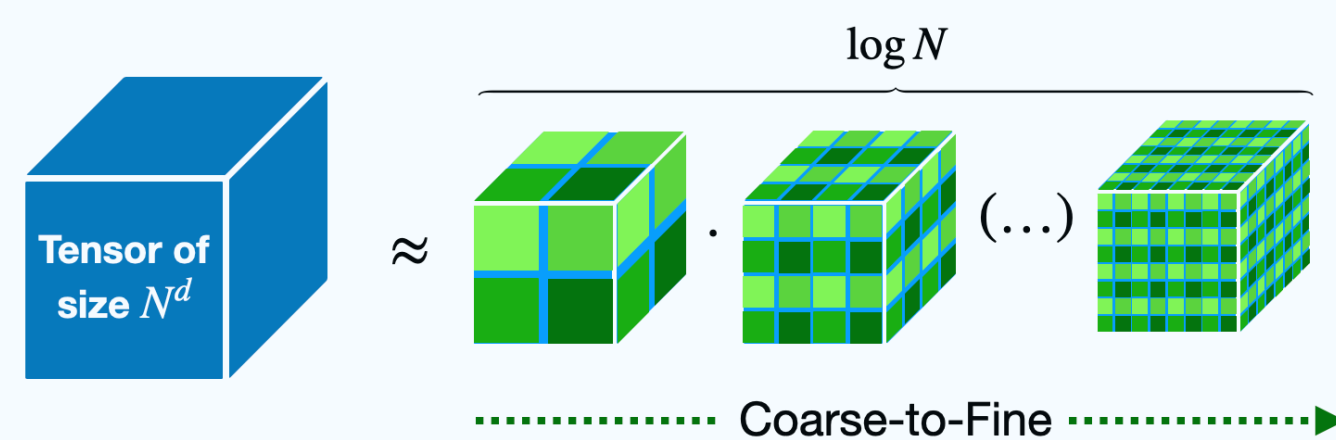


Structured Learning Under Memory Constraints

Sebastian Loeschcke

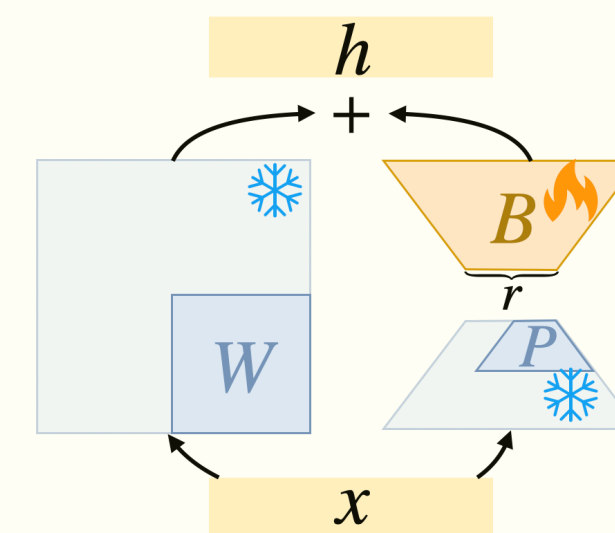
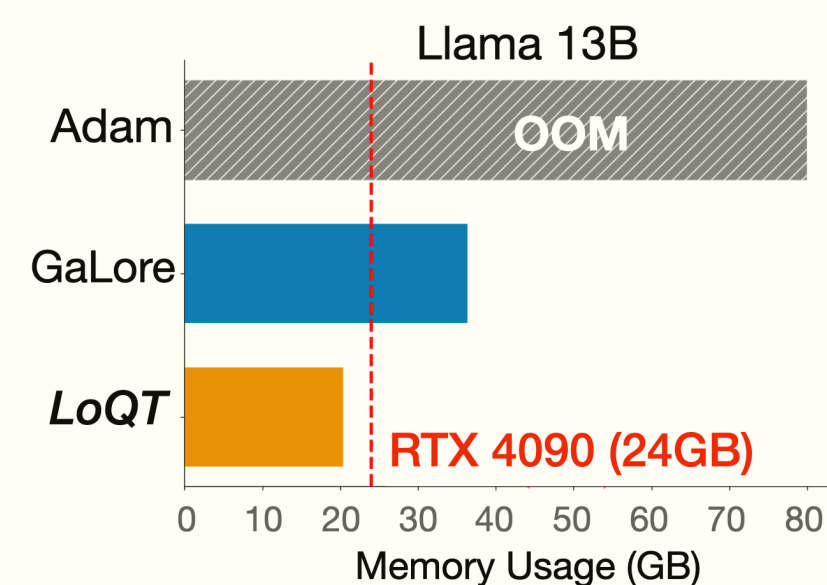
ICML 2024

Coarse-To-Fine Tensor Trains for Compact Visual Representations



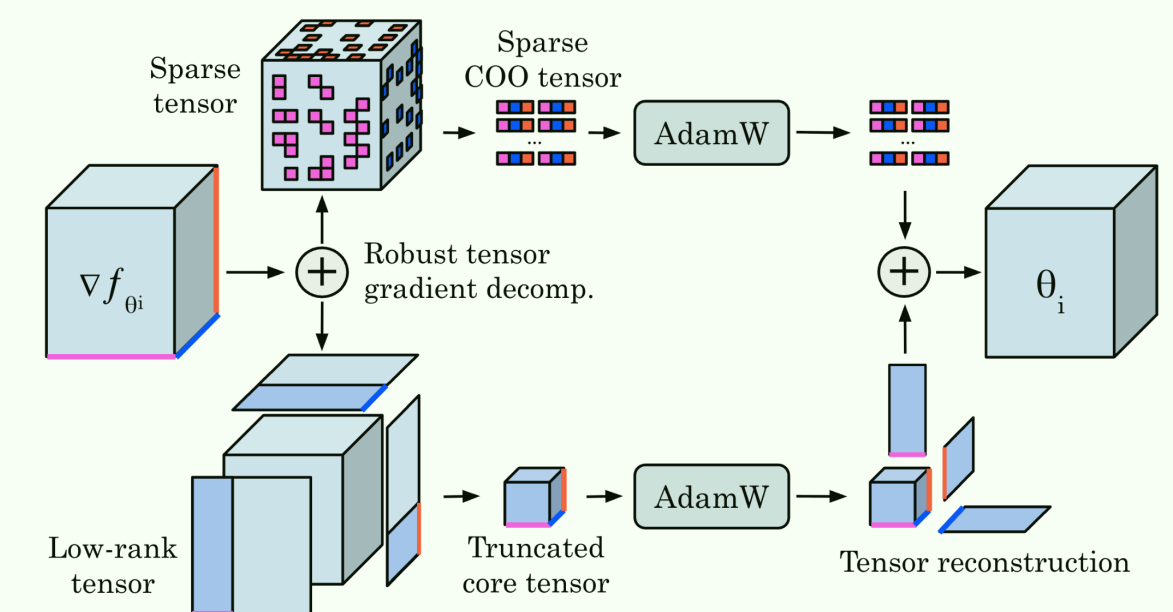
NeurIPS 2024

LoQT: Low Rank Adapters for Quantized Pre-Training



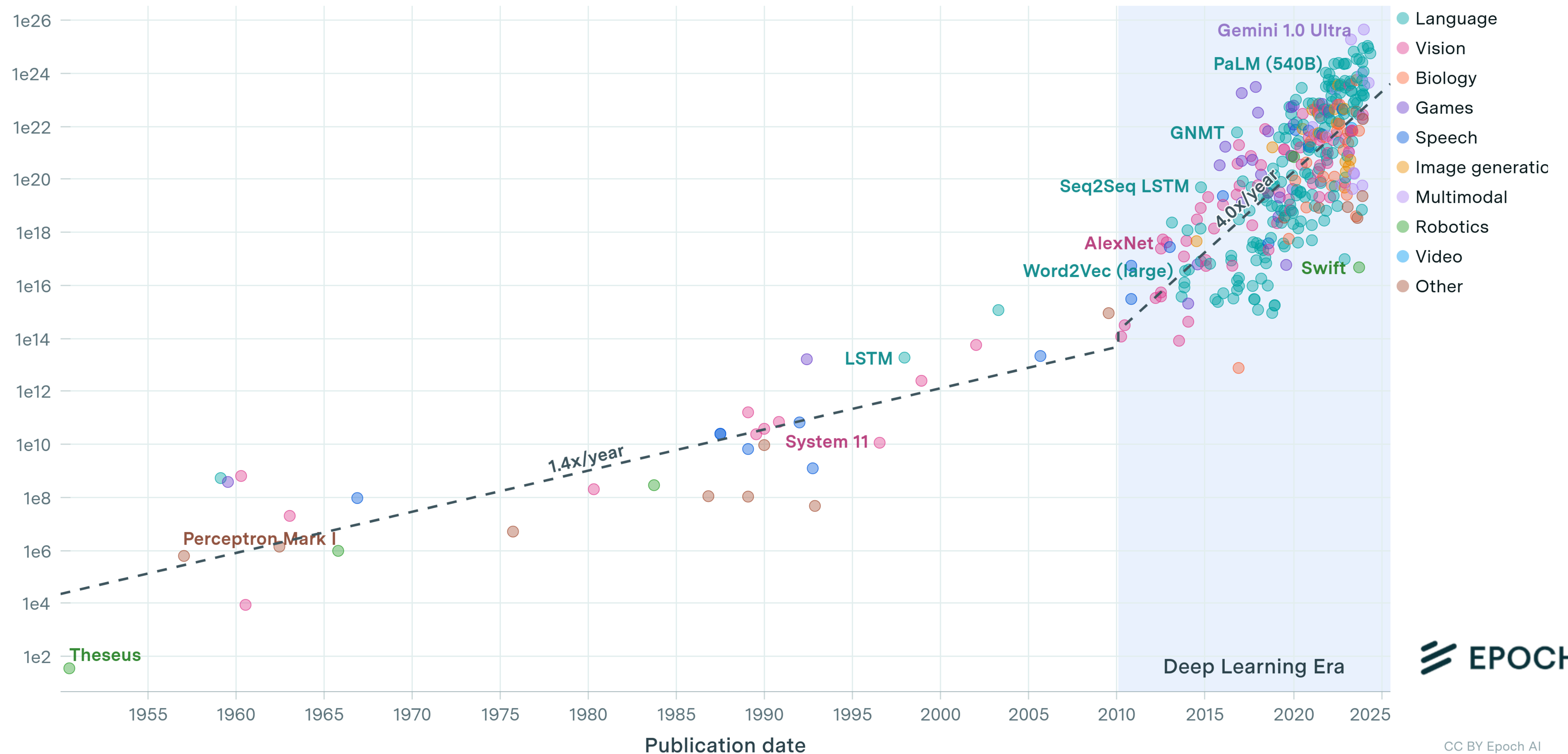
Under review

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



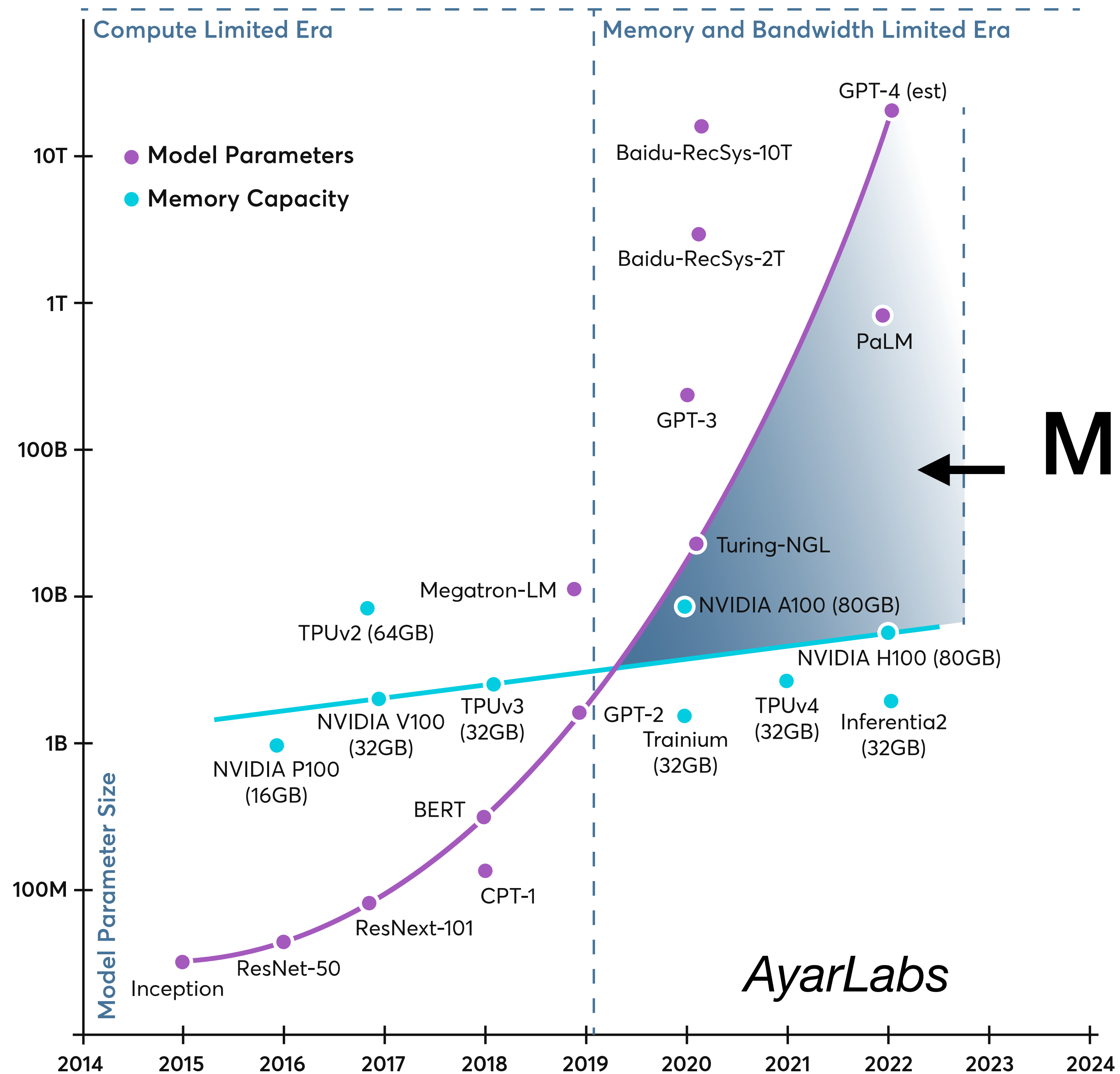
Training compute has increased by a factor of 10 billion since 2010, doubling every 5–6 months.

Training compute (FLOP)



Training modern models increasingly requires large GPU clusters with high memory capacity.

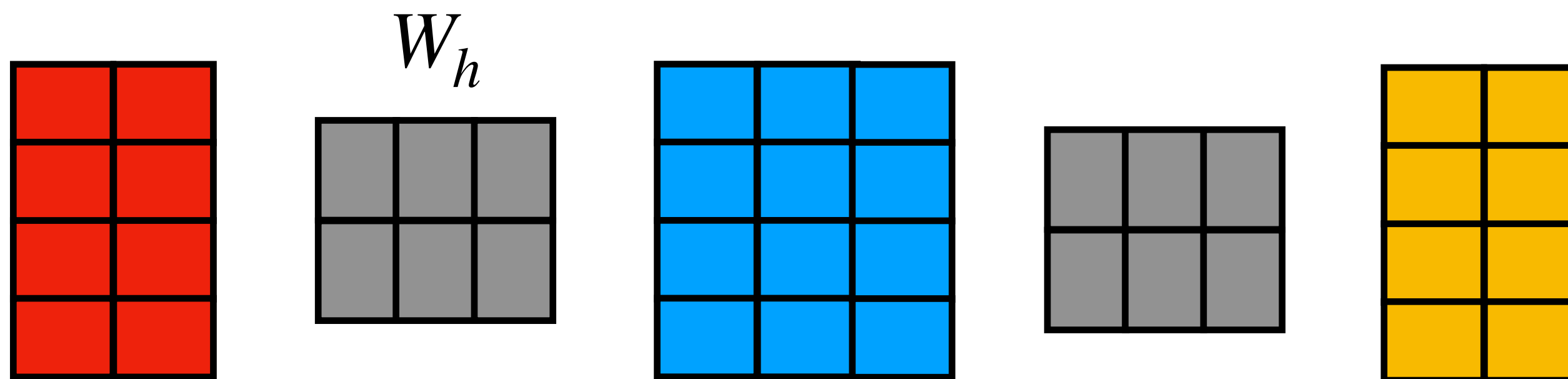
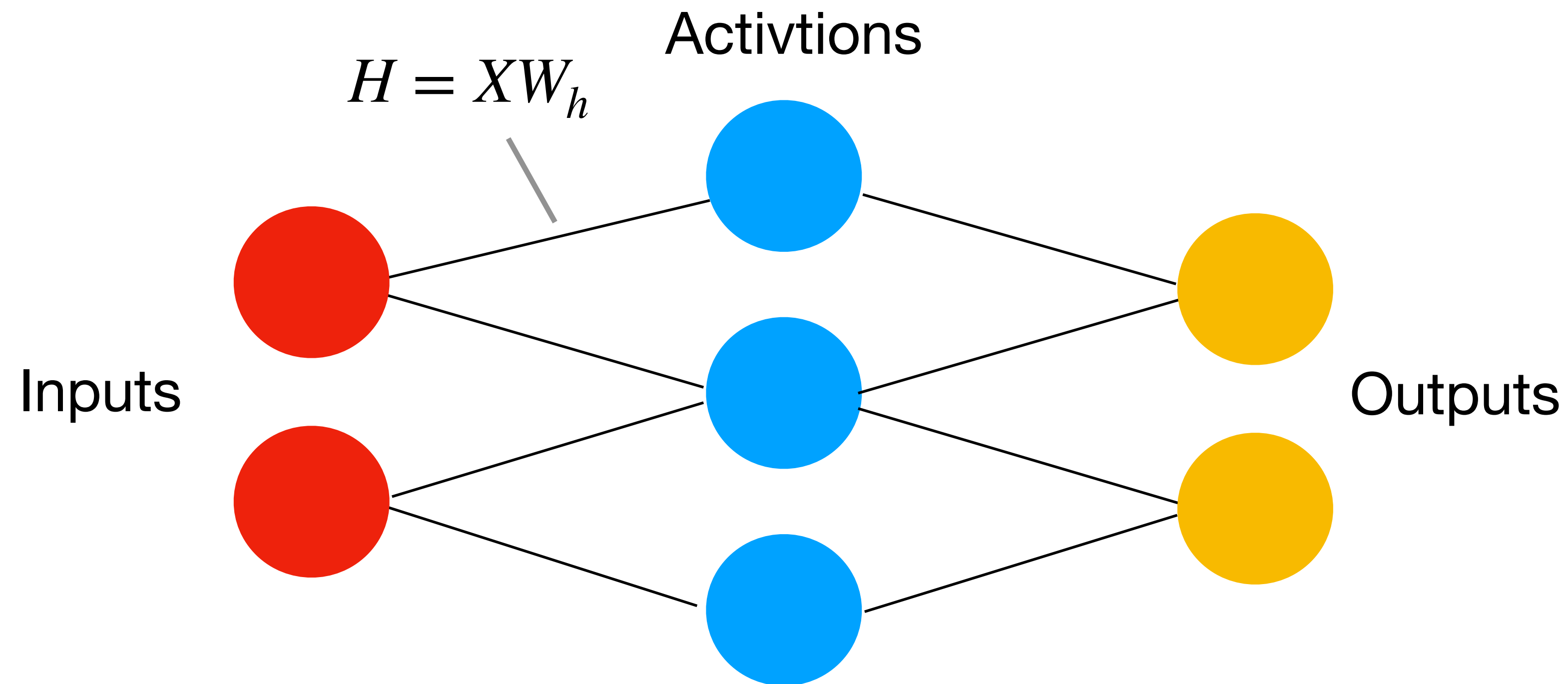




How do we train expressive models
when memory is the bottleneck?

