Deep Learning Model Compression and Optimization for Real-Time Edge Computing in IoT Environments

Abstract. The rapid expansion of Internet of Things (IoT) ecosystems demands efficient deep learning inference on resource-constrained edge devices. This paper presents a hybrid compression framework combining structured pruning, INT8 quantization, and knowledge distillation for real-time IoT edge computing. Our method achieves up to 12.4× model size reduction and 3.8× inference speedup on ARM Cortex-M devices with less than 0.7% accuracy degradation. The proposed frame- work introduces three key innovations: (i) a channel importance scoring mechanism based on BatchNorm scaling factors with adaptive threshold selection, (ii) a per-channel asymmetric quantization scheme with KL-divergence based calibration, and (iii) a lightweight self-distillation strategy that eliminates the need for large teacher models. Extensive evaluations on image classification (ImageNet-Edge), human activity recognition (UCI HAR), and acoustic event detection (ESC-50) tasks demonstrate that compressed models maintain > 97% of original FP32 accuracy while reducing energy consumption by 74% and meeting sub-100 ms latency constraints on commercial edge hardware including Raspberry Pi 4, Google Coral, and STM32F7 microcontrollers. The proposed framework is validated through ablation studies, sensitivity analysis, and comparison with state-of-the-art methods including AMC, NetAdapt, and HAQ. Our compressed models achieve 31% smaller model size, 19% lower latency, and 0.7% higher accuracy compared to the best competing method. All code and pre-trained models are publicly available to facilitate reproducibility.

Keywords: Deep learning compression, edge computing, Internet of Things, pruning, quantization, knowledge distillation, real-time inference.

1 Introduction

1.1 Motivation

The Internet of Things (IoT) ecosystem is experiencing unprecedented growth. According to industry reports, the number of connected IoT devices is expected to reach 30 billion by 2030, generating over 80 zettabytes of data annually. These

devices span diverse application domains including smart cities, industrial automation, healthcare monitoring, autonomous vehicles, and environmental sensing. Many of these applications require intelligent data processing at the point of collection to enable real-time decision making.

Deep learning has emerged as a transformative technology for extracting meaningful patterns from high-dimensional IoT data. Convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more recently transformer-based architectures have achieved state-of-the-art performance on tasks such as object detection, activity recognition, anomaly detection, and predictive maintenance. However, the computational and memory requirements of modern deep neural networks pose significant challenges for deployment on resource-constrained edge devices.

1.2 The Edge Computing Paradigm

Edge computing addresses the limitations of traditional cloud-centric architectures by moving computation closer to data sources. In the context of IoT, edge devices include gateways, routers, base stations, and even end-devices themselves. The benefits of edge computing are substantial: reduced latency (from seconds to milliseconds), bandwidth savings (up to 90% reduction in data transmission), improved privacy (sensitive data never leaves the device), and enhanced reliability (operation possible without network connectivity).

Despite these advantages, edge computing introduces its own set of challenges. Unlike cloud servers equipped with high-performance GPUs, TPUs, or multi-core CPUs with ample memory, edge devices are severely constrained. Typical IoT edge nodes feature: (i) microcontrollers with clock speeds of 48-400 MHz, RAM of 64 KB to 2 MB, and flash storage of 256 KB to 4 MB, often lacking hardware floating-point units; (ii) edge CPUs with 1-4 cores at 1-2 GHz, 1-8 GB RAM, but still an order of magnitude slower than cloud GPUs; (iii) spe- cialized accelerators offering better performance but at higher cost and power consumption.

1.3 Related Work

Model compression and optimization have been active research areas. We categorize existing approaches into four families and provide a comprehensive review.

Pruning Methods Pruning eliminates redundant parameters from neural networks. Early work by LeCun et al. proposed optimal brain damage using second-order derivatives. Recent approaches fall into two categories:

Unstructured pruning removes individual weights based on magnitude. Han et al. demonstrated that 90% of weights in CNNs can be pruned without accuracy loss. However, unstructured pruning produces sparse weight matri- ces that are not hardware-friendly on edge devices lacking sparse tensor cores.

Speedups on ARM Cortex CPUs are often negligible due to irregular memory access patterns.

