TensorGRaD: Tensor Gradient Robust Decomposition for Memory-Efficient Neural Operator Training

Anonymous authors
Paper under double-blind review

Abstract

Scientific problems require resolving multi-scale phenomena across different resolutions and learning solution operators in infinite-dimensional function spaces. Neural operators provide a powerful framework for this, using tensor-parameterized layers to capture complex, multi-dimensional relationships. However, scaling neural operators to high-resolution problems leads to significant computational demands, making the training of industrial-scale models prohibitive. In this work, we introduce **TensorGRaD**, a novel method that directly addresses the memory challenges associated with optimizing large tensor-structured weights. Our approach, based on a *robust tensor decomposition*, factorizes gradients as the sum of a low-rank tensor and a sparse one to efficiently capture information within optimizer states, including outliers. Additionally, we provide a recipe for mixed precision training of TensorGRaD, achieving further memory savings without sacrificing accuracy. We showcase the effectiveness of TensorGRaD for solving partial differential equations (PDEs) using Fourier Neural Operators. We provide theoretical guarantees for TensorGRaD demonstrating its fundamental advantage over matrix-based gradient compression methods. We empirically demonstrate large improvements across various PDE tasks, including the challenging turbulent Navier-Stokes case at a Reynolds number of 10^5. TensorGRaD reduces total memory usage by over 50% while maintaining and sometimes even improving accuracy.

1 Introduction

Modern deep learning has shifted towards large-scale foundation models, which have enabled unprecedented performance across diverse domains such as natural language processing, computer vision, and scientific computing (Brown et al., 2020; Kirillov et al., 2023). This represents a paradigm shift from traditional machine learning, where performance improvements are driven by scaling laws—requiring increases in data, compute, and model size (Xiao, 2025). This scaling comes at the cost of growing memory requirements. Adaptive optimizers such as Adam (Kingma and Ba, 2014), while crucial in training these large models, worsen this issue by storing additional moment tensors (e.g., first and second order moments for Adam) for each weight, which significantly increases the memory overhead (Zhao et al., 2024; Loeschcke et al., 2024a)

This memory requirement is exacerbated in the case of scientific computing, both by the size and the nature of the data and models involved. Solving scientific problems typically involves solving partial differential equations (PDEs) and resolving multi-scale phenomena, on very large-scale data (Azizzadenesheli et al., 2024). This multi-dimensional data is naturally represented using tensors: multidimensional arrays that offer a natural framework for representing and manipulating complex, high-dimensional data structures (Kolda and Bader, 2009). For instance, in weather forecasting, data can span spatial grids, time steps, and atmospheric variables, leading to high-order tensor representations (Bonev et al., 2023).

Neural operators have been proposed as the natural framework to tackle these problems, generalizing deep learning from learning in finite-dimensional spaces to learning in function spaces (Li et al., 2023). Unlike neural networks, neural operators learn a mapping between function spaces, making them naturally suited for capturing the multi-scale structure of scientific data. To capture these multi-scale relationships, Neural Operators leverage the inherent (multi-dimensional) structure in the data, which requires maintaining high-

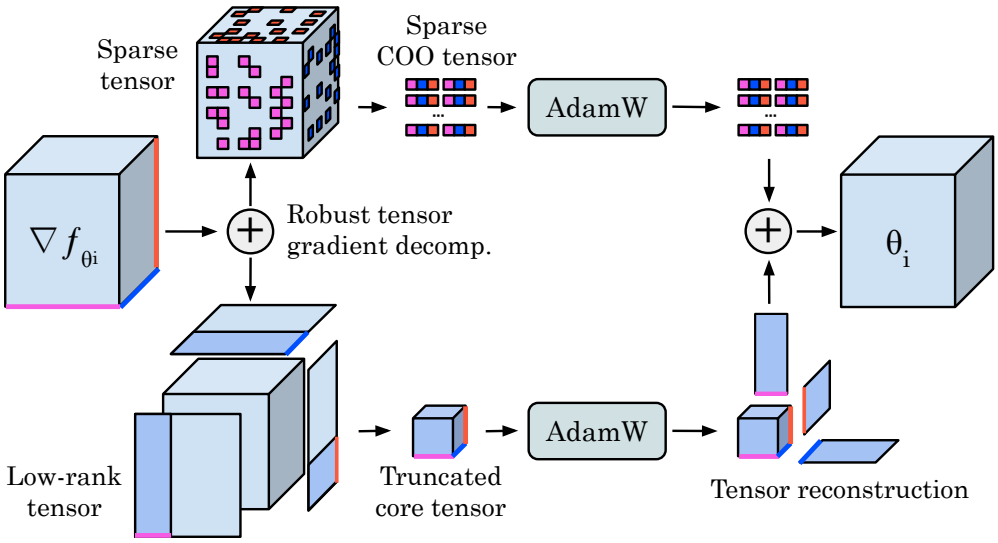

Figure 1: **Overview of TensorGRaD**. Low-rank plus sparse decomposition

order tensor weights and gradients to capture complex spatial, temporal, and channel interactions. As a result, unlike typical models in natural language processing or computer vision, where memory is dominated by activations, the memory overhead in neural operators is primarily driven by the tensor-structured weights and gradients. This memory requirement grows intractably with the scale of the data and has hindered the scaling of neural operators to large and complex scientific problems.

While many recent methods reduce optimizer memory in large language models, they are not directly applicable to neural operators due to the multi-dimensional structure of their weights and gradients. GaLore (Gradient Low-Rank Projection) (Zhao et al., 2024), uses a Singular Value Decomposition (SVD) to compute low-rank approximations of gradient matrices before computing optimizer states, significantly reducing memory usage during training. Extending GaLore to neural operators requires flattening gradient tensors into matrices, which disrupts their multi-dimensional structure, discarding important relationships between modes (e.g., spatial, temporal, or channel interactions). The resulting flattened matrix is not naturally low-rank and hence, standard GaLore has poor performance when low-rank structures are enforced.

This not only leads to a suboptimal low-rank approximation but can also degrade model performance, especially in scientific applications where these interactions are crucial. Alternatives to low-rank gradient projections include GRASS (Muhamed et al., 2024), which uses structured sparse projections that match low-rank methods at higher memory budgets but underperform under strict constraints. Generally, existing methods primarily focus on either low-rank or sparse representations, rarely exploring their combined application to tensor gradients, where multi-dimensional structure is critical.

In this work, we propose **TensorGRaD**, a novel method for efficient training of neural operators that directly addresses the memory challenges associated with tensor-structured gradients. Our approach hinges on a *robust tensor decomposition (RTD)* (Gu et al., 2014) of the gradients during optimization. Specifically, we generalize both low-rank and sparse projections to tensors and unify them in a *robust tensor decomposition* framework. Unlike prior approaches that apply low-rank or sparse factorization to weights or flattened gradients, TensorGRaD operates directly on high-order tensor gradients and maintains optimizer states in decomposed form. We prove analytically that a direct extension of GaLore, relying on matricizing the gradient tensors, fails to preserve the multilinear structure required by Neural Operators. We also verify this empirically in ablation studies. Finally, we show that pure low-rank tensor compression fails to capture the heavy-tailed nature of PDE gradients, and that the sparse residual helps preserve outliers and improve learning under strict memory budgets.

Our robust tensor decomposition framework is able to accurately compress gradients $\mathcal{G}$ by decomposing them as a sum $\mathcal{G} = \mathcal{L} + \mathcal{S}$ of a low rank tensor approximation $\mathcal{L}$ with a sparse part $\mathcal{S}$. We demonstrate that our robust tensor gradient factorization remains stable under a mixed-precision strategy, running activations,

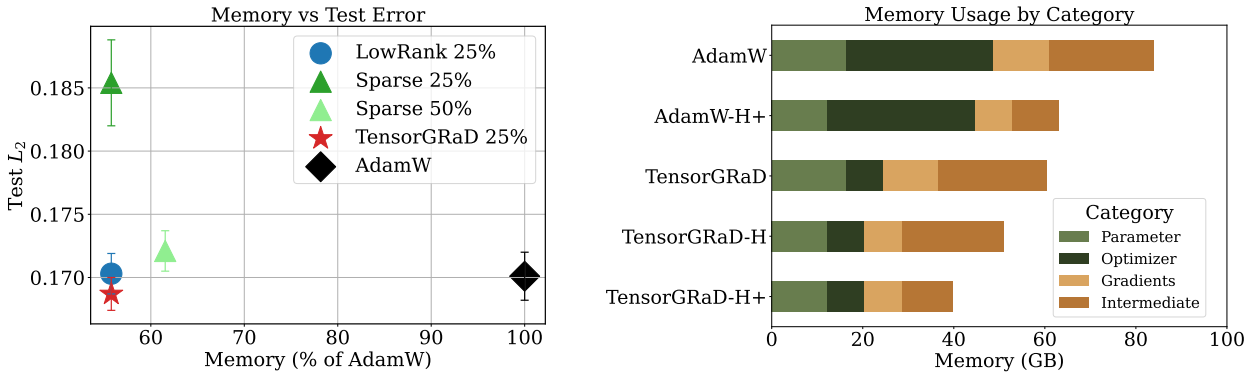

Figure 2: **Memory and performance.** Left: Comparison of low-rank, structured sparse, and TensorGRaD (mixed precision) vs. Adam on Navier–Stokes 1024×1024. TensorGRaD offers the best memory–accuracy trade-off. Right: Peak CUDA memory for FNO models with 256 channels. **H**: uses half precision for weights/gradients and full for optimizer states, and **+** includes activation checkpointing.

weights, and gradients in half precision while maintaining optimizer states in full precision. This setup achieves substantial memory savings without compromising model accuracy. Empirically, we show that using half-precision optimizer states degrades performance, underscoring the importance of our method for preserving gradient information.

Implemented with AdamW, TensorGRaD reduces memory usage by up to 75% for high-resolution neural operator learning, while matching or improving baseline accuracy across several PDE benchmarks. On the challenging Navier–Stokes at 1024×1024 resolution with a Reynolds number of 10^5, where turbulent structures emerge across multiple scales, our mixed precision TensorGRaD matches the test L_2 loss of the full-precision Adam optimizer, while reducing optimizer memory usage by up to 75% and cutting and total memory cost of more than 55% while matching or even improving on baseline performance. Code will be made open-source and publicly available.

2 TensorGRaD

In this section, we first introduce the necessary background before going into detail in our method, illustrated in Fig. 1, its training, implementation, and theoretical properties.

2.1 Background: Tensors and Neural Operators

Tensors are multidimensional arrays (higher-order generalizations of vectors and matrices). An Nth-order tensor $\mathcal{X} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ has mode sizes $\{I_n\}_{n=1}^{N}$. **Neural Operators** learn mappings between function spaces and are often instantiated with Fourier Neural Operators (FNOs) (Li et al., 2020) (Appendix A). FNO layers combine Fourier-mode and pointwise mixing, producing tensor-structured weights and gradients whose optimizer states dominate memory at high resolution.

2.2 TensorGRaD

Robust decomposition of gradients. Our method hinges on a robust decomposition of the gradient during training. Instead of minimizing a reconstruction error under some Gaussian error assumption, robust decomposition decomposes inputs as a low-rank part and a sparse part, even when it contains gross corruption or outliers (Candès et al., 2009). Robust Tensor Decomposition (RTD) (Gu et al., 2014) extends this concept to higher-order tensors, proving that low-rank components can be separated from sparse corruptions using convex optimization. Later work (Zhang et al., 2018) provided convergence guarantees for low-rank plus sparse recovery in matrices.

Leveraging these advances in robust decomposition, we transform the gradients into compressed representations before computing and storing their optimizer moments. Specifically, we use two complementary forms of structure in gradient tensors: unstructured sparsity and low-rank decompositions. Each form compresses

a distinct aspect of the gradient: sparse representations preserve sharp, localized signals, while low-rank approximations model smooth, global structure. These transformations are applied sequentially and combined to form a *robust tensor decomposition*.

Unstructured sparse gradient tensor. We represent localized gradient information using a sparse COO-format tensor $\hat{\mathcal{G}}_S$ supported on a fixed set of indices $\Omega \subseteq [I_1] \times \cdots \times [I_N]$. This index set is constructed by selecting $k = \lceil \rho I \rceil$ entries from $\mathcal{G} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$ according to a sparsification strategy, e.g., by inspiration from GRASS (Muhamed et al., 2024) top-k magnitude, probabilistic sampling, or uniform random selection. The corresponding values are extracted to define $\hat{\mathcal{G}}_S = \text{Sparse}(\mathcal{G}, \Omega)$, where $\hat{\mathcal{G}}_S$ is a k-nonzero sparse tensor in COO format, consisting of index–value pairs. The sparse index set Ω is recomputed only every T steps and reused in between, while the sparse tensor $\hat{\mathcal{G}}_S$ is extracted from the current gradient at every step. When the sparse support changes, newly introduced indices receive zero-initialized Adam moments (m, v), ensuring consistent optimizer behavior while preventing leakage of stale state across supports. Existing indices retain their accumulated moments until they are removed.

This format is compatible with standard sparse tensor operations, enabling direct addition, scaling, and indexing without reconstructing a dense tensor. Overall, this representation requires storing k integer indices and k complex values. It supports efficient computation in the sparse format, such as gather and scatter operations, with no dense intermediates.

Low-rank gradient tensor decomposition. To compress high-dimensional gradient tensors, we use a Tucker decomposition (Tucker, 1966; Kolda and Bader, 2009), a higher-order generalization of low-rank matrix factorization. Given a tensor $\mathcal{G} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$, we approximate it as $\mathcal{G} \approx [\![\mathcal{C}; U^{(1)}, \ldots, U^{(N)}]\!]$, where $\mathcal{C}$ is a core tensor of size $\mathbb{C}^{r_1 \times \cdots \times r_N}$ and $U^{(n)} \in \mathbb{C}^{I_n \times r_n}$ are orthonormal factor matrices. We compute the decomposition once and discard the core, retaining only the factor matrices. These are then reused to compress incoming gradients into a factorized representation: $\hat{\mathcal{G}}_L = \mathcal{G} \times_1 {U^{(1)}}^\top \cdots \times_N {U^{(N)}}^\top$. Optimizer states are maintained directly on $\hat{\mathcal{G}}_L$, and the transformed tensor is reconstructed after the update via: $\tilde{\mathcal{G}}_L = \hat{\mathcal{G}}_L \times_1 U^{(1)} \cdots \times_N U^{(N)}$.

This decomposition reduces memory by maintaining only the factor matrices (each of size $\mathbb{C}^{I_n \times r_n}$) and the compressed optimizer state. It offers three key properties central to our method:

- **SVD generalization:** In the special case of $N = 2$, the Tucker decomposition reduces to the standard matrix SVD, linking our method naturally to GaLore.
- **Orthonormality and efficiency:** The factor matrices $U^{(n)}$ are orthonormal, allowing stable compression via mode-wise multiplication with $U^{(n)\top}$, and reconstruction using $U^{(n)}$ directly without requiring matrix inversion.
- **Structure preservation:** Tucker factorization maintains mode-wise information, avoiding the loss of semantic structure associated with tensor flattening and Kronecker approximations.

Residual and composition. The two components are applied sequentially. After forming the sparse or low-rank approximation $\tilde{\mathcal{G}}_1$, we compute the residual $\mathcal{R} = \mathcal{G} - \tilde{\mathcal{G}}_1$. We then use $\mathcal{R}$ to compute $\tilde{\mathcal{G}}_2$ instead of $\mathcal{G}$, i.e., after computing the residual either the low-rank decomposition or the sparse tensor is computed on $\mathcal{R}$. This composition allows each branch to focus on distinct parts of the tensors. Using sparse top-k removal first can remove high-frequency components or outliers. Alternatively, using low-rank first can capture the smoothness, followed by a sparse projector that can capture the most significant part not converted by the low-rank tensors.

Optimizer update. Each component is updated independently using Adam in its compressed space. First and second moment estimates $(\mathcal{M}_S, \mathcal{V}_S)$ and $(\mathcal{M}_L, \mathcal{V}_L)$ are maintained for the sparse and low-rank parts, respectively. The full update is reconstructed as:

$$\Delta\mathcal{W} = \alpha \left(\tilde{\mathcal{G}}_L + \lambda \, \tilde{\mathcal{G}}_S \right), \qquad \mathcal{W}_{t+1} = \mathcal{W}_t + \eta \cdot \Delta\mathcal{W}.$$

The order of the decompositions matters only during the forward pass, as the first component defines the residual for the second. For memory efficiency, the low-rank component is reconstructed first, and the sparse

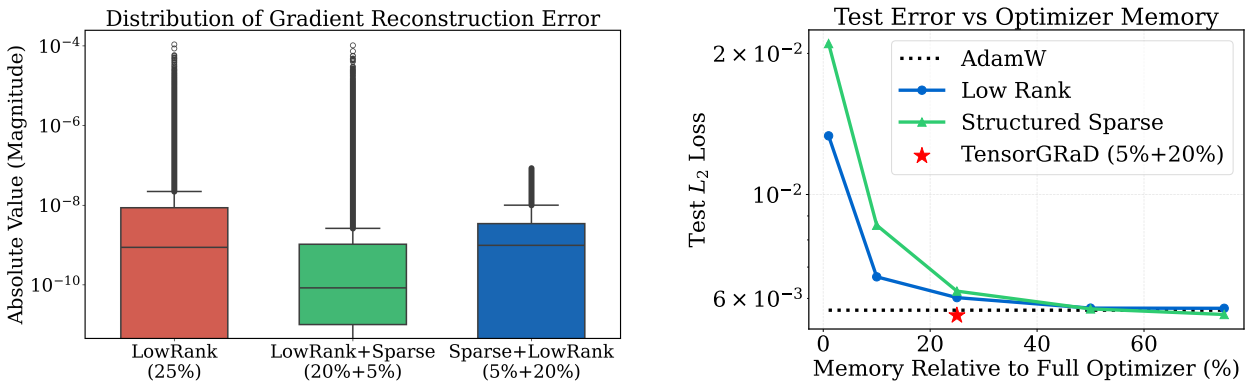

Figure 3: **Left: Gradient reconstruction error.** Box plot of gradient reconstruction error for different compression strategies on a complex FNO layer. TensorGRaD variants reduce high-magnitude outliers. **Right: Accuracy–memory trade-off.** Performance of low-rank and structured sparse across compression ratio vs. TensorGRaD at 25% and Adam baselines. TensorGRaD achieves the best trade-off at.

values are added directly into the same tensor via scatter operations. This way, we avoid having two full tensors in memory at once. This robust decomposition reduces memory overhead by maintaining compact optimizer states in compressed formats. Unstructured sparsity captures fine-grained outliers, while mode-wise low-rank decompositions preserve global structure. By combining them, TensorGRaD achieves higher fidelity under strong memory constraints than either approach alone. We present the pseudocode in 1.

Fig. 3 compares the distribution of reconstruction errors for a complex gradient tensor ($64^3 \times 32$) from an FNO layer measured on a single training step of the Navier–Stokes 128×128 dataset. We evaluate three strategies: (1) unstructured sparse followed by low-rank compression (5% + 20%), (2) low-rank followed by sparse (20%+5%), and (3) pure low-rank compression at 25%. The reconstruction error is measured as the absolute difference between the original gradient $\mathcal{G}$ and its approximation $\tilde{\mathcal{G}}$, i.e., $|\mathcal{G} - \tilde{\mathcal{G}}|$. All layers exhibit similar trends. The pure low-rank method introduces more high-magnitude outliers, while the combined strategies produce tighter distributions. Applying sparse compression first yields fewer large errors but slightly higher average reconstruction error, suggesting a trade-off between error spread and overall magnitude.

2.3 Mixed precision training

We show that our gradient compression method remains stable when activations, weights, and gradients are computed in half precision, provided that optimizer states are maintained in full precision. This extends recent work on FNO training by Tu et al. (Tu et al., 2024), which established approximation guarantees for mixed-precision FNO training using AMP. However, their approach retains weights in full precision and casts them to half precision during computation. In contrast, we explore full half-precision training beyond AMP, including weight storage. Additionally, we find that storing optimizer states in half precision significantly degrades performance, further emphasizing projected optimizer states as an alternative. Combining mixed-precision training with TensorGRaD allows further memory savings without sacrificing performance.

