Orthogonal Finetuning Made Scalable

Anonymous ACL submission

Abstract

Orthogonal finetuning (OFT) offers highly parameter-efficient adaptation while preventing catastrophic forgetting, but its high runtime and memory demands limit practical deployment. We identify the core computational bottleneck in OFT as its weight-centric implementation, which relies on costly matrix-matrix multiplications with cubic complexity. To overcome this, we propose OFTv2, an input-centric reformulation that instead uses matrix-vector multiplications (*i.e.*, matrix-free computation), reducing the computational cost to quadratic. We further introduce the Cayley–Neumann parameterization, an efficient orthogonal parameterization that approximates the matrix inversion in Cayley transform via a truncated Neumann series. These modifications allow OFTv2 to achieve up to 10× faster training and 3× lower GPU memory usage without compromising performance. In addition, we extend OFTv2 to support finetuning quantized foundation models and show that it outperforms the popular QLoRA in training stability, efficiency, and memory usage.

1 Introduction

As foundation models continue to improve in performance, recent years have witnessed a paradigm shift from end-to-end learning to a pretraining-finetuning framework. This shift underscores the need for finetuning methods that are both effective and scalable. Owing to its training stability and adaptation efficiency, orthogonal finetuning (OFT) (Qiu et al., 2023; Liu et al., 2024) has emerged as a promising approach for adapting foundation models to downstream tasks. However, while performing well, OFT incurs high computational and memory costs, limiting its scalability. Motivated by these challenges, we seek to make OFT more scalable to large foundation models.

Towards this goal, we begin by identifying the key bottleneck that limits OFT's scalability. At its core, OFT learns layer-shared orthogonal matrices to transform pretrained weight matrices, resulting in a naive *weight-centric* implementation where forward inference is performed after merging the learned orthogonal matrices into weight matrices during training. The weight-centric implementation thus involves matrix-matrix multiplications with cubic complexity. As weight matrices grow large, this cubic scaling severely limits OFT's applicability to large foundation models. However, these matrix-matrix multiplications are not fundamentally necessary. We draw inspiration from matrix-free methods (Chen, 2005), such as the power method and the Lanczos algorithm, which avoid explicit matrix-matrix operations by treating matrices as linear operators applied to vectors. These methods operate entirely through matrix-vector multiplications, applying a matrix to vectors in the appropriate space without ever forming full matrix products. Guided by the same insight, we introduce an *input-centric* implementation of OFT, in which the learned orthogonal transformations are applied directly to the input vectors during each forward pass, rather than being merged into the weight matrix. This reformulation reduces the complexity from cubic to quadratic. We refer to this new formulation as OFTv2. Despite its simplicity, this change significantly enhances the scalability of

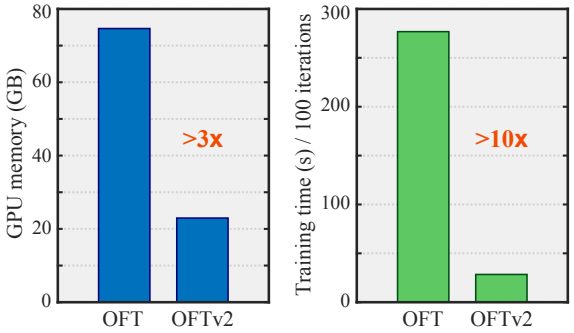


Figure 1: OFTv2 significantly reduces training time and GPU memory usage without sacrificing performance. The finetuning is performed with Qwen2.5-7B.

OFT, making it suitable for finetuning large foundation models that the original OFT could not handle due to memory constraints.

Another scalability bottleneck in OFT arises from the Cayley parameterization used by Qiu et al. (2023); Liu et al. (2024) to preserve orthogonality. While effective, this parameterization involves computing a matrix inverse, which becomes increasingly costly and less numerically stable as weight matrices get larger. To address this, we introduce a numerically stable yet efficient approximation – the Cayley–Neumann parameterization (CNP). By replacing the matrix inverse in the original Cayley transform with a truncated Neumann series, CNP offers improved numerical stability and significantly lower computational cost, particularly in settings where OFT is applied to finetune large foundation models. With CNP, OFTv2 becomes even more scalable and readily applicable for efficient adaptation of such models. In Figure 1, we compare OFT and OFTv2 by performing finetuning tasks on Qwen2.5-7B, which is the largest model the original OFT can finetune within a single Nvidia H100 (80GB). The results show that OFTv2 achieves substantial GPU memory savings and training speed-up over the original OFT formulation (Qiu et al., 2023).

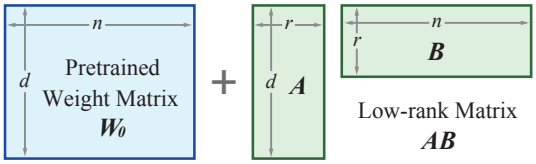
In practice, finetuning ultra-large foundation models (*e.g.*, LLaMA 3.1-70B (Grattafiori et al., 2024), Qwen 2.5-72B (Yang et al., 2024a)) typically requires quantization to fit within GPU memory limits. To support this, we follow the general design of the QLoRA framework (Dettmers et al., 2023) but replace LoRA with OFTv2. Our input-centric implementation of orthogonal finetuning enables a seamless application to the finetuning of quantized foundation models, resulting in QOFT–an efficient orthogonal finetuning that enables efficient adaptation of quantized ultra-large models. Our major contributions are summarized below:

- Inspired by matrix-free methods that avoid matrix-matrix multiplications in solving linear systems, we propose OFTv2–an input-centric reformulation of OFT that achieves significantly better scalability, with more than 10× faster training and 3× lower GPU memory usage.
- We introduce the Cayley–Neumann parameterization, which approximates the Cayley transform with a truncated Neumann series and eliminates numerically unstable matrix inversions.
- Owing to the new input-centric formulation, we adapt OFTv2 to finetuning quantized foundation models. This enables memory-efficient finetuning of ultra-large models.
- We apply OFTv2 and its quantized variant to different foundation models (including large language models and text-to-image generative models) across various model scale.

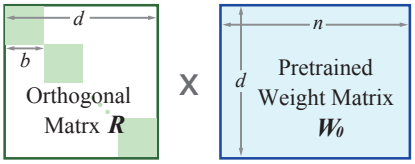
2 Related Work

Parameter-efficient finetuning (PEFT). As foundation models become increasingly large and powerful, there has been growing interest in finetuning them for downstream tasks in a parameter-efficient manner (Houlsby et al., 2019; Aghajanyan et al., 2020; Hu et al., 2022a; Edalati et al., 2022; Wang et al., 2022; Gheini et al., 2021; Zaken et al., 2022; Guo et al., 2020; Sung et al., 2021; Ansell et al., 2022; Lester et al., 2021; Li and Liang, 2021; Vu et al., 2022; He et al., 2021; Mao et al., 2021; Karimi Mahabadi et al., 2021; Liu et al., 2022; Sung et al., 2022; Chen et al., 2023; Jia et al., 2022; Chen et al., 2022; Zhang et al., 2022; Jie and Deng, 2023; Lian et al., 2022; Luo et al., 2023; Zhang et al., 2024; Wu et al., 2024). In particular, reparameterization-based methods (*e.g.*, Aghajanyan et al. (2020); Hu et al. (2022a); Edalati et al. (2022); Zi et al. (2023); Chavan et al. (2023)) are enjoying wide adoption. LoRA (Hu et al., 2022a) learns a pair of small low-rank matrices whose product is added to each weight matrix, enabling task adaptation with a small number of trainable parameters. Building on LoRA, several works dynamically adjust the rank across layers to better balance the parameter budget (Zhang et al., 2023b; Valipour et al., 2022; Zhang et al., 2023a, 2024). To improve scalability, QLoRA (Dettmers et al., 2023) quantizes the frozen base model to 4-bit NormalFloat with double quantization and back-propagates only through LoRA, achieving near full-precision accuracy while drastically lowering memory usage.

