DQT: Dynamic Quantization Training via Dequantization-Free Nested Integer Arithmetic

Hazem Hesham Yousef Shalby[1], **Fabrizio Pittorino**[1], **Francesca Palermo**[2], **Diana Trojaniello**[2], **and Manuel Roveri**[1]

[1]Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy
[2]EssilorLuxottica Smart Eyewear Lab, Milan, Italy
hazemhesham.shalby@polimi.it, fabrizio.pittorino@polimi.it

Abstract

The deployment of deep neural networks on resource-constrained devices relies on quantization. While static, uniform quantization applies a fixed bit-width to all inputs, it fails to adapt to their varying complexity. Dynamic, instance-based mixed-precision quantization promises a superior accuracy-efficiency trade-off by allocating higher precision only when needed. However, a critical bottleneck remains: existing methods require a costly dequantize-to-float and requantize-to-integer cycle to change precision, breaking the integer-only hardware paradigm and compromising performance gains. This paper introduces Dynamic Quantization Training (DQT), a novel framework that removes this bottleneck. At the core of DQT is a nested integer representation where lower-precision values are bit-wise embedded within higher-precision ones. This design, coupled with custom integer-only arithmetic, allows for on-the-fly bit-width switching through a near-zero-cost bit-shift operation. This makes DQT the first quantization framework to enable both dequantization-free static mixed-precision of the backbone network, and truly efficient dynamic, instance-based quantization through a lightweight controller that decides at runtime how to quantize each layer. We demonstrate DQT state-of-the-art performance on ResNet18 on CIFAR-10 and ResNet50 on ImageNet. On ImageNet, our 4-bit dynamic ResNet50 achieves 77.00% top-1 accuracy, an improvement over leading static (LSQ, 76.70%) and dynamic (DQNET, 76.94%) methods at a comparable BitOPs budget. Crucially, DQT achieves this with a bit-width transition cost of only 28.3M simple bit-shift operations, a drastic improvement over the 56.6M costly Multiply-Accumulate (MAC) floating-point operations required by previous dynamic approaches - unlocking a new frontier in efficient, adaptive AI.

Extended version — https://arxiv.org/abs/2508.09176

1 Introduction

The deployment of deep neural networks (DNNs) on resource-constrained hardware is a standard practice, enabled by quantization techniques that represent weights and activations with low-precision integers (Nagel et al. 2021; Jacob, Kligys, and al. 2018). The dominant paradigm is static quantization, where a single, fixed bit-width (e.g., INT8) is used for inference on all inputs. This approach is suboptimal in resource-constrained settings as it allocates a fixed computational budget to every input, regardless of its intrinsic complexity.

Dynamic, instance-aware quantization addresses this limitation by adapting the network precision at runtime based on input characteristics (Liu et al. 2022). This method can achieve a superior accuracy-efficiency trade-off by allocating greater computational resources only to more challenging inputs. However, a critical and unresolved performance bottleneck has prevented its adoption. Existing dynamic methods depend on floating-point arithmetic to change precision, requiring a computationally expensive cycle of dequantization to FP32 followed by requantization to a target integer format INT(b) for each layer and input (Xu et al. 2018). This reliance on floating-point operations breaks the efficient integer-only execution model of modern accelerators and introduces an overhead that undermines the intended efficiency gains.

This paper introduces a general framework to eliminate this dequantization bottleneck. Our core contribution is a nested integer representation where a lower-precision integer is a bit-wise truncation of its higher-precision counterpart. This property, combined with a co-designed set of integer-only arithmetic operators, enables bit-width transitions from a high precision b_1 to a lower precision b_2 via a single logical right bit-shift operation: $q_{b_2} \equiv q_{b_1} \gg (b_1 - b_2)$. This mechanism replaces the expensive floating-point conversion cycle entirely. While this dequantization-free approach provides efficiency benefits for static mixed-precision and standard quantized networks, its impact is most significant in the dynamic, instance-aware setting.

We apply this general framework to create Dynamic Quantization Training (DQT), a complete end-to-end pipeline for training and deploying truly efficient adaptive models. In the DQT architecture, a lightweight controller adaptively selects per-layer bit-widths for each input. At inference, the model loads high-precision weights once and uses bit-shifting to instantiate the desired precision on-the-fly. The entire forward pass, including all precision adjustments, is executed using our custom integer arithmetic, maintaining a pure integer dataflow.

We validate DQT on the CIFAR-10 and ImageNet classification benchmarks. Our models establish a new state-of-the-art (SotA) regarding the accuracy-efficiency trade-off. For instance, our DQT-trained ResNet-50 operates at a computational budget lower than a fixed 4-bit network while achieving 77.00% top-1 accuracy on ImageNet. By solving the dequantization bottleneck, DQT makes instance-aware quantization a practical and powerful tool for efficient AI. Our main contributions are:

1. A general quantization framework based on a nested integer representation and custom integer-only arithmetic. This framework eliminates the floating-point conversion bottleneck common to all mixed-precision systems and enables bit-width changes via low-cost logical shift operations.
2. The first application of this framework to develop an end-to-end training and inference pipeline, DQT, for dynamic, instance-aware networks. We demonstrate that by removing the dequantization overhead, DQT makes adaptive inference computationally practical and achieves state-of-the-art results.

