

Article

Comparative Study of Embedded Deep Learning Models for State of Charge Estimation of Lithium-Ion Batteries

Yiyun Liang, Xun Lin, Xin Tang, Wenjing Shen, Jie Wang and Linfeng Zheng * 

Sino-German College of Intelligent Manufacturing, Shenzhen Technology University, Shenzhen 518118, China

* Correspondence: zhenglinfeng@sztu.edu.cn

Abstract

Accurate state of charge (SOC) estimation is essential for the safe and efficient operation of lithium-ion battery systems, while deploying deep learning-based estimators in embedded battery management systems remains challenging due to limited memory, computational capability, and real-time constraints. To address this issue, this study comparatively investigates four deep learning models, namely CNN, LSTM, CNN-LSTM, and lightweight TinyCNN, for embedded-oriented SOC estimation. A unified framework integrating data preprocessing, model training, post-training INT8 quantization, and TensorFlow Lite Micro deployment on an ESP32 microcontroller is established. Results under the US06 driving condition show that all models can track the SOC trajectory, although their relative performance differs between FP32 and INT8 inference. CNN-LSTM achieves the lowest FP32 error, with an MAE of 1.31%, while LSTM achieves the lowest INT8 error, with an MAE of 1.70%. Although TinyCNN has a higher INT8 MAE of 2.55%, it contains only 2913 parameters, occupies 8.51 KiB in INT8 form, and achieves the lowest measured ESP32 mean inference latency of 41.992 ms. These results demonstrate the feasibility of the evaluated INT8 models on the ESP32 under the US06 condition and indicate that model selection should depend on application-specific trade-offs among estimation accuracy, model size, runtime memory, and inference latency.

Keywords: state-of-charge estimation; embedded deep learning; battery management system; lithium-ion battery

1. Introduction

Lithium-ion batteries are widely used in electric vehicles, unmanned systems, and distributed energy storage systems because of their high energy density and long cycle life [1]. In these applications, the battery management system (BMS) plays a crucial role in ensuring safe, reliable, and efficient operation. Among various BMS functions, the state of charge (SOC) estimation is one of the most fundamental and important state variables, since it directly affects energy management, charge or discharge management, driving range prediction, and battery prediction strategies [2,3]. However, SOC cannot be directly measured and must instead be inferred from measurable signals such as voltage, current, and temperature [4,5]. Owing to the strong nonlinearity, time-varying characteristics, and operating-condition dependence of lithium-ion batteries, accurate SOC estimation remains a challenging problem.

In recent years, deep learning methods have attracted increasing attention for SOC estimation. Compared with conventional methods, such as equivalent circuit model-based methods and electrochemical model-based methods, deep learning approaches have a



Academic Editor: Grzegorz Sierpiński

Received: 20 June 2026

Revised: 24 July 2026

Accepted: 30 July 2026

Published: 3 August 2026

Copyright: © 2026 by the authors.

Published by MDPI on behalf of the

World Electric Vehicle Association.

Licensee MDPI, Basel, Switzerland.

This article is an open access article

distributed under the terms and

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

strong ability to model nonlinear input–output relationships directly from battery voltage, current, and temperature data, without requiring in-depth knowledge of electrochemical mechanisms of batteries [6,7]. Among them, convolutional neural networks (CNN), long short-term memory networks (LSTM), and hybrid CNN–LSTM models have shown promising performance in extracting temporal features from battery operating data and achieving high accuracy of the SOC estimation under dynamic load profiles [8–10].

Despite the encouraging estimation accuracy reported by many existing studies, the practical implementation of deep learning-based SOC estimation in real BMSs is still limited. Most previous investigations are conducted on high-performance computing platforms, such as desktop computers or cloud-based environments, where memory and computational resources are sufficient. In contrast, embedded BMS platforms are commonly built on microcontrollers with limited resources on Flash, RAM, processing capability, and power consumption [11,12]. Under such hardware limitations, complex deep learning models may suffer from excessive memory demand, long inference latency, and limited real-time applicability. As a result, there is an evident gap between algorithm-level SOC estimation research and practical embedded deployment [13–15]. Systematic evaluations that jointly consider leakage-free model comparison, FP32-to-INT8 deployment behavior, and measured microcontroller performance within a unified framework are less frequently reported [13–17].

To bridge this gap, embedded-oriented model design and optimization have become increasingly important. Lightweight neural network architectures can reduce parameter count and computational complexity while maintaining acceptable estimation accuracy, whereas model compression techniques such as quantization can further reduce memory usage and improve inference efficiency on microcontrollers [11,12,16]. Against this background, this work investigates the feasibility of deploying deep learning-based SOC estimation models on resource-constrained embedded platforms [14,15,17]. Specifically, four representative models, namely CNN, LSTM, CNN–LSTM, and TinyCNN, are developed and comparatively evaluated under the US06 driving condition. Post-training INT8 quantization is further applied to all models, and the quantized models are deployed on an ESP32 microcontroller using TensorFlow Lite Micro to assess their estimation performance, computational cost, and practical deployment feasibility. Based on these results, the trade-off between estimation accuracy and deployment efficiency is analyzed, providing useful guidance for the design of SOC estimators in embedded battery management systems.

The main contributions of this study are threefold. First, CNN, LSTM, CNN–LSTM, and TinyCNN are compared under the same leakage-free multi-block holdout strategy and common training protocol. Second, Keras FP32, TFLite FP32, and full-integer TFLite INT8 inference are evaluated separately to distinguish conversion and runtime effects from the additional influence of integer quantization. Third, all four INT8 models are benchmarked on the ESP32 in terms of inference latency, tensor-arena usage, and PC–device output consistency, thereby providing criteria for model selection according to application-specific accuracy and deployment constraints. The conclusions are limited to the present Panasonic 18650PF dataset, the evaluated US06 driving condition, and the current ESP32 deployment configuration.

The remainder of this paper is organized as follows. Section 2 introduces the deep learning models for SOC estimation, including CNN, LSTM, CNN–LSTM, and TinyCNN. Section 3 describes the dataset, data preprocessing, and training settings. Section 4 presents the overall SOC estimation framework, together with the post-training quantization procedure and embedded deployment workflow. Section 5 reports the results and discussion in terms of estimation accuracy, quantization effects, and embedded deployment performance. Section 6 finally concludes the paper and outlines future work.

2. Deep Learning Models for SOC Estimation

To investigate the suitability of deep learning methods for embedded SOC estimation, this study compares several representative neural network architectures with different structural characteristics, including the CNN, LSTM, CNN–LSTM networks, and TinyCNN. In this study, TinyCNN is treated as a compact embedded-oriented CNN configuration rather than a new neural network family. These models were selected because they represent different trade-offs between feature extraction capability, temporal modeling ability, and deployment complexity. The overall goal is not only to compare prediction accuracy, but also to evaluate model compactness, quantization compatibility, and practical feasibility for embedded battery management systems. The four configurations were selected to represent compact convolutional, larger convolutional, recurrent, and hybrid candidates rather than a parameter-matched ablation set.

