HOT: Hadamard-based Optimized Training

Supplementary Material

A. Overview

This supplementary material provides detailed configurations of our experiments, ablation studies, visualization results, and pre-training experiments across various tasks and datasets.

We provide the following items:

- Detailed hyper-parameters used in our experiments are presented in Appendix B.
- Incremental ablation studies on ABC and LQS, the subordinate methods of HOT, are discussed in Appendix C.1.
- Ablation study on the number of low-pass ranks in HLA is presented in Appendix C.2.
- Ablation study on the combination of HOT and LoRA is detailed in Appendix C.3.
- Theoretical analysis of the computational overhead introduced by HOT is provided in Appendix D.
- Experimental results of pre-training across various tasks and datasets are presented in Appendix E.
- Implementation details of CUDA kernels and latency breakdown are documented in Appendix F.
- Additional visualization results of gradient tensor outlier patterns are presented in Appendix G.

B. Detailed Experimental Settings

In this section, we explain the detailed Hadamard Low-rank Approximation (HLA) configuration of low-pass vector selection and hyper-parameter setting for experiment of each task. All experiments except pre-training are conducted for fine-tuning task.

Hyper-paramemter of HLA: We select top 8 low-pass vectors ($r = 8$) based on LP_{L1} criteria of LBP-WHT[16] for our experiment. The ablation study of rank selection is explained on Appendix C.2. The LP_{L1} is a method for selecting low-pass vectors that simultaneously reflect the frequencies of both vertical and horizontal components of the image. Unlike the conventional sequency order of Hadamard matrix basis that only reflects horizontal components, this approach can effectively filter low-frequency components of image patches from both directions.

Hyper-parameter of experiments: For the classification tasks, we employed three different models: EfficientNetV2[11], EfficientFormer[8], and Vision Transformer (ViT)[3]. EfficientNetV2 was trained with a batch size of 64 and learning rate of 0.001, while both EfficientFormer and ViT utilized a batch size of 128 and learning rate of 0.00025. All models were trained for 50 epochs using the AdamW[9] optimizer and cosine annealing scheduler[10], implemented via the Timm library[13].

Method	Memory	Acceleration	Accuracy
HOT	17.48	2.3×	93.2
HOT + **ABC**	**3.8**	2.3×	93.2
HOT + **ABC** + **LQS**	3.8	**2.6**×	92.99

Table 1. Results of incremental ablation study for ViT-B fine-tuning on CIFAR100. Memory represent theoretical calculations, and Acceleration shows the averaged acceleration rates across all layers of ViT-B.

For object detection, we implemented two approaches. The Segformer-mit-b2[15] model from HuggingFace Transformers[14] was trained with a learning rate of 6e-5 for 50 epochs. YOLO-V5[6], implemented using the official repository, was trained for 40 epochs with a learning rate of 0.00334, using SGD optimizer and linear decay scheduling with warmup.

The language model fine-tuning experiments involved BERT-base and LLama3-8B[4]. BERT-base was trained using HuggingFace Transformers with a batch size of 32, maximum sequence length of 128, and learning rate of 2e-5 for 5 epochs. LLama3-8B, used a smaller batch size of 2 with maximum sequence length of 1024, maintaining the same learning rate for 4 epochs. Both models employed AdamW optimizer and cosine annealing scheduler.

For pre-training, we experimented with ResNet[5], EfficientFormer, and ViT models. ResNet was trained using SGD optimizer with a learning rate of 0.1 and MultiStepLR scheduler for 200 epochs. EfficientFormer utilized AdamW optimizer with a learning rate of 0.001 for 200 epochs, while ViT was trained with a learning rate of 0.0001. Both transformer-based models used cosine annealing scheduler. All pre-training experiments maintained a 128 batch and were implemented using the Timm library. During pre-training, all quantization operators use INT8, including the baseline, at the initial 25 epochs to ensure a stable start of training. After the initial phase, the precision is modulated to the target bit-width, and training continues.

C. Ablation Study

To investigate the impact of various components and hyper-parameters of HOT on model performance, we conducted comprehensive ablation studies.