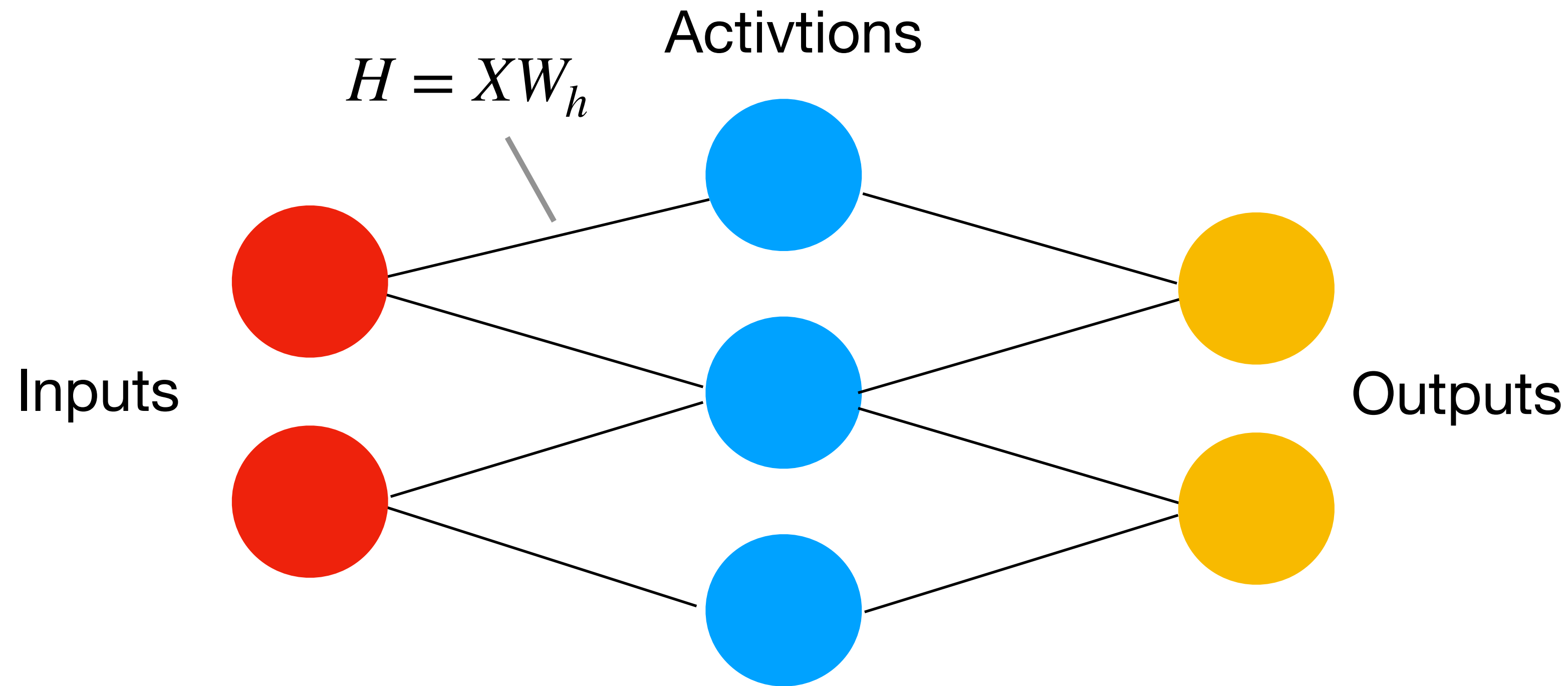
Where does the memory come from?

Forward pass of two layer neural network



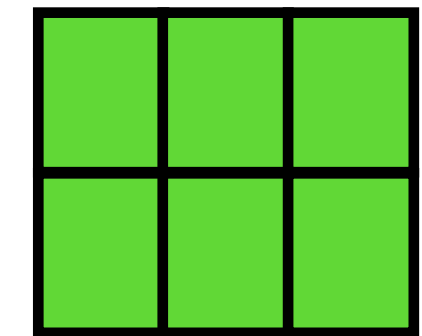
Where does the memory come from?

Forward pass of two layer neural network

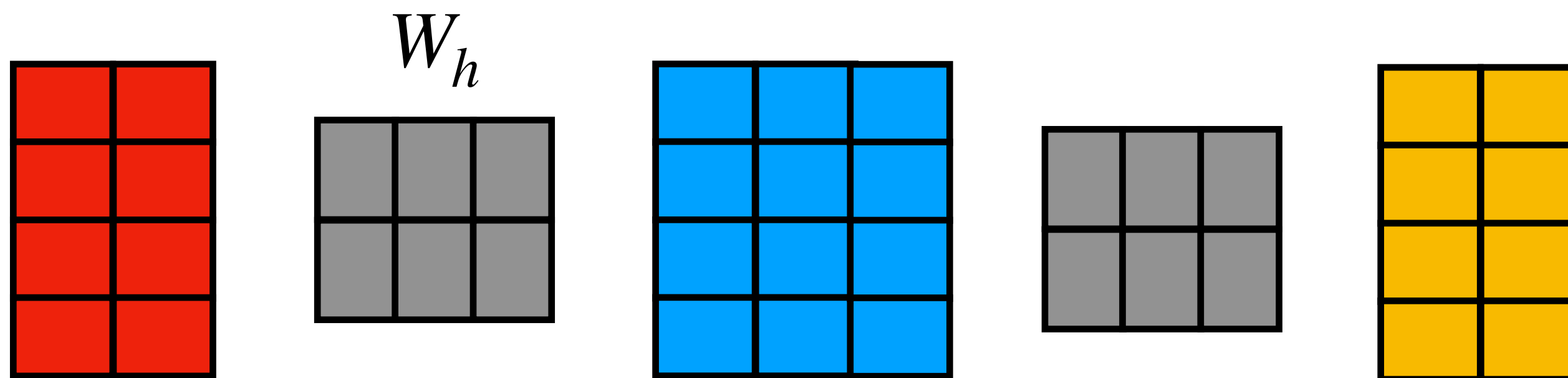
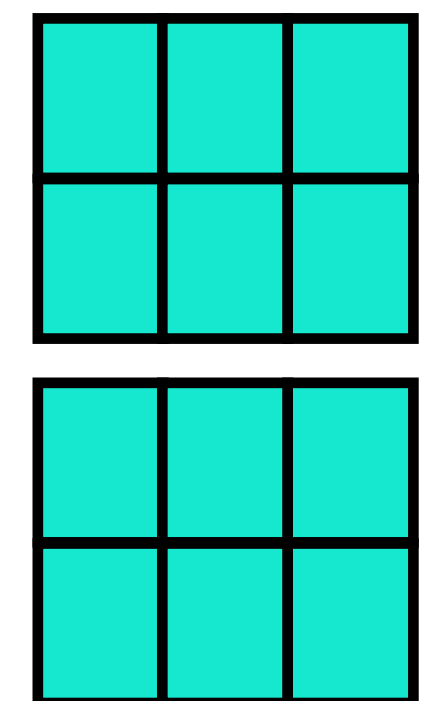


Optimization

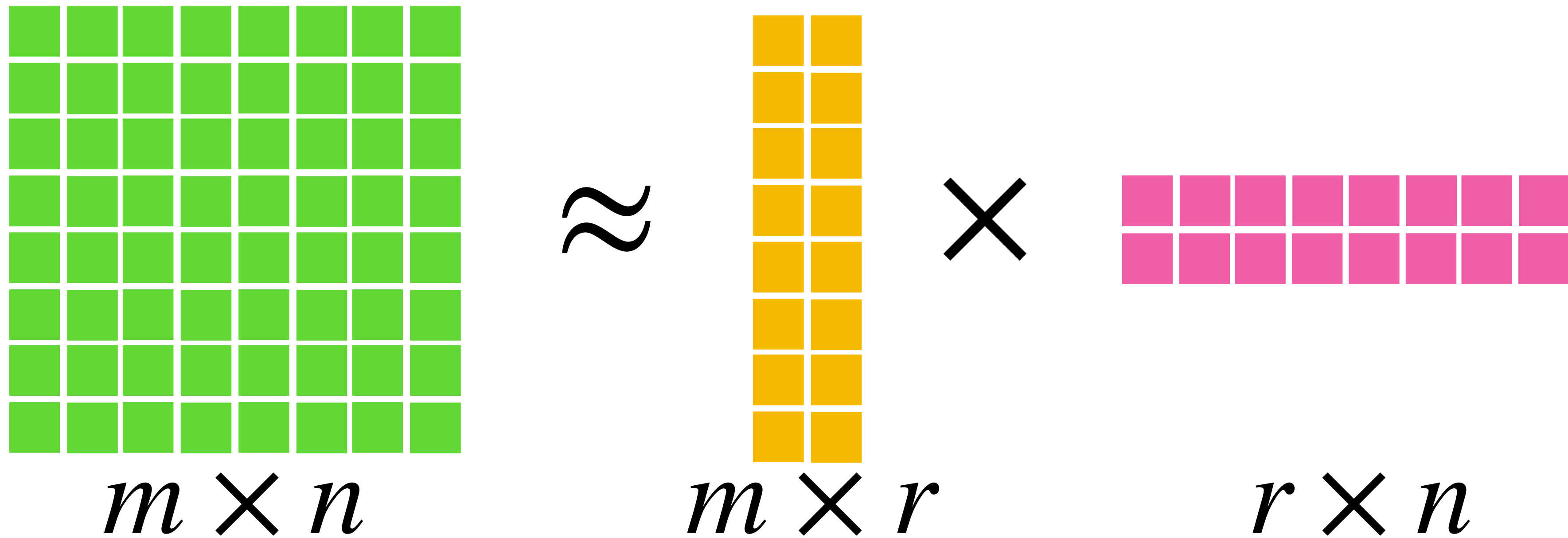
Gradients



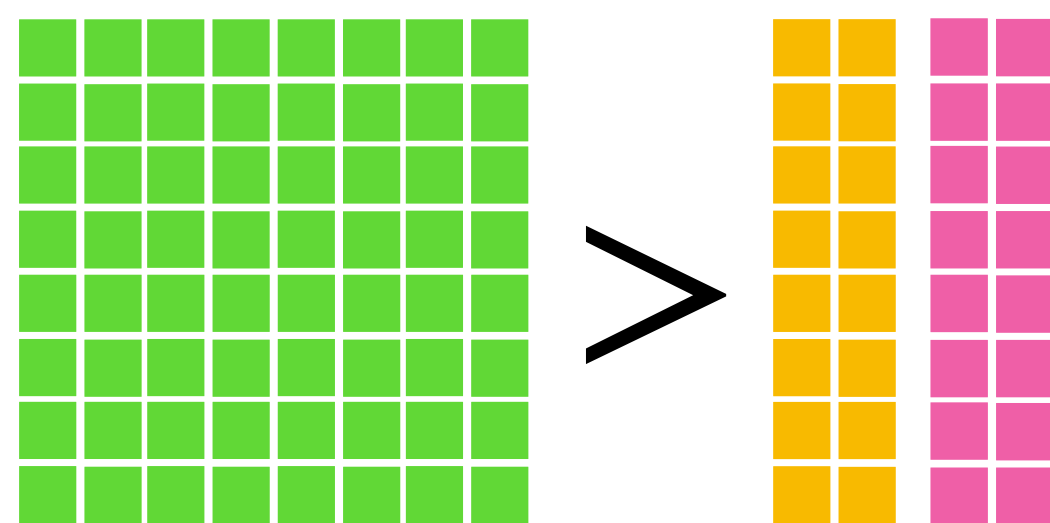
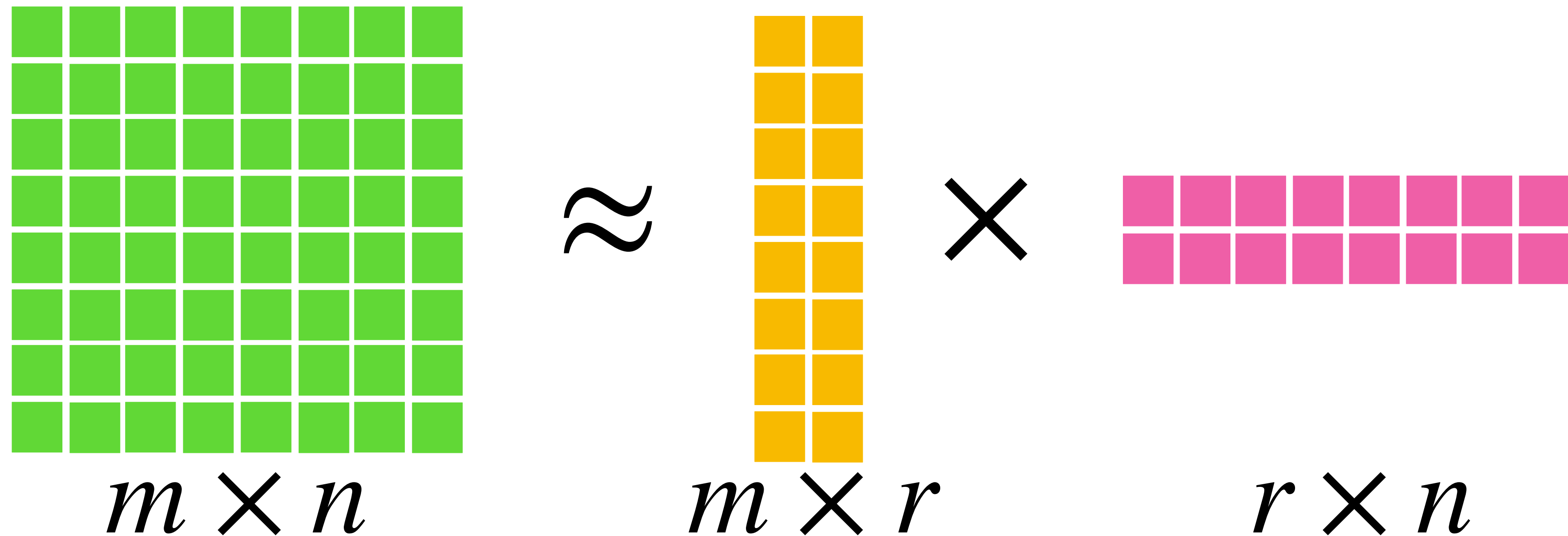
Optimizer state (Adam)



