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

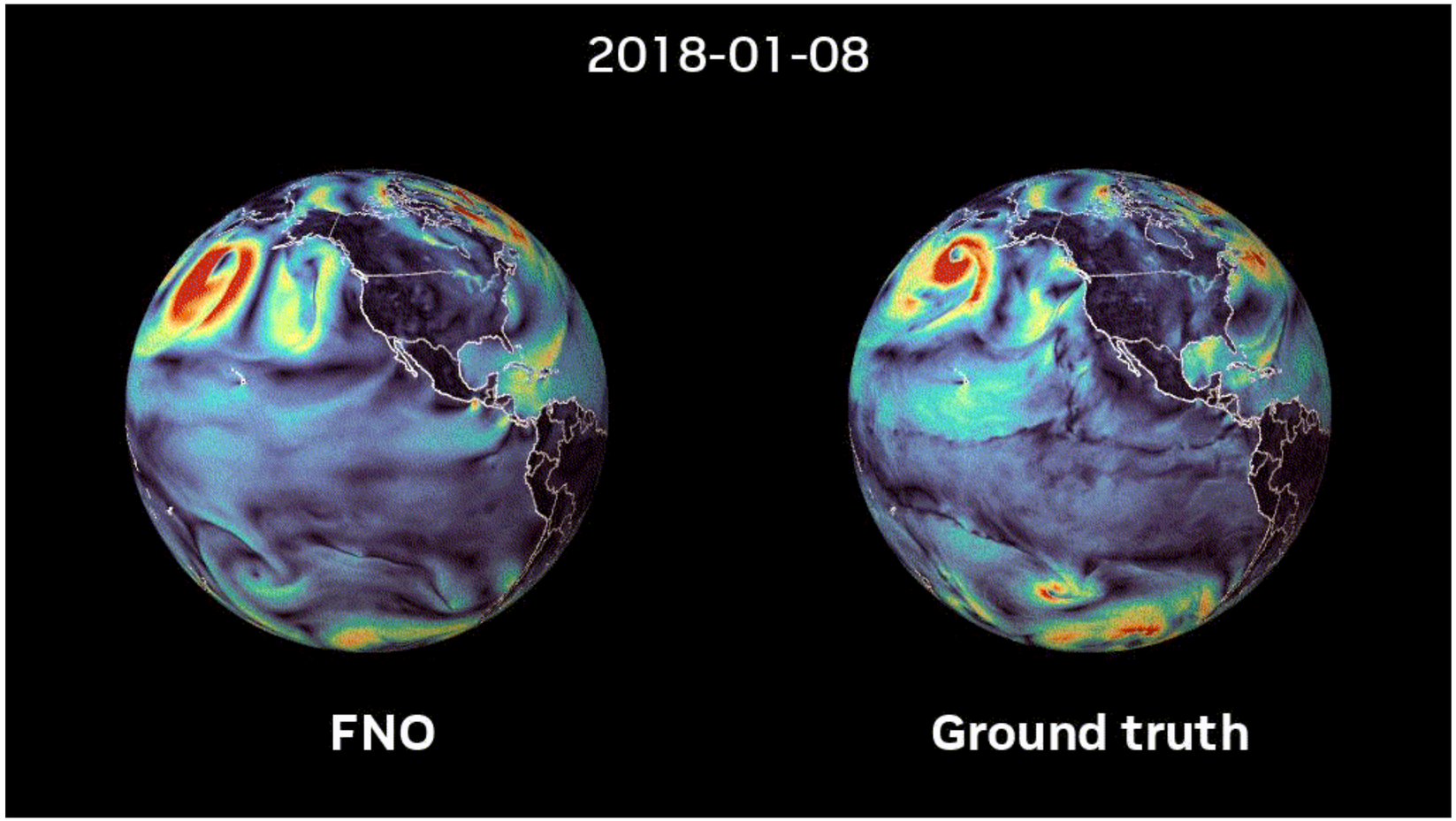
***Sebastian Loeschcke*, David Pitt, Robert Joseph George, Jiawei Zhao, Cheng Luo, Yuandong Tian, Jean Kossaifi , Anima Anandkumar**

UNIVERSITY OF COPENHAGEN

Caltech Meta NVIDIA

Many physical systems are modeled by PDEs

Goal: predict how a state evolves in space and time

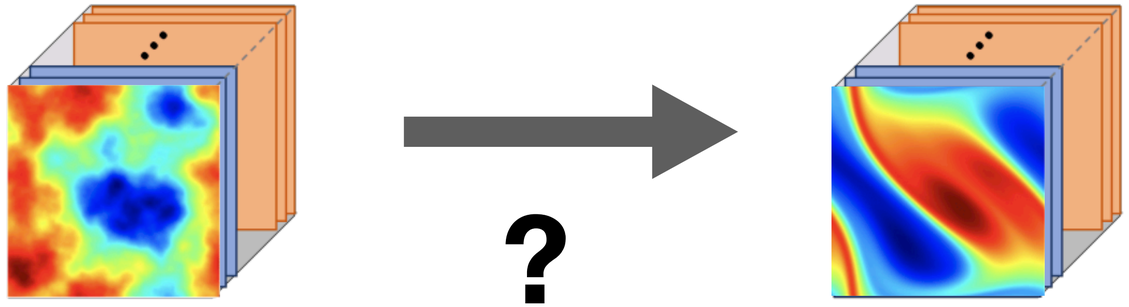


Examples: fluid dynamics, weather/climate, electromagnetics, heat transfer

What do we want to learn?

Initial condition

t=15



?

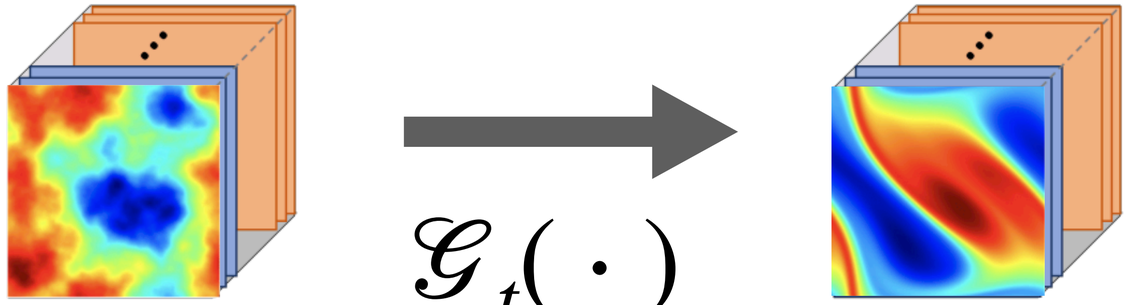
$u(\cdot,0)$

$u(\cdot,15)$

What do we want to learn?

Initial condition

t=15



$\mathcal{G}_t(\cdot)$

$u(\cdot,0)$

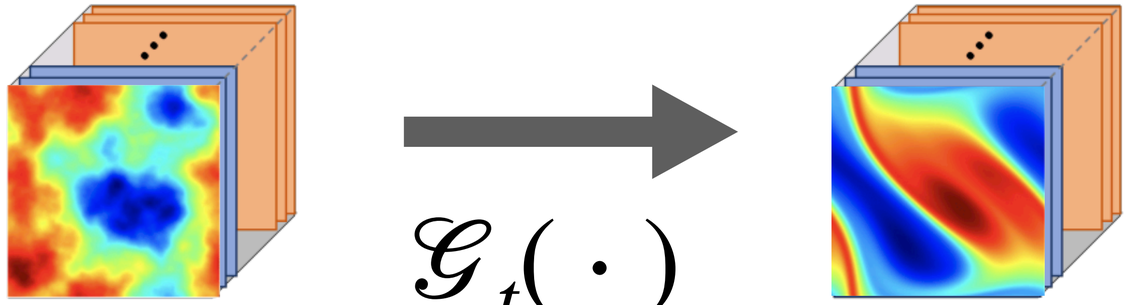
$u(\cdot,15)$

$$u(\cdot,15) = \mathcal{G}_{15}\big(u(\cdot,0)\big)$$

What do we want to learn?