2.4 Implementation

The **low-rank component** of TensorGRaD uses the efficient Tucker decomposition from TensorLy (Kossaifi et al., 2019), implemented via Higher-Order Orthogonal Iteration. See Appendix E.1 for more details. All subsequent operations, like compressing the gradients and reconstructing the low-rank updates, are performed using PyTorch. The **unstructured sparse component** is implemented natively in PyTorch. We extract the top-k or randomly sampled values based on a given sparsification strategy and store them as index–value pairs. This format supports direct operations like elementwise scaling and addition without dense reconstruction. Together with gradient compression and mixed precision training, these techniques allow **TensorGRaD** to scale efficiently to large neural operators. We refer to this full setup as **TensorGRaD+**, and highlight its most memory-efficient in Fig. 2.

Algorithm 1 TensorGRaD: Adam with Unstructured Sparse + Tucker Low-Rank Gradient Compression

Require: Weight tensor $\mathcal{W} \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4}$. Step size η, scale factor α, decay rates β_1, β_2, sparsity ratio $\rho \in (0,1)$, sparse scale λ, Tucker ranks (r_1, r_2, r_3, r_4) (or uniform r), projector update gap T.

$t \leftarrow 0$, $I \leftarrow N_1 N_2 N_3 N_4$, $k \leftarrow \lceil \rho I \rceil$
Initialize low-rank factors $U^{(n)} \in \mathbb{C}^{N_n \times r_n}$ (e.g., random orthonormal), for $n = 1, \ldots, 4$
Initialize low-rank Adam moments $\mathcal{M}_L, \mathcal{V}_L \in \mathbb{C}^{r_1 \times r_2 \times r_3 \times r_4} \leftarrow 0$
Initialize sparse index set $\Omega \leftarrow \emptyset$ and sparse Adam moments $(m_\Omega, v_\Omega) \leftarrow (0, 0)$ ▷ $m_\Omega, v_\Omega \in \mathbb{C}^{|\Omega|}$
repeat
  $\mathcal{G}_t \leftarrow -\nabla_{\mathcal{W}} \phi_t(\mathcal{W}_t)$
  if $t \bmod T = 0$ **then**
    $\Omega \leftarrow \text{SparseIndices}(\mathcal{G}_t, k, \text{strategy})$ ▷ e.g., Top-k, Rand-k, Prob-k
    $(m_\Omega, v_\Omega) \leftarrow \text{ReindexMoments}(m_\Omega, v_\Omega, \Omega)$
    $g_\Omega \leftarrow \text{Gather}(\mathcal{G}_t, \Omega)$
    $\mathcal{R}_L \leftarrow \mathcal{G}_t - \text{Scatter}(\Omega, g_\Omega)$ ▷ dense residual
    $\{U^{(n)}\}_{n=1}^4 \leftarrow \text{TuckerFactors}(\mathcal{R}_L, (r_1, r_2, r_3, r_4))$ ▷ HOI/HOSVD update
  end if
  $g_\Omega \leftarrow \text{Gather}(\mathcal{G}_t, \Omega)$ ▷ unstructured sparse values (COO)
  $\mathcal{G}_{t,\text{res}} \leftarrow \mathcal{G}_t - \text{Scatter}(\Omega, g_\Omega)$ ▷ subtract sparse part (dense)
  $\hat{\mathcal{G}}_L \leftarrow \mathcal{G}_{t,\text{res}} \times_1 U^{(1)^\top} \times_2 U^{(2)^\top} \times_3 U^{(3)^\top} \times_4 U^{(4)^\top}$ ▷ Tucker compression
  $(m_\Omega, v_\Omega, g_\Omega) \leftarrow \text{AdamUpdateSparse}(m_\Omega, v_\Omega, g_\Omega, \beta_1, \beta_2, t)$
  $\hat{\mathcal{G}}_L \leftarrow \text{AdamUpdateDense}(\hat{\mathcal{G}}_L, \mathcal{M}_L, \mathcal{V}_L, \beta_1, \beta_2, t)$
  $\tilde{\mathcal{G}} \leftarrow \alpha \cdot (\hat{\mathcal{G}}_L \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)})$ ▷ reconstruct low-rank part
  $\tilde{\mathcal{G}} \leftarrow \text{ScatterAdd}(\tilde{\mathcal{G}}, \Omega, \lambda g_\Omega)$ ▷ add sparse correction
  $\mathcal{W}_{t+1} \leftarrow \mathcal{W}_t + \eta \cdot \tilde{\mathcal{G}}$
  $t \leftarrow t + 1$
until convergence

2.5 Theoretical Results of TensorGRaD

Our theory explains why tensor-mode projections better preserve gradient structure than matricization. We extend the fixed-subspace analysis of GaLore (Zhao et al., 2024) to tensor gradients and show these projections yield a contraction under standard smoothness and projected-curvature assumptions.

Our analysis considers only the mode-wise low-rank part of TensorGRaD with fixed projections $P_k k = 1^d$; it does not cover the full AdamW update, the sparse residual, or time-varying projections. The purpose is to formalize why preserving multilinear structure avoids the distortion from matricization, rather than to provide a complete convergence theory for the full hybrid method. Proofs and background are in Appendix M, N, and O. We denote the additive gradient term by $\mathcal{A}i$, mode-wise linear operators by $Bit^{(k)} k = 1^d$, the induced tensor projection by P, and the projected mode-k curvature by $\kappa_t^{(k)}$.

Theorem 1 (TensorGRaD convergence with fixed mode-wise projections) Let $\{P_k \in \mathbb{R}^{I_k \times r_k}\}_{k=1}^d$ have orthonormal columns ($P_k^\top P_k = I$). Define the tensor projection and reconstruction

$$\mathsf{P}(\mathcal{G}) := \mathcal{G} \times_1 P_1^\top \times_2 \cdots \times_d P_d^\top, \qquad \mathsf{P}^\top(\mathcal{R}) := \mathcal{R} \times_1 P_1 \times_2 \cdots \times_d P_d,$$

and set $\mathcal{R}_t := \mathsf{P}(\mathcal{G}_t)$ and $\tilde{\mathcal{G}}_t := \mathsf{P}^\top(\mathcal{R}_t)$. Consider the projected update (with $\rho_t \equiv 1$) $\mathcal{W}_t = \mathcal{W}_{t-1} + \eta \tilde{\mathcal{G}}_{t-1}$. Assume the gradient admits the multilinear form $\mathcal{G}_t = \frac{1}{N} \sum_{i=1}^N \Big(\mathcal{A}_i(\mathcal{W}_t) - \mathcal{W}_t \times_1 B_{it}^{(1)} \times_2 \cdots \times_d B_{it}^{(d)} \Big)$, where for each mode k, $B_{it}^{(k)} \in \mathbb{R}^{I_k \times I_k}$ is a square matrix acting on mode k. Fix a mode $k \in \{1, \ldots, d\}$. Suppose: (i) $\mathcal{A}_i$ is mode-k Lipschitz with constant $L_A^{(k)}$, and $B_{it}^{(k)}$ is mode-k Lipschitz with constant $L_B^{(k)}$; (ii) $\|\mathcal{W}_t\|_{(k)} \le D_k$; and (iii) the projected mode-k matrices $\hat{B}_{it}^{(k)} := P_k^\top B_{it}^{(k)} P_k$ are PSD and define the projected curvature lower bound $\kappa_t^{(k)} := \frac{1}{N} \sum_{i=1}^N \lambda_{\min}\big(\hat{B}_{it}^{(k)}\big)$. Then, for η small enough so that the bracket below lies in $(0,1)$, TensorGRaD satisfies the mode-k contraction

$$\|(\mathcal{R}_t)_{(k)}\|_F \;\le\; \Big[1 - \eta\big(\kappa_{t-1}^{(k)} - L_A^{(k)} - L_B^{(k)} D_k^2\big)\Big] \, \|(\mathcal{R}_{t-1})_{(k)}\|_F.$$

In particular, if $\inf_{t,k} \kappa_t^{(k)} > L_A^{(k)} + L_B^{(k)} D_k^2$, then $(\mathcal{R}_t)_{(k)} \to 0$ for all k, hence $\mathcal{R}_t \to 0$ under the fixed projections $\{P_k\}_{k=1}^d$. The proof is given in Appendix O.

Remark 1 (Mode-k continuity) The mode-k continuity assumptions on $\mathcal{A}_i$ and $B_{it}^{(k)}$ are standard smoothness conditions for neural network components and are satisfied by the linear and pointwise operations used in FNO blocks under bounded-weight iterates.

3 Experimental Setup and Results

We conduct a comprehensive evaluation of TensorGRaD on a diverse set of benchmark datasets for NOs, representing a range of PDEs with varying complexity and dimensionality.

Datasets. We report results on several PDE datasets: **Burgers Equation:** A one-dimensional nonlinear PDE with viscosity modeling fluid dynamics, trained on 1000 samples of Gaussian random fields at 128-point resolution. **Darcy Flow:** An elliptic PDE describing fluid flow through porous media with variable coefficients, trained on 4000 samples discretized on a 421×421 grid. **Electromagnetic Wave Propagation:** A complex-valued nonlinear Schrödinger equation modeling optical pulse propagation in waveguides with second-harmonic generation, trained on 800 samples with varying physical parameters. **Navier-Stokes:** We study the 2D Kolmogorov flow, a variant of the incompressible Navier–Stokes equations with periodic forcing (Wang et al., 2024). This dataset is particularly challenging and has a Reynolds number of $Re \approx 2 \times 10^5$, representing a highly turbulent regime. Full dataset specifications are provided in Appendix B.

Model Architecture, Training, and Evaluation. All models are based on the Fourier Neural Operator (FNO) architecture and trained using the Adam optimizer. Training details, including learning rates, batch sizes, and loss functions, are provided in Appendix 15. For memory profiling methodology, see Appendix H. Code will be provided in the supplementary.

In evaluating TensorGRaD, we vary the total compression ratio by adjusting the rank of the low-rank decomposition and the density of the sparse tensor to assess the trade-off between memory efficiency and performance. To further reduce memory usage, we apply **activation checkpointing** (Chen et al., 2016), which recomputes intermediate activations during backpropagation. All models are implemented in PyTorch and trained on NVIDIA A100, H100, and H200 GPUs. The main paper focuses on the most challenging Navier–Stokes at high resolution dataset and ablations of different sparse–low-rank combinations for TensorGRaD. We present additional experiments in the appendix, including comparisons to the direct mode-unfolding extension of GaLore. We report performance using the L_2 test loss, and quantify improvements wrt. baseline performance via relative gain percentage.

Table 1: **Memory and accuracy comparison on Navier–Stokes** 1024×1024 with Reynolds number 10^5. Train and test losses are $L_2 \times 10^{-2}$ (mean $\pm$ 1 standard error over three seeds). "Mixed" uses half-precision weights and gradients with a mixed-precision forward pass. Memory is a rounded peak GPU allocation.

Model	Rank	Memory (GB)	Precision	Train L_2	Test L_2
Low-Rank Only	25%	46	Full	5.37 ± 0.08	17.19 ± 0.23
		29	Mixed	6.92 ± 0.19	17.09 ± 0.19
Galore (Matricized)	25%	46	Full	31.73 ± 1.27	34.56 ± 1.43
	50%	49	Full	29.63 ± 1.46	33.11 ± 1.21
Sparse Only	25%	46	Full	6.39 ± 0.32	18.73 ± 0.08
		29	Mixed	7.37 ± 0.14	18.54 ± 0.34
TensorGRaD (**ours**)	5%+20%	46	Full	5.36 ± 0.05	**16.82** ± 0.18
		29	Mixed	6.42 ± 0.15	16.87 ± 0.15
Adam Baseline	100%	52	Full	3.94 ± 0.22	17.02 ± 0.18
		37	Mixed	4.86 ± 0.26	17.01 ± 0.19

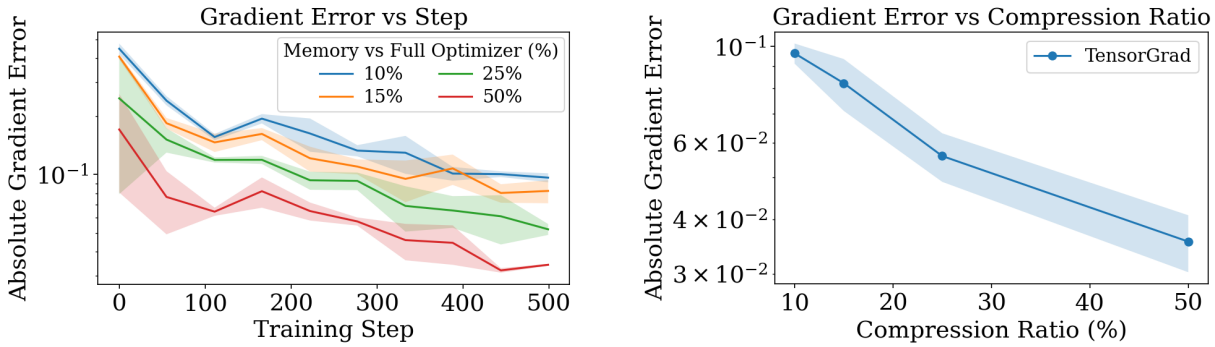

Figure 4: **Gradient reconstruction error under different compression ratios TensorGRaD. Left:** Error vs. training step for 10–50% of the full optimizer state. **Right:** Final-step error vs. memory budget (lower % = stronger compression). Both plots show mean ±1 std across three seeds.

Results on Navier–Stokes 1024 × 1024. We present results on the Navier–Stokes dataset at 1024×1024 resolution with $\mathrm{Re} = 10^5$, focusing on the performance–memory trade-offs achieved by TensorGRaD. Further results for other PDEs and detailed ablations are included in the appendix. We highlight how combining low-rank and sparse gradient compression outperforms each technique in isolation and demonstrate that TensorGRaD is compatible with mixed-precision training.

Tab. 1 summarizes the trade-off between memory and accuracy on the challenging NS1024 dataset. TensorGRaD achieves the best performance across all settings: using just a 25% optimizer state (5% unstructured top-k sparse entries and 20% low-rank), it achieves the lowest test loss of 16.82×10^{-2} surpassing all baselines, including full-precision Adam (17.02×10^{-2}). In comparison, low-rank gradient tensor decomposition (*low-rank-only*) compression at 25% yields 17.19×10^{-2}, and *structured sparsity* at 25% is noticeably worse at 18.54×10^{-2}. These results indicate that turbulent, high-resolution flows benefit from the robust decomposition that combines both low-rank and sparse components. By contrast, the matrix-based GaLore approach (Zhao et al., 2024), which flattens tensor gradients, performs poorly on this task, with test losses of 34.56×10^{-2} and 33.11×10^{-2} at 25% and 50% rank, respectively. This highlights the importance of preserving tensor structure in gradient compression.

Mixed-precision training further reduces memory without harming accuracy. With TensorGRaD, mixed-precision yields 16.87×10^{-2} and still matches full-precision Adam while reducing the total memory by 45%. We also observe that methods that match or exceed Adam in test loss often exhibit higher training loss, suggesting that compression introduces a beneficial regularization effect.

Table 2: **Test L_2 loss ($\times 10^{-3}$) for different combinations of low-rank (LR), structured sparse (SS), and unstructured sparse (US) gradient updates**. Each cell shows top-k / rand-k results. Sequential forms (denoted $A \to B$) apply A to the full gradient and B to the residual. "Sum" applies both independently to the full gradient and sums the results.

Method (topk / randk)	20%+5%	45%+5%	5%+20%
LR → SS	7.09 / 6.44	6.91 / 6.32	6.72 / 6.35
LR → US	6.29 / 6.20	6.22 / 6.19	6.47 / 6.26
SS → LR	6.56 / 6.26	6.66 / 5.96	6.22 / 6.21
US → LR	6.24 / 6.10	6.19 / 6.03	**5.72** / 5.73
LR + US (sum)	6.18 / 6.12	6.17 / 6.06	–

Sparse and low-rank combinations. We evaluate different combinations of low-rank (LR) and sparse gradient compression in TensorGRaD, varying the order, sparsity type, and selection strategy. We distinguish between structured sparsity (SS), which selects aligned slices across modes, and unstructured sparsity (US), which selects arbitrary entries. For selection strategies, we compare top-k (based on magnitude) and rand-k (uniform sampling). In sequential variants (denoted $A \to B$), the first component receives the full

gradient and the second compresses the residual. Additive variants (denoted $A + B$) apply both directly to the gradient and sum their outputs.

The best-performing configuration is US $\rightarrow$ LR with 5% sparsity and 20% low-rank, achieving test losses of 5.72 (top-k) and 5.73 (rand-k), outperforming both compression techniques applied in isolation. In contrast, reversing the order (LR $\rightarrow$ US) leads to higher test losses (e.g., 6.29 and 6.20), indicating that removing outliers first improves the quality of the low-rank basis. Structured sparsity performs worse across all variants; for instance, LR $\rightarrow$ SS at $5\% + 20\%$ results in test errors 6.72 (top-k) and 6.35 (rand-k). Despite some variants achieving similar accuracy, their practicality may differ. For example, configurations with higher unstructured sparsity require storing a large index set, increasing memory and compute overhead. In contrast, the $5\% + 20\%$ US $\rightarrow$ LR setup balances accuracy and memory.

Table 3: **Navier–Stokes (128×128, $\mathrm{Re} = 10^3$) precision ablation**: test $L_2 \times 10^{-3}$ under three precision schemes. **FP**: full precision. **Mixed-1**: gradients, weights, and activations (except FFT) in half precision; optimizer states in full. **Mixed-2** is identical to Mixed-1 but stores optimizer states in half precision. **LR**: low-rank; **US/SS**: unstructured/structured sparse. Lower is better; best per column is **bold**.

Method	Full	Mixed-1	Mixed-2 (Half Optim. states)
Adam	**5.66**	5.62	**6.92**
LR 50%	5.71	5.70	7.08
LR 25%	6.02	5.87	7.21
SS 50%	5.54	**5.56**	7.14
SS 25%	6.22	6.14	7.78
LR+US 25% (5+20)	5.72	5.71	7.10

Gradient reconstruction dynamics. Fig. 4 shows gradient reconstruction error over training for Navier–Stokes 128 (128×128, $\mathrm{Re} = 10^3$) at different memory budgets. Across all budgets (10–50%, using 5% sparse and the remainder low-rank), the error decreases over time, consistent with gradients becoming increasingly low-rank during training (Zhao et al., 2024). Stronger compression (e.g., 10%) gives higher error. Final-step error decreases monotonically with larger budgets, but this does not translate into proportional gains in task loss: Fig. 3 shows only a small L_2 improvement from 25% to 50% despite lower reconstruction error.

Mixed-precision training. We evaluate three different precision configurations on the Navier–Stokes 128 dataset (128×128, $\mathrm{Re} = 10^3$). In the first setting, all tensors and optimizer states are stored in full precision. The second, referred to as Mixed-1, uses half precision for weights, activations, and gradients except the Fast Fourier Transform (FFT) part (see Sec. 2.3), while keeping optimizer states in full precision. Mixed-2 is identical to Mixed-1, except that optimizer states are also stored in half precision.

Results in Tab. 3 show that in full precision, TensorGRaD (25%) matches Adam (5.72 vs. 5.66×10^{-3}) and outperforms low-rank (25%: 6.02) and structured sparse (25%: 6.22). Structured sparsity at 50% performs best (5.54), but TensorGRaD offers better efficiency at lower memory. When moving to Mixed-1 precision TensorGRaD maintains strong performance, with a test loss of 5.71×10^{-3}. This matches the full-precision setting and confirms that TensorGRaD remains stable under reduced numerical precision. Notably, some models improve slightly (e.g., Adam drops to 5.62), consistent with prior observations that mixed precision can act as a mild regularizer (Tu et al., 2024). In contrast, the Mixed-2 setup, where optimizer states are also stored in half precision, leads to significant degradation. TensorGRaD drops to 7.10, Adam to 6.92, and low-rank (25%) to 7.21. These results underscore two key findings. First, TensorGRaD is fully compatible with mixed-precision training, provided that optimizer states are maintained in full precision. Second, TensorGRaD's low-rank and sparse optimizer states preserve essential gradient information more effectively than directly storing them in reduced precision. This makes TensorGRaD a compelling approach for reducing memory without sacrificing accuracy, especially when paired with mixed precision. Our improved memory is shown in Fig. 2.