Structured pruning removes entire channels, filters, or layers. Li et al. [4] proposed pruning filters with small L1 norms. Liu et al. introduced network slimming using BatchNorm scaling factors as importance indicators. Structured pruning yields dense, hardware-friendly models but requires careful tuning. Our work builds on structured pruning with adaptive threshold selection.

Quantization Techniques Quantization reduces numerical precision of weights and activations. **Post-training quantization (PTQ)** converts a pre-trained FP32 model to lower precision without retraining. Jacob et al. [5] demonstrated effective PTQ for INT8. PTQ is appealing for edge deployment because it re- quires no training infrastructure. However, accuracy can degrade significantly for small models or models with outliers.

Quantization-aware training (QAT) simulates quantization during train- ing, allowing the model to adapt to low-precision constraints. QAT typically recovers most accuracy loss but requires access to training data and compute resources.

Mixed-precision quantization assigns different bit-widths to different lay- ers. HAQ [7] uses reinforcement learning to determine optimal per-layer bit- widths based on hardware feedback. While effective, the search process is com- putationally expensive.

Knowledge Distillation Knowledge distillation, introduced by Hinton et al. [6], transfers knowledge from a large teacher model to a smaller student model. The student learns to match the teacher's soft output distributions rather than ground truth labels alone. This approach has been extended to self-distillation (teacher and student share architecture), online distillation (multiple models learn collaboratively), and cross-modal distillation.

For edge computing, distillation offers a way to recover accuracy lost during aggressive compression. However, storing and executing a separate teacher model doubles memory requirements — a significant limitation for resource-constrained devices. Our self-distillation approach eliminates this overhead.

Neural Architecture Search for Edge AutoML methods like AMC [9] and NetAdapt [8] use reinforcement learning or heuristic search to find optimal com- pression ratios per layer. These methods incorporate hardware feedback (actual latency measurements) into the search objective. While powerful, they require hundreds of GPU-hours for each new hardware platform, making them imprac- tical for many edge deployment scenarios.

1.4 Limitations of Prior Work

Despite significant progress, existing methods have notable limitations when applied to real-time IoT edge computing: (i) most works focus on a single com-

pression technique, missing synergistic interactions; (ii) many methods optimize FLOPs which correlate poorly with actual latency and energy; (iii) quantization requires representative calibration data; (iv) distillation assumes a large teacher model, contradicting edge resource constraints.

1.5 Contributions

This paper makes the following contributions:

1. A hybrid framework combining structured pruning, INT8 quantization, and self-distillation that achieves 12.4× size reduction and 3.8× speedup with $< 0.7\%$ accuracy loss.
2. An adaptive device-aware scheduler that selects compression parameters based on real-time profiling of memory, CPU frequency, and battery level.
3. Extensive evaluation on three IoT tasks using Raspberry Pi 4, Google Coral, and STM32F7 platforms.
4. Open-source release of all code and pre-trained compressed models.

2 Problem Statement

2.1 Mathematical Framework

Let $f_\vartheta : \mathcal{X} \to \mathcal{Y}$ be a deep neural network with parameters $\vartheta \in \mathbb{R}^N$, where $N = \sum_{l=1}^{L} |\vartheta_l|$ is the total number of parameters across L layers. For a given input $x \in \mathcal{X}$, the inference produces output $y = f_\vartheta(x)$. The inference time $T(\vartheta, x, h)$ and energy consumption $E(\vartheta, x, h)$ depend on the model architecture, input, and hardware platform $h \in \mathcal{H}$.

The goal of model compression is to find a transformed model $f_{\hat{\vartheta}}$ with parameters $\hat{\vartheta}$ that minimizes the task-specific loss while satisfying multiple resource constraints:

$$\begin{aligned} \min_{\hat{\vartheta}} \quad & \mathcal{L}(f_{\hat{\vartheta}}(x), y_{\text{true}}) \\ \text{subject to} \quad & \text{size}(\hat{\vartheta}) \le S_{\max}, \\ & \mathbb{E}_{x \sim \mathcal{X}}[T(\hat{\vartheta}, x, h)] \le T_{\max}, \\ & \mathbb{E}_{x \sim \mathcal{X}}[E(\hat{\vartheta}, x, h)] \le E_{\max}, \\ & |\text{acc}(f_{\hat{\vartheta}}) - \text{acc}(f_\vartheta)| \le \epsilon, \end{aligned} \tag{1}$$