Initial condition

t=15



$\mathcal{G}_t(\cdot)$

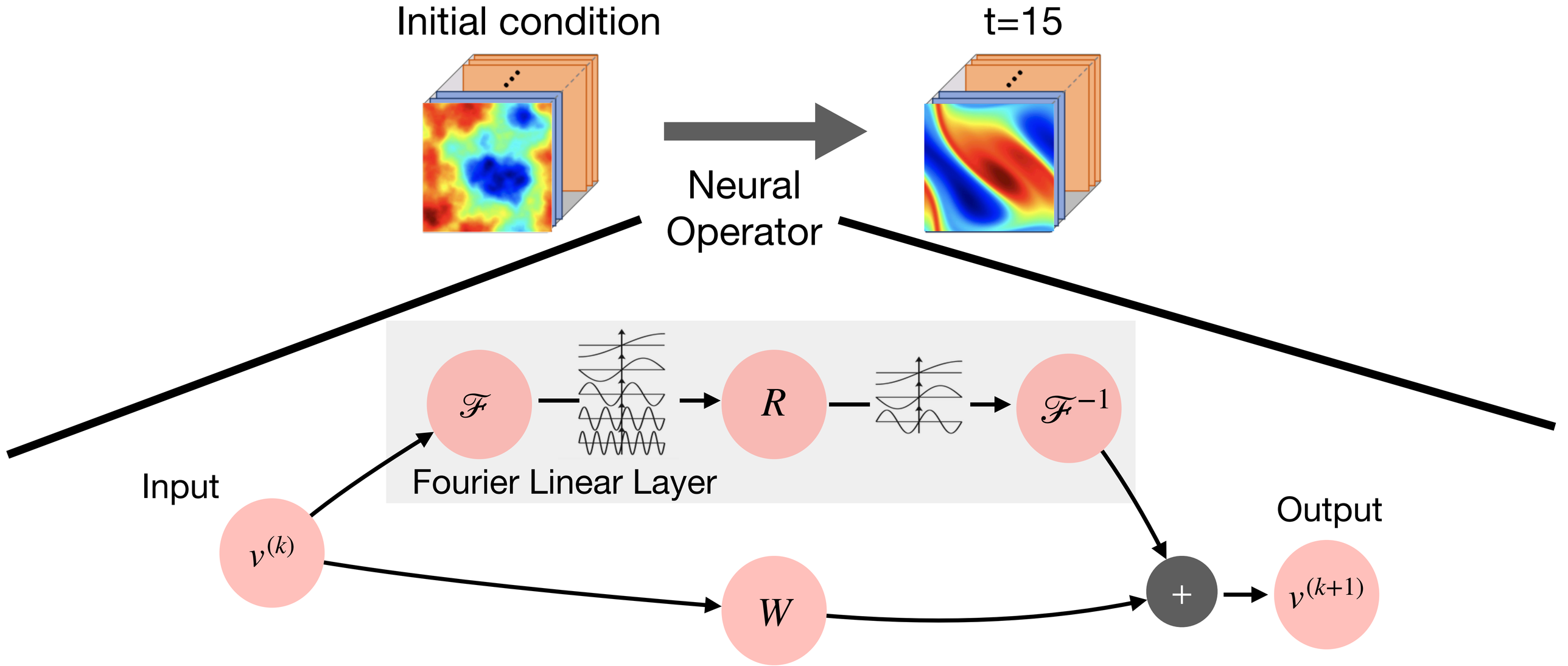
$u(\cdot,0)$

$u(\cdot,15)$

$$u(\cdot,15) = \mathcal{G}_{15}\big(u(\cdot,0)\big)$$

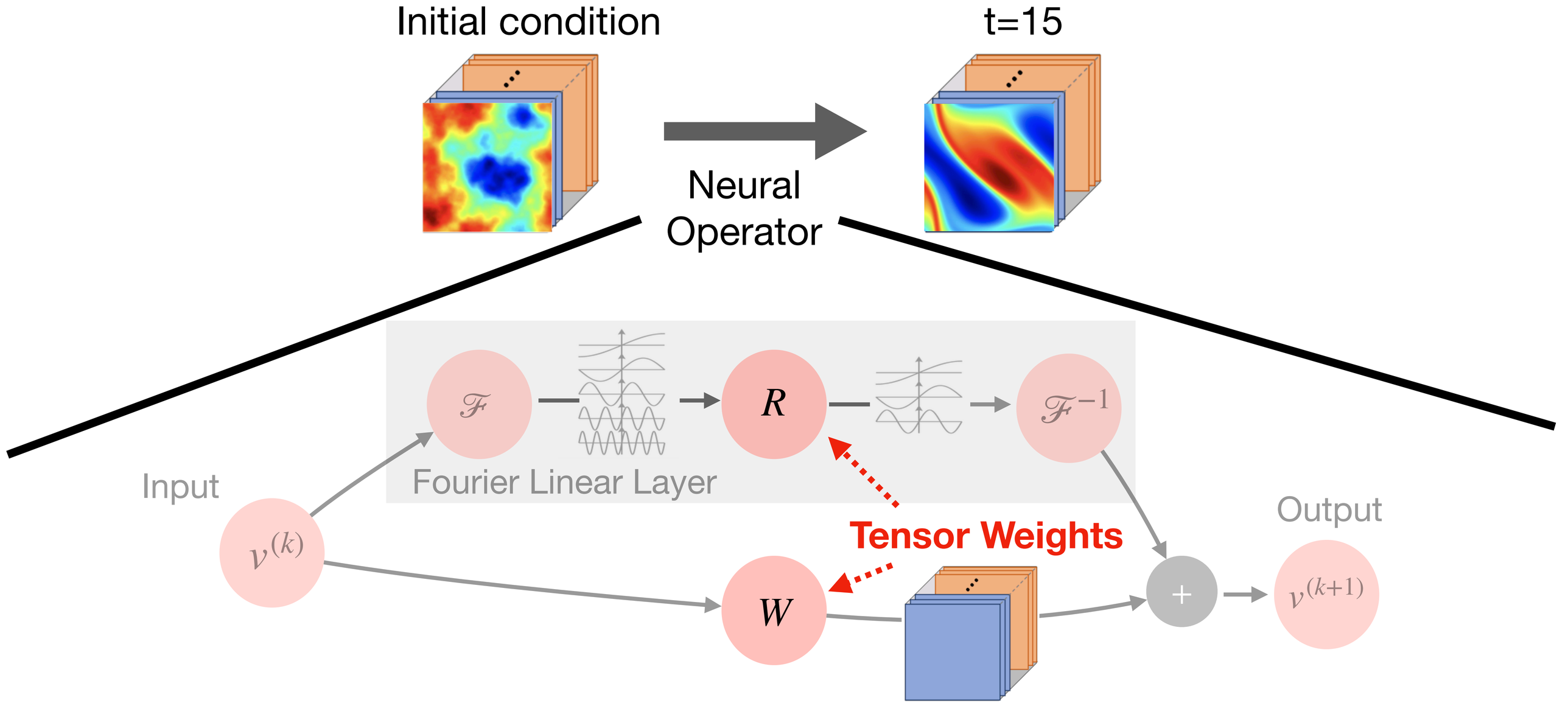
Learn “Neural Operator” from data

Fourier Neural Operator approximates solution operator of unknown PDE



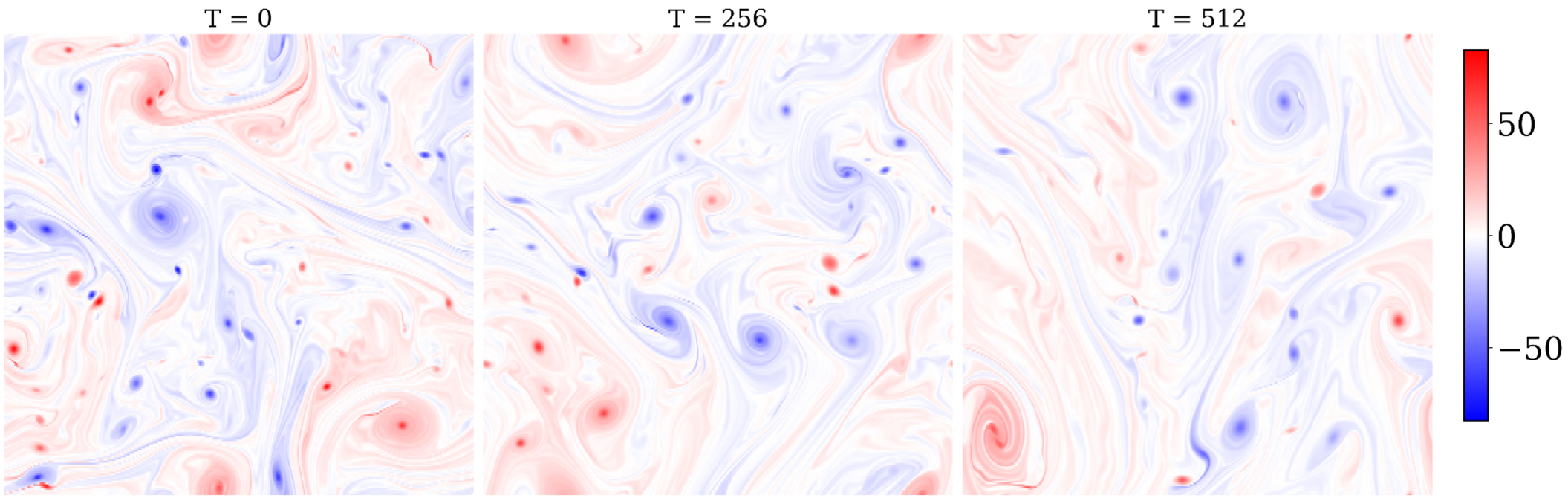
Zongyi et al. “Fourier Neural Operator for Parametric Partial Differential Equations”, 2021

Neural operators with tensor weights



Zongyi et al. “Fourier Neural Operator for Parametric Partial Differential Equations”, 2021

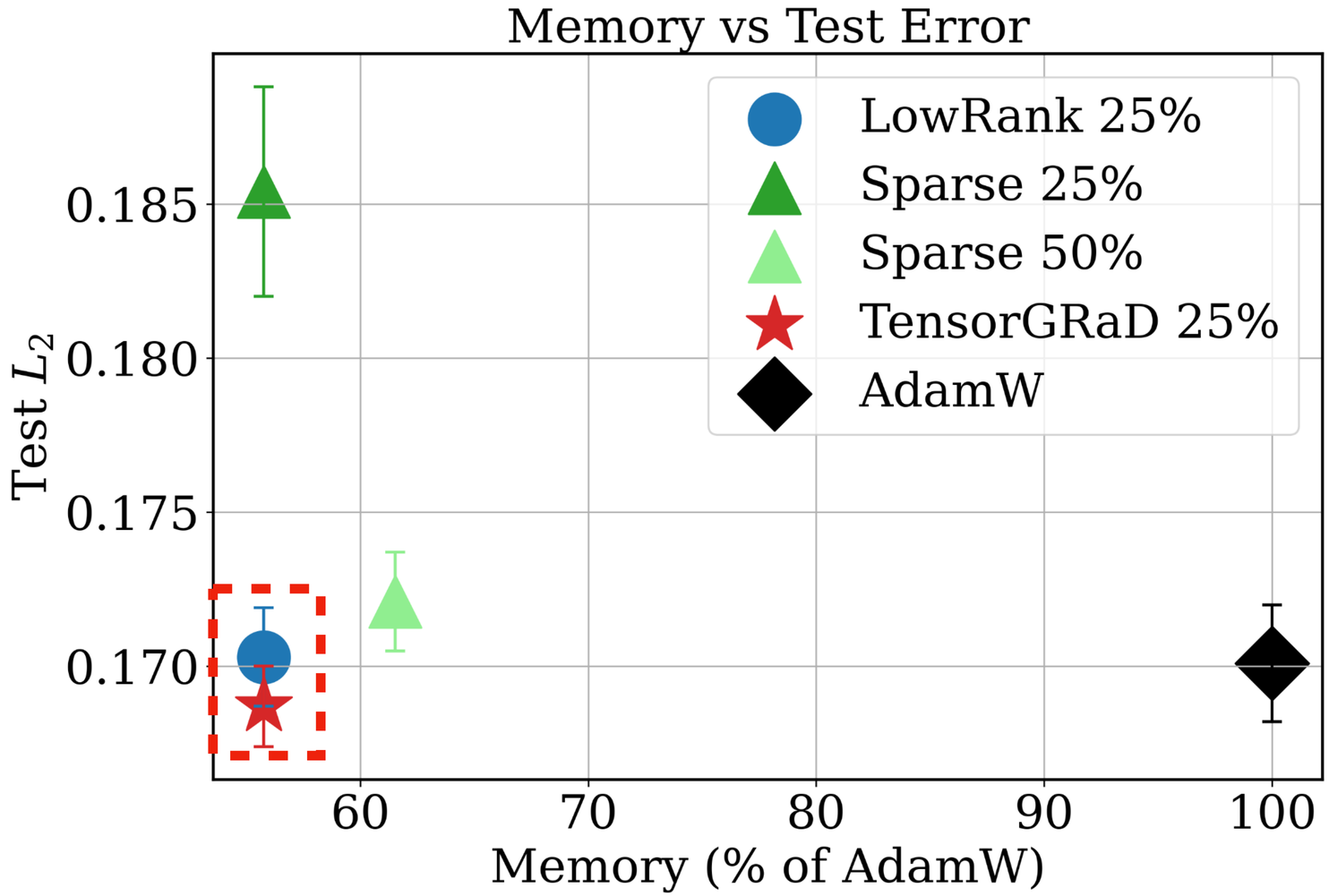
High-resolution PDEs require larger models



Navier-Stokes 1024×1024, Reynolds number 10^5

Memory becomes a bottleneck at large scale

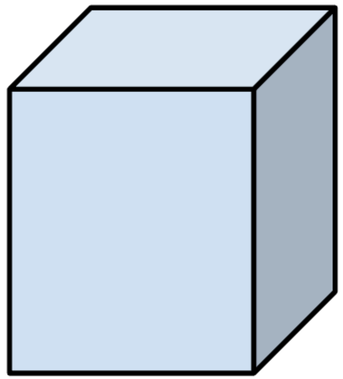
TensorGRaD matches baseline with roughly *half* the memory



Adam memory breakdown

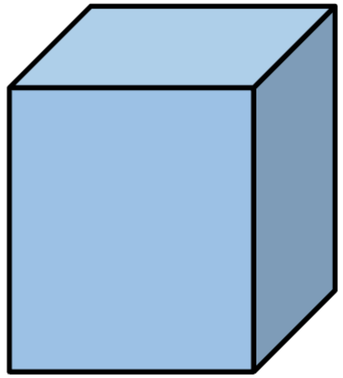
For each tensor-valued linear layer $T \in R^{M \times N \times O}$ Adam stores:

