

# EfficientQuant: An Efficient Post-Training Quantization for CNN-Transformer Hybrid Models on Edge Devices

Shaibal Saha   Lanyu Xu  
Oakland University  
Rochester Hills, MI, USA  
{shaibalsaha, lxu}@oakland.edu

## Abstract

Hybrid models that combine convolutional and transformer blocks offer strong performance in computer vision (CV) tasks but are resource-intensive for edge deployment. Although post-training quantization (PTQ) can help reduce resource demand, its application to hybrid models remains limited. We propose EfficientQuant, a novel structure-aware PTQ approach that applies uniform quantization to convolutional blocks and  $\log_2$  quantization to transformer blocks. EfficientQuant achieves  $2.5\times$ – $8.7\times$  latency reduction with minimal accuracy loss on ImageNet-1K dataset. It further demonstrates low latency and memory efficiency on edge devices, making it practical for real-world deployment.

## 1. Introduction

The rapid advancements of computer vision (CV) tasks have been driven by convolutional neural networks (CNNs) and, more recently, transformers. While CNNs excel at capturing local features, they struggle with global context [13, 23, 26]. Transformers address this through self-attention [6], but at the cost of high computational complexity [20, 24].

To combine the strengths of both architectures, hybrid models have emerged, integrating convolutional and transformer blocks [16, 17, 21, 22]. Figure 1a illustrates architectures with convolutional layers followed by transformer encoders [18], whereas Figure 1b uses transformer encoders with convolutional decoders [9]. While demonstrating strong performance, hybrid models remain computationally demanding and challenging to deploy on resource-constrained edge devices [2].

To enable the deployment of hybrid models on edge devices, quantization has emerged as a promising compression technique. While quantization-aware training (QAT) can reduce accuracy loss [1, 3, 7, 11], its high retraining cost makes post-training quantization (PTQ) more practical for real-time inference [4, 10, 15, 31]. However, existing PTQ

methods designed for CNNs or transformers often struggle to generalize to hybrid architectures [14].

Recent PTQ methods for hybrid models attempt to bridge the gap but face limitations in generalizability, hardware efficiency, and reliance on large calibration sets. Q-HyViT [14] selected granularity and schemes via Hessian-based analysis, but its mixed strategies are not edge-friendly. HyQ [12] proposed quantization-aware distribution scaling (QADS) for convolutional blocks and an integer-only softmax for transformers, yet lack evaluation on resource-constrained edge devices such as Jetson Nano. In contrast, we observe that weights in convolutional block follow a uniform distribution (see Figure 2), while transformer post-Softmax activations follow a power-law pattern (Figure 3), motivating our use of uniform and  $\log_2$ -based quantization, respectively—both efficient for edge deployment [8, 29]. Quantizing both weights and activations in convolutional blocks degrades accuracy, so we apply weight-only quantization. In transformers, weight quantization causes high performance loss, whereas post-Softmax activations tolerate quantization well. Thus, we quantize weights in convolutional blocks and activations in transformer blocks. Based on this, we choose weight quantization for convolutional blocks and activation quantization for transformer blocks. Based on these observations, we propose EfficientQuant, a structure-aware PTQ method that automatically identifies and quantizes convolutional and transformer blocks in hybrid models. EfficientQuant achieved significant latency improvements and low memory usage on edge devices such as Jetson Nano AGX Xavier, making it practical for real-world deployment. Our main contributions are as follows:

- We propose a structure-aware algorithm to identify convolutional and transformer blocks in hybrid models.
- We introduce EfficientQuant, a PTQ method that applies uniform quantization to convolutional blocks and  $\log_2$ -based quantization to transformer activations, tailored to their distinct characteristics.
- EfficientQuant achieves lower latency than existing PTQ methods with acceptable accuracy drop and evaluates Effi-

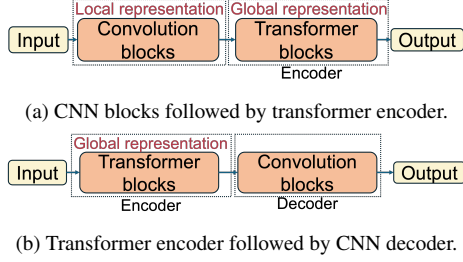


Figure 1. High-level structures of hybrid models [9, 18, 21].

