Hyperparameter-Optimized Quantization-Aware training-based machine learning model compression for microcontrollers

1st Minh Chanh Vo
Computer Science and Computer Engineering
Ho Chi Minh University of Technology
Ho Chi Minh City, VietNam
vmchanh.sdh242@hcmut.edu.vn
0009-0005-1651-8976

2nd Trong Nhan Le
Computer Science and Computer Engineering
Ho Chi Minh University of Technology
Ho Chi Minh City, VietNam
trongnhanle@hcmut.edu.vn
0000-0001-7343-5118

***Abstract*—This paper proposes a lightweight Long Short-Term Memory (LSTM) model using on the Quantization-Aware Training (QAT) method that can operate with high accuracy on the ESP32S3 microcontroller. The study has used Hyperparameter Optimization (HPO) combined with L2 penalty term to find the most optimal parameter set on each dataset and then applied the QAT technique to compress the model while maintaining the highest accuracy. The model achieved an accuracy of 94% on the raw inertial signal data from the Human Activity Recognition (HAR) dataset using only an LSTM layer. The quantized model also achieved 93.79% accuracy and took advantage of Single Instruction, Multiple Data (SIMD) architecture on ESP32S3 to achieve an inference time of 120ms. This work provides a practical foundation for enabling efficient on-device artificial intelligence applications in low-power and resource-constrained devices.**

***Index Terms*—LSTM, Quantization Aware Training, Hyperparameter Optimization, L2 penalty, ESP32S3, HAR dataset.**

I. Introduction

In recent years, machine learning models have achieved remarkable success across various application domains. However, deploying these models on microcontrollers remains a significant challenge. Unlike computers or microprocessors, which possess abundant computational power and memory resources, microcontrollers operate under strict constraints in terms of memory capacity, energy consumption, and processing power. These limitations directly affect the feasibility and performance of conventional machine learning models on such devices. Consequently, model compression techniques have become an important research focus, aiming to enable the deployment of machine learning models on resource-constrained embedded devices.

Among various model compression methods, quantization is highlighted as an effective approach for reducing model size and computational cost [1]. Although Post-Training Quantization (PTQ) provides a simple and convenient solution, it often leads to a degradation in accuracy, especially for small models or sensitive applications. In contrast, QAT incorporates the effects of quantization directly into the training process, allowing the model to adapt to low-precision weights while maintaining high accuracy.

As a result, quantization is particularly well suited for deploying deep learning models on microcontrollers, where efficiency under strict resource constraints is critical. This paper explores the use of QAT to compress machine learning models for microcontrollers, with a focus on balancing memory consumption, inference latency, and accuracy. By applying QAT, high accuracy can be achieved without a significant increase in inference time, opening new opportunities for real-time applications in embedded systems. A comprehensive survey on Tiny Machine Learning (TinyML) [2] provides an overview of its fundamental concepts, challenges, and application scope. The survey highlights the necessity of model compression techniques and hardware optimization to enable efficient inference on microcontrollers and microprocessors. It further discusses strategies for balancing accuracy, memory footprint, and inference latency on edge devices.

In another study, a survey and a set of enhancements for deploying LSTM-based recurrent neural networks on embedded systems were presented [3]. The work identifies key challenges in applying machine learning on microcontrollers, including limitations in memory, energy consumption, and computational resources. To address these constraints, hybrid architectures such as LSTM–CNN models have been proposed for human activity recognition [4]. These models are capable of extracting spatial and temporal features separately while reducing overall model complexity through techniques such as global average pooling and batch normalization. Experimental results on public datasets demonstrate that this approach achieves high accuracy on the UCI-HAR, WISDM, and OPPORTUNITY datasets with fewer parameters compared to traditional feature-based methods.

At the hardware optimization level, structured compression techniques on FPGA [5] reached successful, the proposed method reduces the computational complexity and resource requirements of the LSTM model while maintaining high accuracy. This paper has introduced structured compression

technique on FPGAs. All of methods has proved ability of control model architecture to improve accuracy and inference time.

