# HOT: Hadamard-based Optimized Training

Seonggon Kim[1], Juncheol Shin[2], Seung-taek Woo[2], and Eunhyeok Park[1,2]

[1]Department of Computer Science and Engineering, POSTECH
[2]Graduate School of Artificial Intelligence, POSTECH
{sungonuni, jchshin, wst9909, eh.park}@postech.ac.kr

## Abstract

*It has become increasingly important to optimize backpropagation to reduce memory usage and computational overhead. Achieving this goal is highly challenging, as multiple objectives must be considered jointly while maintaining training quality. In this paper, we focus on matrix multiplication, which accounts for the largest portion of training costs, and analyze its backpropagation in detail to identify lightweight techniques that offer the best benefits. Based on this analysis, we introduce a novel method, Hadamard-based Optimized Training (HOT). In this approach, we apply Hadamard-based optimizations, such as Hadamard quantization and Hadamard low-rank approximation, selectively and with awareness of the suitability of each optimization for different backward paths. Additionally, we introduce two enhancements: activation buffer compression and layer-wise quantizer selection. Our extensive analysis shows that HOT achieves up to 75% memory savings and a 2.6× acceleration on real GPUs, with negligible accuracy loss compared to FP32 precision. Our code is available at* `https://github.com/sungonuni/HOT`.

## 1. Introduction

Recent advances in combining large-scale foundation models with transfer learning have enabled superior performance on new tasks with limited data. However, with the rise of resource-intensive tasks such as high-resolution image generation [26, 29, 43], video synthesis [3, 31, 35], and long-context language processing [5, 11, 15, 21], fine-tuning now requires even more resources, making it nearly unaffordable for most companies or academic institutions.

However, designing an efficient training pipeline is highly challenging, as it involves numerous considerations and multiple tensors with diverse characteristics. Training optimization faces three key challenges: (1) memory usage for model parameters and optimizer states, (2) memory usage for intermediate activations needed for backpropagation

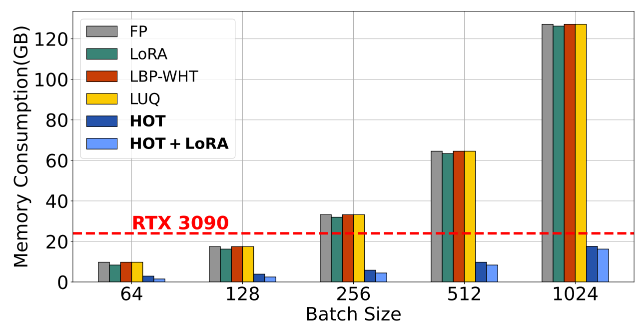


Figure 1. Memory requirements for training ViT-B [12] on ImageNet-1k dataset [9] with varying batch sizes. While FP and other efficienct BP methods fail to train with batch sizes of 256 and above on a single GPU having 24 GB memory, HOT enables training with batch sizes up to 1024.

| | Objectives | | | |
|---|---|---|---|---|
| | Weight & Optimizer Memory | Intermediate Activation | Backpropagation Speed | Trained Model Quality |
| **LoRA** [18] | ✓ | ✗ | ▲ | ✓ |
| **LBP-WHT** [42] | ✗ | ✗ | ✓ | ▲ |
| **LUQ** [7] | ✗ | ✗ | ▲ | ✓ |
| **HOT** | ✗ | ✓ | ✓ | ✓ |
| **HOT + LoRA** | ✓ | ✓ | ✓ | ✓ |

Table 1. Comparison with previous works. HOT is the only method which achieves activation memory reduction and actual speedup altogether without compromising model quality.

(BP), and (3) accelerating BP. While the first challenge is well-addressed by recent advances like Parameter-efficient Fine-tuning (PEFT), such as LoRA [18], the other two challenges remain unsatisfactorily resolved. Approaches like LUQ [7] and LBP-WHT [42] focus on BP acceleration but fail to tackle all these objectives jointly.

In this work, we propose a novel method called Hadamard-based Optimized Training (HOT), which focuses on optimizing matrix multiplication—a major contributor to training overhead. First, We thoroughly analyze the backward characteristics of activations and weights.

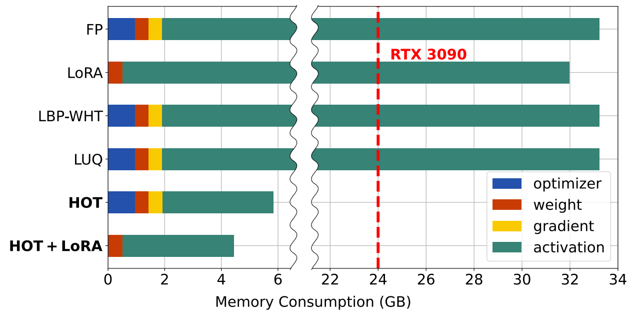


Figure 2. Component-wise memory consumption breakdown for different methods when training ViT-B [12] on ImageNet-1k dataset [9] with a batch size of 256.

Based on this insight, we determine the optimal strategy for selectively applying Hadamard-based optimization techniques tailored to each unique characteristic. Combined with our two new ideas—Activation Buffer Compression (ABC) for activation memory reduction and Layer-wise quantizer Selection (LQS) for fine-quality quantization—the benefits of HOT are maximized. Additionally, we explore and optimize the integration of HOT with LoRA, creating a comprehensive solution for optimized training.

Our experiments validate HOT's efficiency and training quality on both vision and language models using large-scale datasets. As shown in Fig. 1, HOT achieves the best efficiency with large batch input. Our results indicate that even on an actual GPU, HOT achieves a **2.6× speedup** with custom CUDA kernels and a **75% memory reduction**, all without **compromising training accuracy** than FP.