2.1. CNN-Based Method

One-dimensional CNNs (1D-CNNs) are commonly used for SOC estimation because they can extract local temporal features from battery signals, such as voltage, current, and temperature. Instead of manually designing features, convolutional kernels slide along the time axis and automatically learn useful patterns related to battery dynamics [9,18]. Although individual convolution kernels mainly capture short-term temporal patterns, the effective temporal range can be expanded through stacked convolutional layers and pooling operations. Therefore, deeper CNN structures can capture broader temporal dependencies by increasing the receptive field of higher-level feature representations. A typical CNN structure is shown in Figure 1, where convolutional layers are followed by pooling and fully connected layers. In this study, the implemented CNN follows an embedded-oriented configuration consisting of three convolutional layers with ReLU6 activation, one max-pooling layer, one global average pooling layer, and two fully connected layers. The detailed layer configuration and computational characteristics are summarized in Table 1. Global average pooling aggregates the learned feature maps over the full 200-step input window, providing fixed-window context; however, this operation is not equivalent to the gated, order-sensitive state propagation employed by an LSTM.

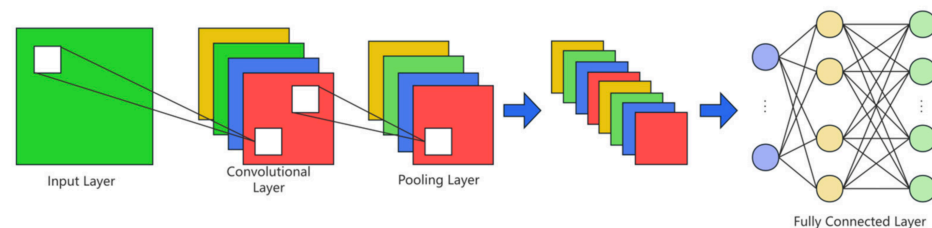


Figure 1. The CNN schematic diagram.

In a typical 1D-CNN, several convolutional filters process the input sequence and generate feature maps. Each filter responds to specific signal variations, and activation functions introduce nonlinearity. The convolution operation can be expressed as:

$$h_k = \sigma \left(\sum_{i=0}^{K-1} w_i \cdot x_{k-i} + b \right), \quad (1)$$

where x_k denotes the input sequence at time step k ; w_i represents the i -th weight of the convolution kernel of size K ; b is the bias term; and $\sigma(\cdot)$ denotes the activation function.

Compared with traditional fully connected architectures, CNNs can improve feature efficiency and reduce sensitivity to local noise. However, their performance depends strongly on network depth and filter configuration, and standard CNN structures may still impose a non-negligible memory burden in embedded scenarios [7,19].

Table 1. Actual architectures and computational complexity of the evaluated models.

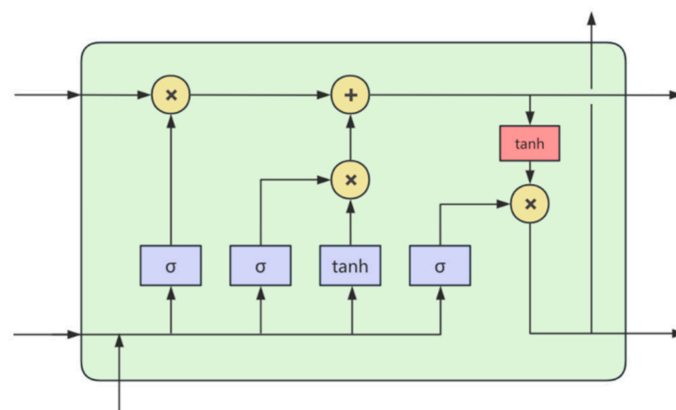
Model	Architecture	Parameters	Receptive Field/Recurrent Sequence Length	MACs	FLOPs
TinyCNN	Input 200×3 ; Conv1D(16, $k = 5$, same) + ReLU; Conv1D(32, $k = 3$, same) + ReLU; GAP; Dense(32) + ReLU; Dense(1) + sigmoid	2913	7	356,256	728,545
CNN	Input 200×3 ; Conv1D(24, $k = 7$, same) + ReLU6; MaxPool(2); Conv1D(32, $k = 5$, same) + ReLU6; Conv1D(64, $k = 3$, same) + ReLU6; GAP; Dense(24) + ReLU6; Dense(1) + sigmoid	12,193	20	1,100,760	2,222,345
LSTM	Input 200×3 ; LSTM(32), tanh/sigmoid; Dense(16) + ReLU; Dense(1) + sigmoid	5153	200	896,528	1,844,273
CNN-LSTM	Input 200×3 ; Conv1D(16, $k = 5$, same) + linear + ReLU; MaxPool(2); LSTM(32); Dense(16) + ReLU; Dense(1) + sigmoid	7073	100	662,928	1,354,673

Note: The fourth column reports the receptive field for TinyCNN and CNN, and the recurrent sequence length for LSTM and CNN-LSTM. For CNN-LSTM, the recurrent sequence length is 100 after max pooling.

2.2. LSTM-Based Method

LSTM networks are widely used in SOC estimation because they can model long-term temporal dependencies in battery behavior. Unlike traditional feedforward networks, LSTM introduces memory cells that store useful past information and filter out short-term noise. This makes LSTM suitable for capturing cumulative charge effects and slow electrochemical dynamics that are difficult to describe with static models [8,20,21].

Figure 2 shows the internal structure of the LSTM cell. The sigmoid function (σ) controls the gating mechanisms, while the tanh function generates candidate cell states. An LSTM unit contains input, forget, and output gates, which control how information is updated and passed through time. With this structure, the network can learn nonlinear relationships among voltage, current, temperature, and SOC while reducing the vanishing gradient problem found in basic recurrent networks. In this study, the implemented LSTM model consists of one LSTM layer with 32 hidden units followed by two fully connected layers. The LSTM layer uses tanh as the main activation function and sigmoid as the recurrent activation function. No dropout, recurrent dropout, Batch Normalization, or L1/L2 regularization is applied. The detailed configuration is summarized in Table 1. Nevertheless, the recurrent structure increases computational complexity and parameter storage requirements, which may limit direct deployment on low-resource microcontrollers [22,23].

**Figure 2.** Structure of an LSTM unit.

2.3. CNN–LSTM Based Method

The CNN–LSTM model combines the strengths of CNN and LSTM. In this architecture, convolutional layers first extract local temporal features from battery signals, and these learned feature maps are then fed into LSTM layers to model longer-term dependencies before SOC prediction [10,24]. A typical CNN–LSTM architecture for SOC estimation is shown in Figure 3. In the implemented CNN–LSTM model, the convolutional layer first transforms the original 200×3 input sequence into a higher-level feature sequence. The convolutional component contains one Conv1D layer with 16 filters and a kernel size of 5, followed by an independent ReLU activation and a valid max-pooling operation. The resulting 100×16 feature sequence is then processed by an LSTM layer with 32 hidden units. The final prediction is generated through fully connected layers with an output sigmoid activation. Detailed architectural information is provided in Table 1.

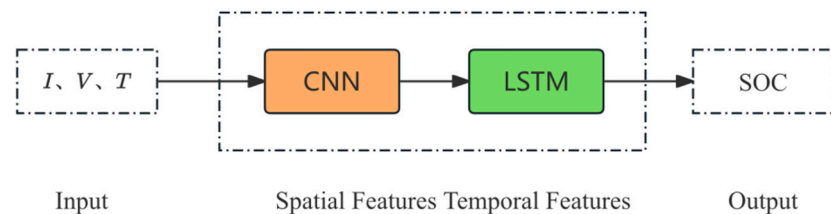


Figure 3. CNN–LSTM architecture for SOC estimation.