We show additional results in the Appendix I. Appendix I.1 evaluates the compatibility of TensorGRaD with TFNO (Kossaifi et al., 2024), showing that optimizer-state compression remains effective even when

the FNO weights are Tucker-factorized. Appendix I.2 compares TensorGRaD vs Adam on the 3D ShapeNet Car benchmark using the Geometry-Informed Neural Operator (GINO). Appendix I.3, we ablate the effect of different update frequencies for computing factor matrices for the tensor low-rank part and for the sparse part. Appendix I.5 compares structured sparsity patterns using top-k versus random-k selection strategies. In Appendix I.4, we provide a detailed comparison between our tensor-based low-rank decomposition and a baseline method that applies a GaLore-style low-rank projection to matricized tensors. Finally, Appendix 8 includes extended benchmark results across multiple datasets: Burgers, Darcy, and ElectroMagnetic.

4 Related Work

Our work, **TensorGRaD**, introduces a novel approach to efficiently training neural operators by decomposing gradients. While significant work has been done in related areas, the specific approach of gradient decomposition in tensors has not been explored. **Tensor Methods in Deep Learning:** Tensor decomposition has been widely used to compress deep networks (Novikov et al., 2015; Lebedev et al., 2015; Kim et al., 2016; Panagakis et al., 2021), but these methods focus on weight tensors rather than gradients during training. **Sparse gradient updates:** GRASS (Muhamed et al., 2024) introduced structured sparsity for matrices using sampling strategies like Top$-k$ magnitude sampling. In tensor settings, this approach requires unfolding tensors, disrupting the inherent mode-wise structure. Instead, we use unstructured sparsity, selecting individual tensor entries directly without unfolding them.

Neural Operators: Recent advancements have led to neural operators (Li et al., 2020; Kovachki et al., 2021), with FNOs showing remarkable success in scientific computing tasks. However, these methods have not explored gradient decomposition for memory efficiency. **Efficient Training Techniques:** Various approaches reduce memory footprints of large models. LoRA (Hu et al., 2022) adds a low-rank weight matrix to a frozen pre-trained matrix. FLoRA (Si et al., 2024) extends this to higher dimensions using Tucker decomposition. LoQT (Loeschcke et al., 2024a) rewrites GaLore as LoRA, optimizing only one low-rank factor in 16-bit precision and keeping the rest in 4-bit. For neural operators, MG-TFNO (Kossaifi et al., 2024) combines tensor decomposition with multi-grid approaches, while iFNO (George et al., 2024) incrementally scales FNO weight ranks during training. **Low-rank Plus Sparse:** Robust PCA (Candès et al., 2009) separates low-rank matrices from sparse noise, extended to tensors by RTD (Gu et al., 2014). Hybrid decompositions have been explored for model weights (Han et al., 2024) and attention matrices (Chen et al., 2021), but not for compressing gradient tensors. **Mixed Precision Training** utilizes lower precision formats for certain operations, reducing memory usage and potentially accelerating training on compatible hardware (Tu et al., 2024). **Combination with existing methods** TensorGRaD can complement many existing techniques, potentially leading to greater memory benefits by integrating with methods like FLoRA or MG-TFNO and frameworks like iFNO.

5 Conclusion

We presented **TensorGRaD**, a memory-efficient optimization framework for training large-scale tensor-structured models. By combining low-rank factorization with unstructured sparse updates in a robust tensor decomposition of the gradients, TensorGRaD achieves substantial memory savings without sacrificing performance. We further introduce a mixed-precision training strategy that complements our method and improves efficiency. We validate our findings on challenging PDE benchmarks and thorough ablations. Our approach enables training of large-scale neural operators on high-resolution PDE data.

Limitations. While TensorGRaD delivers substantial memory savings, it also has limitations. The Tucker decomposition introduces a modest 5–20% slowdown (see Appx. K for detailed timings), even when amortized with infrequent updates. Choosing ranks, sparsity levels, and update intervals remains manual and may vary across layers or training stages. Automatically adapting these parameters remains an open problem. Finally, our evaluation focuses on PDE-driven neural operators; extending TensorGRaD to other domains such as vision or language models is future work.

Broader Impact. By enabling high-resolution scientific models to train on commodity hardware, TensorGrad broadens access to advanced simulation tools and large-scale scientific ML. We hope it will support wider adoption of efficient neural operators in science and engineering.

References

Marcella Astrid and Seung-Ik Lee. Cp-decomposition with tensor power method for convolutional neural networks compression, 2017. URL https://arxiv.org/abs/1701.07148.

Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. *Nat Rev Phys*, 6:320–328, 2024. doi: https://doi.org/10.1038/s42254-024-00712-5.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere, 2023. URL https://arxiv.org/abs/2306.03838.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *CoRR*, abs/0912.3599, 2009. URL http://arxiv.org/abs/0912.3599.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015. URL https://arxiv.org/abs/1512.03012.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL https://arxiv.org/abs/1604.06174.

Robert Joseph George, Jiawei Zhao, Jean Kossaifi, Zongyi Li, and Anima Anandkumar. Incremental spatial and spectral learning of neural operators for solving large-scale pdes, 2024. URL https://arxiv.org/abs/2211.15188.

Quanquan Gu, Huan Gui, and Jiawei Han. Robust tensor decomposition with gross corruption. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/a35e3de33d5f2c599af18f2ac7db2ab0-Paper.pdf.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Jawanpuria, and Bamdev Mishra. Sltrain: a sparse plus low rank approach for parameter and memory efficient pretraining. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Christopher Hillar and Lek-Heng Lim. Most tensor problems are np-hard, 2013. URL https://arxiv.org/abs/0911.1393.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2022.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3): 455–500, 2009. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research*, 20(26):1–6, 2019. URL http://jmlr.org/papers/v20/18-277.html.

Jean Kossaifi, Nikola Kovachki, Kamyar Azizzadenesheli, and Anima Anandkumar. Multi-grid tensorized fourier neural operator for high-resolution pdes. *arXiv preprint arXiv:2403.00071*, 2024.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015.

Donghyun Lee, Dingheng Wang, Yukuan Yang, Lei Deng, Guangshe Zhao, and Guoqi Li. Qttnet: Quantized tensor train neural networks for 3d object and video recognition. *Neural Networks*, 141:420–432, 2021. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2021.05.034. URL https://www.sciencedirect.com/science/article/pii/S0893608021002306.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3d pdes, 2023. URL https://arxiv.org/abs/2309.00583.

Ye Liu and Michael K. Ng. Deep neural network compression by tucker decomposition with nonlinear response. *Know.-Based Syst.*, 241(C), April 2022. ISSN 0950-7051. doi: 10.1016/j.knosys.2022.108171. URL https://doi.org/10.1016/j.knosys.2022.108171.

Sebastian Loeschcke, Mads Toftrup, Michael J. Kastoryano, Serge Belongie, and Vésteinn Snæ bjarnarson. Loqt: Low-rank adapters for quantized pretraining. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 115282–115308. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d0c6bc641a56bebee9d985b937307367-Paper-Conference.pdf.

Sebastian Loeschcke, Dan Wang, Christian Leth-Espensen, Serge Belongie, Michael J. Kastoryano, and Sagie Benaim. Coarse-to-fine tensor trains for compact visual representations, 2024b. URL https://arxiv.org/abs/2406.04332.

Aashiq Muhamed, Oscar Li, David Woodruff, Mona Diab, and Virginia Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients. *arXiv preprint arXiv:2406.17660*, 2024.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569*, 2015.

Yannis Panagakis, Jean Kossaifi, Grigorios G. Chrysos, James Oldfield, Mihalis A. Nicolaou, Anima Anandkumar, and Stefanos Zafeiriou. Tensor methods in computer vision and deep learning. *Proceedings of the IEEE*, 109(5):863–890, 2021. doi: 10.1109/JPROC.2021.3074329.

Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavsky, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable low-rank tensor decomposition for compression of convolutional neural network, 2020. URL https://arxiv.org/abs/2008.05441.

Chongjie Si, Xuehui Wang, Xue Yang, Zhengqin Xu, Qingyun Li, Jifeng Dai, Yu Qiao, Xiaokang Yang, and Wei Shen. Flora: Low-rank core space for n-dimension, 2024. URL https://arxiv.org/abs/2405.14739.

Renbo Tu, Colin White, Jean Kossaifi, Boris Bonev, Nikola Kovachki, Gennady Pekhimenko, Kamyar Azizzadenesheli, and Anima Anandkumar. Guaranteed approximation bounds for mixed-precision neural operators, 2024. URL https://arxiv.org/abs/2307.15034.

Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

Chuwei Wang, Julius Berner, Zongyi Li, Di Zhou, Jiayun Wang, Jane Bae, and Anima Anandkumar. Beyond closure models: Learning chaotic-systems via physics-informed neural operators. *arXiv preprint arXiv:2408.05177*, 2024.

Bijiao Wu, Dingheng Wang, Guangshe Zhao, Lei Deng, and Guoqi Li. Hybrid tensor decomposition in neural network compression. *Neural Networks*, 132:309–320, December 2020. ISSN 0893-6080. doi: 10.1016/j.neunet.2020.09.006. URL http://dx.doi.org/10.1016/j.neunet.2020.09.006.

Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling, 2025. URL https://arxiv.org/abs/2409.15156.

Miao Yin, Yang Sui, Siyu Liao, and Bo Yuan. Towards efficient tensor decomposition-based dnn model compression with optimization framework, 2021. URL https://arxiv.org/abs/2107.12422.

Xiao Zhang, Lingxiao Wang, and Quanquan Gu. A unified framework for nonconvex low-rank plus sparse matrix recovery. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1097–1107. PMLR, 09–11 Apr 2018. URL https://proceedings.mlr.press/v84/zhang18c.html.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Appendix

A Neural Operators

Neural Operators $\mathcal{G}_\theta : \mathcal{A} \times \theta \to \mathcal{U}$ combine linear integral operators $\mathcal{K}$ with pointwise non-linear activations σ to approximate non-linear operators, mapping initial conditions $a \in \mathcal{A}$ to solutions $u \in \mathcal{U}$. Their operation is defined as $\mathcal{G}_\theta := \mathcal{Q} \circ (W_L + \mathcal{K}_L) \circ \cdots \circ \sigma(W_1 + \mathcal{K}_1) \circ \mathcal{P}$, where $\mathcal{P}$ and $\mathcal{Q}$ are pointwise neural networks for encoding and decoding, W_l are linear operators, $\mathcal{K}_l$ are integral kernel operators, and σ are activation functions.

The **Fourier Neural Operator (FNO)** proposes a specific convolution operator for $\mathcal{K}$, defined as $(\mathcal{K}v_l)(x) = \mathcal{F}^{-1}(R \cdot T_K \mathcal{F} v_l)(x)$, where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and its inverse, R is a learnable transformation, and T_K truncates to the lowest K Fourier modes. This formulation allows FNO to be discretization-invariant, producing high-quality solutions for query points not in the training grid and enabling transfer between different grid resolutions and discretizations.

B Dataset

B.1 Navier-Stokes Datasets

Navier-Stokes 1024: We use the 2D Kolmogorov flow of Wang et al. (Wang et al., 2024), a periodically forced, incompressible variant of the Navier–Stokes equations. The velocity field $\mathbf{u}(x, y, t) \in \mathbb{R}^2$ evolves on a periodic domain $[0, 2\pi]^2$ according to:

$$\partial_t \mathbf{u} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p + \nu \Delta \mathbf{u} + (\sin(4y), 0)^T, \quad \nabla \cdot \mathbf{u} = 0, \quad (x, y, t) \in [0, L]^2 \times \mathbb{R}_+,$$

Our analysis focuses on the vorticity form, where the vorticity $\omega = \nabla \times \mathbf{u}$ evolves as:

$$\partial_t \omega = -\mathbf{u} \cdot \nabla \omega + \nu \Delta \omega + \nabla \times (\sin(4y), 0)^T.$$

The behavior of this flow is characterized by the Reynolds number $Re = \frac{\overline{u} l}{\nu}$, where ν is the kinematic viscosity, $\overline{u}$ is the root-mean-square velocity, and l is the characteristic length. Higher Re values correspond to more turbulent flows. In our setup, $L = 2\pi$ and $\nu = 10^{-4}$, leading to $Re \approx 2 \times 10^5$, representing a highly turbulent regime.

Initial Condition and Data Collection: The initial velocity field $\mathbf{u_0}(x)$ is sampled from a Gaussian random field $\mathcal{N}(0, C)$ with covariance

$$C = 7^{3/2}(-\Delta + 49I)^{-2.5}.$$

Data is collected from $t = 30$ onward, ensuring a statistically steady state. Our dataset, sourced from (Wang et al., 2024), comprises $[7681, 1024, 1024]$ entries, from which we generate 1577 training samples and 279 test samples, with $\Delta t = 256$. Figure 5 illustrates 9 representative samples from the dataset, selected at uniform intervals of 256 steps.

Input-Output Pair Construction: Input-output pairs are defined by:

- T: The timestep difference between the input and output.
- t_{skip}: The number of frames skipped between samples.

For example, with indices $[0, 1, 2, \ldots, 32]$, setting $T = 16$ and $t_{\text{skip}} = 8$ creates pairs $[(0, 16), (8, 24), (16, 32)]$. Our setup uses $T = 256$ and $t_{\text{skip}} = 4$, resulting in 1577 training samples and 279 test samples.

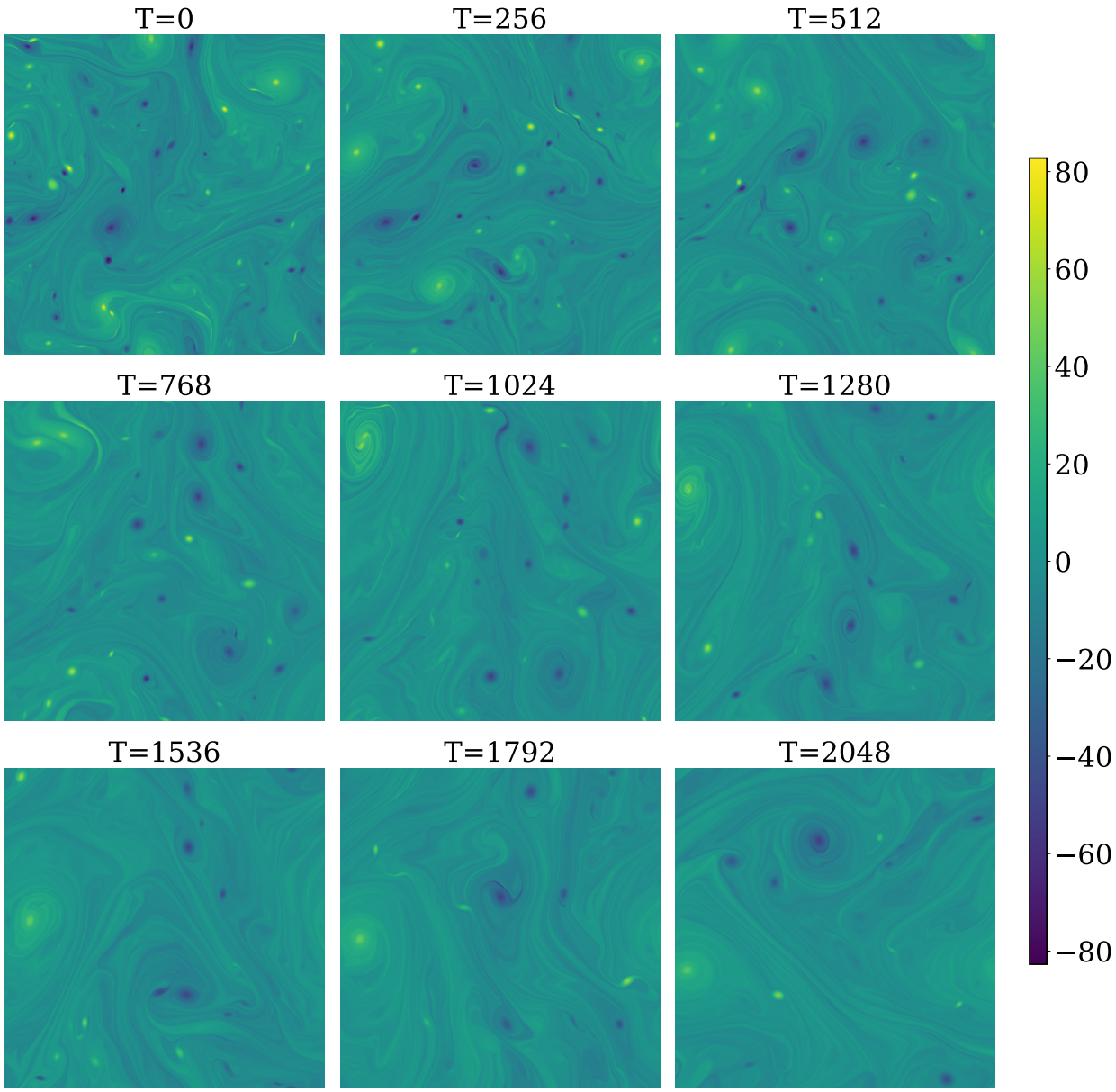

Figure 5: Navier-Stokes 1024×1024 with Reynolds number at 2×10^5

Navier-Stokes (NS128): We also experiment with the two-dimensional Navier-Stokes equation in vorticity form:

$$\begin{aligned} \partial_t \omega + \nabla^\perp \phi \cdot \omega &= \frac{1}{Re}\Delta\omega + f, \quad x \in \mathbb{T}^2, t \in (0, T] \\ -\Delta\phi = \omega, \quad \int_{\mathbb{T}^2} \phi &= 0, \quad x \in \mathbb{T}^2, t \in (0, T] \end{aligned} \tag{1}$$

with a Reynolds number $Re = 10^3$ and final time $T = 5$. The domain is discretized on a 1024×1024 grid. We generate 10,000 training samples and 2,000 test samples using a pseudo-spectral method. To test scalability, we also evaluate our approach on a subsampled resolution of 128×128. Memory profiling is performed at the full 1024×1024 resolution.

Burgers Equation: We consider the one-dimensional Burgers equation on the torus:

$$\partial_t u + u u_x = \nu u_{xx}, \quad x \in \mathbb{T}, t \in (0, T] \tag{2}$$

with initial condition $u_0 \in L^2(\mathbb{T}; \mathbb{C})$ and viscosity $\nu > 0$. We set $T = 1$ and $\nu = 0.01$. Input functions are sampled from a Gaussian random field, and solutions are obtained using a pseudo-spectral method. We use 1000 samples for training and 200 for testing, with 128 resolution.

Darcy Flow: The Darcy flow problem is defined by the elliptic PDE:

$$-\nabla \cdot (a(x)\nabla u(x)) = f(x), \quad x \in (0, 1)^2 \tag{3}$$

with boundary conditions $u(x) = 0$ for $x \in \partial(0, 1)^2$. The input a is sampled from a Gaussian random field, and f is fixed. We use 4000 training samples and 1000 test samples, with the domain discretized on a 421 $\times$ 421 grid.

Electromagnetic Wave Propagation: Lastly, we present a dataset that represents complex-valued data inherently. We consider the propagation of optical pulses in a nonlinear waveguide with second-order nonlinearity (κ^2). The problem is governed by the nonlinear Schrödinger equation (NLSE) with additional terms for second-harmonic generation:

$$\frac{\partial A}{\partial z} = -\mathrm{i}\frac{\beta_2}{2}\frac{\partial^2 A}{\partial t^2} + \mathrm{i}\gamma|A|^2 A + \mathrm{i}\kappa A^* e^{\mathrm{i}\Delta k z} \tag{4}$$

where A is the complex electric field envelope, i is the imaginary unit, z is the propagation distance, t is time, β_2 is the group velocity dispersion, γ is the nonlinear parameter, κ is the coupling coefficient for second-harmonic generation, and Δk is the phase mismatch. Our dataset consists of 800 training samples and 200 testing samples. The input consists of several parameters: the poling region length ranging from 2mm to 15mm, the poling period mismatch varying from -50nm to +50nm, and the pump pulse energy spanning from a few fJ to thousands of fJ. Additionally, the input includes the complex electric field envelope of the input pulse. The output of the system is the complex electric field envelope of the resulting output pulse.

C FNO Memory Usage

Fig. 6 illustrates the memory usage patterns in Fourier Neural Operators (FNOs) as the number of modes increases. This analysis provides crucial insights into the scalability challenges faced when training large FNO models.