Reduce memory and compute with factorization



Reduce memory and compute with low-rank factorization

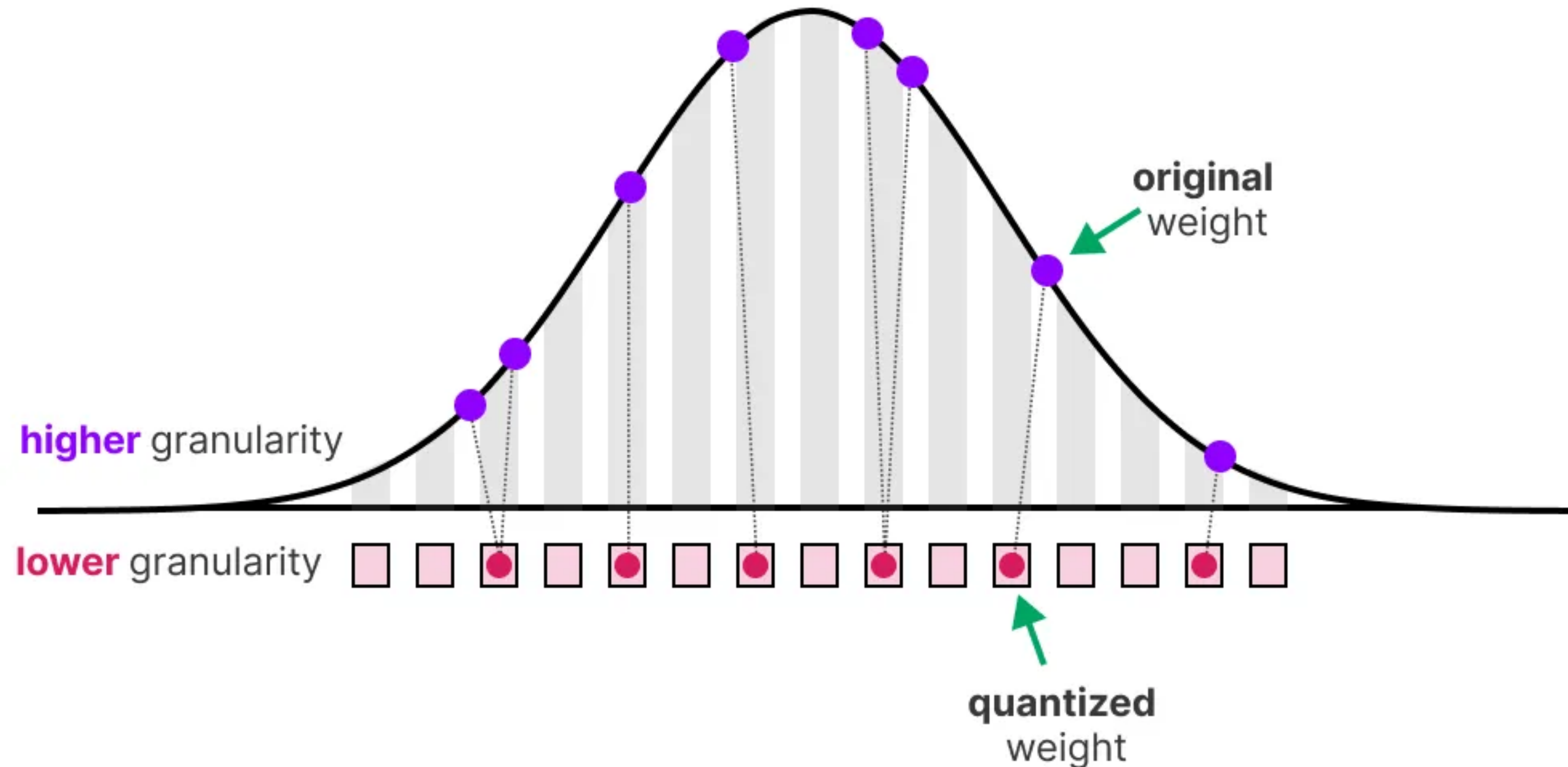


Reduces number of
floating points

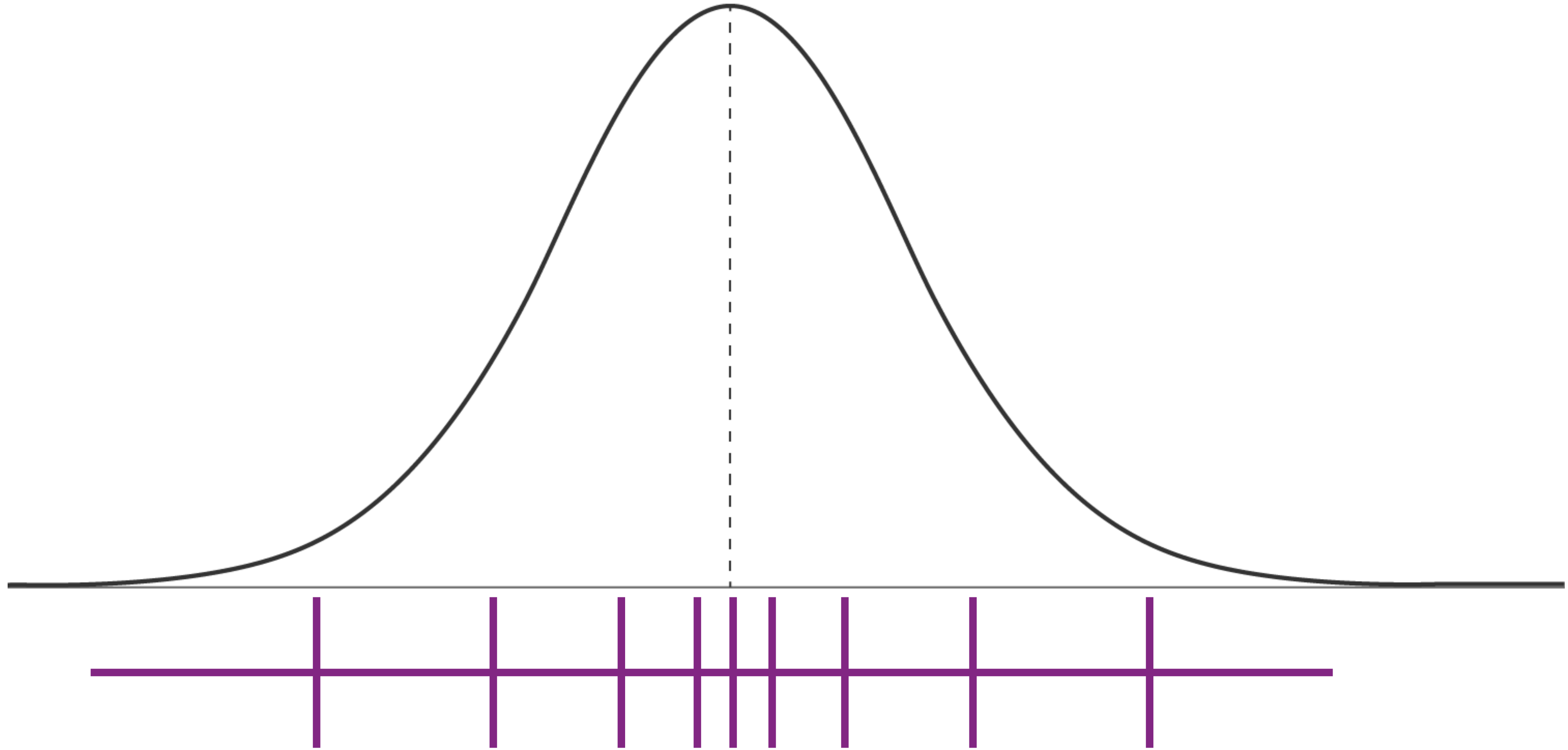
Can cause loss
of precision

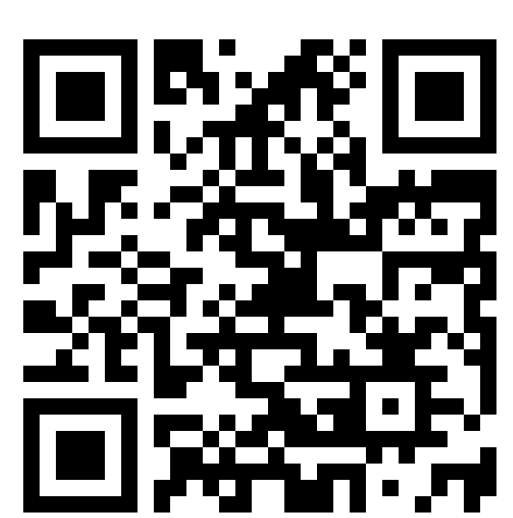
Quantization

Reduce memory usage by converting from high to low precision.



Exploiting Structure of Gaussian Weights





Coarse-To-Fine Tensor Trains for Compact Visual Representations

Sebastian Loeschcke, Dan Wang, Christian Munklinde,
Serge Belongie, Michael Kastoryano, and Sagie Benaim

ICML 2024

UNIVERSITY OF
COPENHAGEN



IT UNIVERSITY OF CPH



PIONEER CENTRE FOR
ARTIFICIAL INTELLIGENCE

Compact visual representations are important for applications like novel view synthesis and 3D modeling