Such a hybrid design is expected to provide more comprehensive feature learning than either CNN or LSTM alone. On the one hand, CNN layers can efficiently identify local fluctuation patterns and suppress local noise. On the other hand, LSTM layers can further model temporal evolution across longer horizons. As a result, the CNN–LSTM model can provide enhanced feature representation capability under dynamic driving conditions, although the increased structural complexity may introduce additional deployment challenges.

However, the enhanced feature extraction and temporal modeling capability usually come at the cost of increased parameter count and computational demand, making deployment more challenging in embedded systems.

2.4. TinyCNN-Based Method

Although CNN, LSTM, and CNN–LSTM provide representative benchmarks for SOC estimation, their model complexity may limit practical deployment on embedded battery management platforms with constrained memory and computational resources. To better align the model design with the requirements of embedded applications, TinyCNN is adopted in this work as a compact embedded-oriented CNN configuration with reduced convolutional filters and simplified feature extraction layers. This choice is also consistent with the overall objective of this study, namely, to evaluate not only prediction accuracy but also model compactness and quantization compatibility [14].

Compared with the standard CNN, TinyCNN adopts a more compact structure with fewer convolutional layers and filters, thereby significantly reducing parameter count, model size, and inference cost. Despite its simplified architecture, TinyCNN still preserves the essential ability to extract informative temporal features from battery voltage, current, and temperature measurements, making it a lightweight candidate for SOC estimation on resource-constrained embedded platforms [11–13].

To further reduce deployment overhead, post-training INT8 (8-bit integer) quantization is applied after model training. Through this process, floating-point weights and activations are mapped into 8-bit integer representations, which effectively compresses storage requirements and improves inference efficiency in TensorFlow Lite Micro-based

execution environments implemented using the Espressif esp-tflite-micro package (version 1.3.5) [16,25]. This strategy is particularly suitable for low-power microcontrollers such as the ESP32, where Flash, RAM, and computing capability are all limited [15,17].

Therefore, TinyCNN together with INT8 quantization is evaluated as a lightweight deployment-oriented configuration for exploring the trade-off between estimation performance and deployment cost in embedded SOC estimation. As shown in Section 5, TinyCNN achieves the smallest model size and the lowest measured inference latency among the evaluated models, although its INT8 estimation error is higher than those of the other architectures.

The detailed implementations of the four evaluated models are summarized in Table 1. All models use a 200×3 input sequence consisting of voltage, current, and temperature measurements. The evaluated models do not employ Batch Normalization, Dropout, or L1/L2 regularization unless explicitly specified in the architecture description.

MACs and FLOPs were calculated per inference for a 200×3 input. Multiplication and addition were counted separately in the FLOP calculation; bias additions, global average pooling, and LSTM element-wise state operations were included, whereas the internal costs of nonlinear activation functions were excluded.

3. Data Processing and Model Training

3.1. Data Preprocessing and Sample Construction

The experiments are conducted using the Panasonic 18650PF lithium-ion battery dataset collected at the University of Wisconsin–Madison [26]. The dataset includes multiple operating profiles, among which the US06 driving cycles is selected in this work as representative dynamic condition for both training and evaluation, due to its highly dynamic characteristics. The Panasonic 18650PF dataset is a publicly available battery cycling dataset provided through Mendeley Data. According to the dataset documentation, a new 2.9 Ah Panasonic 18650PF cell (Panasonic Corporation, Kadoma, Osaka, Japan) was tested in an 8 ft³ thermal chamber using a 25 A, 18 V Digatron Firing Circuits Universal Battery Tester channel. The original test program included HPPC, drive-cycle, and impedance-spectroscopy tests at five temperatures; after each test, the cell was charged at 1C to 4.2 V with a 50 mA cutoff when the battery temperature was at least 12 °C [26]. The original battery cycling experiments were performed by the dataset provider, and no additional cell cycling experiments were conducted by the authors in this study. Therefore, the present work focuses on data-driven SOC estimation and embedded deployment evaluation based on the available experimental measurements. The distinction between the public battery-test data source and the modeling and embedded-deployment workflow of the present study is illustrated in Figure 4.

The measurable battery variables used as model inputs are terminal voltage, current, and temperature. These three signals provide complementary information about battery dynamics and are commonly available in practical BMSs. All input signals are sampled at a frequency of 1 Hz. To reduce scale differences and improve training stability, Min–Max normalization is applied to each feature. The normalization parameters are fitted only using the final training raw samples after temporal partitioning. The same normalization coefficients are then applied to the validation set, test set, and ESP32 inference process to ensure consistency between offline evaluation and embedded deployment.

To improve robustness against potential distribution variations, the normalized features are clipped to the range of $[-0.2, 1.2]$, instead of being strictly limited to $[0, 1]$. After the raw time-series partitioning described in Section 3.2, supervised samples are constructed independently within each valid subset. A sliding-window strategy is adopted to construct supervised learning samples from the time-series data. Each input sample

consists of 200 consecutive time steps, and the SOC value at the final time step is used as the regression target. The window is shifted by one sample each time, generating overlapping sequences and improving data utilization [20].

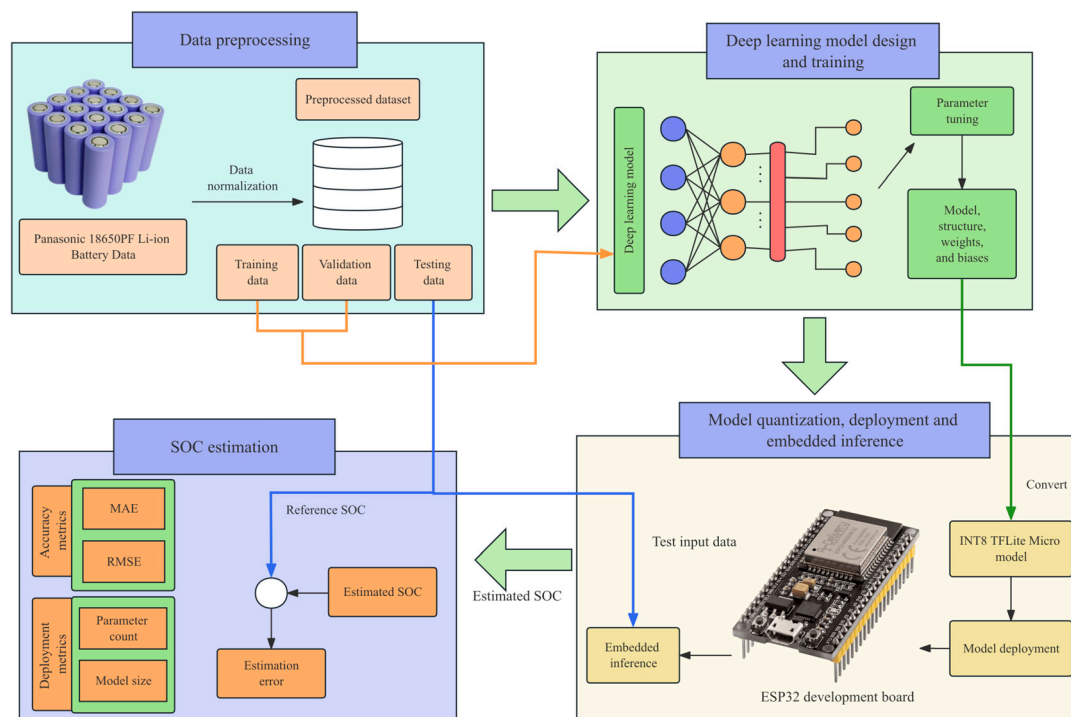


Figure 4. Overall workflow of the proposed SOC estimation framework.