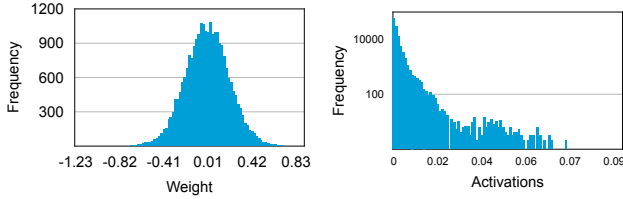


Figure 2. Weight distribution of a convolutional block on MobileViT\_s model. Figure 3. Post-Softmax activation distribution of a transformer block on MobileViT\_s model.

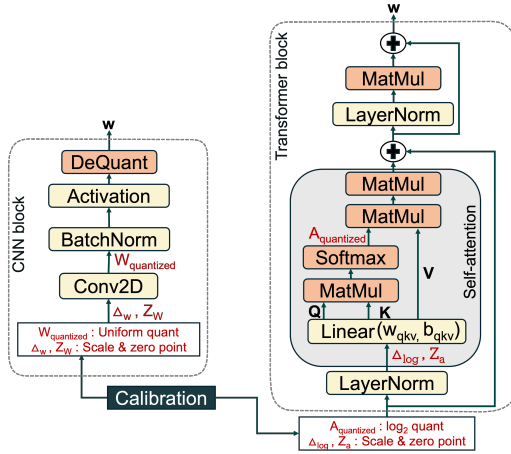


Figure 4. The overview of the EfficientQuant to perform uniform and  $\log_2$  quantization for convolution and transformer blocks in the hybrid model.

cientQuant on edge devices such as Jetson Nano and AGX Xavier.

## 2. EfficientQuant: PTQ for Hybrid Models

**Overview** Figure 4 presents an overview of EfficientQuant. The following sections detail the structure-aware algorithm, uniform quantization for convolution blocks, and  $\log_2$  quantization for transformer blocks.

### 2.1. Structure-Aware Block Identification

We develop a specialized structure-aware algorithm to automatically identify and classify blocks as CNN or transformer within hybrid models, enabling efficient quantization. The algorithm traverses the model using a depth-first search (DFS) with a queue (Q) to systematically explore the architecture (see Algorithm 1). At each step, it dequeues a module, and its parent identifier, iterates through its sub-blocks, and constructs full identifiers by combining parent and child names. The algorithm returns two lists—*CNN* and *Transformer*—to guide type-specific quantization. The time complexity of the proposed algorithm is  $\mathcal{O}(n)$ . Although implemented in PyTorch, it can be adapted to frameworks like TensorFlow or Keras with minimal adjustments in block-type declarations.

#### Algorithm 1: Structure-Aware Block Identification

```

Input: Model  $M$ 
Output: Lists  $CNN$ ,  $Transformer$ 
Initialize  $Q$ , lists  $CNN$ ,  $Transformer$ ;
Enqueue ( $M$ , "") into  $Q$ ;
while  $Q$  is not empty do
    /* Dequeue the last element from the queue */
    ( $module$ ,  $parent\_name$ )  $\leftarrow$  dequeue  $Q$ ;
    foreach ( $layer\_name$ ,  $layer$ ) in  $module.children()$  do
         $fullname \leftarrow parent\_name + layer\_name$ ;
        if  $layer$  is Conv2d then
            Append  $fullname$  to  $CNN$ ;
        else if  $layer$  is Linear, "attn", "Attention", or "Transformer" then
            Append  $fullname$  to  $Transformer$ ;
        Enqueue ( $layer$ ,  $fullname$ ) into  $Q$ ;
return  $CNN$ ,  $Transformer$ ;

```

### 2.2. Uniform Quantization for Convolution Blocks

As we illustrated in Figure 2, the weights in convolution block exhibit uniform distributions. We apply uniform quantization to convolution blocks (stored in the list *CNN*), using fixed scaling factors derived from the block-specific min-max statistics ( $[W_{\min}, W_{\max}]$ ) collected during calibration as like Equation 1.

$$\Delta_w = \frac{W_{\max} - W_{\min}}{2^b - 1}, \quad Z_w = \text{round} \left( \frac{-W_{\min}}{\Delta_w} \right) \quad (1)$$

$$W_{\text{quantized}} = \text{clamp} \left( \left\lfloor \frac{W - W_{\min}}{\Delta_w} + Z_w \right\rfloor \right) \quad (2)$$

$$W_{\text{dequantized}} = (W_{\text{quantized}} - Z_w) \times \Delta_w \quad (3)$$

In Equation 2,  $W_{\text{quantized}}$  represents 8-bit convolutional weights quantized from FP32 inputs  $W \in \mathbb{R}^{\alpha \times \beta \times \gamma}$ , where

$\alpha$  is the number of channels and  $\beta \times \gamma$  are spatial dimensions. The scaling factor  $\Delta_W$  maps values from  $[W_{\min}, W_{\max}]$  to  $[0, 255]$ , and the zero point  $Z_W$  (From Equation 1) aligns the minimum value. Quantized weights are then clamped to the valid  $b$  – bit range (in our method  $b = 8$ ).

