

Article

Lightweight 1D-CNN-Based Battery State-of-Charge Estimation and Hardware Development

Seungbum Kang, Yoonjae Lee, Gahyeon Jang and Seongsoo Lee * 

Department of Intelligent Semiconductors, Soongsil University, Seoul 06978, Republic of Korea; ksb4526@naver.com (S.K.); erici327@soongsil.ac.kr (Y.L.); 0828jjrkus@soongsil.ac.kr (G.J.)

* Correspondence: sslee@ssu.ac.kr

Abstract

This paper presents the FPGA implementation and verification of a lightweight one-dimensional convolutional neural network (1D-CNN) pipeline for real-time battery state-of-charge (SoC) estimation in automotive battery management systems. The proposed model employs separable 1D convolution and global average pooling, and applies aggressive structured pruning to reduce the number of parameters from 3121 to 358, representing an 88.5% reduction, without significant accuracy loss. Using quantization-aware training (QAT), the network is trained and executed in INT8, which reduces weight storage to one-quarter of the 32-bit baseline while maintaining high estimation accuracy with a Mean Absolute Error (MAE) of 0.0172. The hardware adopts a time-multiplexed single MAC architecture with FSM control, occupying 98,410 gates under a 28 nm process. Evaluations on an FPGA testbed with representative drive-cycle inputs show that the proposed INT8 pipeline achieves performance comparable to the floating-point reference with negligible precision drop, demonstrating its suitability for in-vehicle BMS deployment.

Keywords: BMS; lightweight; convolution neural network; inference accelerator

1. Introduction

With the increasing adoption of electric vehicles (EVs), the importance of Battery Management Systems (BMS) continues to grow [1]. A BMS performs key functions—including charge/discharge control, thermal management, fault diagnosis, and state-of-charge (SoC) estimation—to ensure the safety and performance of a battery pack [2]. Among these, SoC is a critical indicator used for driving-range prediction, power limitation, charging strategy, and degradation management, thereby governing the overall reliability of vehicle operation. However, in real-world driving environments, the patterns and distributions of measured signals vary due to abrupt changes in current profiles, temperature variations, sensor noise, and battery aging [3]. Consequently, maintaining consistent SoC estimation performance under diverse conditions remains a persistent challenge.

Conventional SoC estimation methods, such as Coulomb counting and compensation techniques based on the open-circuit voltage (OCV)–SoC relationship, have been widely utilized [4,5]. However, these approaches rely on the assumption that OCV can be measured under rest conditions and require compensation for shifts in the OCV–SoC curve caused by temperature changes and state-of-health (SoH) degradation. In addition, when Coulomb counting errors accumulate due to sensor noise or current-offset drift, the estimation error can increase under operating conditions where OCV-based correction cannot be sufficiently applied. To overcome these limitations, deep-learning-based approaches that directly learn



Academic Editor: Salvador Alepuz

Received: 31 December 2025

Revised: 21 January 2026

Accepted: 5 February 2026

Published: 6 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

SoC from time-series sensor signals—such as voltage, current, and temperature—have recently attracted significant attention. In particular, one-dimensional convolutional neural networks (1D-CNNs) can effectively capture local patterns in time-series data while maintaining a regular computational structure, making them favorable for embedded inference and hardware accelerator implementation compared to recurrent neural networks [6].

Hybrid approaches that combine mechanism-based battery models (e.g., equivalent-circuit models and filtering) with data-driven learning can improve estimation accuracy under certain operating conditions. However, such hybrid pipelines often require additional state/parameter tracking, matrix-update operations, and calibration procedures that increase implementation complexity and hardware resource demand in embedded BMS deployments. Accordingly, this work focuses on a compact 1D-CNN that enables a deterministic INT8/INT32 integer-only inference path and a hardware-friendly accelerator architecture, while leaving hybrid integration as a future extension.

Battery state estimation (e.g., SoC) research spans a broad spectrum of model-based, data-driven, and hybrid approaches. Hybrid pipelines that combine mechanism-based battery models (e.g., electro-thermal or equivalent-circuit models) with estimation algorithms can improve robustness under certain operating conditions; however, they often require state/parameter tracking, iterative matrix-update operations, and calibration procedures, which tend to increase implementation complexity and hardware resource demand in embedded deployments [7]. Machine-learning-assisted structures coupled with Kalman-filter-based tracking can also enhance estimation performance, but the additional computations and state management during filter updates may impose non-trivial design overhead from a hardware implementation perspective [8]. In contrast, ANN-based approaches with lightweight design strategies offer structural simplicity, yet effectively capturing local patterns from short-window time-series inputs typically depends on careful architectural choices, and the accuracy-to-compute efficiency trade-off can vary across designs [9]. Traditional machine-learning methods (e.g., random-forest-based regression) can be attractive due to their relatively simple arithmetic operations; however, achieving high accuracy may require a large ensemble and frequent branching, which can increase control complexity and reduce latency predictability in hardware [10]. Moreover, hybrid data-driven frameworks that incorporate explicit feature-extraction stages may improve accuracy, but additional preprocessing/feature-extraction blocks can increase overall pipeline complexity and memory requirements [11]. Considering these trade-offs, we adopt a compact 1D-CNN because it provides a regular, MAC-dominated computation pattern and predictable dataflow, and it is well-suited to deterministic INT8/INT32 integer-only quantization and verification. This choice enables deterministic latency management under tight ECU constraints and facilitates an area-efficient accelerator implementation via time-multiplexed reuse of a single MAC unit.

Despite these advantages, deploying deep-learning-based SoC estimation in an ECU-level embedded environment requires addressing several practical constraints. First, real-time inference must be feasible under limited computational resources and power budgets. Second, memory usage must be minimized because storage for weights and intermediate activations is constrained. Third, automotive systems require predictable latency and reproducible computation paths for functional safety and reliability, and floating-point arithmetic can impose significant costs in terms of area, power, and verification effort [12]. Therefore, it is necessary to present both a lightweight, integer-based SoC estimation model that preserves accuracy and an architecture that can be reliably implemented with small hardware resources.

In this paper, we propose a lightweight 1D-CNN pipeline for real-time SoC estimation and its hardware implementation targeting automotive BMS applications. The proposed

model reduces the number of parameters and computational complexity by leveraging separable convolution and global average pooling (GAP), and further shrinks redundant operations and parameters through structured pruning. In addition, quantization-aware training (QAT) is applied during training so that inference can be executed entirely through an INT8/INT32 integer computation path. A scale- and shift-based integer requantization rule is adopted to achieve bit-exact reproducibility with a software reference.

From the hardware perspective, the proposed 1D-CNN is implemented in Verilog HDL and adopts an FSM-based control structure that minimizes area by reusing a single MAC unit in a time-multiplexed manner. Specifically, the pipeline consists of (i) a MAC core that performs INT8×INT8 multiplication and INT32 accumulation, (ii) a post-processing unit that executes integer requantization and the activation function (ReLU), and (iii) buffering and address-generation logic to sustain computation under limited on-chip memory. This design provides an integer computation path suitable for embedded environments without requiring an FPU, while ensuring structural regularity and ease of verification. Furthermore, RTL-simulation-based functional verification confirms agreement between the quantized software inference results and the hardware outputs, demonstrating the feasibility of practical deployment.

2. Background

2.1. Problem Definition

In this paper, state-of-charge (SoC) estimation is formulated as a regression problem that predicts SoC from time-series sensor signals [13]. The input is windowed time-series data composed of sensor signals acquired by the BMS, such as voltage, current, and temperature, and is denoted as $X \in \mathbb{R}^{T \times C}$, where T is the window length and C is the number of input channels [14]. The model can be represented as a function $f_{\theta}(\cdot)$ parameterized by θ , and the prediction for input X is denoted by $\hat{y}$.

Model performance is evaluated based on the error between the ground-truth value y and the predicted value $\hat{y}$. In this work, mean absolute error (MAE) and mean squared error (MSE) are used as evaluation metrics. Let N be the number of samples in the test set; the metrics are defined as follows:

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

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

where y_i and $\hat{y}_i$ denote the true SoC and the predicted SoC of the i -th sample, respectively.

For deployment on an automotive ECU, inference must be performed in real time under constrained computational resources and power budgets. Weights and intermediate activations must fit within limited memory capacity, and, at the system level, reproducible computation paths and predictable latency are required. Therefore, it is important to design a model with a regular computation structure that can be implemented using integer arithmetic.

2.2. Conventional SoC Estimation

Conventional SoC estimation methods widely used in practice include Coulomb counting and open-circuit-voltage (OCV)-based estimation. Coulomb counting updates SoC by integrating the current over time. Let the nominal capacity be Q_n ; the SoC can be expressed as

$$\text{SoC}(t) = \text{SoC}(t_0) - \frac{1}{Q_n} \int_{t_0}^t I(\tau) d\tau \quad (3)$$

In practical BMS implementations, this is realized in a discrete-time form based on sampling. With a sampling interval of Δt , the update equation becomes

$$\text{SoC}[k] = \text{SoC}[k-1] - \frac{\Delta t}{Q_n} I[k] \quad (4)$$

Coulomb counting has the advantages of low computational complexity and simple implementation; however, if an initial SoC error exists, its effect can persist over time. In addition, long-term operation may suffer from drift as current-sensor offset or integration errors accumulate.