3.2. Data Partition and Training Settings

To prevent temporal information leakage caused by overlapping sliding windows, a leakage-free multi-block holdout strategy is adopted before sample construction. Specifically, the original time-series data are first divided into training, validation, testing, and guard regions at the raw-sample level. Five test core blocks, each containing 3000 raw samples, are selected from the US06 sequence, whereas the validation subset is retained as one continuous core containing 3106 raw samples. A guard interval of 200 raw samples, equal to the adopted window length, is placed on each side of every core boundary where adjacent data exist. After the raw-data partitioning, sliding windows are generated independently within each valid region. Any window crossing a core block, guard interval, subset boundary, or source-segment boundary is discarded to ensure temporal separation among different datasets.

The final partition contains 27,555 training raw samples, 3106 validation raw samples, 15,000 testing raw samples, and 2400 guard samples. Excluding the guard samples, the training, validation, and testing raw-sample proportions are 60.35%, 6.80%, and 32.85%, respectively. After window construction and boundary filtering, 26,162 training windows, 2907 validation windows, and 14,005 testing windows are obtained, corresponding to 60.74%, 6.75%, and 32.51% of the final windows, respectively. The shared raw-sample count between train-validation, train-test, and validation-test subsets is zero, confirming that no original time points are reused across evaluation subsets (Table 2).

All models use the same split manifest and training protocol to ensure a fair comparison. The Adam optimizer with a learning rate of 0.001 is adopted. The Huber loss function is used for all models with a unified δ value of 0.02. The value of δ was selected by comparing $\delta = 0.01$ and $\delta = 0.02$ according to the average best validation MAE of the four

models, while the test set was not involved in this selection process. All four models were trained for 120 epochs with a batch size of 64 using the same training protocol.

Table 2. Leakage-free dataset partition statistics.

Subset	Raw Samples	Windows
Training	27,555	26,162
Validation	3106	2907
Testing	15,000	14,005
Guard	2400	-

Note: “-” indicates that no windows were generated from the guard samples, as they were excluded to prevent data leakage between subsets.

4. Quantization and Embedded Deployment

After floating-point training, post-training quantization is applied to convert the trained models into 8-bit integer TensorFlow Lite format for embedded deployment. A representative dataset containing 2000 input windows is randomly selected only from the training set for calibration. The calibration process uses a fixed random seed of 42, and identical representative-window indices are applied to all four models. Validation and testing samples are not used during INT8 calibration.

In this study, three evaluation formats are distinguished: Keras FP32 represents the original floating-point model before conversion; TFLite FP32 represents the converted TensorFlow Lite floating-point model; and TFLite INT8 represents the fully quantized deployment model. Therefore, the difference between Keras FP32 and TFLite INT8 reflects the overall deployment conversion effect, while the difference between TFLite FP32 and TFLite INT8 more directly indicates the influence of integer quantization.

To ensure consistency between offline evaluation and embedded inference, the same preprocessing pipeline is applied throughout the workflow. Input feature windows are first normalized and clipped, then quantized on the host side before being transmitted to the embedded device. The ESP32 performs inference using quantized inputs, and the outputs are subsequently dequantized for comparison [12,14,15]. Before dequantization, the raw INT8 outputs generated by the PC-side TensorFlow Lite inference and the ESP32 inference are directly compared to verify implementation consistency.

The quantized models are deployed using the TensorFlow Lite Micro framework on an ESP32-WROOM-32 module (Espressif Systems (Shanghai) Co., Ltd., Shanghai, China), hereafter referred to as the ESP32, operating at 160 MHz. A static tensor arena of 163,840 bytes (160 KiB) is allocated for intermediate tensors and runtime buffers during inference [11,13].

For embedded benchmarking, each model was first executed for 20 warm-up runs, followed by 200 measured inference runs. The reported metrics include mean inference latency, effective inference frequency, tensor-arena usage, and PC–device output consistency.

The reported latency represents only the dedicated TensorFlow Lite Micro inference call. It does not include sensor acquisition, preprocessing, communication, task scheduling, SOC post-processing, or the complete BMS control loop. Power consumption and system-level CPU utilization were not measured in this study and therefore are not inferred from inference latency.

The overall workflow of the proposed SOC estimation framework is illustrated in Figure 4, which consists of data preprocessing, model design and training, quantization, and embedded deployment on the ESP32 platform. First, the lithium-ion battery dataset is filtered, normalized, and divided into training, validation, and testing subsets. The training and validation data are used for deep learning model design and training. The trained model is then quantized and converted for ESP32 deployment. After that, the testing data are input to the ESP32 development board for embedded inference, and the estimated SOC

is generated on-device. Finally, the estimated SOC is compared with the reference SOC from the test set, and the model is evaluated in terms of the mean absolute error (MAE), root mean square error (RMSE), parameter count, and quantized model size.

5. Results and Discussion

Model performance is evaluated from both algorithmic and embedded perspectives. Estimation accuracy is assessed using MAE and RMSE. These metrics are calculated on the leakage-free test windows generated from the predefined test blocks, with a window stride of 1 during sample construction.

Embedded efficiency is then evaluated in terms of model complexity and memory footprint, reflecting the feasibility of real-time SOC estimation on resource-constrained platforms. In addition, ESP32 inference latency and implementation consistency are evaluated to further characterize embedded deployment performance. This dual-perspective evaluation allows the study to identify models that are not only accurate but also practical for embedded BMS applications.

5.1. Accuracy Under a Driving Condition

The experiments evaluate model performance under the US06 driving condition using the leakage-free test blocks defined in Section 3.2. Figure 5 presents the SOC estimation results of different models under the US06 condition. The results indicate that all models are able to follow the overall SOC trajectory, with only small deviations observed in regions characterized by more rapid dynamic variations.

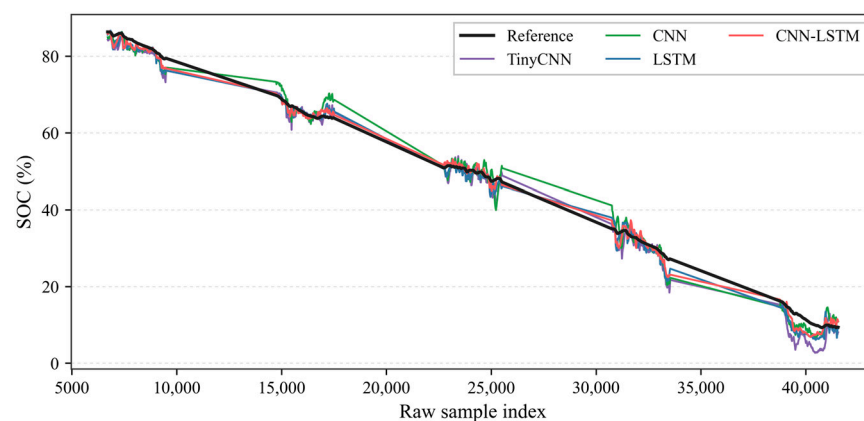


Figure 5. FP32 SOC estimation results of the four evaluated models under the US06 test condition.