Once the model processes inputs through quantized blocks, weights are restored to floating-point precision during inference for meaningful interpretation. We wrapped the model with conditional dequantization specifically for convolutional blocks, using stored  $W_{\min}$  and  $W_{\max}$  along with the scaling factor and zero point to calculate the dequantized weights  $W_{\text{dequantized}}$  using Equation 3.

### 2.3. $\log_2$ Quantization for Transformer blocks

Based on the understanding from Figure 3, we apply  $\log_2$  quantization to post-Softmax activation in the self-attention mechanisms on transformer blocks (listed in *Transformer*), mapping values onto a logarithmic scale. This preserves small yet critical activations, achieving better precision than traditional methods.

We target 8-bit quantization, using the same clamping range  $([0, 255])$  as in uniform quantization. Since logarithmic quantization cannot directly handle zero values ( $\log(0) = -\infty$ ), we introduce a small constant  $\epsilon = 1e^{-5}$  to ensure numerical stability, effectively handling tensors containing zero or very small values.

Since we apply  $\log_2$  quantization on post-Softmax activations ( $0 \leq \text{activation} \leq 1$ ), we only need to handle positive and zero values in transformer blocks. Our quantization approach involves three main components in the transformer blocks: calculating the quantization scaling factor  $\Delta_{\log}$ , determining the zero point  $Z_a$ , and applying logbas-based quantization with clamping. First, the scaling factor  $\Delta_{\log}$  determines the intervals for quantizing the values based on the statistics during calibration. Then, we calculate the  $Z_a$ , which adjusts the quantized values to align with the minimum observed value, ensuring both small and large values are represented within the quantization range. To determine the quantization parameters in the log domain, we first compute:

$$A_{\min} = \log_2(a_{\min}), \quad A_{\max} = \log_2(a_{\max})$$

Here,  $a_{\min}$  and  $a_{\max}$  are the minimum and maximum activation values observed during calibration. Based on the  $\Delta_{\log}$  and  $Z_a$ , we apply  $\log_2$  quantization on the post-Softmax activation  $a$ . Finally, the clamp function ensures that the quantized values  $A_{\text{quantized}}$  remain within the valid b-bit range  $[0, 2^b - 1]$  (see Equation 5).

$$\Delta_{\log} = \frac{A_{\max} - A_{\min}}{2^b - 1}, \quad Z_a = \text{round}\left(\frac{-A_{\min}}{\Delta_{\log}}\right) \quad (4)$$

$$A_{\text{quantized}} = \text{clamp}\left(\left\lfloor \frac{-\log_2(a+\epsilon)}{\Delta_{\log}} + Z_a \right\rfloor, 0, 2^b - 1\right) \quad (5)$$

Where  $A_{\text{quantized}} \in \mathbb{R}^{\alpha \times \beta \times \gamma}$  denotes the b-bit integer value;  $\alpha$  is the number of channels and  $\beta \times \gamma$  is the height and width of the input. The dequantizer  $A_{\text{dequantized}}$  can be defined using Equation 6.

$$A_{\text{dequantized}} = 2^{(A_{\text{quantized}} - Z_a) \cdot \Delta_{\log}} \quad (6)$$

## 3. Experiment

### 3.1. Implementation

**Models and Datasets** We evaluate EfficientQuant using MobileViT and MobileViTv2 variants on the ImageNet-1K [5] validation set. We conduct comprehensive comparisons against state-of-the-art (SOTA) PTQ methods, including EasyQuant [28], PTQ4ViT [30], RepQ-ViT [19], Q-HyViT [14], and HyQ [12], evaluating our method in terms of accuracy, latency, and memory consumption.

**Hardware and Libraries** We deploy MobileViT and MobileViTv2 on RTX 3080, AGX Xavier, and Jetson Nano to evaluate EfficientQuant in terms of accuracy, memory consumption, and latency. EfficientQuant is implemented in PyTorch and accelerated with TensorRT [25] on edge devices. We use pretrained weights from timm [27] library.