## 2. Related Work

### 2.1. Quantization for Efficient Training

Numerous studies have explored the potential of quantization for efficient training. A common approach involves applying quantization to the forward propagation. Chmiel et al. [7] proposed a logarithmic quantizer with a custom FP4 format for gradient optimization, though it had limitations in hardware acceleration. Xi et al. [39] combined forward quantization with structural pruning for gradients, demonstrating success in language tasks. Other approaches, such as FP8 training [24, 25] and integer-only training [16, 37], aim to modify representations across the entire training process. **In contrast, our method maintains full precision in the forward propagation to ensure accurate loss evaluation and preserve training quality, while only quantizing the gradient computation for optimization.**

### 2.2. Rank Reduction for Efficient Training

Several approaches focus on reducing training costs through low-rank decomposition. LoRA [18] reduces the fine-tuning cost of large language models (LLMs) by specifically targeting the memory usage of their massive weights. LBP-WHT [42] was the first to introduce Hadamard low-rank approximation in backpropagation, achieving practical acceleration. However, it suffers from significant accuracy degradation on large-scale models. HOT also employs HLA, but we apply it selectively, taking into account the characteristics of backpropagation paths to minimize its drawbacks.

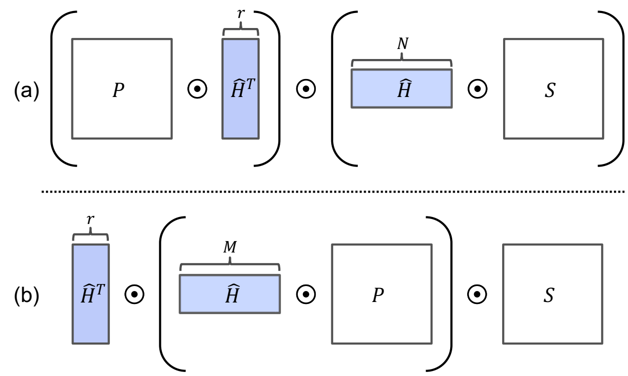


Figure 3. Illustration of (a) internal HLA and (b) external HLA. Internal HLA reduces dimension $N$ to $r$, while external HLA compress $M$ to $r$. The operator $\odot$ represents matrix multiplication.

## 3. Preliminaries

HOT builds upon two advanced optimization techniques—Hadamard Quantization (HQ) and Hadamard Low-rank Approximation (HLA)—and applies them selectively, taking into account the characteristics of the backpropagation for weight and activation gradient, respectively.

In this section, we introduce how HQ and HLA are leveraged for BP paths. Before we begin, we first define the notation for matrix multiplication. The operator $\cdot$, as used in this paper, denotes matrix-matrix multiplication between two matrices. Given two 2-dimensional matrices $P \in \mathbb{R}^{M\times N}$ and $S \in \mathbb{R}^{N\times K}$, their product $R = P \cdot S \in \mathbb{R}^{M\times K}$ is obtained by taking inner products along the $N$ dimension.

### 3.1. Hadamard Transformation

The Hadamard Transformation (HT) [32] is the core building block for mitigating quality loss during quantization and low-rank approximation. HT can be viewed as a type of Fourier transformation, converting an $n$-dimensional vector $v \in \mathbb{R}^{2^d}$ into the spectral domain $\tilde{v} \in \mathbb{R}^{2^d}$ through the following linear transformation:

$$\tilde{v} = H_d \cdot v, \tag{1}$$

where $H_d$ is the Walsh-hadamard matrix, defined by following recurrence equation:

$$\mathrm{H}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathrm{H}_n = \mathrm{H}_1 \otimes \mathrm{H}_{n-1}, \tag{2}$$

where $\otimes$ denotes the Kronecker product.

A key advantage of the HT is its computational efficiency: the Fast Walsh-Hadamard Transform [28] requires only $\mathcal{O}(2^d \cdot \log 2^d)$ operations, relying solely on additions and subtractions. Additionally, we apply a block-diagonal transformation, also known as the order-$n$ 2D HT [39], which divides dimensions into specific tiles (typically 16) and performs transformations for each tile independently.

### 3.2. Hadamard Quantization

Recently, the combination of the HT with quantization, known as Hadamard Quantization (HQ), has gained attention for its lower degradation in low-precision settings. HQ applies HT along the dimension where inner products are computed, leveraging the orthogonality of the Hadamard matrix to ensure consistent outputs in subsequent updates:

$$R = P \cdot S = P \cdot (H^T \cdot H) \cdot S = (P \cdot H^T) \cdot (H \cdot S), \quad (3)$$

$H \in R^{N \times N}$ is a dimension-matched block-diagonal Walsh-Hadamard matrix. For the arbitray quantization $Q(\cdot)$, HQ applies it to P and S after they are mapped to the frequency domain. In this case, the approximated output $\tilde{R}$ is calculated as follows:

$$\tilde{R} = Q(P \cdot H^T) \cdot Q(H \cdot S). \quad (4)$$

HT converts irregular data with outliers into frequency distributions, making the data more resilient to quantization [27, 39]. Although HQ introduces minor computational overhead, the added cost is justified by its benefits.

### 3.3. Hadamard Low-rank Approximation

The Hadamard Low-rank Approximation (HLA) is a technique that reduces computational and memory costs, as illustrated in Fig. 3. HLA leverages HT to convert data to the frequency domain, selecting the low-frequency components after transformation [42].

HLA can be categorized into internal and external HLA, depending on the dimension to which the low-rank approximation is applied. Internal HLA applies the HT to the inner dimension, approximating the operation by selecting only $r \ll N$ dimensions. When we denote this reduced Walsh-Hadamard matrix as $\hat{H}$, the approximated output $\hat{R}$ is calculated as follows:

$$\hat{R} = (P \cdot \hat{H}^T) \cdot (\hat{H} \cdot S). \quad (5)$$

In contrast, external HLA applies low-rank approximation to the $M$ dimension of $P$ or the $K$ dimension of $S$. For example, when applying HLA to the $M$ dimension, the approximated output is calculated as follows:

$$\hat{R} = \hat{H}^T \cdot (\hat{H} \cdot P) \cdot S. \quad (6)$$

| $g_x$ path | $g_w$ path | Accuracy |
|---|---|---|
| FP | FP | 76.46 |
| FP | HT + 4-bit Q | 72.43 |
| FP | Internal-HLA | **76.29** |
| 4-bit Q | FP | 73.4 |
| HT + 4-bit Q | FP | **76.16** |
| External-HLA | FP | 72.01 |
| Internal-HLA | FP | 51.10 |

Table 2. Table showing the application of HT and HLA with varying precision when pretraining ResNet50 with CIFAR100. A stochastic quantization [7] is used as the quantizer.