The rest of the paper is organized as follows. We first review prior work and provide background on uniform quantization. We then introduce our DQT framework and detail its core technical contribution. Subsequently, we describe the training and inference procedure, followed by extensive experimental results on CIFAR-10 and ImageNet. We conclude with a summary of our findings and directions for future work.

2 Related Literature

Our work is positioned at the intersection of static quantization, dynamic instance-aware quantization, and efficient integer arithmetic.

Static Quantization. The most common approach to model compression is static, uniform quantization, where weights and activations are converted to a fixed low bit-width (e.g., INT8) either after training (Post-Training Quantization, PTQ) or during it (Quantization-Aware Training, QAT) (Nagel et al. 2021; Jacob, Kligys, and al. 2018). QAT is the standard for high-performance models, as it allows the network to compensate for quantization error. To improve upon uniform precision, static mixed-precision quantization assigns different bit-widths to different layers based on their sensitivity (Dong et al. 2019; Wang et al. 2019). While these methods offer a better accuracy-efficiency trade-off, they remain fundamentally static; the computational cost is fixed for all inputs.

Dynamic, Instance-Aware Quantization. To address the rigidity of static methods, dynamic instance-aware quantization aims to adapt the computational cost to each input sample. Approaches in this area typically use a lightweight controller or policy network to select bit-widths at runtime. For instance, early work used reinforcement learning to select layer precisions (Xu et al. 2018), while more recent methods like DQNet (Liu et al. 2022) and AdaBit (Jin, Yang, and Liao 2020) employ small controllers trained jointly with the main network. These methods successfully demonstrate that adapting precision to input difficulty is feasible.

However, this entire class of methods is constrained by a critical performance issue: the dequantization bottleneck. To adjust a layer precision, these frameworks must dequantize the integer tensor to a 32-bit floating-point representation and subsequently requantize it to the new target bit-width. This dequantize-requantize cycle, performed per-layer and per-input, introduces substantial computational and memory-access overhead. This reliance on floating-point arithmetic breaks the optimized integer-only dataflow of modern accelerators, and the resulting overhead undermines the efficiency gains that dynamic precision is intended to provide. Consequently, instance-aware quantization has remained a largely theoretical exercise rather than a practical, deployable solution.

Nested Quantization and Integer Arithmetic. Recognizing this efficiency challenge, some research has explored alternative quantization schemes. Matryoshka Quantization (MatQuant) (Nair et al. 2025) introduced a nested quantization scheme where low-bit representations are embedded within high-bit ones. However, the goal of MatQuant is to extract multiple, independent static models of varying precisions from a single trained model. It relies on standard *fake quantization* during training and does not provide the integer-only arithmetic necessary to perform dynamic, on-the-fly precision adjustments within a single inference pass.

Our work is the first to bridge this gap. We extend the nested representation concept by introducing a co-designed set of integer-only arithmetic operators compatible with bit-shift-based precision changes. This combination eliminates the dequantization bottleneck entirely, finally making efficient, instance-aware dynamic inference practical.

3 Background: The Dequantization Bottleneck

To formalize the problem our framework solves, we first review the principles of standard uniform quantization and then detail the computational bottleneck that arises when applying it in a dynamic, mixed-precision context.

Uniform quantization maps a real-valued tensor x from a clipping range $[m, M]$ to a b-bit integer tensor using the quantization function $Q^b(\cdot)$:

$$Q^b(x) = \text{clip}\left(\left\lfloor \frac{x-m}{\Delta} \right\rceil, 0, 2^b - 1\right), \; \Delta = \frac{M-m}{2^b - 1} \quad (1)$$

with Δ being the quantization step. The corresponding dequantization $D(\cdot)$ function reconstructs the real value $\hat{x}$:

$$\hat{x} = D(x) = Q^b(x) \cdot \Delta + m \quad (2)$$

The quantization error, $|x - \hat{x}|$, is bounded by the step size Δ and is the primary source of accuracy degradation in quantized models.

The rounding and clipping operations in Equation (1) are non-differentiable, which prevents gradient-based optimization. Quantization-Aware Training (QAT) circumvents this issue using the Straight-Through Estimator (STE) (Bengio 2013). In the forward pass, a *fake quantization* step is performed where a tensor is quantized and immediately dequantized ($x \rightarrow Q^b(x) \rightarrow \hat{x}$), simulating the precision loss.

In the backward pass, the STE approximates the gradient of the quantization function as an identity, i.e., $\frac{\partial \hat{x}}{\partial x} = 1$. This allows gradients to flow through the quantization node to update the underlying full-precision weights.

The standard QAT process is designed for a single, static bit-width b. This paradigm is computationally inefficient for dynamic, instance-aware models that must switch between different bit-widths at runtime. Consider changing a tensor precision from b_1 to b_2, with corresponding quantization parameters (Δ_1, m_1) and (Δ_2, m_2). In the standard framework, this requires a full dequantize-requantize cycle, as illustrated in Figure 1. First, the tensor must be dequantized to a 32-bit floating-point representation using its current parameters: $\hat{x} = Q^{b_1}(x) \cdot \Delta_1 - m_1$. Then, it must be requantized to the new target precision using the new parameters: $Q^{b_2}(\hat{x}) = \text{clip}\left(\left\lfloor \frac{\hat{x}-m_2}{\Delta_2} \right\rceil, 0, 2^{b_2} - 1\right)$. This dequantize-requantize cycle is the fundamental bottleneck. It requires performing expensive floating-point operations for every layer whose precision is changed, for every input sample. This computational overhead breaks the integer-only dataflow on hardware accelerators and compromises the performance benefits of using lower-precision integers. Our work eliminates this bottleneck entirely.