Overall, current methods has push TinyML to 2 ways:

-Firstly, change the architecture and compressing of model to improve accuracy in reallife such as recognize human activity and sign language, predicting anomalies in mechanics.

-Second, Optimizing hardware to permit implementation efficiently on resource-constrained platform, such as FPGA.

A. Existing Study

A recent project investigated temperature prediction using historical weather data collected in Lo Barnechea, Chile, covering the period from 2010 to 2024 [6]. The study employed a single LSTM network to model temporal dependencies in the data. The trained model was converted using TensorFlow lite and deployed on an ESP32-S3 microcontroller through the Edge Impulse SDK. The results demonstrate the feasibility of applying machine learning techniques on resource-constrained hardware, particularly microcontrollers.

The feasibility of deploying machine learning models on embedded devices has gained increasing attention due to continuous improvements in microcontroller performance [7]. TensorFlow lite for microcontrollers (TFLM) provides a lightweight machine learning framework tailored for resource-constrained platforms. However, its limited support for tensor operations and optimized kernels restricts the efficient deployment of recurrent architectures such as LSTM.

Furthermore, several studies have demonstrated the deployment of gesture recognition and keyword spotting models on microcontrollers, particularly the ESP32-S3 [8]. These works show that deep learning models can achieve real-time performance under strict hardware constraints.

B. Contribution

Machine learning models are widely applied on processors, but their deployment on microcontrollers remains inefficient in terms of both model optimization and prediction accuracy. Previous approaches often relied on PTQ for compression, which limits the model's ability to learn dataset-specific features. In this study, we propose a novel compression method for LSTM models tailored to the ESP32-S3 microcontroller by employing quantization-aware training (QAT) combined with Hyperband-based hyperparameter optimization and L2 penalty to maximize prediction accuracy. The resulting lightweight LSTM model achieves a size reduction to one-fourth of the original while maintaining 93.55% accuracy on the HAR dataset—nearly equivalent to the uncompressed model. This approach enables efficient deployment on the ESP32-S3 by leveraging SIMD and FPU architectures, achieving an inference time of 120 ms with a 32-cell LSTM and an input sequence length of 128.

II. Background Knowledge

A. Tiny Machine Learning

TinyML refers to the deployment of machine learning models on highly resource-constrained devices, typically microcontrollers with limited memory, computational power, and energy capacity. Unlike traditional machine learning systems that rely on powerful GPUs or cloud servers, TinyML enables real-time inference directly on embedded devices with a power budget often below than 1 mW. This capability makes it particularly suitable for applications such as wearable devices, smart home automation, industrial monitoring, and edge computing scenarios where low latency and data privacy are critical.

B. Long Short Term Memory

Long Short Term Memory model is designed to memorize long-term information, overcoming the weakness of RNN model, LSTM model adds a memory cell along with three gate control mechanisms including input gate, forget gate and output gate to help solve the vanishing gradient problem of RNN.

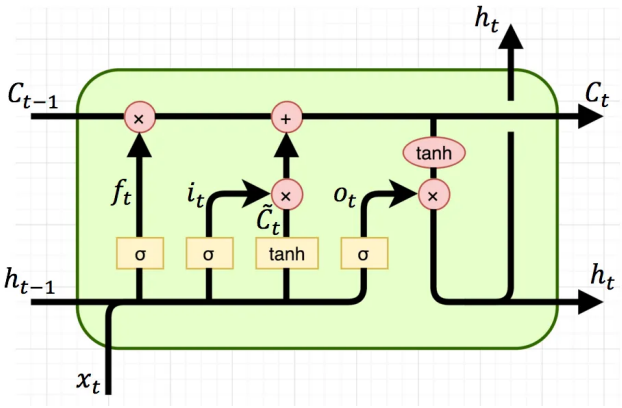

Fig. 1. LSTM CELL

An output of the LSTM model will go through the following calculation steps:

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f) \tag{1}$$
$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i) \tag{2}$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, X_t] + b_C) \tag{3}$$
$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o) \tag{4}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{5}$$
$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