Both methods offer computational and memory advantages by reducing dimensionality. Although HT introduces minor overhead, it is a manageable cost given the benefits.

In the existing LBP-WHT [42] paper, external HLA is used for lightweight computation along the $g_x$ path, while internal HLA is applied when calculating $g_w$. However, this approach results in notable accuracy loss. Our observation indicates that HLA is unsuitable for the activation gradient path, suggesting a need to explore alternative methods.

## 4. Optimization Sensitivity Analysis

For optimal acceleration and memory savings with minimal accuracy loss, we conduct an in-depth analysis of the optimization sensitivity in the BP of matrix multiplication. Although the two operands in backpropagation, $g_x$ and $g_w$, serve similar functions, their tensor properties differ significantly. Consequently, each path may exhibit different sensitivities to optimization.

### 4.1. Notataion for BP

In our explanation, we use unified notation for each dimension. Input channels are denoted as (I), output channels as (O), and the sequence dimension as (L). Although activations include a batch dimension $B$, we omit it here for brevity. Based on this notation, in the forward pass, for input activation $x \in \mathbb{R}^{L \times I}$ and weight $w \in \mathbb{R}^{O \times I}$, the output $y \in \mathbb{R}^{L \times O}$ is calculated as follows:

$$y = x \cdot w^T \quad (7)$$

In the backward pass, given the gradient of the output activation $g_y \in \mathbb{R}^{L \times O}$, we can calculate the gradients of the input activation $g_x \in \mathbb{R}^{L \times I}$ and the weight $g_w \in \mathbb{R}^{O \times I}$ using the chain rule:

$$g_w = g_y^T \cdot x, \quad g_x = g_y \cdot w. \quad (8)$$

The computation of each gradient incurs the same cost as the forward pass operations, so both paths should be optimized judiciously using appropriate methods. Notably, when optimizing Vision Transformers (ViT) [12] or fully convolution layers, the same optimization techniques can be applied by substituting $L = W \times H$, where $W$ and $H$ represent the spatial width and height, respectively.

## 4.2. Activation Backpropagation Optimization

The $g_x$ path is computed as a matrix multiplication of the output gradient ($g_y$) and the weight ($w$), as shown in Eq. (8). Since $g_y$ is stored locally and $w$ is persistent, optimization for $g_x$ should prioritize acceleration over memory reduction. Although $g_x$ computations are performed across all layers for each instance, there is an opportunity for robustness in the batch dimension, as the instance-wise gradient is averaged over this dimension.

To optimize the $g_x$ path, we analyzed three approaches: HQ, internal HLA, and external HLA, finding that HQ is the most effective. The existing SOTA, LBP-WHT, reduces computational costs by applying external HLA to the $L$ dimension of $g_y$. We also evaluated internal HLA by reducing the rank of the common $O$ channel. However, as shown in Table 2, both HLA methods resulted in large accuracy degradation in the $g_x$ path.

This damage seems to arise from frequency loss patterns generated during HLA. When these patterns accumulate across the network, they create trends difficult to correct, even with batch dimension averaging. Moreover, these errors worsen rapidly in deeper layers, as shown in Fig. 4.

Meanwhile, quantization can be modeled as adding random noise [2], which is mitigated by averaging across the batch dimension. Moreover, applying HT plays a crucial role in error reduction. Notably, combining HT with 4-bit quantization performs surprisingly well and opens up opportunities for acceleration, as 4-bit GEMM can be over $3\times$ faster than FP16 GEMM via Tensorcore acceleration [38].

## 4.3. Parameter Backpropagation Optimization

In BP, $g_w$ is computed as the product of intermediate activations and the transpose of $g_y$ Eq. (8). Since $g_w$ stores and reuses activations from the forward pass, reducing its memory consumption is a key optimization goal.

After analyzing the characteristics of $g_w$, we confirmed that internal HLA is highly suitable, as shown in Tab. 2. Fundamentally, $g_w$ updates are performed by taking the averaged value across the $L$ dimension, which acts as a low-pass filter in the frequency domain [42]. When performing internal HLA, the accuracy loss from selecting only low-frequency ranks for the $L$ dimension is mitigated, as it aligns with this filtering effect.

In contrast, low-bit quantization caused significant accuracy degradation, even with HT applied. This is likely due to weight gradients directly influencing the weight update trajectory through repeated accumulation, making quantization errors immediately impactful on quality. Observing $g_w$ in Fig. 4, we can see that HQ induces larger errors than HLA across $g_w$ paths in all models. While quantization could be considered for $g_w$ to manage memory consumption, it must be applied conservatively.