OCV-based methods estimate SoC using the static relationship between open-circuit voltage and SoC (the OCV–SoC curve) when the battery reaches a sufficiently rested state. Typically, an OCV–SoC lookup table or interpolation model is constructed for different temperature conditions, and SoC is inversely inferred from the measured voltage V_{oc} :

$$\text{SoC} = g(V_{oc}, T) \quad (5)$$

OCV-based estimation is useful as a correction mechanism for Coulomb counting because it does not accumulate integration drift. However, in real driving scenarios, the battery is often under load, making it difficult to obtain the true open-circuit voltage directly. If the rest period is insufficient, polarization and relaxation dynamics can prevent the voltage from accurately reflecting SoC. Moreover, the OCV–SoC relationship can vary with temperature and degradation (SoH), and the presence of hysteresis can further introduce errors when mapping SoC using a single curve [15,16].

Therefore, in traditional BMS designs, Coulomb counting is commonly used as the primary estimator, while OCV-based estimation is applied under specific conditions (e.g., rest intervals or low-load regions) to correct the initial value or compensate accumulated errors.

2.3. Data-Driven SoC Estimation and Rationale for 1D-CNN

Recently, data-driven SoC estimation that directly learns time-series sensor signals such as voltage, current, and temperature has been actively studied. This approach can reduce dependence on explicit model-parameter identification and condition-dependent compensation. It is also beneficial in that nonlinear characteristics and patterns under varying operating conditions can be learned directly from data. Nevertheless, from the perspective of deployment on an automotive ECU, not only accuracy but also computational cost, memory requirements, and implementation complexity must be considered simultaneously.

Various deep-learning architectures can be employed for time-series estimation; however, from a hardware-implementation standpoint, the regularity of the computation structure and the simplicity of the data flow are crucial. A 1D-CNN features a convolution-centric and regular computation pattern and is well suited to window-based input processing [17]. Therefore, it is often considered a favorable model family for embedded inference and accelerator implementation, as it can reduce implementation burden compared to alternatives with more complex control flow. In addition, when designing to meet a target performance level, 1D-CNNs can often be configured with a relatively small number of parameters, which naturally aligns with on-chip-memory-centric implementations and time-multiplexed reuse of computational units [18].

2.4. Lightweight Network Building Blocks for Embedded Inference

In embedded environments such as automotive BMSs, SoC inference must be performed reliably under constrained computational resources, memory bandwidth, and power budgets. In particular, neural-network-based SoC estimation repeatedly applies convolution and nonlinear operations to time-series inputs; thus, as the number of pa-

rameters and operations increases, runtime latency and memory-access cost can grow rapidly. Moreover, embedded implementations typically require regular data flow and predictable latency. Accordingly, rather than relying on irregular sparsity, it is effective to apply lightweighting techniques that are structurally simple and can be mapped to an integer-only computation path. This subsection summarizes the lightweight building blocks adopted to satisfy these requirements and describes separable convolution, global average pooling, structured pruning, and PTQ/QAT-based integer inference.

2.4.1. Separable Convolution

For lightweight network design, it is important to reduce the number of parameters and computational complexity of convolution operations while maintaining representational capacity. Here, we define the parameter count of a standard 1D convolution as P_{conv} . Let C_{in} and C_{out} denote the numbers of input and output channels, respectively, and let K denote the kernel length. The number of parameters in a standard convolution, P_{conv} , is given by

$$P_{\text{conv}} = K \cdot C_{\text{in}} \cdot C_{\text{out}} \quad (6)$$

Separable convolution factorizes a standard convolution into depthwise convolution and pointwise convolution to reduce the parameter count and computational cost [19]. We define the total parameter count of separable convolution as P_{sep} . In the depthwise stage, convolutions are applied independently per channel, resulting in $K \cdot C_{\text{in}}$ parameters. In the pointwise stage, channel mixing is performed using 1×1 kernels, yielding $C_{\text{in}} \cdot C_{\text{out}}$ parameters. Therefore, the total parameter count of the separable structure, P_{sep} , is

$$P_{\text{sep}} = K \cdot C_{\text{in}} + C_{\text{in}} \cdot C_{\text{out}} \quad (7)$$

The reduction in parameters compared to standard convolution becomes more pronounced when C_{out} is large or when K increases. Because this difference directly translates into the required computation and memory-access volume in hardware implementations, it is an important design criterion for lightweight architectures. Figure 1 illustrates the structural difference between standard convolution and depthwise–pointwise separable convolution.

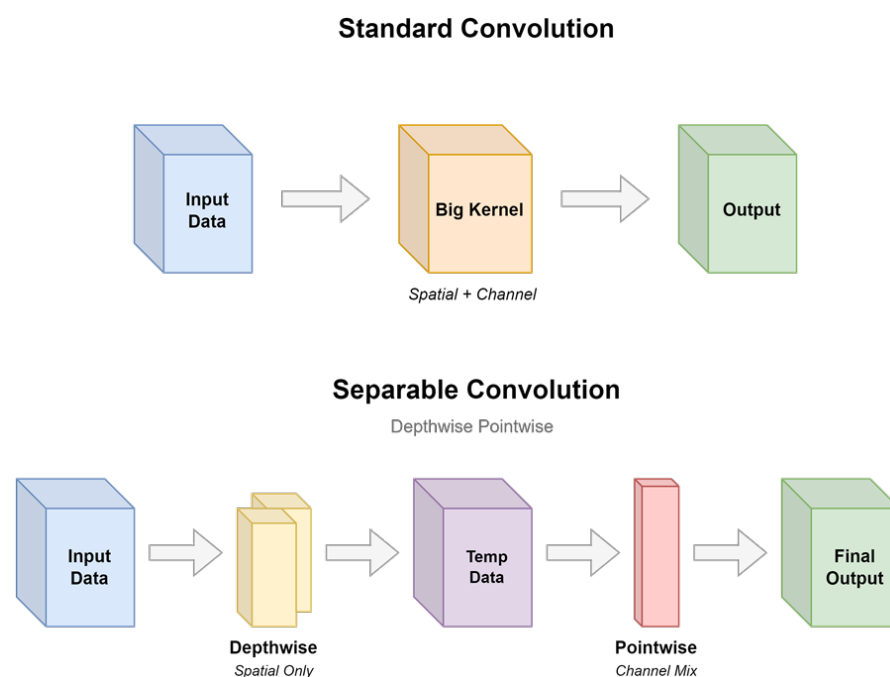


Figure 1. Comparison of standard convolution and depthwise–pointwise separable convolution.

2.4.2. Global Average Pooling

Global average pooling (GAP) compresses the temporal dimension of a feature map by averaging, producing a representative value for each channel. For a feature $x_m[t]$ of length T , the GAP output g_m is defined as

$$g_m = \frac{1}{T} \sum_{t=1}^T x_m[t] \quad (8)$$

where m is the channel index, $x_m[t]$ denotes the feature value of the m -th channel at time step t , and g_m represents the temporal mean for the m -th channel, i.e., a scalar summary of the temporal information. Separately, structured pruning is hardware-friendly and enables realistic acceleration [20]. GAP is advantageous for avoiding head designs in which the number of parameters grows in proportion to the temporal length. Since it has a simple data flow dominated by averaging operations, GAP also helps reduce implementation complexity from the perspectives of embedded deployment and hardware mapping.

2.4.3. Structured Pruning

Pruning reduces model size and computational cost by removing parameters deemed less important. Structured pruning removes parameters at the level of structured units such as channels or filters, so the corresponding computations associated with the removed structures are eliminated as well [21]. As a result, parameter reduction tends to translate directly into actual reductions in computation. In contrast, unstructured sparsity can lead to irregular distributions of nonzero values, which may require additional indexing or irregular memory accesses. Therefore, in resource-constrained environments that favor regular data flow, structured pruning can be a more suitable choice.

2.4.4. PTQ and QAT for Integer-Only Inference

Quantization reduces storage and computational cost by converting weights and activations of a trained network into low-precision integer representations. Two representative approaches are post-training quantization (PTQ) and quantization-aware training (QAT). PTQ applies quantization after training a float32 model, offering simplicity and negligible additional training cost. In contrast, QAT incorporates quantization effects during training so that parameters can adapt to quantization error, and it is generally known to better mitigate accuracy degradation than PTQ under the same integer inference conditions [22,23].

In embedded environments, model parameters and intermediate activations must fit within limited memory capacity. In this context, float32 representations can be burdensome in terms of storage and memory bandwidth. Accordingly, integer-only inference commonly represents both weights and activations in INT8, while multiplication results are accumulated in INT32. The accumulated outputs are then requantized—via scaling and rounding—to match the representable range of the next layer, and finally saturated into the INT8 range. Such an integer-only path simplifies computation and storage and is advantageous for resource-efficient embedded implementations.

3. Algorithm Design and Model Optimization

3.1. Data Preparation and Input Construction

Following the experimental framework discussed in previous sections, this study constructed training and evaluation datasets using the LG HG2 18650-series lithium-ion battery data released by McMaster University [24]. The raw dataset includes voltage, current, and temperature signals, which are typical inputs for BMS-based estimation. To prepare these for the 1D-CNN, the time-series signals were segmented into fixed-length

windows, consistent with the input representation $\mathbf{X} \in \mathbb{R}^{T \times C}$ defined in Section 2.1. This window-based approach ensures a constant input size, facilitating deterministic data flow for hardware mapping and real-time inference.