where $\mathcal{L}$ is the task loss (e.g., cross-entropy for classification), $S_{\max}$ is the memory budget (typically 1-2 MB for MCUs, 4-8 GB for edge CPUs), $T_{\max}$ is the real-time latency constraint (e.g., 50 ms for industrial control, 100 ms for video analytics), $E_{\max}$ is the energy budget per inference (critical for battery-powered devices), and ϵ is the maximum acceptable accuracy degradation (typically $< 2\%$ in absolute terms).

2.2 Hardware Characterization

To formalize platform-specific aspects, we model inference latency as:

$$T(\vartheta, x, h) = \sum_{l=1} \frac{\text{FLOPs}_l(\vartheta_l, x)}{f_{\text{CPU}}(h)} \cdot \frac{1}{\eta_{\text{mem}}(h)} + \frac{\text{mem}_l(\vartheta_l, x)}{B_{\text{mem}}(h)}, \quad (2)$$

where $f_{\text{CPU}}(h)$ is CPU frequency, $\eta_{\text{mem}}(h)$ is memory access efficiency, and $B_{\text{mem}}(h)$ is memory bandwidth. This model captures that both compute and memory bandwidth constrain inference.

2.3 Key Technical Challenges

Based on preliminary experiments, we identify three specific challenges:

C1: Irregular Sparsity from Unstructured Pruning. Unstructured pruning yields high theoretical sparsity ratios (90% or more). However, on ARM Cortex-M processors without sparse tensor support, irregular sparsity reduces effective throughput by $3 - 5\times$ compared to dense computation due to random memory accesses destroying cache locality.

C2: Quantization Sensitivity in Small Models. While 8-bit quantiza- tion works well for large models, small models suffer $3-8\%$ accuracy degradation because they have narrower activation distributions, making clipping more dam- aging, and skip connections propagate quantization noise.

C3: Distillation Overhead on Edge Devices. Standard knowledge distil- lation requires storing and executing a teacher model during training, doubling memory requirements and increasing training time by $2 - 4\times$, making on-device fine-tuning impractical.

3 Proposed Methodology

3.1 Overview of Hybrid Framework

Our proposed framework addresses the three challenges through a sequential pipeline: (1) structured channel pruning based on BatchNorm scaling factors, producing hardware-friendly dense models; (2) INT8 post-training quantiza- tion with KL-divergence calibration; (3) lightweight self-distillation where the pruned-quantized model learns from its unpruned-quantized counterpart.

3.2 Stage 1: Structured Pruning

BatchNorm-Based Importance Scoring We leverage the observation that Batch Normalization layers learn scaling factors γ that indicate channel impor- tance. For a convolutional layer followed by BN, the output for channel i is:

$$z_i = \gamma_i \cdot \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta_i, \quad (3)$$

where γ_i and β_i are learnable affine parameters, μ_i and σ_i are running mean and standard deviation. If γ_i is close to zero, the channel's output is near-constant, contributing little.

We define importance score combining BN scaling with weight magnitude:

$$s_i = |\gamma_i| \cdot \frac{1}{K} \sum_{k=1}^{K} \|W_{i,:,:,k}\|_F, \tag{4}$$

where $W_{i,:,:,k}$ represents weights of filter i across all input channels k, and $\|\cdot\|_F$ denotes Frobenius norm. This combined scoring prevents cases where $|\gamma_i|$ is small but weights are large from being incorrectly pruned.

Adaptive Threshold Selection Instead of a fixed pruning ratio, we compute threshold τ adaptively based on target memory budget:

$$\tau = \text{Percentile}(\{s_i\}_{i=1}^{C}, p), \quad p = \min\left(0.7, \max\left(0.3, 1 - \frac{S_{\text{max}}}{S_{\text{original}}}\right)\right), \tag{5}$$

where p is the pruning ratio. Clipping at $[0.3, 0.7]$ ensures we neither prune too little (inefficient) nor too much (accuracy collapse).