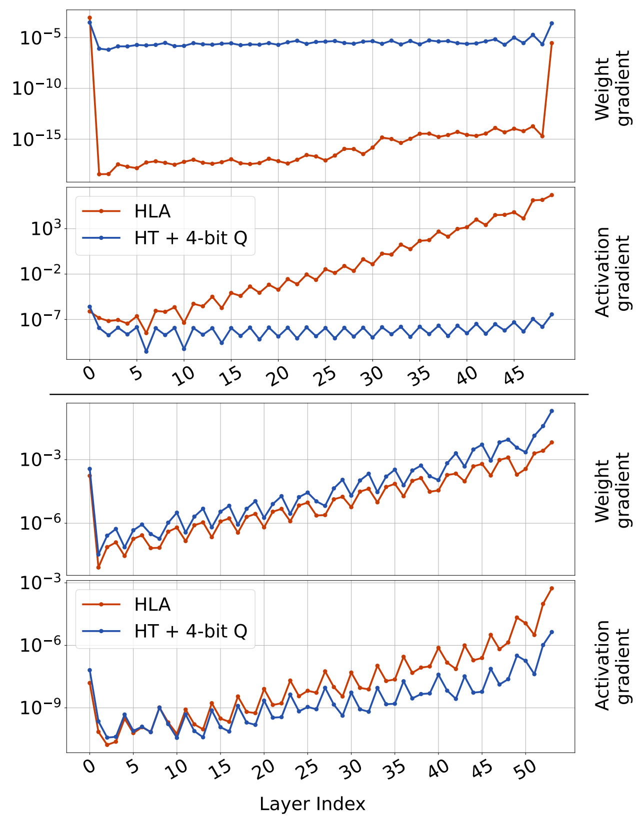


Figure 4. Layer-wise MSE error analysis for ViT-B (top) and ResNet-50 (bottom). The graphs demonstrate higher errors in HT+INT4 for the weight gradient ($g_w$) path, while showing accumulated errors in HLA for the activation gradient ($g_x$) path.

Through additional observations, we identified a potential approach to further mitigate the quantization sensitivity of $g_w$. In Transformer-based architectures, as shown in Fig. 6, we observed gradient outliers in certain layers. When gradients generated from specific tokens have unusually large values compared to others, applying per-tensor quantization results in significant errors. Instead, by applying per-token quantization [40] based on token-specific statistics, we can greatly reduce quantization errors. This approach is practically meaningful: because per-token quantization maintains a consistent scale across the $L$ dimension, we can still utilize INT8 GEMM, with the scaled output achieved by multiplying the token-wise scale with the GEMM output. This new approach allows us to implement additional optimizations in the $g_w$ path, alongside HLA.

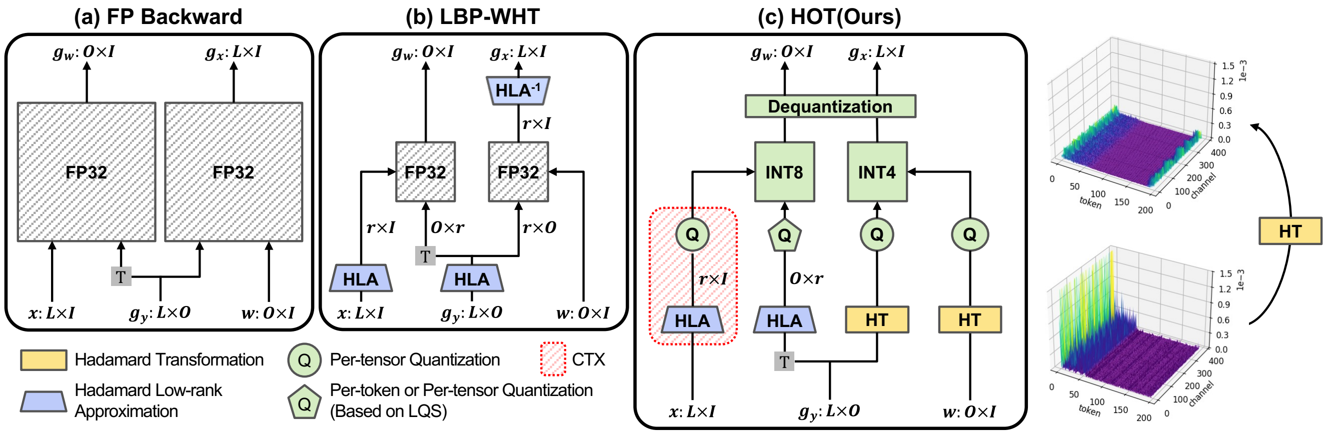


Figure 5. The pipelines for (a) standard BP, (b) LBP-WHT [42], and (c) HOT. HOT reduces memory consumption by compressing activations for BP using HLA and INT8 quantization, storing them in CTX (shown in red in (c)). It accelerates computations through integer matrix multiplication for gradient calculations (represented by INT8 and INT4 rectangular sections in (c)).

## 5. HOT: Hadamard-based Optimized Training

Based on our previous observations, we propose a holistic optimization pipeline, HOT: Hadamard-based Optimized Training. HOT employs optimal strategies tailored to the characteristics of both the $g_w$ and $g_x$ paths. Furthermore, for the $g_w$ path, we introduce Activation Buffer Compression (ABC), which achieves memory savings, and Layer-wise Quantizer Selection (LQS), which balances stable quantization and training speed.

### 5.1. Optimization on activation gradient path

HOT applies only INT4 quantization to $g_x$ without low-rank approximation. First, HT is applied to $g_y$ along the $O$ dimension to generate the projected tensor, after which quantization is applied to the projected $g_y$. For $w$, HT is similarly applied along the $O$ dimension to produce the projected $w$, which is then quantized to INT4. Since both $g_y$ and $w$ are quantized to integers, the integer $g_x$ can be efficiently computed through INT4 GEMM and dequantized to FP32.

We accelerate computation through INT4 matrix multiplication using NVIDIA TensorCores. Our optimization combines a custom CUDA implementation and operator fusion for HT and quantization to maximize acceleration.

For quantization, unbiased quantization is essential; biased quantization can significantly degrade training quality [6]. It is known that stochastic rounding provides an unbiased estimate [4], so HOT employs min-max stochastic quantization. However, to minimize the overhead of generating random numbers, we use a pseudo-stochastic quantizer [37], leveraging the lower 11 bits of FP32 as pseudo-random values to determine whether to round FP numbers.

### 5.2. Optimization on weight gradient path

For $g_w$, HOT combines HLA and INT8 quantization. As mentioned in section 4.3, this reflects the characteristic that the $g_w$ computation path is robust to HLA but sensitive to quantization.