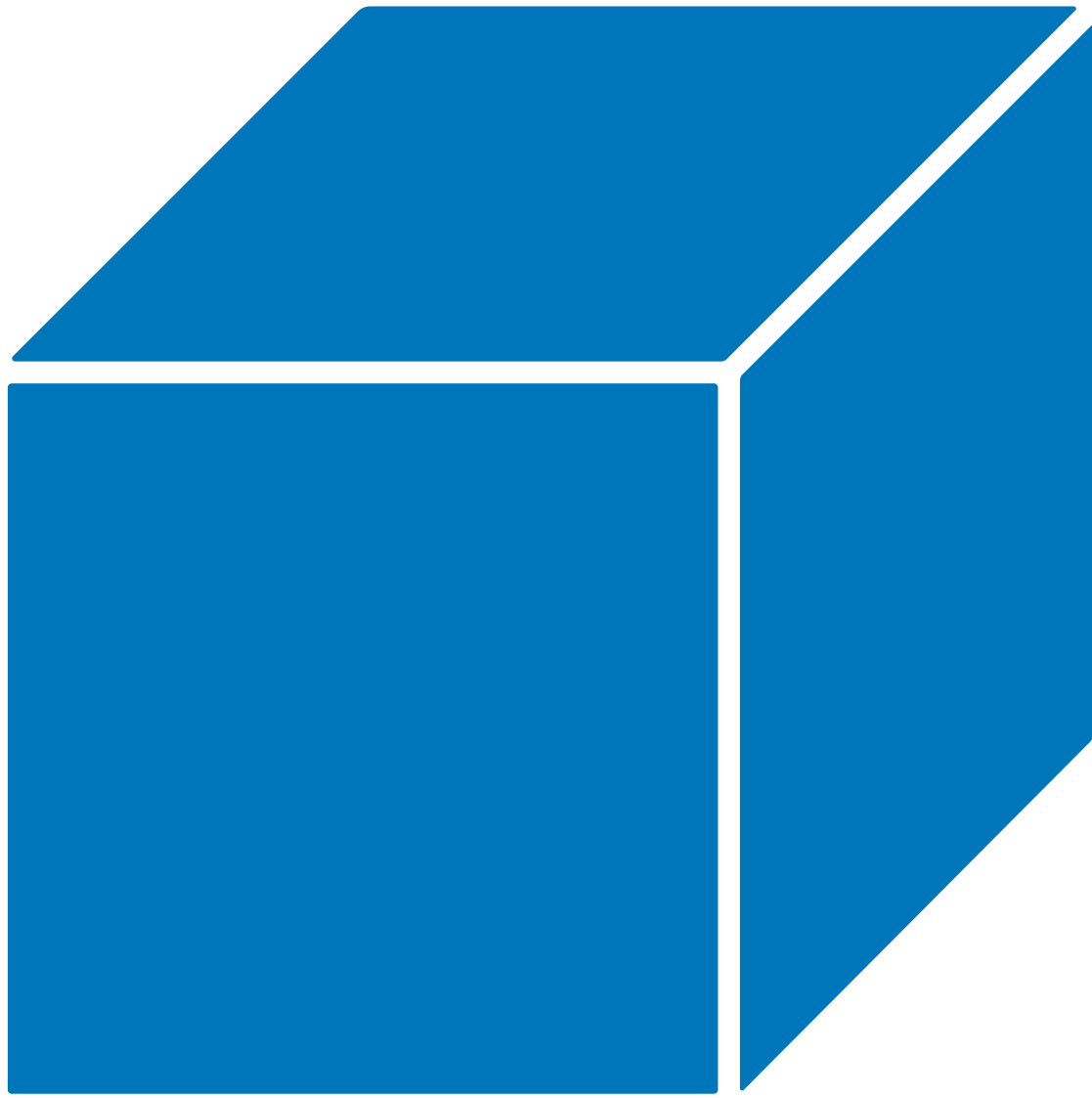
Neural Radiance Fields



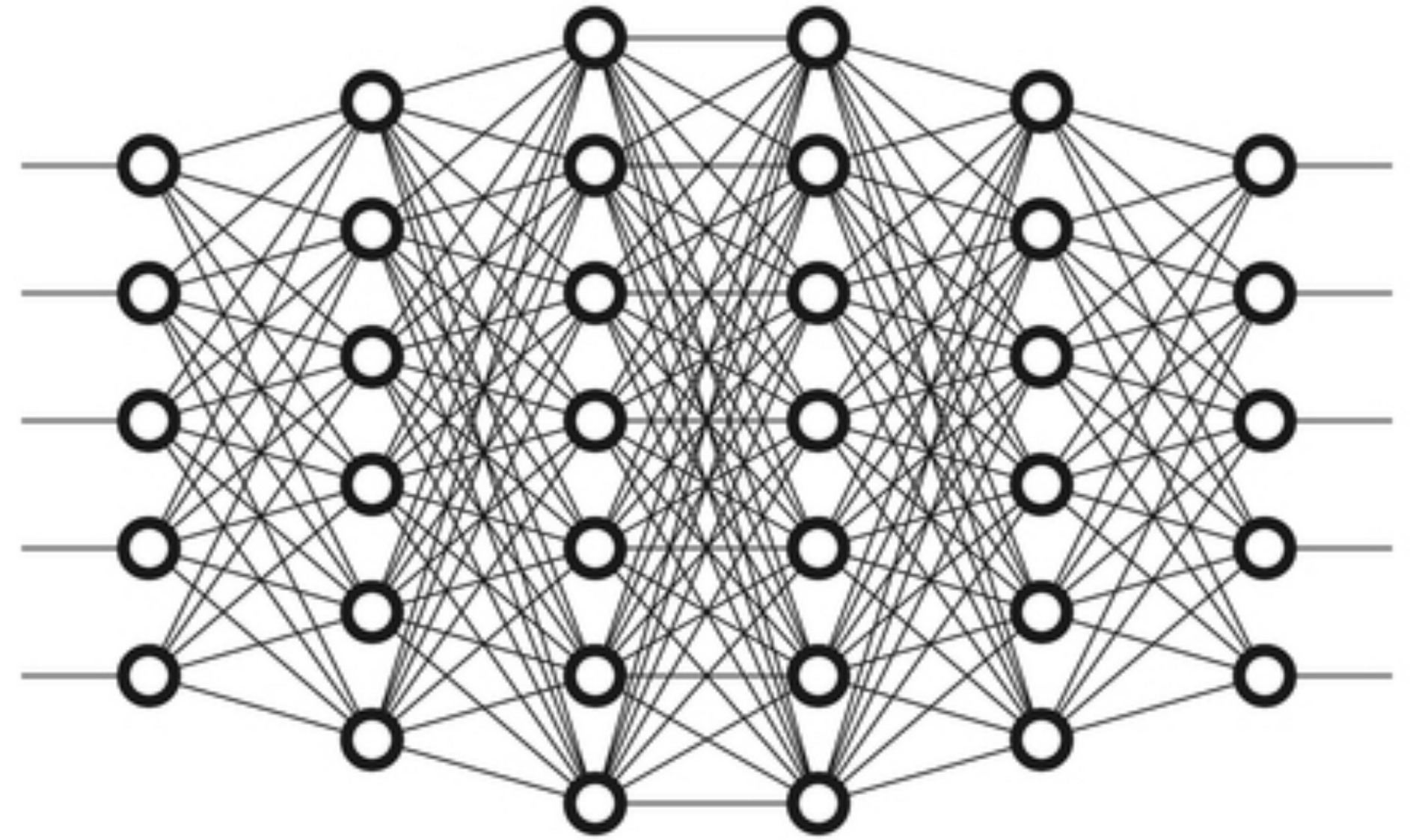
2D Images

3D Signed Distance Fields

Compact Visual Representations

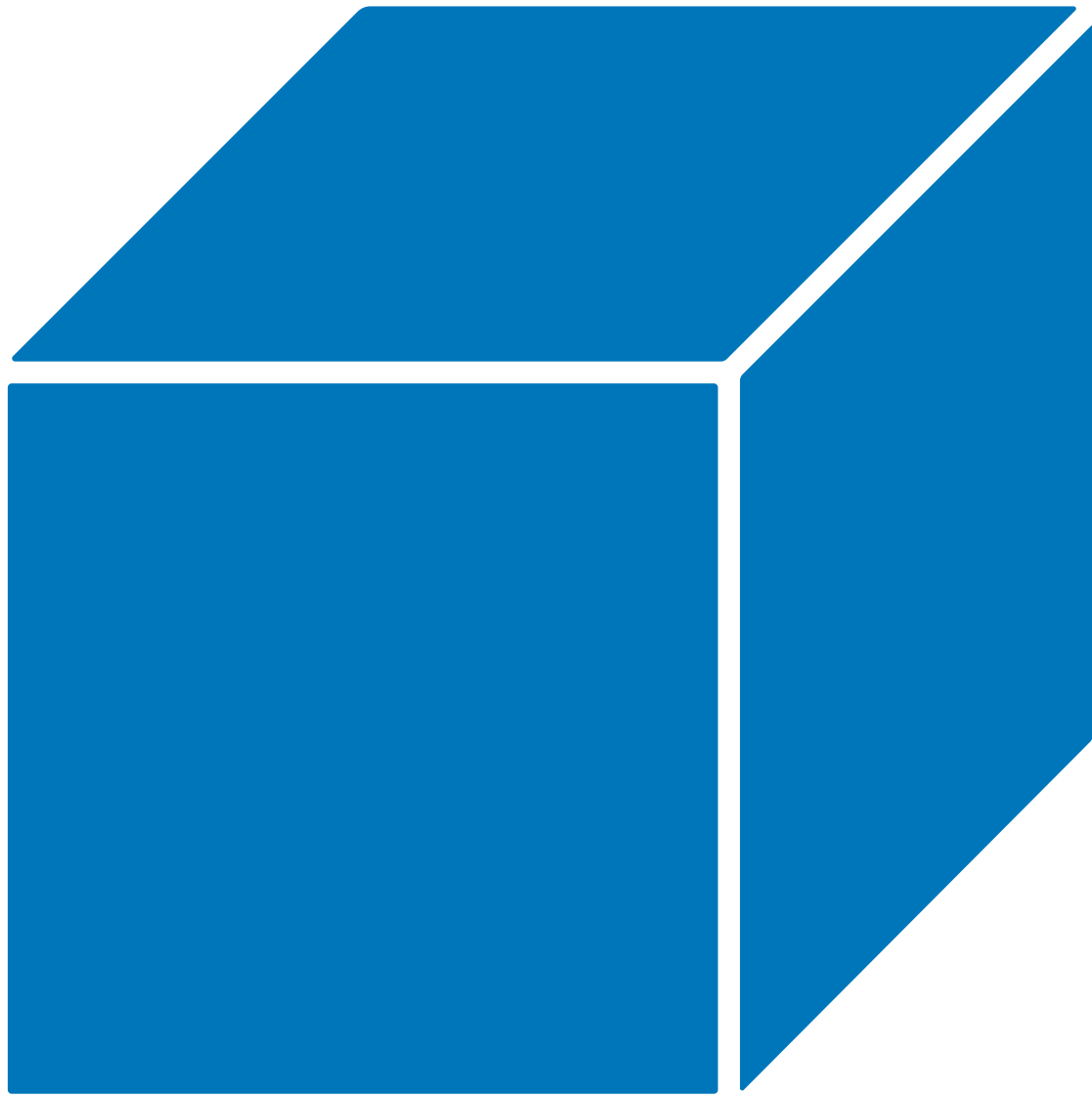


Explicit



Implicit

Compact Visual Representations



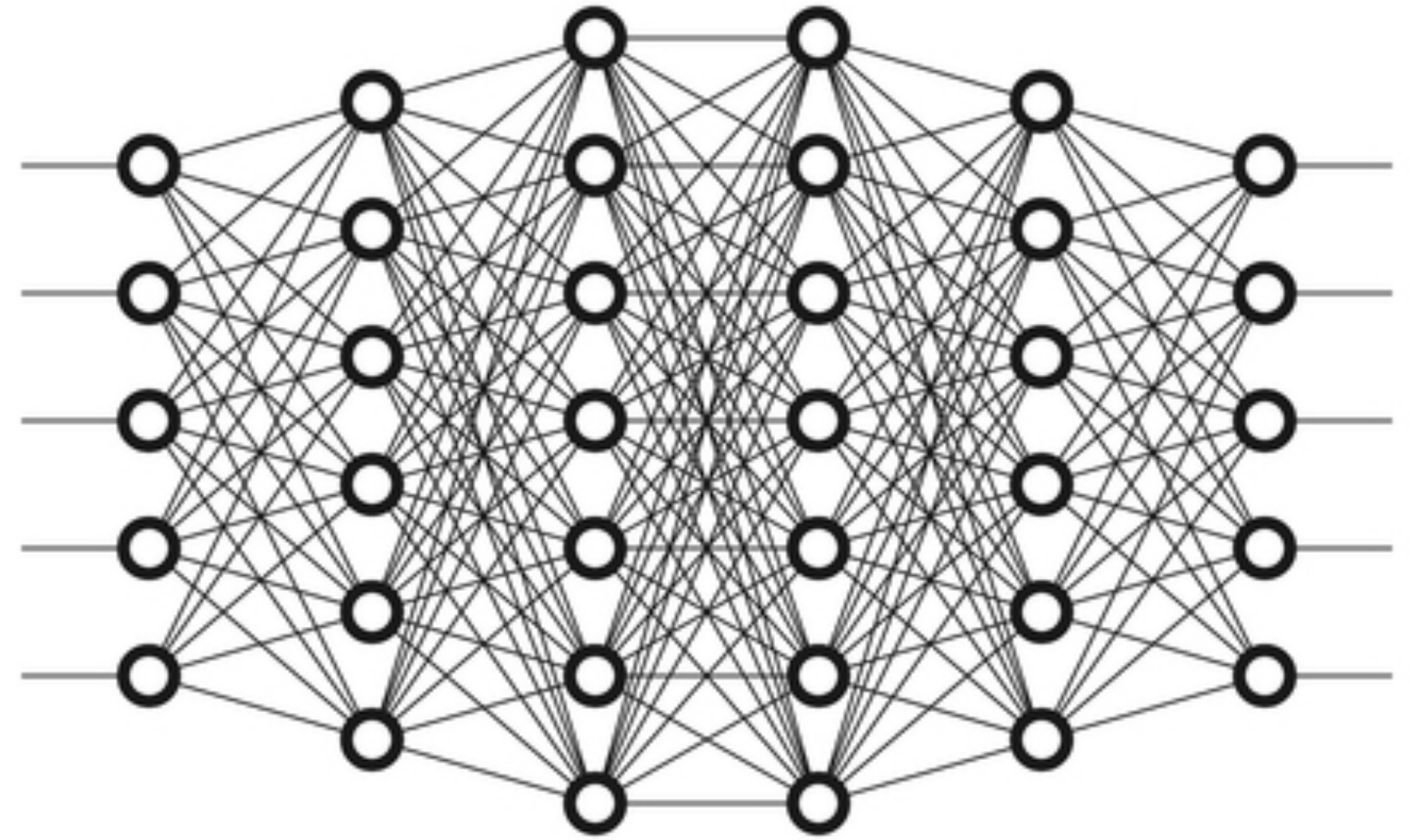
Explicit



High Memory

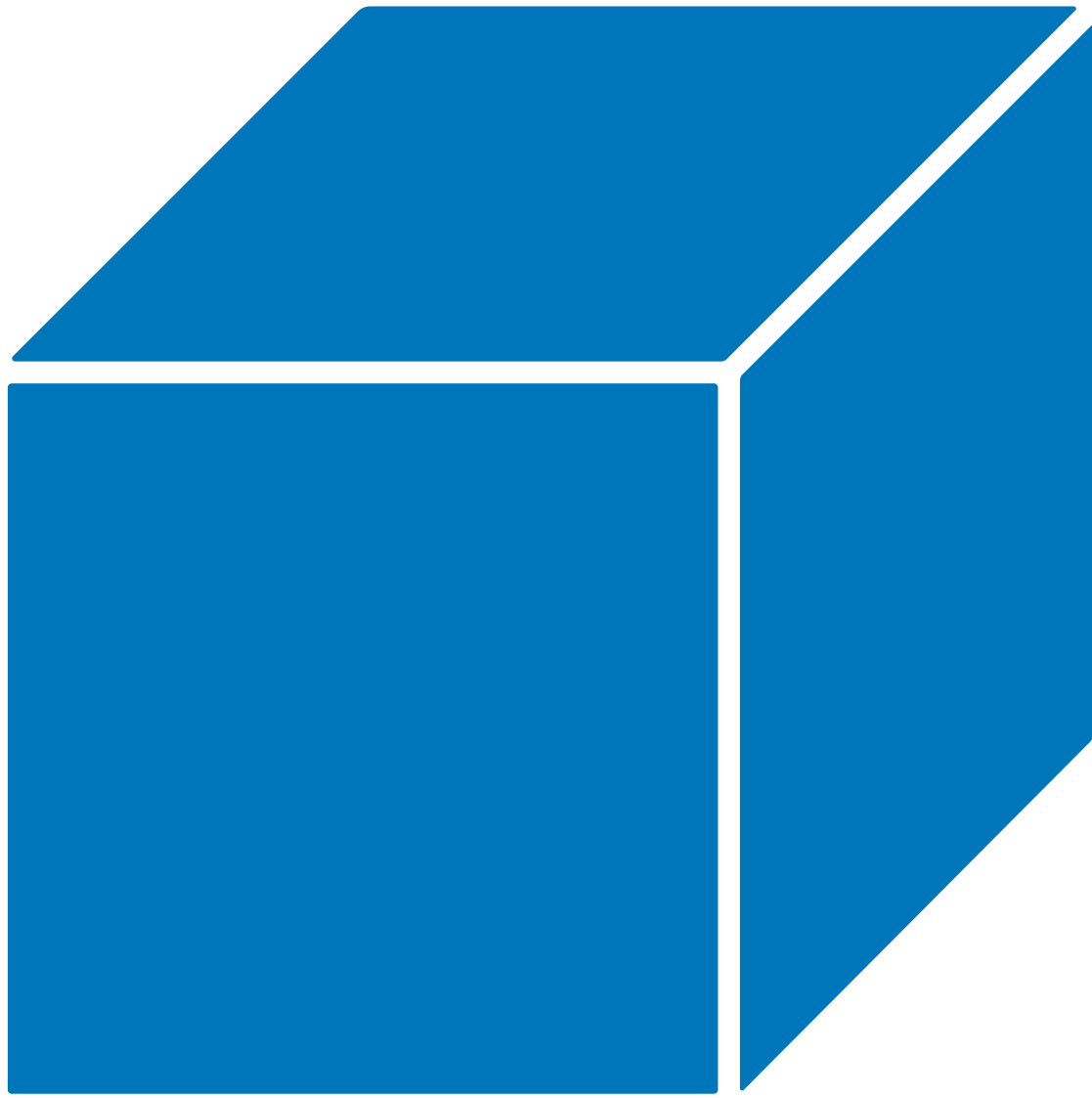


Efficient Sampling



Implicit

Compact Visual Representations



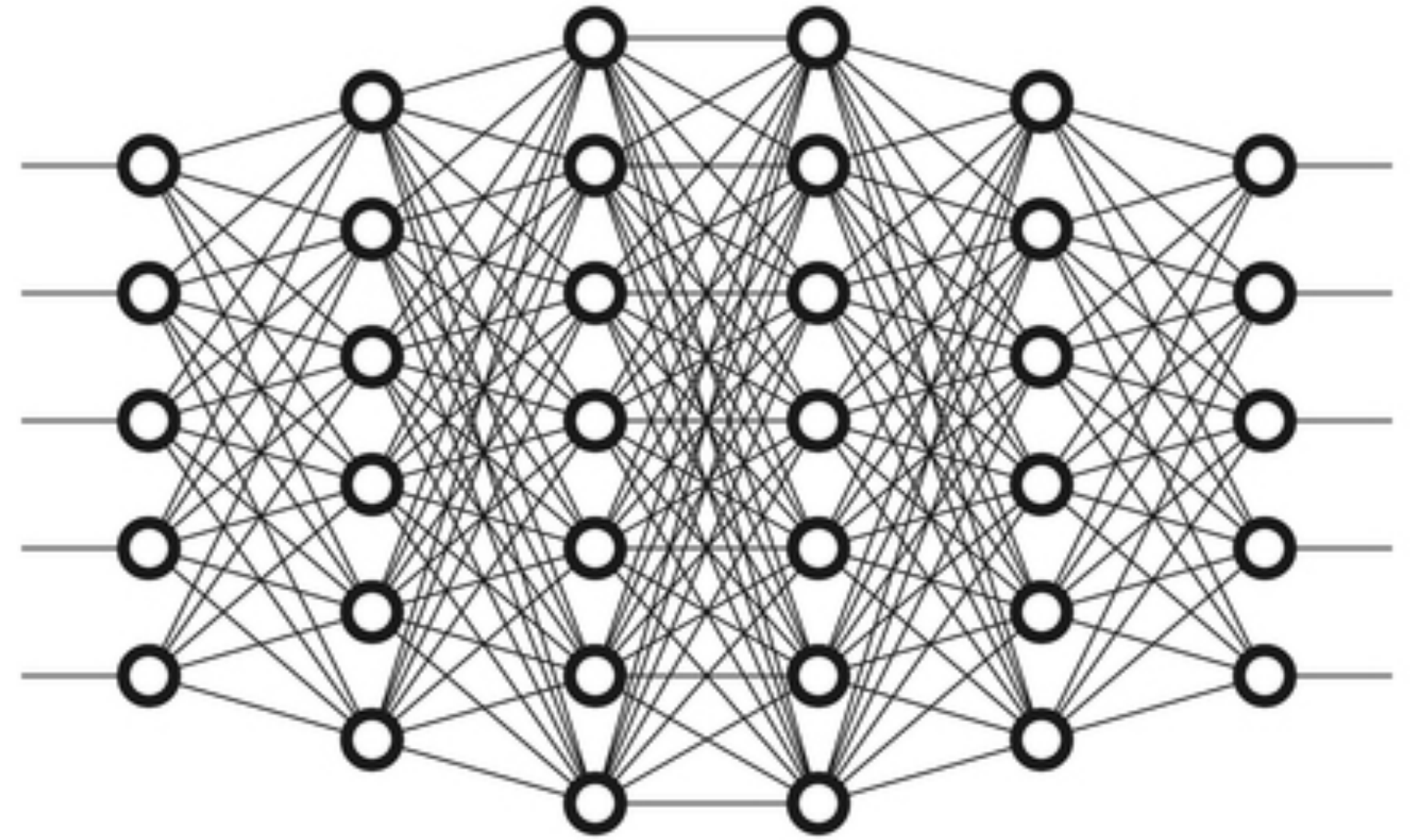
Explicit



High Memory



Efficient Sampling



Implicit

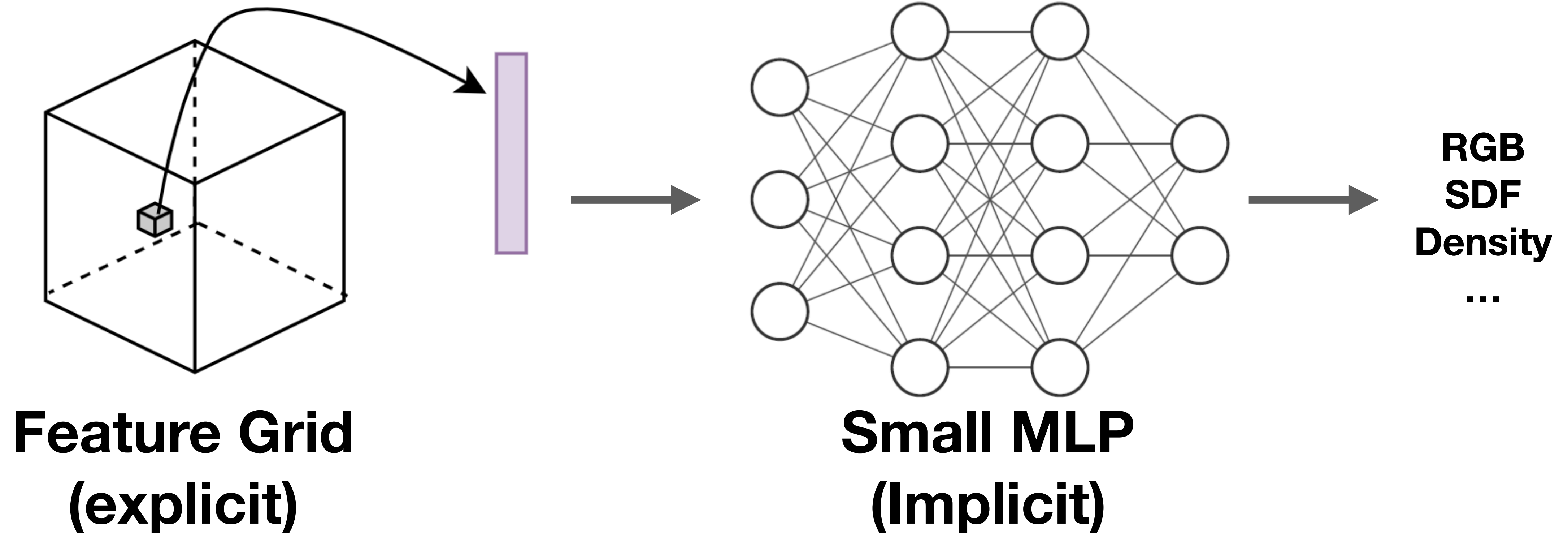


Low Memory

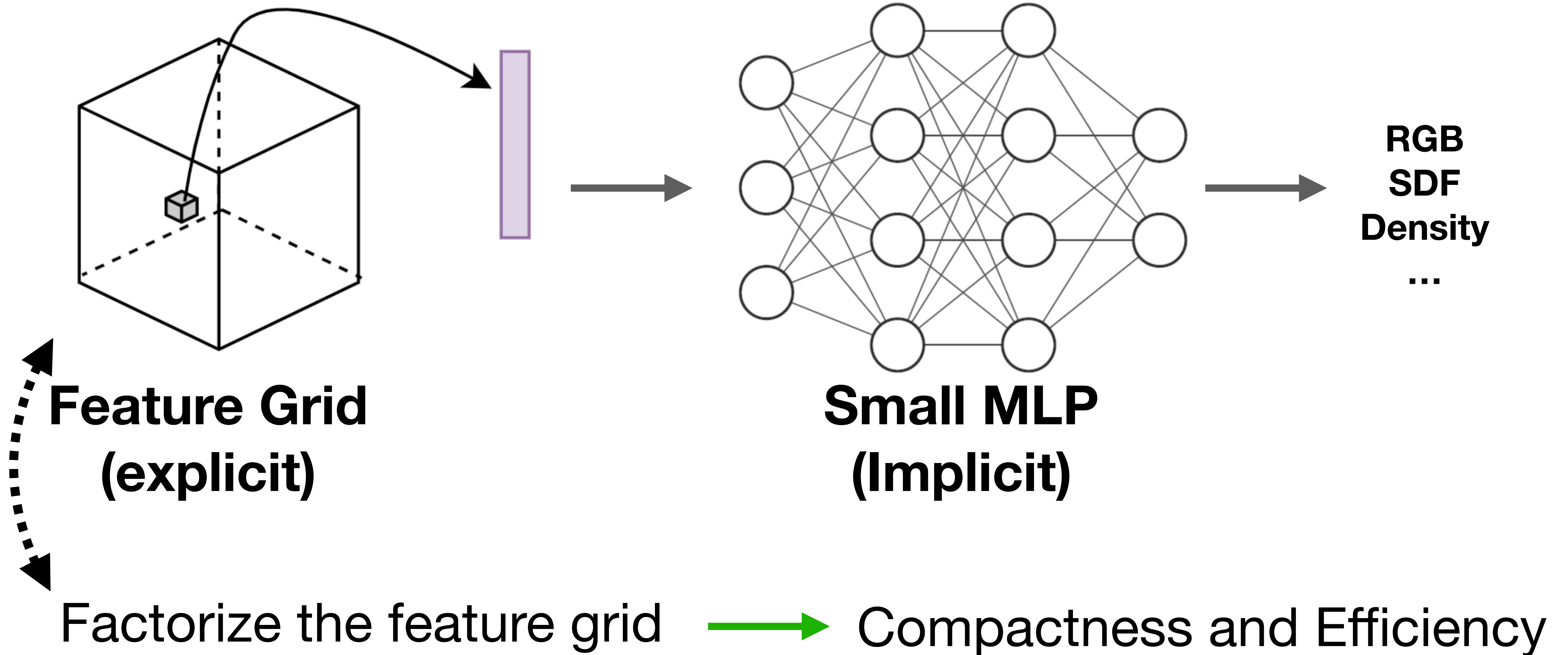


Slow Sampling

Explicit-Implicit Representations

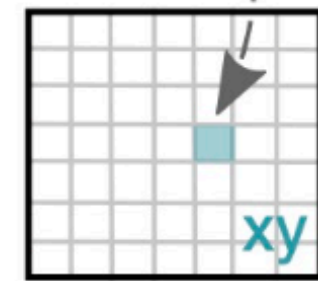


Explicit-Implicit Representations



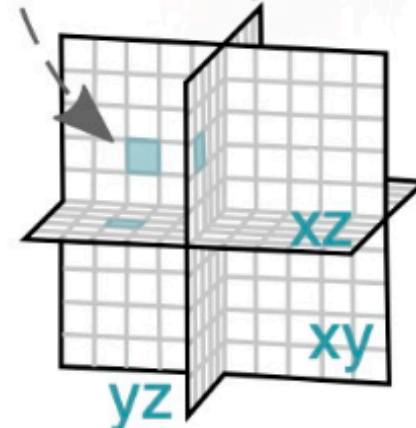
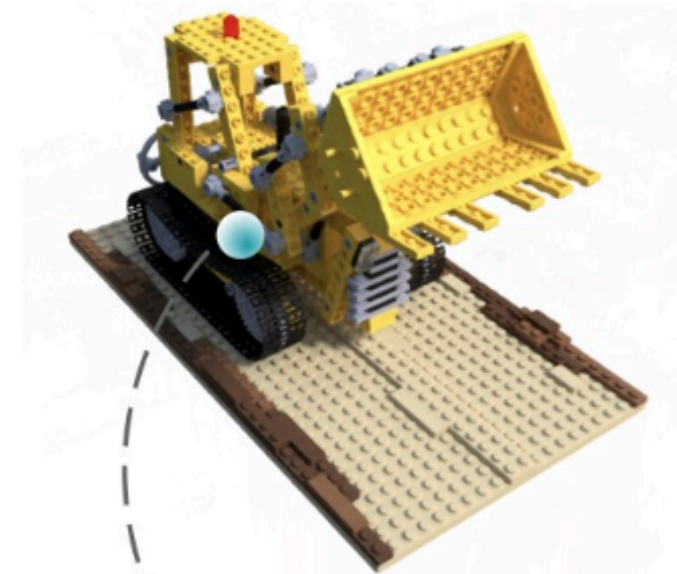
Factorized Representations

K-Planes
(Fridovich-Keil et al., 2023)



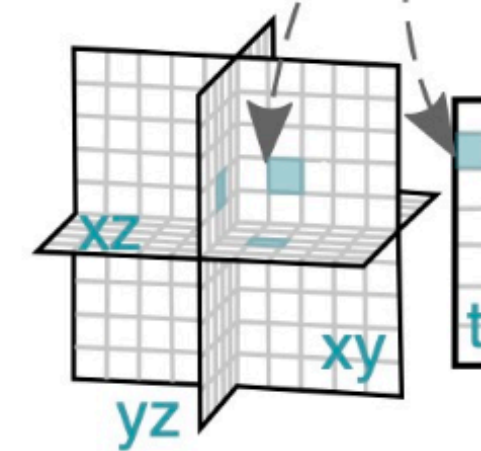
Memory: $\mathcal{O}(N^2)$

a) 2D Images



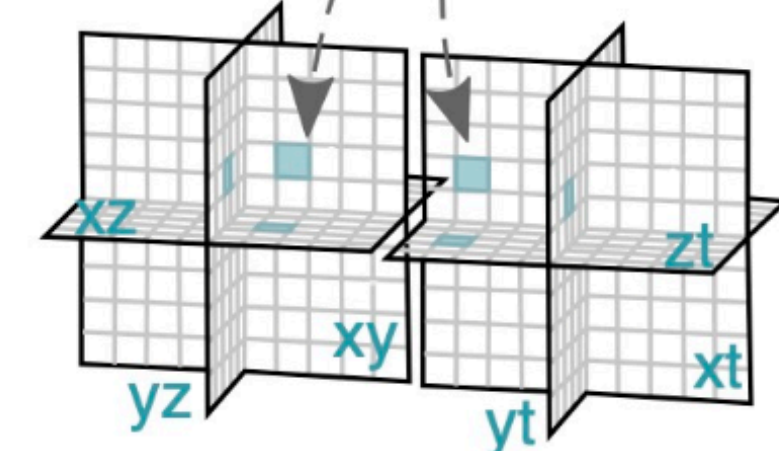
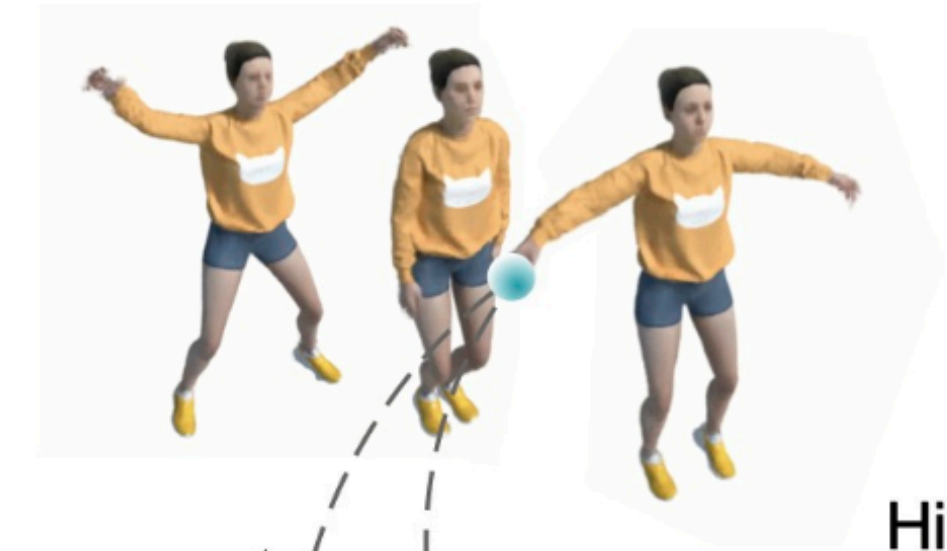
$\mathcal{O}(3N^2)$