To mitigate the impact of sensor noise and abrupt fluctuations mentioned in the Introduction, we evaluated various smoothing preprocessing techniques to stabilize the inputs. Specifically, we compared a Moving Average (MA) filter, an Exponential Moving Average (EMA) filter, and a Savitzky–Golay (SG) filter [25–27]. While MA and EMA filters are simple to implement, they often introduce response latency or over-smooth critical local variations. In contrast, the SG filter preserves local waveform characteristics by fitting low-order polynomials within the window. As summarized in Table 1, the SG filter demonstrated the lowest error metrics. Consequently, the SG filter was adopted as the standard preprocessing step to ensure that the effects of subsequent architectural optimizations and quantization could be evaluated consistently.

In addition, the training and evaluation datasets were constructed by mixing multiple standard driving cycles, including LA92, US06, UDDS, and HWFET [24]. By incorporating these heterogeneous driving profiles with distinct dynamic characteristics, the dataset covers a wide range of current and voltage excitation patterns. This design choice aims to reduce potential bias toward a specific driving condition and to mitigate cycle-specific overfitting, thereby improving the robustness of the learned representation under diverse operational scenarios.

Table 1. Comparison of smoothing preprocessing methods.

	SG Filter	MA Filter	EMA Filter
MAE	0.0147	0.0181	0.0174
MSE	0.00048	0.00058	0.00052

3.2. Baseline Model Architecture

This subsection details the baseline model architecture prior to the application of lightweighting and quantization, as illustrated in Figure 2. To ensure a fair and consistent evaluation of each optimization stage, all models discussed in this work—from the baseline to the final optimized version—were trained under identical experimental conditions.

As summarized in Table 2, the models were trained using the Adam optimizer with a constant learning rate of 1×10^{-3} and a batch size of 128. The training process was conducted for 30 epochs, which was found to be sufficient for the loss to reach stable convergence. This unified training setup ensures that the performance improvements observed in subsequent sections are solely due to architectural and quantization optimizations, thereby addressing the hyperparameter reporting requirements suggested during peer review.

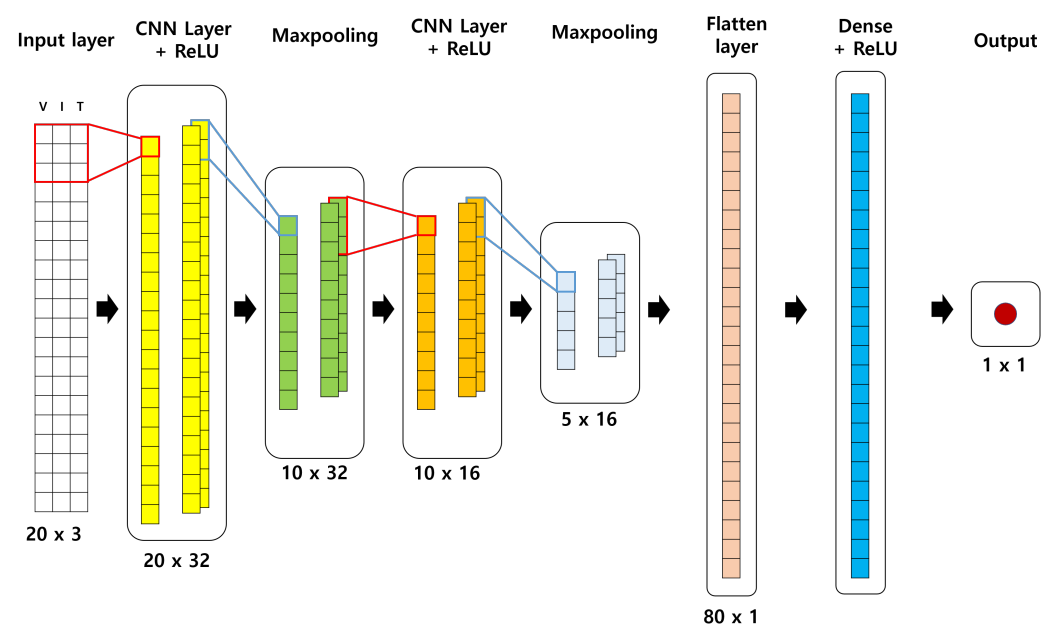
As established in Section 2.3, the model utilizes a 1D-CNN backbone to extract features from temporal variations in the battery signals. The baseline feature extractor consists of multiple 1D-convolutional layers followed by nonlinear activations. In this initial configuration, the resulting feature maps are converted into a 1D vector through a Flatten operation and passed to fully connected (FC) layers for SoC regression. While intuitive, this Flatten-based head creates a bottleneck for embedded deployment; the number of parameters in the FC layers grows proportionally with the temporal length T , increasing the storage and memory bandwidth requirements. As summarized in Table 3, this baseline model serves as the float32 reference point for evaluating the resource efficiency of the proposed optimizations.

Table 2. Summary of training hyperparameters for the proposed models.

Configuration Item	Value
Optimizer	Adam
Initial Learning Rate	1×10^{-3}
Batch Size	128
Total Epochs	30

Table 3. Baseline model (initial) summary (float32).

Metric	Value
Total parameters	3121
MAE	0.0147
MSE	0.00048

**Figure 2.** Baseline model architecture.

3.3. Head Structure Optimization: Transitioning from Flatten to GAP

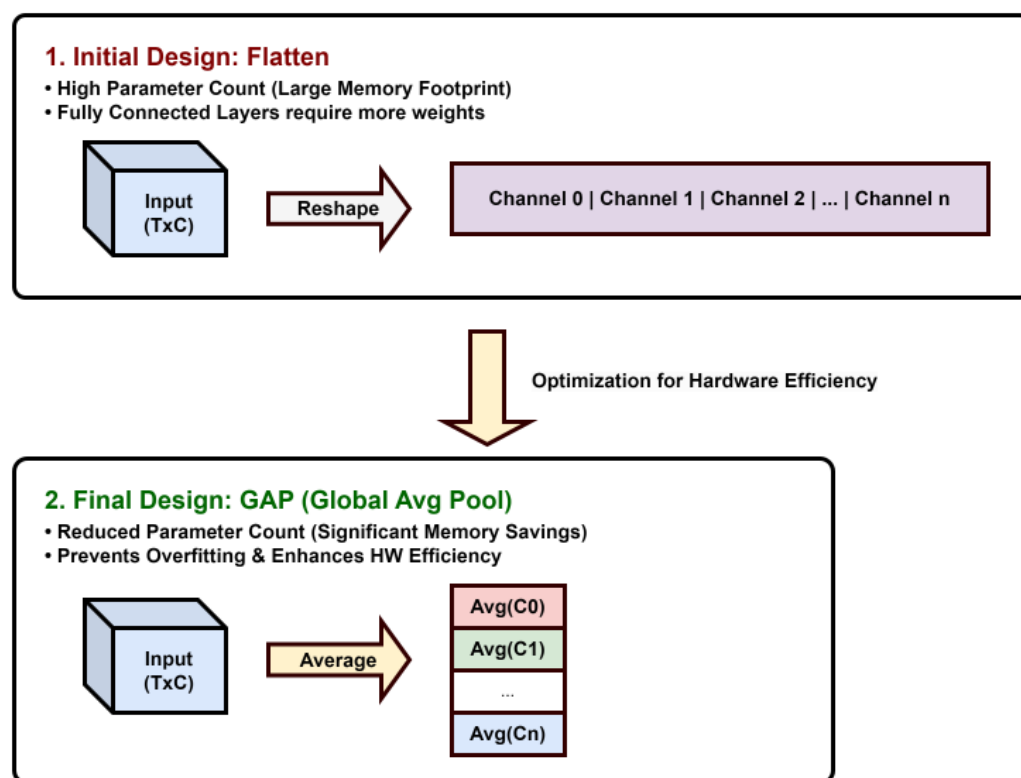
To address the parameter inflation caused by the Flatten operation, we optimized the head structure by adopting Global Average Pooling (GAP), as introduced in Section 2.4.2. In the proposed architecture, the GAP layer averages the temporal dimension of the feature maps, reducing each channel to a single representative scalar. This summary vector is then fed into a compact FC layer for SoC estimation.

This structural refinement effectively decouples the head's parameter count from the window length T , allowing the model to focus on channel-wise features. As shown in Figure 3, this transition significantly reduces the redundant connections inherent in Flatten-based designs. Table 4 provides a quantitative comparison: the GAP-based head reduced the total parameter count from 3121 to 2289 (a 26.6% reduction) while slightly improving accuracy (MAE: 0.0142). This reduction is crucial for the ROM-centric parameter storage and integer-only inference targeted in this work.

Table 4. Comparison of head architectures (Flatten vs. GAP).

Metric	Flatten-Based Head (Baseline)	GAP-Based Head (Proposed)
# Parameters	3121	2289
MAE	0.0147	0.0142
MSE	0.00048	0.00044

Number.

**Figure 3.** Comparison of Flatten-based head and GAP-based head for lightweight regression.

3.4. Impact of Depthwise Separable Convolution

Based on the *Baseline (GAP)* architecture, this subsection evaluates the effect of replacing standard convolutional blocks in the feature extractor with depthwise separable convolutions, as introduced in Section 2.4.1. To ensure a fair comparison, all experimental conditions—including input configuration, preprocessing, and training/evaluation protocols—were kept identical. The architectural backbone remained unchanged, with only the internal convolution operations being substituted. The primary objective of this evaluation is to quantitatively assess the impact of this substitution on both SoC estimation accuracy and model complexity.