The computation process is as follows. First, $g_y$ is transposed and HLA is applied along the L dimension. Next, INT8 quantization is applied to the compressed $g_y$. For $x$, this operation is also applied along the L dimension to generate compressed $x$, which is then quantized to 8-bit integers. Since both $g_y$ and $x$ are quantized to 8 bits, integer $g_w$ is calculated through INT8 matrix multiplication. Finally, integer $g_w$ is dequantized to FP32.

#### 5.2.1. Activation Buffer Compression

Additionally, reducing activation memory is a critical factor in the $g_w$ path. To address this, we introduce Activation Buffer Compression (ABC) to minimize memory costs.

To further improve memory efficiency in the $g_w$ path, HLA and quantization for $x$ can be applied immediately after the forward pass, rather than during the backward pass. After the forward pass, when storing $x$ in main memory for the backward pass of each layer, we can significantly reduce activation memory usage by storing $x$ with HLA and 8-bit quantization already applied. This technique allows activation size to be compressed by up to 50% during the HLA process, with an additional compression of 25% achieved during quantization when converting from FP32 to INT8. This theoretically enables memory savings of up to 12.5% compared to the original. We apply it as a fundamental memory optimization strategy.

#### 5.2.2. Layer-wise Quantizer Selection

To further reduce quantization errors in $g_w$, per-token quantization can be applied to $g_y$. However, this approach involves the trade-off of additional inference costs due to the need for token-wise scale calculation. Since the characteristics of $g_y$ vary by layer, HOT can selectively apply either per-token or per-tensor quantization.

By analyzing the patterns of $g_y$ across layers, we identified two distinct cases: (a) layers that consistently show gradient outliers for specific tokens, and (b) layers where gradient outliers are difficult to recognize. For example, as shown in Figure 6, the first case consistently appears in the attention projection and fc2 layers of ViT-S [12], while the second case occurs consistently in its fc1 layer. In Case (a), tensor-wise quantization results in significant accuracy loss, so token-wise quantization should be applied despite its associated overhead. In Case (b), tensor-wise quantization is preferable, as the computational overhead of token-wise quantization outweighs its accuracy benefits.

To implement this, HOT uses a hand-crafted, empirical selection process by performing a backward pass on a small calibration set prior to training. Specifically, for each layer's $g_y$, we calculate the Mean Squared Error (MSE) between FP and per-token quantization, and between FP and per-tensor quantization. If the error difference is less than 50%, per-tensor quantization is applied; if the difference is 50% or greater, per-token quantization is used. We refer to this layer-wise quantizer selection method as LQS. Although the trade-offs require further detailed evaluation, this straightforward strategy provides satisfactory benefits in both computational throughput and training quality.

### 5.3. Joint optimization with LoRA

LoRA [18] and HOT have distinct optimization objectives, making them complementary in application. We have confirmed that by applying LoRA in the forward pass and HOT in the backward pass, we can effectively compress model weights and optimizer states through LoRA, while compressing intermediate activations through HOT.

The key characteristic of using HOT and LoRA together is the differentiated approach to its frozen weight and decomposed weight. In frozen weight, where parameters remain unchanged, HOT skips the calculation of $g_w$ and only computes $g_x$ for the backward pass to the next layer.

In contrast, within LoRA's decomposed weight, updates are applied to the decomposed weights after the backward pass. We found that performing traditional BP without applying HOT in this part is crucial for preserving accuracy. Since LoRA already reduces the cost of $g_w$, the additional cost reduction from HOT in these regions is minimal, making this trade-off beneficial. By optimally combining the strengths of LoRA and HOT, we have developed a holistic and efficient training method that maximizes training performance while minimizing memory usage.

## 6. Experiment

We conducted comprehensive experiments to validate the superiority of the proposed technique. We set FP, LUQ [7], LBP-WHT [42], and LoRA [18] as comparison groups, with LoRA results reported only for models with attention layers. Unless otherwise specified, ABC and LQS were applied by default. Detailed experimental settings including low-pass vector selection criteria of HLA and hyper-parameters are described in Supplementary Materials.

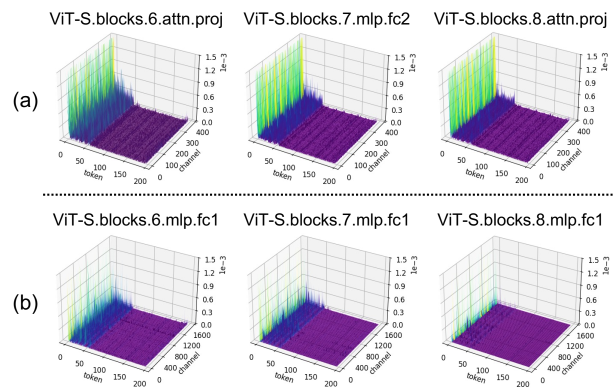


Figure 6. The illustration for output gradient tensor of (a) attention proj layer of the 6, 8th block and fc2 layer of the 7th block in ViT-S [12]; (b) fc1 layers of the 6th, 7th and 8th blocks in ViT-S. While (a) benefits from per-token quantization, (b) shows comparable performance with per-tensor quantization.

### 6.1. Efficient fine-tuning on vision tasks

First, we compared fine-tuning quality across representative vision tasks: Classification, object detection, and semantic segmentation, with results shown in table 3. For classification, we conducted experiments on EfficientNetV2 [33], EfficientFormer [22], and ViT [12], fine-tuning models pretrained on ImageNet-1k [9] using CIFAR10 and CIFAR100 datasets [20]. For object detection and semantic segmentation, we conducted experiments using Segformer [41] and YOLO-V5 [19] models respectively, fine-tuning models pretrained on COCO dataset [23] using PASCAL VOC [14] and Cityscape [8] datasets.

As shown in table 3, HOT demonstrates performance significantly better than previous SOTA in fine-tuning. Notably, LUQ, which is a former state-of-the-art method, fails to train on certain architectures, but our method achieves stable fine-tuning regardless of the architecture. It also shows superior quality in almost all cases compared to LBP-WHT. Additionally, we confirmed that combining with LoRA proceeds successfully with minimal degradation.