C.1. Incremental evaluation on each component

We analyzed the effects of ABC and LQS in the g_w path of HOT on model accuracy. The experiments were performed by fine-tuning a ViT-B model on the CIFAR100 dataset

Selected vectors r	Computation Cost	Accuracy
16 (Full rank)	1647.48	76.35
8	1383.54	**76.25**
4	1251.56	73.09
2	1185.58	68.46
1	1152.59	47.28

Table 2. Ablation study of varying the number of low-pass vectors (r) in HLA during pre-training of EfficientFormer-L1 on CIFAR100. The optimal r related with computation cost (Gbops) of backward pass is 8.

with 50 epochs, measuring accuracy, memory consumption, and computational speed while incrementally applying both techniques. Here, Memory refers to theoretical calculations, while Acceleration represents the average GPU acceleration across ViT layers. Note that 'HOT' in this experiment refers to the baseline methodology without ABC and LQS.

The results in Table 1 demonstrate that the combination of ABC and LQS effectively optimizes model performance. Specifically, implementing ABC resulted in a significant reduction in memory usage from 17.48GB to 3.8GB, approximately a 79% decrease. When LQS was additionally applied, the computational speed improved from 2.3× to 2.6× while maintaining the reduced memory footprint. Notably, despite these substantial efficiency improvements, the accuracy degradation was limited to merely 0.5%.

C.2. Rank selection of HLA

We investigated the impact of HLA rank selection on model performance. The experiments were conducted by pre-training EfficientFormer-L1 on the CIFAR100 dataset for 200 epochs, progressively decreasing the number of ranks in powers of two while monitoring performance changes.

As referred to Tab. 2, utilizing 8 ranks in HLA provides optimal accuracy relative to rank reduction. Eight low-frequency vectors appear to be sufficient to represent the spatial information of gradient features. However, further reduction below 4 ranks leads to gradual deterioration, with a particularly sharp decline observed at 2 ranks. Based on these results, we determined that the optimal number of ranks for HLA in HOT implementation is 8.

C.3. LoRA application method

To determine the optimal strategy for combining HOT with LoRA, we conducted ablation study focusing on frozen weights and decomposed weights. The experiments were conducted under the same conditions as the Appendix C.1, analyzing four different configurations based on combinations of HOT to these two weight types.

The results presented in Tab. 3 validate the effectiveness of our proposed HOT-LoRA integration. The configuration

HOT on Frozen weight	HOT on Decomposed weight	Accuracy
✗	✗	92.61
✗	✓	57.96
✓	✗	**92.51**
✓	✓	58.68

Table 3. Experimental results HOT-LoRA combination during fine-tuning ViT-B on CIFAR100. HOT is applied to different combinations of LoRA weight types (Frozen, Decomposed).

applying HOT exclusively to frozen weights achieved the highest accuracy of 92.51% among all tested combinations. In contrast, configurations that applied HOT solely to decomposed or to both decomposed and frozen weights shows significant performance degradation. These findings experimentally demonstrate that direct training of decomposed weights plays a crucial role in model performance.

D. Overhead Calculation

To analyze the computational overhead of HOT, we adopt a standardized notation for layer dimensions. Each layer is represented as a tuple (L, O, I), where L denotes the spatial dimension size, O represents the output channel size, and I indicates the input channel size. For instance, in a layer denoted as (128, 64, 256), the spatial dimension is 128, with 64 output channels and 256 input channels.

The additional transformation, reshaping, and quantization/dequantization processes introduce some overhead to HOT compared to the vanila BP. However, by leveraging low-precision arithmetic, we can achieve practical performance benefits. In this section, we present three tables to explain the computational benefits of our approach in detail.

The added overhead of HOT can be negligible, as shown in Table 5, especially when $\log n$ is sufficiently small relative to other dimensions. In our work, we use $n = 16$ for applying order-4 block-diagonal HT, making this condition valid. For example, in the 'stages.3.fc2' (49, 448, 1792) layer of EfficientFormer-L1, vanilla BP requires 137.3 MFlops, whereas our method only requires 11.5 MFlops. Although HOT incurs some additional overhead, it can achieve significant computational cost reduction through efficient low-precision arithmetic.