As shown in Table 5, the adoption of separable convolution reduced the total parameter count from 2289 to 1178 (a 48.5% reduction). Despite this significant reduction in model size, the estimation performance remained stable; the MAE changed slightly from 0.0142 to 0.0144, and the MSE from 0.00044 to 0.00047. These results confirm that separable convolution can substantially decrease the computational and memory footprint while maintaining a comparable level of accuracy. Consequently, the depthwise separable structure was adopted as the fundamental building block for the final optimized model.

Table 5. Performance comparison between Baseline (GAP) and Baseline (GAP) + Separable Conv.

Metric	Baseline (GAP)	Baseline (GAP) + Separable Conv
Parameters	2289	1178
MAE	0.0142	0.0144
MSE	0.00044	0.00047

3.5. Structured Pruning and Model Compression

To achieve a model size suitable for ROM-based parameter storage and a single-MAC computational unit, we applied structured pruning to the separable 1D-CNN. Rather than applying the target compression rate in a single step, an iterative pruning and fine-tuning procedure was employed to minimize the degradation of SoC estimation accuracy [28]. All experiments were conducted under identical data split and training conditions to isolate the impact of the pruning strategy.

In this work, we compared two structured pruning criteria while applying the same iterative procedure. The first strategy utilizes the L_1 -norm of filters (channels) as an importance metric, prioritizing the removal of filters with smaller L_1 values. Given a filter weight vector $\mathbf{w}$, the importance is defined as $\|\mathbf{w}\|_1 = \sum_j |w_j|$, under the assumption that filters with smaller weight magnitudes contribute less to the output [29]. The second strategy is global-magnitude pruning, which sets a global threshold across the entire network and removes filters containing a high proportion of weights below this threshold [30].

Experimental results, summarized in Table 6, demonstrate that the L_1 -based strategy yields a smaller accuracy loss and more stable performance variations during the iterative process compared to the global-magnitude approach. Therefore, L_1 -based structured pruning was selected as the final compression strategy.

Table 6. Comparison of iterative structured pruning criteria.

Metric	Iterative L_1 -Based Pruning	Iterative Global-Magnitude Pruning
MAE	0.0140	0.0153
MSE	0.00048	0.0050

The application of L_1 -based pruning successfully reduced the number of parameters from 1178 to 358, significantly lowering the storage requirements. Furthermore, as shown in Table 7, the MAE improved from 0.0144 to 0.0140 and the MSE from 0.00047 to 0.00042. This outcome demonstrates that structured pruning can achieve both aggressive model compression and enhanced estimation accuracy through regularization effects.

Table 7. Performance and parameter count before and after applying the proposed pruning strategy.

Metric	Before Pruning	After Pruning
MAE	0.0144	0.0140
MSE	0.00047	0.00042
Param #	1178	358

Number.

Integer-Only Inference Configuration and QAT Selection

Targeting deployment on an automotive ECU, we prioritized a model configuration that allows for ROM-compatible integer parameters rather than maintaining float32 values. The integer inference path was designed to store weights in INT8 format, while performing accumulations and bias additions with INT32 precision. Notably, our memory requirement analysis considers the total ROM footprint, including weights as well as quantization metadata such as biases and layer/channel-wise scaling factors.

We evaluated both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). In regression-based SoC estimation, even minor quantization noise can directly lead to output errors, and lightweight models are particularly sensitive to such perturbations. Therefore, QAT was selected as the final approach to preserve integer-only inference accuracy. Initial comparisons between float32 and INT8 inference (summarized in Table 8) confirmed that simple quantization without fine-tuning could lead to increased error. Accordingly, all subsequent evaluations and final model validations were conducted under identical integer-only inference conditions.

Table 8. Performance comparison before and after INT8 quantization (float32 vs. INT8 inference).

	Float32	INT8
MAE	0.0140	0.0172
MSE	0.00042	0.00055

3.6. Overall Optimization Workflow and ROM-Deployable Parameter Generation

This subsection delineates the comprehensive optimization workflow designed for hardware implementation and the subsequent transformation of the model into a ROM-deployable parameter format. First, as the Flatten-based head—which expands the temporal dimension—can lead to a significant increase in the parameter count, it was replaced with a Global Average Pooling (GAP)-based head to structurally suppress parameter inflation. Furthermore, the feature extractor was optimized by substituting standard 1D convolutions with depthwise separable convolutions, thereby further reducing the model’s computational footprint under identical input conditions. The finalized network topology, incorporating these structural modifications, is illustrated in Figure 4.

Building upon this optimized architecture, iterative structured pruning was applied to achieve channel-wise compression. We employed a prune–retrain cycle based on the L_1 -norm criterion, where filters or channels with low importance were progressively removed, followed by fine-tuning to minimize the degradation of SoC estimation accuracy. To ensure the optimality of this strategy, a global-magnitude-based approach was also evaluated as a comparative baseline. The final pruning strategy was finalized after a comprehensive assessment of both SoC estimation metrics and the achieved parameter reduction efficiency.

Finally, Quantization-Aware Training (QAT) was performed to obtain optimized parameters for integer-only inference. Post-QAT, these parameters were transformed into a specific format suitable for direct integration into the hardware datapath. In this stage, rather than merely storing the weights, we organized the entire set of required parameters into a single Parameter ROM to ensure a hardware-centric memory evaluation. This unified ROM comprises: (i) INT8 weights, (ii) INT32 biases, and (iii) fixed-point scaling and shifting coefficients (metadata) required for integer requantization. Consequently, the total memory requirement was evaluated based on the total parameter volume within the ROM, rather than the weight size alone. The integrated workflow—spanning from structured pruning and QAT to final Parameter ROM packaging—is summarized in Figure 5.

The quantitative impact of each incremental optimization stage discussed in this paper is summarized in Table 9. From the baseline (Stage 0) to the final optimized model (Stage 4), the parameter count was reduced by 88.5% (from 3121 to 358), and the computational workload was decreased by 86.6% (from 44.7k to 6.0k FLOPs). These results demonstrate that the proposed structural modifications, including GAP, Separable Convolution, and structured pruning, effectively minimize the hardware resource requirements while maintaining a reliable SoC estimation accuracy of 0.0172 MAE under INT8 quantization.

Furthermore, as verified through Table 10, the SoC estimation performance exhibits minimal deviation across diverse temperature ranges, maintaining a consistent accuracy of 0.0172 MAE on average. By evaluating the model using 173,544 samples from $-10\text{ }^{\circ}\text{C}$ to $25\text{ }^{\circ}\text{C}$, we confirmed that the proposed 1D-CNN does not suffer from significant performance degradation even under harsh environmental changes. This stability across various temperature profiles demonstrates the environmental robustness required for practical automotive BMS applications, ensuring reliable deployment suitability.

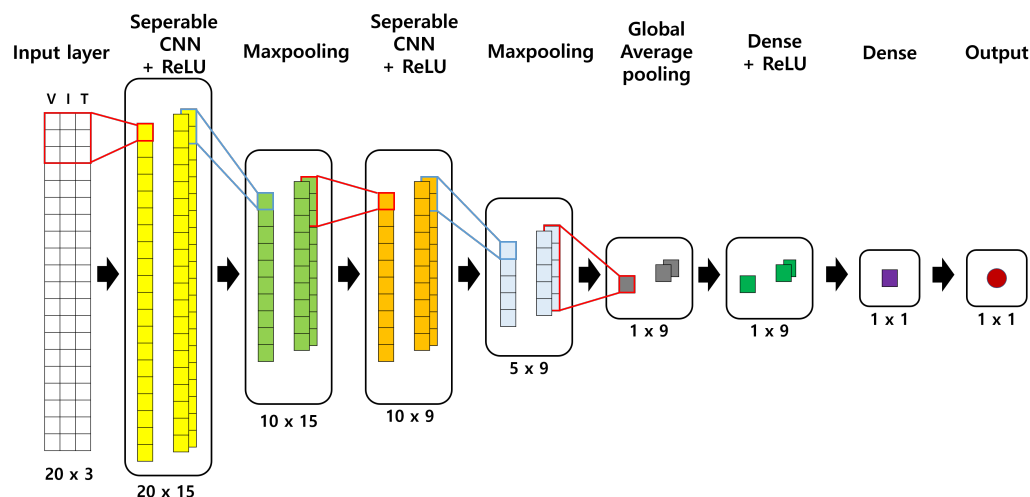


Figure 4. The final optimized network architecture reflects the proposed structural modifications.