Orthogonal Finetuning (OFT). Qiu et al. (2023); Liu et al. (2024) propose a reparameterization-based method that learns an orthogonal matrix to transform the neurons within the same layer, yielding strong generalization and stable finetuning. It is motivated by the idea that hyperspherical energy (*i.e.*, a function of the geometric relationships among neurons on the unit sphere) influences generalization (Liu et al., 2018, 2021b), and that orthogonal transformations keep this energy invariant (Liu et al., 2021a). A growing body of research (Ma



(a) Low-rank Structure in LoRA



(b) Sparse Orthogonal Structure in OFT

Figure 2: Comparison between LoRA and OFT.

et al., 2024; Yang et al., 2024b; Gorbunov et al., 2024; Yuan et al., 2024; Feng et al., 2025; Raj and Coyle, 2025; Lingam et al., 2024; Bini et al., 2024; Liao and Monz, 2024) builds upon the core idea of OFT. Figure 2 provides an high-level comparison of OFT and LoRA. While OFT achieves parameter efficiency via sparsity, LoRA leverages low rank.

3 OFTv2: Faster and More Scalable

3.1 Preliminaries

Let $\boldsymbol{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_n] \in \mathbb{R}^{d\times n}$ be a weight matrix with columns $\boldsymbol{w}_i \in \mathbb{R}^d$. In a linear layer, the forward pass is $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^d$ is the input and $\boldsymbol{z} \in \mathbb{R}^n$ is the output. OFT reparameterizes the weight matrix with $\boldsymbol{W}_{\text{OFT}} = \boldsymbol{R}\boldsymbol{W}_0$ where $\boldsymbol{W}_0$ is the pretrained weight matrix and $\boldsymbol{R} \in \mathbf{R}^{d\times d}$ is an orthogonal matrix. OFT only learns $\boldsymbol{R}$ for adapting the pretrained model to downstream tasks. To enforce orthogonality, Liu et al. (2021b); Qiu et al. (2023); Liu et al. (2024) parameterize $\boldsymbol{R}$ using the Cayley transform: $\boldsymbol{R} = (\boldsymbol{I}+\boldsymbol{Q})(\boldsymbol{I}-\boldsymbol{Q})^{-1}$, where $\boldsymbol{Q}$ is a skew-symmetric matrix satisfying $\boldsymbol{Q} = -\boldsymbol{Q}^\top$. To further improve parameter-efficiency, OFT constrains the orthogonal matrix $\boldsymbol{R}$ to have a block-diagonal structure: $\boldsymbol{R} = \text{Diag}(\boldsymbol{R}_1, \cdots, \boldsymbol{R}_r)$ where for any i, $\boldsymbol{R}_i \in \mathbb{R}^{b\times b}$ is a small orthogonal matrix and $b\cdot r = d$. Each $\boldsymbol{R}_i$ can be parameterized using the Cayley transform. This block-diagonal form imposes a sparsity pattern on $\boldsymbol{R}$, effectively making it a sparse orthogonal matrix. Leveraging this structure, Liu et al. (2024) further enhance parameter efficiency using butterfly factorization.

3.2 From Weight-centric Implementation to Input-centric Implementation

OFT performs finetuning by learning an orthogonal matrix to directly transform the weight matrix, which naturally leads to a weight-centric implementation of the forward pass:

$$\boldsymbol{z} = \underbrace{\overbrace{\boldsymbol{W}_0^\top \boldsymbol{R}^\top}^{\text{(1) Weight transform: matrix-matrix mult.}} \boldsymbol{x}}_{\text{(2) Linear map: matrix-vector mult.}} \quad (1)$$

The original OFT first performs a weight transform by computing $\boldsymbol{W}_{\text{OFT}}^\top = \boldsymbol{W}_0^\top \boldsymbol{R}^\top$ (*i.e.*, a matrix-matrix multiplication) and then computes the results of a linear layer with the equivalent weight matrix $\boldsymbol{W}_{\text{OFT}}^\top$ (*i.e.*, a matrix-vector multiplication). This incurs $\mathcal{O}(nd^2)$ complexity due to the matrix-matrix multiplication. Inspired by matrix-free methods for solving linear systems, we observe that OFT's forward pass can be interpreted as two linear maps applied to the input. This leads to an input-centric implementation

$$\boldsymbol{z} = \underbrace{\boldsymbol{W}_0^\top \overbrace{\boldsymbol{R}^\top \boldsymbol{x}}^{\text{(1) Linear map: matrix-vector mult.}}}_{\text{(2) Linear map: matrix-vector mult.}} \quad (2)$$

where only two matrix-vector multiplications are required, reducing the complexity from cubic to quadratic: $\mathcal{O}(nd + d^2)$. This simple conceptual shift in implementation entails substantial speed-up in training time and reduction in GPU memory.

3.3 Approximate Orthogonality via Cayley-Neumann Parameterization

The Cayley parameterization constructs an orthogonal matrix $\boldsymbol{R}$ as $\boldsymbol{R} = (\boldsymbol{I}+\boldsymbol{Q})(\boldsymbol{I}-\boldsymbol{Q})^{-1}$, where $\boldsymbol{Q}$ is a skew-symmetric matrix. One limitation of this formulation is that it only generates rotation matrices, though empirical studies (Liu et al., 2021a; Qiu et al., 2023; Liu et al., 2024) suggest that this restriction does not negatively affect performance. More critically, computing a matrix inverse introduces numerical instability and additional computational overhead, making it challenging to scale to large orthogonal matrices. To avoid numerical instability, we replace the matrix inverse with a truncated Neumann series:

$$\begin{aligned}\boldsymbol{R} &= (\boldsymbol{I}+\boldsymbol{Q})(\boldsymbol{I}-\boldsymbol{Q})^{-1} = (\boldsymbol{I}+\boldsymbol{Q})\Big(\sum_{i=0}^{\infty}\boldsymbol{Q}^i\Big) \\ &\approx (\boldsymbol{I}+\boldsymbol{Q})\Big(\boldsymbol{I}+\sum_{i=1}^{k}\boldsymbol{Q}^i\Big),\end{aligned} \quad (3)$$

where larger k leads to better approximation. Removing the matrix inversion improves training stability. The Neumann series approximation converges in the operator norm if $\|\boldsymbol{Q}\| < 1$. This

condition is naturally satisfied in practice: to start from the pretrained model, OFT initializes the orthogonal matrix $\boldsymbol{R}$ as the identity, which requires $\boldsymbol{Q}$ to start as a zero matrix. Since finetuning begins with a small learning rate and typically involves relatively few steps, $\boldsymbol{Q}$ tends not to drift far from zero. Empirically, even if $\|\boldsymbol{Q}\|$ slightly exceeds 1, it does not harm OFT's training stability, as we use only a finite number of Neumann terms.