Weights



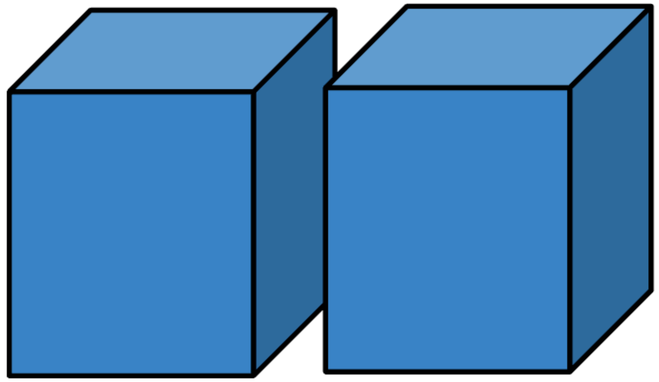
MNO

Gradients



MNO

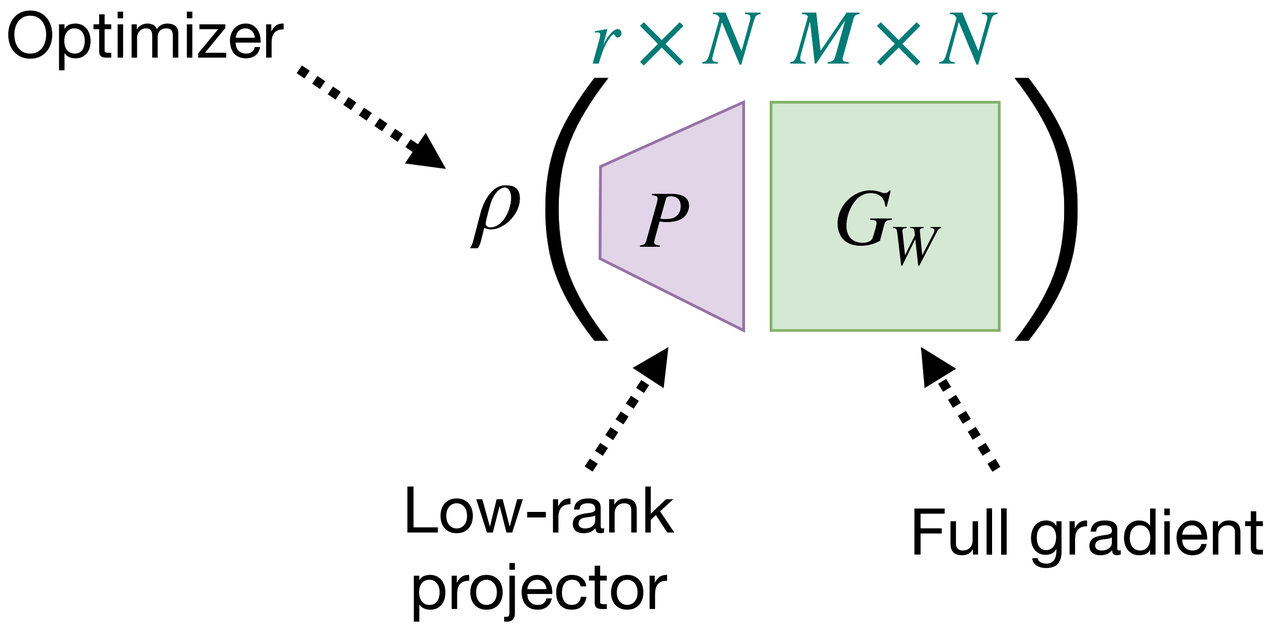
Optimizer States



$2MNO$

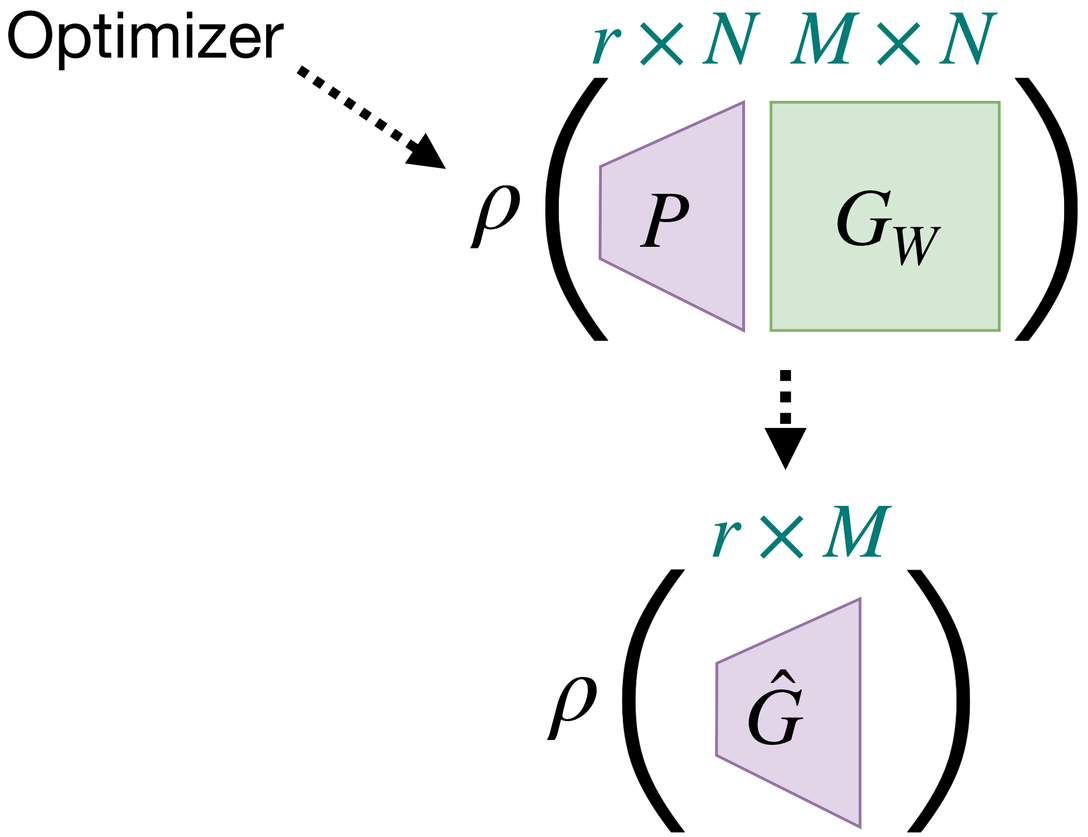
Total Memory: $4MNO$

Matrix low-rank methods reduce optimizer memory



Zhao et al., GaLore, ICML 2024

Matrix low-rank methods reduce optimizer memory

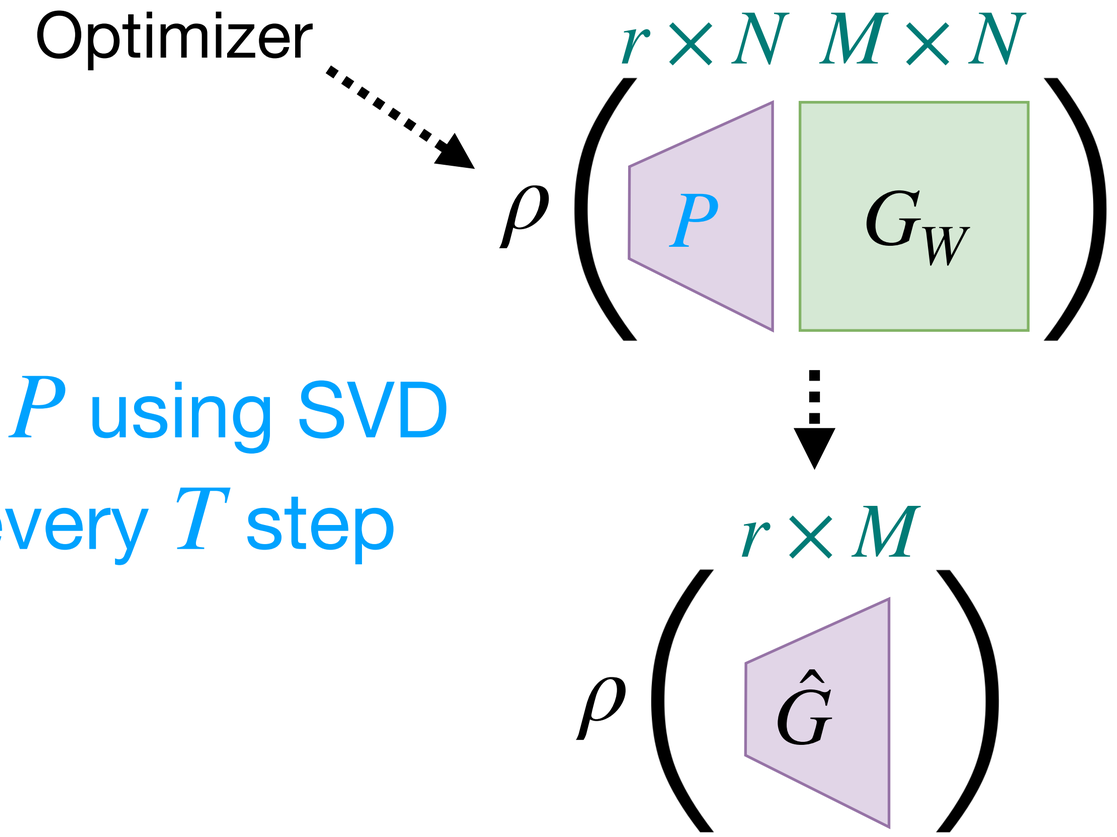


Saves memory for optimizer states

$2NM \rightarrow 2rM$

Zhao et al., GaLore, ICML 2024

Matrix low-rank methods reduce optimizer memory

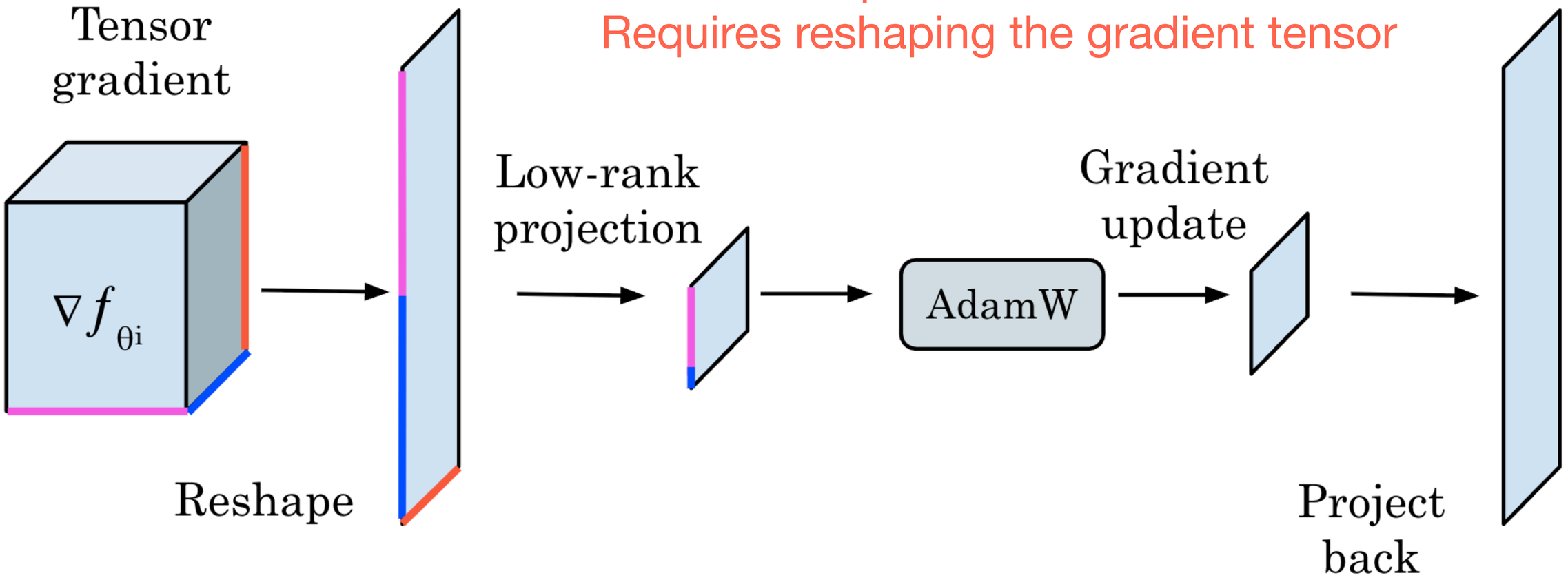


Update P using SVD of G every T step

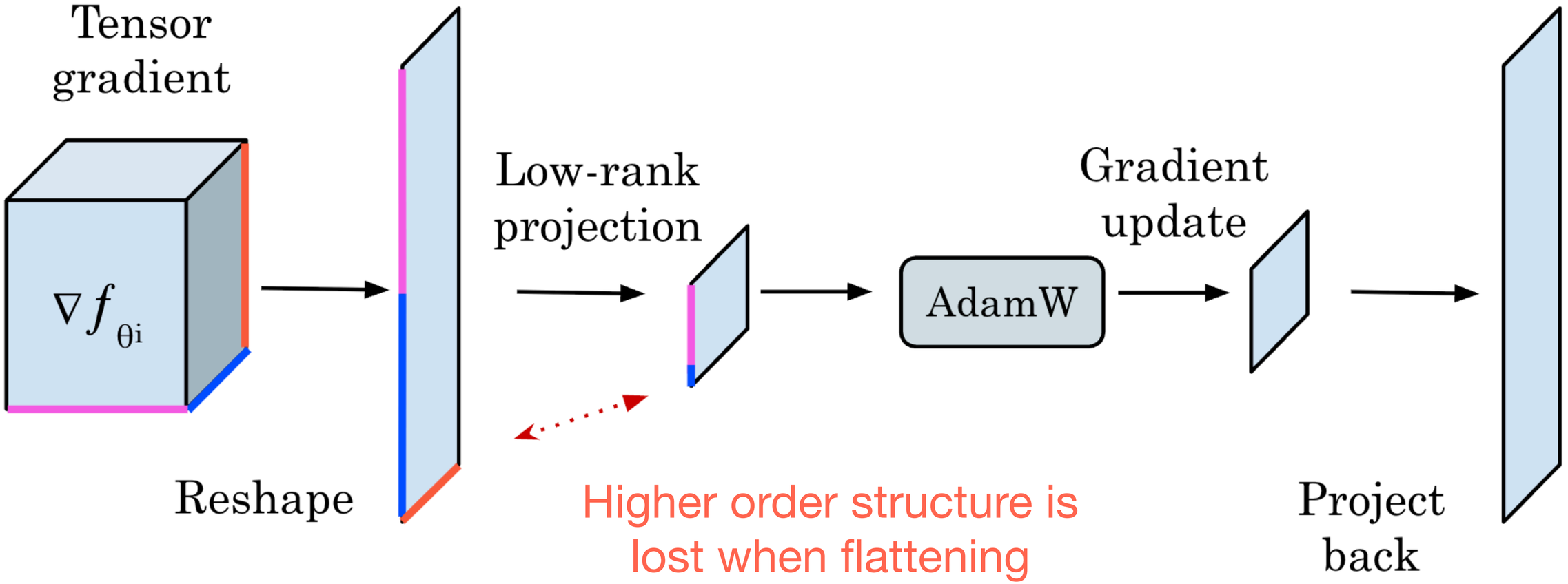
Saves memory for optimizer states $2NM \rightarrow 2rM$

Zhao et al., GaLore, ICML 2024

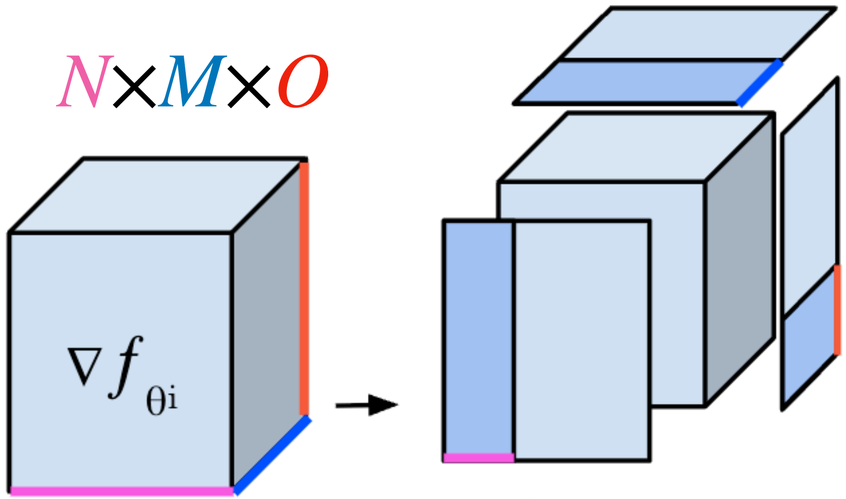
Can we apply low-rank methods to tensors?



Flattening loses higher-order tensor structure



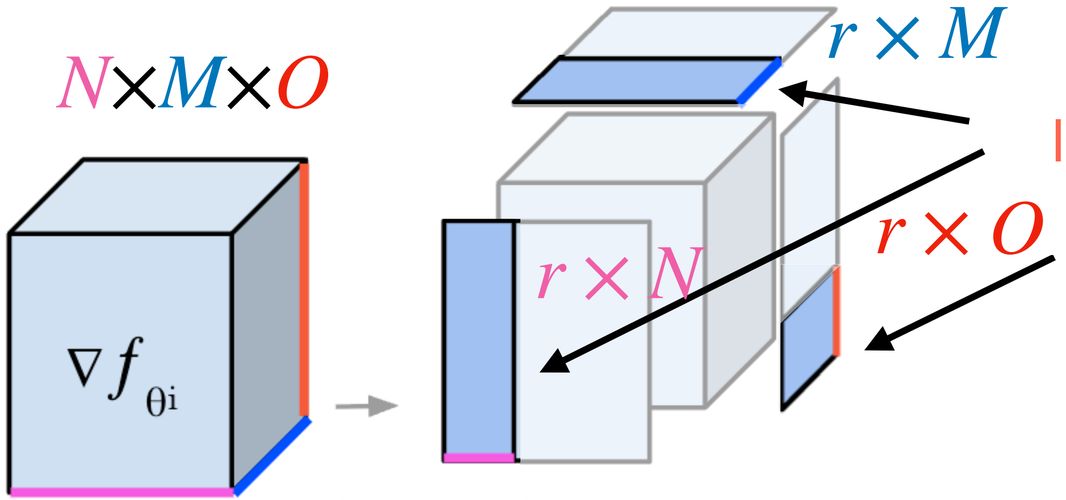
We propose to project tensor gradients using a low-rank Tucker factorization.



Tucker factorization of gradient

Tucker et al. “Some mathematical notes on three-mode factor analysis.” 1966.

We propose to project tensor gradients using a low-rank Tucker factorization.