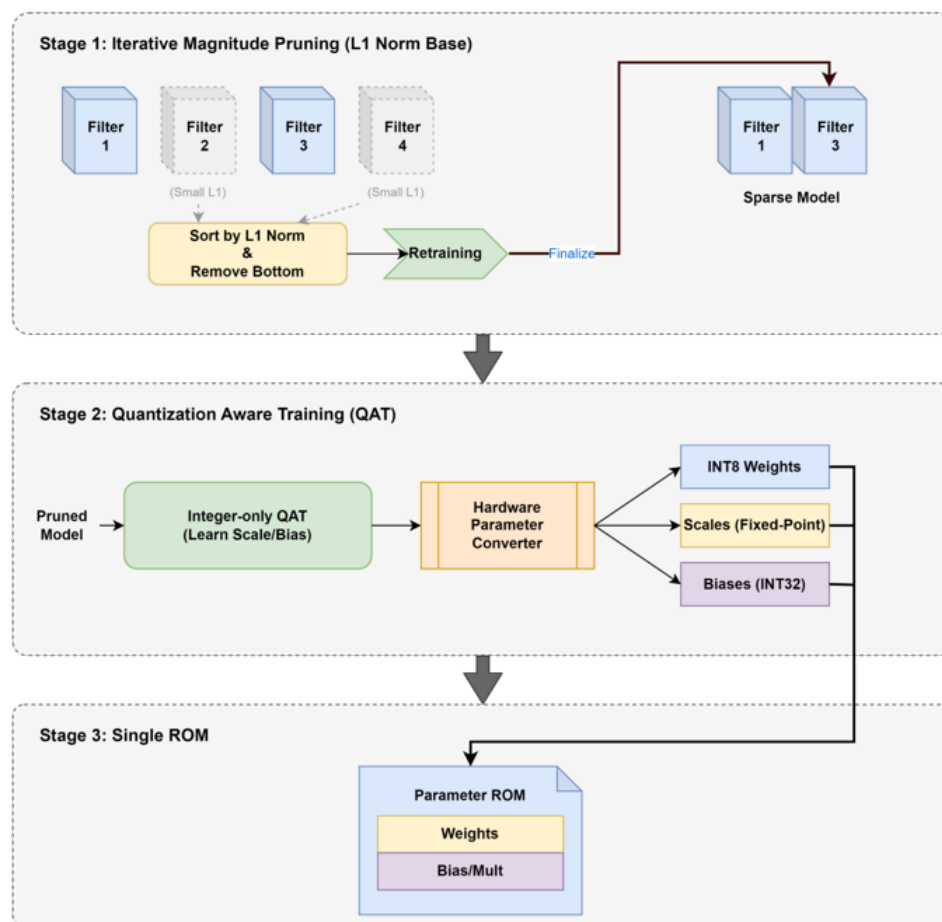


Figure 5. End-to-end pipeline of iterative pruning, QAT, and ROM-ready parameter generation.

Table 9. Evaluation of the incremental optimization steps: Comparison of MAE, MSE, the number of parameters (#Params), and FLOPs.

Optimization Step	#Params	FLOPs	MAE	MSE
Baseline (Flatten)	3121	44.7 k	0.0147	0.00048
Baseline(GAP)	2289	42.8 k	0.0142	0.00044
Baseline(GAP) + SepConv	1178	17.3 k	0.0144	0.00047
Baseline(GAP) + SepConv + Pruning	358	6.0 k	0.0140	0.00042
Baseline(GAP) + SepConv + Pruning + INT8 QAT	358	–	0.0172	0.00055

Number.

Table 10. Detailed SoC estimation performance across various temperatures (173,544 samples).

Temperature (°C)	# Samples	MAE	MSE
−10 °C	28,579	0.0246	0.00091
0 °C	32,576	0.0191	0.00057
10 °C	36,168	0.0141	0.00039
25 °C	76,221	0.0151	0.00049
Total/ Average	173,544	0.0172	0.00055

Number.

3.7. Comparative Analysis of Quantization Robustness

To validate the effectiveness of the proposed quantization strategy, we identified a recent study that implemented an INT8-quantized 1D-CNN for battery SoC estimation [12]. Although the dataset utilized in the reference study differs from ours, we conducted a comparative analysis with this work to evaluate the robustness of the quantization schemes. A key distinction lies in the optimization pipeline: while the reference work focused on converting a standard pre-trained model using Post-Training Quantization (PTQ), our approach adopts Quantization-Aware Training (QAT) to preserve the accuracy of a model that has already undergone aggressive parameter reduction strategies, such as structured pruning.

Table 11 presents the performance degradation caused by the quantization process in both studies. The reference study exhibited a significant increase in error metrics after applying PTQ, with the RMSE increasing by approximately 2.09% to 2.62% compared to its floating-point baseline [12]. This indicates that standard PTQ limits the model's ability to recover from information loss during bit-width reduction.

In contrast, the proposed design demonstrates superior robustness. Although our model operates with a significantly reduced parameter count due to prior architectural optimizations, the QAT process effectively suppressed the quantization noise. Consequently, our model achieved a minimal RMSE increase of only 0.296% and an MAE increase of 0.320%. These results confirm that employing QAT ensures high inference reliability even when applied to a highly compact, pruned architecture, whereas PTQ struggles to maintain precision.

To further contextualize this robustness, Table 12 presents an extended literature comparison for embedded TinyML/SoC-friendly battery SoC estimation. The table summarizes representative prior works using common comparison axes, including model type, number of parameters, quantization type, hardware platform, and error metrics. For a uniform presentation, all errors are reported in fraction form (rather than %), and when only RMSE is available, MSE is derived using $MSE = RMSE^2$, as noted below Table 12. Although direct one-to-one numerical comparison is inherently limited by differences in datasets, driving profiles, and evaluation protocols, the overall trend indicates that QAT more consistently preserves accuracy under aggressive model compression, whereas PTQ-based INT8 deployments tend to exhibit larger precision degradation.

Gaussian noise was injected into the current and voltage inputs to evaluate the sensitivity of the proposed pipeline to measurement perturbations. As summarized in Table 13, the MAE increased by 9.88% at the 1.0% noise level, and the degradation remained gradual even at 1.5% noise while keeping the absolute MAE around 2% SoC. In addition, the RMSE increased by 7.05% at 1.0% noise and by 15.21% at 1.5% noise. Note that MSE can exhibit a larger relative increase because the squared-error metric is more sensitive to occasional larger deviations.

Table 11. Comparison of quantization-induced performance degradation (drop in accuracy).

Metric	Reference Study [12]	Proposed Work
RMSE Increase (Drop)	2.09–2.62%	0.296%
MAE Increase (Drop)	2.06–2.73%	0.320%
MSE Increase (Δ)	1.92×10^{-3}	0.13×10^{-3}

Table 12. Comparative analysis of state-of-the-art lightweight SoC estimation models for resource-constrained embedded systems.

Model Type	#Params	Quantization	Hardware Platform	Implementation	MAE	MSE
1D-CNN (Proposed)	358	QAT (INT8)	FPGA (Nexys Video) ASIC (TSMC 28 nm)	Custom RTL accel.	0.0172	0.00055
1D-CNN [12]	3.8 k	PTQ (INT8)	STM32L476RG (M4)	Embedded SW infer.	0.0431	0.00234
ANN [31]	6.5 k	PTQ (INT8)	Infineon PSoC 62S3 (Cortex-M4/M0+)	Embedded SW infer.	0.0282	0.00151
1D-CNN [31]	24.5 k	PTQ (INT8)	Infineon PSoC 62S3 (Cortex-M4/M0+)	Embedded SW infer.	0.0407	0.00243
Parallel TCNs [32]	14,529	FP32	STM32H573I-DK STM32MP257F	Embedded SW infer.	0.0311	0.00062

Number. Note: “Custom RTL accel.” indicates a dedicated hardware accelerator designed in this work. “Embedded SW infer.” indicates software inference executed on the listed embedded processor/board. MAE and MSE for quantized models are reported for their respective bit-widths. For [32], values are converted to a decimal scale for consistent comparison.

Table 13. Input-noise sensitivity analysis (Gaussian noise injection).

Noise Level	MAE	Δ MAE	Inc. (%)	RMSE	Δ RMSE	Inc. (%)	MSE	Δ MSE	Inc. (%)
0% (baseline)	0.0172	—	—	0.02345	—	—	0.00055	—	—
0.5%	0.0174	0.0002	1.16	0.02387	0.00043	1.81	0.00057	0.00002	3.64
1.0%	0.0189	0.0017	9.88	0.02510	0.00165	7.05	0.00063	0.00008	14.55
1.5%	0.0201	0.0029	16.86	0.02702	0.00357	15.21	0.00073	0.00018	32.73

4. Hardware Implementation and Validation

4.1. Overall Accelerator Architecture

This subsection delineates the overall hardware architecture of the proposed 1D-CNN SoC estimation accelerator utilized for FPGA verification. The accelerator comprises a Main Controller (FSM) for high-level orchestration, a Weight ROM for providing parameters, a Ping-Pong Feature RAM for storing intermediate feature maps, and a Compute Engine. The Compute Engine consists of depthwise and pointwise convolutional blocks, activation and downsampling (ReLU/MaxPool) units, a Global Average Pooling (GAP) block for temporal reduction, and a Dense block for generating the final regression output.

The data flow is initiated by loading the input time-series into the feature buffer. Subsequently, the controller selects the feature read/write banks according to the layer execution sequence and generates addresses for both the Weight ROM and feature RAM to

sequentially supply the required data and parameters to each computational block. During layer execution, one feature RAM bank is dedicated to input reads, while the other is used for write-back of the output. Upon the completion of a layer, the roles of the two banks are swapped to provide the input for the subsequent layer. This ping-pong buffering mechanism simplifies inter-layer data movement within resource-constrained environments and enables continuous inference execution by mitigating read/write conflicts.

Each operational block is activated by an enable signal from the controller, while internal iterations—such as kernel positioning and channel/filter traversal—are managed by address generators and iteration counters. The results are stored in the buffers on a layer-by-layer basis, and the final dense output is transmitted to an external host via a UART interface for data collection and analysis. Figure 6 illustrates the overall architecture and the interconnections between the functional blocks of the proposed accelerator verified on the FPGA.