CUDA kernel for skew-symmetric matrices. To maximize GPU memory efficiency, we leverage the skew-symmetric structure of $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, where $Q_{ii} = 0$, $Q_{ij} = -Q_{ji}$. By storing only the upper triangular part as a vector, we reduce the storage requirement from n^2 to $\frac{n(n-1)}{2}$. During the forward pass, $\boldsymbol{Q}$ is reconstructed on-the-fly using a highly optimized custom CUDA kernel that significantly accelerates this process.

4 QOFT: Adapting OFTv2 to Finetuning Quantized Foundation Models

While PEFT methods primarily aim to reduce optimizer memory by minimizing trainable parameters, the growing scale of foundation models has shifted the memory bottleneck to the pretrained weights themselves. As model dimensions grow, these frozen parameters increasingly dominate memory consumption during training (Kim et al., 2023). To address this emerging challenge, we argue that truly scalable OFT must operate directly on quantized model representations, such as NormalFloat4 (Dettmers et al., 2023) and AWQ (Lin et al., 2024). This represents a critical shift that enables OFT to scale effectively.

To this end, we introduce QOFT, a natural extension of OFTv2 for quantized foundation models. QOFT largely follows the framework of QLoRA (Dettmers et al., 2023). Specifically, the quantized low-bit weight matrices are first dequantized to higher precision, after which the parameter-efficient adaptation is carried out in the higher-precision space. Formally, the forward pass of QOFT can be written as

$$\boldsymbol{z} = \underbrace{\text{Dequant}(\boldsymbol{W}_{\text{quant}})^\top}_{\text{Fronzen}} \underbrace{\boldsymbol{R}^\top}_{\text{Trainable}} \boldsymbol{x} \qquad (3)$$

The update of OFTv2's orthogonal matrix $\boldsymbol{R}$ is performed in high precision (e.g., BF16). We denote the dequantization function as $\text{Dequant}(\cdot)$ and follow QLoRA's design by adopting a double quantization strategy, where the quantization parameters of the weight matrices are themselves quantized to further reduce GPU memory usage.

Flexible quantized finetuning via OFTv2. We now explain why the weight-centric implementation of OFT is ill-suited for quantized foundation models. Computing the matrix product $\boldsymbol{W}_{\text{quant}}^\top \boldsymbol{R}^\top$ involves rotating (or reflecting) a quantized weight matrix, which requires first dequantizing it to higher precision before applying the transformation. While this is mathematically valid, it makes OFT dependent on the specific quantization method used. Different quantization schemes may require different treatments for computing $\text{Dequant}(\boldsymbol{W}_{\text{quant}})^\top \boldsymbol{R}^\top$, introducing unnecessary complexity. In contrast, the input-centric implementation avoids this issue by fully decoupling OFT from weight quantization. It applies the learned orthogonal matrix $\boldsymbol{R}^\top$ to the input $\boldsymbol{x}$. The subsequent forward pass proceeds as usual under any quantization strategy. As a result, OFTv2 becomes a quantization-agnostic PEFT method compatible with arbitrary weight quantization schemes.

QOFT vs. QLoRA. We now look into the forward pass of QLoRA: $\boldsymbol{z} = \text{Dequant}(\boldsymbol{W}_{\text{quant}})^\top \boldsymbol{x} + (\boldsymbol{AB})^\top \boldsymbol{x}$ where $\boldsymbol{A} \in \mathbb{R}^{d\times r}$ and $\boldsymbol{B} \in \mathbb{R}^{r\times n}$ are low-rank matrices and $r \ll \min(d, n)$ is usually quite small. First, QOFT is more suitable for post-training quantization when merging the finetuned weights back into the quantized model. In QLoRA, the equivalent weight $\boldsymbol{W} + \boldsymbol{AB}$ can alter the dynamic range (*i.e.*, the possible minimum and maximum values) of the weight matrix, potentially complicating requantization. In contrast, the equivalent weight in QOFT, $\boldsymbol{RW}$, preserve the dynamic range of individual elements. The worse-case requantization error for QLoRA is always larger than QOFT by $\|\boldsymbol{AB}\|_\infty$. This advantage is also partially supported by recent evidence (Tseng et al., 2024; Ashkboos et al., 2024) suggesting that orthogonal transformations can homogenize weight magnitudes and suppress outliers.

Another practical limitation of QLoRA is its training instability. Across various experiments, we observe that QLoRA is prone to loss divergence and unstable optimization. We suspect this arises from the inherently noisier gradients in QLoRA, which adversely affect the finetuned weights. In contrast, QOFT benefits from the orthogonality of $\boldsymbol{R}$, which also regularizes the back-propagated gradients. As a result, the adaptation weights in QOFT are better conditioned, and when merged into the pretrained model, they yield a more sta-

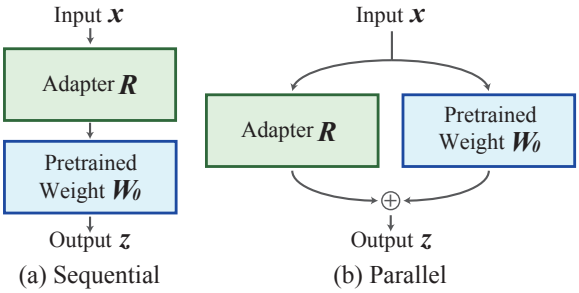


Figure 3: Comparison between sequential (*e.g.*, OFT) and parallel (*e.g.*, LoRA) adaptation.

ble finetuned model. This observation is supported by prior work (Qiu et al., 2023; Liu et al., 2024) showing that OFT significantly improves training stability and mitigates catastrophic forgetting.

5 Discussion

Sparse vs. low-rank PEFT. As shown in Figure 2, OFT and LoRA achieve parameter-efficiency through sparsity and low rank, respectively. This suggests an intriguing analogy between OFT and LoRA, as sparsity and low rank represent arguably two of the most widely studied and exploited structural properties in matrices. To further enhance the scalability of OFT, more structured sparsity should be exploited, *e.g.*, butterfly factorization (Liu et al., 2024). Moreover, similar to AdaLoRA (Zhang et al., 2023c), the sparsity level in OFT can be conditioned on the task and layer. Compared to low-rank PEFT, sparse PEFT approaches like OFT remain relatively underexplored, leaving many interesting open problems for future investigation.

Sequential vs. parallel adaptation. As shown in Figure 3, OFT and LoRA exemplify two distinct adaptation strategies: sequential adaptation and parallel adaptation, respectively. This contrast is particularly intriguing, as it explains why sequential adaptation benefits from orthogonality, while parallel adaptation naturally aligns with low rank. Sequential adaptation offers great expressiveness but is also more susceptible to error propagation and distortion of the pretrained model's spectral properties. Enforcing orthogonality on $\boldsymbol{R}$ is therefore a natural choice, as it preserves these properties and helps prevent the accumulation of errors. Sparsity is the natural choice if we want to save parameters in orthogonal matrices. Parallel adaptation adds the adapter $\boldsymbol{R}$ to the pretrained model. In this case, we want $\boldsymbol{R}$ to be a dense update while maintaining parameter efficiency–a goal naturally achieved through low-rank matrices. This perspective may inspire new directions in adapter design.