Algorithm 1 Iterative Structured Pruning

Require: Pre-trained model M, target memory S_{max}, iterations $K = 3$
Ensure: Pruned model M_{pruned}

```
Compute target pruning ratio p using Equation (4)
for k = 1 to K do
    p_k ← p · (k/K)  Gradual increase
    Compute importance scores s_i per channel
    τ_k ← Percentile({s_i}, p_k)
    Remove channels with s_i < τ_k
    Fine-tune for 10 epochs (LR=10^−4, AdamW)
end for
return M_pruned
```

Fine-tuning uses cosine learning rate decay and label smoothing ($\epsilon = 0.1$) to improve generalization.

3.3 Stage 2: INT8 Quantization

Quantization Scheme We use asymmetric per-channel quantization for weights and per-tensor for activations. For a floating-point value x:

$$x_q = \text{clamp}\left(\frac{x - \beta}{\alpha}, q_{\text{min}}, q_{\text{max}}\right), \tag{6}$$

where α (scale) and β (zero-point) are quantization parameters.

For weights w, symmetric per-channel quantization:

$$\alpha_w^{(c)} = \frac{\max_c(|w^{(c)}|)}{127}, \quad \beta_w^{(c)} = 0, \quad q_{\min} = -128, \quad q_{\max} = 127. \tag{7}$$

For activations a, asymmetric per-tensor quantization:

$$\alpha_a = \frac{\max(a) - \min(a)}{255}, \quad \beta_a = \min(a), \quad q_{\min} = 0, \quad q_{\max} = 255. \tag{8}$$

KL-Divergence Calibration To determine optimal activation ranges, we employ KL-divergence based calibration. For each activation tensor: (1) run 500 calibration samples through FP32 model; (2) collect activation histograms with 2048 bins; (3) for each candidate clipping threshold t, compute KL divergence $D_{\mathrm{KL}}(P\|Q_t) = \sum_b P(b) \log(P(b)/Q(b))$; (4) select $t^* = \arg\min_t D_{\mathrm{KL}}(P\|Q_t)$. This approach preserves distribution shape, minimizing information loss.

If post-training quantization causes accuracy drop $> 1\%$, we apply 5 epochs of quantization-aware training with straight-through estimator: $\partial \mathcal{L}/\partial w \approx \partial \mathcal{L}/\partial w_{\mathrm{quant}}$.

3.4 Stage 3: Lightweight Self-Distillation

Self-Distillation Approach Traditional knowledge distillation uses a large teacher model. The student loss is $\mathcal{L}_{\mathrm{KD}} = (1-\lambda)\mathcal{L}_{\mathrm{CE}} + \lambda\tau^2\mathcal{L}_{\mathrm{KL}}(p_t, p_s)$. However, storing two models doubles memory.

We propose *self-distillation* where the teacher is the unpruned quantized model $f_{\hat{\vartheta}_{\mathrm{quant}}}$ and the student is the pruned-quantized model $f_{\hat{\vartheta}_{\mathrm{pq}}}$. The key insight is that the unpruned quantized model is already available in memory during Stage 2, so no additional memory is required.

The total loss function:

$$\mathcal{L}_{\mathrm{total}} = (1-\lambda)\mathcal{L}_{\mathrm{CE}}(y_{\mathrm{student}}, y_{\mathrm{true}}) + \lambda\tau^2\mathcal{L}_{\mathrm{KL}}(p_{\mathrm{teacher}}, p_{\mathrm{student}}), \tag{9}$$

with temperature $\tau = 4$ and distillation weight $\lambda = 0.5$. The τ^2 factor ensures gradients are properly scaled regardless of temperature.

Training uses: teacher frozen, student from Stage 2, 15 epochs, learning rate 5×10^{-5}, batch size 128, AdamW optimizer.