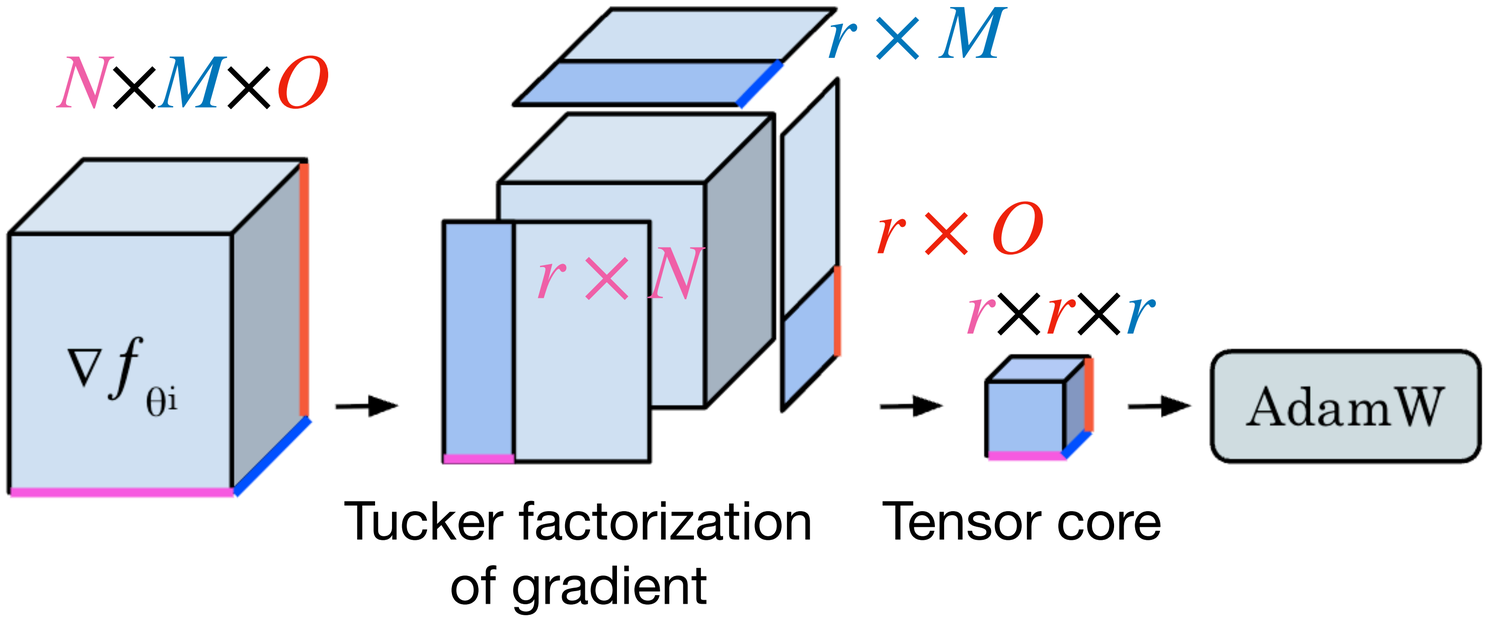


Only store a *single* low-rank matrix per dimension

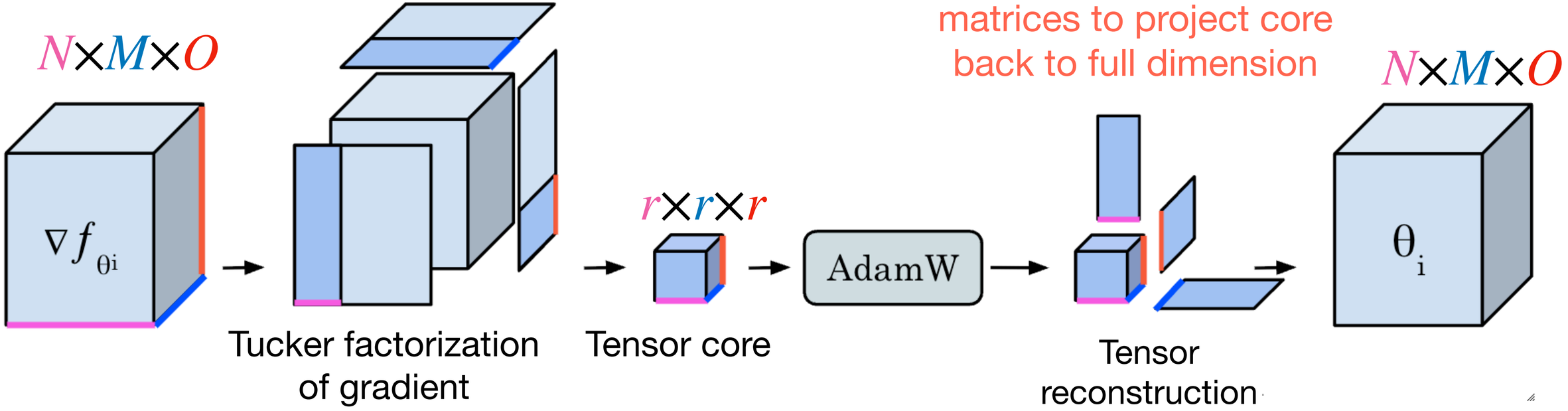
Tucker factorization of gradient

Tucker et al. “Some mathematical notes on three-mode factor analysis.” 1966.

Run Adam on the compressed tensor core



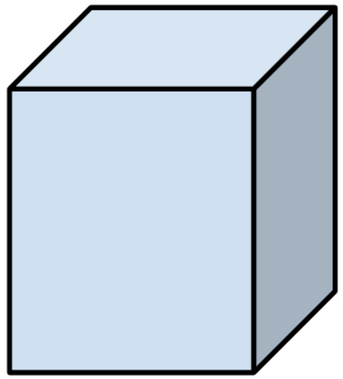
Reconstruct the update in the original tensor space



Tensor low-rank optimizer states reduce memory

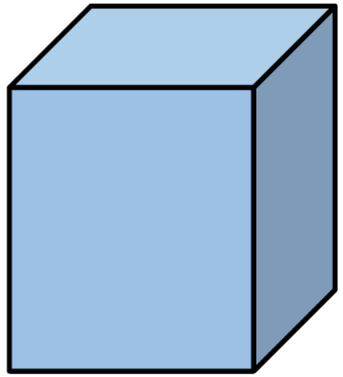
Adam

Weights



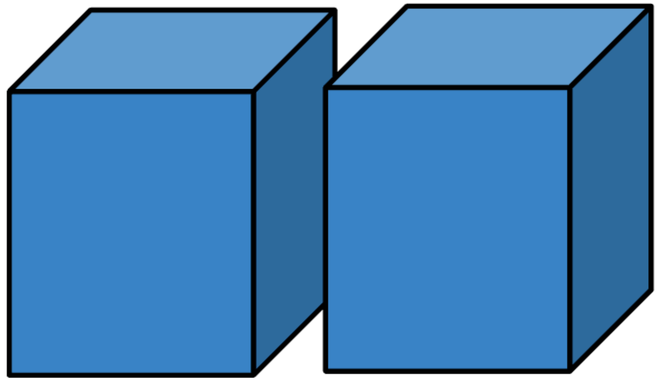
MNO

Gradients



MNO

Optimizer States

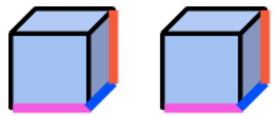


$2MNO$

Reduced

TensorGRaD

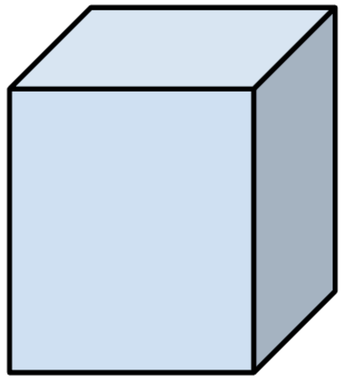
$2r^3$



Tensor low-rank optimizer states reduce memory

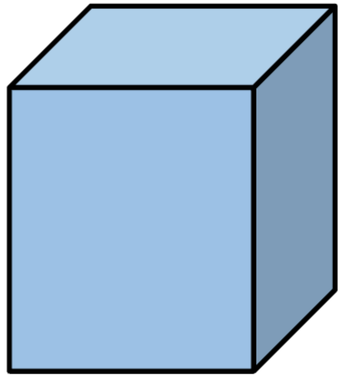
Adam

Weights



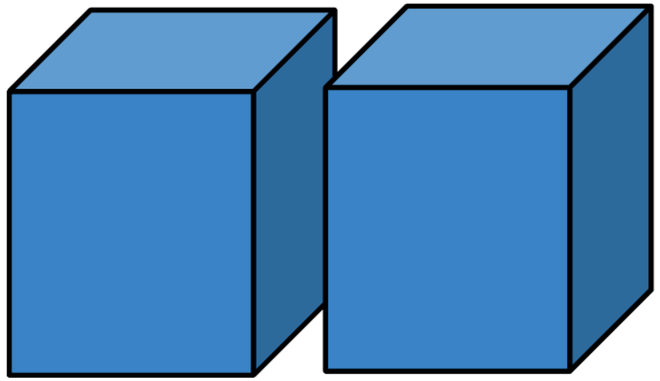
MNO

Gradients



MNO

Optimizer States

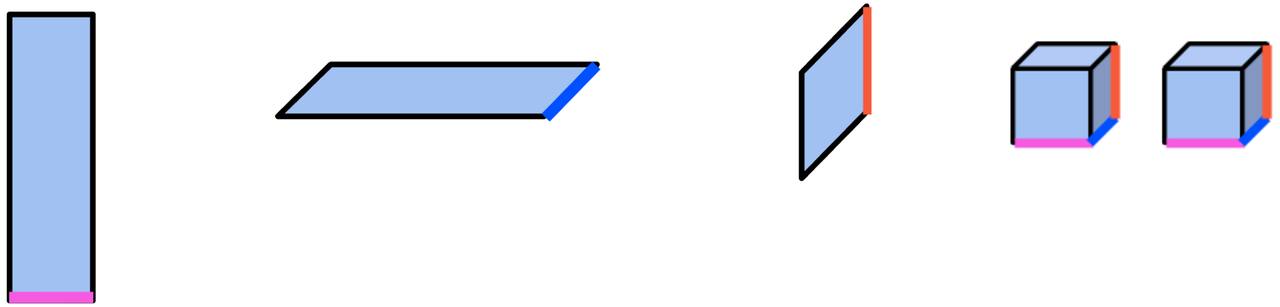


$2MNO$

Reduced

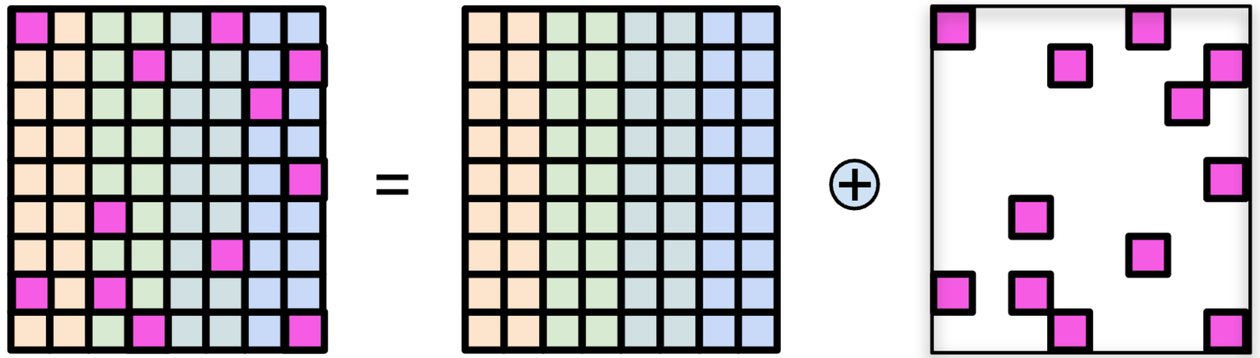
TensorGRaD

$$(N \times r) + (M \times r) + (O \times r) \quad + \quad 2r^3$$



Low-rank + Sparse

- **Robust PCA** [1] decompose matrix into a **low-rank + sparse** part

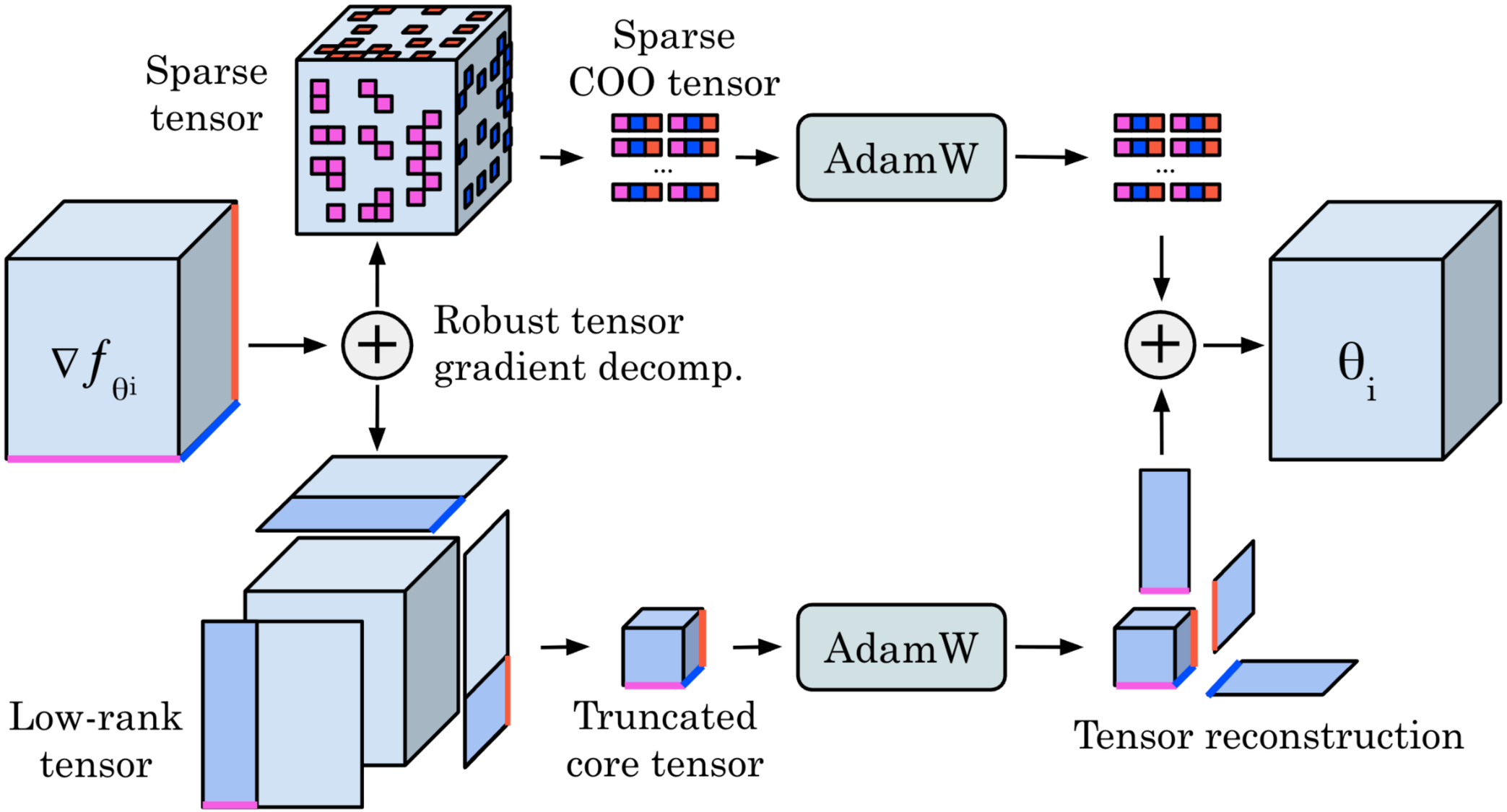


- **Reduces overfitting** and provably recovers the true subspace even under gross corruptions.