Efficient orthogonality parameterization. OFT also highlights the importance of efficient parameterization of orthogonal matrices. In fact, the efficiency is closely tied to two factors: (1) the degree to which orthogonality needs to be approximated, and (2) the size of the set of orthogonal matrices considered. Our experiments indicate that exact orthogonality and the full orthogonal group are not strictly necessary, as parameterizations from the special orthogonal group and approximate orthogonality perform equally well in practice. This raises an open question: can we find even more efficient parameterizations with comparable performance?

6 Experiments on Scalability

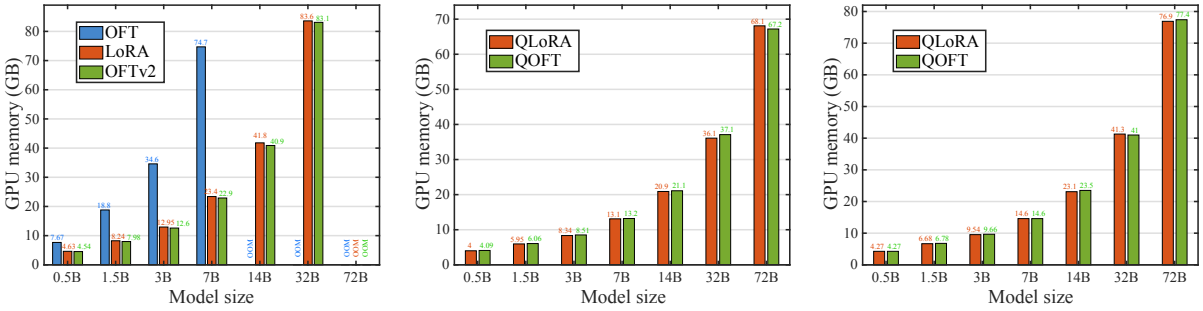
Our experiments systematically evaluate OFTv2 along two key dimensions: (1) its scalability improvements over the original OFT, and (2) its finetuning performance across a diverse set of tasks from multiple domains. For both aspects, we compare OFTv2 and QOFT against the well-established, memory- and compute-efficient low-rank adaptation methods LoRA (Hu et al., 2022b) and QLoRA (Dettmers et al., 2023).

6.1 GPU Memory Efficiency

As depicted in Figure 1, OFTv2 achieves a $3\times$ reduction in **GPU memory consumption** compared to the original OFT when finetuning the Qwen2.5-7B model. Furthermore, QOFT significantly reduces memory consumption by enabling the orthogonal finetuning of quantized base models. In the following ablation studies comparing against both LoRA and QLoRA baselines – where we define QLoRA broadly as low-rank adaptation of quantized models without restricting to NormalFloat 4-bit quantization – we evaluate the actual GPU memory consumption during finetuning of Qwen2.5 models across scales from 0.5B to 72B parameters. For a comprehensive analysis, we incorporate the widely adopted quantization method AWQ (Lin et al., 2024) for activation-aware quantization. The results are summarized in Figure 4. Our experimental results demonstrate that OFTv2 and QOFT achieve memory efficiency comparable to low-rank adaptation methods, with consistent performance across model scales and data formats.

6.2 Computational Efficiency

We begin by evaluating the training speed of OFTv2 relative to the original OFT. To this end,



(a) Original Qwen2.5 (a) BnB-quantized Qwen2.5 (c) AWQ-quantized Qwen2.5

Figure 4: Results of GPU memory usage for the same finetuning task. (a) OFT, LoRA and OFTv2 on Qwen2.5; (b) QLoRA and QOFT on NF4-quantized Qwen2.5; (c) QLoRA and QOFT on AWQ-qunatized Qwen2.5.

Model Size	GPUs	LoRA	OFTv2
Llama-2-7B	8×H100	**00:12:10**	00:15:10
Llama-2-13B	8×H100	**00:17:00**	00:19:50

Table 1: Training time (clock time) comparison: OFTv2 vs. LoRA on GSM8K for mathematical reasoning.

Model Size	GPUs	QLoRA	QOFT
Qwen2.5-1.5B	8×H100	01:20:00	**01:17:30**
Qwen2.5-7B	8×H100	03:25:00	**03:19:30**
Qwen2.5-32B	8×H100	12:51:45	**12:27:45**

Table 2: Clock time comparison of QOFT and QLoRA on OpenR1-Math-220k for mathematical reasoning.

we fine-tune a Qwen2.5-7B model on the OASST1-Guanaco-9K dataset (Dettmers et al., 2023) for instruction following and measure the training time. As shown in Figure 1, OFTv2 achieves a 3× speed-up over the original OFT. We further compare the end-to-end training speed of OFTv2 and LoRA across different model scales and precisions. Results from the GSM8K experiment (Table 4) and the OpenR1-Math-220k experiment (Face, 2025) (Table 5) are used for comparison. Clock times for each setting are reported in Table 1 and Table 2. While low-rank adaptation methods like LoRA benefit from PyTorch's highly optimized GEMM operations via NVIDIA cuBLAS/cuDNN libraries, the simple designs in OFTv2 significantly narrow this optimization gap in full-precision settings. Notably, OFTv2 outperforms LoRA in quantized settings (Table 2), demonstrating that its quantization-agnostic design effectively leverages underlying quantization-layer optimizations.

7 Experiments on Performance

Having established that OFTv2 achieves comparable memory and computational efficiency to low-rank adaptation methods, we then test its performance on a variety of tasks.

7.1 Encoder-Decoder Model: BART

We evaluate the finetuning of BART-large (Lewis et al., 2019) on the XSum (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets for text summarization, reporting ROUGE-1/2/L scores for LoRA and OFTv2 under both full-precision and NormalFloat4 4-bit quantization. We further investigate different configurations by increasing the rank r for LoRA and the block size b for OFTv2. The results from these finetuning tasks are reported in Table 3. We observe that OFTv2/QOFT consistently outperforms LoRA/QLoRA across all tested configurations, while notably utilizing 47–53% fewer trainable parameters. The performance gap gets more obvious with increasing model capacity: at the maximum parameter budget, QOFT outperforms QLoRA by +0.93 ROUGE-1 on XSum (44.16 vs. 43.23), suggesting a more effective utilization of expanded adapters. Furthermore, the finetuning performance of OFTv2/QOFT improves accordingly with an increase in trainable parameters.

7.2 Decoder-only Model: Llama-2 Series

We finetune Llama-2 7B and 13B models on the NLG datasets GSM8K (Cobbe et al., 2021) and WikiText-2 (Merity et al., 2016). To ensure fairness, we use the same set of hyperparameters for each method across datasets, precisions, and model scales. Both LoRA and QLoRA set rank to $r = 16$. Both OFTv2 and QOFT set block size to $b = 32$. Table 4 shows that OFTv2 consistently outperforms the low-rank adapter across different settings.

