of-health estimation for li-ion battery using partial charging segment based on support vector machine,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8583–8592, 2019.
- [82] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *J. Mach. Learn. Res.*, vol. 3, no. null, p. 1157–1182, Mar. 2003.
- [83] G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Computers Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014, 40th-year commemorative issue. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0045790613003066>
- [84] D. Reshef, Y. Reshef, H. Finucane, S. Grossman, G. McVean, P. Turnbaugh, E. Lander, M. Mitzenmacher, and P. Sabeti, “Detecting novel associations in large data sets,” *Science*, vol. 334, pp. 1518–1524, 12 2011.
- [85] Y. Choi, P. Jang, and J. Yun, “Multi-teacher knowledge distillation framework for lightweight deep learning-based state-of-health estimation,” *International Journal of Energy Research*, vol. 2026, no. 1, p. 5535455, 2026. [Online]. Available: <https://doi.org/10.1155/er/5535455>
- [86] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: <https://doi.org/10.1023/A:1010933404324>
- [87] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, “Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5695–5705, 2018.
- [88] M. Jankauskas, A. Katkevičius, and A. Serackis, “Investigation of exponent-free lstm cells for virtual sensing applications,” *Electronics*, vol. 15, no. 3, 2026. [Online]. Available: <https://www.mdpi.com/2079-9292/15/3/576>
- [89] C. Liu, H. Li, K. Li, Y. Wu, and B. Lv, “Deep learning for state of health estimation of lithium-ion batteries in electric vehicles: A systematic review,” *Energies*, vol. 18, no. 6, 2025. [Online]. Available: <https://www.mdpi.com/1996-1073/18/6/1463>
- [90] S. Arlot and A. Celisse, “A survey of cross-validation procedures for model selection,” *Statistics Surveys*, vol. 4, no. none, Jan. 2010. [Online]. Available: <http://dx.doi.org/10.1214/09-SS054>

-
- [91] K. Sechidis, G. Tsoumakas, and I. Vlahavas, *On the Stratification of Multi-label Data*, 09 2011, vol. 3, pp. 145–158.
 - [92] J. Heaton, “Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618,” *Genetic Programming and Evolvable Machines*, vol. 19, 10 2017.
 - [93] D. Masters and C. Luschi, “Revisiting small batch training for deep neural networks,” 2018. [Online]. Available: <https://arxiv.org/abs/1804.07612>
 - [94] J. Bergstra, “Random search for hyper-parameter optimization,” *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, 03 2012.
 - [95] X. Ying, “An overview of overfitting and its solutions,” *Journal of Physics: Conference Series*, vol. 1168, p. 022022, 02 2019.
 - [96] L. Prechelt, *Early Stopping - But When?*, 03 2002, pp. 55–69.
 - [97] T. Hastie, R. Tibshirani, and J. Friedman, “The elements of statistical learning: Data mining, inference, and prediction, second edition (springer series in statistics),” 02 2009.
 - [98] A. Singh, C. Feltner, J. Peck, and K. I. Kuhn, “Data driven prediction of battery cycle life before capacity degradation,” 2021. [Online]. Available: <https://arxiv.org/abs/2110.09687>
 - [99] A. Botchkarev, “A new typology design of performance metrics to measure errors in machine learning regression algorithms,” *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, pp. 045–076, 01 2019.
 - [100] M. El-Dalahmeh, J. Lillystone, M. Al-Greer, and M. El-Dalahmeh, “State of health estimation of lithium-ion batteries based on data-driven techniques,” in *2021 56th International Universities Power Engineering Conference (UPEC)*, 2021, pp. 1–6.
 - [101] X. Hu, L. Xu, X. Lin, and M. Pecht, “Battery lifetime prognostics,” *Joule*, vol. 4, no. 2, pp. 310–346, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542435119305859>
 - [102] R. Xiong, L. Li, and J. Tian, “Towards a smarter battery management system: A critical review on battery state of health monitoring methods,” *Journal of Power Sources*, vol. 405, pp. 18–29, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S037877531831111X>
 - [103] C. Willmott and K. Matsuura, “Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance,” *Climate Research*, vol. 30, p. 79, 12 2005.
 - [104] D. Chicco, M. Warrens, and G. Jurman, “The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation,” *PeerJ Computer Science*, vol. 7, p. e623, 07 2021.
 - [105] P. Warden and D. Situnayake, “Tinymml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers,” 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:226814321>