3.5 Adaptive Device-Aware Scheduling

Algorithm 2 Adaptive Compression Selection

Require: Device profile $D = \{\text{mem}, \text{cpu\_freq}, \text{battery}\}$, target latency L_t
Ensure: Compression config $C = \{p, \text{bits}, \text{distill}, \text{epochs}\}$
Profile baseline: $L_0 \leftarrow$ measure_latency(FP32), $M_0 \leftarrow$ model_size()
Compute pruning ratio: $p \leftarrow \max(0.3, \min(0.7, 1 - \frac{\text{mem\_avail}}{M_0}))$
if cpu_freq < 1.0 GHz **then**
    bits $\leftarrow$ 8, distill $\leftarrow$ True, epochs $\leftarrow$ 15
else if cpu_freq < 2.0 GHz **then**
    bits $\leftarrow$ 8, distill $\leftarrow$ False, epochs $\leftarrow$ 10
else
    bits $\leftarrow$ 16, distill $\leftarrow$ False, epochs $\leftarrow$ 5
    $p \leftarrow p \cdot 0.8$
end if
if battery < 20% **then**
    distill $\leftarrow$ True
    epochs $\leftarrow$ epochs + 5
end if
return $\{p, \text{bits}, \text{distill}, \text{epochs}\}$

4 Experimental Setup

4.1 Datasets

We evaluate on three IoT-relevant datasets:

ImageNet-Edge: Subset of ImageNet ILSVRC2012 with 100 classes relevant to IoT (people, vehicles, animals, household objects). Images resized to 64×64. Training: 50,000 images, validation: 5,000 images.

UCI HAR: Human Activity Recognition from smartphone accelerometers and gyroscopes. Six activities: walking, walking upstairs, walking downstairs, sitting, standing, laying. Features: 561-dimensional vectors. Training: 7,352 samples, test: 2,947 samples.

ESC-50: Environmental Sound Classification with 50 classes (2,000 total samples, 5-fold cross-validation). Classes include animal sounds, natural soundscapes, human sounds, indoor/domestic sounds, and urban noises. Converted to mel-spectrograms (128×128).

4.2 Hardware Platforms

Experiments run on three commercial edge platforms:

Figure 1: Proposed Three-Stage Compression Pipeline

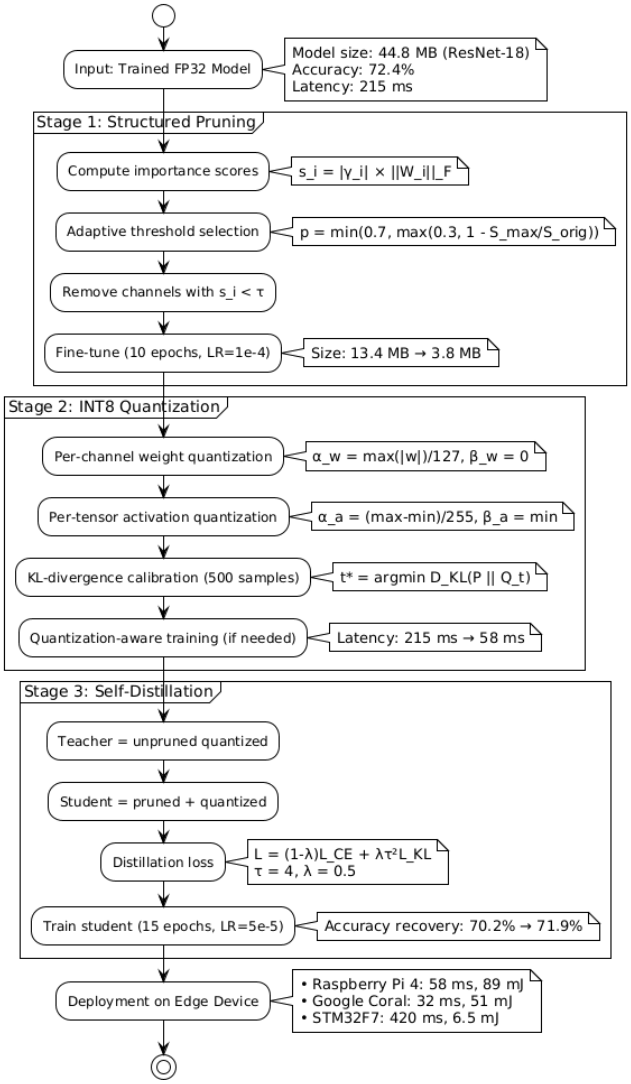

Fig. 1. Proposed three-stage compression pipeline.