7.3 Decoder-only Model: Qwen2.5 Series

We perform supervised finetuning on the Huggingface OpenR1-Math-220k (Face, 2025) dataset—a

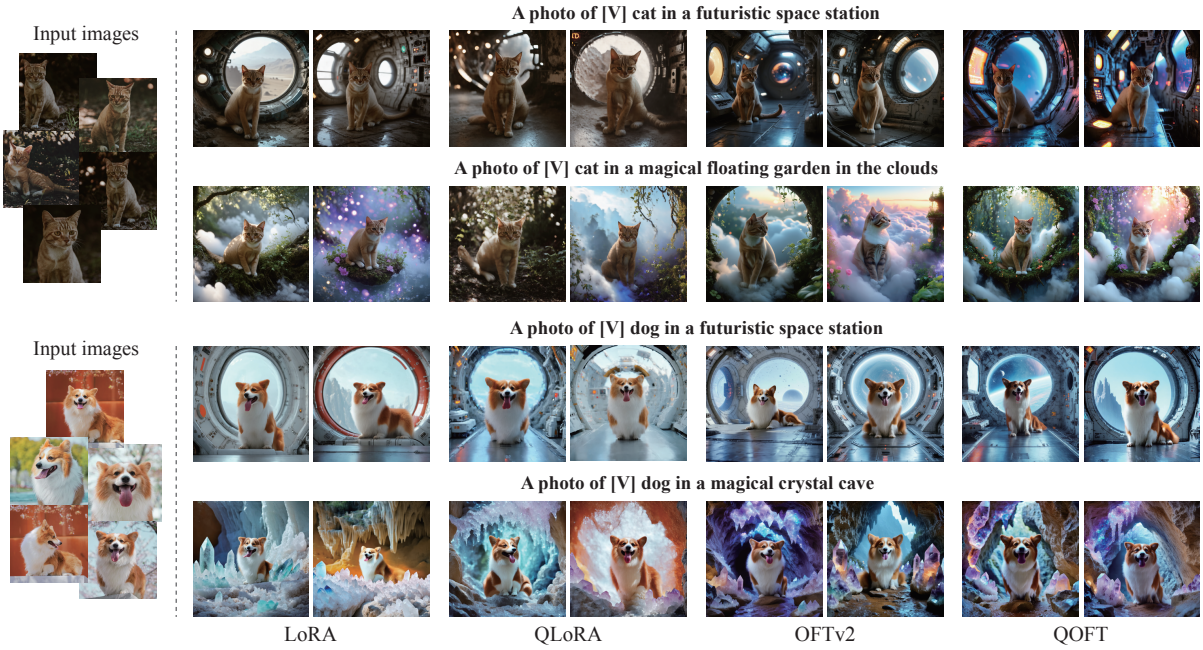


Figure 5: Qualitative results from Dreambooth finetuning of Stable Diffusion 3.5 Large (8.1B parameters), with peak allocated GPU memory: LoRA (**52.33 GB**), OFT (**52.32 GB**), QLoRA (**41.60 GB**) and QOFT (**41.53 GB**).

Quant.	LoRA / QLoRA			OFTv2 / QOFT		
	# Params	XSum↑	CNN/DailyMail↑	# Params	XSum↑	CNN/DailyMail↑
Full Prec.	4.33M	43.33/20.06/35.11	43.11/20.22/29.69	2.03M	**43.36/20.21/35.31**	**43.27/20.29/29.71**
	8.65M	43.47/20.19/35.21	43.20/20.31/29.71	4.19M	**43.85/20.69/35.83**	**43.72/20.73/30.22**
	17.30M	43.38/20.20/35.25	43.17/20.31/29.72	8.52M	**44.12/20.96/36.01**	**44.08/21.02/30.68**
NF4	4.33M	43.09/19.82/34.92	43.17/20.25/29.66	2.03M	**43.10/19.92/35.00**	**43.31/20.37/29.74**
	8.65M	43.15/19.80/34.92	43.10/20.24/29.65	4.19M	**43.72/20.58/35.68**	**43.71/20.74/30.22**
	17.30M	43.23/19.92/35.10	43.11/20.23/29.63	8.52M	**44.16/20.98/36.09**	**44.10/21.05/30.69**

Table 3: ROUGE-1, ROUGE-2, and ROUGE-L scores for BART-large fine-tuned on XSum and CNN/DailyMail.

Model	Metric	16-bit		4-bit	
		LoRA	OFTv2	QLoRA	QOFT
7B	# Params	39.98M	17.65M	39.98M	17.65M
	WikiText-2↓	6.63	**6.14**	5.74	**5.60**
	GSM8K↑	33.81	**34.65**	34.12	**37.23**
13B	# Params	62.59M	27.62M	62.59M	27.62M
	WikiText-2↓	5.23	**4.98**	5.31	**5.05**
	GSM8K↑	45.94	**46.02**	44.20	**47.92**

Table 4: Finetuning results of Llama-2 models on WikiText-2 (perplexity) and GSM8K (test accuracy).

large-scale mathematical reasoning corpus containing challenging problems and two to four reasoning traces distilled from DeepSeek R1 (Guo et al., 2025). Following the evaluation protocol of Qwen2.5-Math (Yang et al., 2024a), we report pass@1 performance on established math benchmarks: CMATH (Wei et al., 2023), AMC23 (Project-Numina), AQUA (Ling et al., 2017), Olympiad Bench (He et al., 2024), Gaokao 2023 En (Liao et al., 2024), and Minerva Math (Lewkowycz et al., 2022). Finetuning was only performed on NormalFloat 4-bit quantized base models due to the substantial memory requirements imposed by the large context window size (16384), necessary for training on a reasoning dataset. The results are reported in Table 5. The baseline type refers to the pre-trained Qwen2.5 models without any continual training. We observe that QOFT consistently outperforms both QLoRA and baseline models across all evaluated scales and tasks, despite using significantly fewer trainable parameters. For instance, on the Qwen2.5-7B instruction-tuned model, QOFT achieves a 96.9% SAT Math accuracy compared to QLoRA's 68.8%, while utilizing only 17.55M parameters (57% fewer than QLoRA's 40.37M). This advantage scales robustly: the Qwen2.5-32B variant fine-tuned with QOFT attains 100% SAT Math accuracy, surpassing both the baseline (65.6%) and QLoRA (96.9%).

Model	Type	# Params	AMC23	AQUA	CMATH	GaoKao 2023 En	Minerva Math	Olympiad/ Bench	SAT Math
Qwen2.5-1.5B-it	**baseline**	-	17.5	49.2	65.2	36.4	9.6	12.0	59.4
	QLoRA	18.46M	15.0	42.5	61.5	29.6	8.1	8.9	59.4
	QOFT	7.89M	**27.5**	**53.1**	**68.5**	**41.0**	**11.8**	**14.4**	**81.2**
Qwen2.5-1.5B	**baseline**	-	0.0	18.9	4.0	4.2	2.6	2.4	28.1
	QLoRA	18.46M	15.0	37.4	**64.2**	26.8	**8.5**	6.8	62.5
	QOFT	7.89M	**22.5**	**53.1**	56.3	**36.1**	**8.5**	**12.7**	**87.5**
Qwen2.5-7B-it	**baseline**	-	50.0	16.5	89.3	61.8	**33.5**	36.6	53.1
	QLoRA	40.37M	30.0	48.0	88.8	50.1	25.4	19.7	68.8
	QOFT	17.55M	**52.5**	**70.9**	**90.5**	**63.6**	**33.5**	**37.6**	**96.9**
Qwen2.5-7B	**baseline**	-	25.0	55.1	61.2	42.9	11.8	29.9	71.9
	QLoRA	40.37M	35.0	48.8	73.7	49.9	18.8	18.5	62.5
	QOFT	17.55M	**52.5**	**59.4**	**80.7**	**55.6**	**21.7**	**34.7**	**87.5**
Qwen2.5-32B-it	**baseline**	-	62.5	18.5	92.5	70.1	**41.5**	44.4	65.6
	QLoRA	134.22M	62.5	71.7	94.0	71.2	39.7	46.8	96.9
	QOFT	57.90M	**75.0**	**83.1**	**94.7**	**73.5**	**41.5**	**48.7**	**100.0**
Qwen2.5-32B	**baseline**	-	35.0	23.2	35.7	46.8	20.2	25.2	62.5
	QLoRA	134.22M	40.0	52.4	90.5	61.0	32.0	29.8	65.6
	QOFT	57.90M	**70.0**	**68.5**	**90.7**	**71.4**	**36.0**	**44.9**	**93.8**

Table 5: The pass@1 performance of the Qwen2.5 series large language models and its QLoRA/QOFT fine-tuned variants by the Chain-of-Thought reasoning.