4 DQT Dequantization-Free Quantization Framework

Quantized integer arithmetic is inconsistent with quantized floating-point results, i.e., $Q(x_1) \bullet Q(x_2) \neq Q(x_1 \bullet x_2)$, requiring dequantization. We eliminate this bottleneck via a novel framework built on two core principles: a nested integer representation that enables precision changes via bit-shifting, and custom bit-shift-compatible integer operators compatible with this representation.

The central mechanism of our framework is a nested quantization scheme where lower-precision representations are bit-wise embedded within higher-precision ones. We define a master bit-width, n, and a corresponding master scale, Δ_n. The quantization scale for any other bit-width, $b < n$, is defined by a power-of-two relationship:

$$\Delta_b = \Delta_n \cdot 2^{n-b} \tag{3}$$

This specific constraint on the quantization scales allows for a direct, dequantization-free conversion from a high-precision integer $Q^n(x)$ to a lower-precision integer $Q^b(x)$ via a single logical right bit-shift:

$$\begin{aligned} Q^b_{\text{DQT}}(x) &= \text{clip}\left(\left\lfloor \frac{Q^n(x)}{2^{n-b}} \right\rceil, 0, 2^b - 1\right) \\ &= Q^n(x) \gg (n-b) + \epsilon \end{aligned} \tag{4}$$

where ϵ denotes the approximation error of the bit-shift operation and is analyzed in Appendix C. This operation is computationally inexpensive and avoids any floating-point arithmetic. To leverage this property throughout a network, all arithmetic operations must be reformulated to operate directly on these integer representations and their corresponding scales.

We now present the integer-only arithmetic for addition and multiplication that is compatible with our nested quantization scheme. Let two real values, x_1 and x_2, be quantized to integers q_1 and q_2 with parameters (Δ_1, m_1) and (Δ_2, m_2) respectively, where Δ and m follow the definitions from Section 3. The output of an operation, $y = x_1 \bullet x_2$, will be quantized with parameters (Δ_y, m_y). The dequantized value of an input q is $\hat{x} = q \cdot \Delta + m$. The goal is to compute the quantized output $q_y = \lfloor \frac{y - m_y}{\Delta_y} \rceil$ using only integer operations.

Integer Addition ($\oplus$): For addition, $y = x_1 + x_2$, the exact quantized output can be reformulated into a pure integer form:

$$q_1 \oplus q_2 = k_1 q_1 + k_2 q_2 + k_3 \tag{5}$$

where the scaling factors k_i are pre-computed numbers:

$$k_1 = \left\lfloor \frac{\Delta_1}{\Delta_y} \right\rceil, \quad k_2 = \left\lfloor \frac{\Delta_2}{\Delta_y} \right\rceil, \quad k_3 = \left\lfloor \frac{m_1 + m_2 - m_y}{\Delta_y} \right\rceil \tag{6}$$

The round operation $\lfloor \cdot \rceil$ on k_i allows keeping the entire computation of the quantized output in the integer domain.

Integer Multiplication ($\otimes$): For multiplication, $y = x_1 \times x_2$, the exact quantized output leads to the integer-only formulation:

$$q_1 \otimes q_2 = k_1 q_1 q_2 + k_2 q_1 + k_3 q_2 + k_4 \tag{7}$$

where the scaling factors k_i are pre-computed numbers:

$$\begin{aligned} k_1 &= \left\lfloor \frac{\Delta_1 \Delta_2}{\Delta_y} \right\rceil, \quad k_2 = \left\lfloor \frac{\Delta_1 m_2}{\Delta_y} \right\rceil, \\ k_3 &= \left\lfloor \frac{\Delta_2 m_1}{\Delta_y} \right\rceil, \quad k_4 = \left\lfloor \frac{m_1 m_2 - m_y}{\Delta_y} \right\rceil \end{aligned} \tag{8}$$

The derivation of Eqs. (5) and (7) together with the constants in Eqs. (6) and (8) are reported in Appendix A.

Dynamic-Aware Scale Management. The key to enabling dynamic precision is that the multipliers k are simple ratios of scales. Due to the power-of-two relationship in Equation (3), changing the bit-width of any input or output from b_1 to b_2 corresponds to multiplying or dividing its scale Δ by $2^{|b_1 - b_2|}$. This means the multipliers k can also be updated with a simple, low-cost bit-shift at runtime, rather than being re-computed with floating-point division.

Integer-Only Layer Operations and Scale Management

We now apply the integer arithmetic from Section 4 to construct complete neural network layers that operate in a dequantization-free manner. To execute the sequence of multiply-accumulate (MAC) operations required by NN layers in the integer domain, we must manage the quantization parameters (Δ, m) for weights, activations, and layer outputs in a way that is compatible with our nested quantization scheme.