Figure 6 presents the SOC estimation results of the four models after full-integer INT8 quantization under the same US06 test condition. Among them, LSTM achieves the lowest INT8 estimation error, with an MAE of 1.70% and an RMSE of 2.14%. CNN-LSTM follows with an MAE of 1.87%, while TinyCNN exhibits the largest INT8 error among the evaluated models, with an MAE of 2.55%. Although TinyCNN has higher estimation errors than the other models, its compact architecture provides advantages for embedded deployment, which will be further discussed in Section 5.3.

Under full-integer INT8 inference, the four models exhibit different estimation-error levels. Among them, LSTM achieves the highest accuracy, with the lowest MAE and RMSE values of 1.70% and 2.14%, respectively. CNN-LSTM follows, with an MAE of 1.87% and an RMSE of 2.37%, whereas the conventional CNN exhibits slightly inferior performance, with an MAE of 2.00% and an RMSE of 2.49%. By comparison, TinyCNN produces the largest estimation deviation, with an MAE of 2.55% and an RMSE of 3.31%.

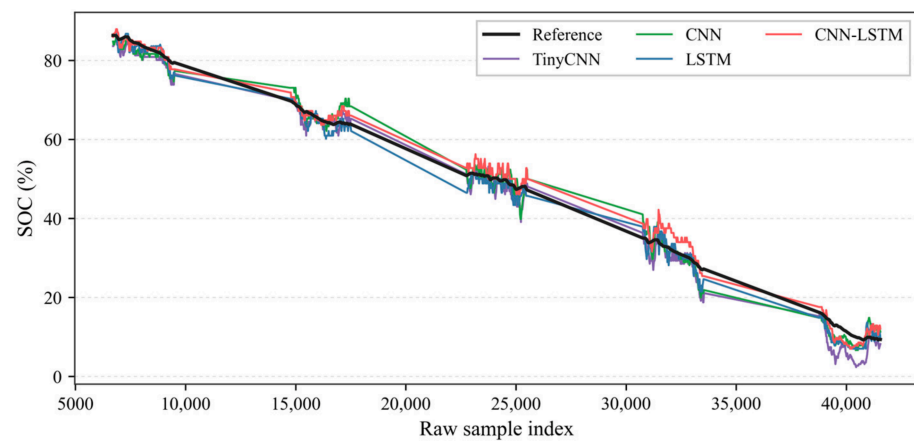


Figure 6. Full-integer INT8 SOC estimation results of the four evaluated models under the US06 test condition.

5.2. Impact of INT8 Quantization

The quantization analysis evaluates the impact of low-precision deployment on model performance under the US06 condition. To separate model-conversion and runtime effects from the additional influence of integer quantization, three inference stages are considered: Keras FP32, TFLite FP32, and full-integer TFLite INT8. The difference between Keras FP32 and TFLite FP32 represents conversion and runtime implementation effects, whereas the difference between TFLite FP32 and TFLite INT8 more directly reflects the influence of integer quantization. The difference between Keras FP32 and TFLite INT8 is therefore treated as the end-to-end deployment difference. Table 3 summarizes the stage-wise MAE values and output differences, while Figure 7 compares the Keras FP32 and TFLite INT8 MAE and RMSE values.

Table 3. Comparison of Keras FP32, TFLite FP32, and full-integer TFLite INT8 inference.

Model	Keras FP32 MAE (%)	TFLite FP32 MAE (%)	TFLite INT8 MAE (%)	Keras–TFLite FP32 Mean Absolute Output Difference (SOC Fraction)	TFLite FP32–INT8 Mean Absolute Output Difference (SOC Fraction)
TinyCNN	2.3339	2.3339	2.5527	4×10^{-8}	0.00490136
CNN	1.9693	1.9693	2.0032	3×10^{-8}	0.00332533
LSTM	1.6329	1.7020	1.7040	0.00908163	0.00359048
CNN-LSTM	1.3053	1.5683	1.8661	0.00956251	0.00660348

As shown in Figure 7, all four models exhibit higher end-to-end deployment errors in TFLite INT8 than in Keras FP32, although the magnitude of the increase differs among the architectures. From Keras FP32 to TFLite INT8, the MAE increases by 0.2188, 0.0339, 0.0711, and 0.5608 percentage points for TinyCNN, CNN, LSTM, and CNN-LSTM, respectively. The corresponding RMSE increases are 0.1763, 0.0363, 0.0490, and 0.6697 percentage points. CNN-LSTM therefore exhibits the largest end-to-end increase among the four evaluated models.

However, the end-to-end increase observed for CNN-LSTM should not be attributed entirely to integer quantization. As shown in Table 3, its MAE increases from 1.31% in Keras FP32 to 1.57% in TFLite FP32, corresponding to a conversion/runtime difference of 0.2630 percentage points. Its MAE then increases further to 1.87% after full-integer quantization, corresponding to an additional TFLite FP32-to-INT8 difference of 0.2978 percentage points. A similar distinction is observed for LSTM, whose MAE changes from 1.63% to 1.70% during Keras-to-TFLite FP32 conversion and then changes only slightly to 1.70% after INT8

quantization. By contrast, the Keras FP32 and TFLite FP32 MAE values of TinyCNN and CNN are numerically unchanged at the reported precision, indicating that their observed end-to-end increases arise mainly during the TFLite FP32-to-INT8 transition.

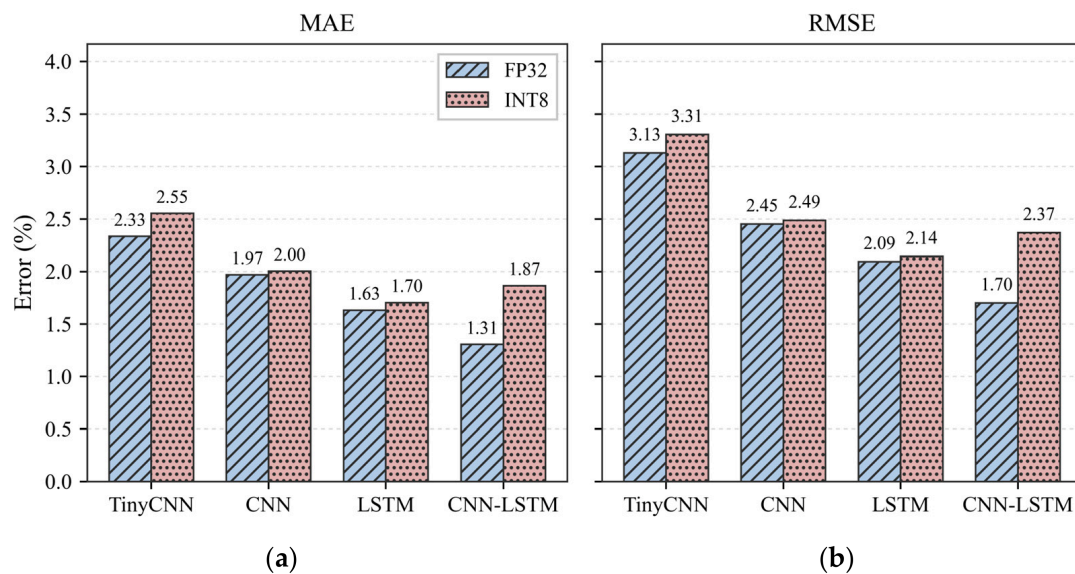


Figure 7. Comparison of estimation errors between FP32 and INT8-quantized models: (a) MAE and (b) RMSE.