These gains persist across mathematical reasoning tasks (e.g., 70.0% on AMC23 for QOFT-32B vs. QLoRA's 40.0%), suggesting that orthogonal adaptation in quantized space better preserves the model's reasoning capabilities compared to low-rank adaptation. The results demonstrate QOFT's dual strength: parameter efficiency without sacrificing task performance, particularly in the quantized setting. In contrast, QLoRA fine-tuned models can exhibit training instabilities (Li et al., 2023), leading to instances where their performance fell below baseline methods. Appendix B gives more results on finetuning math-specific Qwen2.5.

7.4 Text-to-image Diffusion Models: SD-3.5

To assay the generality of the proposed methods across modalities, we perform Dreambooth (Ruiz et al., 2023) finetuning on the latest Stable Diffusion 3.5 models (Esser et al., 2024). Dreambooth fine-tunes text-to-image models using a limited set of images depicting the same subject. This process binds the subject to a unique token identifier, enabling subject-driven generation where the model synthesizes this subject in novel scenes beyond the training data. Qualitative results are shown in Figure 5 and Appendix C. We also report the actual peak GPU memory usage during the finetuning process in Appendix C. For finetuning the NormalFloat 4-bit quantized Stable Diffusion 3.5 Large model, QOFT requires slightly less GPU memory (38.68 GiB) than the QLoRA method (38.75 GiB).

8 Concluding Remarks

OFTv2 advances orthogonal finetuning through three key innovations: (i) an input-centric reformulation using matrix–vector products, reducing training time by over 10× and peak memory by 3× without loss in performance; (ii) a Neumann series based approximation of the Cayley transform, improving numerical stability while preserving approximate orthogonality; and (iii) an extension to quantized models, which matches or surpasses QLoRA in speed, stability, and memory efficiency. Across BART, LLaMA2, Qwen2.5, and Stable Diffusion3.5 (0.5B–72B), OFTv2 achieves competitive performance with roughly half the trainable parameters and consistent memory savings.

9 Limitations

OFTv2 substantially improves upon OFT in both memory and computational efficiency, matching low-rank methods in memory usage across data types and training speed in the quantized setting. However, its full-precision fine-tuning remains slower. This limitation arises from fundamental differences: low-rank can be naturally maintained efficiently through two simple linear layers, while preserving orthogonality presents a greater optimization challenge. Additionally, low-rank approaches benefit from extensive community-driven engineering and optimization. Bridging this computational gap presents an interesting research direction.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *ACL*.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. In *NeurIPS*.

Massimo Bini, Karsten Roth, Zeynep Akata, and Anna Khoreva. 2024. Ether: Efficient finetuning of large-scale models with hyperplane reflections. In *ICML*.

Arnav Chavan, Zhuang Liu, Deepak Gupta, Eric Xing, and Zhiqiang Shen. 2023. One-for-all: Generalized lora for parameter-efficient fine-tuning. *arXiv preprint arXiv:2306.07967*.

Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. In *ICLR*.

Ke Chen. 2005. *Matrix preconditioning techniques and applications*. 19. Cambridge University Press.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. 2022. Adaptformer: Adapting vision transformers for scalable visual recognition. In *NeurIPS*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *NeurIPS*.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. 2024. Scaling rectified flow transformers for high-resolution image synthesis. *Preprint*, arXiv:2403.03206.

Hugging Face. 2025. [link].

Jinyuan Feng, Zhiqiang Pu, Tianyi Hu, Dongmin Li, Xiaolin Ai, and Huimu Wang. 2025. Omoe: Diversifying mixture of low-rank adaptation by orthogonal finetuning. *arXiv preprint arXiv:2501.10062*.

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-attention is all you need: Adapting pretrained transformers for machine translation. In *EMNLP*.

Mikhail Gorbunov, Kolya Yudin, Vera Soboleva, Aibek Alanov, Alexey Naumov, and Maxim Rakhuba. 2024. Group and shuffle: Efficient structured orthogonal parametrization. In *NeurIPS*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Preprint*, arXiv:1506.03340.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *ICML*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022a. Lora: Low-rank adaptation of large language models. In *ICLR*.

Edward J. Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022b. LoRA: Low-rank adaptation of large language models. In *ICLR*.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. 2022. Visual prompt tuning. In *ECCV*.

Shibo Jie and Zhi-Hong Deng. 2023. Fact: Factor-tuning for lightweight adaptation on vision transformer. In *AAAI*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. 2023. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Preprint*, arXiv:2305.14152.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *Preprint*, arXiv:1910.13461.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. In *NeurIPS*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL*.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. 2023. Loftq: Lora-fine-tuning-aware quantization for large language models. *Preprint*, arXiv:2310.08659.

Dongze Lian, Daquan Zhou, Jiashi Feng, and Xinchao Wang. 2022. Scaling & shifting your features: A new baseline for efficient model tuning. In *NeurIPS*.

Baohao Liao and Christof Monz. 2024. 3-in-1: 2d rotary adaptation for efficient finetuning, efficient batching and composability. *arXiv preprint arXiv:2409.00119*.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. Mario: Math reasoning with code interpreter output–a reproducible pipeline. *arXiv preprint arXiv:2401.08190*.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for llm compression and acceleration. *Preprint*, arXiv:2306.00978.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Vijay Chandra Lingam, Atula Neerkaje, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Eunsol Choi, Alex Dimakis, Aleksandar Bojchevski, and Sujay Sanghavi. 2024. Svft: Parameter-efficient fine-tuning with singular vectors. In *NeurIPS*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *NeurIPS*.

Weiyang Liu, Rongmei Lin, Zhen Liu, Lixin Liu, Zhiding Yu, Bo Dai, and Le Song. 2018. Learning towards minimum hyperspherical energy. In *NeurIPS*.

Weiyang Liu, Rongmei Lin, Zhen Liu, James M Rehg, Liam Paull, Li Xiong, Le Song, and Adrian Weller. 2021a. Orthogonal over-parameterized training. In *CVPR*.

Weiyang Liu, Rongmei Lin, Zhen Liu, Li Xiong, Bernhard Schölkopf, and Adrian Weller. 2021b. Learning with hyperspherical uniformity. In *AISTATS*.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. 2024. Parameter-efficient orthogonal finetuning via butterfly factorization. In *ICLR*.

Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. 2023. Towards efficient visual adaption via structural re-parameterization. *arXiv preprint arXiv:2302.08106*.

Xinyu Ma, Xu Chu, Zhibang Yang, Yang Lin, Xin Gao, and Junfeng Zhao. 2024. Parameter efficient quasi-orthogonal fine-tuning via givens rotation. In *ICML*.

Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Project-Numina. Ai-mo/aimo-validation-amc · datasets at hugging face.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2023. Controlling text-to-image diffusion by orthogonal finetuning. In *NeurIPS*.