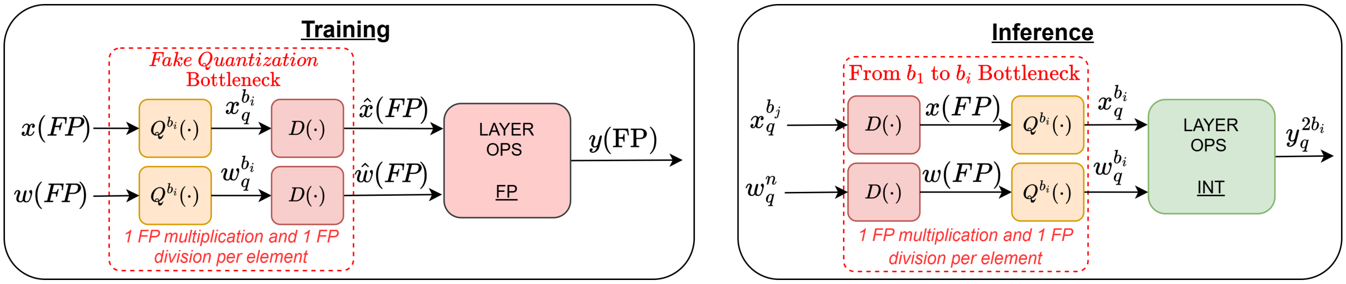

Figure 1: The dequantization bottleneck in conventional quantization frameworks at training time (left) and inference time (right). (Left) In standard QAT, dynamic precision requires a simulated dequantize-requantize cycle. (Right) At inference, transitioning between different integer bit-widths also necessitates this costly conversion to a floating-point intermediate. Our DQT framework eliminates this bottleneck in both scenarios.

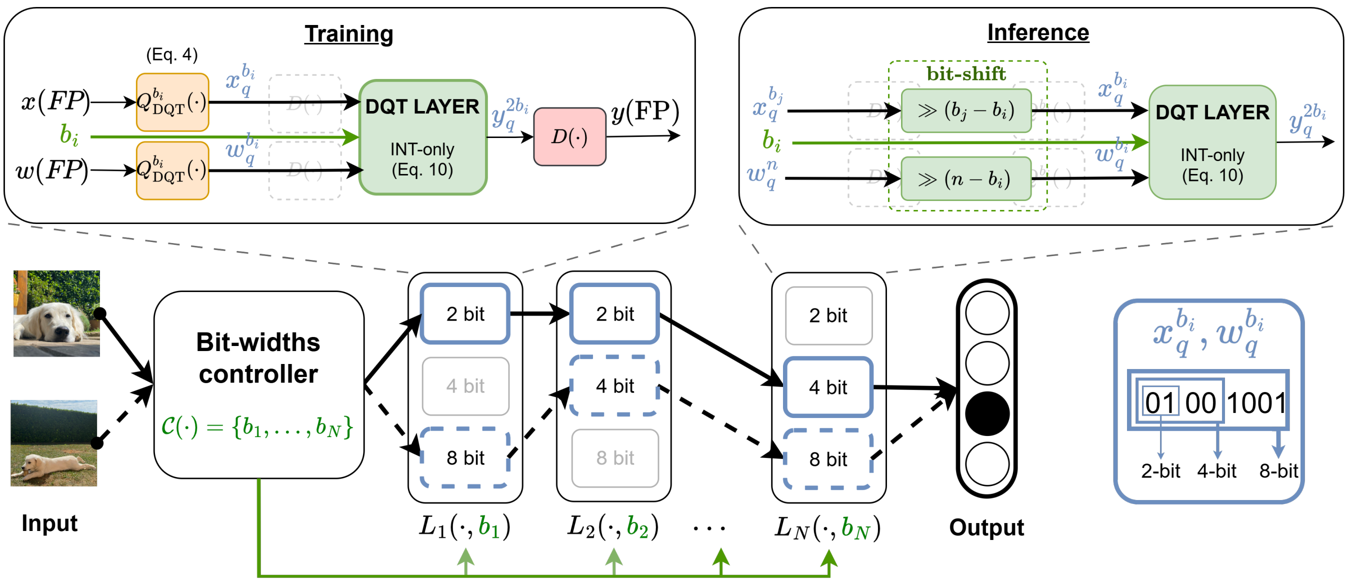

Figure 2: Overview of the Dynamic Quantization Training (DQT) framework. For each input, a lightweight controller $\mathcal{C}$ predicts per-layer bit-widths $\{b_i\}$. The backbone network operates on integer tensors, generating the required precision for weights (W_q) and activations (x_q) on-the-fly from a master representation via bit-shifting. The layer operations L_i are executed using the dequantization-free integer arithmetic defined in Section 4.

Layer Formulation. A floating-point layer $L_{\mathrm{fp}}(x, W, c)$ is reformulated into its integer-only equivalent, $L_{\mathrm{int}}(x_q, W_q, c_q)$. The core MAC operations are performed using the integer multiplication ($\otimes$) and addition ($\oplus$) operators defined in Equations (5) and (7). The inputs x_q, weights W_q, and bias c_q are integer tensors quantized to a dynamically selected bit-width b. The final integer output y_q approximates the correctly quantized result of the original floating-point layer:

$$y_q = L_{\mathrm{INT}}(x_q, W_q, c_q) = Q^b\big(L_{\mathrm{FP}}(x, W, c)\big) + \epsilon \qquad (9)$$

where ϵ denotes the approximation error of our formulation and is analyzed in Appendix C.