### 6.2. Fine-tuning result of Large Language Model

We extended the scope beyond vision models to LLM fine-tuning to validate its effectiveness. Table 4 shows the results of fine-tuning BERT-base [10] and Llama3-8B [13], two representative language models, on MRPC [36] and Alpaca [34] datasets respectively.

The experimental results with Llama3-8B particularly demonstrate superiority of HOT. State-of-the-art BP opti-

| Dataset | Model | FP | LoRA [18] | LUQ [7] | LBP-WHT [42] | HOT | HOT + LoRA |
|---|---|---|---|---|---|---|---|
| CIFAR10 [20] | EfficientNetV2-s [33] | 97.61 | - | NaN | 94.09 | **96.79** | - |
| | EfficientNetV2-m [33] | 98.01 | - | NaN | 95.85 | **96.17** | - |
| | EfficientFormer-L1 [22] | 97.31 | 94.14 | 96.42 | 94.6 | **96.49** | 94.01 |
| | EfficientFormer-L7 [22] | 98.65 | 97.10 | 97.86 | 97.22 | **98.03** | 96.90 |
| | ViT-S [12] | 98.68 | 98.70 | 97.37 | 97.24 | 98.21 | **98.60** |
| | ViT-B [12] | 96.36 | 96.07 | 95.70 | 91.37 | 95.01 | **96.03** |
| CIFAR100 [20] | EfficientNetV2-s [33] | 87.36 | - | NaN | **87.10** | 86.29 | - |
| | EfficientNetV2-m [33] | 84.89 | - | NaN | 55.08 | **83.91** | - |
| | EfficientFormer-L1 [22] | 85.73 | 84.84 | 78.70 | 83.27 | **84.07** | 83.42 |
| | EfficientFormer-L7 [22] | 88.04 | 85.09 | 76.33 | 85.10 | **87.39** | 85.01 |
| | ViT-S [12] | 86.81 | 85.67 | **86.37** | 85.96 | 85.81 | 85.49 |
| | ViT-B [12] | 93.45 | 92.61 | 91.76 | 92.47 | **92.99** | 92.51 |

(a) **Fine-tuning on classification task**

| Dataset | Model | FP | LoRA [18] | LUQ [7] | LBP-WHT [42] | HOT | HOT + LoRA |
|---|---|---|---|---|---|---|---|
| VOC 2012 [14] | Segformer-mit-b2 [41] | 80.93 | 79.40 | 78.70 | 78.40 | 78.86 | **79.10** |
| Cityscape [8] | | 72.16 | 70.39 | 70.31 | 71.16 | **71.72** | 70.37 |
| VOC 2007 [14] | YOLO V5-S [19] | 85.80 | - | NaN | 85.00 | **85.10** | - |

(b) **Fine-tuning on semantic segmentation and object detection**

Table 3. Quality degradation comparison of BP optimization techniques across classification and semantic segmentation tasks. HOT demonstrates superior performance over other methods across almost all models and maintains comparable accuracy when integrated with LoRA [18].

| Dataset | Model | FP | LoRA [18] | LUQ [7] | LBP-WHT [42] | HOT | HOT + LoRA |
|---|---|---|---|---|---|---|---|
| MRPC [36] | BERT-base [10] | 86.52 | 81.61 | **84.56** | 80.88 | 84.31 | 81.6 |
| Alpaca [34] | Llama3-8B [13] | 3.28 | 3.8 | NaN | NaN | **3.29** | 3.78 |

Table 4. Performance comparison of MRPC accuracy(%) and Alpaca perplexity(%) in LLM fine-tuning. Higher MRPC accuracy and lower Alpaca perplexity indicate better performance.

mization techniques like LUQ and LBP-WHT showed severe degradation as model depth increased, sometimes resulting in failure. In contrast, HOT achieved stable training even with the deep architecture of Llama3-8B. In fine-tuning the relatively smaller BERT-base, HOT demonstrated performance comparable to existing SOTA methods.

## 6.3. Performance Analysis

A key factor of HOT is actual acceleration while reducing memory footprint. In this section, we compare the performance advantages of HOT against existing methods.

### 6.3.1. Memory Consumption

HOT dramatically reduces memory usage through activation compression using ABC. In its basic implementation, HOT shows memory efficiency similar to LBP-WHT [42]. However, with the introduction of ABC, it successfully achieves an additional 75% memory reduction, further emphasizing its effectiveness in managing memory.

| Dataset | Model | FP | LUQ [7] | LBP-WHT [42] | HOT |
|---|---|---|---|---|---|
| ViT-B [12] | ImageNet-100 [9] | 77.87 | 75.97 | 54.25 | **77.31** |
| | ImageNet-1k [9] | 70.01 | 67.1 | NaN | **69.4** |

Table 5. Accuracy results for pre-training tasks of ViT-B model on ImageNet-1k and ImageNet-100 datasets. Unlike other methodologies, HOT shows less than 1% degradation with FP.

Figure 7 shows memory usage analysis for ResNet-50 [17], ViT-B [12], and EfficientFormer-L7 [22] models with a batch size of 256 in ImageNet [9]. Notably, while LBP-WHT and LUQ [7] consume the same memory as FP32, HOT achieves up to 86% memory reduction in ResNet-50 and up to 75% in ViT models. Memory optimization of HOT makes it possible to train with a single NVIDIA RTX 3090 having 24GB main memory, which has practical point in accessibility of deep learning training.