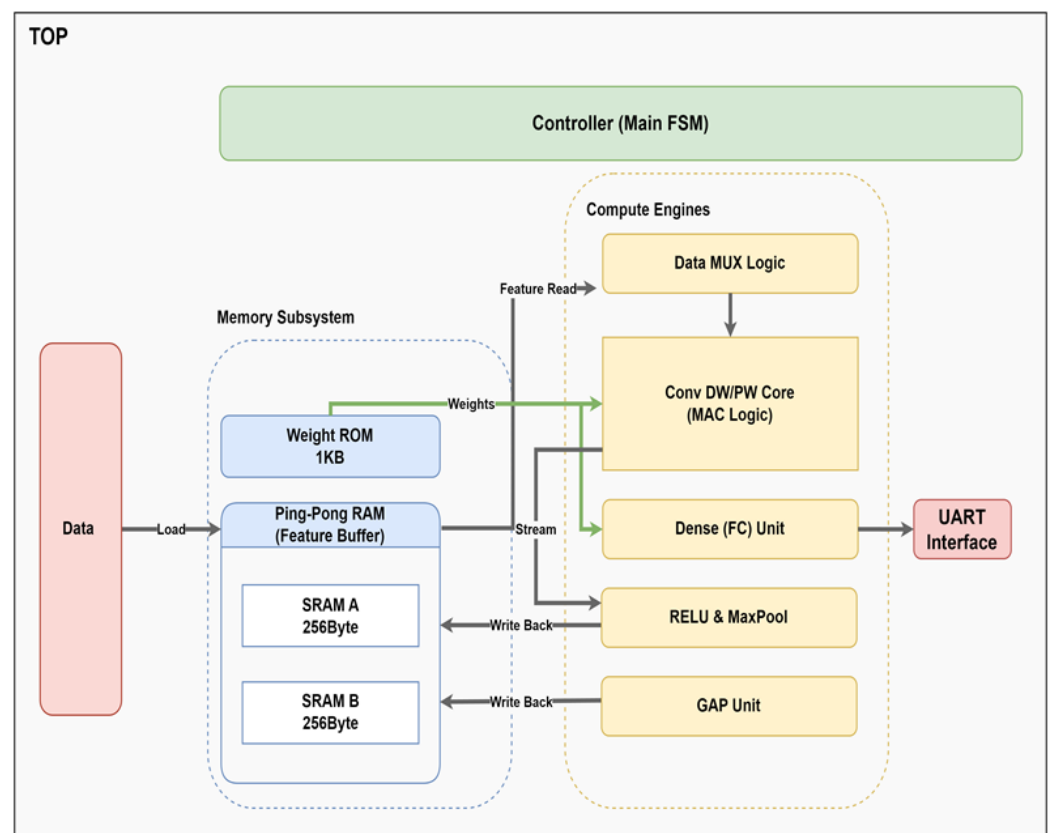


Figure 6. Overall hardware architecture of the proposed INT8 1D-CNN accelerator for real-time SoC estimation.

4.2. Resource-Reuse Architecture for Separable Convolution Units

This section describes the single MAC (time-multiplexed) reuse architecture applied to the proposed 1D-CNN accelerator to efficiently perform separable convolution (depthwise and pointwise) operations in terms of area. Generally, separable convolution performs spatial (temporal) filtering per channel during the depthwise convolution (DW) stage, followed by channel combination in the pointwise convolution (PW) stage to generate output channels. In this implementation, rather than configuring separate parallel computational units for DW and PW, DSP and LUT resources are minimized by sequentially reusing a single MAC core for both DW and PW operations.

As illustrated in Figure 7, input feature maps are read from either Bank A or Bank B of the ping-pong feature RAM and supplied to the separable convolution unit. The controller

selects the read and write banks according to the current layer execution stage. During the DW operation stage, data from the selected input bank is read and supplied to the single MAC core. The MAC core performs per-channel accumulation using a K-tap (e.g., $K = 3$) input window and DW weights, subsequently storing the DW results in an internal buffer. In the following PW stage, the DW results stored in the internal buffer are reused as inputs, and channel-combination accumulation is performed using the same MAC core with PW weights. The final PW output is recorded back into the opposite bank (Bank B) of the ping-pong RAM to be used as input for the next layer.

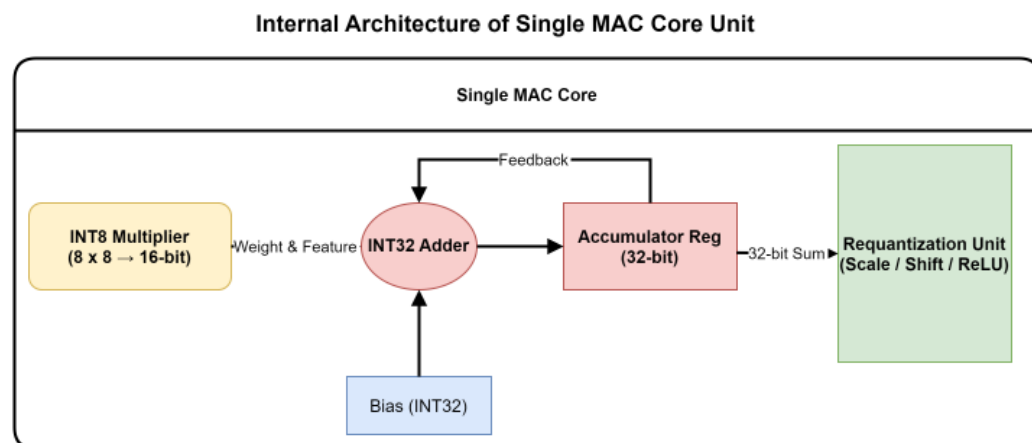


Figure 7. Architecture of the time-multiplexed separable convolution unit with ping-pong buffering and an internal DW-to-PW buffer.

In this architecture, the internal buffer serves as temporary storage for DW outputs and minimizes external memory access during the DW-to-PW transition, thereby reducing data movement overhead. Furthermore, since DW and PW share the same MAC core, the controller strictly manages the operational sequence through state transitions that include operational modes (DW/PW), weight selection, input selection, accumulation initialization, and completion conditions. Consequently, the proposed architecture processes the two-stage operations of separable convolution using a single computational resource, enabling end-to-end inference even under limited DSP and BRAM resource constraints in an FPGA environment.

4.3. Internal Datapath and Integer Arithmetic Path of the Single MAC Core

This section describes the internal structure and integer-based datapath of the single MAC core, which is the key computational block of the proposed accelerator. The design is configured to perform inference using only INT8/INT32 integer paths without floating-point operations. Input features and weights are provided in INT8 format, while accumulation is performed in INT32. This approach ensures sufficient accumulation range for CNN operations while simplifying the hardware implementation.

As shown in Figure 8, the single MAC core consists of (1) an INT8 multiplier, (2) an INT32 adder, (3) an accumulator register, and (4) a requantization unit. First, the INT8 input features and INT8 weights are multiplied, and the product is forwarded to the internal accumulation path. Subsequently, the INT32 adder combines the previous accumulation value (from the accumulator register) with the product result to generate a new accumulation value, which is then stored back in the accumulator register. The feedback path of the accumulator register is structured to maintain accumulation throughout kernel tap and channel iterations. Additionally, since bias values for convolution and dense operations are provided in INT32 format, a path is included to add the bias during the accumulation process to form the final accumulated value.

Once accumulation is complete, the INT32 result is transferred to the requantization unit. The requantization unit converts the INT32 value back into the INT8 range based on learned quantization parameters (scale and shift). This process sequentially performs (i) scale application (fixed-point multiplication), (ii) scale correction via shift operations, (iii) saturation and clipping, and (iv) the activation function (ReLU). Finally, the generated INT8 output is stored in either the ping-pong RAM or the internal buffer for use as input in the subsequent layer.

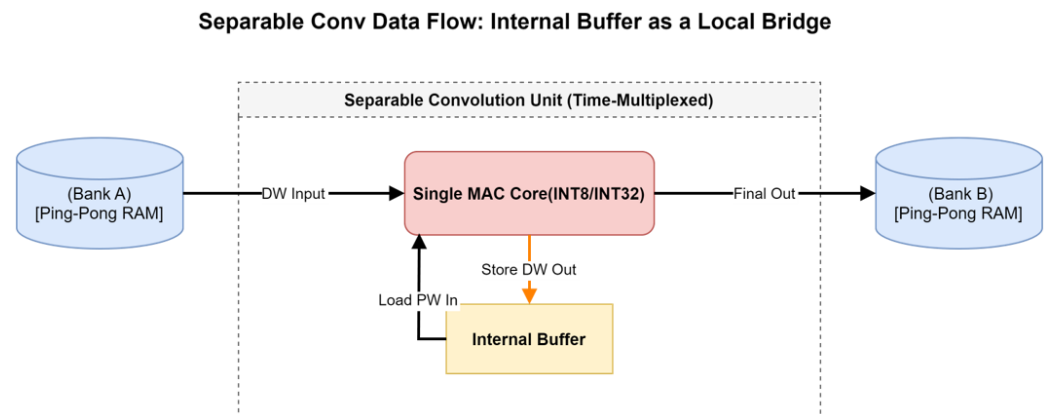


Figure 8. Datapath of the single MAC core and post-processing unit for integer-only inference.