Weight and Bias Quantization. The weights W and biases c are static after training. We determine their clipping range $[m, M]$ and compute their master quantization steps $\Delta_{W,n}$ and $\Delta_{c,n}$ for the master bit-width n. These parameters are stored. At runtime, if a bit-width $b < n$ is selected, the required scale is derived directly using the power-of-two relationship from Equation (3): $\Delta_{W,b} = \Delta_{W,n} \cdot 2^{n-b}$.

Activation Quantization. Activation ranges are dynamic and input-dependent. To establish a stable quantization range, we adopt a PACT-like approach (Choi et al. 2018), where activations are clipped to a range $[0, \alpha]$. The clipping bound α is a learnable parameter optimized via gradient descent during training. This provides a fixed range from which we compute a stable master scale for activations, $\Delta_{A,n}$, ensuring compatibility with our nested scheme.

Layer Output Quantization. To determine the quantization parameters for a layer output, (Δ_Y, m_Y), we must estimate its output range. This is done dynamically during training by tracking the running minimum and maximum values of the layer output using an Exponential Moving Average (EMA). For a layer output tensor $y^{(t)}$ at training step t, we update: $y_{\max}^{(t)} = \gamma y_{\max}^{(t-1)} + (1-\gamma)\max(y^{(t)})$ and $y_{\min}^{(t)} = \gamma y_{\min}^{(t-1)} + (1-\gamma)\min(y^{(t)})$, where γ is the EMA momentum. After training, the final range $[y_{\min}, y_{\max}]$ is used to compute the master output scale $\Delta_{Y,n}$.

This hierarchical scale management ensures that all quantization scales required for a layer computation - for weights, activations, and outputs - are defined relative to the master bit-width n. Consequently, all fixed-point multipliers k needed for the integer arithmetic can be adjusted for any target bit-width b with efficient bit-shifts, enabling a truly dynamic, integer-only layer execution.

5 The Dynamic Quantization Training (DQT) Framework

This section presents Dynamic Quantization Training (DQT), an end-to-end framework for efficient, instance-aware inference. DQT leverages the dequantization-free methods from Section 4 and integrates them into a complete system architecture. The system comprises two main components: a *dequantization-free backbone network* that performs all computations in the integer domain, and a lightweight *dynamic controller* that predicts per-layer bit-widths for each input instance. An overview of the DQT architecture is shown in Figure 2.

Dequantization-Free Backbone Network

The backbone network executes the primary DNN computation. Each layer in the backbone, including *convolution*, *fully-connected (FC)*, *batch normalization*, *ReLU*, and *skip connections*, has been completely re-implemented to operate in the nested integer domain. This is achieved by building each layer upon the DQT integer-only operations and scale management techniques detailed in Section 4. This design allows each layer L_i to operate on integer inputs and weights and to dynamically adapt to a bit-width b_i selected by the controller.

The core of this adaptability is the nested quantization scheme. All weights are stored at a single master bit-width, n, and all activations are processed relative to this precision. At runtime, a lower target precision b_i for any tensor x is generated on-the-fly via the bit-shift operation from Equation (4): $Q_{\text{DQT}}^{b_i}(x) = Q^n(x) \gg (n - b_i)$. This approach is both memory- and compute-efficient: only the n-bit weights require storage, and bit-width transitions require a single low-cost logical operation, completely eliminating the dequantize-requantize overhead.

The forward pass of each layer L_i is therefore an integer-only function $L_{\text{int},i}$ that operates on tensors dynamically quantized to the chosen bit-width b_i:

$$y_q^{(i)} = L_{\text{int},i}\left(Q_{\text{DQT}}^{b_i}(x_q^{(i-1)}), Q_{\text{DQT}}^{b_i}(W_i), Q_{\text{DQT}}^{b_i}(c_i)\right) \quad (10)$$

This design ensures that the entire forward pass, including all dynamic precision adjustments, maintains a purely integer dataflow.

Algorithm 1: DQT End-to-End Training Procedure

Input: Training data ($\mathcal{D}$), backbone with FP32 weights W, controller $\mathcal{C}$ with weights $W_{\mathcal{C}}$, hyperparameters α, β, master bit-width n, candidate bit-widths $\{d_k\}$.
Output: Trained weights W and $W_{\mathcal{C}}$.