As evident from the figure, the memory consumption is divided into two main categories: activation memory and optimizer memory. The activation memory, represented by the dark green bars, remains relatively constant and low across different numbers of modes. This stability in activation memory is a positive

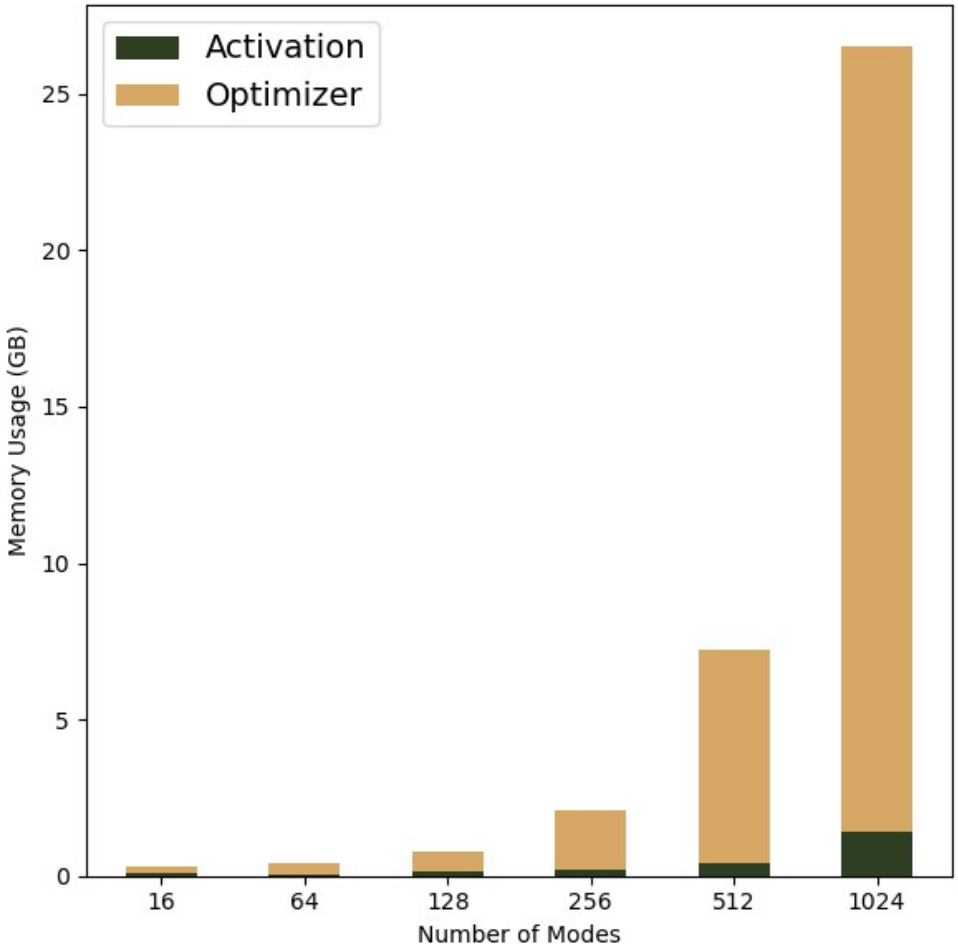

Figure 6: Memory usage in FNO as a function of the number of modes

attribute of FNOs, indicating that the forward and backward passes do not significantly increase memory requirements as the model complexity grows.

However, the optimizer memory, shown in yellow, exhibits a dramatic increase as the number of modes grows. This exponential growth in optimizer memory becomes particularly pronounced for models with more than 128 modes. For instance, when the number of modes reaches 1024, the optimizer memory dominates the total memory usage, far exceeding the memory required for activations.

This trend highlights a critical bottleneck in scaling FNO models to higher resolutions or more complex problems. The optimizer's memory footprint, which includes storage for gradients, momentum, and adaptive learning rate parameters, becomes the primary limiting factor. This observation motivates the need for memory-efficient optimization techniques like **TensorGRaD**, which specifically target the reduction of optimizer memory usage while maintaining model performance.

D Matrix GaLore

GaLore (Gradient Low-Rank Projection) (Zhao et al., 2024) is a memory-efficient optimization method designed to reduce the memory overhead of gradient updates by leveraging the low-rank structure often present in gradient matrices. Specifically, GaLore operates on weight matrices $W \in \mathbb{R}^{m \times n}$ and their corresponding gradient matrices $G \in \mathbb{R}^{m \times n}$. For a given rank r, GaLore computes the SVD of the gradient matrix every T steps, forms projection matrices using the first r singular vectors, then projects the gradient onto this

low-rank subspace to perform optimization. After computing the optimizer update, the gradients are projected back to their full rank for use in the model. This approach allows GaLore to maintain a low memory footprint by storing and updating only the low-rank representations of gradients.

Challenges of applying GaLore to neural operators. In order to apply standard GaLore to tensor weights, the weights must first be reshaped into a matrix to compute the SVD for projection into a low-rank space. GaLore takes one rank parameter, r, and projects high-rank gradients onto the first r basis vectors of the corresponding SVD rotation matrix. When the weight matrix corresponds to an operator that maps between vectors, a single rank cutoff can be applied while preserving most information.

However, in the tensor case, weights correspond to higher-order maps between function spaces. Depending on the chosen strategy for reshaping tensor weights into a matrix, applying a single-dimension rank cutoff to the matrix may discard key information - for instance, for a tensor $W \in \mathbb{C}^{A\times B\times m\times m}$, where A is the number of input channels, B is the number of output channels, and m is the number of truncated Fourier basis modes along each dimension, reshaping W into $W' \in \mathbb{C}^{ABm\times m}$ and cutting off the first dimension at rank r may remove all information about Fourier modes along the first dimension, making function learning impossible. We call this method *GaLore* and provide several comparisons to demonstrate its flaws.

One flaw is the **Loss of mode-specific information**: by collapsing multiple tensor dimensions into one matrix dimension, we lose the ability to preserve different amounts of information along each tensor mode. The other is that we have an **imbalanced projection**: Projecting only on one side of the reshaped matrix (e.g., only U or only V from the SVD) can severely limit the operator's capacity. However, projecting on both sides often leads to training instability and failure to converge. This method also encounters **rank selection issues**: Choosing a single rank cutoff for the reshaped matrix makes it difficult to balance information preservation across all the original tensor dimensions. A rank that preserves enough information for one dimension may be too restrictive for another.

E Tensor Decomposition

Tensors are multidimensional arrays that generalize the concepts of vectors (first-order tensors) and matrices (second-order tensors) to higher orders. An Nth-order tensor $\mathcal{X} \in \mathbb{C}^{I_1\times I_2\times\cdots\times I_N}$ is an N-way array where each mode n has dimension I_n. Like matrices, in tensors, we can decompose the tensors into low-rank factors using the Tucker decomposition, also known as the higher-order SVD (HOSVD), which decomposes a tensor into a core tensor multiplied by a matrix along each mode:

$$\mathcal{X} \approx \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \cdots \times_N U^{(N)} = [\![\mathcal{G}; U^{(1)}, U^{(2)}, \ldots, U^{(N)}]\!] \tag{5}$$

where $\mathcal{G} \in \mathbb{C}^{R_1\times R_2\times\cdots\times R_N}$ is the core tensor, $U^{(n)} \in \mathbb{C}^{I_n\times R_n}$ are factor matrices, and $\times_n$ denotes the n-mode product. Two critical aspects of the Tucker decomposition make it particularly suitable for our **TensorGRaD** method:

1. **Equivalence to SVD in 2D**: In the special case of 2D tensors (matrices), the Tucker decomposition reduces to the familiar SVD. The core tensor $\mathcal{G}$ becomes equivalent to the diagonal matrix Σ in SVD, while the factor matrices correspond to the orthogonal matrices U and V (Kolda and Bader, 2009). This property ensures that our method seamlessly extends the principles of matrix-based techniques to higher-order tensors.

2. **Orthogonality of factor matrices**: The factor matrices $U^{(n)}$ in Tucker decomposition are orthogonal, mirroring the properties of U and V in SVD. This orthogonality is crucial for the efficiency and stability of the GaLore method. Specifically:

(a) *Projection efficiency*: The orthogonality allows us to project tensors onto the subspace spanned by these matrices through simple matrix multiplication, without the need for costly inverse computations.

(b) *Easy inversion*: When we need to reverse the projection, we can simply use the transpose of these orthogonal matrices instead of computing their inverses. This property is expressed mathematically as $(U^{(n)})^T U^{(n)} = I$, where I is the identity matrix.

(c) *Numerical stability*: Orthogonal matrices have a condition number of 1, ensuring that the projection and its inverse are numerically stable operations, even for high-dimensional tensors.

Tensor decompositions such as Tucker have been extensively applied to deep learning architectures, but primarily for compressing or accelerating *weights* rather than *gradients*. Prior work includes applications of low-rank and tensorized models in convolutional networks, recurrent networks, transformers, 3D rendering, and operator learning (Phan et al., 2020; Wu et al., 2020; Astrid and Lee, 2017; Lee et al., 2021; Yin et al., 2021; Loeschcke et al., 2024b; Liu and Ng, 2022; Kossaifi et al., 2024; Novikov et al., 2015; Lebedev et al., 2015; Kim et al., 2016; Panagakis et al., 2021)

E.1 Implementation

The **low-rank component** of TensorGRaD is implemented using TensorLy (Kossaifi et al., 2019), which provides an efficient implementation of the Tucker decomposition based on Higher-Order Orthogonal Iteration (HOI). Given an input tensor $\mathcal{X}$, HOI approximates the factor matrices ${U^{(n)}}_n$ by computing truncated SVDs of mode-n unfoldings and iteratively refines them to minimize the Frobenius reconstruction error. The decomposition supports warm restarts, allowing factors to be reused across steps to reduce iteration cost. All subsequent operations, like compressing the gradients and reconstructing the low-rank updates, are performed using PyTorch.

E.2 Rank selection heuristic.

Since optimal multilinear Tucker ranks are NP-hard (Hillar and Lim, 2013) to determine, we use a memory-budget based heuristic. Given a tensor $\mathcal{G} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$ and a desired compression ratio $\rho \in (0, 1)$ for the low-rank component, we select ranks (r_1, r_2, r_3, r_4) such that:

$$\sum_{i=1}^{4} n_i r_i \leq \rho \cdot \prod_{i=1}^{4} n_i.$$

In practice, we use uniform rank percentages per mode unless otherwise specified. Ablations in Appendix I.3 show that performance is robust across a wide range of ranks around this target.

F Structured Sparse Projections

Sparse projections provide an alternative to low-rank compression by selectively retaining a subset of gradient values. GRASS (Muhamed et al., 2024) introduced this approach for gradient matrices, where optimizer statistics are maintained only for a selected subset of rows, leading to substantial memory savings without requiring an SVD.

This idea can be directly extended to tensor gradients by applying structured sparsity, where the mask Ω selects entire slices along one or more tensor modes. For a gradient tensor $\mathcal{G} \in \mathbb{C}^{I_1 \times \cdots \times I_N}$, the structured mask is defined as a Cartesian product of per-mode index sets:

$$\Omega = M^{(1)} \times M^{(2)} \times \cdots \times M^{(N)},$$

and the projected tensor is $\hat{\mathcal{G}} = \mathcal{G}[\Omega]$. This form of sparsity is efficient, as the mask stores only $\sum_n r_n$ indices, and projection is implemented via multi-dimensional slicing.

The selection strategies presented in GRASS (Muhamed et al., 2024) can also be applied to tensors for constructing the sets $M^{(n)}$, including random sampling (Rand-k), and norm-based rules such as Top-k and Prob-k. However, norm-based selection requires unfolding the tensor along each mode and computing slice norms $s^{(n)}i = \|\mathcal{G}i :: \cdots\|_2$, which may discard important structural information depending on the unfolding.

Structured sparsity is particularly effective when gradient mass is concentrated along specific tensor modes. However, it lacks flexibility in targeting scattered or irregularly positioned high-importance entries.

To restore the sparse tensor to its original shape during optimizer updates, we use a back-projection operation that places the retained values back into their original positions and fills all other entries with zeros. This is implemented using the adjoint of the projection operator:

$$\tilde{\mathcal{G}} = \mathcal{P}\Omega^\top(\hat{\mathbf{g}}) =, \mathrm{unvec}\left(P\Omega^\top \hat{\mathbf{g}}\right),$$

where $\hat{\mathbf{g}} = P_\Omega \,\mathrm{vec}(\mathcal{G})$ is the compressed vector of retained entries.

G Mixed Precision.

Our empirical findings are consistent with the mixed-precision analysis of Tu et al. (Tu et al., 2024), who show that reduced-precision errors in FNOs can be bounded by discretization errors. We go one step further by storing both weights and gradients in half precision while keeping optimizer states in full precision. When we quantize the optimizer states themselves to half precision (Mixed-2), performance degrades substantially (Tab. 3), likely because the limited mantissa cannot represent the small second-moment values (often around 10^{-5}).

H Profiling Methodology

To analyze the performance and memory usage of our **TensorGRaD** method, we implemented a comprehensive profiling setup using PyTorch's built-in profiler. This allowed us to gain detailed insights into the computational and memory requirements of our algorithm compared to baseline methods.

Detailed Memory Breakdown. We implemented a detailed memory tracking system to break GPU memory usage down into key categories.

PyTorch provides access to a memory profiler, which collects granular information about each block of memory allocated on the CPU and GPU within the context window in which it is invoked. The profiler is run over a set number of iterations, each of which corresponds to a forward and backward pass through the model and one step of the optimizer. The profiler discards data from the first iteration due to the additional overhead of initialization and allocation, as well as a specified number of warmup steps. The profiler collects data for a set number of iterations, and averages this process over a series of repeats. IN our experiments, we discarded 1 step, warmed up for 1 step, collected data for 3 steps, and performed 3 repeats.

To break down memory by category, we relied on the automatic categorization utility provided by the profiler's Memory Timeline, which coalesces all individual blocks into one of eight categories, which we enumerate and explain below.

- **Model parameters**: Tensors registered as instances of `torch.nn.Parameter`.
- **Optimizer states:** Optimizer states are tensors stored within the optimizer that are used to compute the final gradient. For Adam optimizer, this includes first and second moment estimates, which are each a tensor of the same size as the gradients themselves.
- **Inputs:** memory allocations that occur during data loading and preprocessing.
- **Activations:** Temporary tensors created during the forward pass of the model.
- **Gradients:** The tensors added to model parameters during the optimizer step. They are computed for each parameter by backpropagating loss through the model.
- **Autograd Detail:** Extra memory allocated during PyTorch's autograd operation, including memory used for storing computational graphs and intermediate results needed for backpropagation.
- **Temporary:** Temporary Buffers are short-lived tensors that are created and destroyed within a single operation or a small set of operations. For **TensorGRaD**, it is often used in complex computations like FFTs or tensor decompositions within galore.

- **"None":** PyTorch's profiler is often unable to categorize memory immediately upon allocation. In our tests, we compared figures generated by coalescing a profiler's memory timeline one-to-one with more granular block-level traces acquired from PyTorch's CUDA memory snapshot to conclude that memory tagged as None is either eventually reclassified as another type, or corresponds to an intermediate activation that is deallocated during backpropagation. For this reason, we chose to classify "None" memory "Intermediate". For simplicity in the final figure, we also included temporary buffers, autograd detail buffers and activations in the Intermediate category, though our profiler maintains the capability to collect data for each category separately.

To accurately break down the profiler's memory timeline into these categories, we took the following approach. First, we noticed that the peak memory allocation often occured at a step when the total of None memory was very high, and that totals reported during that timestep for other categories were zero or very low. To obtain informative breakdowns for those categories, informed by our analysis of memory tagged as None, we identified the timestep at which peak memory usage occured *after excluding None-tagged memory from the total*. This allowed us to break down all non-intermediate memory by its true classification once properly tagged. Once we obtained this breakdown, we subtracted the total from the true total memory allocation observed at the peak to obtain the true amount of intermediate memory allocated at peak.

Algorithm 2 GaLore

Require: A layer weight tensor $\mathcal{W} \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4}$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T, chosen dimension d.

Initialize first-order moment $\mathcal{M}_0 \in \mathbb{C}^{r \times N_2 \times N_3 \times N_4} \leftarrow 0$ (Assuming matrization 1)
Initialize second-order moment $\mathcal{V}_0 \in \mathbb{R}^{r \times N_2 \times N_3 \times N_4} \leftarrow 0$ (Assuming matrization 1)
Initialize step $t \leftarrow 0$
repeat
  $\mathcal{G}_t \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4} \leftarrow -\nabla_{\mathcal{W}} \phi_t(\mathcal{W}_t)$
  $G_t^{(d)} \leftarrow \text{Reshape}(\mathcal{G}_t, (N_d, \prod_{i \neq d} N_i))$ ▷ Reshape tensor to matrix
  if $t \bmod T = 0$ **then**
    $U, \Sigma, V^\top \leftarrow \text{SVD}(G_t^{(d)})$ ▷ Compute SVD
    $P \leftarrow V[:, : r]^\top$ ▷ Select r right singular vectors
  end if
  $R_t \leftarrow G_t^{(d)} P^\top$ ▷ Project gradient into compact space
  UPDATE(R_t) by Adam:
    $M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot R_t$
    $V_t \leftarrow \beta_2 \cdot V_{t-1} + (1 - \beta_2) \cdot |R_t|^2$
    $M_t \leftarrow M_t / (1 - \beta_1^t)$
    $V_t \leftarrow V_t / (1 - \beta_2^t)$
    $N_t \leftarrow M_t / (\sqrt{V_t} + \epsilon)$
  $\tilde{G}_t^{(d)} \leftarrow \alpha \cdot N_t P$ ▷ Project back to original space
  $\tilde{\mathcal{G}}_t \leftarrow \text{Reshape}(\tilde{G}_t^{(d)}, (N_1, N_2, N_3, N_4))$ ▷ Reshape back to tensor
  $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} + \eta \cdot \tilde{\mathcal{G}}_t$
  $t \leftarrow t + 1$
until convergence criteria met
return $\mathcal{W}_t$

I Additional Results

This section provides extended experimental results demonstrating the effectiveness and generality of **TensorGRaD** across a wide range of PDE datasets, architectures, and dimensional settings. In addition to standard 2D benchmarks such as Burgers, Darcy flow, Navier–Stokes, and Electromagnetic wave propagation, we also include results on the 3D ShapeNet Car benchmark using the Geometry-Informed Neural Operator (GINO), highlighting that TensorGRaD extends beyond FNO-based architectures.

Table 4: TensorGRaD with tensorized weights (TFNO) on Navier–Stokes at two resolutions. We report L2 test loss; TFNO rank denotes retained weight percentage. NS 128 uses Re $\approx 10^3$ (500 steps) and NS 1024 uses Re $\approx 10^5$ (50 steps).

	NS 128 ($\times 10^{-3}$)		NS 1024 ($\times 10^{-2}$)	
TFNO Rank	**Adam**	**TensorGRaD**	**Adam**	**TensorGRaD**
15%	5.787	5.821	24.09	24.21
25%	5.672	5.729	23.13	23.17
50%	5.912	5.967	22.12	22.43
Baseline	5.660	5.720	20.08	20.24

Overall, **TensorGRaD** consistently achieves strong memory–accuracy trade-offs. For Burgers, performance improves steadily with increasing rank. On Darcy flow, **TensorGRaD** achieves up to a 49% gain in test loss at rank 0.25 while reducing optimizer memory by 76%. Navier–Stokes experiments demonstrate that **TensorGRaD** maintains comparable performance at lower ranks while significantly reducing memory usage, and Electromagnetic simulations show improvements of up to 11%.

I.1 Compatibility with TFNO (factorized weights).

Tensorized FNO (TFNO) (Kossaifi et al., 2024) replaces the standard dense convolutional weight tensors in FNO with Tucker-factorized representations, decomposing each weight tensor into low-rank mode factors and a small core tensor to reduce parameter count and model memory while preserving the operator's expressive capacity. Importantly, TFNO constrains the *weights* to remain low-rank throughout training, whereas TensorGRaD instead targets only the *optimizer states*.

We evaluate TensorGRaD on top of TFNO at varying weight ranks on Navier-Stokes 128×128 and 1024×1024. As shown in Tab. 4, TensorGRaD closely matches Adam across all TFNO ranks, demonstrating that optimizer-state compression remains effective even when the weights themselves are factorized. Here, the *baseline* corresponds to the standard FNO trained with full-rank weights (no TFNO factorization), using the same architecture, optimizer, and training schedule.