E. Pre-training result on various models

In this section, we present additional experimental results on pre-training task, evaluated across various architectures and datasets. As shown in Table 4, HOT consistently outperforms naive INT4 training and previous methods in almost all scenarios, following trend of fine-tuning result.

Dataset	Model	FP	INT4	LUQ [1]	LBP-WHT [16]	HOT
CIFAR10 [7]	ResNet-18 [5]	95.23	93.1	94.73	93.03	**94.77**
	ResNet-34 [5]	95.23	93.57	**94.74**	93.59	93.83
	ResNet-50 [5]	94.98	90.65	**93.62**	92.12	92.85
	EfficientFormer-L1 [8]	95.03	92.9	**94.1**	91.07	94.01
	EfficientFormer-L3 [8]	95.18	93.8	94.63	91.18	**95.01**
CIFAR100 [7]	ResNet-18 [5]	75.66	75.22	**75.63**	72.26	75.53
	ResNet-34 [5]	76.75	72.87	76.26	75.36	**76.95**
	ResNet-50 [5]	76.46	61.28	NaN	69.24	**76.06**
	EfficientFormer-L1 [8]	76.65	64.15	75.79	73.69	**76.06**
	EfficientFormer-L3 [8]	77.26	66.93	**76.19**	63.04	**76.19**
ImageNet-100 [2]	ResNet-18 [5]	86.77	NaN	82.77	82.31	**86.26**
	ResNet-34 [5]	86.87	NaN	82.6	83.31	**86.7**
	ResNet-50 [5]	85.51	NaN	81.45	69.98	**85.2**
	EfficientFormer-L1 [8]	83.38	81.3	82.46	77.52	**83.05**
	EfficientFormer-L3 [8]	83.3	78.09	**83.13**	78.1	83.01
	ViT-B [3]	77.87	NaN	75.97	54.25	**77.31**
ImageNet-1k [2]	ViT-B [3]	70.01	NaN	67.1	NaN	**69.4**

Table 4. Accuracy results for pre-training tasks. In ImageNet training, while other efficiency methods either show substantial degradation (LUQ [1]) or fail to train (LBP-WHT [16]), HOT achieves performance nearly equivalent to FP.

HOT surpasses LBP-WHT [16] in all cases, highlighting the importance of proper optimization of each gradient paths. LUQ [1] achieves competitive performance on the CIFAR10 dataset [7], but it occasionally fails to train or exhibits significantly degraded results on more complex datasets, such as CIFAR100 [7] and ImageNet [2].

Notably, HOT is the only approach that provides stable training while achieving performance comparable to FP training. The extensive pre-training results demonstrate that HOT is a robust and comprehensive solution, applicable not only to fine-tuning but also to pre-trainig tasks.

F. Details of CUDA kernel

Implementation: This section details the implementation of CUDA kernels in HOT. The CUDA kernels consist of five core modules: Hadamard transformation(HT), Hadamard Low-rank Approximation(HLA), Quantization, INT4 and INT8 matrix multiplication, and Dequantization.

The HT and HLA kernel, which inherently has an efficient computational complexity of $O(nlogn)$ with FWHT algorithm, is optimized to maximize computational efficiency by extensively utilizing shared memory of GPU to minimize memory access latency. FWHT is employed not only for HT but also for HLA, where vectors corresponding to low-pass vector indices selected based on the LP_{L1} criterion are extracted from FWHT outputs and concatenated into a single tensor.

The Quantization module implements pseudo-stochastic quantization[12], ensuring unbiased estimation while minimizing quantization overhead.

Both INT4 and INT8 matrix multiplication are implemented using the NVIDIA CUTLASS framework. Unlike CUBLAS, which is closed-source, CUTLASS is open-source and provides extensive customization options for matrix operation configurations, enabling the implementation of optimized matrix multiplication with high throughput. Specifically, to address PyTorch's lack of native INT4 data type support, we efficiently compressed tensors by packing two INT4 values adjacently within an INT8.

Finally, for the Dequantization stage, which requires matrix multiplication in FP32 format, we utilized NVIDIA CUBLAS through PyTorch's default matrix multiplication implementation.