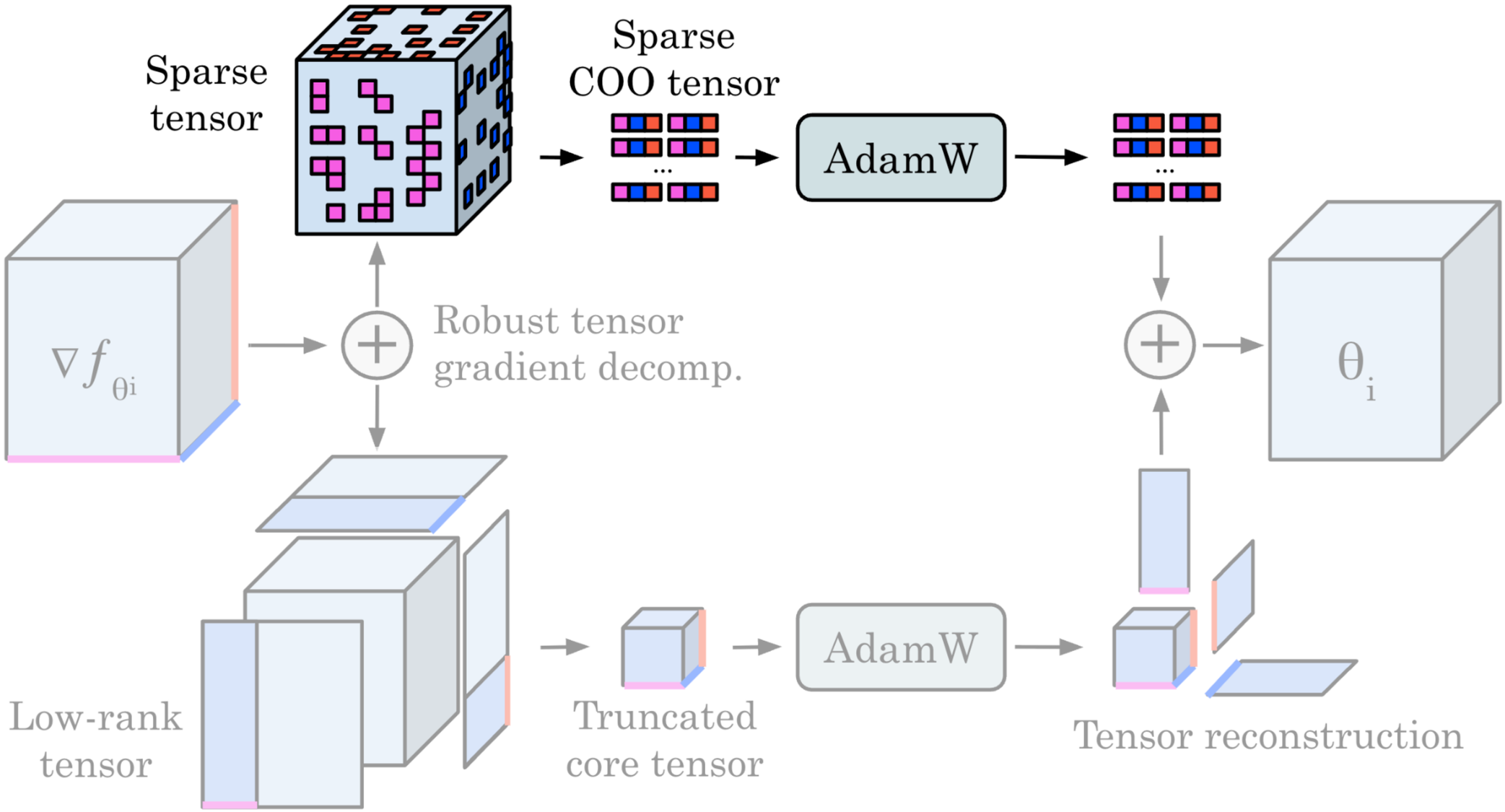
- **Robust tensor decomposition** [2] generalizes idea to tensors

[1] Candes et al., "Robust Principal Component Analysis?", 2009. [2] Gu et al., "Robust tensor decomposition with gross corruption”, 2014

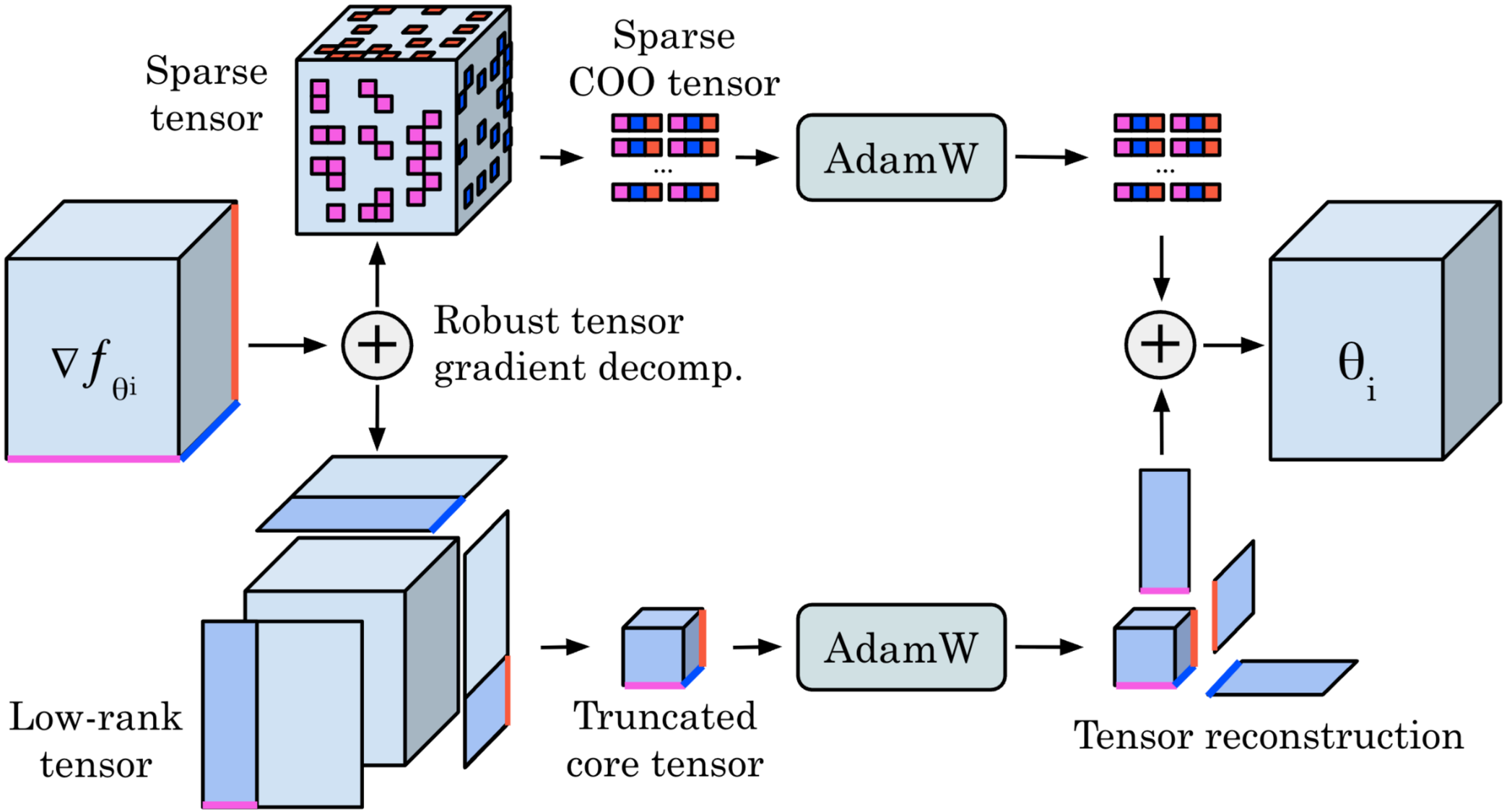
TensorGRaD: Low-rank + Sparse



TensorGRaD: Low-rank + Sparse

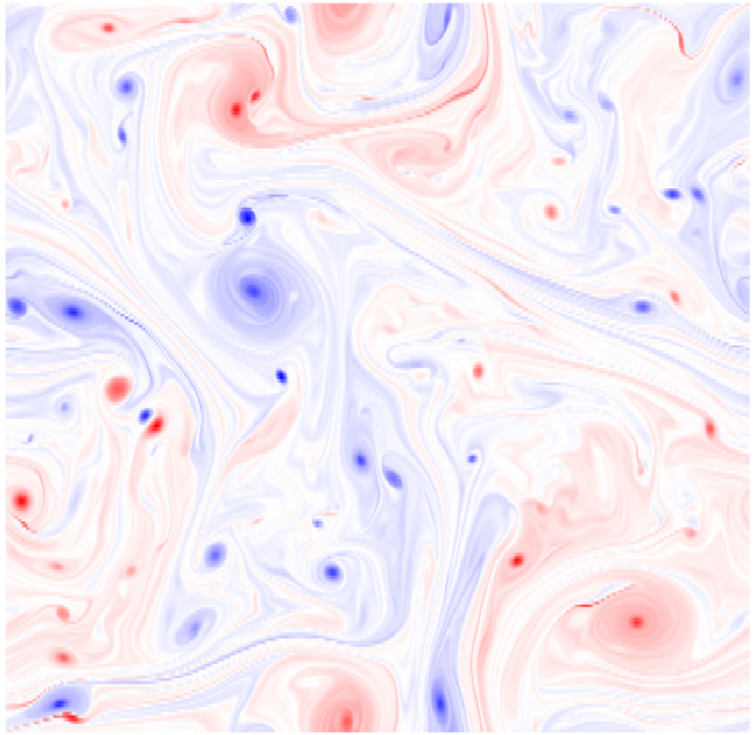


TensorGRaD: Low-rank + Sparse

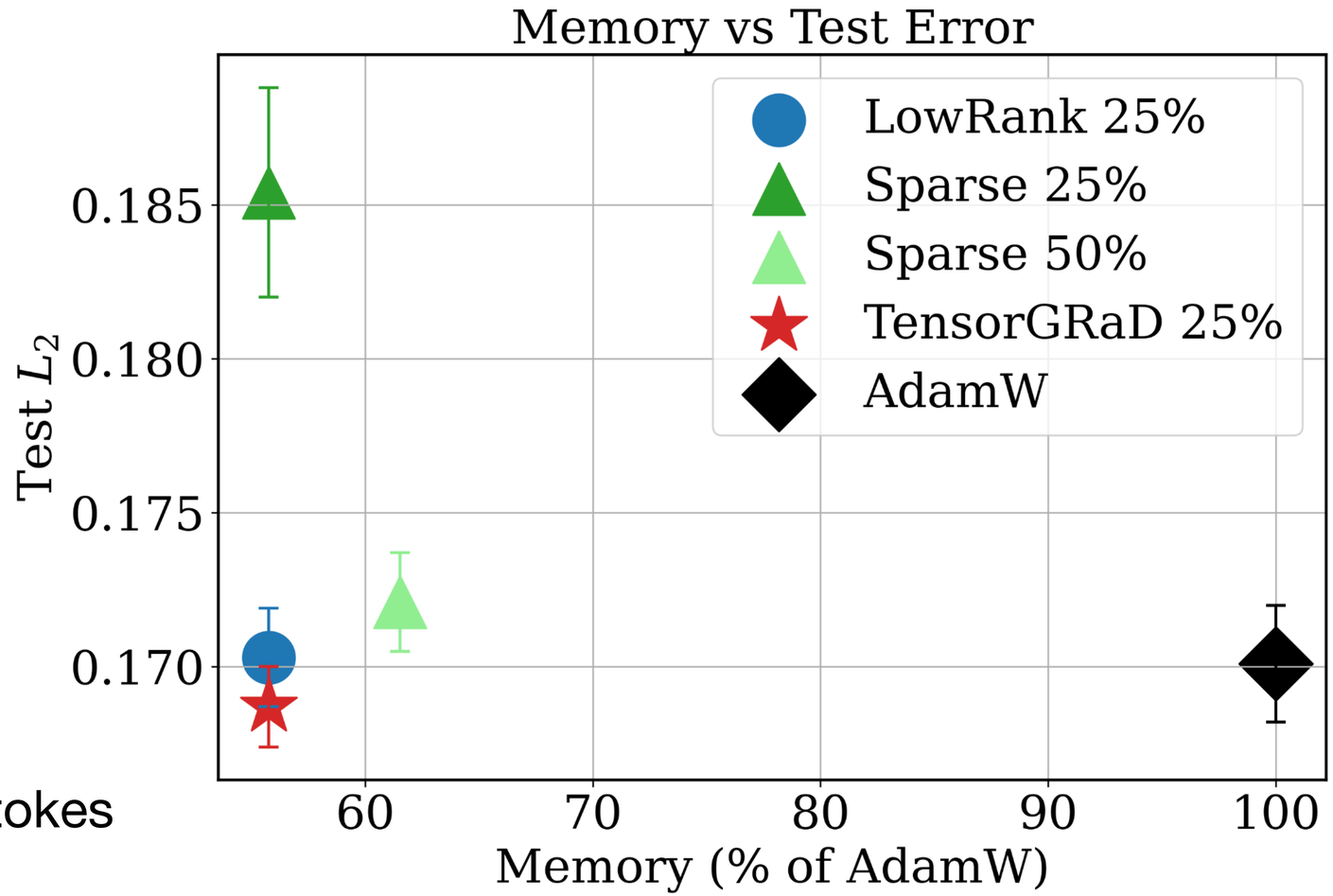


Results

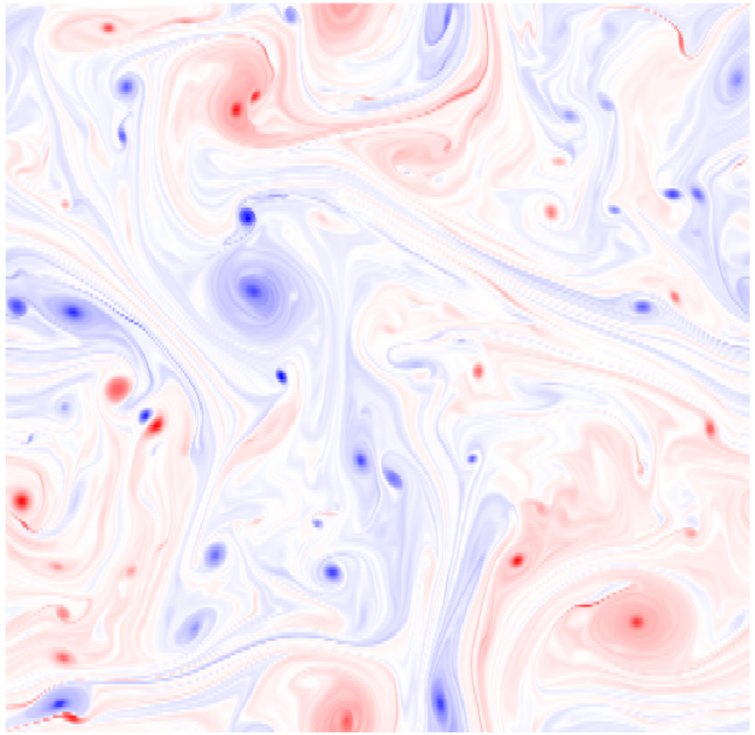
TensorGRaD matches Adam on turbulent Navier–Stokes with ~50% memory