The internal state transitions of the LSTM model are governed by a sequence of gated operations at each time step, as described in formula (1),(2),(3),(4),(5),(6). The forget gate activation vector at time step t, denoted as f_t in (1), determines which components of the previous cell state C_{t-1} should be retained or discarded. This gate employs the sigmoid activation function $\sigma()$, which maps its input to the range $[0, 1]$, thereby enabling a soft selection mechanism over the stored memory. The operator $[h_{t-1}, X_t]$ represents the concatenation of the previous hidden state h_{t-1} and the current input vector X_t. The resulting activation vector is then multiplied element-wise

with the previous cell state, as shown in (5), to regulate the memory retention process.

The input gate i_t, defined in (2), controls the extent to which new information derived from the current input X_t is incorporated into the cell state. This mechanism operates jointly with the candidate cell state $\tilde{C}_t$, computed using the hyperbolic tangent activation function in (3). Since the $\tanh()$ function constrains its output to the interval $(-1, 1)$, it contributes to stabilizing the internal memory dynamics and mitigates the risk of uncontrolled growth in the cell state values. The element-wise product between i_t and $\tilde{C}_t$ produces the proposed memory update, which is subsequently integrated into the long-term memory according to (5).

The output gate o_t, given in (4), regulates which portions of the updated cell state C_t are exposed as the hidden state h_t, as formulated in (6). The hidden state serves both as the output representation at the current time step and as contextual information propagated to the next temporal step. Through this gating mechanism, the LSTM architecture can selectively preserve and propagate relevant temporal features across long input sequences. This gating structure also alleviates the vanishing gradient problem commonly encountered in conventional recurrent neural networks.

The weight matrices W_f, W_i, W_C, and W_o, together with the corresponding bias vectors b_f, b_i, b_C, and b_o, constitute the trainable parameters of the LSTM model. These parameters are optimized during training via backpropagation through time, enabling the network to learn suitable temporal representations from sequential data.

C. Compression method

As illustrated in Fig. 2, PTQ and QAT follow fundamentally different optimization pipelines for deploying low–precision neural networks. In the PTQ approach, a pre–trained model is first obtained using full–precision training. Subsequently, a calibration stage is performed using a representative calibration dataset to estimate the dynamic range of weights and activations. Based on these statistics, the model parameters are quantized into fixed–point representations without further weight adaptation. Although this workflow enables rapid deployment and low computational overhead, the absence of retraining often results in accuracy degradation, particularly for recurrent architectures and time–series models operating under strict numerical precision constraints.

In contrast, the QAT workflow integrates quantization into the training loop. After initializing from a pre–trained floating–point model, quantization operators are introduced to simulate low-precision arithmetic during forward propagation. The quantized model is then retrained using the original training dataset, allowing the network to adapt its internal representations to quantization noise and reduced numerical resolution. This retraining stage enables the optimization process to compensate for rounding errors and clipping effects, thereby improving robustness and preserving model accuracy after deployment.

From an implementation perspective, the key distinction between the two approaches lies in the availability of gradient–based parameter updates under quantized constraints. While PTQ relies solely on post hoc statistical calibration, QAT performs end–to–end optimization in the presence of simulated integer arithmetic. Consequently, QAT typically produces quantized models with superior predictive performance and higher numerical stability. These advantages are particularly important for TinyML applications on microcontroller platforms, where efficient integer–only inference and minimized memory bandwidth are critical design requirements.

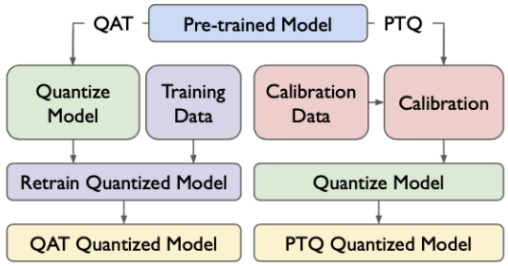

Fig. 2. Comparison between QAT and PTQ

D. Hyperparameter Optimization

The performance of machine learning models strongly depends on the choice of hyperparameters, such as learning rate, batch size, number of hidden units, and dropout rate.