These results show that TensorGRaD integrates with TFNO, preserving performance across weight ranks while reducing optimizer memory, and highlight the orthogonality between low-rank structure in weights and low-rank structure in gradients. Importantly, the performance difference introduced by TensorGRaD alone is smaller than that caused by TFNO weight factorization. On NS 1024, the baseline increases from 20.08 to 20.24 with TensorGRaD, whereas TFNO at 50% rank increases the error to 22.12, indicating that most of the degradation originates from the weight factorization rather than the compressed optimizer states.

I.2 3D ShapeNet with Geometry-Informed Neural Operator.

In Tab. 5, we show that TensorGRaD also applies to 3D problems and architectures beyond FNO. We evaluate it on the 3D ShapeNet Car (Chang et al., 2015) benchmark using the Geometry-Informed Neural Operator (GINO) (Li et al., 2023), which lifts mesh data to a 3D Eulerian grid, mixes features on the grid, and projects predictions back to the mesh. Using the official training setup, we replace only the optimizer with TensorGRaD (20% low-rank + 5% sparse). TensorGRaD closely matches Adam's performance while reducing optimizer memory by up to 75%, confirming its applicability to 3D data and architectures beyond FNO.

I.3 Update frequency of tensor decomposition ablation.

Fig. 7 illustrates how the projection gap affects TensorGRaD on the NS128 task over the first 150 epochs, comparing 25% and 50% rank settings. In line with findings from (Zhao et al., 2024; Muhamed et al., 2024),

Table 5: Performance on the 3D ShapeNet Car (Chang et al., 2015) benchmark using 3D GINO. We report relative train and test errors (%). TensorGRaD uses 20% low-rank + 5% sparse optimizer states.

Metric	**Adam**	**TensorGRaD**
Train Error (%)	2.94	3.37
Test Error (%)	8.53	8.74

Table 6: Full results for memory and accuracy comparison on Navier–Stokes 1024×1024 with Reynolds number 10^5. Train and test losses are $L_2 \times 10^{-2}$ (mean $\pm$ 1 standard error over three seeds). "Mixed" uses half-precision weights and gradients with a mixed-precision forward pass. Memory is a rounded peak GPU allocation.

Model	Memory (GB)	Precision	Train L_2	Test L_2
Low-Rank Only				
25%	46	Full	5.37 ± 0.08	17.19 ± 0.23
	29	Mixed	6.92 ±0.19	17.09 ±0.19
25% rand-svd	46	Full	5.93 ±0.36	17.23 ±0.21
50% Matrix (d=1)	49	Full	29.63 ± 1.46	33.11 ± 1.21
Sparse Only				
25%	46	Full	6.39 ± 0.32	18.73 ± 0.08
	29	Mixed	7.37 ± 0.14	18.54 ± 0.34
50%	49	Full	5.35 ± 0.08	17.21 ± 0.16
	32	Mixed	6.02 ± 0.28	17.38 ± 0.31
TensorGRaD				
25% (5+20)	46	Full	5.36 ± 0.05	16.82 ± 0.18
	29	Mixed	6.42 ± 0.15	16.87 ± 0.15
Adam Baseline				
No Compression	52	Full	3.94 ± 0.22	17.02 ± 0.18
	37	Mixed	4.86 ± 0.26	17.01 ± 0.19

increasing the projection gap can improve generalization, as switching subspaces introduces a small amount of noise that can harm the performance, if done too frequently

We observe that low-rank projections are less sensitive to the projection gap and tend to perform better in the early stages of training. In our experiments, they consistently lead during roughly the first half of training, after which structured sparsity gradually catches up. A likely explanation is that low-rank projections quickly capture the dominant modes of the gradient with fewer parameters, enabling faster initial convergence. In contrast, structured sparsity requires more iterations to recover the same signal, as it discards parts of the gradient during each step. This slower adaptation may, however, contribute to better generalization by reducing overfitting in later epochs.

I.4 Tensor Low-rank vs matrix GaLore

We highlight that our higher-order low-rank factorization version of TensorGRaD shows superior performance to the direct extension of GaLore as described in D. In Tab. 7), we show results for Darcy Flow comparing a higher-order low-rank factorization to construct the projection matrix instead of naively unfolding the tensor to 2d and removing the inherent structure. Comparing different unfoldings (variations of d), we observed up to a 48% improvement in test loss with a rank of 0.25, while reducing the optimizer state memory from 2.09GB to 0.5GB.

We evaluate three approaches to matricizing a tensor gradient with shape $C_{in} \times C_{out} \times M_x \times M_y$. The first, which we call "rollout=1", combines the last 3 dimensions into one matrix dimension, resulting in a

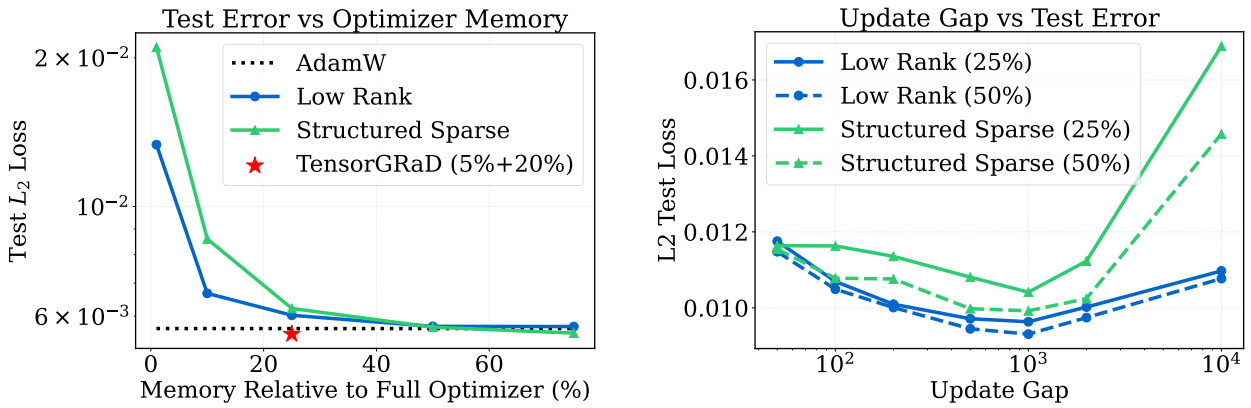

Figure 7: Left: Low-rank, structured, at various rank percentages vs TensorGRaD at 25% and baseline Adam optimizer. Right: Effect of projection gap on L_2 test loss for sparse and low-rank projections at 25% and 50% compression, trained for 150 epochs on the NS128 dataset.

Table 7: Model performance on Darcy-flow.

Model	Test Loss (1e-2) at Rank Ratio						Gain (%)
	0.01	0.1	0.25	0.5	0.75	1.0	
FNO Baseline	-	-	-	-	-	0.205	/
FNO - Tensor Low-rank	**0.147**	**0.108**	**0.105**	**0.107**	**0.140**	**0.173**	**49**
FNO - GaLore (d=1)	0.256	0.232	0.212	0.245	0.201	0.190	8
FNO - GaLore (d=2)	0.203	0.192	0.168	0.178	0.170	0.180	19
FNO - GaLore (d=3)	0.234	0.212	0.201	0.193	0.196	0.182	11

matrix of shape $C_{in} \times (C_{out} * M_x * M_y)$. The second, "rollout=2", combines the first two dimensions into the first matrix dimension and the last two dimensions into the second matrix dimension, resulting in a matrix of shape $(C_{in} * C_{out}) \times (M_x * M_y)$. The last, "rollout=3", combines the last three dimensions into the second matrix dimension, resulting in a matrix of shape $C_{in} \times (C_{out} * M_x * M_y)$. We showcase results and comparisons for all three approaches in Tab. 12.

All of the subsequent results are with varying rank ratios on the **TensorGRaD** method for all datasets. We report both the training and testing loss/accuracy.

I.5 Structured Sparsity Mask Strategy

Tab. 13 shows an ablation of three structured sparse projection strategies: **Top-k**, **Probability**, and **Rand-k**, evaluated at 25% and 50% sparsity.

We find that **Rand-k** consistently outperforms both Top-k and Probability sampling, especially at higher sparsity (50%). This aligns with prior findings in GRASS (Muhamed et al., 2024).

The performance gap stems from how each method selects entries:

- **Top-k** selects slices with the highest norms.
- **Probability** sampling assigns selection probabilities proportional to slice norms.
- **Rand-k** uniformly samples slices at random, without relying on norm computations.

Table 8: Evaluating low-rank tensor factorization across various tasks.

Model	Rank Ratio	Memory (GB)	Train (Loss ($\times 10^{-2}$))	Test H_1 (Loss ($\times 10^{-2}$))	Test L_2 (Loss ($\times 10^{-2}$))	Gain (%)
Darcy						
Baseline	1.0	8.88	0.7151	1.6230	0.2050	/
GaLore (d=2)	0.25	7.34	0.4200	1.3210	0.1680	19
Tensor low-rank	0.25	7.32	**0.2930**	**0.8680**	**0.1050**	**48.8**
Navier-Stokes						
Baseline	1.0	77	1.0630	1.9010	0.6152	/
GaLore (d=1)	0.5	68	4.3340	5.5830	1.9952	-223
Tensor low-rank	0.5	55	1.2340	2.0850	0.6480	-5.4
ElectroMagnetic						
Baseline	1.0	4.83	2.973	0.1902	0.2000	/
GaLore (d=2)	0.25	4.83	2.392	0.1802	0.1900	5
Tensor low-rank	0.25	4.63	**2.132**	**0.1681**	**0.1782**	**11**
Burgers						
Baseline	1.0	3.94	0.0064	0.0050	0.0026	/
GaLore (d=2)	0.5	3.88	0.0052	0.0100	0.0062	-250
Tensor low-rank	0.5	3.87	**0.0026**	**0.0041**	**0.0025**	**+5**

Table 9: Model performance on Burgers

Model	Rank Ratio	Train Loss (1e-4)	Test Loss(1e-4)	Gain (%)
FNO Baseline	Full Rank	0.205	0.262	/
FNO - **Tensor low-rank**	0.1	0.115	0.321	-19
FNO - **Tensor low-rank**	0.25	0.095	0.271	-4
FNO - **Tensor low-rank**	0.5	0.086	0.253	+5
FNO - **Tensor low-rank**	0.75	0.083	0.246	+8
FNO - **Tensor low-rank**	1.00	0.083	**0.242**	**+9**

Table 10: Model performance on Darcy-flow

Model	Rank Ratio	Train Loss (1e-2)	Test Loss(1e-2)	Gain (%)
FNO Baseline	Full Rank	0.715	0.205	/
FNO - **Tensor low-rank**	0.01	0.465	0.147	+30
FNO - **Tensor low-rank**	0.1	0.323	0.108	+48
FNO - **Tensor low-rank**	0.25	**0.293**	**0.105**	**+49**
FNO - **Tensor low-rank**	0.5	0.275	0.107	+49
FNO - **Tensor low-rank**	0.75	0.379	0.140	+40
FNO - **Tensor low-rank**	1.00	0.715	0.173	+16

Table 11: Model performance on EM.

Model	Test Loss (1e-2) at Rank Ratio						Gain (%)
	0.01	0.1	0.25	0.5	0.75	1.0	
FNO Baseline	-	-	-	-	-	0.200	/
FNO - **Tensor low-rank**	0.187	0.185	**0.178**	0.176	0.174	0.206	**11**
FNO - GaLore (d=1)	0.213	0.192	0.193	0.189	0.194	0.200	7
FNO - GaLore (d=2)	0.205	0.206	0.195	0.196	0.201	0.199	3

Table 12: Ablation: GaLore and **Tensor low-rank** Rank Comparison

Method	% orig. parameters	GaLore Test L_2 ($\times 10^{-2}$)	**Tensor low-rank** Test L_2 ($\times 10^{-2}$)
GaLore (d=1)	25	2.495 ± 0.920	**0.9141** ± 0.0064
	50	3.594 ± 0.885	**0.7622** ± 0.0984
	75	3.298 ± 1.96	**0.6697** ± 0.0746
GaLore (d=2)	25	8.715 ± 0.252	**0.9141** ± 0.0064
	50	8.683 ± 0.0014	**0.7622** ± 0.0984
	75	8.950 ± 0.0141	**0.6697** ± 0.0746
GaLore (d=3)	25	8.723 ± 0.0149	**0.9141** ± 0.0064
	50	8.702 ± 0.0108	**0.7622** ± 0.0984
	75	8.585 ± 0.0171	**0.6697** ± 0.0746

Table 13: Ablation of sparse projection methods at 25% and 50% sparsity. Train and test losses are reported as L_2 loss ($\times 10^{-3}$) with mean $\pm$ std over multiple runs.

Method	Train loss	Test L_2
Top-k		
25%	12.05	6.50
50%	12.92	7.93
Probability		
25%	10.91	6.20
50%	9.59	6.07
Rand-k		
25%	12.01	6.22
50%	**9.14**	**5.54**

Table 14: Model performance on EM

Model	Rank Ratio	Train Loss	Test Loss	Gain (%)
Complex FNO Baseline	Full Rank	2.973	0.200	/
Complex FNO - **Tensor low-rank**	0.01	4.198	0.249	-20
Complex FNO - **Tensor low-rank**	0.1	2.936	0.217	-8
Complex FNO - **Tensor low-rank**	0.25	**2.132**	**0.178**	+11
Complex FNO - **Tensor low-rank**	0.5	2.430	0.184	+8
Complex FNO - **Tensor low-rank**	0.75	2.719	0.192	+4
Complex FNO - **Tensor low-rank**	1.00	2.397	0.185	+8

J Architecture and Training Details

Sobolev Loss for PDE Training. In training NOs for PDEs we employ both the L^2 and Sobolev H^1 losses to provide a comprehensive assessment of model performance. While the L^2 loss measures point-wise accuracy of predictions, the H^1 loss, defined as $\|u - \hat{u}\|_{H^1}^2 = \|u - \hat{u}\|_{L^2}^2 + \|\nabla u - \nabla \hat{u}\|_{L^2}^2$, accounts for both the function values and their gradients. This is particularly crucial for PDEs, as it ensures that the learned solutions not only match the target values but also preserve the smoothness and differential properties inherent in the physical systems being modeled.

Sobolev Loss for Complex Wave Phenomena. The Sobolev H^1 loss proves especially valuable when dealing with complex wave phenomena, as demonstrated in our experiments with the EM Dataset using Complex-FNOs. In this case, the H^1 loss not only measures the accuracy of the predicted complex electric field envelope but also ensures that its spatial derivatives are correctly captured. This is crucial for accurately representing the rapid oscillations and sharp peaks characteristic of EM waves. Our results show that **Tensor**

low-rank with a rank ratio of 0.25 achieved an 11% improvement in overall test loss compared to the baseline, with the H^1 loss decreasing from 0.1902 to 0.1681. This improvement is particularly significant given the challenging nature of the EM dataset, which involves predicting the complex electric field envelope resulting from nonlinear interactions in waveguides. The enhanced performance in H^1 loss indicates that our model not only matches the amplitude of the EM waves more accurately but also better captures the rapid spatial variations and peak formations. This is critical in applications such as optical pulse propagation, where precise modeling of field gradients and peak intensities is essential for predicting phenomena like second-harmonic generation and phase matching.

K Slowdown in Training

In this section we quantify the runtime overhead introduced by TensorGRaD. Tab. 16 reports the per-epoch wall-clock time for AdamW, tensor low-rank only, the sparse+low-rank variant (TensorGRaD), and matricized GaLore. All experiments use the same hardware and batch configuration. We use update frequency 1000 for this experiment.

The dominant overhead arises from the Tucker decomposition step; the sparse projection adds only a minor additional cost. Because the decomposition is recomputed every T steps, its cost is amortized, resulting in a modest 5–20% slowdown depending on the chosen rank and update frequency.

Despite this overhead, TensorGRaD reduces optimizer memory by up to 75%, which in practice allows training at higher resolutions or with larger batch sizes on the same hardware. We also note that TensorGRaD is slightly slower than tensor low-rank alone due to the additional sparse projection, but both remain within a relatively small overhead range.

Moreover, we have incorporated techniques such as "warm-restart" initialization of the tensor decomposition to amortize the computational overhead across training iterations. This helps minimize the impact on the overall training efficiency. We have also explored opportunities to further optimize the tensor decomposition computations, which could potentially reduce the training time slowdown even further.

Remark 2 (Real-Valued Analysis) For clarity of presentation, we develop the theory of **Tensor low-rank** assuming all tensors are real-valued, i.e., $\mathcal{W}_l, \mathcal{G}_t \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times N_4}$ and all associated operations are in real space. This simplification allows us to focus on the core geometric and algebraic properties without the additional complexity of complex conjugates and Hermitian operations. The extension to complex-valued tensors (as needed for Fourier Neural Operators where weights may be complex in the frequency domain) is straightforward: inner products become Hermitian inner products, transposes become conjugate transposes, and orthogonality conditions incorporate complex conjugates. All main results remain valid with these natural modifications.

L Parameter Complexity Analysis

To understand the theoretical advantages of **Tensor low-rank** over matrix-based GaLore, we provide a detailed analysis of the parameter complexity for both approaches. This analysis demonstrates why tensor decomposition leads to more efficient memory usage while maintaining expressiveness.

L.1 Memory Analysis

We provide a theoretical analysis of the memory requirements for **Tensor low-rank** compared to baseline methods and matrix GaLore variants. Consider a weight tensor $W \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4}$ in a FNO Spectral layer. Tab. 17 summarizes the memory requirements for different methods. The baseline approach stores the full tensor and its corresponding optimizer states. For a rank ratio r $(0 < r \leq 1)$, **Tensor low-rank** requires storing the factor matrices, resulting in substantial memory savings, especially for the optimizer states. In this table, we assume the use of a complex-valued Adam optimizer, which typically requires two additional tensors (first and second moments) for each parameter.

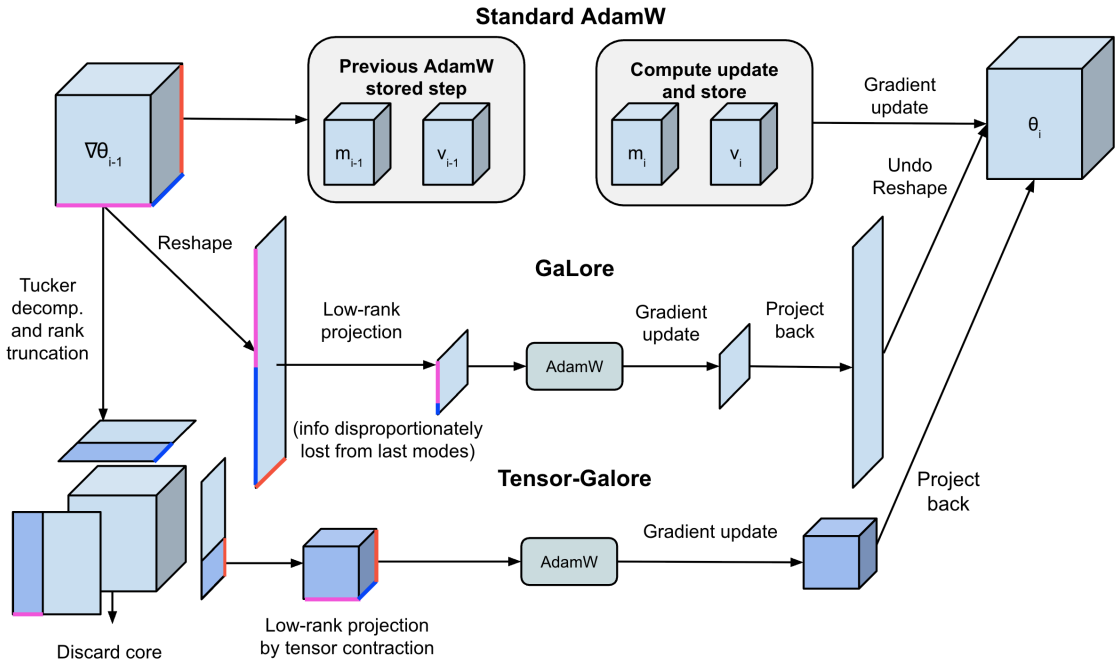

Figure 8: Comparison of the higher-order low-rank decomposition Tensor Low-Rank with standard Adam and GaLore. GaLore applies matrix-based low-rank projection after reshaping tensors. Our **Tensor low-rank** method leverages tensor decomposition to perform low-rank projection directly on tensor gradients, preserving multidimensional structure.

Algorithm 3 Adam with low-rank tensor decomposition

Require: A layer weight tensor $\mathcal{W} \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4}$. Step size η, scale factor α, decay rates β_1, β_2, rank r, subspace change frequency T.