Snehal Raj and Brian Coyle. 2025. Hyper compressed fine-tuning of large foundation models with quantum inspired adapters. *arXiv preprint arXiv:2502.06916*.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. 2023. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *Preprint*, arXiv:2208.12242.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Lst: Ladder side-tuning for parameter and memory efficient transfer learning. In *NeurIPS*.

Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *NeurIPS*.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. 2024. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2022. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2022. Spot: Better frozen model adaptation through soft prompt transfer. In *ACL*.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. In *EMNLP*.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. 2023. Cmath: Can your language model pass chinese elementary school math test? *arXiv preprint arXiv:2306.16636*.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. Mixture-of-subspaces in low-rank adaptation. *arXiv preprint arXiv:2406.11909*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Chenxu Yang, Ruipeng Jia, Naibin Gu, Zheng Lin, Siyuan Chen, Chao Pang, Weichong Yin, Yu Sun, Hua Wu, and Weiping Wang. 2024b. Orthogonal finetuning for direct preference optimization. *arXiv preprint arXiv:2409.14836*.

Shen Yuan, Haotian Liu, and Hongteng Xu. 2024. Bridging the gap between low-rank and orthogonal adaptation via householder reflection adaptation. In *NeurIPS*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *ACL*.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. 2023a. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023c. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. 2024. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113*.

Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. 2022. Neural prompt search. *arXiv preprint arXiv:2206.04673*.

Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. 2023. Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*.

A Experiment Details

This section outlines the specifics of our experimental setup, including the optimizer, code frameworks, computational resources, evaluation methods, and detailed hyperparameters used for each experiment.

Training details. We employed the Adam optimizer (Kingma and Ba, 2015) for all our training runs. The specific hyperparameters used for each experiment are detailed in the tables referenced below. These include learning rates, batch sizes, number of training epochs, and method-specific configurations: the rank r for LoRA-based methods and the block size b for OFTv2/QOFT. If not explicitly specified, the r for LoRA-based methods is 16 and the block size b for OFTv2/QOFT is set as 32. For the Wikitext dataset, hyperparameters are listed in Table 8. For the GSM8K dataset, hyperparameters are listed in Table 9. For the XSum dataset, hyperparameters are listed in Table 6. For the CNN/DailyMail dataset, hyperparameters are listed in Table 7. Since it is known that merging QLoRA adapter weights to its quantized base models leads to performance degradation[1] and distort the real performance, for every experiment, we evaluate the fine-tuned model without merging the trainable parameters, but load them as extra adapter layers.

Code framework. Our method is implemented using the **Hugging Face PEFT**[2] framework, a widely adopted open-source framework providing state-of-the-art parameter-efficient fine-tuning of pre-trained large language models and diffusion models. The implementation of OFTv2 will be released on Hugging Face PEFT soon, to allow for easy reproduction of our training results. We utilized the Hugging Face TRL library for supervised fine-tuning[3]. For the base model quantization, we leveraged bitsandbytes[4] for the NormalFloat 4-bit quantization and the QLoRA finetuning, and AutoAWQ[5] for AWQ quantization.

Pretrained models. Our work utilized several pre-trained large language models. Specifically, we employed models from the Qwen2.5 model series[6], which are available under the permissive **Apache 2.0 license**. We also leveraged the Llama 2 models[7], governed by the **Llama 2 license**. Additionally, for the text summarization tasks, the BART-large model was used, which is also distributed under the **Apache 2.0 license**. For the text-to-image generation, we utilized the Stable Diffusion 3.5 models, which are under the **Stability AI Community license**. We have adhered to all respective licensing agreements for these models throughout our work.

Dataset. The experiments in this study utilized a diverse range of publicly available datasets to ensure comprehensive evaluation. For finetuning language modeling tasks, we employed the Wikitext-2[8] dataset, which is distributed under the **CC-BY-SA-3.0 license**. Text summarization performance was assessed by fine-tuning on the CNN / DailyMail Dataset[9], also licensed under **Apache 2.0**, and the XSum dataset[10], which is available under the **MIT license**. For finetuning mathematical reasoning capabilities, we used the GSM8K[11] dataset, available under the **MIT license**, and the OpenR1-Math-220k[12] dataset, which can be used under the **Apache 2.0 license**. The Dreambooth dataset[13] for fine-tuning the diffusion models are under the **cc-by-4.0 license**.

Compute Resources. All the training tasks are performed on a **NVIDIA HGX H100 8-GPU System** node with 80GB memory each. We used a single **NVIDIA H100 NVL** GPU with 94GB memory to benchmark the memory usage.

B Mathematical reasoning

Training details. We fine-tuned the Qwen2.5 models using QLoRA or QOFT on a random subset of 50,000 samples from the Huggingface OpenR1-

[1] See this article comparing different merging methods: https://kaitchup.substack.com/p/lora-adapters-when-a-naive-merge

[2] https://huggingface.co/docs/peft/en/index

[3] https://github.com/huggingface/trl

[4] https://github.com/bitsandbytes-foundation/bitsandbytes

[5] https://github.com/casper-hansen/AutoAWQ

[6] https://huggingface.co/collections/Qwen/qwen25-66e81a666513e518adb90d9e

[7] https://huggingface.co/collections/meta-llama/metas-llama2-models-675bfd70e574a62dd0e40541

[8] https://huggingface.co/datasets/Salesforce/wikitext

[9] https://huggingface.co/datasets/abisee/cnn_dailymail

[10] https://huggingface.co/datasets/EdinburghNLP/xsum

[11] https://huggingface.co/datasets/openai/gsm8k

[12] https://huggingface.co/datasets/open-r1/OpenR1-Math-220k

[13] https://huggingface.co/datasets/google/dreambooth

Hyperparameter	LoRA						OFTv2					
	BF16			NF4			BF16			NF4		
	$r=8$	$r=16$	$r=32$	$r=8$	$r=16$	$r=32$	$b=16$	$b=32$	$b=64$	$b=16$	$b=32$	$b=64$
Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	4e-4	4e-4	4e-4	4e-4	4e-4	4e-4
Epoch	10	10	10	10	10	10	5	5	5	5	5	5
Batch size	32	32	32	32	32	32	32	32	32	32	32	32
Gradient Accumulation	4	4	4	4	4	4	4	4	4	4	4	4

Table 6: Hyper-parameter setup of fine-tuning BART-large on XSum with LoRA and OFTv2.

Hyperparameter	LoRA						OFTv2					
	BF16			NF4			BF16			NF4		
	$r=8$	$r=16$	$r=32$	$r=8$	$r=16$	$r=32$	$b=16$	$b=32$	$b=64$	$b=16$	$b=32$	$b=64$
Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4	4e-4	4e-4	4e-4	4e-4	4e-4	4e-4
Epoch	5	5	5	5	5	5	5	5	5	5	5	5
Batch size	64	64	64	64	64	64	64	64	64	64	64	64
Gradient Accumulation	4	4	4	4	4	4	4	4	4	4	4	4

Table 7: Hyper-parameter setup of fine-tuning BART-large on CNN/DailyMail with LoRA and OFTv2.