HPO refers to the systematic process of searching for the best hyperparameter configuration to maximize model performance. Traditional methods such as grid search and random search are straightforward but computationally expensive and often infeasible for large hyperparameter spaces. To overcome this limitation, more advanced strategies have been proposed, including Bayesian optimization [9], evolutionary algorithms [10], and gradient-based approaches [11], which aim to reduce the number of required evaluations while effectively exploring the search space. In the context of TinyML, HPO plays a critical role in balancing accuracy, latency, and memoryfootprint.

Therefore, combining HPO with model compression techniques can significantly enhance the feasibility of deploying deep learning models on microcontrollers.

Among these methods, Bayesian Optimization and Hyperband are particularly suitable for TinyML applications. Bayesian Optimization reduces expensive evaluations while Hyperband avoids overfitting by dynamically allocating resources and stopping poor configurations early. This approach significantly reduces the time required to search for an optimal set of parameters using a principled and efficient strategy.

E. ESP32S3 Architecture

The ESP32-S3 [12] is a dual-core microcontroller based on the Xtensa® LX7 architecture, designed by Espressif Systems for low-power Artificial Intelligence of Things (AIoT) applications. The ESP32-S3 introduces enhanced support for artificial

intelligence acceleration through specialized hardware and optimized libraries.

One of the key components enabling efficient TinyML deployment on the ESP32-S3 is ESP-NN, an optimized neural network inference library developed by Espressif. ESP-NN provides hand-tuned implementations of common operations such as matrix multiplication, convolution, and activation functions, leveraging hardware-specific features to maximize performance. This library is designed to accelerate machine learning models on the ESP32-S3 platform, thereby reducing inference latency and energy consumption.

In addition, the ESP32-S3 is equipped with SIMD instruction and a FPU [13]. These features enable parallel processing of multiple data elements within a single instruction, which significantly accelerates vectorized operations commonly used in deep learning workloads. By combining ESP-NN optimizations with SIMD and FPU acceleration, the ESP32-S3 can efficiently execute quantized neural networks and achieve real-time inference performance under strict memory and power constraints.

III. Method Description

A. Dataset preparation

The proposed architecture is evaluated on the Human Activity Recognition (HAR) dataset [14], which is a widely used benchmark for assessing the performance of machine learning models in. The dataset was collected from 30 volunteers aged 19–48 performing six daily activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying down, as illustrated in Fig. 3.. The original sensor signals were preprocessed using noise filtering and segmented into fixed-length windows of 2.56 seconds (128 readings per window) with 50% overlap. From these segments, both raw time-series signals and a set of 561 handcrafted features, covering time-domain and frequency-domain characteristics such as mean, standard deviation, correlation, and energy are provided. However, in this study, only the raw sensor signals are used as input to the model, without employing the preprocessed feature-based data.

Fig. 3. Examples of human activities performed in the experiment

For model development, the dataset is divided into training and testing subsets following the standard split provided in the original dataset. Furthermore, 20% of the training data is randomly selected as a validation set to support hyperparameter optimization and to monitor model generalization during training.

B. LSTM implementation

The proposed model architecture was inspired by the work of Brownlee [15], which demonstrates the design of recurrent neural networks for time-series classification in the context of human activity recognition. The architecture consists of the following layers: an LSTM layer with 100 hidden units is used as the first layer to model temporal patterns in the input sequence. A fully connected layer with 100 neurons and ReLU activation is introduced to learn high-level feature representations after the sequential modeling stage.

In this study, the LSTM layer, the dense layer with 100 units, and the output layer with 6 neurons are retained, and the HPO method is applied to determine the optimal number of LSTM units as well as other associated hyperparameters.

C. Implement Hyperparameter Optimization Methods

Before applying QAT, it is necessary to determine an appropriate set of hyperparameters to ensure stable model convergence and maximize classification accuracy. In this study, two complementary hyperparameter optimization strategies, namely Hyperband and Bayesian optimization, were employed to systematically explore the search space.