**Experiment Settings** We use batch size 32 for calibration and batch size 1 for edge device inference. For  $\log_2$  quantization, we set  $\epsilon = 1e^{-5}$  based on trial and error. Smaller  $\epsilon$  increases quantization errors, while larger  $\epsilon$  distorts small values which makes  $10^{-5}$  the optimal compromise.

### 3.2. Comparison with SOTA PTQ Approaches

**Accuracy** Table 1 compares the top-1 accuracy of EfficientQuant with PTQ methods designed for CNNs (EasyQuant), ViTs (PTQ4ViT, RepQ-ViT), and hybrid models (Q-HyViT, HyQ) on the ImageNet-1K dataset. The results for EasyQuant, PTQ4ViT, and RepQ-ViT are adopted from Q-HyViT and show significant accuracy degradation when applied to hybrid models due to architecture-specific design. Our implementation of Q-HyViT on RTX 3080 yields lower accuracy than originally reported, while HyQ closely matches its original results. Although HyQ achieves the highest top-1 accuracy overall, it does not support complex MobileViTv2 variants (MobileViTv2\_100 to MobileViTv2\_200). In contrast, EfficientQuant maintains stable accuracy across all models, with slightly higher degradation in smaller variants (e.g., 6.86% on MobileViTv2\_050) and minimal degradation in larger variants (e.g., 0.69% on MobileViTv2\_175). In some scenarios like MobileViT\_s, EfficientQuant performs comparably to Q-HyViT. These results highlight EfficientQuant robustness and scalability across diverse hybrid architectures.

**Latency** EfficientQuant outperforms Q-HyViT and HyQ in inference latency on RTX 3080 across all MobileViT variants. For fairness, we re-evaluated all methods on the RTX 3080 GPU. As shown in Figure 5, HyQ supports only smaller

Table 1. Top-1 (%) accuracy of hybrid models on ImageNet-1K [5] after applying PTQ methods originally proposed for pure CNNs, Vision Transformers (ViTs), and hybrid models. ‘-’ indicates unavailable data.

Models	FP32 (%)	PTQ for Pure CNN/ ViTs (%)			PTQ For Hybrid Models (%)				
		EasyQuant [28]	PTQ4ViT [30]	RepQ-ViT [19]	Q-HyViT [14]	Q-HyViT (Reproduced)	HyQ [12]	HyQ (Reproduced)	EfficientQuant (Ours)
MobileViT_xxs	68.91	36.13	37.75	1.85	68.20	62.73	68.15	67.94	66.86
MobileViT_xs	74.62	73.16	65.52	41.96	74.31	69.37	73.99	73.99	73.8
MobileViT_s	78.31	74.21	68.19	59.01	77.92	75.71	77.93	77.93	77.87
MobileViTv2_050	70.1	66.80	39.39	26.60	69.89	64.99	69.16	-	65.59
MobileViTv2_075	75.57	62.91	65.54	55.52	75.29	69.81	74.47	-	72.50
MobileViTv2_100	77.9	69.34	51.02	40.61	77.63	69.66	-	-	72.03
MobileViTv2_125	79.64	77.31	67.39	41.65	79.31	71.72	-	-	77.58
MobileViTv2_150	80.36	75.83	68.61	62.12	79.97	73.09	-	-	79.08
MobileViTv2_175	80.86	79.93	72.30	63.52	80.45	75.20	-	-	80.30
MobileViTv2_200	81.12	80.04	75.50	64.65	80.76	75.85	-	-	80.48

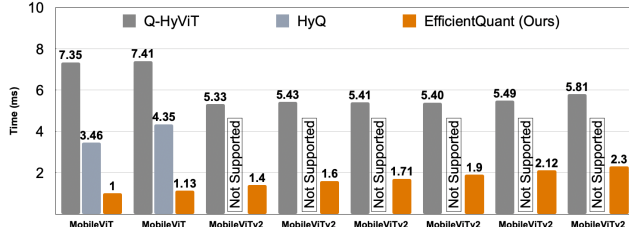


Figure 5. EfficientQuant achieves better latency on RTX 3080 GPU than the SOTA techniques on MobileViT and MobileViTv2 models.

models, while EfficientQuant scales to the full MobileViTv2 range. Notably, EfficientQuant achieves a  $2.5\times$  -  $8.7\times$  latency reduction over Q-HyViT, with MobileViTv2\_175 running at **2.36 ms** vs **5.81 ms**. Compared to HyQ, EfficientQuant delivers up to  $3.85\times$  faster inference (e.g., **1.13 ms** vs. **4.35 ms** on MobileViT\_s) while maintaining comparable accuracy. These results highlight EfficientQuant’s efficiency, scalability, and practicality for edge deployments.