The model-dependent differences may be associated with recurrent-state propagation, activation dynamic ranges, quantization scales, and differences in TensorFlow Lite operator implementation. In particular, the nonzero Keras-to-TFLite FP32 differences observed for LSTM and CNN-LSTM indicate that conversion and runtime behavior contributes to their end-to-end deployment differences before integer quantization is applied. Nevertheless, the present experiments do not isolate a single causal mechanism, and these factors should therefore be regarded as plausible explanations rather than confirmed causes.

Overall, the end-to-end difference from Keras FP32 to TFLite INT8 is architecture-dependent and should not be attributed entirely to integer quantization, because part of the change for LSTM and CNN-LSTM already appears after conversion to TFLite FP32. Among the four models, LSTM demonstrates the strongest robustness after quantization and still achieves the lowest estimation error. These results indicate that the effect of INT8 deployment should be evaluated separately for each architecture.

5.3. Computational Cost and Embedded Feasibility

The feasibility of embedded deployment is further analyzed from the perspectives of model complexity and memory footprint after INT8 quantization. Figure 8 compares the parameter counts and quantized model sizes of the four models. As can be observed from Figure 8a, TinyCNN contains only 2913 parameters, which is substantially fewer than those of LSTM (5153), CNN-LSTM (7073), and CNN (12,193). This reduction in parameter count directly translates into a smaller model size after quantization.

In terms of memory consumption, as shown in Figure 8b, TinyCNN achieves the smallest INT8 model size, at only 8.51 KiB, whereas LSTM and CNN-LSTM require 10.23 KiB and 14.45 KiB, respectively. The standard CNN exhibits the largest memory footprint, reaching 21.27 KiB.

It is worth noting that these differences are closely related to the deployment-oriented architectural design of the models. The standard CNN has a higher parameter count than CNN-LSTM mainly because its three convolutional layers use larger filter banks and

channel dimensions. The final dense layers contribute additional parameters, whereas most parameters in CNN–LSTM are concentrated in the recurrent layer. The different deployment-error changes observed for LSTM and CNN–LSTM cannot be attributed solely to architectural complexity, because conversion and runtime differences are also present before full-integer quantization, as discussed in Section 5.2.

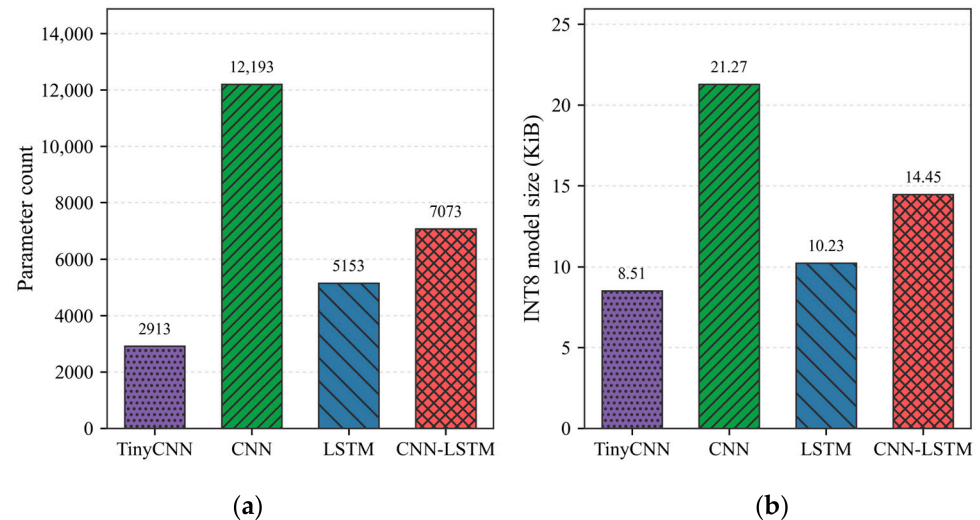


Figure 8. Comparison of model complexity and deployment cost: (a) parameter count and (b) INT8 quantized model size.

For resource-constrained microcontroller platforms such as the ESP32, these differences are of clear practical significance. Models with larger parameter counts generally require more Flash storage and runtime memory, and may also incur higher inference latency, thereby constraining real-time performance. In contrast, TinyCNN has the smallest parameter count and INT8 model size among the evaluated models and also achieves the lowest measured ESP32 inference latency. However, its INT8 estimation error is higher than those of the other evaluated models, and it should therefore be regarded as a resource-efficient candidate rather than the universally best model. CNN–LSTM is preferred when Keras FP32 estimation accuracy is the primary criterion, whereas LSTM is preferred when full-integer INT8 estimation accuracy is prioritized. The final model choice should depend on the allowable SOC error, available Flash and RAM, required update period, and the operating requirements of the target BMS.

To further evaluate actual embedded performance, the four full-integer INT8 models were benchmarked on an ESP32 operating at 160 MHz. Table 4 summarizes the measured mean inference latency, effective inference frequency, tensor-arena usage, and PC–ESP32 output consistency.

Table 4. ESP32-WROOM-32 inference benchmark results for the full-integer INT8 models.

Model	CPU (MHz)	Arena Configured (Bytes)	Arena Used (Bytes)	Mean Latency (ms)	Frequency (Hz)	PC/ESP32 Match
TinyCNN	160	163,840	14,412	41.992	23.814	200/200
CNN	160	163,840	15,436	99.169	10.084	200/200
LSTM	160	163,840	9164	148.151	6.750	200/200
CNN-LSTM	160	163,840	8972	111.443	8.973	200/200

As shown in Table 4 and Figure 9, TinyCNN achieves the lowest measured mean inference latency of 41.992 ms and the highest effective inference frequency of 23.814 Hz. CNN requires 99.169 ms per inference, followed by CNN-LSTM at 111.443 ms, while LSTM exhibits the highest mean latency of 148.151 ms.

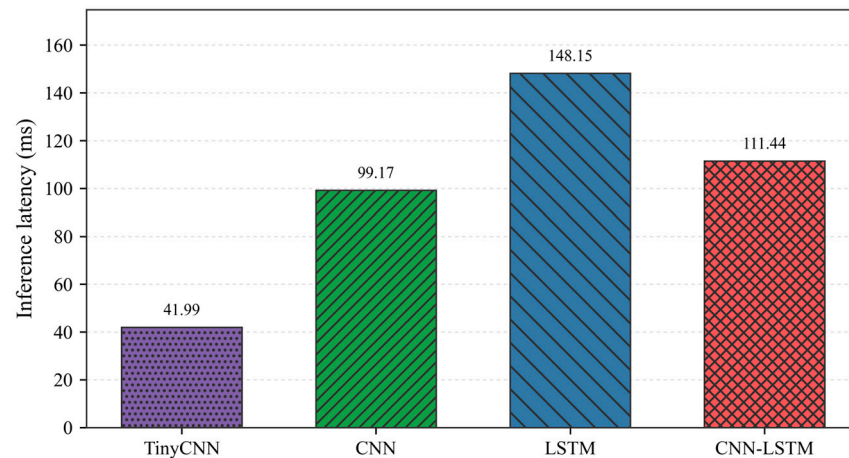


Figure 9. Mean inference latency of the four INT8 models on the ESP32-WROOM-32.