b) 3D Static Volumes



$\mathcal{O}(3N^2 + N)$

c) 3D Volume w. Varying Appearance



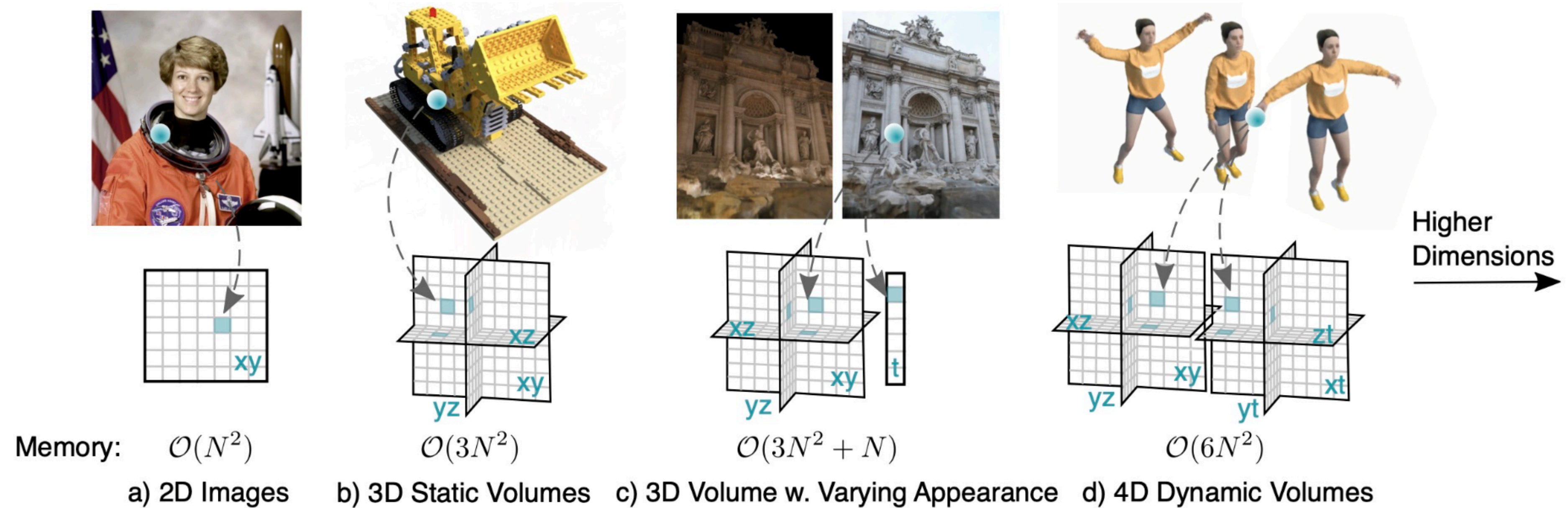
$\mathcal{O}(6N^2)$

d) 4D Dynamic Volumes

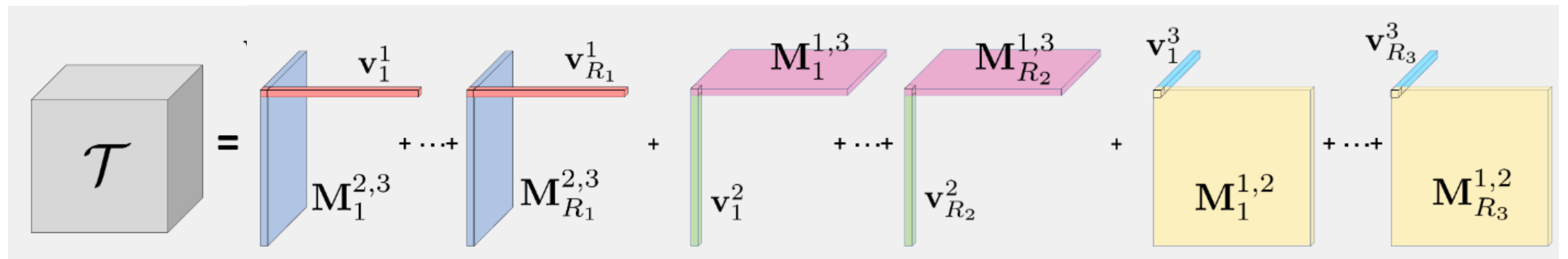
Higher
Dimensions
→

Factorized Representations

K-Planes (Fridovich-Keil et al., 2023)

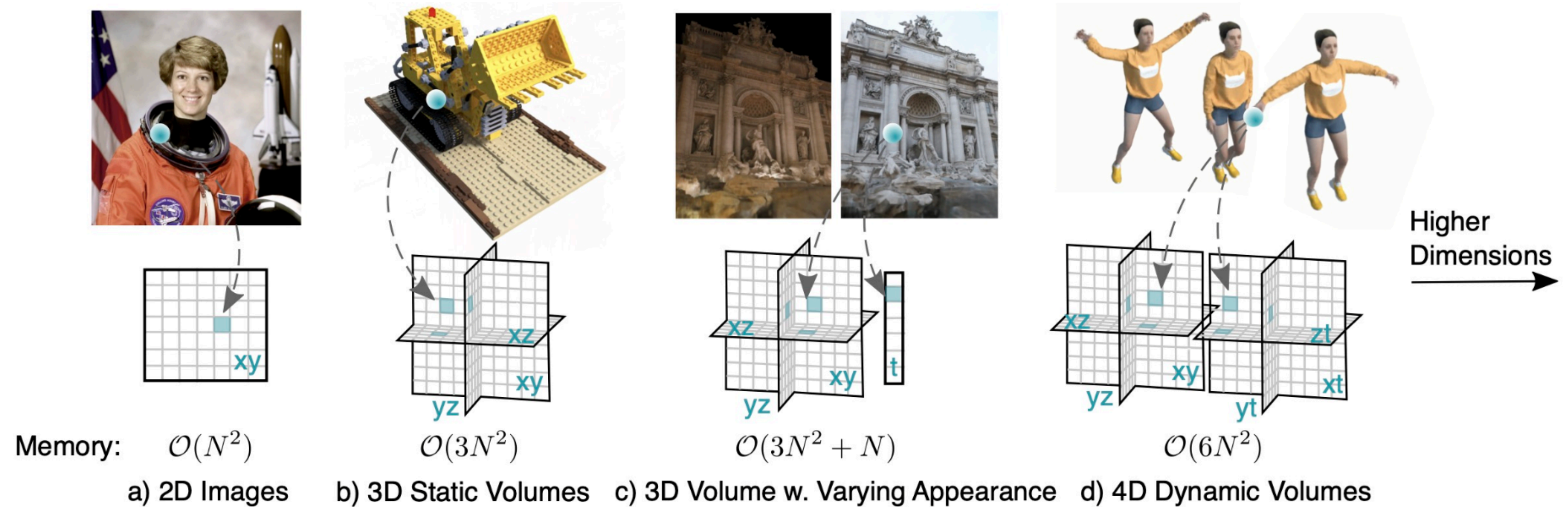


TensorRF (Chen et al., 2022)

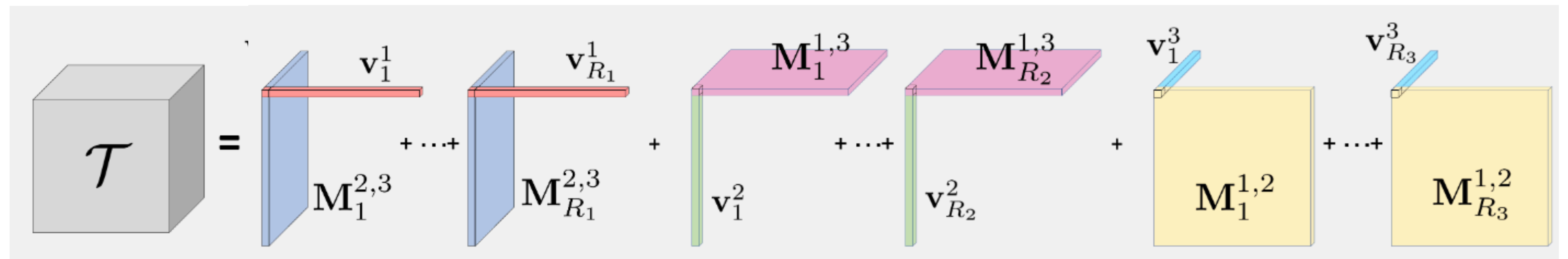


Factorized Representations

K-Planes (Fridovich-Keil et al., 2023)

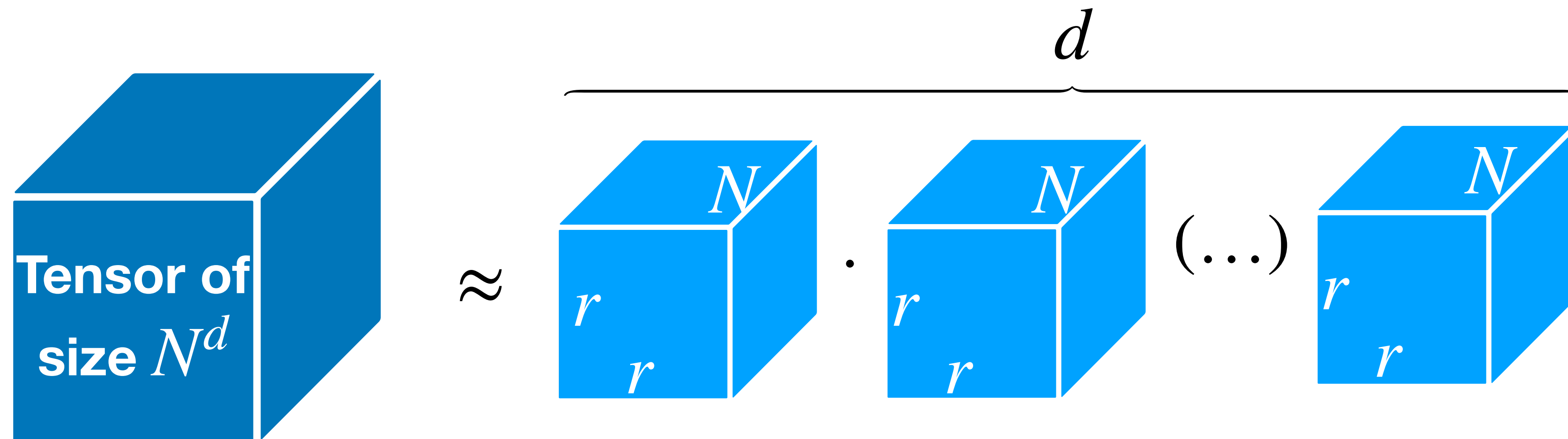


TensoRF (Chen et al., 2022)



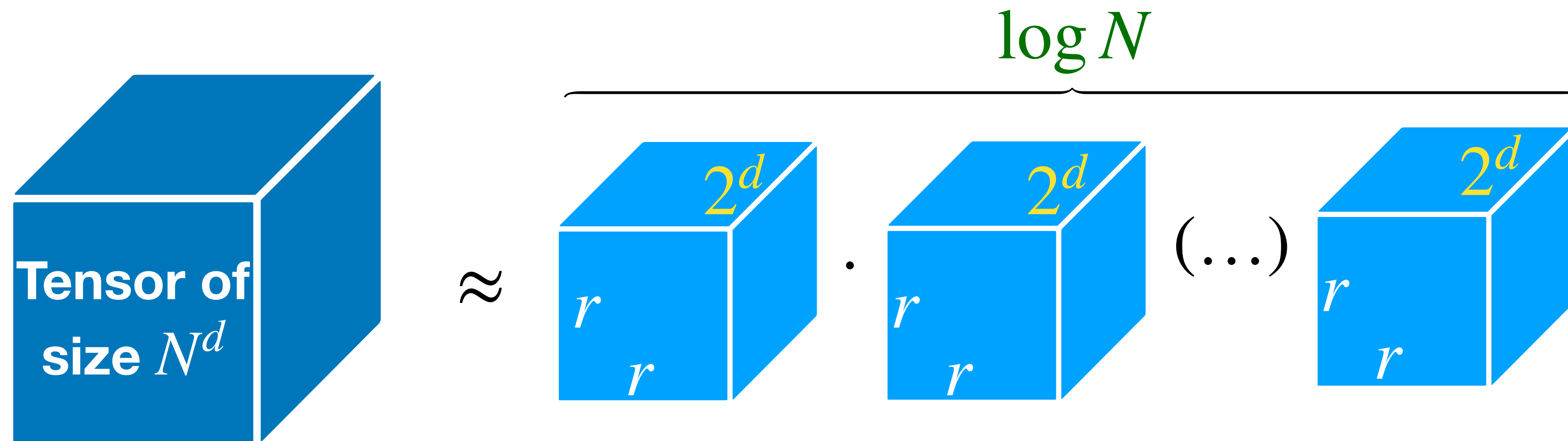
Both are **linear** in d , **quadratic** in N for a tensor N^d

Tensor Trains (TT)



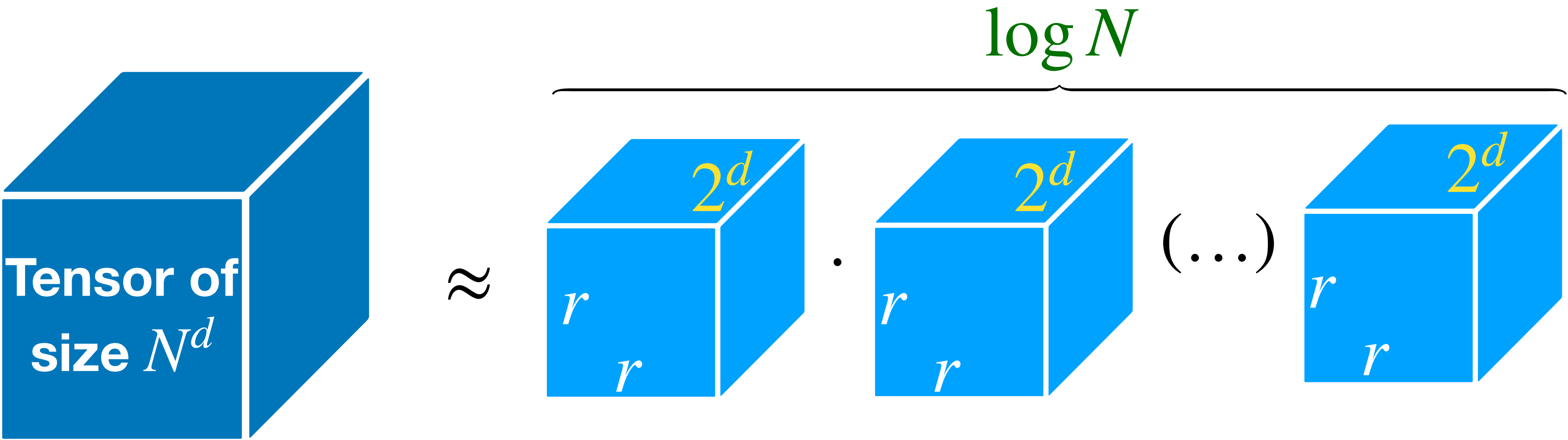
Linear in N and Linear in d - $O(dNr^2)$ parameters

Quantized Tensor Trains (QTT)



Logarithmic in N and exponential in d - $O(\log(N)2^d r^2)$ parameters

Quantized Tensor Trains (QTT)

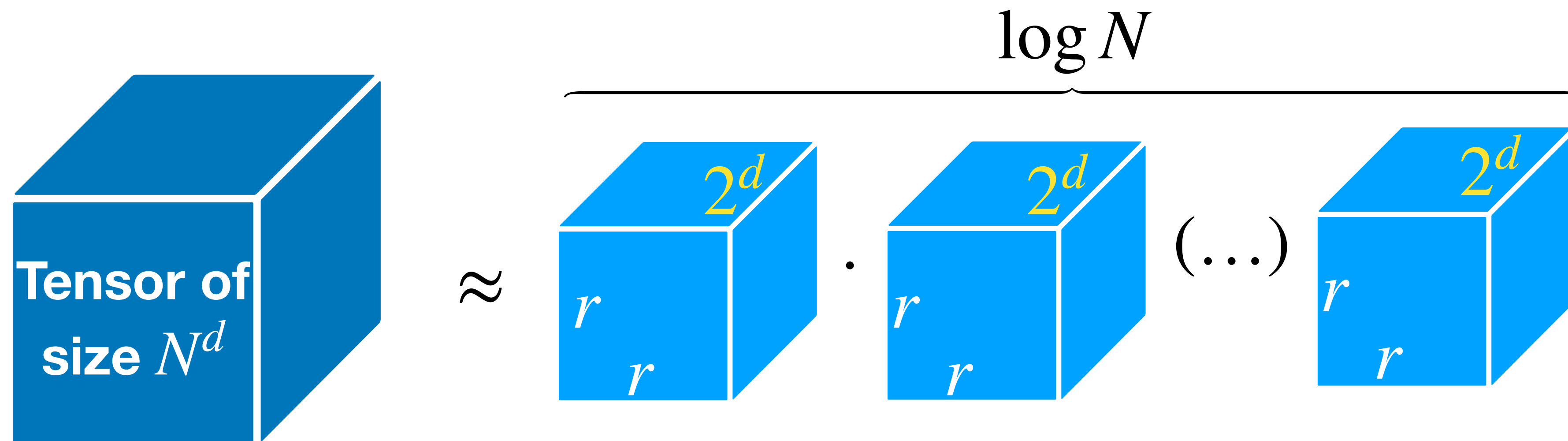


Logarithmic in N and exponential in d - $O(\log(N)2^d r^2)$ parameters

16k  16k

QTT: $\log(16 \cdot 10^3) \cdot 2^2 \cdot r^2 = 14 \cdot 4 \cdot r^2$

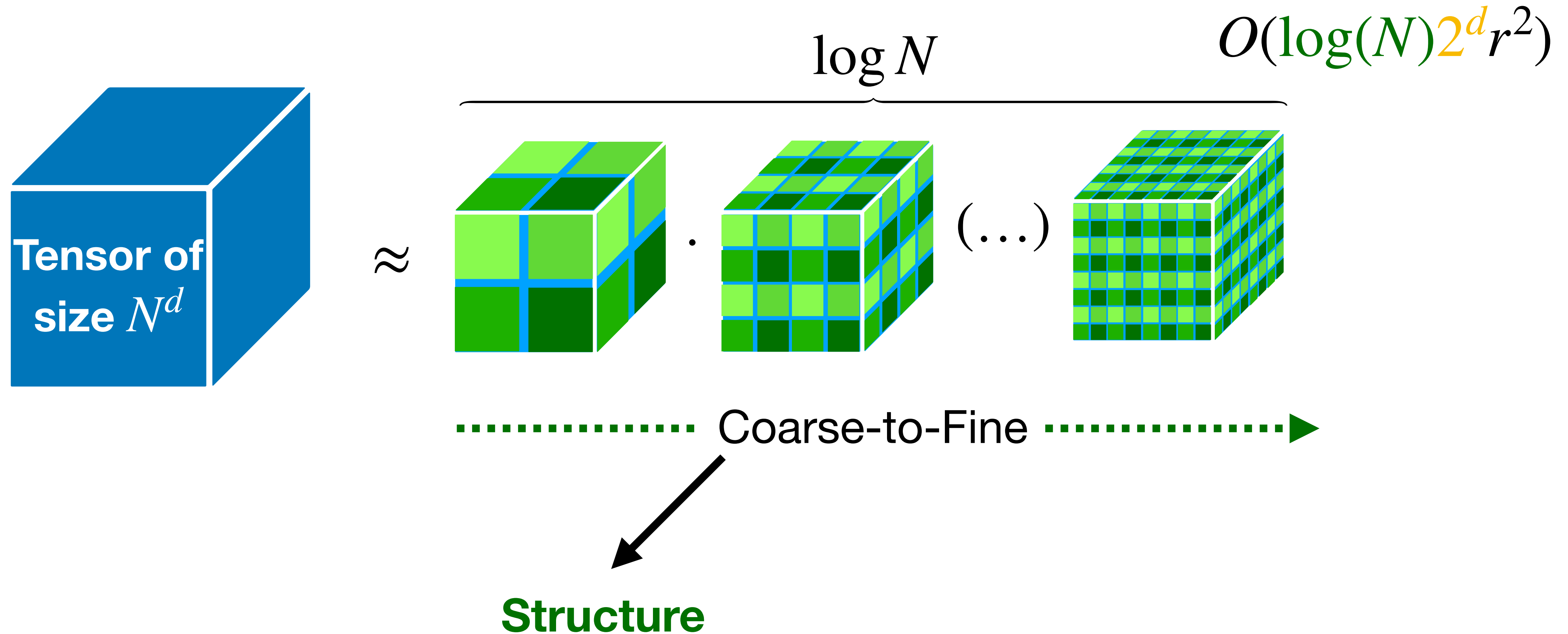
Quantized Tensor Trains (QTT)