### 3.3. Evaluation with Edge Devices

**Latency** We evaluate EfficientQuant on edge devices based on TensorRT engine load and inference time. Since no prior work has explored hybrid model deployment on the edge device, baseline comparisons are not available. Figure 6 illustrates the engine load time for MobileViT variants. The average model load times for the RTX 3080, AGX Xavier, and Jetson Nano are 187.4 ms, 320.2 ms, and 1477.0 ms, respectively, which are acceptable for edge deployment. Figure 7 presents inference time comparisons for EfficientQuant across RTX 3080, AGX Xavier, and Jetson Nano. Jetson Nano exhibits significantly higher latency, achieving  $3\times$  to  $5\times$  slower than AGX Xavier and up to  $63\times$  slower than RTX 3080. The average inference times—1.51 ms on RTX 3080, 40.7 ms on AGX Xavier, and 17.8 ms on Jetson Nano—demonstrate acceptable latency for edge deployment. Based on our observations, Jetson Nano is better suited for static deployments given its higher load times. In contrast, AGX Xavier offers a balanced trade-off between efficiency

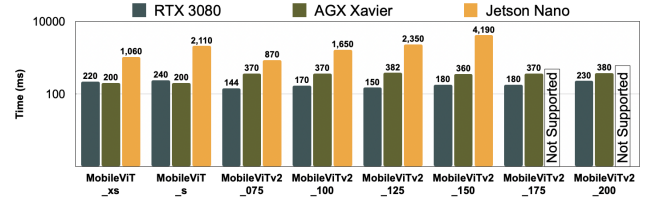


Figure 6. Load time of MobileViT models on edge devices with EfficientQuant using TensorRT engine.

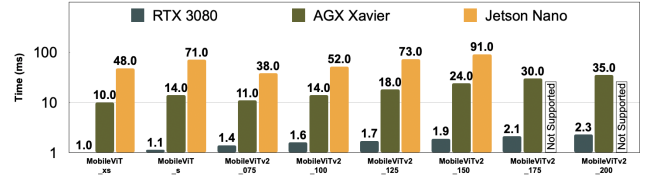


Figure 7. Inference time of MobileViT models on edge devices with EfficientQuant using TensorRT engine.

and flexibility, making it well-suited for CV tasks.

**Memory** We also measure the maximum memory usage during the inference times for all three devices. We utilize *pynvml* for RTX 3080 and *tegrastats* library for AGX Xavier and Jetson Nano for measuring memory usage. Each validation set was run ten times to obtain an average maximum memory usage per model. Results indicated slight variations among the devices, with average memory usage being 2,568 MiB for the RTX 3080, 2,344 MiB for AGX Xavier, and 2,265 MiB for the Jetson Nano. The overall differences in memory usage are  $\sim 2\%$  because of different memory management across those devices.

## 4. Conclusion

In this study, we proposed EfficientQuant, a novel structure-aware PTQ method for hybrid models that automatically identifies block types to enable targeted quantization. Leveraging the observed distribution patterns in convolutional and transformer blocks, EfficientQuant applies uniform and  $\log_2$  quantization accordingly. It achieves up to  $8.7\times$  la-



tency reduction with minimal accuracy loss. Evaluations on RTX 3080, AGX Xavier, and Jetson Nano validate the efficiency and adaptability of EfficientQuant, highlighting its suitability for low-latency deployment on a wide range of resource-constrained edge devices.

## Acknowledgements

This work was supported in part by the U.S. National Science Foundation under Grant No. 2245729, and by the University Research Committee Faculty Research Fellowship at Oakland University.