As such, the single MAC core provides integrated fundamental operations (multiply-accumulate-requantization) commonly used in DW/PW convolution and dense operations. The controller manages input sources (ping-pong RAM or internal buffer), weight selection, accumulation initialization, the number of accumulation iterations, and output recording timing according to the operation stage. Therefore, the proposed structure reduces hardware area through the reuse of the computational unit while maintaining consistent integer-based inference rules, providing an advantageous structure for functional verification against software references.

4.4. FPGA Verification Setup and Functional Validation Procedure

To validate the operational integrity and functional consistency of the proposed accelerator, this study established an FPGA-based verification environment and performed an end-to-end inference-based validation procedure. The implemented RTL design was deployed on a Digilent Nexys Video board (Artix-7 series), and the inference results were transmitted to a host PC via a UART interface. The host PC logs the received SoC estimates in chronological order and calculates corresponding values from a software reference inference using the same input data. A sample-by-sample comparison is then conducted between the hardware and software outputs to verify the functional correctness and output reliability. This setup ensures the robust execution of end-to-end inference on the FPGA while simultaneously validating the stability of the data transmission path. Figure 9 illustrates the FPGA-based verification environment utilized in this study.

Validation was conducted by performing inference on the input windows, after which the SoC results output by the FPGA were collected at the host side for comparison with the reference results. The input data consisted of the same window-based time-series format used during the training phase, and computations within the FPGA were executed according to the predefined integer datapath rules. In the event of observed discrepancies, the system was configured to selectively monitor intermediate results at layer boundaries, allowing for the isolation and analysis of inconsistent segments on a layer-by-layer basis. Furthermore, output consistency was verified through repetitive executions with identical inputs, thereby confirming both the stability of the UART communication path and the

reproducibility of the results. These procedures confirmed that the proposed accelerator operates on the FPGA with the intended computational sequence and data flow.

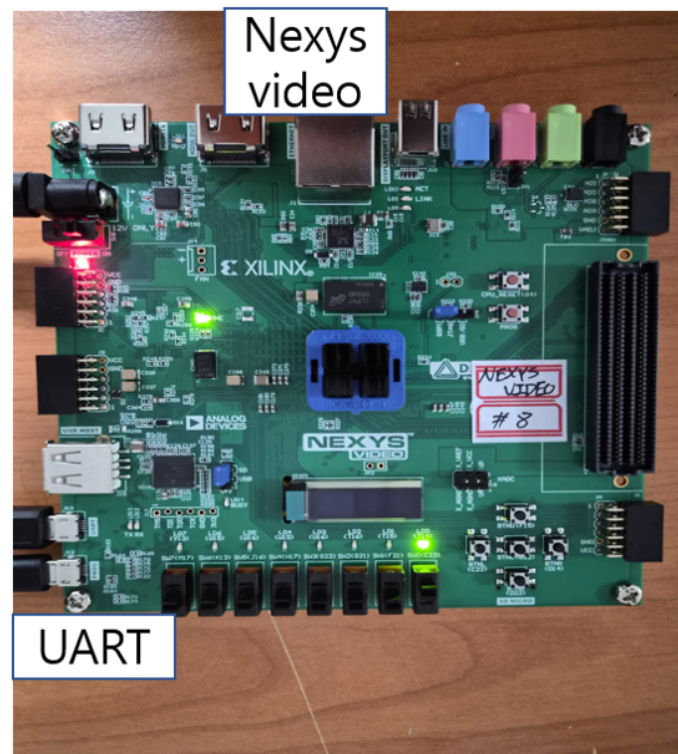


Figure 9. FPGA-based validation setup using the Digilent Nexys Video board.

4.5. FPGA Resource Utilization

This subsection quantitatively presents the results of implementing the proposed 1D-CNN SoC estimation accelerator on the Nexys Video FPGA board. Following synthesis and implementation, the utilized resources are summarized in terms of LUTs, FFs, BRAMs, and DSPs, and it was confirmed that the design operates reliably at the target operating frequency. Table 14 summarizes the resource utilization and operating frequency on the Nexys Video board. The implementation focuses on an integer-based datapath and FSM-based control, specifically designed to execute end-to-end inference within the constraints of limited on-chip resources.

Table 14. FPGA resource utilization and operating frequency on the Nexys Video platform.

Device	LUT	FF	BRAM (Bits)	DSP	Frequency
Nexys Video	3239	4520	55,296	12	100 MHz

4.6. ASIC Synthesis and Area Estimation

To evaluate the silicon area and hardware complexity for potential mass production, we performed logic synthesis using Synopsys Design Compiler. The proposed accelerator was mapped to a TSMC 28nm HPC+ CMOS standard cell library under a target system clock of 100 MHz, consistent with the FPGA verification environment. This process allows for a quantitative estimation of the logic gate count and memory footprint in an ASIC environment.

Table 15 summarizes the comprehensive ASIC synthesis and hardware performance results of the proposed accelerator. The total estimated cell area, including the compute engine, controller, and all on-chip memory macros (SRAM and Weight ROM), is approxi-

mately 37,199 μm^2 , which corresponds to an equivalent gate count of 98,410 gates when normalized to a 2-input NAND gate. At an operating frequency of 100 MHz, the proposed design requires 6000 clock cycles per input window, resulting in an inference latency of 60 μs and a throughput of 16,666 FPS. Post-synthesis power estimation further indicates a total power consumption of 1.9988 mW, corresponding to an energy cost of 119.93 nJ per inference. Notably, with 60 μs latency, a 128-cell battery pack can be processed sequentially in only 7.68 ms, ensuring reliable real-time operation within standard 10–100 ms BMS sampling intervals.

Table 15. Comprehensive ASIC implementation and hardware performance results.

Metric	Proposed Accelerator
Process Technology	TSMC 28 nm HPC+
Supply Voltage	1.62 V
Operating Frequency	100 MHz
Total Area of Chip	98,410 Gate
– Logic Area (excluding memory)	66,862 Gate
– SRAM Area	24,861 Gate
– Weight ROM Area	6687 Gate
Clock Cycles per Window	6000 cycles
Inference Latency	60 μs
Throughput	16,666 FPS
Total Power Consumption	1.9988 mW
Energy per Inference	119.93 nJ

5. Conclusions

This study presents a lightweight 1D-CNN-based inference pipeline and its FPGA implementation results, targeting real-time battery State-of-Charge (SoC) estimation for automotive BMS applications. While maintaining a 1D-convolution-based feature extraction structure for time-series sensor inputs, the proposed model significantly reduces the parameter count and computational complexity by incorporating depthwise separable convolutions and global average pooling (GAP). Furthermore, the model footprint was minimized through iterative structured pruning. By leveraging quantization-aware training (QAT), the entire inference pipeline was configured with INT8/INT32 integer-only computation paths, ensuring a bit-exact, reproducible computational flow that aligns with software references without the need for floating-point units.

In terms of hardware implementation, the proposed 1D-CNN was implemented in Verilog HDL, adopting an FSM-based control structure that enhances area efficiency by reusing a single MAC unit through time-multiplexing. Parameters are stored in on-chip ROM, and intermediate feature maps are managed via a ping-pong buffering mechanism to sustain continuous layer execution under limited memory resources. Verification was conducted using a Nexys Video FPGA board, confirming end-to-end operational integrity by transmitting inference results to a host PC via UART. The implementation results demonstrated that the proposed accelerator operates reliably at 100 MHz, with resource utilization measured at 3239 LUTs, 4520 FFs, 55,296 bits of BRAM, and 12 DSPs. Moreover, synthesis using Design Compiler reported approximately 98,410 gates, quantitatively demonstrating the feasibility of the proposed design for lightweight embedded deployment.

However, as this validation primarily focused on FPGA-based functional verification and software-reference comparison, further comprehensive evaluation regarding power consumption and long-term operating conditions, such as temperature and aging variations in actual vehicle ECU environments, is required. Future research will extend this

work within the same integer-datapath framework to (i) reduce latency through memory access optimization and improved scheduling, (ii) optimize power and area specifically for ASIC/SoC integration, and (iii) enhance generalization performance by expanding the datasets to encompass diverse driving scenarios and degradation states.

In addition, although the present study focuses on a single-cell dataset (LG HG2 18650) to establish and validate the proposed lightweight inference architecture, extending the evaluation to multiple 18650-series cells with different manufacturers and characteristics remains an important future direction. Such an extension would enable a more comprehensive assessment of cell-to-cell generalization capability.

6. Limitations and Future Work

While this study focuses on an open single-cell dataset (LG HG2 18650) to establish and validate the proposed lightweight integer-only inference architecture, further verification is required to fully assess generalization and long-term reliability in practical automotive deployment.

Future work will expand the evaluation to 18650-series cells from various manufacturers and chemistries, employing stricter cross-cycle hold-out protocols to quantitatively assess robustness against unseen driving patterns. Furthermore, we intend to extend the framework to aging-aware scenarios by incorporating State-of-Health (SoH) information. By leveraging long-term degradation datasets, future studies will investigate joint SoC–SoH estimation while maintaining the same hardware-efficient design principles, ensuring sustained accuracy under real-world operating conditions.