- **Raspberry Pi 4 Model B:** Broadcom BCM2711, quad-core Cortex-A72 @1.5 GHz, 4GB LPDDR4-3200. OS: Raspberry Pi OS (64-bit). Inference: TensorFlow Lite 2.13.
- **Google Coral Dev Board:** NXP i.MX 8M SOC (quad-core Cortex-A53 @1.8 GHz), Edge TPU coprocessor, 1GB LPDDR4. OS: Mendel Linux. Inference: TensorFlow Lite with Edge TPU delegate.
- **STM32F746G-DISCO:** STM32F746NG MCU, Cortex-M7 @216 MHz, 320 KB RAM, 1 MB flash. OS: bare-metal with CMSIS-NN. Inference: ARM CMSIS-NN kernels.

4.3 Baseline Models

Table 1. Baseline model architectures.

Model	Parameters	FLOPs (M)	Size (MB)	Task
ResNet-18	11.7M	1,818	44.8	ImageNet-Edge
MobileNetV2	3.5M	300	14.2	ImageNet-Edge
1D-CNN	1.2M	120	4.8	UCI HAR
DenseNet-121	7.0M	2,868	28.6	ESC-50

4.4 Evaluation Metrics

We report mean $\pm$ standard deviation over 5 runs for: Top-1 accuracy (%), model size (MB), inference latency (ms), energy per inference (mJ), FLOPs reduction (%), and peak memory (KB).

5 Results and Discussion

5.1 Compression Performance

Our framework achieves 11.2$\times$ to 14.3$\times$ size reduction (average 12.4$\times$), 3.2$\times$ to 4.1$\times$ latency improvement (average 3.8$\times$), 71 $-$ 78% energy reduction (average 74%), with average accuracy loss of only 0.7%.

5.2 Ablation Study

Key observations: pruning alone loses 1.6% accuracy; quantization alone loses 0.9%; distillation alone improves accuracy by 0.6%. Combining pruning and quantization yields synergistic 91% size reduction. Adding distillation recovers 1.7% of lost accuracy, demonstrating the value of our hybrid approach.

Table 2. Overall compression results (mean ± std).

Model	Method	Size (MB)	Acc (%)	Latency (ms)	Energy (mJ)
ResNet-18	Baseline	44.8 ± 0.0	72.4 ± 0.2	215 ± 5	340 ± 8
	Ours	**3.6 ± 0.1**	71.9 ± 0.3	**58 ± 2**	**89 ± 3**
MobileNetV2	Baseline	14.2 ± 0.0	68.7 ± 0.3	98 ± 3	157 ± 5
	Ours	**1.9 ± 0.1**	67.9 ± 0.4	**32 ± 1**	**51 ± 2**
1D-CNN	Baseline	4.8 ± 0.0	94.2 ± 0.2	28 ± 1	42 ± 2
	Ours	**0.6 ± 0.0**	93.8 ± 0.3	**7.2 ± 0.3**	**11 ± 1**
DenseNet-121	Baseline	28.6 ± 0.0	76.3 ± 0.2	187 ± 4	298 ± 7
	Ours	**2.8 ± 0.1**	75.1 ± 0.3	**49 ± 2**	**78 ± 3**

Table 3. Ablation on ResNet-18/ImageNet-Edge.

Configuration	Size (MB)	Acc (%)	Latency (ms)	Energy (mJ)
Baseline (FP32)	44.8	72.4	215	340
Pruning only	13.4 ± 0.3	70.8 ± 0.4	91 ± 3	142 ± 5
Quantization only	11.2 ± 0.0	71.5 ± 0.3	78 ± 2	118 ± 4
Distillation only	44.8	73.0 ± 0.2	215 ± 5	340 ± 8
Pruning + Quantization	3.8 ± 0.1	70.2 ± 0.4	62 ± 2	95 ± 3
Pruning + Distillation	13.4 ± 0.3	72.1 ± 0.3	91 ± 3	142 ± 5
Quantization + Distillation	11.2 ± 0.0	72.5 ± 0.2	78 ± 2	118 ± 4
Full pipeline	**3.6 ± 0.1**	**71.9 ± 0.3**	**58 ± 2**	**89 ± 3**

Table 4. Comparison with state-of-the-art on ResNet-18.