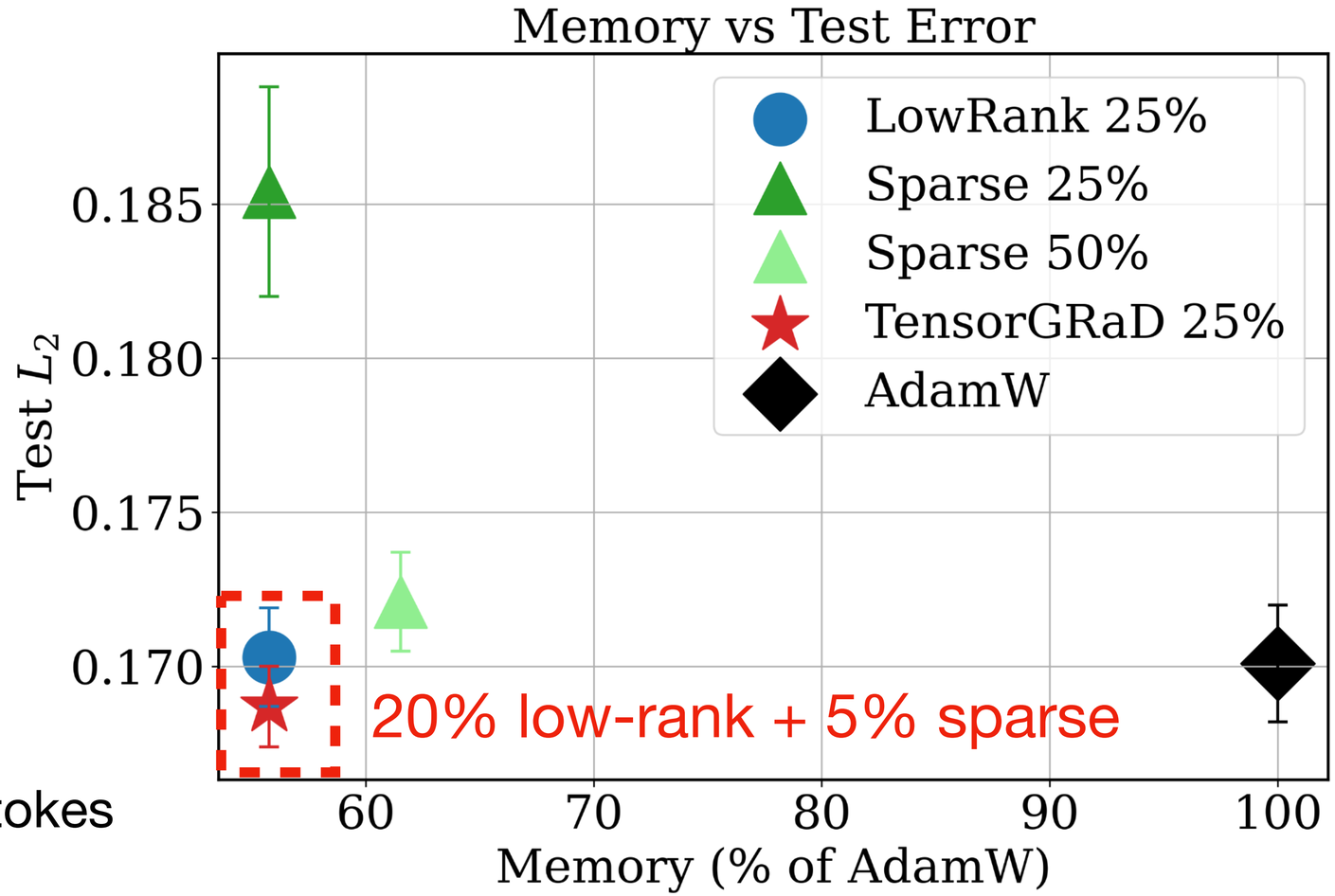
On a highly turbulent ($\mathrm{Re} = 10^5$) Navier Stokes with 1024×1024 resolution



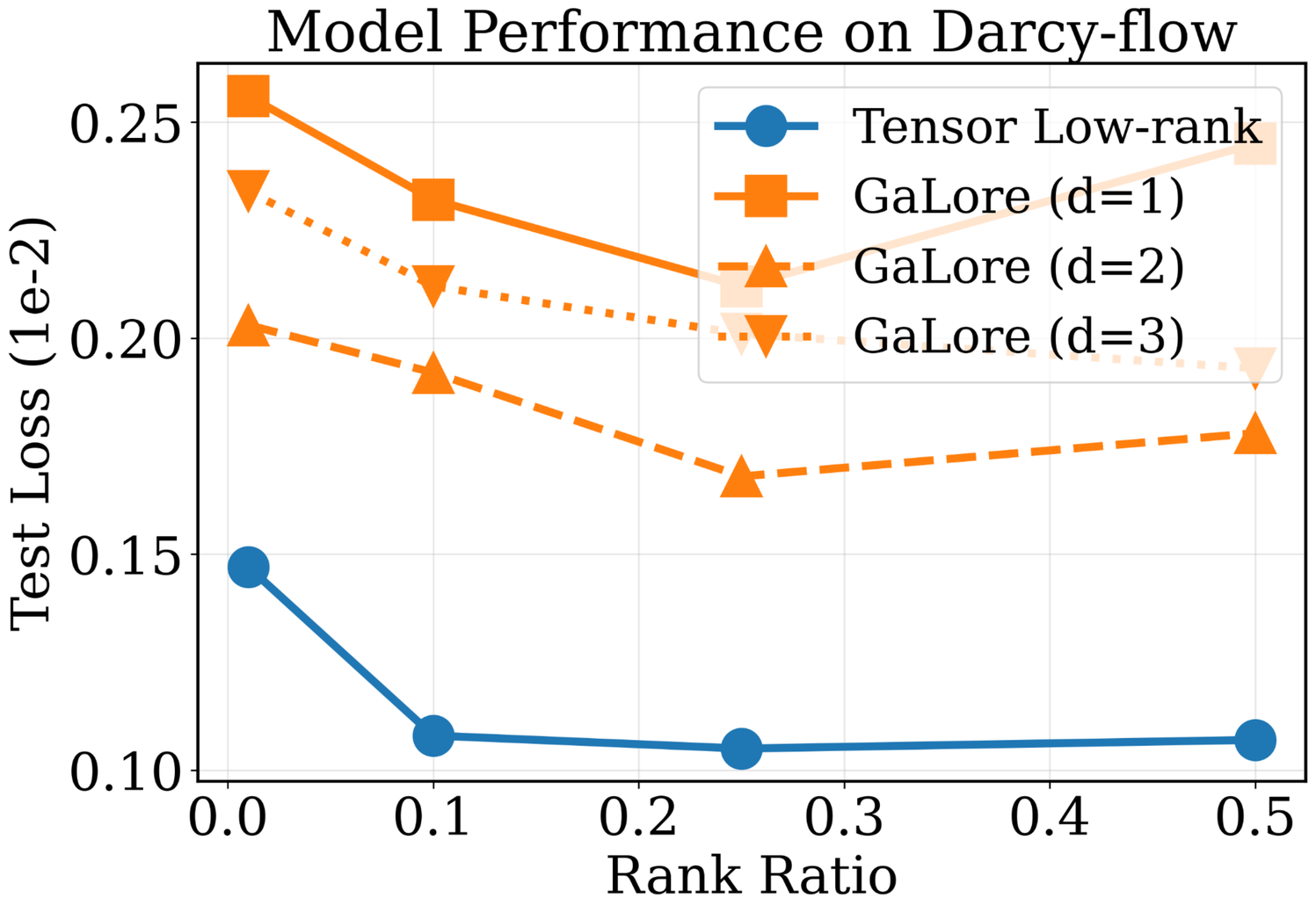
TensorGRaD matches Adam on turbulent Navier–Stokes with ~50% memory



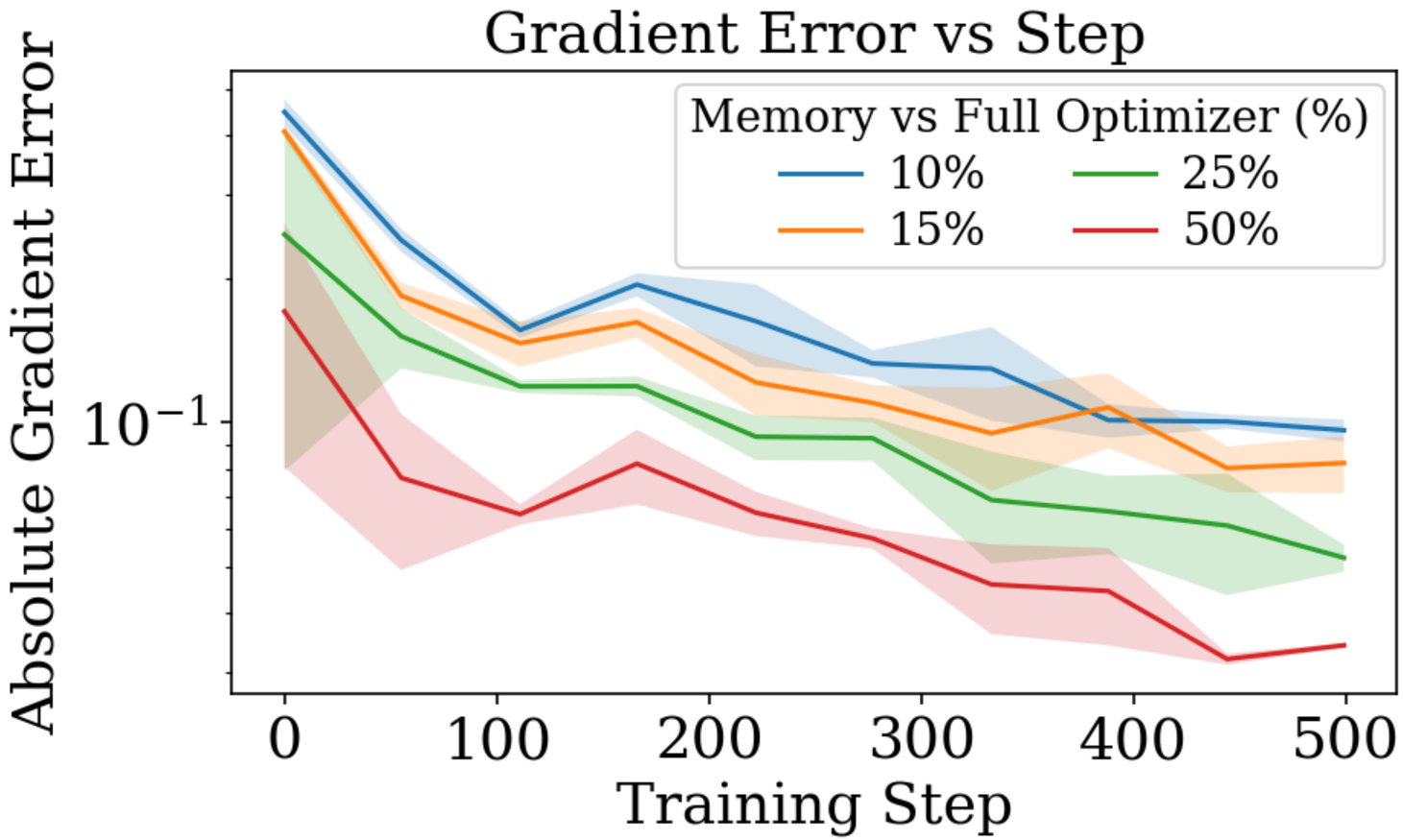
On a highly turbulent (Re $= 10^5$) Navier Stokes with 1024×1024 resolution



Flattened matrix low-rank projections struggle even on simple PDEs

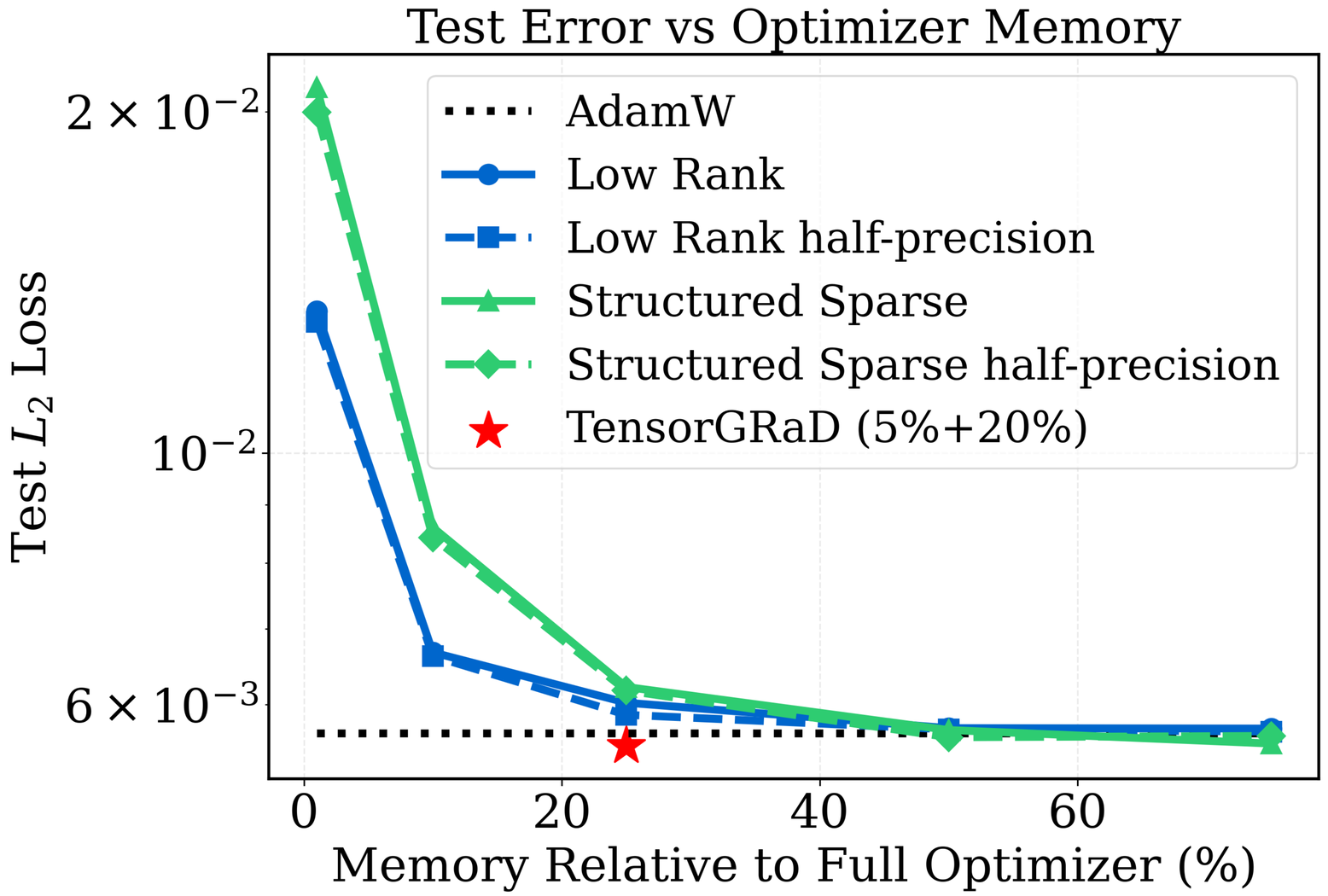


Low-rank tensor projection error decreases during training



Navier-Stokes at 128×128, Re=10^3

Compression ablation Navier-Stokes (Re=10^3)
TensorGRaD matches Adam with a 25% optimizer state