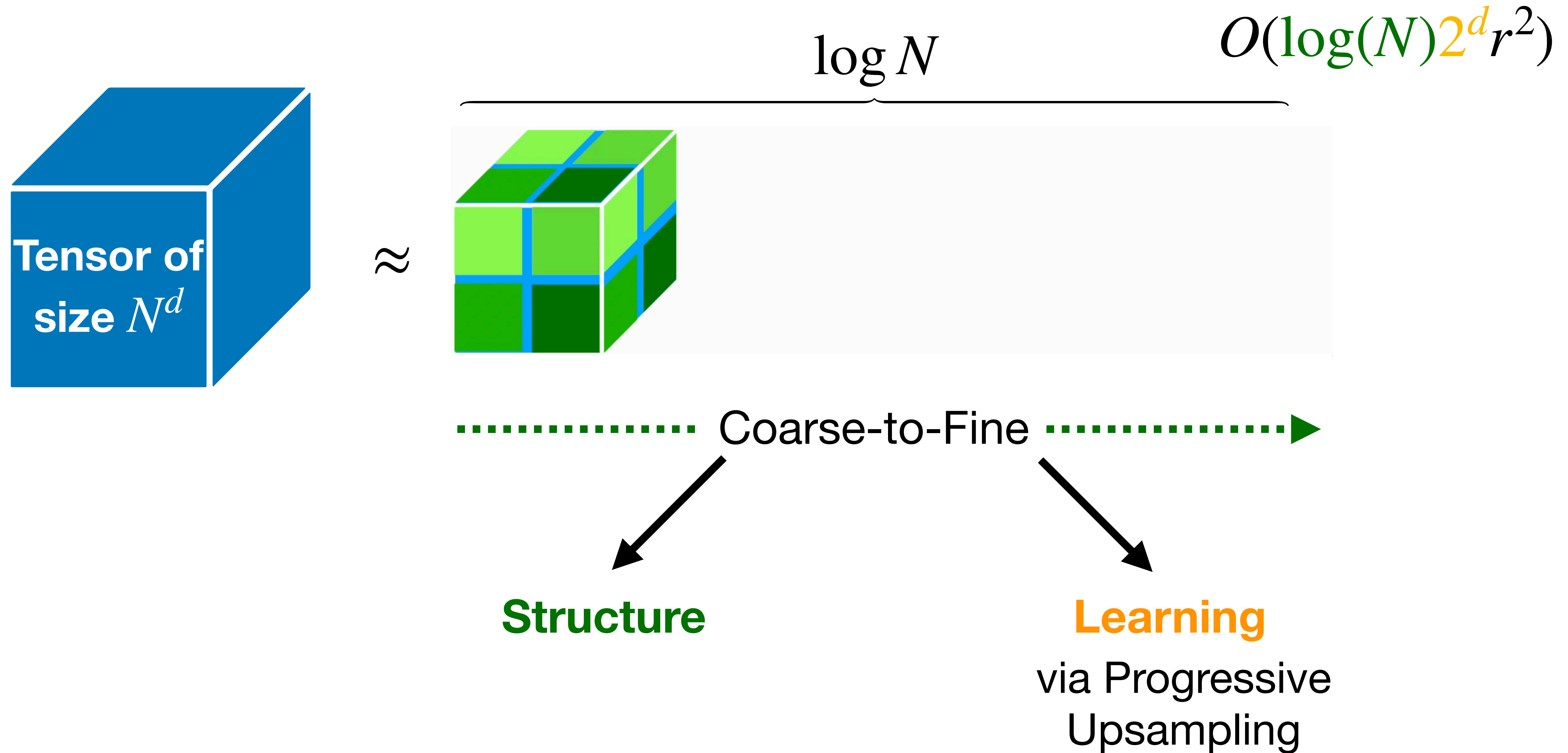
Logarithmic in N and exponential in d - $O(\log(N)2^d r^2)$ parameters

QTTs are **difficult to optimize** -
Obukhov et al. 2024

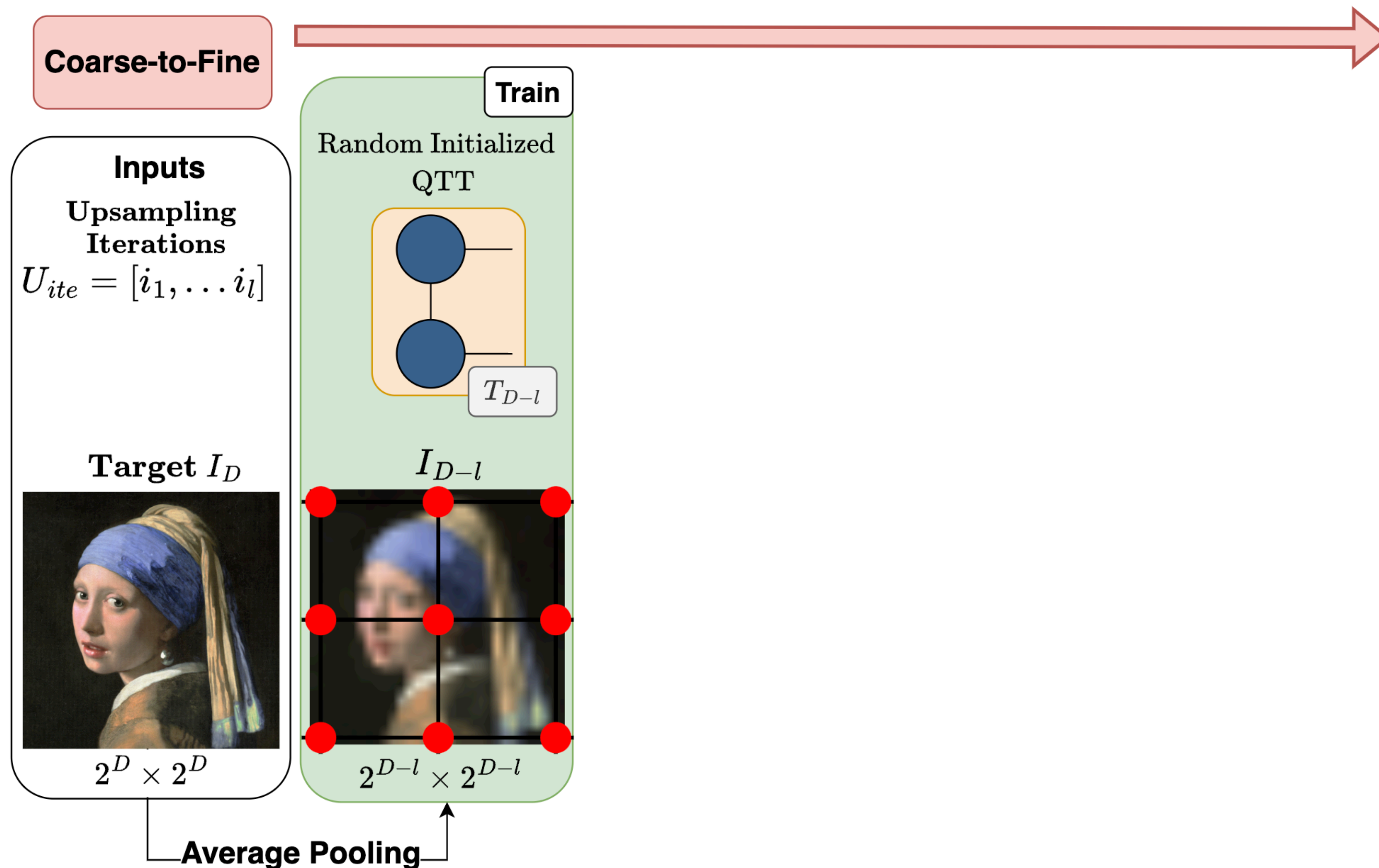
Coarse-to-fine Learning of QTTs



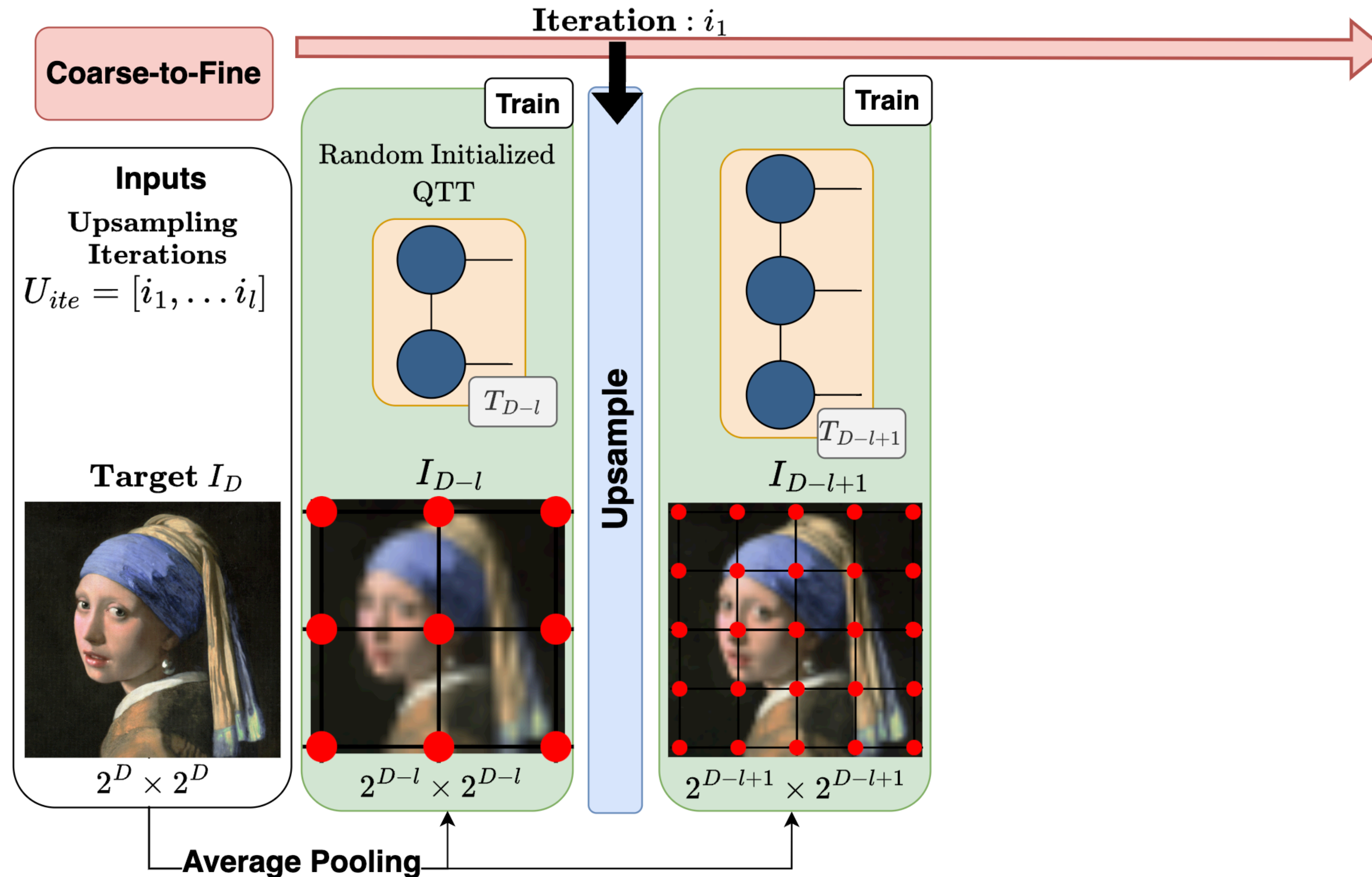
Coarse-to-fine Learning of QTTs



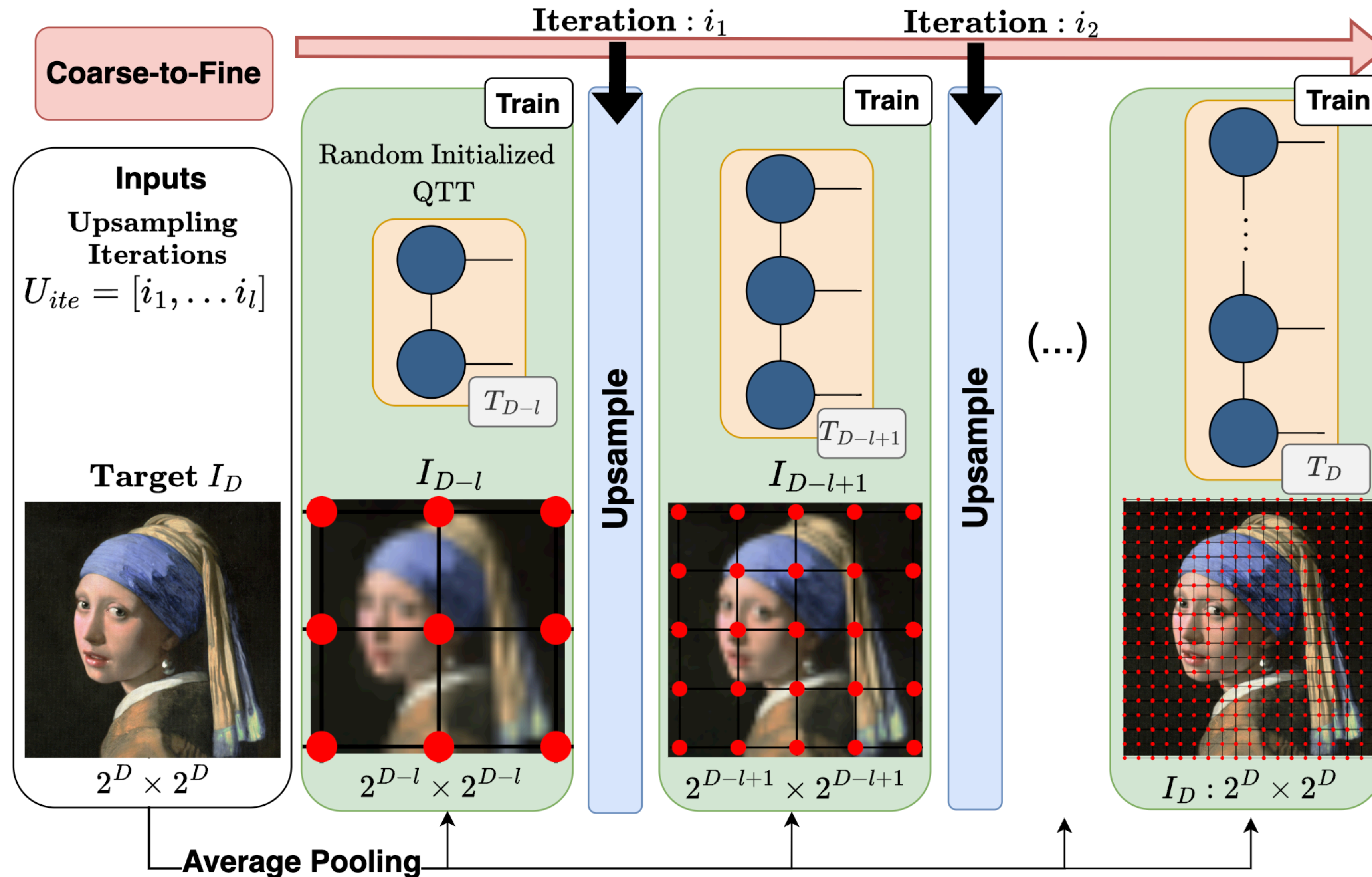
Coarse-to-fine Learning of QTTs



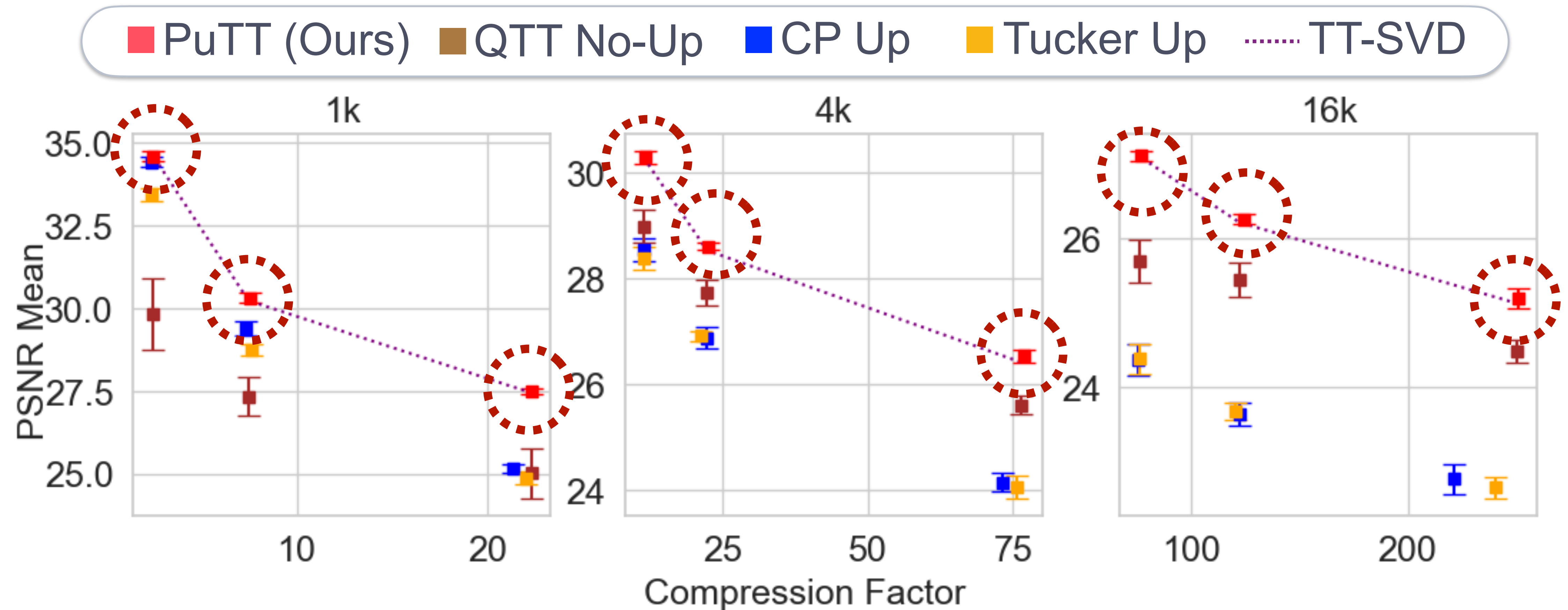
Coarse-to-fine Learning of QTTs



Coarse-to-fine Learning of QTTs



Results - 2D and 3D compression

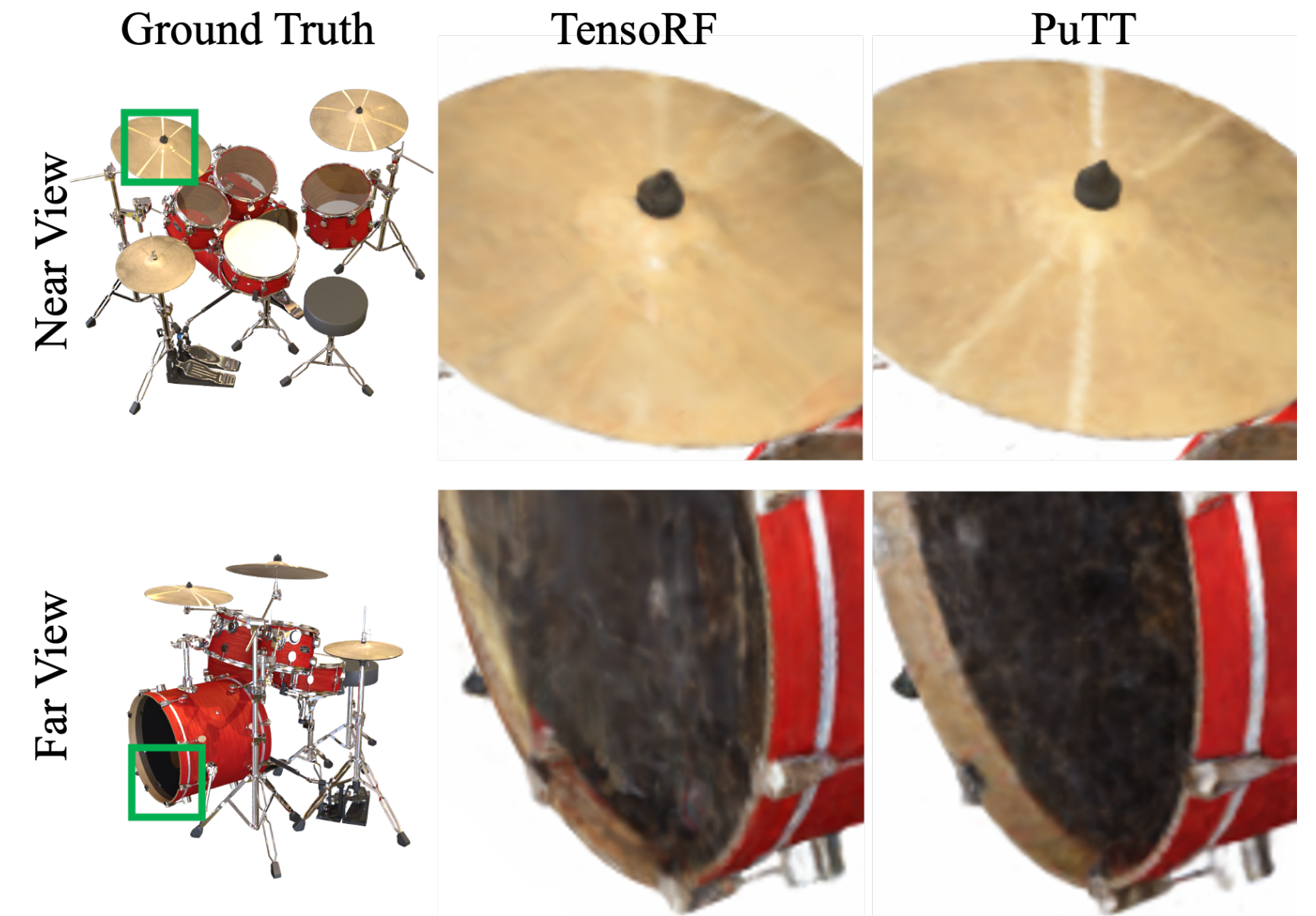


Novel View Synthesis

- Significantly better than all baselines for small model sizes
- Competitive with SOTA and even better on some benchmarks