### 6.3.2. Computation Cost

While HOT incurs slight overhead compared to basic BP in transformation and quantization, it significantly reduces overall computation cost through low-precision operations. The overhead analysis is described in Supplementary Materials. Figure 7 shows the comparison of bit operations (bops) [1, 30] across ResNet-50 [17], ViT-B [12], and EfficientFormer-L7 [22]. Specifically, HOT achieves approximately 64% reduction in computational cost compared

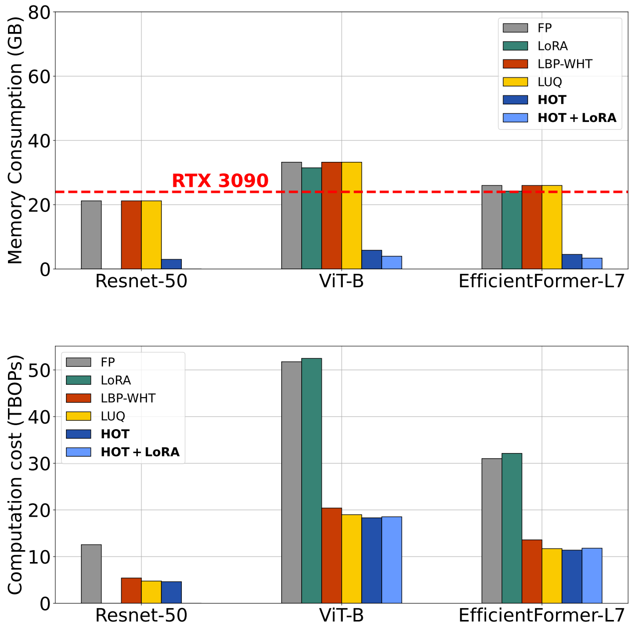


Figure 7. Estimated memory usage with 256 batch of ImageNet dataset [9] and computational costs when applying HOT and existing methods across various models. HOT effectively reduces activation memory footprint, making training feasible on a single RTX 3090 GPU by significantly lowering resource requirements.

to FP in ResNet-50, which is more efficient than both LBP-WHT [42] and LUQ [7]. Similarly, it recorded 65% reduction in both EfficientFormer-L7 and ViT-B.

We also measured the actual latency of BP through CUDA kernels on an RTX 3090 GPU, which is shown in Table 6. Experimental results show that HOT achieves up to 3.3× speedup in the fc2 layer of ViT-B, 2.6× in average of all layers of the model. The implementation detail of CUDA kernels is detailed in Supplementary Materials.

### 6.4. Pre-training Result

We evaluated the performance in the more challenging task of full training. Full training is known to be more vulnerable to gradient quantization or low-rank approximation as it starts from randomly initialized weights.

Table 5 shows the results of full training ViT-B [12] model on ImageNet-1k and ImageNet-100 datasets [9]. In ImageNet training, while other efficiency methods either show substantial degradation (LUQ) or train failure (LBP), HOT achieves performance nearly equivalent to FP. The experimental results show that HOT achieved superior performance compared to existing methods even under these demanding conditions, demonstrating stable convergence throughout the training. More detailed experimental results and analysis can be found in Supplementary Materials.

| (L, O, I) | Name | FP | LBP-WHT | HOT |
|---|---|---|---|---|
| | **ResNet-50** | | | |
| (3136, 64, 256) | layer1.conv1 | 115 | 106 | 62 (1.9×) |
| (3136, 64, 576) | layer1.conv2 | 134 | 117 | **59 (2.3×)** |
| (784, 128, 512) | layer2.conv1 | 117 | 99 | 67 (1.8×) |
| (784, 128, 1152) | layer2.conv2 | 124 | 81 | 60 (2.1×) |
| (196, 256, 2304) | layer3.conv2 | 114 | 85 | 64 (1.8×) |
| (49, 512, 4608) | layer4.conv2 | 137 | 102 | 72 (1.9×) |
| | **ViT-B** | | | |
| (197, 2304, 768) | qkv | 182 | 110 | 70 (2.6×) |
| (197, 768, 768) | proj | 122 | 108 | 71 (1.7×) |
| (197, 3072, 768) | fc1 | 226 | 120 | 73 (3.1×) |
| (197, 768, 3072) | fc2 | 233 | 112 | **72 (3.3×)** |
| | **EfficientFormer-L7** | | | |
| (3136, 384, 96) | stages.0.fc1 | 125 | 123 | 63 (2.0×) |
| (784, 768, 192) | stages.1.fc1 | 129 | 108 | 68 (1.9×) |
| (196, 1536, 384) | stages.2.fc1 | 126 | 102 | 66 (1.9×) |
| (49, 1536, 768) | stages.3.qkv | 128 | 105 | 62 (2.1×) |
| (49, 768, 1024) | stages.3.proj | 111 | 105 | 69 (1.6×) |
| (49, 3072, 768) | stages.3.fc1 | 146 | 110 | **66 (2.2×)** |

Table 6. Latency($\mu s$) profiling results measured on RTX 3090 GPU with varying layer dimensions. HOT achieves significant speedups across different layers and architectures, outperforming LBP-WHT [42] by large margin.

## 7. Limitation and Broader impact

As model size and depth increase, HOT tends to show more performance degradation. It is a clear limitation of HOT, despite incurring fewer errors compared to existing methods. Nevertheless, HOT can still be considered an optimal solution for training in resource-constrained environments with limited memory and computational capacity.

## 8. Conclusion

In this paper, we introduce HOT, a novel training methodology that reduces memory usage, speeds up computation, and maintains quality. We accelerate the activation gradient path using low-precision HQ while enhancing training stability in the weight gradient path through high-precision HLA quantization. We also improve memory efficiency by compressing activations with ABC and stabilize training with layer-wise quantizers (LQS). As a result, HOT achieves state-of-the-art performance across various vision and language tasks, delivering both high memory efficiency and significant acceleration.


## Acknowledgement

This work was supported by IITP grant funded by the Korea government (MSIT) (Nos. RS-2019-II191906, RS-2023-00228970, RS-2024-00396013, RS-2024-00457882).