Broader evaluation in the paper

- **Complementary** to factorized weights and mixed-precision training
- **Works across architectures:** FNO, TFNO, GINO
- **Evaluated on diverse benchmarks:** Navier–Stokes, Burgers, Darcy Flow, Electromagnetics, and 3D ShapeNet

Limitations and Future Work

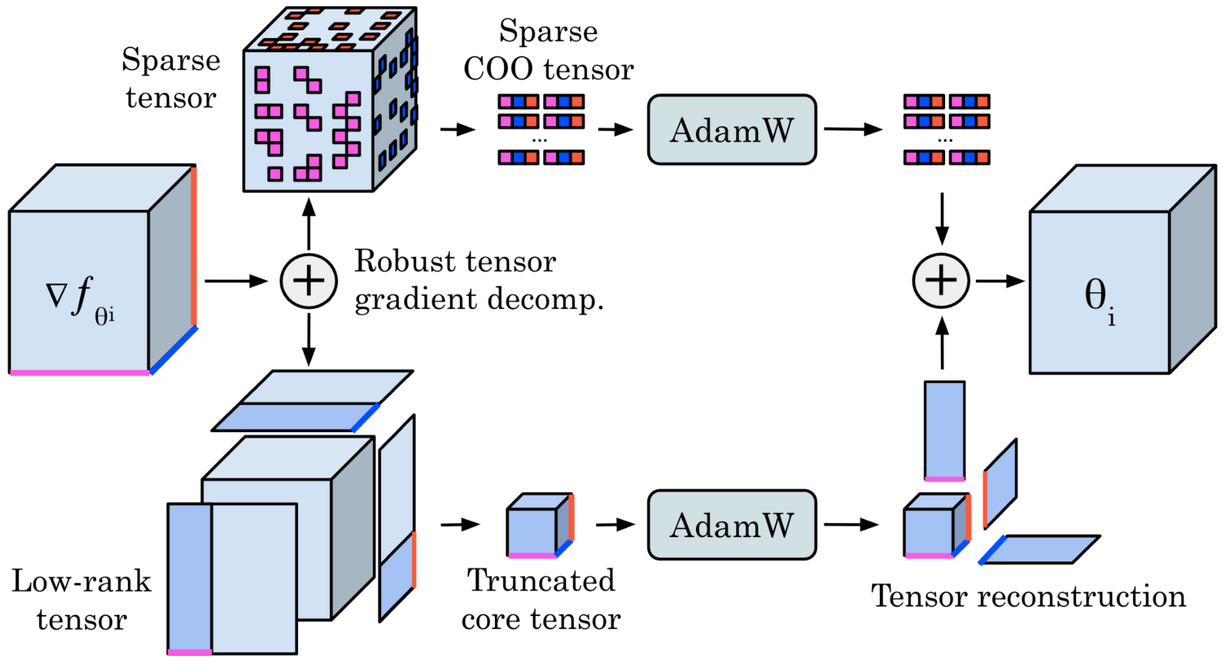
- **Projection overhead (~17% slowdown):** Memory–speed tradeoff
- **Hyperparameter sensitivity:** Need automatic selection of low-rank and sparse percentages
- **Adaptive compression:** Vary compression across layers and training stages

Key Takeaways

- **TensorGRaD**: Robust gradient decomposition (low-rank Tucker + unstructured sparse) reduces optimizer memory while matching baseline accuracy

- **Outcome**: Enables more memory-efficient training of large neural operators on high-resolution PDE tasks.

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



Github

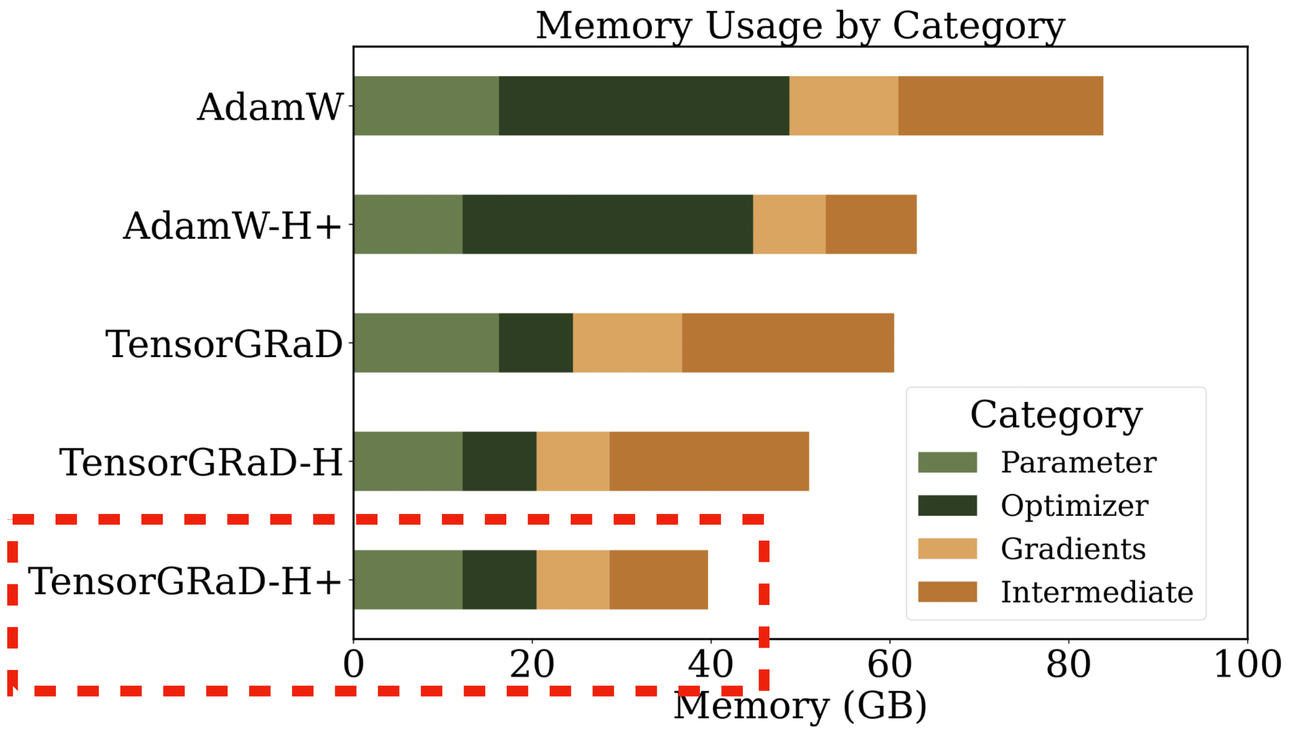
Sebastian Loeschcke, David Pitt, Robert Joseph George, Jiawei Zhao, Cheng Luo, Yuandong Tian, Jean Kossaifi , Anima Anandkumar