Initialize W, $W_{\mathcal{C}}$.
for each training epoch **do**
  for each batch $(X, Y) \in \mathcal{D}$ **do**
    $p \leftarrow \mathcal{C}(X)$; $b_{\text{dyn}} \leftarrow$ GumbelSoftmax(p)
    $Y_{\text{pred}} \leftarrow$ ForwardPass$(X, W, b_{\text{dyn}}, n, \text{True})$
    $J \leftarrow J_{\text{task}} + \alpha J_{\text{con.cy}} + \beta J_{\text{cost}}$ ▷ Equation (11)
    Update W and $W_{\mathcal{C}}$ with gradients ∇J.
  end for
end for

Algorithm 2: DQT Dequantization-Free Inference

function INFERENCE$(X_{\text{sample}}, W_q^n, \mathcal{C}, n)$
  $p \leftarrow \mathcal{C}(X_{\text{sample}})$; $b \leftarrow$ argmax(p)
  $Y_{\text{pred}} \leftarrow$ ForwardPass$(X_{\text{sample}}, W_q^n, b, n, \text{False})$
  return Y_{pred}
end function

function FORWARDPASS$(X, W, b, n, \text{is_train})$
  $W_q \leftarrow Q^n(W)$ if is_train, else $W_q \leftarrow W$
  $x_q^{(0)} \leftarrow Q^n(X)$
  for $i = 1, \ldots, N$ **do**
    $W_{q,b_i}^{(i)} \leftarrow W_q^{(i)} \gg (n - b_i)$
    $x_{q,b_i}^{(i-1)} \leftarrow x_q^{(i-1)} \gg (n - b_i)$
    $x_q^{(i)} \leftarrow L_{\text{int},i}(x_{q,b_i}^{(i-1)}, W_{q,b_i}^{(i)})$ ▷ Equation (10)
  end for
  return Dequantize$(x_q^{(N)})$
end function

Dynamic Bit-Width Controller

The dynamic controller $\mathcal{C}$ is a lightweight neural network that selects per-layer bit-widths for each input $\mathbf{x}$ for each of the N backbone layers. It outputs logits for each of these layers, producing a probability distribution $\mathbf{p}_i = \text{softmax}(\mathcal{C}(\mathbf{x})_i)$ over K candidate bit-widths $d_1, \ldots, d_K$, where $\mathbf{p}_i \in \mathbb{R}^K$ is the probability vector for layer L_i.. At inference, bit-widths b_i are chosen via a non-differentiable argmax operation on the output logits. To enable end-to-end gradient-based training, we employ the Gumbel-Softmax reparameterization (Jang, Gu, and Poole 2017), providing a differentiable approximation of the discrete selection process, allowing for joint optimization of the controller $\mathcal{C}$ and the backbone network using standard backpropagation.

Training and Inference

The DQT backbone and controller are trained jointly in an end-to-end fashion. The training objective is designed to optimize for task accuracy while regularizing for computational cost and model robustness.

Method	Model	W-Bits	A-Bits	BitOPs(G)	BW Transition Cost*	Top-1 (%)
			— CIFAR-10 Dataset —			
DoReFa (Zhou et al. 2016)	ResNet18	3	3	0.34	∅	80.90
PACT (Choi et al. 2018)	ResNet18	3	3	0.34	∅	91.10
DQNET (Liu et al. 2022)†	ResNet18	∼3 MP	∼3 MP	0.36	26.4M FLOPs	<u>91.38</u>
Ours†	ResNet18	∼3 MP	∼3 MP	0.36	13.2M bit-shifts	**91.65**
DoReFa (Zhou et al. 2016)	ResNet18	4	4	0.61	∅	90.50
PACT (Choi et al. 2018)	ResNet18	4	4	0.61	∅	91.30
DQNET (Liu et al. 2022)†	ResNet18	∼4 MP	∼4 MP	0.65	26.4M FLOPs	<u>91.60</u>
Ours†	ResNet18	∼4 MP	∼4 MP	0.64	13.2M bit-shifts	**91.73**
DoReFa (Zhou et al. 2016)	ResNet18	5	5	0.95	∅	90.40
PACT (Choi et al. 2018)	ResNet18	5	5	0.95	∅	91.70
DQNET (Liu et al. 2022)†	ResNet18	∼5 MP	∼5 MP	0.98	26.4M FLOPs	<u>92.01</u>
Ours†	ResNet18	∼5 MP	∼5 MP	0.97	13.2M bit-shifts	**92.76**
			— ImageNet Dataset —			
DoReFa (Zhou et al. 2016)	ResNet50	4	4	61.27	∅	71.40
PACT (Choi et al. 2018)	ResNet50	4	4	61.27	∅	76.50
LQ-Nets (Zhang et al. 2018)	ResNet50	4	4	61.27	∅	74.89
LSQ (Esser et al. 2020)	ResNet50	4	4	61.27	∅	76.70
HAQ (Wang et al. 2019)	ResNet50	∼4 MP	∼4 MP	-	56.6M FLOPs	76.14
DQNET (Liu et al. 2022)†	ResNet50	∼4 MP	∼4 MP	61.49	56.6M FLOPs	<u>76.94</u>
Ours†	ResNet50	∼4 MP	∼4 MP	61.39	28.3M bit-shifts	**77.00**
DoReFa (Zhou et al. 2016)	ResNet50	5	5	95.73	∅	71.40
PACT (Choi et al. 2018)	ResNet50	5	5	95.73	∅	76.70
DQNET (Liu et al. 2022)†	ResNet50	∼5 MP	∼5 MP	96.27	56.6M FLOPs	<u>77.12</u>
Ours†	ResNet50	∼5 MP	∼5 MP	95.97	28.3M bit-shifts	**77.25**
MPDNN (Neda et al. 2022)	MobileNetV2	∼4 MP	∼4 MP	-	∅	69.74
AutoQ (Lou et al. 2020)	MobileNetV2	∼4 MP	∼4 MP	-	∅	70.80
FracBits (Yang and Jin 2020)	MobileNetV2	∼4 MP	∼4 MP	-	∅	71.30
DQNET (Liu et al. 2022)†	MobileNetV2	∼4 MP	∼4 MP	2.81	7.0M FLOPs	**72.05**
Ours†	MobileNetV2	∼4 MP	∼4 MP	2.65	3.5M bit-shifts	<u>72.01</u>

† Instance-based architecture.
* Inference Bit-Width (BW) transition cost from the masted BW to the selected one in the worst-case.