## References

- [1] Thomas Avé, Kevin Mets, Tom De Schepper, and Steven Latré. Quantization-aware policy distillation (qpd). In *Deep Reinforcement Learning Workshop, NeurIPS 2022, 9 December, 2022*, pages 1–15, 2022. [1](#)
- [2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. [1](#)
- [3] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. [1](#)
- [4] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019. [1](#)
- [5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. [3, 4](#)
- [6] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. [1](#)
- [7] Amir Gholami, Schoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022. [1](#)
- [8] Hai Victor Habi, Reuven Peretz, Elad Cohen, Lior Dikstein, Oranit Dror, Idit Diamant, Roy H Jennings, and Arnon Netzer. Hptq: Hardware-friendly post training quantization. *arXiv preprint arXiv:2109.09113*, 2021. [1](#)
- [9] Ali Hatamizadeh, Yucheng Tang, Vishwesh Nath, Dong Yang, Andriy Myronenko, Bennett Landman, Holger R Roth, and Daguang Xu. Unetr: Transformers for 3d medical image segmentation. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 574–584, 2022. [1, 2](#)
- [10] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018. [1](#)
- [11] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4350–4359, 2019. [1](#)
- [12] Nam Joon Kim, Jongho Lee, and Hyun Kim. Hyq: Hardware-friendly post-training quantization for cnn-transformer hybrid networks. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 4291–4299. International Joint Conferences on Artificial Intelligence Organization, 2024. Main Track. [1, 3, 4](#)
- [13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. [1](#)
- [14] Jemin Lee, Yongin Kwon, Sihyeong Park, Misun Yu, Jeman Park, and Hwanjun Song. Q-hyvit: Post-training quantization of hybrid vision transformers with bridge block reconstruction for iot systems. *IEEE Internet of Things Journal*, 2024. [1, 3, 4](#)
- [15] Jun Haeng Lee, Sangwon Ha, Saerom Choi, Won-Jo Lee, and Seungwon Lee. Quantization for rapid deployment of deep neural networks. *arXiv preprint arXiv:1810.05488*, 2018. [1](#)
- [16] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *Advances in Neural Information Processing Systems*, 35:12934–12949, 2022. [1](#)
- [17] Yanyu Li, Ju Hu, Yang Wen, Georgios Evangelidis, Kamyar Salahi, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Re-thinking vision transformers for mobilenet size and speed. In *Proceedings of the IEEE international conference on computer vision*, 2023. [1](#)
- [18] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. Bevformer: Learning bird’s-eye-view representation from multi-camera images via spatiotemporal transformers. In *European conference on computer vision*, pages 1–18. Springer, 2022. [1, 2](#)
- [19] Zhikai Li, Junrui Xiao, Lianwei Yang, and Qingyi Gu. Repqv: Scale reparameterization for post-training quantization of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17227–17236, 2023. [3, 4](#)
- [20] Muhammad Maaz, Abdelrahman Shaker, Hisham Cholakkal, Salman Khan, Syed Waqas Zamir, Rao Muhammad Anwer, and Fahad Shahbaz Khan. Edgenext: efficiently amalgamated cnn-transformer architecture for mobile vision applications. In *European conference on computer vision*, pages 3–20. Springer, 2022. [1](#)
- [21] Sachin Mehta and Mohammad Rastegari. Mobilevit: lightweight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021. [1, 2](#)
- [22] Sachin Mehta and Mohammad Rastegari. Separable self-attention for mobile vision transformers. *arXiv preprint arXiv:2206.02680*, 2022. [1](#)

- [23] Muhammad Muzammal Naseer, Kanchana Ranasinghe, Salman H Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Intriguing properties of vision transformers. *Advances in Neural Information Processing Systems*, 34: 23296–23308, 2021. 1
- [24] Duy Thanh Nguyen, Nguyen Huy Hung, Hyun Kim, and Hyuk-Jae Lee. An approximate memory architecture for energy saving in deep learning applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(5):1588–1601, 2020. 1
- [25] NVIDIA. TensorRT. <https://developer.nvidia.com/tensorrt/>, 2021. Retrieved July 1, 2021. 3
- [26] Shaibal Saha and Lanyu Xu. Vision transformers on the edge: A comprehensive survey of model compression and acceleration strategies. *Neurocomputing*, page 130417, 2025. 1
- [27] Ross Wightman. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019. 3
- [28] Di Wu, Qi Tang, Yongle Zhao, Ming Zhang, Ying Fu, and Debing Zhang. Easyquant: Post-training quantization via scale optimization. *arXiv preprint arXiv:2006.16669*, 2020. 3, 4
- [29] Zhuguanyu Wu, Jiaxin Chen, Hanwen Zhong, Di Huang, and Yunhong Wang. Adalog: Post-training quantization for vision transformers with adaptive logarithm quantizer. *arXiv preprint arXiv:2407.12951*, 2024. 1
- [30] Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization. In *European conference on computer vision*, pages 191–207. Springer, 2022. 3, 4
- [31] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR, 2019. 1