- [106] U. Bajpai, S. Sharma, H. Kumar, and R. Gouda, “Edge-ai battery prognostics system for ev batteries using tinymml implementation,” 01 2026.
- [107] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally, “Dsd: Dense-sparse-dense training for deep neural networks,” 2017. [Online]. Available: <https://arxiv.org/abs/1607.04381>
- [108] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, “Group sparse regularization for deep neural networks,” *Neurocomputing*, vol. 241, p. 81–89, 2017. [Online]. Available: <http://dx.doi.org/10.1016/j.neucom.2017.02.029>
- [109] E. J. Candes and B. Recht, “Exact matrix completion via convex optimization,” 2008. [Online]. Available: <https://arxiv.org/abs/0805.4471>
- [110] R. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” 2014. [Online]. Available: <https://arxiv.org/abs/1404.0736>
- [111] W. Park, D. Kim, Y. Lu, and M. Cho, “Relational knowledge distillation,” 2019. [Online]. Available: <https://arxiv.org/abs/1904.05068>
- [112] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” 2017. [Online]. Available: <https://arxiv.org/abs/1608.03983>
- [113] M. Raissi, P. Perdikaris, and G. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0021999118307125>
- [114] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, Jun. 2021. [Online]. Available: <https://doi.org/10.1038/s42254-021-00314-5>
- [115] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” vol. 129, no. 6. Springer Science and Business Media LLC, Mar. 2021, p. 1789–1819. [Online]. Available: <http://dx.doi.org/10.1007/s11263-021-01453-z>
- [116] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, “Integer quantization for deep learning inference: Principles and empirical evaluation,” 2020. [Online]. Available: <https://arxiv.org/abs/2004.09602>
- [117] M. Verhelst, L. Benini, and N. Verma, “How to keep pushing ml accelerator performance? know your rooflines!” *IEEE Journal of Solid-State Circuits*, vol. 60, no. 6, p. 1888–1905, 2025. [Online]. Available: <http://dx.doi.org/10.1109/JSSC.2025.3553765>
- [118] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” 2017. [Online]. Available: <https://arxiv.org/abs/1703.09039>

-
- [119] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for multicore architectures,” *Commun. ACM*, vol. 52, pp. 65–76, 04 2009.
- [120] P. Marwedel, “Embedded system design: Embedded systems foundations of cyber-physical systems, and the internet of things,” 01 2021.
- [121] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, “Tvm: An automated end-to-end optimizing compiler for deep learning,” 2018. [Online]. Available: <https://arxiv.org/abs/1802.04799>
- [122] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, “TVM: An automated End-to-End optimizing compiler for deep learning,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/chen>
- [123] C. Liu, M. Jobst, L. Guo, X. Shi, J. Partzsch, and C. Mayr, “Deploying machine learning models to ahead-of-time runtime on edge using microtvm,” 06 2024, pp. 37–40.
- [124] S. Saha, S. S. Sandha, and M. Srivastava, “Machine learning for microcontroller-class hardware: A review,” *IEEE Sensors Journal*, vol. 22, no. 22, pp. 21 362–21 390, 2022, epub 2022 Oct 5.
- [125] Y. Bondarenko, M. Nagel, and T. Blankevoort, “Understanding and overcoming the challenges of efficient transformer quantization,” 2021. [Online]. Available: <https://arxiv.org/abs/2109.12948>
- [126] R. Lai, J. Shao, S. Feng, S. Lyubomirsky, B. Hou, W. Lin, Z. Ye, H. Jin, Y. Jin, J. Liu, L. Jin, Y. Cai, Z. Jiang, Y. Wu, S. Park, P. Srivastava, J. Roesch, T. C. Mowry, and T. Chen, “Relax: Composable abstractions for end-to-end dynamic machine learning,” in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS ’25. ACM, Mar. 2025, pp. 998–1013. [Online]. Available: <https://doi.org/10.1145/3676641.3716249>
- [127] J. Millar, Y. Huang, S. Sethi, H. Haddadi, and A. Madhavapeddy, “Benchmarking ultra-low-power μ npus,” 2025. [Online]. Available: <https://arxiv.org/abs/2503.22567>
- [128] G. L. Peterson, “Concurrent reading while writing,” *ACM Transactions on Programming Languages and Systems*, vol. 5, no. 1, pp. 46–55, Jan. 1983. [Online]. Available: <https://doi.org/10.1145/357195.357198>
- [129] D. Kafetzis, “Synapticos: An inference-first runtime architecture for neural processing units on resource-constrained microcontrollers,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.12606>
- [130] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, “Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks,” 2018. [Online]. Available: <https://arxiv.org/abs/1711.02257>

- [131] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” 2020. [Online]. Available: <https://arxiv.org/abs/2001.06782>
- [132] H. H. Schütt, “Bayesian comparisons between representations,” 2025. [Online]. Available: <https://arxiv.org/abs/2411.08739>
- [133] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. 11, 2008.