Latency breakdown: In this section, we conduct a detailed latency analysis of five core modules in CUDA kernels to understand the specific mechanisms of computational acceleration. Fig. 1 presents a comparative analysis of latencies across FP operations, LBP-WHT, and HOT kernels for representative layers showing average acceleration in ResNet-50, ViT-B, and EfficientFormer-L7 architectures. Specifically, HOT achieved 1.9× acceleration in ResNet-50's 'layer4.conv2' (49, 512, 4608), 2.6× acceleration in ViT-B's 'qkv' (197, 2304, 768), and 1.9× acceleration in EfficientFormer-L7's 'stages.1.fc1' (784, 768, 192).

The experimental results demonstrate that integer matrix multiplication significantly reduced the latency compared to FP32 across all target models. A noteworthy case

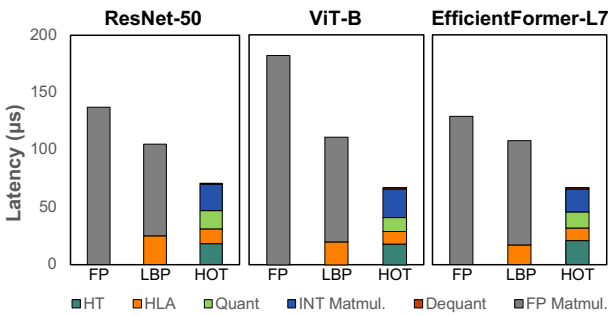

Figure 1. Comparison of kernel latency (μs) between FP32, LBP-WHT, and HOT for representative layers of ResNet-50, ViT-B, and EfficientFormer-L7 models. The selected representative layers can be found in the 'Latency breakdown' section of Appendix F.

is observed in the ViT-B model, where FP32 matrix multiplication shows $182\mu s$, while integer matrix multiplication consumes only $25\mu s$. This substantial performance improvement can be attributed to the synergistic effect of low-precision integer matrix multiplication combined with tensor size reduction through HLA.

Meanwhile, the HT and HLA modules in ViT-B show latencies of $18\mu s$ and $11\mu s$ respectively, resulting in approximately 16% computational overhead compared to FP operations. This figure exceeds the theoretical overhead prediction of 7% calculated in Appendix D. This discrepancy can be attributed to additional operations required in practical implementation beyond FWHT algorithm operations, including transpose, reshape, and contiguous operations. Consequently, we anticipate potential further latency reductions through the optimization techniques such as kernel fusing.

G. Various case of output gradient tensor

We visualized two distinctive outlier patterns of output gradient g_y that form the theoretical baseline of LQS, extending our analysis to various layers within ViT-S and ResNet34. Fig. 2 presents 3D visualizations from ImageNet-1k training with LQS, categorized layers into (a) per-token quantization friendly and (b) per-tensor quantization friendly case.

The analysis revealed distinct pattern across models. In ViT-S, fc2 layers and attention projection layers consistently exhibited token-wise gradient outliers, demonstrating it is suited for per-token quantization. Similarly, conv1 layers across different stages in ResNet-34 displayed non-zero gradient value at the token level, indicating the effectiveness of per-token quantization for these layers.

Conversely, some layers exhibited contrasting characteristics. In the case of ViT-S's fc1 layers, there was a significant reduction in the magnitude of token-level non-zero gradients compared to fc2 layers, with notably sparse occurrence patterns, suggesting limited effectiveness of per-token quantization. Certain conv1 and conv2 layers in ResNet-34 showed large non-zero gradients at irregular positions independent of token locations, indicating limited advantages in terms of quantization error when applying per-token quantization.

These analysis results empirically demonstrate the necessity for different quantization strategies based on layer characteristics, suggesting that LQS represents an optimized approach that considers characteristics of each layer.

Name	FLOPs
Vanilla BP	$4LIO$
g_x	$2LOlogn + 2IOlogn + 2LO + 2IO$
g_w	$2LIlogn + 2LOlogn + 2I(L * \frac{r}{n}) + 2O(L * \frac{r}{n})$
Dequant	$2IO + 2LI$

Table 5. The additional FLOPs induced by optimization path.