The tensor-arena results do not follow the same ordering as the parameter count or model-file size. CNN-LSTM uses the smallest measured tensor-arena allocation of 8972 bytes, followed by LSTM at 9164 bytes, TinyCNN at 14,412 bytes, and CNN at 15,436 bytes. Therefore, the smallest model file does not necessarily imply the lowest runtime tensor-arena requirement. All four models achieve identical raw INT8 outputs between the PC implementation and the ESP32 for all 200 evaluated samples, confirming implementation consistency under the tested inputs.

The reported latency covers the TensorFlow Lite Micro inference call and should not be interpreted as the execution time of a complete BMS control cycle. Sensor acquisition, input preprocessing, communication, task scheduling, and SOC post-processing are not included. CPU utilization and power consumption were not measured and are therefore not inferred from the latency results.

Because the overall MAE does not reveal whether the estimation error is uniformly distributed across the SOC range, Figure 10 further compares the INT8 MAE in the 0–20%, 20–80%, and 80–100% SOC intervals.

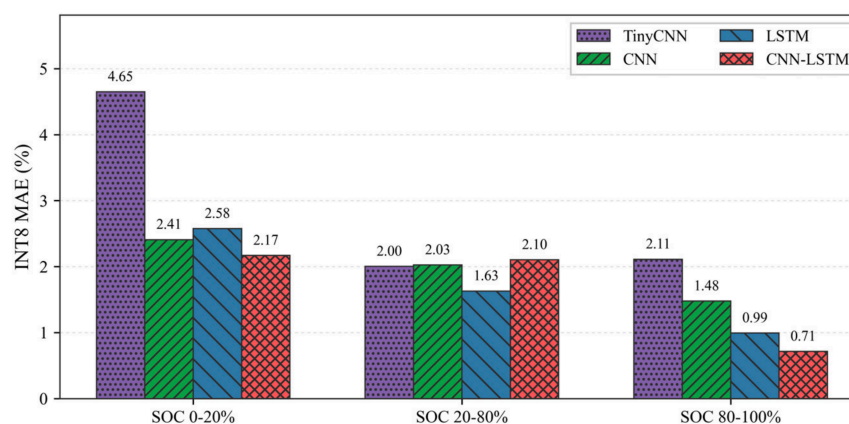


Figure 10. INT8 MAE of the four evaluated models in the 0–20%, 20–80%, and 80–100% SOC intervals.

The results show that estimation difficulty varies across the SOC range. TinyCNN exhibits an INT8 MAE of 4.65% in the 0–20% SOC interval, compared with 2.00% in the

20–80% interval and 2.11% in the 80–100% interval. The corresponding low-SOC errors of CNN, LSTM, and CNN–LSTM are 2.41%, 2.58%, and 2.17%, respectively. In the 80–100% interval, CNN–LSTM achieves the lowest MAE of 0.71%, followed by LSTM at 0.99%.

These results indicate that the practical suitability of a model cannot be determined from its overall MAE alone. In particular, the higher TinyCNN error in the low-SOC interval may be important for applications that require reliable estimation near the lower charge boundary. Therefore, whether its error level is acceptable depends on the allowable SOC error, safety margin, operating SOC range, and calibration requirements of the target BMS. The present results support TinyCNN as a compact and fast candidate, but they do not establish compliance with a universal commercial BMS accuracy requirement.

To further position the present work within recent SOC estimation studies, Table 5 compares representative studies using the same Panasonic 18650PF dataset or related embedded deployment settings.

Table 5. Comparison with representative SOC estimation studies.

Study	Dataset and Test Conditions	Method	Reported Performance	Embedded Evaluation
Ref. [27]	Panasonic 18650PF; −20 to 25 °C; Cycles 1–4 testing	Residual CNN	Mean MAE: 0.998%; RMSE: 1.260%	No; 2.24×10^6 FLOPs
Ref. [22]	Panasonic 18650PF; 25 °C; eight test cycles	Parallel LSTM	Best MAE: 0.75%	No; laptop-based evaluation
Ref. [17]	Self-collected AVRFB and ZBRFB datasets	Quantized LSTM	$R^2 = 0.975$ for AVRFB	Yes; ESP32, 1.72 s inference
This study	Panasonic 18650PF; US06; leakage-free block-based testing	CNN, LSTM, CNN–LSTM, and TinyCNN with INT8 quantization	Best INT8 MAE: 1.704%	Yes; measured latency and tensor-arena usage on ESP32

As shown in Table 5, the lower errors reported by previous same-dataset studies were obtained using different data partitions, operating conditions, and predominantly floating-point models evaluated on desktop- or GPU-class platforms. Ozer and Arabaci, for example, reported MAEs ranging from 0.75% to 2.13% across different test cycles. Yadav et al. demonstrated quantized inference on an ESP32, but their study considered redox-flow batteries and required 1.72 s per inference. In contrast, the present study evaluates four full-integer INT8 architectures under a leakage-free protocol and provides direct ESP32 latency and tensor-arena measurements.

6. Conclusions

To address the challenge of deploying deep learning-based SOC estimation on resource-constrained embedded battery management systems, this paper presents a comparative study of CNN, LSTM, CNN–LSTM, and lightweight TinyCNN models under a unified embedded-oriented framework. By integrating post-training INT8 quantization with TensorFlow Lite Micro deployment on an ESP32 microcontroller, the proposed framework enables a systematic evaluation of estimation accuracy, quantization robustness, and deployment feasibility. Experimental results under the US06 driving condition show that model performance depends on the inference format and deployment criterion. CNN–LSTM achieves the lowest Keras FP32 error, with an MAE of 1.31% and an RMSE of 1.70%, whereas LSTM achieves the lowest full-integer INT8 error, with an MAE of 1.70% and an RMSE of 2.14%. TinyCNN does not provide the highest estimation accuracy; however, its 2913 parameters, 8.51 KiB INT8 model size, and mean ESP32 inference latency of 41.992 ms

make it a compact and fast candidate for resource-constrained deployment. No single model is optimal across all evaluated metrics, and model selection should therefore consider the allowable estimation error, model size, tensor-arena usage, and required inference latency of the target BMS. The end-to-end deployment-error changes are architecture-dependent and should not be attributed entirely to integer quantization, because part of the change for LSTM and CNN–LSTM already occurs during conversion to TFLite FP32. The ESP32 results demonstrate deployment feasibility under the tested configuration; however, the reported latency covers only the TensorFlow Lite Micro inference call and does not establish the real-time performance of a complete BMS control loop. These conclusions are limited to the present dataset, the evaluated US06 driving condition, the selected model configurations, and the current ESP32 implementation. Future work will extend the evaluation to additional driving conditions, temperatures, battery aging stages, and battery types, while investigating quantization-aware training, model pruning, knowledge distillation, parameter-matched baselines, and longer-term validation in a complete BMS.