Author Contributions: Conceptualization, S.K.; methodology, S.K.; software, Y.L. and G.J.; validation, Y.L. and G.J.; formal analysis, S.K.; investigation, S.K.; resources, S.K.; data curation, S.K. and Y.L.; writing—original draft preparation, S.K.; writing—review and editing, S.L.; visualization, S.K.; supervision, S.L.; project administration, S.L.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korea Government (MOTIE) (RS-2023-00232192, RS-2024-00403397, RS-2022-00155731, RS-2024-00403483); the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (2021M3H2A1038042); and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea Government (MSIT) (RS-2025-02214672).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Pfrang, A.; Kriston, A.; Ruiz, V.; Lebedeva, N.; di Persio, F. Chapter Eight—Safety of Rechargeable Energy Storage Systems with a focus on Li-ion Technology. In *Emerging Nanotechnologies in Rechargeable Energy Storage Systems: Micro and Nano Technologies*; Elsevier: Amsterdam, The Netherlands, 2017; pp. 253–290. [\[CrossRef\]](#)
2. Oh, S.; Kim, J.; Moon, I. Hybrid data-driven deep learning model for state of charge estimation of Li-ion battery in an electric vehicle. *J. Energy Storage* **2024**, *97*, 112887. [\[CrossRef\]](#)
3. Cui, Z.; Wang, L.; Li, Q.; Wang, K. A comprehensive review on the state of charge estimation for lithium-ion battery based on neural network. *Int. J. Energy Res.* **2022**, *46*, 5423–5440. [\[CrossRef\]](#)
4. Nguyen, N.; Balasingam, B. Lithium-Ion Battery Open-Circuit Voltage Analysis for Extreme Temperature Applications. *Energies* **2026**, *19*, 27. [\[CrossRef\]](#)

5. Movassagh, K.; Raihan, A.; Balasingam, B.; Pattipati, K. A Critical Look at Coulomb Counting Approach for State of Charge Estimation in Batteries. *Energies* **2021**, *14*, 4074. [\[CrossRef\]](#)
6. Mou, J.; Yang, Q.; Tang, Y.; Liu, Y.; Li, J.; Yu, C. Prediction of the Remaining Useful Life of Lithium-Ion Batteries Based on the 1D CNN-BLSTM Neural Network. *Batteries* **2024**, *10*, 152. [\[CrossRef\]](#)
7. Yu, C.; Zhu, J.; Liu, W.; Dai, H.; Wei, X. Co-estimation of state-of-charge and state-of-temperature for large-format lithium-ion batteries based on a novel electrothermal model. *Green Energy Intell. Transp.* **2024**, *3*, 100152. [\[CrossRef\]](#)
8. Li, Y.; Ye, M.; Wang, Q.; Lian, G.; Xia, B. An improved model combining machine learning and Kalman filtering architecture for state of charge estimation of lithium-ion batteries. *Green Energy Intell. Transp.* **2024**, *3*, 100163. [\[CrossRef\]](#)
9. Mehta, C.; Sant, A.V.; Sharma, P. Optimized ANN for LiFePO₄ battery charge estimation using principal components based feature generation. *Green Energy Intell. Transp.* **2024**, *3*, 100175. [\[CrossRef\]](#)
10. Sulaiman, M.H.; Mustaffa, Z. State of charge estimation for electric vehicles using random forest. *Green Energy Intell. Transp.* **2024**, *3*, 100177. [\[CrossRef\]](#)
11. Chen, Y.; Duan, W.; He, Y.; Wang, S.; Fernandez, C. A hybrid data driven framework considering feature extraction for battery state of health estimation and remaining useful life prediction. *Green Energy Intell. Transp.* **2024**, *3*, 100160. [\[CrossRef\]](#)
12. Mazzi, Y.; Ben Sassi, H.; Gaga, A.; Errahimi, F. State of charge estimation of an electric vehicle's battery using tiny neural network embedded on small microcontroller units. *Int. J. Energy Res.* **2022**, *46*, 8102–8119. [\[CrossRef\]](#)
13. Wang, Y.-C.; Shao, N.-C.; Chen, G.-W.; Hsu, W.-S.; Wu, S.-C. State-of-Charge Estimation for Lithium-Ion Batteries Using Residual Convolutional Neural Networks. *Sensors* **2022**, *22*, 6303. [\[CrossRef\]](#)
14. Bhattacharjee, A.; Verma, A.; Mishra, S.; Saha, T.K. Estimating State of Charge for xEV Batteries Using 1D Convolutional Neural Networks and Transfer Learning. *arXiv* **2020**, arXiv:2011.00841. [\[CrossRef\]](#)
15. Zhang, R.; Xia, B.; Li, B.; Cao, L.; Lai, Y.; Zheng, W.; Wang, H.; Wang, W. A Study on the Open Circuit Voltage and State of Charge Characterization of High Capacity Lithium-Ion Battery Under Different Temperature. *Energies* **2018**, *11*, 2408. [\[CrossRef\]](#)
16. Barcellona, S.; Paccanaro, A.; Meloni, A.; Malavasi, S.; Martinetti, S.; Berrueta, A. Calendar Aging Effect on the Open Circuit Voltage of Lithium-Ion Battery. *Energies* **2023**, *16*, 4869. [\[CrossRef\]](#)
17. Ahmed, A.A.; Ali, W.; Abdullah, T.A.A.; Malebary, S.J. Classifying Cardiac Arrhythmia from ECG Signal Using 1D CNN Deep Learning Model. *Mathematics* **2023**, *11*, 562. [\[CrossRef\]](#)
18. Mittal, S. A Survey of FPGA-Based Accelerators for Convolutional Neural Networks. *Neural Comput. Appl.* **2020**, *32*, 1109–1139. [\[CrossRef\]](#)
19. Sheng, T.; Feng, C.; Zhuo, S.; Zhang, X.; Shen, L.; Aleksic, M. A Quantization-Friendly Separable Convolution for MobileNets. In Proceedings of the 2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), Williamsburg, VA, USA, 25 March 2018; pp. 14–18. [\[CrossRef\]](#)
20. Malla, P.P.; Sahu, S.; Alutaibi, A.I. Classification of Tumor in Brain MR Images Using Deep Convolutional Neural Network and Global Average Pooling. *Processes* **2023**, *11*, 679. [\[CrossRef\]](#)
21. He, Y.; Xiao, L. Structured Pruning for Deep Convolutional Neural Networks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 2900–2919. [\[CrossRef\]](#) [\[PubMed\]](#)
22. Nahshan, Y.; Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Bronstein, A.M.; Mendelson, A. Loss Aware Post-Training Quantization. *Mach. Learn.* **2021**, *110*, 3245–3262. [\[CrossRef\]](#)
23. Choi, D.; Park, J.; Kim, H. HLQ: Hardware-Friendly Logarithmic Quantization Aware Training for Power-Efficient Low-Precision CNN Models. *IEEE Access* **2024**, *12*, 159611–159621. [\[CrossRef\]](#)
24. Kollmeyer, P.; Vidal, C.; Naguib, M.; Skells, M. LG 18650HG2 Li-ion Battery Data and Example Deep Neural Network xEV SOC Estimator Script. Mendeley Data. 2020. V3. Available online: <https://data.mendeley.com/datasets/cp3473x7xv/3> (accessed on 4 February 2026).
25. Jiao, M.; Wang, D. The Savitzky-Golay Filter Based Bidirectional Long Short-Term Memory Network for SOC Estimation. *Int. J. Energy Res.* **2021**, *45*, 19467–19480. [\[CrossRef\]](#)
26. Raudys, A.; Lenčiauskas, V.; Malčius, E. Moving Averages for Financial Data Smoothing. In *Information and Software Technologies (ICIST 2013)*; Skersys, T., Butleris, R., Butkiene, R., Eds.; Communications in Computer and Information Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 403, pp. 34–45. [\[CrossRef\]](#)
27. Klinker, F. Exponential Moving Average Versus Moving Exponential Average. *Math. Semesterber.* **2011**, *58*, 97–107. [\[CrossRef\]](#)
28. Chang, J.; Lu, Y.; Xue, P.; Xu, Y.; Wei, Z. Iterative Clustering Pruning for Convolutional Neural Networks. *Knowl.-Based Syst.* **2023**, *265*, 110386. [\[CrossRef\]](#)
29. Kumar, A.; Shaikh, A.M.; Li, Y.; Bilal, H.; Yin, B. Pruning Filters with L1-Norm and Capped L1-Norm for CNN Compression. *Appl. Intell.* **2021**, *51*, 1152–1160. [\[CrossRef\]](#)
30. Gupta, M.; Camci, E.; Keneta, V.R.; Vaidyanathan, A.; Kanodia, R.; James, A. Is Complexity Required for Neural Network Pruning? A Case Study on Global Magnitude Pruning. In Proceedings of the 2024 IEEE Conference on Artificial Intelligence (CAI), Singapore, 25–27 June 2024. [\[CrossRef\]](#)

31. Giazitzis, S.; Sakwa, M.; Leva, S.; Ogliari, E.; Badha, S.; Rosetti, F. A Case Study of a Tiny Machine Learning Application for Battery State-of-Charge Estimation. *Electronics* **2024**, *13*, 1964. [[CrossRef](#)]
32. Pau, D.P.; Aniballi, A. Tiny Machine Learning Battery State-of-Charge Estimation Hardware Accelerated. *Appl. Sci.* **2024**, *14*, 6240. [[CrossRef](#)]

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