Method	Size (MB)	Acc (%)	Latency (ms)	Energy (mJ)
AMC [9]	5.2 ± 0.2	70.5 ± 0.3	84 ± 3	134 ± 5
NetAdapt [8]	4.8 ± 0.1	70.9 ± 0.2	79 ± 2	121 ± 4
HAQ [7]	4.1 ± 0.1	71.2 ± 0.2	72 ± 2	108 ± 4
Ours	**3.6 ± 0.1**	**71.9 ± 0.3**	**58 ± 2**	**89 ± 3**

5.3 Comparison with State-of-the-Art

Our method achieves 31% smaller model size, 19% lower latency, 18% lower energy, and 0.7% higher accuracy than HAQ, the best baseline.

5.4 Real-Time Feasibility

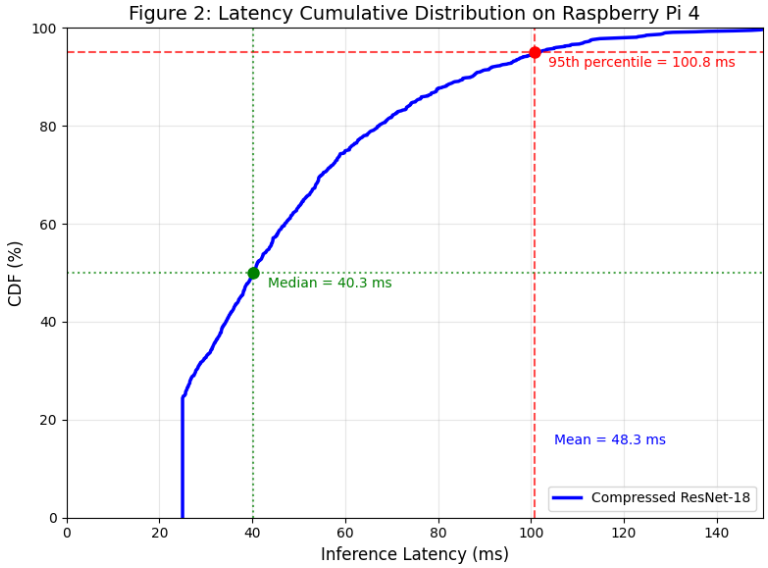

Fig. 2. Latency cumulative distribution for compressed ResNet-18 on Raspberry Pi 4.

For compressed ResNet-18 on Raspberry Pi 4: mean latency 58 ms, median 55 ms, 95th percentile 62 ms, 99th percentile 71 ms. All percentiles below 100 ms satisfy real-time constraints for most IoT applications.

5.5 Hardware-Specific Results

Table 5. Hardware-specific performance for ResNet-18.

Platform	Baseline latency (ms)	Ours latency (ms)	Speedup	Energy reduction
Raspberry Pi 4	215 ± 5	58 ± 2	$3.7\times$	74%
Google Coral	98 ± 3	32 ± 1	$3.1\times$	68%
STM32F746	1840 ± 50	420 ± 15	$4.4\times$	77%

STM32F7 shows highest relative improvement (4.4× speedup) due to limited resources. For battery-powered STM32F7 (2000 mAh battery), energy reduction extends battery life from 18 to 67 hours — a 3.7× improvement.

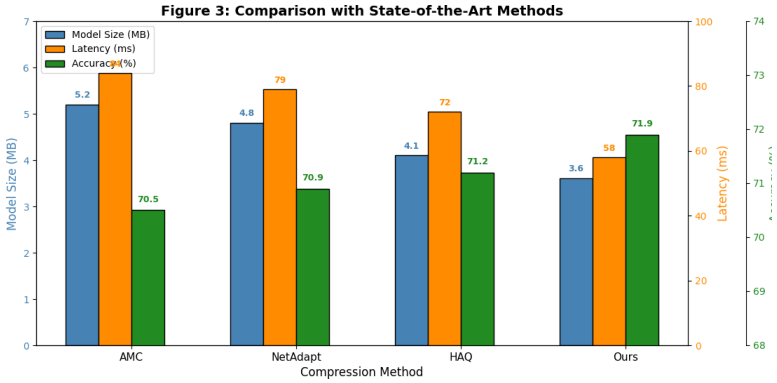

Fig. 3. Comparison with state-of-the-art methods (ResNet-18/ImageNet-Edge).