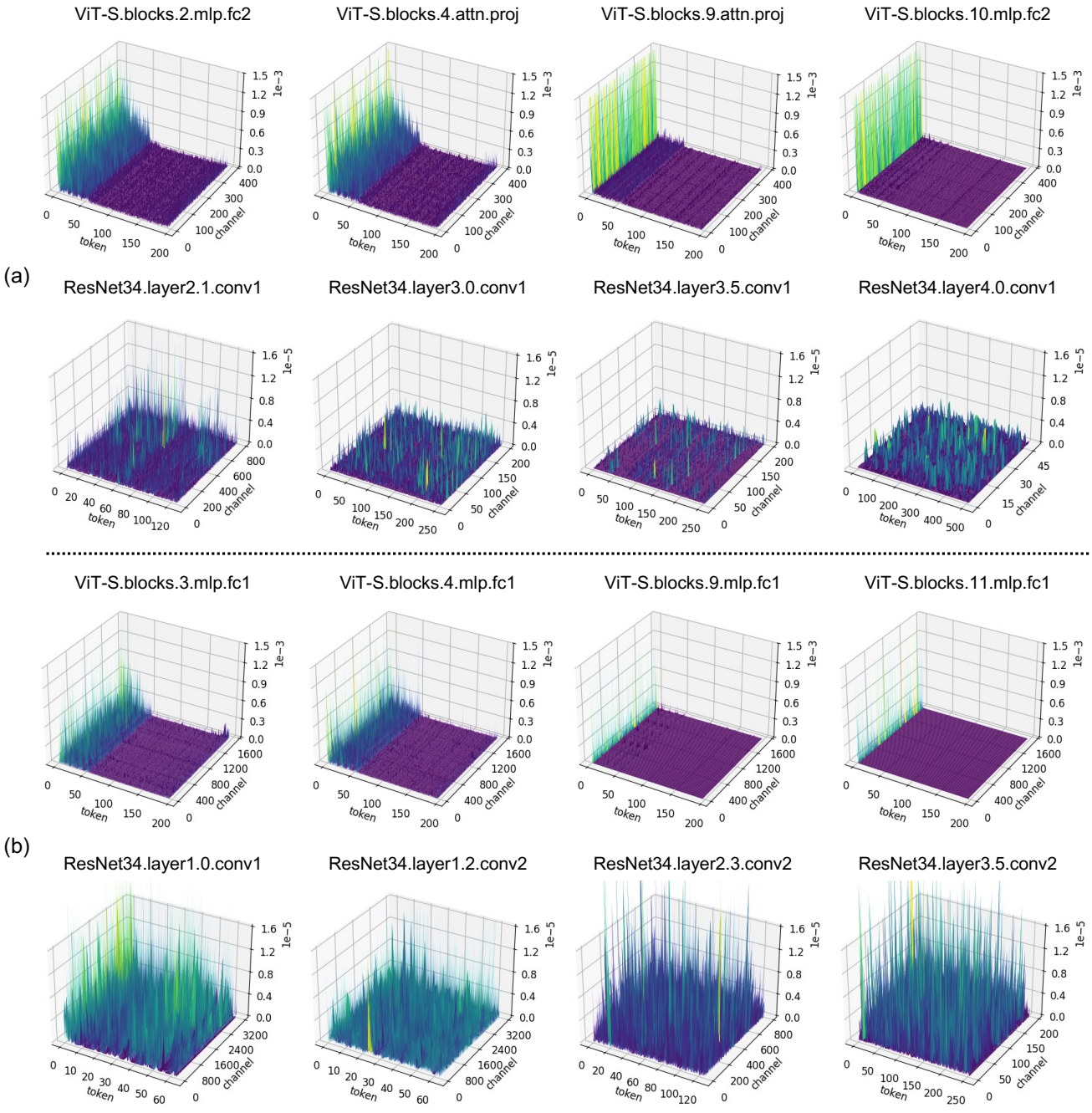

Figure 2. The illustration for output gradient of (a) Per-token quantization friendly case and (b) Per-tensor quantization friendly case.

References

[1] 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. 3

[2] 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. 3

[3] 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, 3

[4] 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. 1

[5] 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. 1, 3

[6] 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. 1

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009. 3

[8] 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. 1, 3

[9] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 1

[10] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 1

[11] 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. 1

[12] 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. 3

[13] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 1

[14] T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019. 1

[15] 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. 1

[16] 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, 3