In this work, the Hyperband tuner was configured with a maximum training budget of 200 epochs and a reduction factor of three, enabling progressive refinement of candidate solutions. Early stopping based on validation loss was additionally incorporated to prevent unnecessary computations and reduce the risk of overfitting.

To further improve the robustness of the search process, Bayesian optimization was subsequently applied. This approach constructs a probabilistic surrogate model of the objective function, defined in terms of validation accuracy, and iteratively selects hyperparameter configurations that maximize the expected performance improvement.

The explored search space includes the number of LSTM hidden units, dropout rate, recurrent dropout rate, optimizer type, learning rate, and the coefficient of ℓ_2 regularization applied to both kernel and recurrent weight matrices. In particular, the number of hidden units was varied from 32 to 128 with a step size of 32, while the learning rate was sampled on a logarithmic scale to capture both coarse- and fine-grained optimization behaviours. The ℓ_2 regularization factor of 1×10^{-4} was applied to balance model generalization capability and training stability. Multiple gradient-based optimizers, including Adam, stochastic gradient descent (SGD), and RMSProp, were also evaluated during the search process.

After identifying the best-performing hyperparameter configuration from each tuning strategy, the corresponding model was retrained using the selected settings and subsequently evaluated on an independent test set to obtain the final performance metrics.

D. Integration with TensorFlow lite for microcontrollers

To efficiently deploy the proposed LSTM model on the ESP32-S3 microcontroller, we leverage the ESP-NN library, an

optimized neural network kernel library developed by Espressif. ESP-NN provides hand-optimized implementations of core neural network operations such as matrix multiplication, convolution, and activation functions, which are specifically tuned to exploit the hardware features of the ESP32-S3. By directly utilizing SIMD instructions and the integrated Floating-Point Unit, ESP-NN significantly accelerates vectorized computations and reduces inference latency.

In addition, the proposed system integrates ESP-NN with TFLM. While TFLM provides a lightweight runtime for executing deep learning models on microcontrollers, ESP-NN serves as a hardware-aware backend, replacing default operators with optimized counterparts that can execute in parallel. This combination enables efficient support for computationally intensive operations in recurrent architectures, particularly LSTM layers, which involve multiple matrix multiplications and non-linear activations.

Through the synergy between TFLM's portable operator framework and ESP-NN's hardware-level optimizations, the system is able to exploit the full computational potential of the ESP32-S3.

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_22 (LSTM)               (None, 96)                40704