Author Contributions: Conceptualization, methodology, validation, Y.L., X.L. and X.T.; formal analysis, resources, W.S. and J.W.; writing—original draft preparation, Y.L.; writing—review and editing, supervision, funding acquisition, L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Shenzhen Science and Technology Program (JCYJ20250604145036046) and the Natural Science Foundation of Top Talent of SZTU (GDRC202405).

Data Availability Statement: The Panasonic 18650PF battery dataset used in this study is publicly available through Mendeley Data, as cited in Ref. [26]. No new battery-cycling data were generated in this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SOC	State of charge
BMS	Battery management system
CNN	Convolutional neural network
LSTM	Long short-term memory network
RAM	Random access memory
MAE	Mean absolute error
RMSE	Root mean square error
INT8	8-bit integer
FP32	single-precision floating-point

References

1. Hasan, M.; Haque, R.; Jahirul, M.; Rasul, M.; Fattah, I.; Hassan, N.; Mofijur, M. Advancing energy storage: The future trajectory of lithium-ion battery technologies. *J. Energy Storage* **2025**, *120*, 116511.
2. Wang, C.; Yang, M.; Wang, X.; Xiong, Z.; Qian, F.; Deng, C.; Yu, C.; Zhang, Z.; Guo, X. A review of battery SOC estimation based on equivalent circuit models. *J. Energy Storage* **2025**, *110*, 115346. [[CrossRef](#)]
3. Tripp-Barba, C.; Aguilar-Calderón, J.A.; Urquiza-Aguilar, L.; Zaldívar-Colado, A.; Ramírez-Noriega, A. A systematic mapping study on state estimation techniques for lithium-ion batteries in electric vehicles. *World Electr. Veh. J.* **2025**, *16*, 57. [[CrossRef](#)]
4. Pisani Orta, M.A.; García Elvira, D.; Valderrama Blaví, H. Review of state-of-charge estimation methods for electric vehicle applications. *World Electr. Veh. J.* **2025**, *16*, 87. [[CrossRef](#)]
5. Xie, Y.; Han, Z.; Zhang, J.; Liu, L. A physics-constrained Informer-LSTM network for battery state-of-charge estimation. *J. Energy Storage* **2025**, *136*, 118381. [[CrossRef](#)]
6. Takyi-Aninakwa, P.; Wang, S.; Liu, G.; Fernandez, C.; Kang, W.; Song, Y. Deep learning framework designed for high-performance lithium-ion batteries state monitoring. *Renew. Sustain. Energy Rev.* **2025**, *218*, 115803. [[CrossRef](#)]

7. Guo, S.; Ma, L. A comparative study of different deep learning algorithms for lithium-ion batteries on state-of-charge estimation. *Energy* **2023**, *263*, 125872. [[CrossRef](#)]
8. Wang, S.; Takyi-Aninakwa, P.; Jin, S.; Yu, C.; Fernandez, C.; Stroe, D.-I. An improved feedforward-long short-term memory modeling method for the whole-life-cycle state of charge prediction of lithium-ion batteries considering current-voltage-temperature variation. *Energy* **2022**, *254*, 124224. [[CrossRef](#)]
9. Hannan, M.A.; How, D.N.T.; Lipu, M.S.H.; Ker, P.J.; Dong, Z.Y.; Mansur, M.; Blaabjerg, F. SOC estimation of Li-ion batteries with learning rate-optimized deep fully convolutional network. *IEEE Trans. Power Electron.* **2021**, *36*, 7349–7353. [[CrossRef](#)]
10. Gezer, U.; Küçük, M.; Yazar, O.; Coskun, S.; Zhang, F. A hybrid CNN-LSTM framework for state of charge estimation of lithium-ion batteries. In *Proceedings of the 2025 9th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*; IEEE: Piscataway, NJ, USA, 2025.
11. Liu, H.-I.; Galindo, M.; Xie, H.; Wong, L.-K.; Shuai, H.-H.; Li, Y.-H.; Cheng, W.-H. Lightweight deep learning for resource-constrained environments: A survey. *ACM Comput. Surv.* **2024**, *56*, 267. [[CrossRef](#)]
12. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [[CrossRef](#)]
13. 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](#)]
14. 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](#)]
15. Dvsr, S.; Badachi, C.; Nagawaram, C.; Kondoju, P.C.; Dhanamjayulu, C.; Kamwa, I. State of charge estimation for Li-ion batteries: An edge-based data-driven approach. *IEEE Access* **2025**, *13*, 106703–106723. [[CrossRef](#)]
16. 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](#)]
17. Yadav, A.K.; Kumar, R.; Saini, A.K.; Ragupathy, P.; Ranjan, A.; Abhishek, A.; Dhiraj. Edge computing enabled battery state of charge (SoC) estimation using TinyML techniques. In *Proceedings of the 2024 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*; IEEE: Piscataway, NJ, USA, 2024.
18. Wang, S.; Ren, P.; Takyi-Aninakwa, P.; Jin, S.; Fernandez, C. A critical review of improved deep convolutional neural network for multi-timescale state prediction of lithium-ion batteries. *Energies* **2022**, *15*, 5053. [[CrossRef](#)]
19. Bhattacharyya, H.S.; Yadav, A.; Choudhury, A.B.; Chanda, C.K. Convolution neural network-based SOC estimation of Li-ion battery in EV applications. In *Proceedings of the 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*; IEEE: Piscataway, NJ, USA, 2021.
20. Ma, L.; Hu, C.; Cheng, F. State of charge and state of energy estimation for lithium-ion batteries based on a long short-term memory neural network. *J. Energy Storage* **2021**, *37*, 102440. [[CrossRef](#)]
21. Eleftheriadis, P.; Leva, S.; Ogliari, E. Bayesian hyperparameter optimization of stacked bidirectional long short-term memory neural network for the state of charge estimation. *Sustain. Energy Grids Netw.* **2023**, *36*, 101160. [[CrossRef](#)]
22. Ozer, O.; Arabaci, H. State of charge estimation in Li-ion batteries using a parallel LSTM-based approach: The impact of modelling based on operating states. *IEEE Access* **2025**, *13*, 130719–130730. [[CrossRef](#)]
23. Yang, B.; Wang, Y.; Zhan, Y. Lithium battery state-of-charge estimation based on a Bayesian optimization bidirectional long short-term memory neural network. *Energies* **2022**, *15*, 4670. [[CrossRef](#)]
24. Ardeshiri, R.R.; Ma, C. State of charge estimation of lithium-ion battery using deep convolutional stacked bidirectional LSTM. In *Proceedings of the 2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*; IEEE: Piscataway, NJ, USA, 2021.
25. Nagel, M.; Fournarakis, M.; Amjad, R.A.; Bondarenko, Y.; Van Baalen, M.; Blankevoort, T. A white paper on neural network quantization. *arXiv* **2021**, arXiv:2106.08295.
26. Kollmeyer, P. *Panasonic 18650PF Li-ion Battery Data*. Mendeley Data 2018, V1; University of Wisconsin Madison: Madison, WI, USA, 2018. [[CrossRef](#)]
27. 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](#)] [[PubMed](#)]

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.