Table 1: Comparison of Top-1 Accuracy (%) and computational cost (BitOPs in G) on CIFAR-10 and ImageNet. DQT is compared against state-of-the-art static and dynamic quantization methods. For dynamic methods (†), BitOPs are averaged over the validation set. BW Transition Cost is the operational overhead for changing bit-widths per inference. Our method replaces high-latency FLOPs with low-cost logical shifts. The best Top-1 accuracy is reported in **bold**, the second-best is <u>underlined</u>.

Training and Inference Procedures. The complete end-to-end procedures for training and inference are outlined in Algorithm 1 and Algorithm 2, respectively. During training, the forward pass simulates the full dynamic process using the Gumbel-Softmax for bit-width selection. The backward pass computes gradients with respect to both W and $W_{\mathcal{C}}$, using the Straight-Through Estimator (STE) for quantization nodes. At inference, the trained floating-point weights W are quantized once to the master bit-width n and stored. For each input, the controller deterministically selects the per-layer bit-widths, and the backbone performs a dequantization-free forward pass using bit-shifting and our custom integer arithmetic. Dequantization to floating-point occurs only once for the final network output.

Training Objective. We optimize a multi-term loss J over the full-precision backbone weights W and controller parameters $W_{\mathcal{C}}$: $W^*, W_{\mathcal{C}}^* = \arg\min_{W,W_{\mathcal{C}}} J$, where:

$$J = J_{\text{task}} + \alpha J_{\text{consistency}} + \beta J_{\text{cost}}. \tag{11}$$

J_{task} is the standard loss (e.g., cross-entropy) computed using the controller dynamic bit-widths. $J_{\text{consistency}}$ ensures the backbone network performs well across all candidate bit-widths, not just those frequently chosen by the controller. This term is the sum of J_{task} computed over a small, fixed set of uniform bit-width configurations, $K' \subset \{d_1, \dots, d_K\}$ (e.g., the minimum and maximum available bit-widths): $J_{\text{consistency}} = \sum_{b \in K'} J_{\text{task}}^{(b)}$, where $J_{\text{task}}^{(b)}$ is the task loss when the entire network is statically set to bit-width b. This stabilizes training by ensuring the shared weights are well-conditioned for any potential bit-width selection. J_{cost} penalizes large the choice of large bit-widths by the controller, by computing the expected bit-width across all N layers, averaged over the batch. $J_{\text{cost}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} p_{i,k} \cdot d_k$, where $p_{i,k}$ is the Gumbel-Softmax probability of selecting bit-width d_k for layer i. The hyperparameters α and β control the trade-off between accuracy and computational cost; their impact is analyzed in Appendix F.

6 Experiments

We conduct a series of experiments to validate the performance of the DQT framework. Our primary hypothesis is that by eliminating the dequantization bottleneck, DQT can achieve a SotA accuracy-efficiency trade-off compared to both static and existing dynamic quantization methods.

Experimental Setup

We evaluate our method on two standard image classification benchmarks: CIFAR-10 (Krizhevsky 2009) and the large-scale ImageNet ILSVRC 2012 dataset (Russakovsky et al. 2015). We use widely adopted architectures such as ResNet-18/50 (He et al. 2015) and MobileNetV2 (Sandler et al. 2019). We compare DQT against a comprehensive set of baseline methods. For static uniform quantization, we include learned methods such as DoReFa-Net (Zhou et al. 2016), PACT (Choi et al. 2018), and LSQ (Esser et al. 2020). For static mixed-precision, we compare against HAWQ (Dong et al. 2019) and HAQ (Wang et al. 2019). Our primary dynamic baseline is DQNet (Liu et al. 2022), the leading method for instance-aware dynamic quantization.

We implement all models in PyTorch and train them using SGD. Key hyperparameters, such as learning rates, schedules, weight decay, the controller structure, and the values for α and β, are detailed in Appendix D to ensure reproducibility. We evaluate all models by Top-1 Accuracy (%) on the test set and computational cost in Bit-Operations (BitOPs), computed as $\text{BitOPs} = \sum_i (\text{MACs}_i \cdot b_{w,i} \cdot b_{a,i})$, where $b_{w,i}$ and $b_{a,i}$ are the bit-widths for weights and activations of layer i. For dynamic models, BitOPs are averaged over the validation set. We also report the operational cost of bit-width transitions, a key point of comparison.