## References


[1] Chaim Baskin, Natan Liss, Eli Schwartz, Evgenii Zheltonozhskii, Raja Giryes, Alex M Bronstein, and Avi Mendelson. Uniq: Uniform noise injection for non-uniform quantization of neural networks. *ACM Transactions on Computer Systems (TOCS)*, 37(1-4):1–15, 2021. 7

[2] Chaim Baskin, Evgenii Zheltonozhkii, Tal Rozen, Natan Liss, Yoav Chai, Eli Schwartz, Raja Giryes, Alexander M Bronstein, and Avi Mendelson. Nice: Noise injection and clamping estimation for neural network quantization. *Mathematics*, 9(17):2144, 2021. 4

[3] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22563–22575, 2023. 1

[4] Jianfei Chen, Yu Gai, Zhewei Yao, Michael W Mahoney, and Joseph E Gonzalez. A statistical framework for low-bitwidth training of deep neural networks. *Advances in neural information processing systems*, 33:883–894, 2020. 5

[5] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023. 1

[6] Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben-Yaacov, and Daniel Soudry. Accurate neural training with 4-bit matrix multiplications at standard formats. In *The Eleventh International Conference on Learning Representations*, 2022. 5

[7] Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben-Yaacov, and Daniel Soudry. Accurate neural training with 4-bit matrix multiplications at standard formats. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 2, 3, 6, 7, 8

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding . In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, Los Alamitos, CA, USA, 2016. IEEE Computer Society. 6, 7

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 1, 2, 6, 7, 8

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. 6, 7

[11] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023. 1

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. 1, 2, 4, 6, 7, 8

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models, 2024. 6, 7

[14] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.*, 88(2):303–338, 2010. 6, 7

[15] Junjie Fang, Likai Tang, Hongzhe Bi, Yujia Qin, Si Sun, Zhenyu Li, Haolun Li, Yongjian Li, Xin Cong, Yankai Lin, et al. Unimem: Towards a unified view of long-context large language models. *arXiv preprint arXiv:2402.03009*, 2024. 1

[16] Alireza Ghaffari, Marzieh S Tahaei, Mohammadreza Tayaranian, Masoud Asgharian, and Vahid Partovi Nia. Is integer arithmetic enough for deep learning training? *Advances in Neural Information Processing Systems*, 35:27402–27413, 2022. 2

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 7

[18] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. 1, 2, 6, 7

[19] Glenn Jocher, Alex Stoken, Jirka Borovec, Liu Changyu, Adam Hogan, L Diaconu, F Ingham, J Poznanski, J Fang, L Yu, et al. ultralytics/yolov5: v3. 1-bug fixes and performance improvements. *Zenodo*, 2020. 6, 7

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009. 6, 7

[21] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024. 1

[22] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: vision transformers at mobilenet speed. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024. Curran Associates Inc. 6, 7

[23] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 6

[24] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu,

et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022. 2

[25] Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, Ruihang Li, Miaosen Zhang, Chen Li, Jia Ning, Ruizhe Wang, Zheng Zhang, Shuguang Liu, Joe Chau, Han Hu, and Peng Cheng. Fp8-lm: Training fp8 large language models, 2023. 2

[26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1

[27] Martin Schiemer, Clemens JS Schaefer, Jayden Parker Vap, Mark James Horeni, Yu Emma Wang, Juan Ye, and Siddharth Joshi. Hadamard domain training with integers for class incremental quantized learning. *arXiv preprint arXiv:2310.03675*, 2023. 3

[28] John L Shanks. Computation of the fast walsh-fourier transform. *IEEE Transactions on Computers*, 100(5):457–459, 1969. 3

[29] Shuwei Shi, Wenbo Li, Yuechen Zhang, Jingwen He, Biao Gong, and Yinqiang Zheng. Resmaster: Mastering high-resolution image generation via structural and fine-grained guidance. *arXiv preprint arXiv:2406.16476*, 2024. 1

[30] Juncheol Shin, Junhyuk So, Sein Park, Seungyeop Kang, Sungjoo Yoo, and Eunhyeok Park. Nipq: Noise proxy-based integrated pseudo-quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3852–3861, 2023. 7

[31] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022. 1

[32] James Joseph Sylvester. Lx. thoughts on inverse orthogonal matrices, simultaneous signsuccessions, and tessellated pavements in two or more colours, with applications to newton's rule, ornamental tile-work, and the theory of numbers. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):461–475, 1867. 2

[33] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021. 6, 7

[34] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 6, 7

[35] Vikram Voleti, Alexia Jolicoeur-Martineau, and Chris Pal. Mcvd-masked conditional video diffusion for prediction, generation, and interpolation. *Advances in neural information processing systems*, 35:23371–23385, 2022. 1

[36] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, 2018. Association for Computational Linguistics. 6, 7

[37] Maolin Wang, Seyedramin Rasoulinezhad, Philip HW Leong, and Hayden K-H So. Niti: Training integer neural networks using integer-only arithmetic. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3249–3261, 2022. 2, 5

[38] Xiaoxia Wu, Cheng Li, Reza Yazdani Aminabadi, Zhewei Yao, and Yuxiong He. Understanding int4 quantization for language models: latency speedup, composability, and failure cases. In *International Conference on Machine Learning*, pages 37524–37539. PMLR, 2023. 4

[39] Haocheng Xi, Changhao Li, Jianfei Chen, and Jun Zhu. Training transformers with 4-bit integers. *Advances in Neural Information Processing Systems*, 36:49146–49168, 2023. 2, 3

[40] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023. 4

[41] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems*, pages 12077–12090. Curran Associates, Inc., 2021. 6, 7

[42] Yuedong Yang, Hung-Yueh Chiang, Guihong Li, Diana Marculescu, and Radu Marculescu. Efficient low-rank backpropagation for vision transformer adaptation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 1, 2, 3, 4, 5, 6, 7, 8

[43] Bowen Zhang, Shuyang Gu, Bo Zhang, Jianmin Bao, Dong Chen, Fang Wen, Yong Wang, and Baining Guo. Styleswin: Transformer-based gan for high-resolution image generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11304–11314, 2022. 1