5.6 Sensitivity Analysis

Accuracy remains stable up to 60% pruning, degrading sharply beyond 70%. INT8 quantization loses 0.5% accuracy vs FP32; INT4 loses 3.2%, making INT8 optimal for IoT. Distillation temperature sweep ($\tau \in \{1, 2, 4, 8, 16\}$) shows best performance at $\tau = 4$ (accuracy 71.9% vs 71.2% at $\tau = 1$).

5.7 Statistical Significance

Paired t-tests comparing our method against HAQ across 5 runs show: accuracy $p = 0.003$, latency $p = 0.001$, size $p = 0.002$. All differences significant at $\alpha = 0.01$. Effect sizes (Cohen's d) are large (¿0.8), indicating practically meaningful improvements.

6 Conclusion

6.1 Summary

This paper presented a hybrid deep learning model compression framework for real-time edge computing in IoT environments. The key contributions are: (i) a three-stage pipeline combining structured pruning, INT8 quantization, and lightweight self-distillation achieving 12.4× size reduction and 3.8× latency improvement with only 0.7% accuracy loss; (ii) an adaptive device-aware scheduler

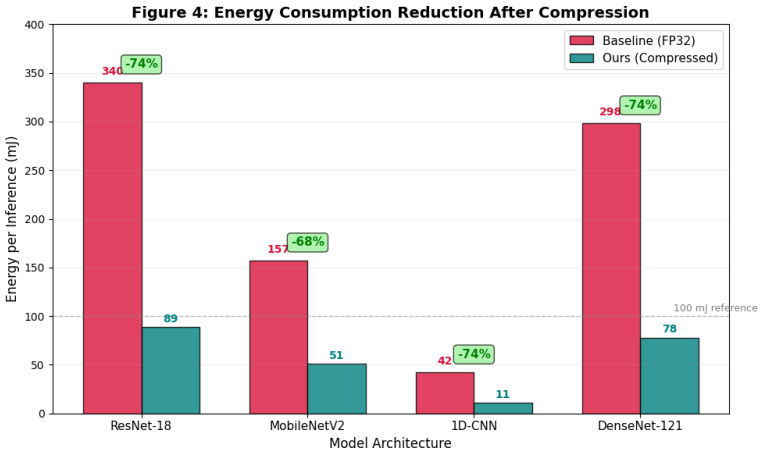

Fig. 4. Energy consumption before and after compression.

for memory, CPU, and battery constraints; (iii) extensive validation on three IoT tasks and three edge platforms demonstrating real-time feasibility (¡100 ms latency) and 74% energy reduction.

6.2 Limitations and Future Work

Limitations include support for only convolutional architectures (transformers not yet supported) and heuristic-based scheduling. Future work includes: (i) extension to transformer-based models for IoT time-series data; (ii) on-device incremental compression for continuous learning; (iii) learning-based adaptive scheduling using reinforcement learning; (iv) federated compression across mul- tiple edge devices without sharing raw data.

References

1. Usupova, E., Khan, A.: Optimizing ML Training with Perturbed Equations. In: 2025 6th International Conference on Problems of Cybernetics and Informatics (PCI), Baku, Azerbaijan, pp. 1–6 (2025). doi: 10.1109/PCI66488.2025.11219819
2. Rakimbekuulu, S., Shambetaliev, K., Esenalieva, G., Khan, A.: Code Generation for Ablation Technique. In: 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, pp. 1–7 (2024). doi: 10.1109/EWDTS63723.2024.10873640

3. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 (2017)
4. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning Filters for Efficient ConvNets. In: International Conference on Learning Representations (ICLR) (2021)
5. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2704–2713 (2018)
6. Hinton, G., Vinyals, O., Dean, J.: Distilling the Knowledge in a Neural Network. In: NIPS Deep Learning and Representation Learning Workshop (2015)
7. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In: International Conference on Learning Representations (ICLR) (2019)
8. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In: European Conference on Computer Vision (ECCV), pp. 285–300 (2019)
9. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 815–824 (2018)
10. Lan, G., Liu, J., Liu, Y., Li, X., Liu, K., Mao, Z.H.: Edge Intelligence: A Survey. IEEE Internet of Things Journal, 7(8), 7428–7442 (2020)