Performance Analysis

Table 1 presents a comprehensive comparison of DQT against all baselines. Across all architectures and datasets, DQT consistently establishes a new SotA accuracy-efficiency trade-off. On ImageNet with ResNet-50, our 4-bit dynamic DQT model achieves 77.00% top-1 accuracy, outperforming the SotA static method LSQ (76.70%) and the leading dynamic method DQNet (76.94%) at a comparable BitOPs budget. This result demonstrates that DQT can exceed the performance of static quantization while retaining the benefits of adaptivity. Critically, the bit-width transitions in DQT require only 28.3M logical shift operations, whereas prior dynamic methods incur an overhead of 56.6M high-latency floating-point operations. Similarly, on CIFAR-10 with ResNet-18, DQT consistently outperforms all baselines across 3, 4, and 5-bit configurations, achieving up to 92.76% accuracy. This shift from floating-point arithmetic to logical operations is able to provide a substantial speedup across hardware, both on CPUs by reducing instruction latency and on GPUs by maintaining an efficient integer-only dataflow. We provide precise per-element cost estimations to quantify the computational advantage in Appendix E.

Memory Footprint. The DQT framework is designed for computational optimization. The model memory footprint is determined by the master bit-width n, as the full n-bit weights must be stored. We use $n = 8$, that yields a $4\times$ memory reduction relative to FP32, which is standard for INT8 models. The storage overhead for the integer arithmetic multipliers and the controller parameters is negligible, constituting less than 1% of the total model size. DQT therefore maintains a memory footprint comparable to a static high-precision model while enabling substantial savings in computational cost.

Computational Cost of Integer Arithmetic. While our framework eliminates floating-point operations, the custom integer operators in Section 4 require more than a single hardware MAC. For instance, our integer multiplication (Equation (7)) involves three multiplications and an accumulation. To reduce this cost, we apply PACT (Choi et al. 2018) clamping to the activations, setting the zero-point to zero and removing the k_3 term in Eq. 7, saving one multiplication. In Appendix G, we analyze the precise MAC cost of our solution.

7 Conclusion and Future Work

This paper addressed a fundamental performance bottleneck that has hindered the practical application of dynamic, instance-aware quantization: the computational overhead of runtime dequantization and requantization. We introduced a dequantization-free quantization framework built on two core components: a nested integer representation and a compatible set of custom integer-only arithmetic operators. This design enables bit-width transitions via a single, low-cost logical bit-shift operation, eliminating the need for floating-point conversions. We applied this framework to develop Dynamic Quantization Training (DQT), an end-to-end pipeline for training and deploying efficient, adaptive NNs.

Our experimental results on CIFAR-10 and ImageNet demonstrate that DQT establishes a new SotA considering the accuracy-efficiency trade-offs. By removing the dequantization bottleneck, DQT makes dynamic, instance-aware inference a computationally practical and superior alternative to static quantization. While our experiments focused on this most challenging application of dynamic quantization, the underlying dequantization-free arithmetic represents a general advance for any quantized system, offering a more efficient execution path for both static and dynamic mixed-precision models.

Future research can extend this work in several promising directions. A natural direction is to eliminate the remaining dependency in the training phase. While DQT enables integer-only inference, the backward pass still relies on floating-point gradients.

Acknowledgments

This work was carried out in the EssilorLuxottica Smart Eyewear Lab, a Joint Research Center between EssilorLuxottica and Politecnico di Milano.

References

Bengio, Y. 2013. Estimating or Propagating Gradients Through Stochastic Neurons. arXiv:1305.2982.

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.-J.; Srinivasan, V.; and Gopalakrishnan, K. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. arXiv:1805.06085.

Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M.; and Keutzer, K. 2019. HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision. arXiv:1905.03696.

Esser, S. K.; McKinstry, J. L.; Bablani, D.; Appuswamy, R.; and Modha, D. S. 2020. Learned Step Size Quantization. arXiv:1902.08153.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

Jacob, B.; Kligys, S.; and al. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2704–2713.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. arXiv:1611.01144.

Jin, Q.; Yang, L.; and Liao, Z. 2020. AdaBits: Neural Network Quantization with Adaptive Bit-Widths. arXiv:1912.09666.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Liu, Z.; Wang, Y.; Han, K.; Ma, S.; and Gao, W. 2022. Instance-aware dynamic neural network quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 12434–12443.

Lou, Q.; Guo, F.; Liu, L.; Kim, M.; and Jiang, L. 2020. AutoQ: Automated Kernel-Wise Neural Network Quantization. arXiv:1902.05690.

Nagel, M.; Fournarakis, M.; Amjad, R. A.; Bondarenko, Y.; Van Baalen, M.; and Blankevoort, T. 2021. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*.

Nair, P.; Datta, P.; Dean, J.; Jain, P.; and Kusupati, A. 2025. Matryoshka Quantization. *arXiv preprint arXiv:2502.06786*.

Neda, N.; Ullah, S.; Ghanbari, A.; Mahdiani, H.; Modarressi, M.; and Kumar, A. 2022. Multi-Precision Deep Neural Network Acceleration on FPGAs. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 454–459.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. arXiv:1811.08886.

Xu, Y.; Zhang, S.; Qi, Y.; Guo, J.; Lin, W.; and Xiong, H. 2018. DNQ: Dynamic Network Quantization. arXiv:1812.02375.

Yang, L.; and Jin, Q. 2020. FracBits: Mixed Precision Quantization via Fractional Bit-Widths. arXiv:2007.02017.

Zhang, D.; Yang, J.; Ye, D.; and Hua, G. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. arXiv:1807.10029.

Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; and Zou, Y. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.