dense_44 (Dense)             (None, 100)               9700

dense_45 (Dense)             (None, 6)                 606

=================================================================
Total params: 51,010
Trainable params: 51,010
Non-trainable params: 0
```

Fig. 4. LSTM layer

IV. EXPERIMENT RESULTS

A. LSTM HPO result

The model achieving the highest accuracy was obtained using the Hyperband hyperparameter optimization method. However, in terms of diversity in the number of LSTM cells explored, the Bayesian optimization method produced more diverse results. Table I compares the three best results from both optimization methods.

The hyperparameter configuration with the highest accuracy, namely Hyperband set 3 with 96 LSTM cells in Fig. 4. In Fig. 5 indicate that this configuration enables effective learning behavior for the LSTM architecture. As shown in the Fig. 5, both the training and validation losses converge rapidly and remain stable with minimal fluctuations thereafter. This convergence behavior suggests that the model has reached an optimal solution.

B. Quantization-Aware Training for LSTM

Fig. 2 illustrates the training dynamics of the LSTM model when QAT is applied using the hyperparameters obtained from

TABLE I
COMPARE HPO

HPO \Result	LSTM Unit	Dropout-Recurrent dropout	Learning rate	Optimizer	Acc
Bayesian set 1	32	0.1-0.2	0.0029591736434515996	Adam	91.653
Bayesian set 2	64	0.2-0.2	0.0027568210330104867	Adam	92.162
Bayesian set 3	128	0.5-0.2	0.01	Adam	91.788
Hyperband set 1	64	0.1-0.1	0.009551153000449404	Adam	92.603
Hyperband set 2	96	0.3-0.2	0.004545416702014981	Adam	93.281
Hyperband set 3	96	0.1-0.1	0.004627602634393576	Rmsprop	91.890

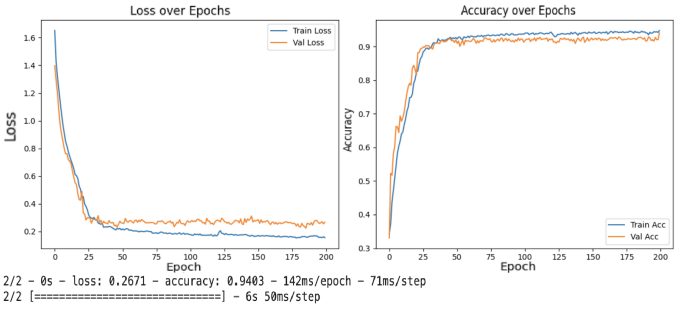

Fig. 5. Training and validation loss and accuracy of the uncompressed model

the Hyperband optimization procedure. In this work, a customized QAT configuration is adopted to better accommodate the recurrent structure of the LSTM layer. Specifically, the quantization process is applied to the kernel weights, recurrent kernel weights, and bias parameters using an 8-bit symmetric fixed-point representation. This design choice enables a direct reduction in model memory footprint while maintaining numerical stability during sequential state propagation.

As shown in Fig. 6, both the training and validation losses decrease steadily across epochs, eventually reaching values comparable to those of the full-precision baseline model. Although minor oscillations can be observed in the validation curve, the overall convergence remains stable within the range of approximately 0.25–0.30, indicating that the quantized model maintains effective generalization capability. In addition, the training accuracy rapidly increases and saturates at around 94%, while the validation accuracy stabilizes at approximately 91–92%. This small performance gap demonstrates that the proposed QAT configuration successfully preserves the representational capacity of the LSTM network despite the use of reduced numerical precision.

From an embedded deployment perspective, the adoption of weight-only quantization provides a favorable trade-off between computational efficiency and predictive performance. The resulting quantized model exhibits significantly reduced parameter storage requirements and improved compatibility

TABLE II
COMPARE INFERENCE TIME WITH LSTM UNITS

LSTM Unit	Inference time (ms)	Accuaracy (%)
96	640	93.79
64	352	93.55
32	120	92.73

with integer-oriented execution units, thereby facilitating efficient real-time inference on resource-constrained microcontroller platforms such as the ESP32-S3.

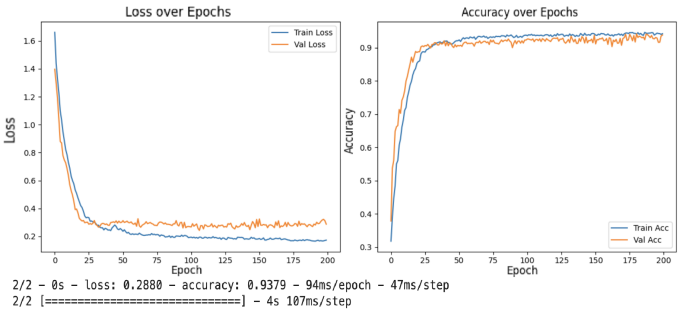

Fig. 6. Training and validation loss and accuracy of the compressed model

C. Inference Time in ESP32S3

The inference time heavily depends on the number of LSTM cells. According to the LSTM formulation, each input must pass through four gates within a single LSTM cell.

Therefore, the computational cost increases proportionally with the number of LSTM cells. Table II compares inference time and accuracy when varying the number of LSTM cells.