Initialize first-order moment $\mathcal{M}_0 \in \mathbb{C}^{r \times r \times r \times r} \leftarrow 0$
Initialize second-order moment $\mathcal{V}_0 \in \mathbb{C}^{r \times r \times r \times r} \leftarrow 0$
Initialize step $t \leftarrow 0$
repeat
  $\mathcal{G}_t \in \mathbb{C}^{N_1 \times N_2 \times N_3 \times N_4} \leftarrow -\nabla_{\mathcal{W}} \phi_t(\mathcal{W}_t)$
  if $t \bmod T = 0$ **then**
    $\mathcal{C}, \{U^{(n)}\}_{n=1}^4 \leftarrow \text{Projection}(\mathcal{G}_t, \text{rank} = r)$ instead of tucker
  else
    $\mathcal{C}, \{U^{(n)}\}_{n=1}^4 \leftarrow \mathcal{C}_{t-1}, \{U_{t-1}^{(n)}\}_{n=1}^4$ ▷ Reuse the previous projector.
  end if
  $\mathcal{R}_t \leftarrow \mathcal{G}_t \times_1 {U^{(1)}}^\top \times_2 {U^{(2)}}^\top \times_3 {U^{(3)}}^\top \times_4 {U^{(4)}}^\top$ ▷ Project gradient into compact space.
  UPDATE($\mathcal{R}_t$) by Adam:
    $\mathcal{M}_t \leftarrow \beta_1 \cdot \mathcal{M}_{t-1} + (1 - \beta_1) \cdot \mathcal{R}_t$
    $\mathcal{V}_t \leftarrow \beta_2 \cdot \mathcal{V}_{t-1} + (1 - \beta_2) \cdot |\mathcal{R}_t \bar{\mathcal{R}}_t|$ ▷ We use the complex conjugate update.
    $\mathcal{M}_t \leftarrow \mathcal{M}_t / (1 - \beta_1^t)$
    $\mathcal{V}_t \leftarrow \mathcal{V}_t / (1 - \beta_2^t)$
    $\mathcal{N}_t \leftarrow \mathcal{M}_t / (\sqrt{\mathcal{V}_t} + \epsilon)$
  $\tilde{\mathcal{G}}_t \leftarrow \alpha \cdot \mathcal{N}_t \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)}$ ▷ Project back to original space.
  $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} + \eta \cdot \tilde{\mathcal{G}}_t$
  $t \leftarrow t + 1$
until convergence criteria met.
return $\mathcal{W}_t$

L.1.1 Memory analysis: matricized GaLore vs tensor low-rank.

Consider a 4D FNO spectral weight tensor $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$, where (I_1, I_2) are input/output channels and (I_3, I_4) are Fourier modes. A direct matrix GaLore extension first matricizes $\mathcal{W}$, e.g. $\mathbf{W}_{(1)} \in \mathbb{R}^{I_1 \times (I_2 I_3 I_4)}$ (and permutations) or $\mathbf{W}_{(12)} \in \mathbb{R}^{(I_1 I_2) \times (I_3 I_4)}$, and then applies a rank-R SVD. Storing the low-rank factors (ignoring the lower-order $+O(R)$ singular values) costs $P_{(1)} \approx R(I_1 + I_2 I_3 I_4)$ or $P_{(12)} \approx R(I_1 I_2 + I_3 I_4)$ parameters, so the memory scales with products of tensor mode sizes once the tensor is flattened.

In tensor low-rank, we instead use a Tucker decomposition with ranks (R_1, R_2, R_3, R_4): $\mathcal{W} \approx \mathcal{G} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)} \times_4 U^{(4)}$, where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3 \times R_4}$ and $U^{(n)} \in \mathbb{R}^{I_n \times R_n}$. The storage is $P_{\text{Tucker}} = R_1 R_2 R_3 R_4 + \sum_{n=1}^{4} I_n R_n$, which preserves mode-wise structure and replaces multiplicative scaling with additive mode-wise terms plus a small core.

For a symmetric layer with $I_1 = I_2 = N$ and $I_3 = I_4 = M$, using equal Tucker ranks $R_1 = \cdots = R_4 = r$ and a matched matrix rank $R = r^2$ for the best-case unfolding $\mathbf{W}_{(12)} \in \mathbb{R}^{N^2 \times M^2}$, we obtain $P_{\text{Matrix}} \approx r^2(N^2 + M^2)$ versus $P_{\text{Tensor}} = r^4 + 2rN + 2rM$. For example, with $N = 64$, $M = 128$, and $r = 16$, $P_{\text{Matrix}} \approx 5{,}242{,}880$ while $P_{\text{Tensor}} = 71{,}680$, a $\approx 73\times$ reduction in stored scalars. This gap increases further as the mode resolution M grows, which matches the high-resolution neural operator regime where optimizer-state memory dominates.

M Tensor Operations and Notation

Definition 1 (Tensor) An order-d tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is a d-dimensional array with entries $a_{i_1, i_2, \ldots, i_d}$, where $1 \le i_k \le I_k$ for $k = 1, \ldots, d$.

Definition 2 (Mode-k Unfolding) The mode-k unfolding of a tensor $\mathcal{A}$, denoted $\mathcal{A}_{(k)} \in \mathbb{R}^{I_k \times (I_1 \cdots I_{k-1} I_{k+1} \cdots I_d)}$, arranges the mode-$k$ fibers as columns of the resulting matrix. Concretely,

$$(\mathcal{A}_{(k)})_{i_k, j} = a_{i_1, \ldots, i_d},$$

where j is the lexicographic index corresponding to $(i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_d)$ under the chosen unfolding convention.

Definition 3 (Mode-k Product) The mode-k product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with a matrix $U \in \mathbb{R}^{J \times I_k}$, denoted $\mathcal{A} \times_k U$, is a tensor $\mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times J \times I_{k+1} \times \cdots \times I_d}$ with entries

$$(\mathcal{A} \times_k U)_{i_1, \ldots, i_{k-1}, j, i_{k+1}, \ldots, i_d} = \sum_{i_k = 1}^{I_k} a_{i_1, \ldots, i_k, \ldots, i_d} \, u_{j, i_k}.$$

Proposition 1 (Properties of the Mode-k Product) For a tensor $\mathcal{A}$ and matrices U, V of compatible sizes:

1. $(\mathcal{A} \times_k U)_{(k)} = U \, \mathcal{A}_{(k)}$.
2. If $k \neq l$, then $\mathcal{A} \times_k U \times_l V = \mathcal{A} \times_l V \times_k U$.
3. $\mathcal{A} \times_k U \times_k V = \mathcal{A} \times_k (VU)$ (when VU is defined).

Definition 4 (Tensor Inner Product) For tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$, the (Frobenius) inner product is

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1 = 1}^{I_1} \cdots \sum_{i_d = 1}^{I_d} a_{i_1, \ldots, i_d} \, b_{i_1, \ldots, i_d}.$$

Definition 5 (Tensor Norms) For a tensor $\mathcal{A}$:

1. Frobenius norm: $\|\mathcal{A}\|_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$.
2. Mode-k spectral norm: $\|\mathcal{A}\|_{(k)} = \|\mathcal{A}_{(k)}\|_2$.

3. Spectral (multilinear operator) norm:

$$\|\mathcal{A}\| = \max_{\|x^{(1)}\|_2=\cdots=\|x^{(d)}\|_2=1} \left|\mathcal{A} \times_1 x^{(1)} \times_2 \cdots \times_d x^{(d)}\right|.$$

Definition 6 (Tensor Outer Product) The outer product of vectors $u^{(k)} \in \mathbb{R}^{I_k}$ for $k = 1, \ldots, d$ is the tensor $\mathcal{A} = u^{(1)} \circ u^{(2)} \circ \cdots \circ u^{(d)}$ with entries

$$a_{i_1,\ldots,i_d} = u^{(1)}_{i_1} u^{(2)}_{i_2} \cdots u^{(d)}_{i_d}.$$

Definition 7 (Tensor Contraction) Let $\mathcal{A} \in \mathbb{R}^{I_1\times\cdots\times I_d}$ and let $p \neq q$ with $I_p = I_q$. The contraction of $\mathcal{A}$ along modes p and q is the order-$(d-2)$ tensor $\mathrm{Contract}_{p,q}(\mathcal{A})$ with entries

$$(\mathrm{Contract}_{p,q}(\mathcal{A}))_{i_1,\ldots,i_{p-1},i_{p+1},\ldots,i_{q-1},i_{q+1},\ldots,i_d} = \sum_{t=1}^{I_p} a_{i_1,\ldots,i_{p-1},t,i_{p+1},\ldots,i_{q-1},t,i_{q+1},\ldots,i_d}.$$

M.1 Tensor Trace and Inner Products

Definition 8 (Trace and partial trace) For a matrix $M \in \mathbb{R}^{n\times n}$, $\mathrm{tr}(M) = \sum_{i=1}^n M_{ii}$. For a tensor $\mathcal{A} \in \mathbb{R}^{I_1\times\cdots\times I_d}$ and two modes $p \neq q$ with $I_p = I_q$, the (partial) trace over modes (p, q) is the contraction

$$\mathrm{tr}_{p,q}(\mathcal{A}) \;:=\; \mathrm{Contract}_{p,q}(\mathcal{A}) \in \mathbb{R}^{I_1\times\cdots\times I_{p-1}\times I_{p+1}\times\cdots\times I_{q-1}\times I_{q+1}\times\cdots\times I_d},$$

i.e., it sums entries where the p-th and q-th indices are tied. In particular, for $d = 2$ (a matrix), $\mathrm{tr}_{1,2}(\mathcal{A}) = \mathrm{tr}(\mathcal{A})$.

Proposition 2 (Trace/inner-product identities used in gradient derivations) Let $\mathcal{W} \in \mathbb{R}^{I_1\times\cdots\times I_d}$ and let $\mathcal{G} \in \mathbb{R}^{I_1\times\cdots\times I_d}$. Then

$$\langle \mathcal{W}, \mathcal{G}\rangle \;=\; \langle \mathcal{W}_{(k)}, \mathcal{G}_{(k)}\rangle \;=\; \mathrm{tr}\big(\mathcal{W}_{(k)}^\top \mathcal{G}_{(k)}\big) \quad \text{for any mode } k,$$

since unfolding is a permutation of entries. In particular, if $\mathcal{G}$ is rank-1 in the first mode in the sense that $\mathcal{G}_{(1)} = x\, y^\top$ for some $x \in \mathbb{R}^{I_1}$ and $y \in \mathbb{R}^{I_2\cdots I_d}$ (equivalently, $\mathcal{G} = \mathrm{fold}_1(x \otimes y)$), then

$$\langle d\mathcal{W}, \mathcal{G}\rangle \;=\; \mathrm{tr}\big(d\mathcal{W}_{(1)}^\top x\, y^\top\big) \;=\; \mathrm{tr}\big(x^\top d\mathcal{W}_{(1)}\, y\big).$$

Consequently, for any scalar ϕ,

$$d\phi \;=\; \langle d\mathcal{W}, \mathcal{G}\rangle \quad\Longrightarrow\quad \frac{\partial\phi}{\partial\mathcal{W}} \;=\; \mathcal{G}.$$

Proof 1 The equality $\langle \mathcal{W}, \mathcal{G}\rangle = \langle \mathcal{W}_{(k)}, \mathcal{G}_{(k)}\rangle$ holds because unfolding preserves all entries and the Frobenius inner product is entrywise. For matrices, $\langle A, B\rangle = \mathrm{tr}(A^\top B)$, hence $\langle \mathcal{W}_{(k)}, \mathcal{G}_{(k)}\rangle = \mathrm{tr}(\mathcal{W}_{(k)}^\top \mathcal{G}_{(k)})$. If $\mathcal{G}_{(1)} = x\, y^\top$, then $\mathrm{tr}(d\mathcal{W}_{(1)}^\top x\, y^\top) = \mathrm{tr}(x^\top d\mathcal{W}_{(1)}\, y)$ by cyclicity of trace. Finally, if $d\phi = \langle d\mathcal{W}, \mathcal{G}\rangle$ for all perturbations $d\mathcal{W}$, then by definition of the gradient under the Frobenius pairing we have $\partial\phi/\partial\mathcal{W} = \mathcal{G}$.

Example 1 (Logsoftmax-style rank-1 gradient form) In the logsoftmax computation (under the reversible-network setup), one encounters differentials of the form

$$-d\phi = \mathrm{tr}\Big(d\mathcal{W}_l^\top \left(\mathcal{J}_l^\top P_1^\perp y\right) f_{l-1}^\top\Big) = \Big\langle d\mathcal{W}_l,\ \left(\mathcal{J}_l^\top P_1^\perp y\right) \otimes f_{l-1}\Big\rangle,$$

which yields the gradient tensor (matrix in this case) $\mathcal{G}_l \;=\; \left(\mathcal{J}_l^\top P_1^\perp y\right) \otimes f_{l-1}$..

M.2 Stable Rank for Tensors

Definition 9 (Matrix stable rank) For a nonzero matrix A, the stable rank is

$$\mathrm{sr}(A) := \frac{\|A\|_F^2}{\|A\|_2^2},$$

where $\|\cdot\|_F$ is the Frobenius norm and $\|\cdot\|_2$ is the spectral norm.

Definition 10 (Tensor stable rank (mode-wise)) For a nonzero tensor $\mathcal{T} \in \mathbb{R}^{N_1 \times \cdots \times N_d}$, define the mode-wise stable ranks

$$\mathrm{sr}_k(\mathcal{T}) := \frac{\|\mathcal{T}\|_F^2}{\|\mathcal{T}_{(k)}\|_2^2}, \qquad k = 1, \ldots, d,$$

and collect them as $\mathrm{sr}(\mathcal{T}) := [\mathrm{sr}_1(\mathcal{T}), \ldots, \mathrm{sr}_d(\mathcal{T})]$. Here $\mathcal{T}_{(k)}$ denotes the mode-k unfolding and $\|\mathcal{T}\|_F^2 = \sum_{i_1,\ldots,i_d} \mathcal{T}_{i_1,\ldots,i_d}^2$.

Lemma 1 (Unfolding preserves Frobenius norm) For any tensor $\mathcal{T}$ and any mode k, $\|\mathcal{T}\|_F = \|\mathcal{T}_{(k)}\|_F$.

Proposition 3 (Basic properties of tensor stable rank) Let $\mathcal{T} \neq 0$. For each mode k:

1. $\mathrm{sr}_k(\mathcal{T}) \geq 1$.
2. $\mathrm{sr}_k(\mathcal{T}) \leq \mathrm{rank}(\mathcal{T}_{(k)})$.
3. $\mathrm{sr}_k(\mathcal{T})$ is invariant under orthogonal transforms applied in mode k: if $U \in \mathbb{R}^{N_k \times N_k}$ is orthogonal, then $\mathrm{sr}_k(\mathcal{T} \times_k U) = \mathrm{sr}_k(\mathcal{T})$.
4. If $\mathcal{T} = a^{(1)} \circ \cdots \circ a^{(d)}$ is rank-1, then $\mathrm{sr}_k(\mathcal{T}) = 1$ for all k.

Proof 2 (1) For any matrix M, $\|M\|_F^2 \geq \|M\|_2^2$. Using $\|\mathcal{T}\|_F = \|\mathcal{T}_{(k)}\|_F$ gives $\mathrm{sr}_k(\mathcal{T}) = \|\mathcal{T}_{(k)}\|_F^2 / \|\mathcal{T}_{(k)}\|_2^2 \geq 1$. (2) If $\mathrm{rank}(M) = r$, then $\|M\|_F^2 \leq r\,\|M\|_2^2$, so $\|M\|_F^2/\|M\|_2^2 \leq r$; apply this to $M = \mathcal{T}_{(k)}$. (3) If U is orthogonal, then $\|U\mathcal{T}_{(k)}\|_F = \|\mathcal{T}_{(k)}\|_F$ and $\|U\mathcal{T}_{(k)}\|_2 = \|\mathcal{T}_{(k)}\|_2$, hence the ratio defining sr_k is unchanged. (4) For rank-1 $\mathcal{T} = a^{(1)} \circ \cdots \circ a^{(d)}$, every unfolding $\mathcal{T}_{(k)}$ is a rank-1 matrix, and rank-1 matrices satisfy $\|M\|_F = \|M\|_2$, giving $\mathrm{sr}_k(\mathcal{T}) = 1$.

Definition 11 (Multilinear stable rank) For a nonzero tensor $\mathcal{T}$, define the multilinear stable rank

$$\mathrm{msr}(\mathcal{T}) := \min_{k \in \{1,\ldots,d\}} \mathrm{sr}_k(\mathcal{T}).$$

Remark 3 (Relation to Tucker ranks) Mode-wise stable ranks provide a scale-free proxy for the effective ranks of the unfoldings $\mathcal{T}_{(k)}$. In Tucker-style compression, one chooses mode ranks $(r_1, \ldots, r_d)$ and approximates $\mathcal{T} \approx \mathcal{G} \times_1 U^{(1)} \times_2 \cdots \times_d U^{(d)}$; empirically, taking r_k on the order of $\mathrm{sr}_k(\mathcal{T})$ often preserves most of the Frobenius energy of $\mathcal{T}$ in mode k.

Remark 4 (Application to FNO) For FNO tensors $\mathcal{R} \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times N_4}$, modes 1–2 correspond to input/output channels and modes 3–4 correspond to Fourier modes. In many PDE settings, gradients and weights exhibit spectral decay in the Fourier directions, so $\mathrm{sr}_3(\cdot)$ and $\mathrm{sr}_4(\cdot)$ can be substantially smaller than channel-mode stable ranks, motivating mode-wise (tensor) low-rank projections rather than a single rank after matricization.

M.3 Projected PSD assumption (used in our theory)

Our contraction and stable-rank results only require positive semidefiniteness after projection onto the fixed mode-wise subspaces. Concretely, for each mode k we use the projected matrices

$$\hat{\mathcal{B}}_{it}^{(k)} := P_k^\top (\mathcal{B}_i(\mathcal{W}_t))_{(k)} P_k, \qquad \hat{\mathcal{C}}_{it}^{(k)} := P_k^\top (\mathcal{C}_i(\mathcal{W}_t))_{(k)} P_k,$$

which are square by construction. We assume $\hat{\mathcal{B}}_{it}^{(k)} \succeq 0$ and $\hat{\mathcal{C}}_{it}^{(k)} \succeq 0$ for the relevant modes k. This ensures the projected curvature quantity

$$\kappa_t^{(k)} := \frac{1}{N} \sum_{i=1}^{N} \lambda_{\min}\big(\hat{\mathcal{B}}_{it}^{(k)}\big)\, \lambda_{\min}\big(\hat{\mathcal{C}}_{it}^{(k)}\big)$$

is well-defined and nonnegative, which is exactly what is used in Theorem 2 and Lemma 11.

N Reversibility of Fourier Neural Operators

N.1 Definition and preliminaries

Definition 12 (Reversibility (from (Zhao et al., 2024))) A network $\mathcal{N}$ mapping $x \mapsto y = \mathcal{N}(x)$ is reversible if there exists a matrix-valued map $J(x)$ (depending on x and the weights) such that

$$y = J(x)\,x, \qquad dx = J(x)^\top dy.$$

This notion is structural and does not require $J(x)$ to be invertible.

Lemma 2 (Closure under composition) If $\mathcal{N}_1$ and $\mathcal{N}_2$ are reversible with $y_1 = J_1(x)x$ and $y_2 = J_2(y_1)y_1$, then $\mathcal{N}_2 \circ \mathcal{N}_1$ is reversible with

$$y_2 = J_{2\circ 1}(x)\,x, \quad \text{where} \quad J_{2\circ 1}(x) = J_2(y_1)J_1(x),$$

and its backward map satisfies $dx = J_{2\circ 1}(x)^\top dy_2$.

Proof 3 Substitute $y_1 = J_1(x)x$ into $y_2 = J_2(y_1)y_1$ to obtain $y_2 = (J_2(y_1)J_1(x))x$. For backprop, apply the given backward relations in sequence: $dy_1 = J_2(y_1)^\top dy_2$ and then $dx = J_1(x)^\top dy_1$. Combining gives $dx = J_1(x)^\top J_2(y_1)^\top dy_2 = (J_2(y_1)J_1(x))^\top dy_2 = J_{2\circ 1}(x)^\top dy_2$.