Hyperparameter	LoRA				OFTv2			
	BF16		NF4		BF16		NF4	
	7B	13B	7B	13B	7B	13B	7B	13B
Learning rate	2e-4	2e-4	2e-4	2e-4	2e-4	2e-4	2e-4	2e-4
Epoch	10	10	10	10	10	10	10	10
Batch size	16	16	16	16	16	16	16	16
Gradient Accumulation	2	2	2	2	2	2	2	2

Table 8: Hyper-parameter setup of fine-tuning Llama 2 on Wikitext-2 with LoRA and OFTv2.

Hyperparameter	LoRA				OFTv2			
	BF16		NF4		BF16		NF4	
	7B	13B	7B	13B	7B	13B	7B	13B
Learning rate	2e-4	2e-4	2e-4	2e-4	8e-4	8e-4	8e-4	8e-4
Epoch	10	10	10	10	10	10	10	10
Batch size	16	16	16	16	16	16	16	16
Gradient Accumulation	4	4	4	4	4	4	4	4

Table 9: Hyper-parameter setup of fine-tuning Llama 2 on GSM8K with LoRA and OFTv2.

```
<|im_start|>system\n
Please reason step by step, and put your final answer within \\boxed{{}}.
<|im_end|>\n
<|im_start|>user\n{input}<|im_end|>\n
<|im_start|>assistant\n{output}\n\n
```

Figure 6: Prompt template used for evaluating Qwen2.5 series models on mathematical reasoning benchmarks.

Math-220k dataset (Face, 2025). For each method and benchmark, we selected the best-performing

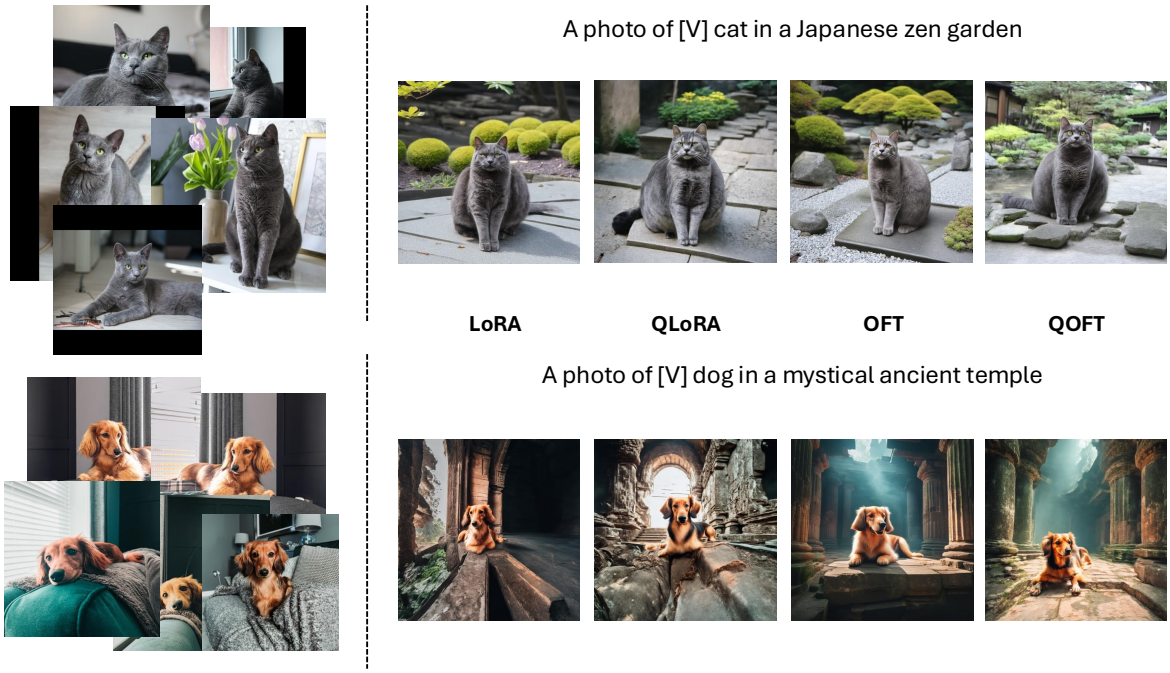


Figure 7: Qualitative results from Dreambooth fine-tuning of Stable Diffusion 3.5 Medium (8.1B parameters), with peak allocated GPU memory: LoRA (**38.00 GB**), OFT (**38.02 GB**), QLoRA (**35.03 GB**) and QOFT (**35.02 GB**).

model after trying learning rates of 1×10^{-5}, 2×10^{-5}, 5×10^{-5}, and 1×10^{-4}. We used a batch size of 16 for the 1.5B models and 8 for the 7B and 32B models, with 2 gradient accumulation steps for all. A cosine learning rate scheduler was employed, with a minimum learning rate set to 10% of the initial value.

Evaluation details. For evaluating the Qwen2.5 base models and the QLoRA or QOFT fine-tuned versions, we utilized the same evaluation pipeline as Qwen2.5-Math[14]. This framework provides robust tools for parsing and evaluating mathematical expressions and problem-solving steps, ensuring accurate and consistent assessment of model performance on these mathematical benchmarks. More specifically, we report the model's pass@1 performance, *i.e.*, the performance on the first attempt for a given task, obtained by utilizing the Qwen2.5 Chain-of-Though question prompt (Figure 6).

C Stable diffusion 3.5

Here we provide additional qualitative results of fine-tuning the Stable Diffusion 3.5 Medium model in Figure 7.

The actual GPU memory usage during LoRA and OFTv2 fine-tuning is summarized in Table 11. As shown, OFTv2/QOFT demonstrates memory efficiency similar to LoRA and QLoRA, regardless of data precision or model scale.

[14] https://github.com/QwenLM/Qwen2.5-Math

Model	Type	# Params	AMC23	AQUA	CMATH	GaoKao 2023 En	Minerva Math	Olympiad/ Bench	SAT Math
Qwen2.5-1.5B-math-it	**QLoRA**	18.46M	27.5	33.5	86.8	43.6	15.4	15.1	46.9
	QOFT	7.89M	**45.0**	**70.9**	**87.2**	**60.5**	**25.4**	**32.0**	**93.8**
Qwen2.5-1.5B-math	**QLoRA**	18.46M	25.0	31.5	49.0	36.9	10.7	12.9	**50.0**
	QOFT	7.89M	**27.5**	**31.5**	**55.5**	**37.7**	**13.6**	**14.4**	37.5
Qwen2.5-7B-math-it	**QLoRA**	40.37M	32.5	34.6	89.8	47.0	18.8	18.2	53.1
	QOFT	17.55M	**52.5**	**76.8**	**92.7**	**66.8**	**35.7**	**41.6**	**93.8**
Qwen2.5-7B-math	**QLoRA**	40.37M	**30.0**	38.6	75.7	48.6	21.0	**20.4**	**50.0**
	QOFT	17.55M	**30.0**	**40.6**	**81.7**	**49.4**	**21.3**	**20.4**	**50.0**

Table 10: The pass@1 performance of the Qwen2.5 series math-specific large language fine-tuned with QLoRA/QOFT by the Chain-of-Thought reasoning.

	SD 3.5 Medium	**SD 3.5 Large**
LoRA	38.00 GB	52.33 GB
OFTv2	38.02 GB	52.32 GB
QLoRA	35.03 GB	41.60 GB
QOFT	35.02 GB	41.53 GB

Table 11: Actual GPU memory usage during fine-tuning: LoRA, QLoRA, OFTv2, and QOFT applied on Stable Diffusion 3.5 Medium and Large.