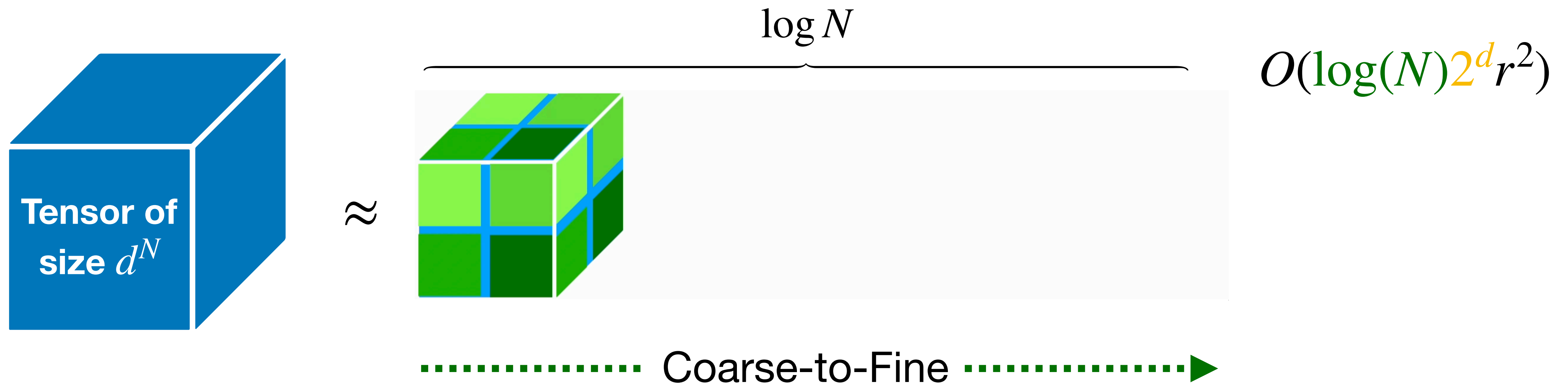
Method	Steps	Size(MB)↓	Synthetic-NeRF		NSVF		TanksTemples	
			PSNR↑	SSIM↑	PSNR↑	SSIM↑	PSNR↑	SSIM↑
PlenOctrees* (Yu et al., 2021b)-L	200k	1976.3	31.71	0.958	-	-	27.99	0.917
Plenoxels* (Yu et al., 2021a)-L	128k	778.1	31.71	0.958	-	-	27.43	0.906
DVGO* (Sun et al., 2022)-L	30k	612.1	31.95	0.957	35.08	0.975	28.41	0.911
K-planes* (Fridovich-Keil et al., 2023)-L	30k	122.1	32.36	0.962	-	-	-	-
Instant-NGP* (Müller et al., 2022)-L	35k	45.5	33.18	0.963	-	-	-	-
TensorRF (Chen et al., 2022) VM-192*-L	30k	71.8	33.14	0.963	36.52	0.982	28.56	0.920
Our PuTT-600-L	80k	60.3	32.79	0.958	36.57	0.982	28.37	0.917
NeRF* (Mildenhall et al., 2020)-S	300k	5.0	31.01	0.947	30.81	0.952	25.78	0.864
TT-NF * (Obukhov et al., 2022)-S	80k	7.9	31.09	0.945	-	-	-	-
TensorRF CP-384*-S	30k	3.9	31.56	0.949	34.48	0.971	27.59	0.897
TensorRF VM-30 (no shrinking)-S	80k	6.9	31.10	0.948	34.57	0.974	27.67	0.898
Our PuTT-200-S	80k	6.7	31.66	0.956	35.58	0.976	27.99	0.901
TensorRF VM-48 (no shrinking)-M	80k	12.0	31.62	0.952	35.44	0.976	28.03	0.901
Our PuTT-280-M	80k	12.0	31.95	0.957	36.04	0.977	28.15	0.905

Table 1. Comparison to baselines for the task of novel view synthesis for the Synthetic-NeRF (Mildenhall et al., 2020), NSVF (Liu et al., 2020a), and TanksTemples (Knapitsch et al., 2017) datasets. Scores of the baseline methods with a * are taken directly from corresponding papers whenever possible. L, M and S indicate a Large (> 60 MB), Medium (12 MB) or small (< 8 MB) model size. We compare our method on Medium and Small-sized models but also include Large-sized models for reference.



Key Takeaway

- Quantized Tensor Trains learned in a coarse-to-fine manner are a parameter-efficient and effective way of learning compact visual representations.





LoQT: Low-Rank Adapters for Quantized Pretraining

Sebastian Loeschcke*, Mads Tofttrup*, Michael J. Kastoryano,
Serge Belongie, and Vésteinn Snæbjarnarson

NeurIPS 2024

UNIVERSITY OF
COPENHAGEN



AARHUS UNIVERSITY



PIONEER CENTRE FOR
ARTIFICIAL INTELLIGENCE

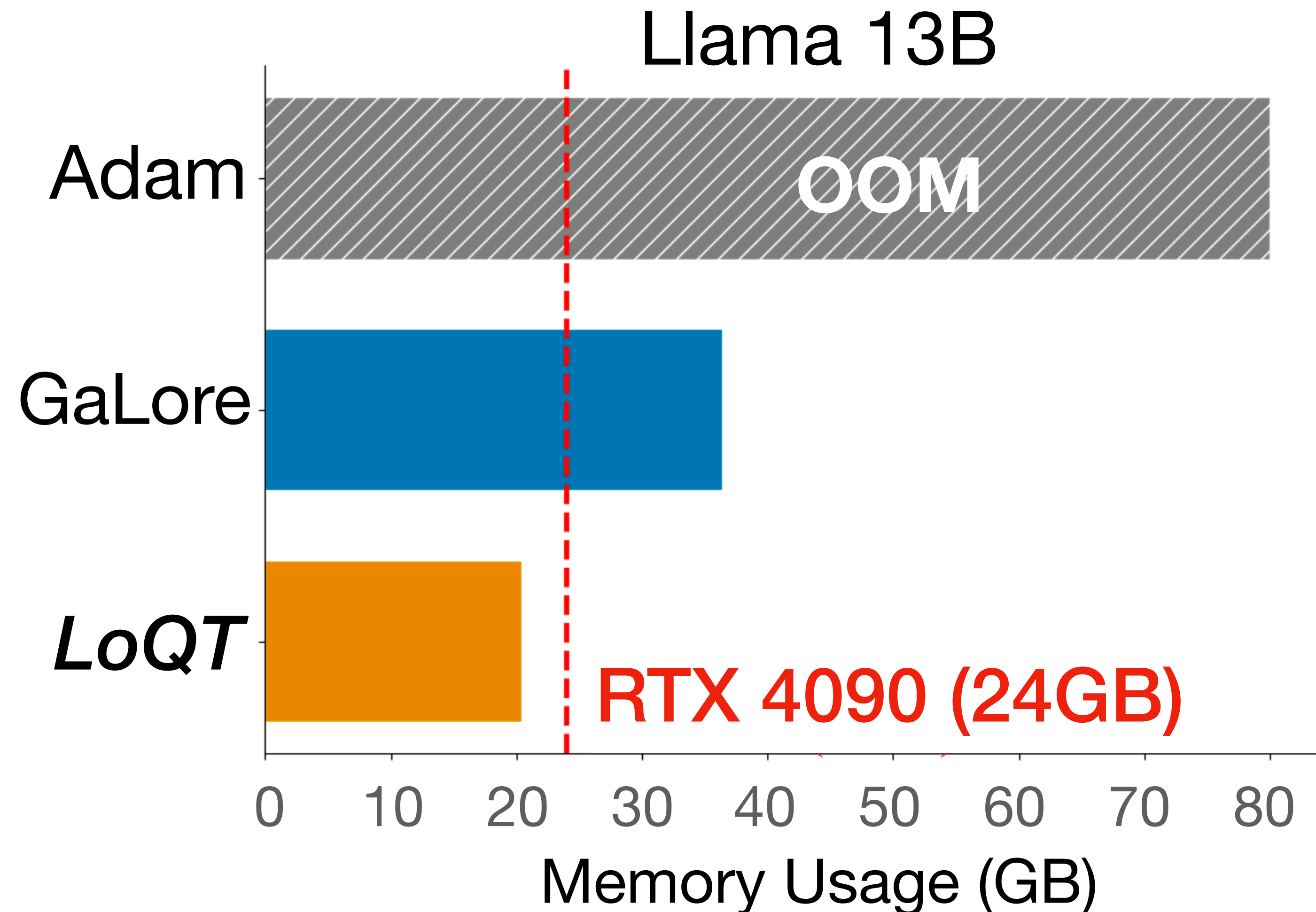
**Equal Contribution*

Pretraining LLMs require big GPU clusters

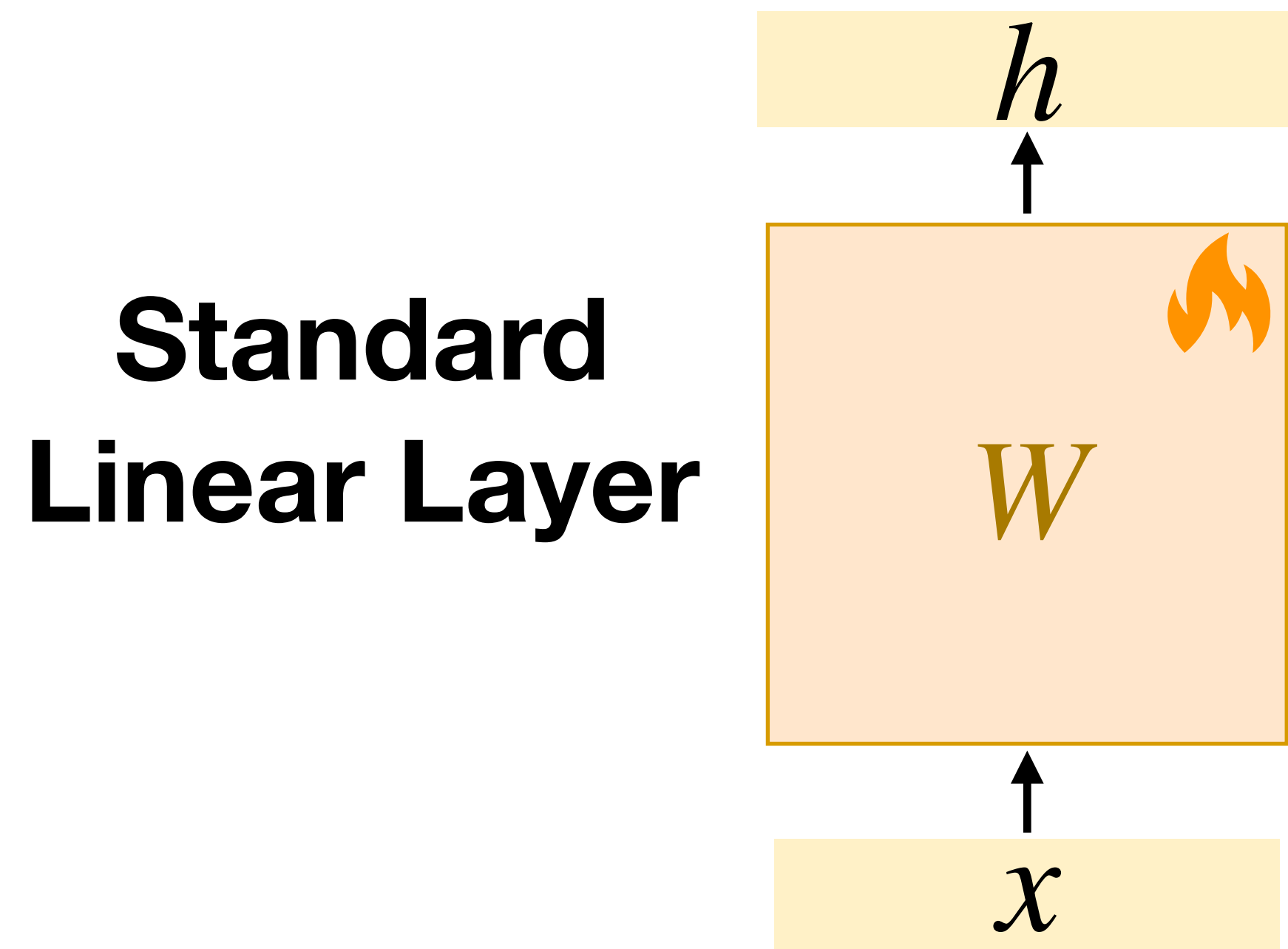
Pretraining a **LLaMA 13B** model with Adam and a *batch size of one* requires over **100 GB** of memory.

Pretraining LLMs require big GPU clusters

Pretraining a **LLaMA 13B** model with Adam and a *batch size of one* requires over **100 GB** of memory.