Lemma 3 (Closure under residual addition) If $\mathcal{N}(x) = J(x)x$ is reversible, then the residual map $\mathcal{R}(x) = x + \mathcal{N}(x)$ is reversible with

$$\mathcal{R}(x) = (I + J(x))x, \qquad dx = (I + J(x))^\top d\mathcal{R}(x).$$

Proof 4 Immediate from linearity: $\mathcal{R}(x) = x + J(x)x = (I + J(x))x$ and $dx = (I + J(x))^\top d\mathcal{R}(x)$ follows by applying the reversible backward rule to $J(x)x$ and adding the identity contribution.

N.2 Spectral convolution layer

Lemma 4 (Spectral convolution layer is reversible) Consider the (bias-free) FNO spectral convolution layer K acting on v via

$$Kv := \mathcal{F}^{-1}\big(R\,\mathcal{F}v\big),$$

where $\mathcal{F}$ is the discrete Fourier transform and R is the learnable Fourier-space weight operator. Then K is reversible with

$$J_K = \mathcal{F}^{-1}R\mathcal{F}, \qquad dv = J_K^\top dy.$$

Proof 5 Forward: the map is linear in v, hence $Kv = J_K v$ with $J_K = \mathcal{F}^{-1}R\mathcal{F}$. For the backward relation, view K as a composition of three linear maps: $v \mapsto \hat{v} = \mathcal{F}v$, $\hat{v} \mapsto R\hat{v}$, and $R\hat{v} \mapsto \mathcal{F}^{-1}(R\hat{v})$. The Jacobian of a linear map is itself, so the adjoint/backprop operator is the transpose of the composed operator:

$$dv = (\mathcal{F}^{-1}R\mathcal{F})^\top dy = \mathcal{F}^\top R^\top (\mathcal{F}^{-1})^\top dy.$$

Under the standard real/unitary FFT convention, $\mathcal{F}^\top = \mathcal{F}^{-1}$ and $(\mathcal{F}^{-1})^\top = \mathcal{F}$, giving

$$dv = \mathcal{F}^{-1}R^\top \mathcal{F}\,dy = J_K^\top dy,$$

which is exactly the reversibility condition. (For complex-valued FFT/weights, replace $(\cdot)^\top$ by the Hermitian adjoint $(\cdot)^*$.)

N.3 Channel-mixing linear layer

Lemma 5 (Bias-free linear layer is reversible) A linear layer $v \mapsto Wv$ (no bias) is reversible with $J_W(x) = W$.

Proof 6 Forward: $y = Wv = J_W v$. Backward: $dv = W^\top dy = J_W^\top dy$.

N.4 Pointwise activation

Lemma 6 (Reversible pointwise activations) Let $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ be applied elementwise. Suppose there exists a diagonal matrix $J_\sigma(z)$ (depending on z) such that $\sigma(z) = J_\sigma(z)z$ and $dz = J_\sigma(z)^\top d\sigma(z)$. Then σ is reversible in the GaLore sense.

Proof 7 The assumed representation $\sigma(z) = J_\sigma(z)z$ gives the forward form. The assumed backward rule $dz = J_\sigma(z)^\top d\sigma(z)$ gives the reverse-mode relation with the same $J_\sigma(z)$.

Example 2 (LeakyReLU) For $\sigma(z) = \max(az, z)$ with $0 < a < 1$, define $J_\sigma(z) = \mathrm{diag}(\mathbf{1}[z > 0] + a\,\mathbf{1}[z \leq 0])$. Then $\sigma(z) = J_\sigma(z)z$ and $dz = J_\sigma(z)^\top d\sigma(z)$.

N.5 Full FNO block and full network

Lemma 7 (One FNO block is reversible) Consider a bias-free FNO block

$$\mathcal{B}(v) := \sigma\big(W(Kv)\big),$$

where K is the spectral convolution layer, W is a bias-free channel-mixing linear map, and σ is a reversible pointwise activation. Then $\mathcal{B}$ is reversible, and one valid choice is

$$J_{\mathcal{B}}(v) = J_\sigma\big(W(Kv)\big)\, W\, J_K.$$

Moreover, its backward map satisfies $dv = J_{\mathcal{B}}(v)^\top d\mathcal{B}(v)$.

Proof 8 We expand the forward computation as $v \xrightarrow{K} Kv \xrightarrow{W} W(Kv) \xrightarrow{\sigma} \sigma(W(Kv))$. By Lemma N (spectral layer) we have $Kv = J_K v$. By Lemma (bias-free linear layer) we have $W(Kv) = WJ_K v$. By the activation lemma, $\sigma(z) = J_\sigma(z)z$ with $z = WJ_K v$, hence

$$\mathcal{B}(v) = \sigma(WJ_K v) = J_\sigma(WJ_K v)\, W\, J_K\, v = J_{\mathcal{B}}(v)\, v.$$

For backward, apply reverse-mode in the same order: from $u = \sigma(z)$ we have $dz = J_\sigma(z)^\top du$; from $z = Ww$ we have $dw = W^\top dz$; from $w = J_K v$ we have $dv = J_K^\top dw$. Combining yields $dv = J_K^\top W^\top J_\sigma(z)^\top du = (J_\sigma(z)WJ_K)^\top du = J_{\mathcal{B}}(v)^\top d\mathcal{B}(v)$.

Lemma 8 (Full FNO is reversible) Let $\mathcal{N} = \mathcal{B}_L \circ \cdots \circ \mathcal{B}_1$ be a stack of reversible FNO blocks. Then $\mathcal{N}$ is reversible with

$$J_{\mathcal{N}}(x) = J_{\mathcal{B}_L}(h_{L-1}) \cdots J_{\mathcal{B}_1}(x),$$

where $h_\ell = \mathcal{B}_\ell \circ \cdots \circ \mathcal{B}_1(x)$.

Proof 9 Apply Lemma (closure under composition) repeatedly to $\mathcal{B}_1, \ldots, \mathcal{B}_L$.

Lemma 9 (Gradient form for tensor reversible models (single sample)) Consider a network $\mathcal{N}(x) = \mathcal{N}_L \circ \cdots \circ \mathcal{N}_1(x)$ and let $f_\ell := \mathcal{N}_\ell \circ \cdots \circ \mathcal{N}_1(x)$ denote the layer outputs (with $f_0 = x$, $f_L = \mathcal{N}(x)$). Fix a layer index l and assume that the l-th layer is a (bias-free) linear tensor map in the sense that

$$f_l \;=\; \mathcal{W}_l \times_1 f_{l-1},$$

where $\mathcal{W}_l \in \mathbb{R}^{N_1 \times N_2 \times N_3 \times N_4}$ and $\times_1$ is the mode-1 product (Sec. M). Let $\mathcal{J}_l$ denote the Jacobian of the tail network $\mathcal{N}_{l+1:L} := \mathcal{N}_L \circ \cdots \circ \mathcal{N}_{l+1}$ evaluated at f_l, i.e.

$$\mathcal{J}_l \;:=\; D\big(\mathcal{N}_{l+1:L}\big)(f_l), \qquad \text{so that} \qquad df_L \;=\; \mathcal{J}_l\, df_l.$$

Then for batch size 1:

(a) Squared loss. For $\phi := \frac{1}{2}\|f_L - y\|_2^2$, the gradient w.r.t. $\mathcal{W}_l$ is

$$\nabla_{\mathcal{W}_l}\phi \;=\; \big(\mathcal{J}_l^\top (f_L - y)\big) \otimes f_{l-1}.$$

Equivalently, the negative gradient is $-\nabla_{\mathcal{W}_l}\phi = \big(\mathcal{J}_l^\top (y - f_L)\big) \otimes f_{l-1}$.

(b) Softmax cross-entropy and small-logit approximation. For $\phi(y; f_L) := -y^\top \log \mathrm{softmax}(f_L)$ (with $y^\top \mathbf{1} = 1$), the exact gradient is

$$\nabla_{\mathcal{W}_l} \phi \;=\; \big(\mathcal{J}_l^\top (\mathrm{softmax}(f_L) - y)\big) \otimes f_{l-1}.$$

Moreover, if $\|P_1^\perp f_L\|_\infty \ll \sqrt{K}$ (small logits), then using the first-order expansion $\mathrm{softmax}(f_L) \approx \frac{1}{K}\mathbf{1} + \frac{1}{K} P_1^\perp f_L$ gives the approximation

$$\nabla_{\mathcal{W}_l} \phi \;\approx\; \Big(\mathcal{J}_l^\top \Big(\tfrac{1}{K} P_1^\perp f_L - P_1^\perp y\Big)\Big) \otimes f_{l-1} \;=\; \Big(-\mathcal{J}_l^\top P_1^\perp y + \tfrac{1}{K} \mathcal{J}_l^\top P_1^\perp f_L\Big) \otimes f_{l-1}.$$

If additionally $f_L = \mathcal{N}_{l+1:L}(f_l)$ is linearized as $f_L \approx \mathcal{J}_l f_l$ and $f_l = \mathcal{W}_l \times_1 f_{l-1}$, then $\mathcal{J}_l^\top P_1^\perp f_L \approx \mathcal{J}_l^\top P_1^\perp \mathcal{J}_l(\mathcal{W}_l \times_1 f_{l-1})$.

Proof 10 We use the tensor inner product from Sec. M and the trace/inner-product identity from Sec. **??**: for any perturbation $d\mathcal{W}_l$, $d\phi = \langle \nabla_{\mathcal{W}_l} \phi,\, d\mathcal{W}_l \rangle$.

Because $f_l = \mathcal{W}_l \times_1 f_{l-1}$ is linear in $\mathcal{W}_l$, its differential is $df_l = d\mathcal{W}_l \times_1 f_{l-1}$. By definition of $\mathcal{J}_l$ (Jacobian of the tail network), $df_L = \mathcal{J}_l\, df_l = \mathcal{J}_l(d\mathcal{W}_l \times_1 f_{l-1})$. For the squared loss $\phi = \frac{1}{2}\|f_L - y\|_2^2$, we have $d\phi = (f_L - y)^\top df_L$, hence

$$d\phi = (f_L - y)^\top \mathcal{J}_l(d\mathcal{W}_l \times_1 f_{l-1}) = (\mathcal{J}_l^\top (f_L - y))^\top (d\mathcal{W}_l \times_1 f_{l-1}).$$

Using the tensor trace/inner-product identity (Sec. **??**), the right-hand side equals $\langle (\mathcal{J}_l^\top (f_L - y)) \otimes f_{l-1},\, d\mathcal{W}_l \rangle$, which proves $\nabla_{\mathcal{W}_l} \phi = (\mathcal{J}_l^\top (f_L - y)) \otimes f_{l-1}$.

For softmax cross-entropy, the standard identity is $d\phi = (\mathrm{softmax}(f_L) - y)^\top df_L$. Substituting $df_L = \mathcal{J}_l(d\mathcal{W}_l \times_1 f_{l-1})$ and repeating the same inner-product step yields $\nabla_{\mathcal{W}_l} \phi = (\mathcal{J}_l^\top (\mathrm{softmax}(f_L) - y)) \otimes f_{l-1}$. Finally, under $\|P_1^\perp f_L\|_\infty \ll \sqrt{K}$, the first-order expansion $\mathrm{softmax}(f_L) \approx \frac{1}{K}\mathbf{1} + \frac{1}{K} P_1^\perp f_L$ implies $\mathrm{softmax}(f_L) - y \approx \frac{1}{K} P_1^\perp f_L - P_1^\perp y$ (since $y^\top \mathbf{1} = 1$), giving the displayed approximation.

Remark 5 (Shape check (one-sample case)) If $f_L \in \mathbb{R}^K$ and $f_l \in \mathbb{R}^{N_l}$, then $\mathcal{J}_l \in \mathbb{R}^{K \times N_l}$ and $\mathcal{J}_l^\top (f_L - y) \in \mathbb{R}^{N_l}$. The outer product $(\mathcal{J}_l^\top (f_L - y)) \otimes f_{l-1}$ has the same shape as the linear tensor map $\mathcal{W}_l$ under the convention that $\mathcal{W}_l \times_1 f_{l-1} \in \mathbb{R}^{N_l}$ (i.e., mode-1 contraction against f_{l-1} produces the layer output).

Lemma 10 (Tensor gradient form for logsoftmax (small-logit approximation)) Consider the same setup as Lemma 9. Let the loss be the K-way logsoftmax (cross-entropy)

$$\phi(y; f_L) := -y^\top \log \mathrm{softmax}(f_L), \qquad y^\top \mathbf{1} = 1,$$

and define the zero-mean projector $P_1^\perp := I - \frac{1}{K}\mathbf{1}\mathbf{1}^\top$. Under the small-logit assumption $\|P_1^\perp f_L\|_\infty \ll \sqrt{K}$, the gradient w.r.t. $\mathcal{W}_l$ admits the approximation

$$\nabla_{\mathcal{W}_l} \phi \;\approx\; \Big(-\mathcal{J}_l^\top P_1^\perp y + \gamma K^{-1} \mathcal{J}_l^\top P_1^\perp f_L\Big) \otimes f_{l-1}, \qquad \gamma \approx 1,$$

and, using the linearization $f_L \approx \mathcal{J}_l f_l$ with $f_l = \mathcal{W}_l \times_1 f_{l-1}$, equivalently

$$\nabla_{\mathcal{W}_l} \phi \;\approx\; \Big(-\mathcal{J}_l^\top P_1^\perp y + \gamma K^{-1} \mathcal{J}_l^\top P_1^\perp \mathcal{J}_l(\mathcal{W}_l \times_1 f_{l-1})\Big) \otimes f_{l-1}.$$

Proof 11 Write $f_L = \alpha \mathbf{1} + \hat{f}$ where $\hat{f} := P_1^\perp f_L$ and $\alpha := \frac{1}{K}\mathbf{1}^\top f_L$. For $\|\hat{f}\|_\infty \ll \sqrt{K}$, a first-order expansion of softmax around the uniform distribution gives

$$\mathrm{softmax}(f_L) \;\approx\; \frac{1}{K}\mathbf{1} + \frac{1}{K}\hat{f},$$

hence

$$\mathrm{softmax}(f_L) - y \;\approx\; \frac{1}{K}\hat{f} - P_1^\perp y,$$

using $y^\top \mathbf{1} = 1$ (so $(I - \frac{1}{K}\mathbf{1}\mathbf{1}^\top) y = P_1^\perp y$). The standard differential for cross-entropy is

$$d\phi = (\mathrm{softmax}(f_L) - y)^\top df_L.$$

Substitute the approximation above to obtain

$$d\phi \;\approx\; \Big(\tfrac{1}{K}\hat{f} - P_1^{\perp} y\Big)^{\top} df_L = \Big(-P_1^{\perp} y + \tfrac{1}{K} P_1^{\perp} f_L\Big)^{\top} df_L,$$

where we used $\hat{f} = P_1^{\perp} f_L$. By definition of the tail Jacobian $\mathcal{J}_l$ (Lemma 9), $df_L = \mathcal{J}_l\, df_l$, and for the (bias-free) linear tensor layer $f_l = \mathcal{W}_l \times_1 f_{l-1}$ we have $df_l = d\mathcal{W}_l \times_1 f_{l-1}$. Therefore

$$d\phi \;\approx\; \Big(-P_1^{\perp} y + \tfrac{1}{K} P_1^{\perp} f_L\Big)^{\top} \mathcal{J}_l\,(d\mathcal{W}_l \times_1 f_{l-1}) = \big(\mathcal{J}_l^{\top}(-P_1^{\perp} y + \tfrac{1}{K} P_1^{\perp} f_L)\big)^{\top} (d\mathcal{W}_l \times_1 f_{l-1}).$$

Applying the trace/inner-product identity from Sec. **??** yields

$$d\phi \;\approx\; \Big\langle \big(\mathcal{J}_l^{\top}(-P_1^{\perp} y + \gamma K^{-1} P_1^{\perp} f_L)\big) \otimes f_{l-1},\; d\mathcal{W}_l \Big\rangle,$$

which implies the stated approximate gradient. Finally, substituting the linearization $f_L \approx \mathcal{J}_l f_l$ and $f_l = \mathcal{W}_l \times_1 f_{l-1}$ gives $\mathcal{J}_l^{\top} P_1^{\perp} f_L \approx \mathcal{J}_l^{\top} P_1^{\perp} \mathcal{J}_l (\mathcal{W}_l \times_1 f_{l-1})$, yielding the second displayed form.

O Theoretical Results of Tensor low-rank for Neural Operators

Following (Zhao et al., 2024), we analyze a simplified setting where the gradient admits a multilinear parametric form. For FNO layers, the tensors $\mathcal{A}_i, \mathcal{B}_i, \mathcal{C}_i$ can be interpreted as contributions from (i) feature mixing, (ii) Fourier-mode mixing, and (iii) pointwise nonlinearities. We state the result mode-wise, using the stable-rank definitions in Sec. M.

Lemma 11 (Tensor gradient becomes low-rank during training) Assume the gradient tensor satisfies the multilinear form

$$\mathcal{G}_t \;=\; \frac{1}{N}\sum_{i=1}^{N} \Big(\mathcal{A}_i \;-\; \mathcal{W}_t \times_1 B_i^{(1)} \times_2 \cdots \times_d B_i^{(d)}\Big),$$

where $\mathcal{A}_i$ and $\{B_i^{(k)}\}_{k=1}^{d}$ are constant for all $t \geq t_0$ and each $B_i^{(k)} \in \mathbb{R}^{I_k \times I_k}$ is PSD. Consider SGD updates $\mathcal{W}_t = \mathcal{W}_{t-1} + \eta \mathcal{G}_{t-1}$. Fix a mode k and define the PSD operator $\mathcal{S}_k$ on $\mathrm{vec}(X) \in \mathbb{R}^{I_k J_k}$ by

$$\mathrm{vec}\big(\mathcal{S}_k(X)\big) := \Big(\frac{1}{N}\sum_{i=1}^{N}\Big(\bigotimes_{m\neq k} B_i^{(m)}\Big) \otimes B_i^{(k)}\Big)\mathrm{vec}(X), \qquad X \in \mathbb{R}^{I_k \times J_k},$$

where $J_k = \prod_{m\neq k} I_m$. Let $0 \leq \lambda_1^{(k)} < \lambda_2^{(k)} \leq \cdots$ be the eigenvalues of $\mathcal{S}_k$ and choose $0 < \eta < 2/\lambda_{\max}^{(k)}$. Let $\mathcal{V}_1^{(k)}$ be the eigenspace corresponding to $\lambda_1^{(k)}$, and let $(\mathcal{G}_{t_0}^{\|(k)})_{(k)}$ be the orthogonal projection of $(\mathcal{G}_{t_0})_{(k)}$ onto $\mathcal{V}_1^{(k)}$. Then

$$\mathrm{sr}_k(\mathcal{G}_t) \leq \mathrm{sr}_k(\mathcal{G}_{t_0}^{\|(k)}) + \Big(\frac{1-\eta\lambda_2^{(k)}}{1-\eta\lambda_1^{(k)}}\Big)^{2(t-t_0)} \frac{\|(\mathcal{G}_{t_0})_{(k)} - (\mathcal{G}_{t_0}^{\|(k)})_{(k)}\|_F^2}{\|(\mathcal{G}_{t_0}^{\|(k)})_{(k)}\|_2^2},$$

and therefore $\mathrm{msr}(\mathcal{G}_t) = \min_k \mathrm{sr}_k(\mathcal{G}_t)$ satisfies the corresponding $\min_k$ bound.