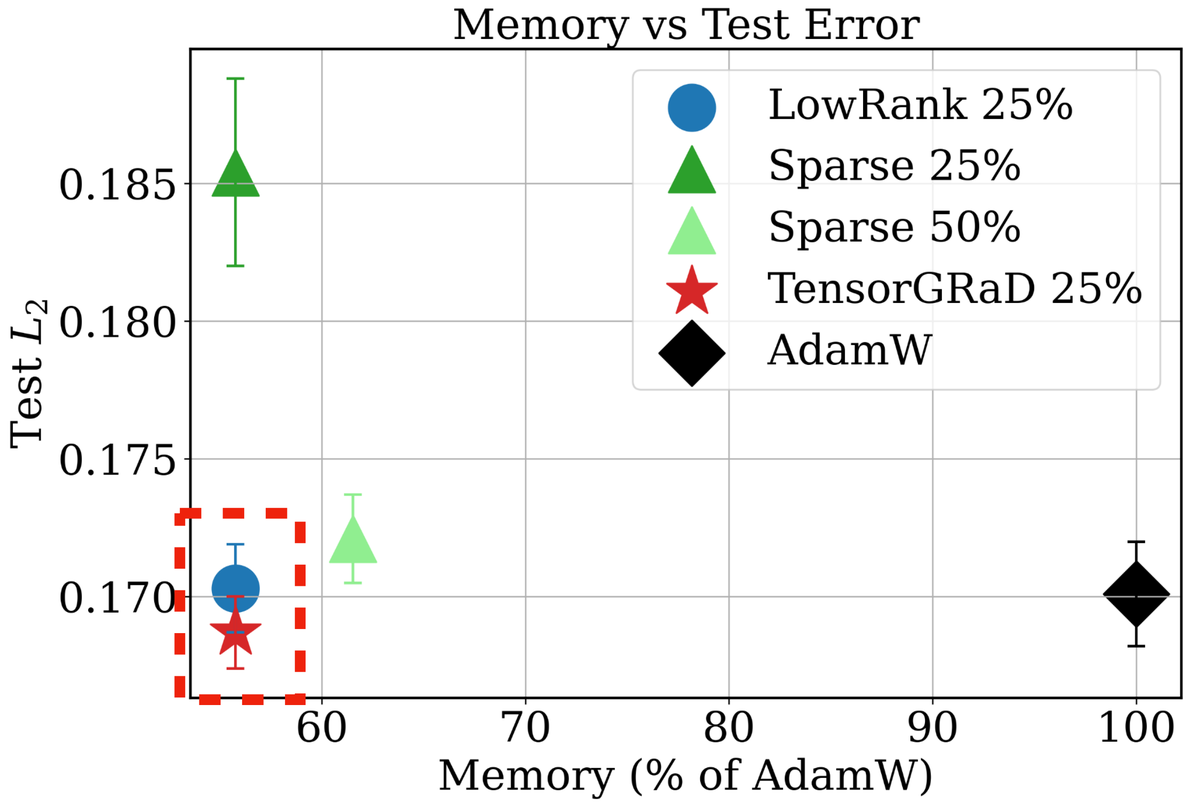
UNIVERSITY OF COPENHAGEN

Caltech Meta NVIDIA

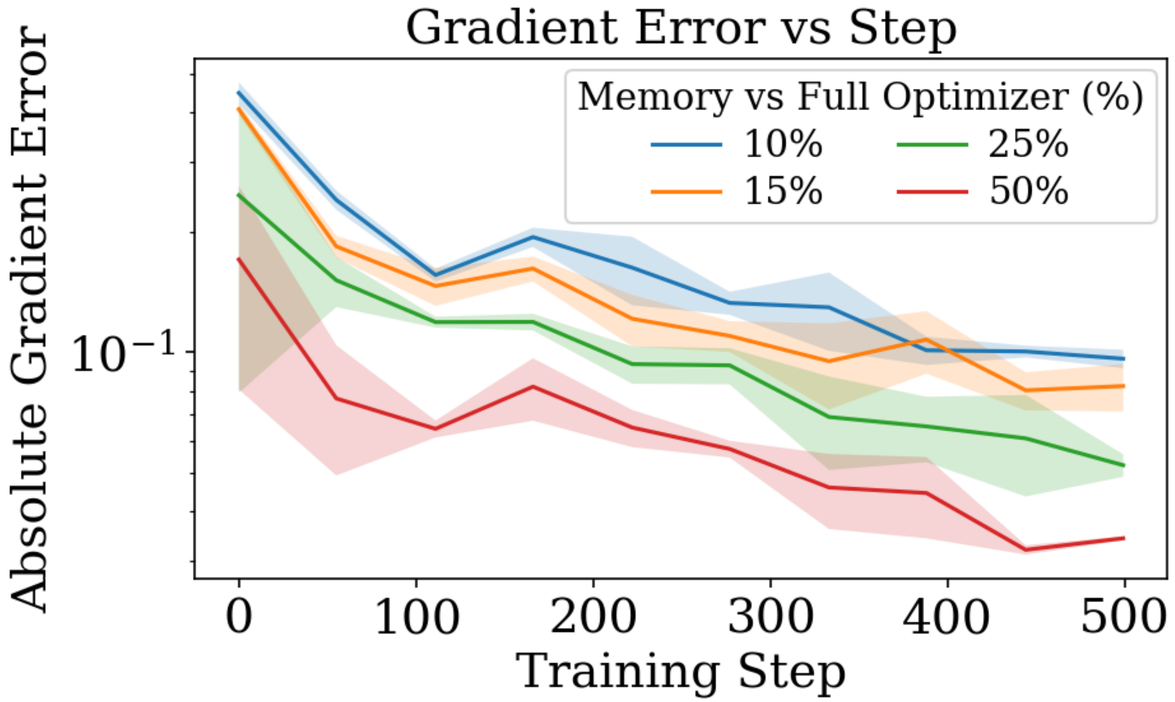
Extra

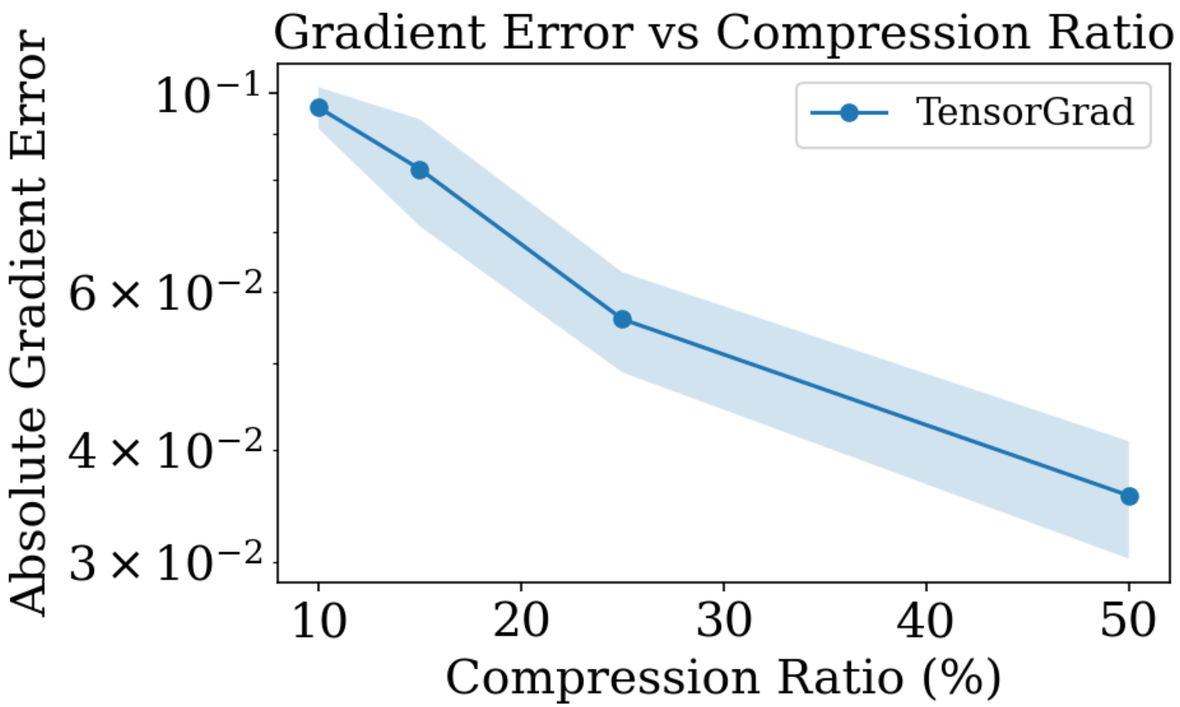
TensorGRaD matches baselines with roughly *half* the memory





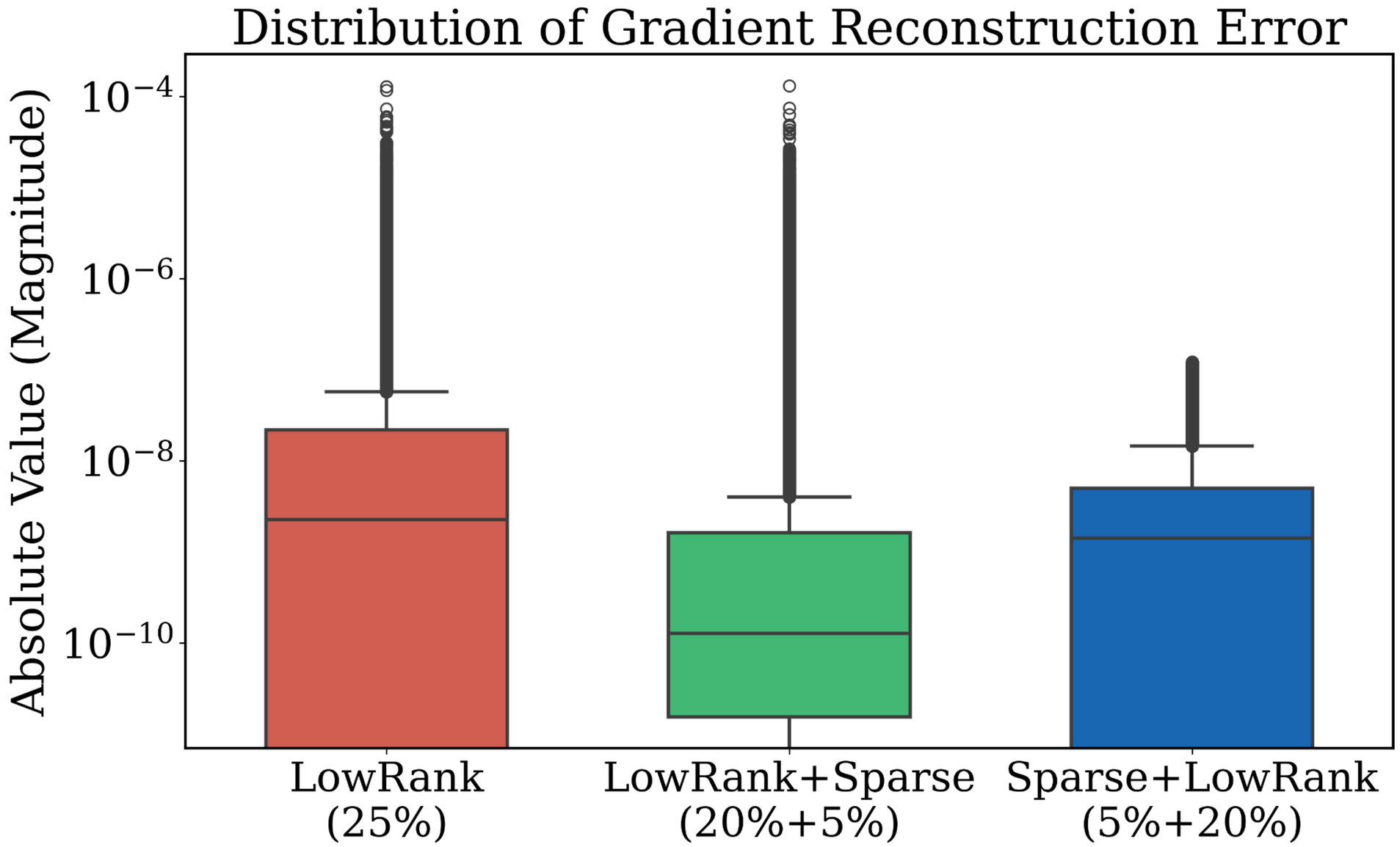
Gradient error at final step decreases when we compress less



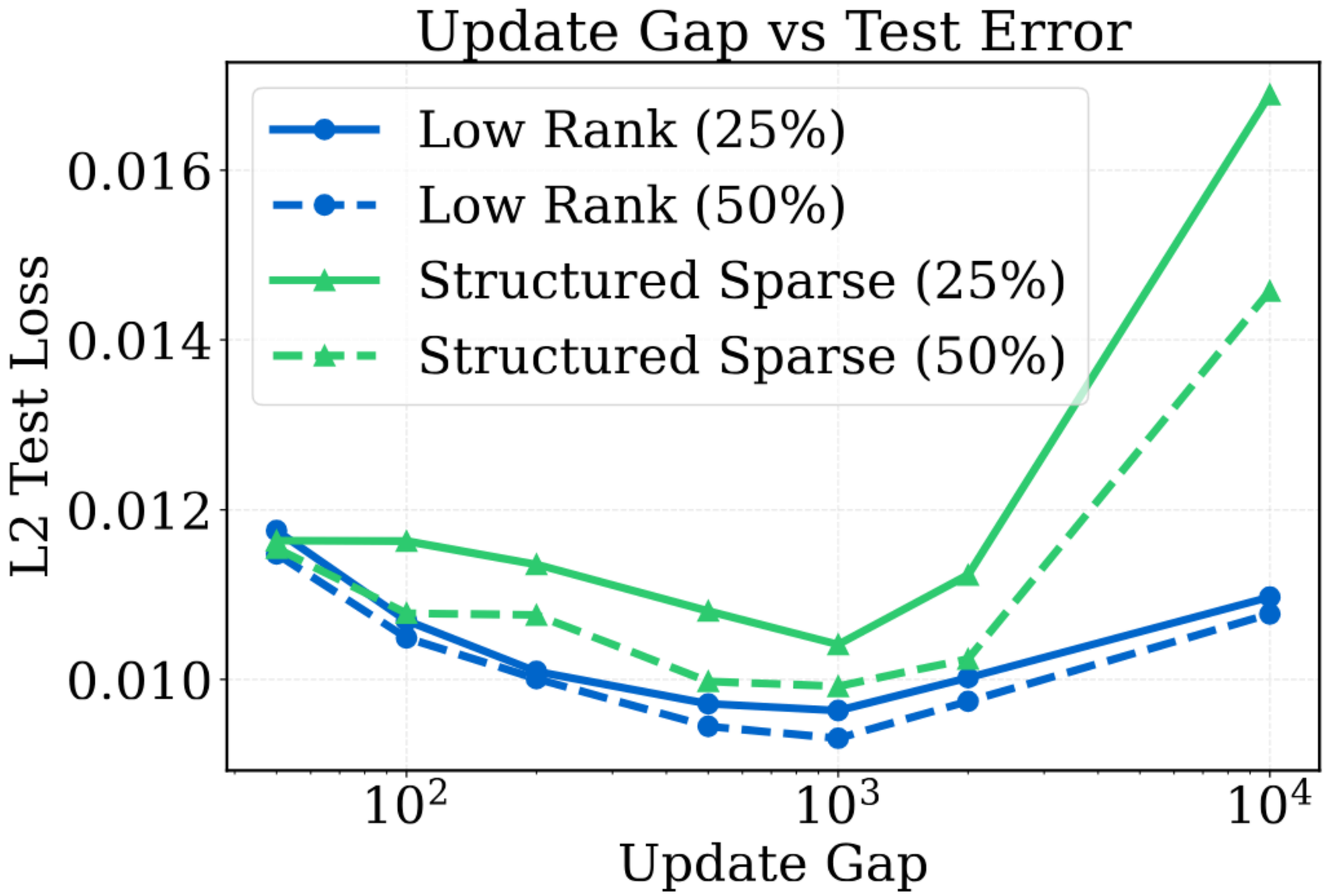


Navier-Stokes at 128×128, Re=10^3

Sparse + low-rank can better capture outliers in the gradient



Sparse + low-rank can better capture outliers in the gradient



Low-rank + Sparse Combinations

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 $\to$ SS	7.09 / 6.44	6.91 / 6.32	6.72 / 6.35
LR $\to$ US	6.29 / 6.20	6.22 / 6.19	6.47 / 6.26
SS $\to$ LR	6.56 / 6.26	6.66 / 5.96	6.22 / 6.21
US $\to$ LR	6.24 / 6.10	6.19 / 6.03	**5.72** / 5.73
LR + US (sum)	6.18 / 6.12	6.17 / 6.06	–

Orthogonal to mixed-precision training

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

Orthogonal to Tensorized weights

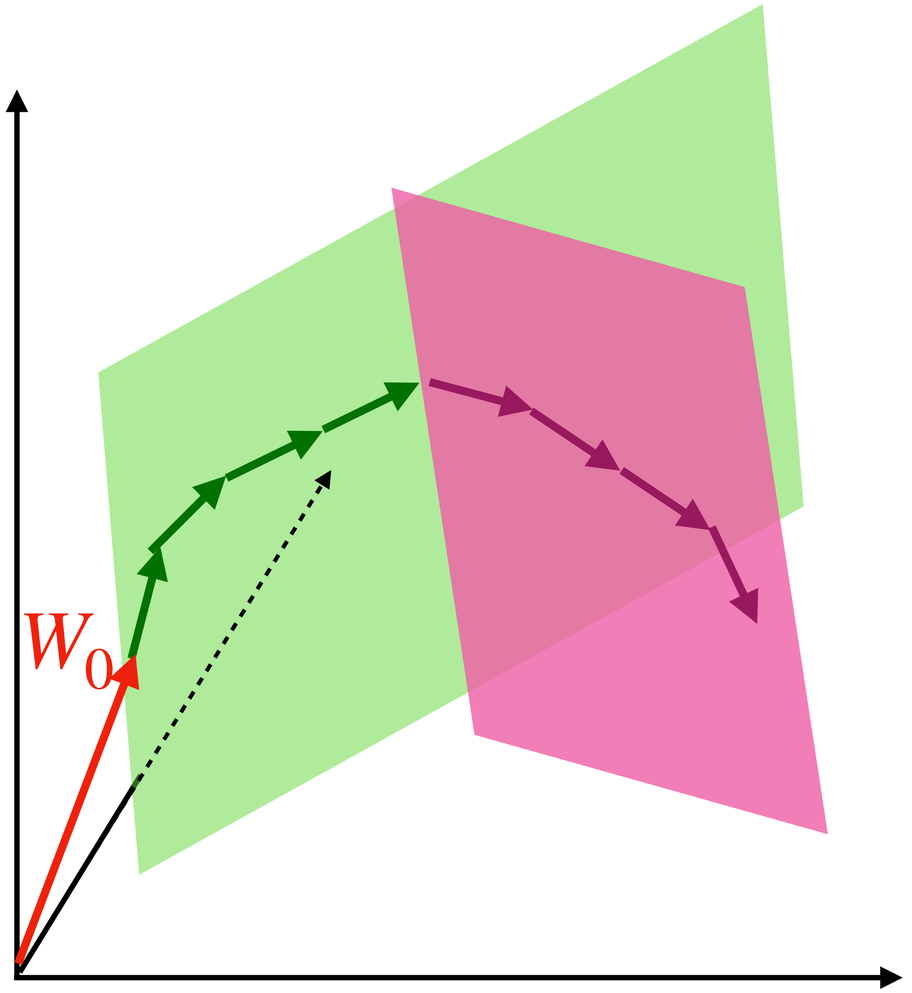
Table 4: TensorGRaD with tensorized weights (TFNO) on Navier–Stokes at two resolutions. We report L2 test loss; TFNO rank denotes retained weight percentage. NS 128 uses Re $\approx 10^3$ (500 steps) and NS 1024 uses Re $\approx 10^5$ (50 steps).

	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

Performance on 3D input data

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



$$W_N = W_0 + P_0B_0 + P_1B_1 + \ldots + P_NB_N$$