With 32 LSTM cells, the lowest inference time is 120 ms for a data sequence of 128 inputs (equivalent to 128 × 16 bytes), achieving an acceptable accuracy of 92.73%, which is approximately 2% lower than the original model. For 96 LSTM cells, the inference time reaches 650 ms, yielding the highest accuracy of 93.79%. Meanwhile, the 64-cell configuration provides a balance between inference time and accuracy, achieving 92.55% accuracy with an inference time of 352 ms.

V. CONCLUSION

The experimental results demonstrate that the QAT technique effectively reduces model size while maintaining nearly unchanged accuracy compared to the uncompressed baseline model. In addition, the application of hyperparameter optimization strategies enables the discovery of more optimal machine learning architectures. Specifically, the Bayesian optimization approach identified a configuration with 32 LSTM units while still preserving high accuracy. Meanwhile, the Hyperband method achieved the highest classification accuracy on the evaluated dataset by determining an optimal set of hyperparameters.

The primary contribution of this study is the development of a lightweight model compression pipeline for embedded systems that achieves high accuracy on each specific dataset. This work establishes a practical foundation for the deployment of artificial intelligence applications on resource-constrained devices, particularly the ESP32-S3 platform.

However, several limitations remain in the current implementation. The computational cost associated with QAT training, as well as the repeated evaluations required during the Hyperband optimization process, is relatively high and may hinder scalability when applied to larger datasets or more complex architectures.

Future work should investigate lightweight recurrent neural network architectures such as GRU or lightweight transformer models, explore mixed-precision quantization strategies, and incorporate automated neural architecture search techniques to further optimize the trade-off among accuracy, model size, and inference latency

REFERENCES

[1] Z. Li, H. Li, and L. Meng, "Model compression for deep neural networks: A survey," *Computers*, vol. 12, no. 3, p. 60, 2023, doi: 10.3390/computers12030060.

[2] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on TinyML," *IEEE Access*, vol. 11, pp. 96892–96922, 2023, doi: 10.1109/ACCESS.2023.3294111.

[3] G. Abib, F. Castel, N. Satouri, H. Afifi, and A. M. Said, "Survey and enhancements on deploying LSTM recurrent neural networks on embedded systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Rome, Italy, 2023, pp. 949–953, doi: 10.1109/ICC45041.2023.10278766.

[4] K. Xia, J. Huang, and H. Wang, "LSTM-CNN architecture for human activity recognition," *IEEE Access*, vol. 8, pp. 56855–56866, 2020, doi: 10.1109/ACCESS.2020.2982225.

[5] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2018, pp. 11–20, doi: 10.1145/3174243.3174253.

[6] M. J. Robot, "Temperature prediction using a TinyML LSTM model," Hackster.io, 2024. [Online]. Available: https://www.hackster.io

[7] C. Siegl, "TensorFlow lite for microcontrollers adds support for efficient LSTM implementation," Medium, 2022. [Online]. Available: https://medium.com

[8] Espressif Systems, "Hand gesture recognition on ESP32-S3 with ESP-Deep Learning," Espressif Developer Blog, 2022. [Online]. Available: https://developer.espressif.com

[9] A. H. Victoria and G. Maragatham, "Automatic tuning of hyperparameters using Bayesian optimization," *Evolving Systems*, vol. 12, pp. 217–223, 2021, doi: 10.1007/s12530-020-09345-2.

[10] J.-Y. Kim and S.-B. Cho, "Evolutionary optimization of hyperparameters in deep learning models," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Wellington, New Zealand, 2019, pp. 831–837, doi: 10.1109/CEC.2019.8790354.

[11] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, Aug. 2000, doi: 10.1162/089976600300015187.

[12] Espressif Systems, *ESP32-S3 Wi-Fi and BLE 5 SoC Datasheet*. Espressif Systems, 2022.

[13] C. J. Hughes, *Single-Instruction Multiple-Data Execution*. Cham, Switzerland: Springer Nature, 2022.

[14] J. L. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra, "Human activity recognition using smartphones," UCI Mach. Learn. Repository, 2013, doi: 10.24432/C54S4K.

[15] J. Brownlee, "LSTMs for human activity recognition time series classification," Machine Learning Mastery, 2020. [Online]. Available: https://machinelearningmastery.com