Proof 12 For $t \geq t_0$, substitute $\mathcal{W}_t = \mathcal{W}_{t-1} + \eta\mathcal{G}_{t-1}$ into the multilinear form and use linearity of mode products:

$$\mathcal{G}_t = \frac{1}{N}\sum_{i=1}^{N}\Big(\mathcal{A}_i - \big(\mathcal{W}_{t-1} + \eta\mathcal{G}_{t-1}\big) \times_1 B_i^{(1)} \times_2 \cdots \times_d B_i^{(d)}\Big) = \mathcal{G}_{t-1} - \eta\,\frac{1}{N}\sum_{i=1}^{N}\Big(\mathcal{G}_{t-1} \times_1 B_i^{(1)} \times_2 \cdots \times_d B_i^{(d)}\Big).$$

Fix mode k and unfold. Using the standard unfolding identity for multilinear products,

$$\big(\mathcal{X} \times_1 B^{(1)} \times_2 \cdots \times_d B^{(d)}\big)_{(k)} = B^{(k)}\, \mathcal{X}_{(k)} \Big(\bigotimes_{m\neq k} B^{(m)}\Big)^{\top},$$

we obtain the recursion

$$(\mathcal{G}_t)_{(k)} = (\mathcal{G}_{t-1})_{(k)} - \eta\, \mathcal{S}_k\big((\mathcal{G}_{t-1})_{(k)}\big), \qquad \text{hence} \qquad (\mathcal{G}_t)_{(k)} = (I - \eta \mathcal{S}_k)^{t-t_0} (\mathcal{G}_{t_0})_{(k)}.$$

Decompose $(\mathcal{G}_{t_0})_{(k)} = (\mathcal{G}_{t_0}^{\|(k)})_{(k)} + (\mathcal{G}_{t_0}^{\perp(k)})_{(k)}$ into components parallel/perpendicular to $\mathcal{V}_1^{(k)}$. Since $\mathcal{S}_k$ is PSD and diagonalizable, $(I-\eta\mathcal{S}_k)^{t-t_0}$ scales the component in the eigenspace of eigenvalue λ by $(1-\eta\lambda)^{t-t_0}$. Using $\lambda_2^{(k)}$ as the smallest eigenvalue outside $\mathcal{V}_1^{(k)}$ gives

$$\|(\mathcal{G}_t)_{(k)}\|_F^2 \le (1-\eta\lambda_1^{(k)})^{2(t-t_0)} \|(\mathcal{G}_{t_0}^{\|(k)})_{(k)}\|_F^2 + (1-\eta\lambda_2^{(k)})^{2(t-t_0)} \|(\mathcal{G}_{t_0}^{\perp(k)})_{(k)}\|_F^2.$$

On the minimal eigenspace, $(\mathcal{G}_t)_{(k)}$ contains the factor $(1-\eta\lambda_1^{(k)})^{t-t_0}(\mathcal{G}_{t_0}^{\|(k)})_{(k)}$, so

$$\|(\mathcal{G}_t)_{(k)}\|_2^2 \;\ge\; (1-\eta\lambda_1^{(k)})^{2(t-t_0)} \|(\mathcal{G}_{t_0}^{\|(k)})_{(k)}\|_2^2.$$

Dividing the Frobenius bound by this spectral-norm lower bound yields the stated stable-rank inequality, using $\|(\mathcal{G}_{t_0}^{\perp(k)})_{(k)}\|_F^2 = \|(\mathcal{G}_{t_0})_{(k)} - (\mathcal{G}_{t_0}^{\|(k)})_{(k)}\|_F^2$. Taking $\min_k$ yields the multilinear stable-rank bound.

Remark 6 (FNO interpretation) Channel modes ($k = 1, 2$) and Fourier modes ($k = 3, 4$) can exhibit different eigengaps and decay rates: Fourier directions often have stronger spectral decay, while channel directions may preserve higher effective rank to retain feature capacity. The bound above captures this mode-wise behavior through $(\lambda_1^{(k)}, \lambda_2^{(k)})$ and $\mathrm{sr}_k(\cdot)$.

Corollary 1 (A sufficient condition for eventual mode-wise low rank) Under the assumptions of Lemma 11, fix a mode k and suppose $\lambda_1^{(k)} = 0$ and the minimal eigenspace $\mathcal{V}_1^{(k)}$ has dimension $r_k^\star$ (equivalently, the projector $P_{\mathcal{V}_1^{(k)}}$ has rank $r_k^\star$). Then

$$\limsup_{t\to\infty} \mathrm{sr}_k(\mathcal{G}_t) \;\le\; \mathrm{sr}_k(\mathcal{G}_{t_0}^{\|(k)}) \;\le\; r_k^\star.$$

Proof 13 If $\lambda_1^{(k)} = 0$ and $\lambda_2^{(k)} > 0$, then $(1-\eta\lambda_2^{(k)})^{2(t-t_0)} \to 0$ while $(1-\eta\lambda_1^{(k)})^{2(t-t_0)} = 1$. Lemma 11 then yields $\limsup_{t\to\infty} \mathrm{sr}_k(\mathcal{G}_t) \le \mathrm{sr}_k(\mathcal{G}_{t_0}^{\|(k)})$. Finally, $\mathrm{sr}_k(\cdot) \le \mathrm{rank}((\cdot)_{(k)})$ and the projection lies in $\mathcal{V}_1^{(k)}$, so $\mathrm{rank}((\mathcal{G}_{t_0}^{\|(k)})_{(k)}) \le r_k^\star$.

Corollary 2 (Rank-1 limit under a decomposable minimal eigenvector) Under the assumptions of Lemma 11, fix a mode k and assume the minimal eigenspace $\mathcal{V}_1^{(k)}$ is 1-dimensional and spanned by a decomposable vector in the unfolded space, i.e. there exist $y_k \in \mathbb{R}^{I_k}$ and $z_k \in \mathbb{R}^{J_k}$ such that the (unit) eigenvector is $\mathrm{vec}(y_k z_k^\top)$. Then $(\mathcal{G}_{t_0}^{\|(k)})_{(k)}$ has rank 1, hence $\mathrm{sr}_k(\mathcal{G}_{t_0}^{\|(k)}) = 1$, and Lemma 11 implies $\mathrm{sr}_k(\mathcal{G}_t) \to 1$ as $t \to \infty$.

Proof 14 If $\mathcal{V}_1^{(k)} = \mathrm{span}\{\mathrm{vec}(y_k z_k^\top)\}$, then the projection $(\mathcal{G}_{t_0}^{\|(k)})_{(k)}$ is a scalar multiple of $y_k z_k^\top$, which is rank 1, so $\mathrm{sr}_k(\mathcal{G}_{t_0}^{\|(k)}) = \|M\|_F^2/\|M\|_2^2 = 1$ for rank-1 matrices. The conclusion follows from Lemma 11 with $\lambda_2^{(k)} > \lambda_1^{(k)}$.

Theorem 2 (Tensor low-rank convergence with fixed projections) Let $\{P_k \in \mathbb{R}^{I_k \times r_k}\}_{k=1}^d$ have orthonormal columns ($P_k^\top P_k = I$). Define the projection and reconstruction

$$\mathsf{P}(\mathcal{G}) := \mathcal{G} \times_1 P_1^\top \times_2 \cdots \times_d P_d^\top, \qquad \mathsf{P}^\top(\mathcal{R}) := \mathcal{R} \times_1 P_1 \times_2 \cdots \times_d P_d,$$

and let $\mathcal{R}_t := \mathsf{P}(\mathcal{G}_t)$ and $\tilde{\mathcal{G}}_t := \mathsf{P}^\top(\mathcal{R}_t)$. Consider the projected SGD update $\mathcal{W}_t = \mathcal{W}_{t-1} + \eta\, \tilde{\mathcal{G}}_{t-1}$. Assume the gradient admits the multilinear form $\mathcal{G}_t = \frac{1}{N}\sum_{i=1}^N \Big(\mathcal{A}_i(\mathcal{W}_t) - \mathcal{W}_t \times_1 B_{it}^{(1)} \times_2 \cdots \times_d B_{it}^{(d)}\Big)$, where each $B_{it}^{(k)} \in \mathbb{R}^{I_k \times I_k}$ acts on mode k. Fix a mode $k \in \{1, \dots, d\}$ and define

$$Q_k := \bigotimes_{m \ne k} P_m, \qquad \hat{B}_{it}^{(k)} := P_k^\top B_{it}^{(k)} P_k, \qquad \hat{C}_{it}^{(k)} := Q_k^\top \Big(\bigotimes_{m \ne k} B_{it}^{(m)}\Big) Q_k.$$

Suppose: (i) $\mathcal{A}_i$ is mode-k Lipschitz with constant $L_A^{(k)}$, and each $B_{it}^{(k)}$ is mode-k Lipschitz with constant $L_B^{(k)}$; (ii) $\|\mathcal{W}_t\|_{(k)} \le D_k$; and (iii) $\hat{B}_{it}^{(k)} \succeq 0$ and $\hat{C}_{it}^{(k)} \succeq 0$ for all i, t. Define the projected curvature lower

bound $\kappa_t^{(k)} := \frac{1}{N}\sum_{i=1}^N \lambda_{\min}\big(\hat{B}_{it}^{(k)}\big)\,\lambda_{\min}\big(\hat{C}_{it}^{(k)}\big)$. If $\inf_{t,k}\kappa_t^{(k)} > L_A^{(k)} + L_B^{(k)} D_k^2$ and $\eta > 0$ is small enough that $q_k := 1 - \eta\big(\inf_t \kappa_t^{(k)} - L_A^{(k)} - L_B^{(k)} D_k^2\big) \in (0,1)$, then for each mode k we have the linear convergence bound

$$\|(\mathcal{R}_t)_{(k)}\|_F \;\le\; q_k^{\,t-t_0}\,\|(\mathcal{R}_{t_0})_{(k)}\|_F,$$

hence $\mathcal{R}_t \to 0$ and the projected dynamics converge in the fixed subspace defined by $\{P_k\}$.

Proof 15 Fix a mode k and work with mode-k unfoldings. For the multimode projection $\mathsf{P}(\mathcal{G}) = \mathcal{G} \times_1 P_1^\top \times_2 \cdots \times_d P_d^\top$, the standard unfolding identity gives

$$(\mathcal{R}_t)_{(k)} = P_k^\top (\mathcal{G}_t)_{(k)}\, Q_k, \qquad Q_k = \bigotimes_{m\neq k} P_m,$$

and $\|Q_k\|_2 = 1$. Using the multilinear form and the mode-k unfolding identity for multilinear products, $\big(\mathcal{W}_t \times_1 B_{it}^{(1)} \times_2 \cdots \times_d B_{it}^{(d)}\big)_{(k)} = B_{it}^{(k)}\,\mathcal{W}_{t(k)}\Big(\bigotimes_{m\neq k} B_{it}^{(m)}\Big)^\top$, we obtain $(\mathcal{R}_t)_{(k)} = \frac{1}{N}\sum_{i=1}^N \Big(P_k^\top(\mathcal{A}_i(\mathcal{W}_t))_{(k)} Q_k - \hat{B}_{it}^{(k)}\,\big(P_k^\top \mathcal{W}_{t(k)} Q_k\big)\,\hat{C}_{it}^{(k)\top}\Big)$. Insert $\mathcal{W}_t = \mathcal{W}_{t-1} + \eta\,\tilde{\mathcal{G}}_{t-1}$ and add/subtract the same expression evaluated at $\mathcal{W}_{t-1}$ to write the exact decomposition

$$(\mathcal{R}_t)_{(k)} \;=\; (\mathcal{R}_{t-1})_{(k)} \;-\; \eta\,\mathcal{H}_{t-1}^{(k)}\big((\mathcal{R}_{t-1})_{(k)}\big) \;+\; \mathcal{E}_t^{(k)},$$

where

$$\mathcal{H}_{t-1}^{(k)}(X) := \frac{1}{N}\sum_{i=1}^N \hat{B}_{i,t-1}^{(k)}\, X\, \hat{C}_{i,t-1}^{(k)\top}, \qquad X \in \mathbb{R}^{r_k \times \prod_{m\neq k} r_m},$$

and $\mathcal{E}_t^{(k)}$ collects all drift terms coming from evaluating $\mathcal{A}_i$ and $B_{it}^{(m)}$ at $\mathcal{W}_t$ instead of $\mathcal{W}_{t-1}$ (mode-k continuity).

We now bound the two contributions in Frobenius norm. First, since $\hat{B}_{i,t-1}^{(k)} \succeq 0$ and $\hat{C}_{i,t-1}^{(k)} \succeq 0$, for any X,

$$\langle X, \mathcal{H}_{t-1}^{(k)}(X)\rangle = \frac{1}{N}\sum_{i=1}^N \mathrm{tr}\big(X^\top \hat{B}_{i,t-1}^{(k)} X \hat{C}_{i,t-1}^{(k)\top}\big) \ge \kappa_{t-1}^{(k)}\|X\|_F^2,$$

which implies $\|X - \eta\mathcal{H}_{t-1}^{(k)}(X)\|_F \le (1-\eta\kappa_{t-1}^{(k)})\|X\|_F$ for η small enough. Applying this with $X = (\mathcal{R}_{t-1})_{(k)}$ gives

$$\|(\mathcal{R}_{t-1})_{(k)} - \eta\,\mathcal{H}_{t-1}^{(k)}((\mathcal{R}_{t-1})_{(k)})\|_F \le (1-\eta\kappa_{t-1}^{(k)})\|(\mathcal{R}_{t-1})_{(k)}\|_F.$$

Second, by the mode-k Lipschitz assumptions and $\|\mathcal{W}_t\|_{(k)} \le D_k$, the drift term is bounded proportionally to the step: $\|\mathcal{E}_t^{(k)}\|_F \le \eta\big(L_A^{(k)} + L_B^{(k)} D_k^2\big)\,\|(\mathcal{R}_{t-1})_{(k)}\|_F$. Combining the recursion with the two bounds yields

$$\|(\mathcal{R}_t)_{(k)}\|_F \le \Big(1 - \eta\kappa_{t-1}^{(k)} + \eta(L_A^{(k)} + L_B^{(k)} D_k^2)\Big)\|(\mathcal{R}_{t-1})_{(k)}\|_F.$$

Under the uniform gap condition $\inf_t \kappa_t^{(k)} > L_A^{(k)} + L_B^{(k)} D_k^2$ and the stated choice of η, the factor is a constant $q_k \in (0,1)$, so iterating gives $\|(\mathcal{R}_t)_{(k)}\|_F \le q_k^{\,t-t_0}\|(\mathcal{R}_{t_0})_{(k)}\|_F$, proving linear convergence. Since this holds for every k, we have $\mathcal{R}_t \to 0$.

Lemma 12 (Tensor low-rank vs. matricized GaLore: rank structure control) Let $\mathcal{G}_t \in \mathbb{R}^{N_1\times N_2\times N_3\times N_4}$ follow the multilinear parametric form

$$\mathcal{G}_t \;=\; \frac{1}{N}\sum_{i=1}^N \Big(\mathcal{A}_i(\mathcal{W}_t) \;-\; \mathcal{W}_t \times_1 B_{it}^{(1)} \times_2 \cdots \times_d B_{it}^{(d)}\Big).$$

and consider two low-rank projection schemes applied to optimizer updates:

(a) Matricized GaLore. Fix an unfolding index d and reshape $\mathcal{G}_t$ into a matrix $G_t^{(d)} \in \mathbb{R}^{N_d \times (N_1N_2N_3N_4/N_d)}$, then apply a rank-$r$ GaLore projection/update in this matrix space.

(b) Tensor low-rank. Use fixed mode-wise projections $\{P_k\}_{k=1}^4$ and apply the tensor projection $\mathsf{P}(\mathcal{G}_t) = \mathcal{G}_t \times_1 P_1^\top \times_2 P_2^\top \times_3 P_3^\top \times_4 P_4^\top$.

Then:

1. The GaLore analysis controls the stable-rank evolution only for the *chosen* matricization $G_t^{(d)}$ (i.e., it yields bounds on $\mathrm{sr}(G_t^{(d)})$ in the matrix space). In general, this does not imply any mode-wise stable-rank bound for the original tensor unfoldings $\mathrm{sr}_k(\mathcal{G}_t)$ for $k \neq d$.

2. Under the assumptions of Theorem 2 (fixed mode-wise projections and a positive projected curvature gap in each mode), tensor low-rank yields simultaneous mode-wise contraction of the projected quantities, hence a mode-wise decay guarantee that holds for all modes k through the same fixed projector family $\{P_k\}$.

Proof 16 For (1), matricized GaLore applies a rank-r projection to $G_t^{(d)}$ and its guarantee is stated in terms of the matrix stable rank $\mathrm{sr}(G_t^{(d)})$ for this particular unfolding (as in (Zhao et al., 2024)). The other mode-k unfoldings $(\mathcal{G}_t)_{(k)}$ for $k \neq d$ are not obtained from $G_t^{(d)}$ by left/right multiplication with orthogonal matrices; they correspond to different reshape operations that change which tensor modes are grouped as rows versus columns. Therefore, a bound on $\mathrm{sr}(G_t^{(d)}) = \|G_t^{(d)}\|_F^2 / \|G_t^{(d)}\|_2^2$ does not yield a bound on $\mathrm{sr}_k(\mathcal{G}_t) = \|(\mathcal{G}_t)_{(k)}\|_F^2 / \|(\mathcal{G}_t)_{(k)}\|_2^2$ for $k \neq d$ without additional assumptions linking the singular spectra across different unfoldings.

For (2), Theorem 2 gives, for each mode k, a one-step contraction bound on $\|(\mathsf{P}(\mathcal{G}_t))_{(k)}\|_F$ under the projected curvature gap condition. Because tensor low-rank assigns a separate projector P_k to each mode, the same fixed subspace is controlled in every mode simultaneously, yielding a multi-mode guarantee that has no analogue in a single-matricization method.

Remark 7 The practical implication is that matricized GaLore enforces low-rank structure in one chosen matrix view of the tensor, whereas tensor low-rank allocates rank per mode and preserves mode identity. This is the structural reason tensor low-rank can control all modes at once, while matricization targets only a single unfolding.

Dataset	Model	Architecture Details	Optimizer & Scheduler
Burgers	FNO	• 4 layers, 90 modes • 256 hidden channels, 256 projection channels • Skip Connections: 'linear' • Positional embedding: 'grid'	Adam with step LR $3e-4$, weight decay $2e-6$ 500 epochs, batch size 16. Trained with H_1 loss.
NS128	FNO	• 4 layers, 64 x 64 modes • 64 hidden channels, 256 projection channels • Skip: 'linear' • Use channel MLP: 1 • Channel MLP expansion: 0.5, dropout: 0	Complex Adam with step LR 3e-4, weight decay 1e-4, 500 epochs, batch size 8. Update decomposition frequency: 1000. Trained with H_1 loss.
NS1024 - max memory test	FNO	• 4 layers, 128 modes • 255 hidden channels, 256 projection channels • Skip: 'linear' • Channel MLP expansion: 0.5, dropout: 0	Complex Adam with step LR 5e-3, weight decay 1e-4, 100 epochs in total: batch size 8 for 50 iterations and resolution 256, then batch size 2 for resolution 1024. Update decomposition frequency: 500. Trained with $L2$ loss.
NS1024 Re = 10^5	FNO	• 4 layers, 128 modes • 128 hidden channels, 256 projection channels • Skip: 'linear' • Channel MLP expansion: 0.5, dropout: 0	Complex Adam with step LR 5e-3, weight decay 1e-4, 100 epochs in total: batch size 8 for 50 iterations and resolution 256, then batch size 2 for resolution 1024. Update decomposition frequency: 500. Trained with $L2$ loss.
Darcy Flow	FNO	• 4 layers, 64 modes • 128 hidden channels, 128 projection channels • Skip: 'linear'	Adam with step LR $1e-3$, weight decay $1e-4$, 250 epochs, batch size 2. Trained with L_2 loss.
EM Wave	Complex-FNO	• 8 layers, 128 modes • 128 hidden channels, 128 projection channels • Skip: 'linear' • Complex data: True • Complex activation function: True	Complex Adam with step LR 1e-4, weight decay 2e-6, batch size 32, 1000 epochs. Trained with H_1 loss.

Table 15: Detailed FNO Architecture Specifications for Different Datasets

Model	**Rank**	**Time/epoch(s)**	**Slowdown (%)**
Baseline	1.0	34.96	–
TensorGRaD	0.20 +0.05	40.74	12.67
GaLore	0.20	34.47	-1.40
GaLore	0.25	34.79	-0.48
GaLore	0.50	36.27	3.75
GaLore	0.75	37.50	7.26
TensorGRaD (40, 40, 40, 24)	0.20	36.53	5.98
Tensor low-rank (40, 40, 40, 24)	0.20	36.53	5.98
Tensor low-rank (48, 48, 48, 24)	0.25	38.30	10.08
Tensor low-rank (56, 56, 56, 24)	0.50	40.63	12.03
Tensor low-rank (64, 64, 56, 32)	0.75	44.93	19.84

Table 16: Comparison of model execution times, ranks, and relative slowdown

Table 17: Theoretical memory requirements for different methods

Method	Weight Parameters	Optimizer States (Adam)
Baseline	$N_1N_2N_3N_4$	$2N_1N_2N_3N_4$
Matrix GaLore (rollup dim 1)	$N_1N_2N_3N_4$	$2r(N_1 + N_2N_3N_4)$
Tensor low-rank (Tucker)	$N_1N_2N_3N_4$	$2r(N_1 + N_2 + N_3 + N_4)$