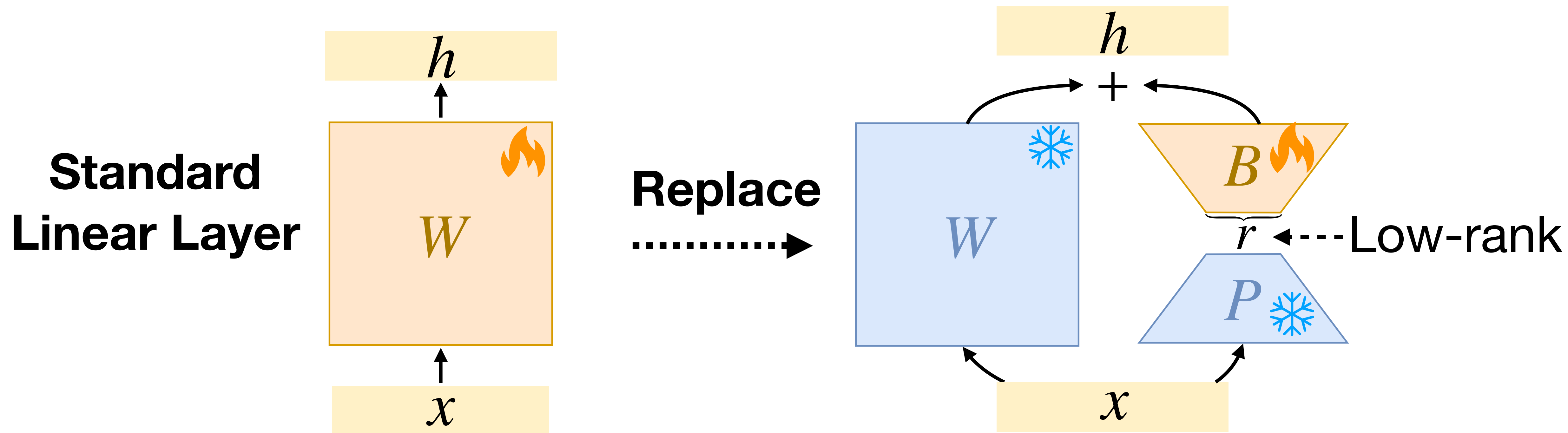


LoQT: Low-Rank Quantized Pretraining



 Optimize  Freeze

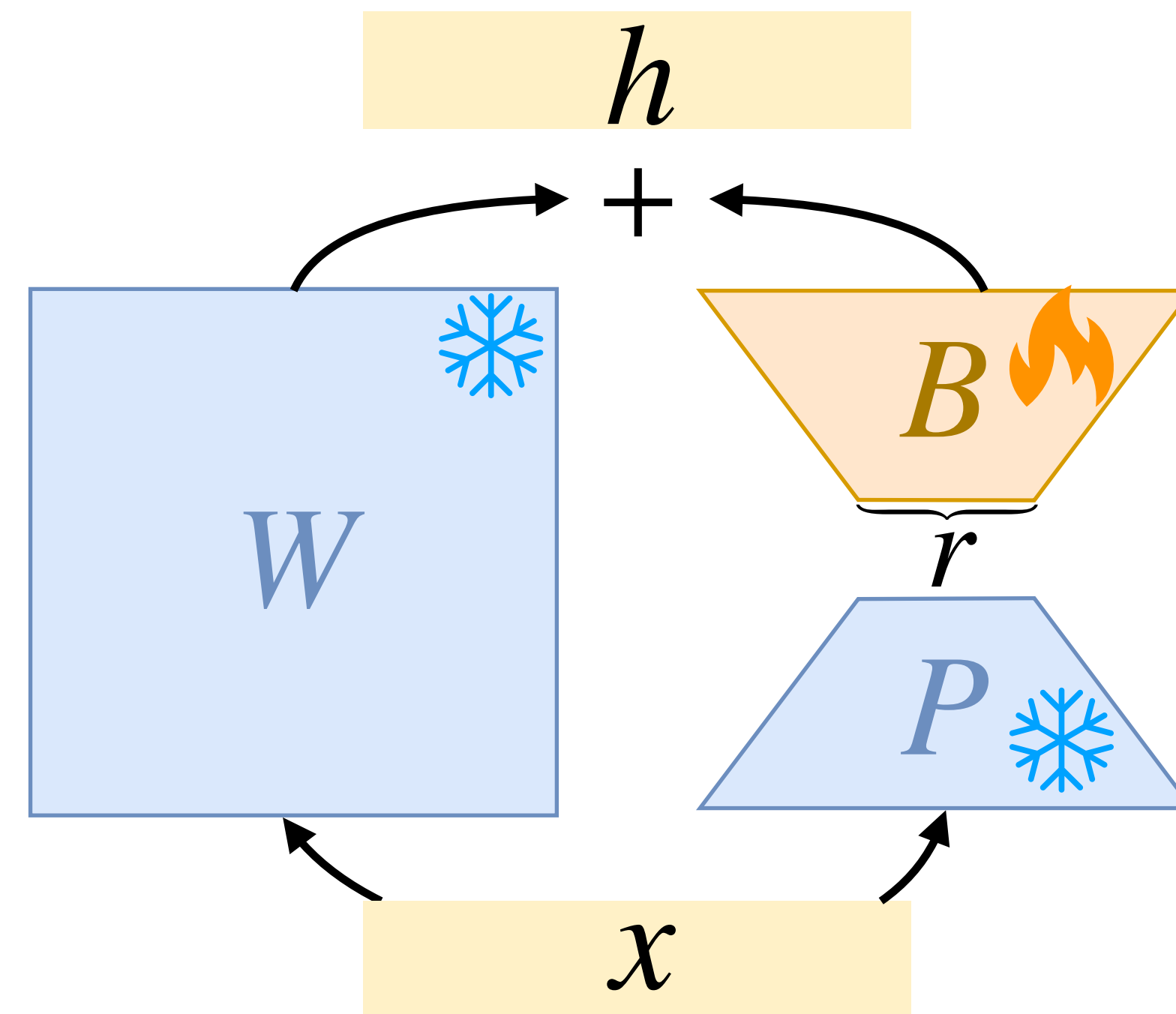
LoQT: Low-Rank Quantized Pretraining



🔥 Optimize ❄️ Freeze

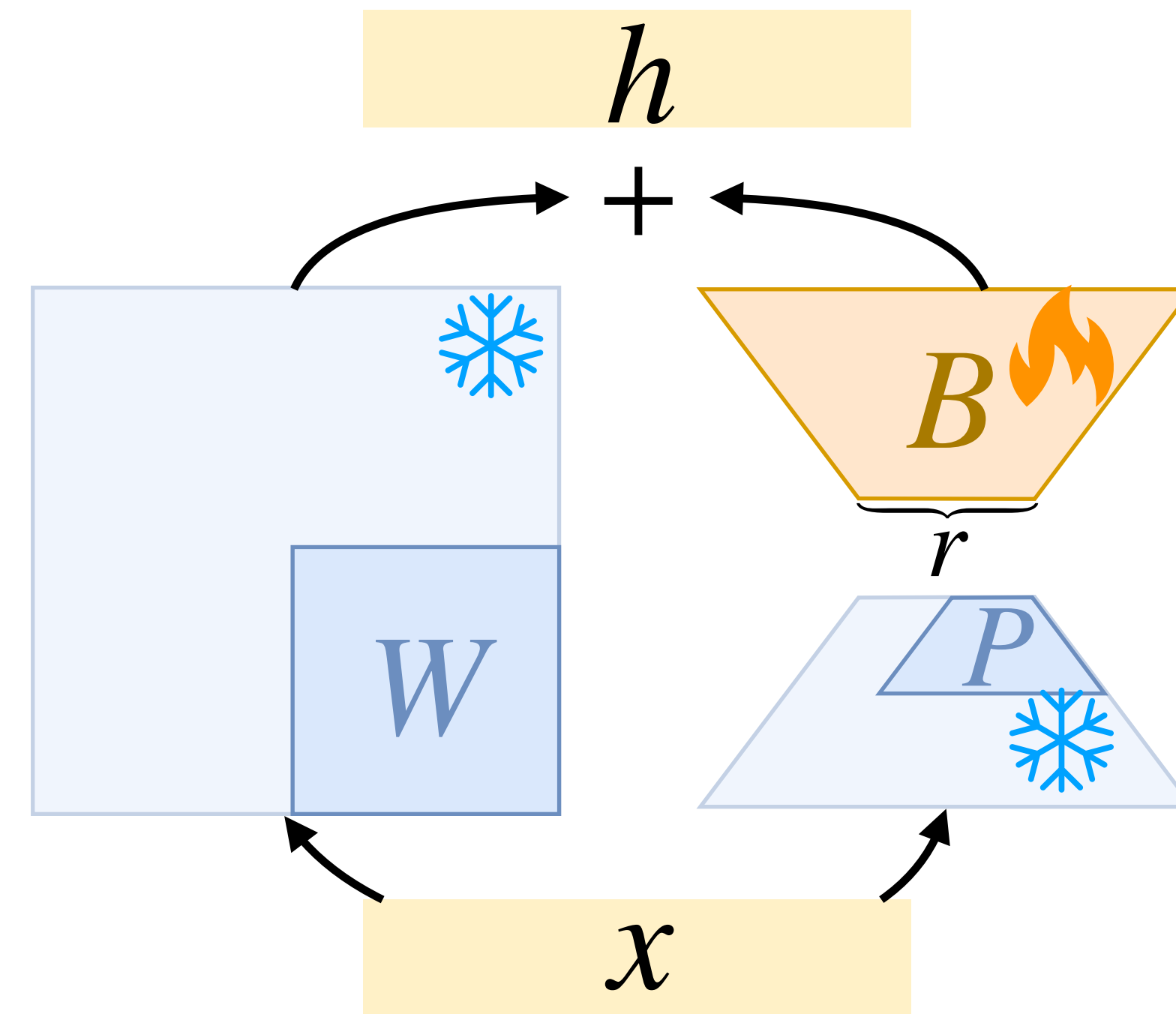
LoQT: Low-Rank Quantized Pretraining

- **Pretraining from scratch** while only *optimizing a **single low-rank factor per layer***.



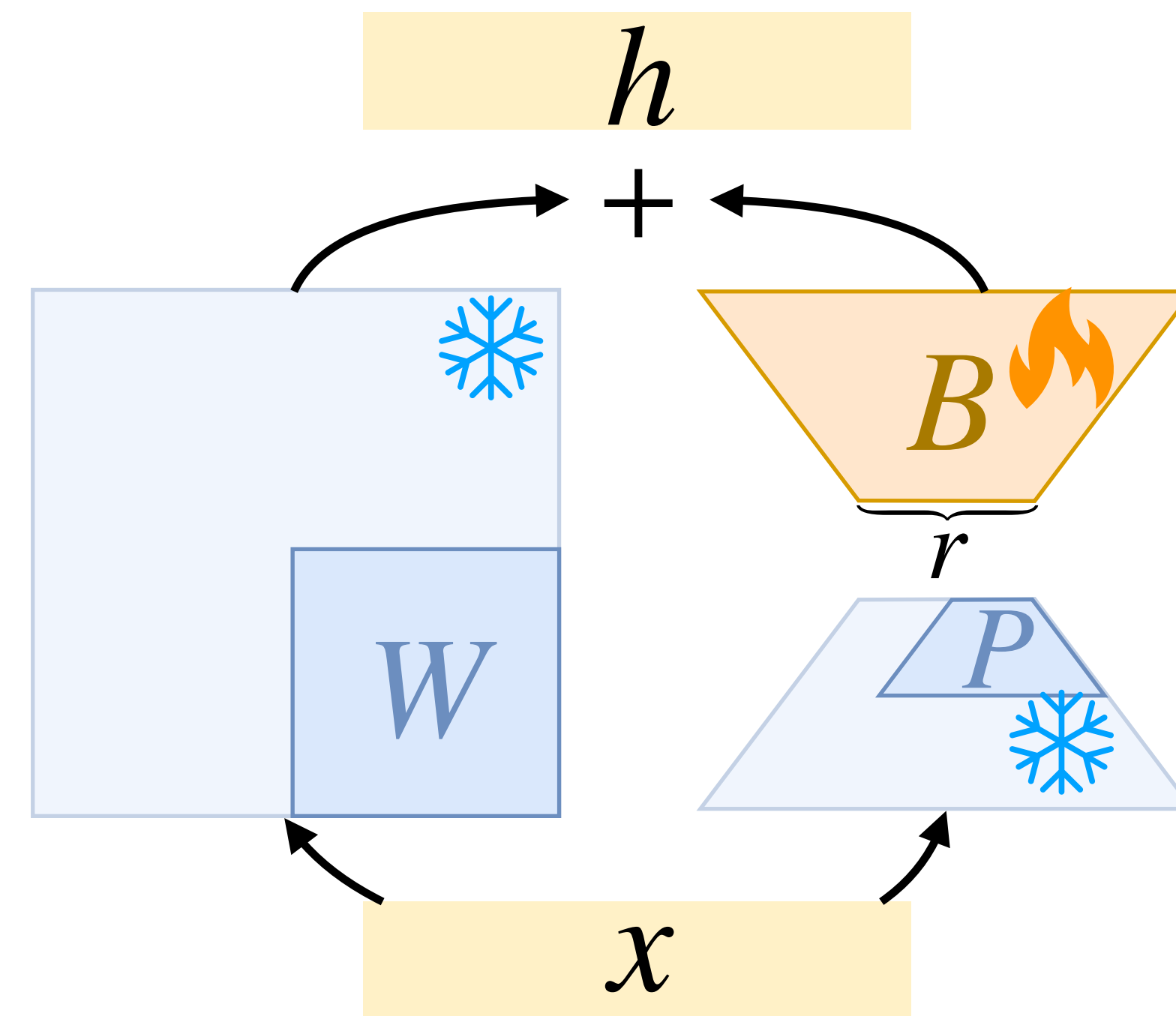
LoQT: Low-Rank Quantized Pretraining

- **Pretraining from scratch** while only *optimizing a **single low-rank factor** per layer.*
- Keep most *weights in **4-bit precision**.*



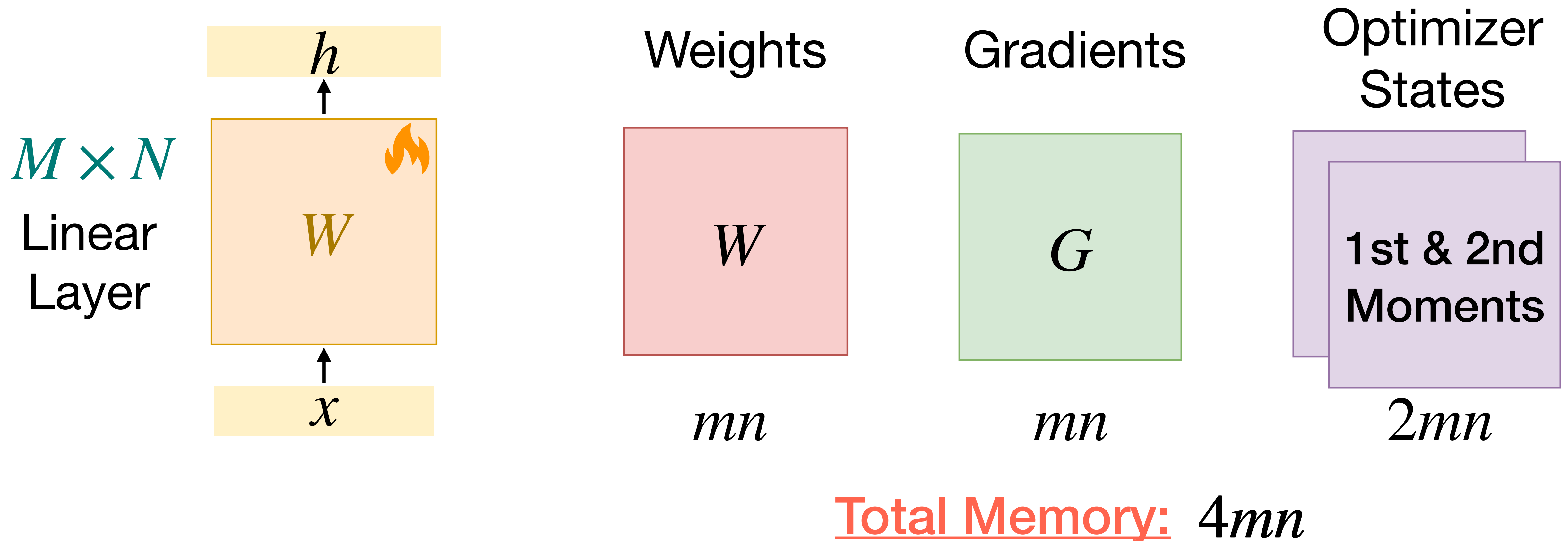
LoQT: Low-Rank Quantized Pretraining

- **Pretraining from scratch** while only *optimizing a **single low-rank factor** per layer.*
- Keep most *weights in **4-bit precision.***
- Matches regular BF16 training

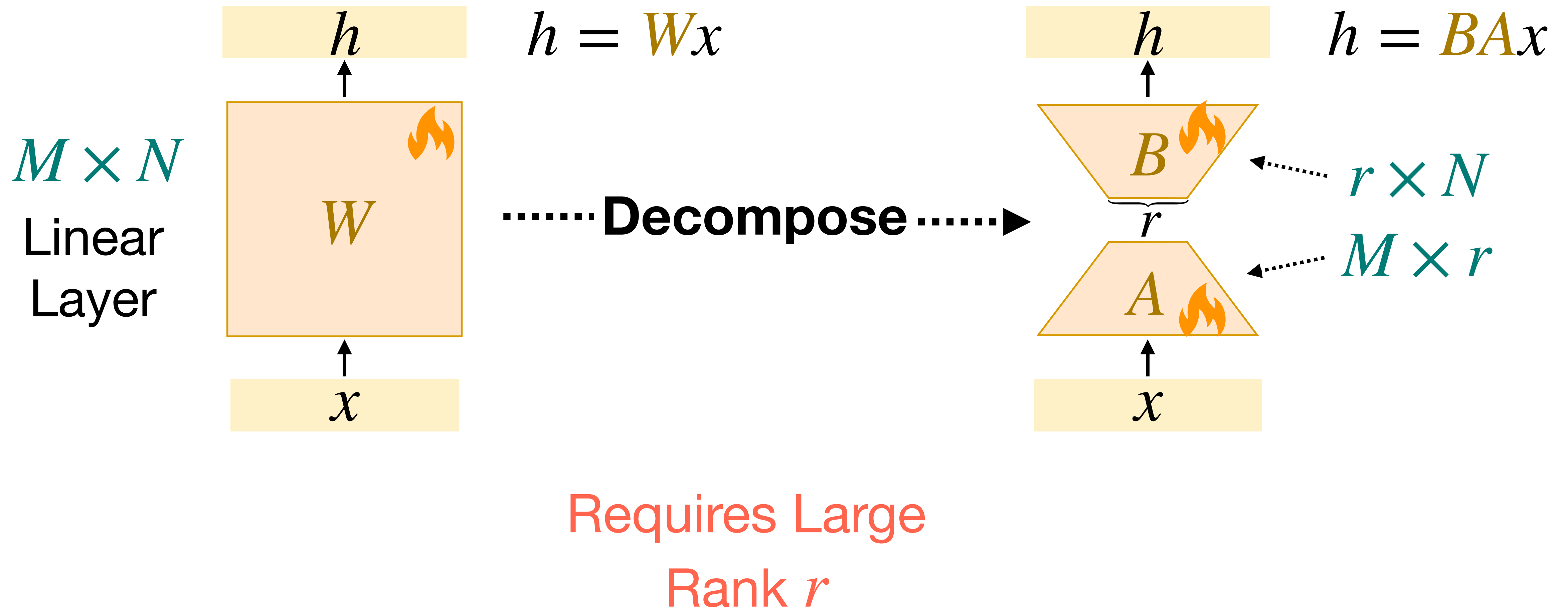


Training with the Adam Optimizer

Memory Requirements



Low-Rank Training



Low-Rank Properties of Gradients

- **Adam** (Kingma et al. 2014)

$$\boxed{W_t} = \boxed{W_{t-1}} - \rho \left(\overset{M \times N}{\boxed{G_{W_{t-1}}}} \right)$$

Optimizer

Low-Rank Properties of Gradients

- **Adam** (Kingma et al. 2014)

$$W_t = W_{t-1} - \rho \left(\overset{M \times N}{G_{W_{t-1}}} \right)$$

- **GaLore** (Zhao et al., 2024)

$$\rho \left(\overset{r \times N}{P} \overset{M \times N}{G_{W_{t-1}}} \right) \downarrow \hat{G}_{t-1} \quad r \times M$$

Low-Rank Properties of Gradients

- **Adam** (Kingma et al. 2014)

$$W_t = W_{t-1} - \rho \left(\overset{M \times N}{G_{W_{t-1}}} \right)$$

- **GaLore** (Zhao et al., 2024)

$$W_t = W_{t-1} - \overset{N \times r}{P^T} \rho \left(\overset{r \times N}{P} \overset{M \times N}{G_{W_{t-1}}} \right)$$

↓

$$\overset{r \times M}{\hat{G}_{t-1}}$$

Low-Rank Properties of Gradients

- **Adam** (Kingma et al. 2014)

$$W_t = W_{t-1} - \rho \left(\overset{M \times N}{G_{W_{t-1}}} \right)$$

- **GaLore** (Zhao et al., 2024)

$$W_t = W_{t-1} - \overset{N \times r}{P^T} \rho \left(\overset{r \times N}{P} \overset{M \times N}{G_{W_{t-1}}} \right)$$

Saves memory for
optimizer states

↓

$$\hat{G}_{t-1} \quad r \times M$$

Low-Rank Properties of Gradients

- **Adam** (Kingma et al. 2014)

$$W_t = W_{t-1} - \rho \left(\overset{M \times N}{G_{W_{t-1}}} \right)$$

- **GaLore** (Zhao et al., 2024)

$$W_t = W_{t-1} - \overset{N \times r}{P^T} \rho \left(\overset{r \times N}{P} \overset{M \times N}{G_{W_{t-1}}} \right)$$

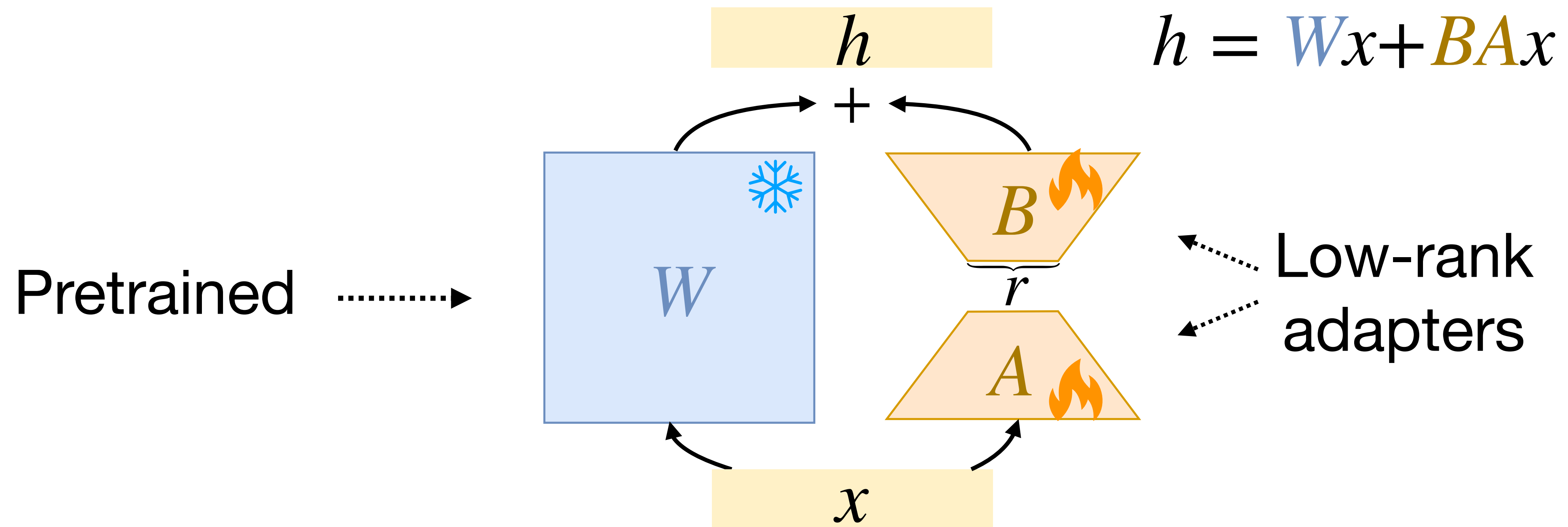
Saves memory for
optimizer states

No savings for
 W and G

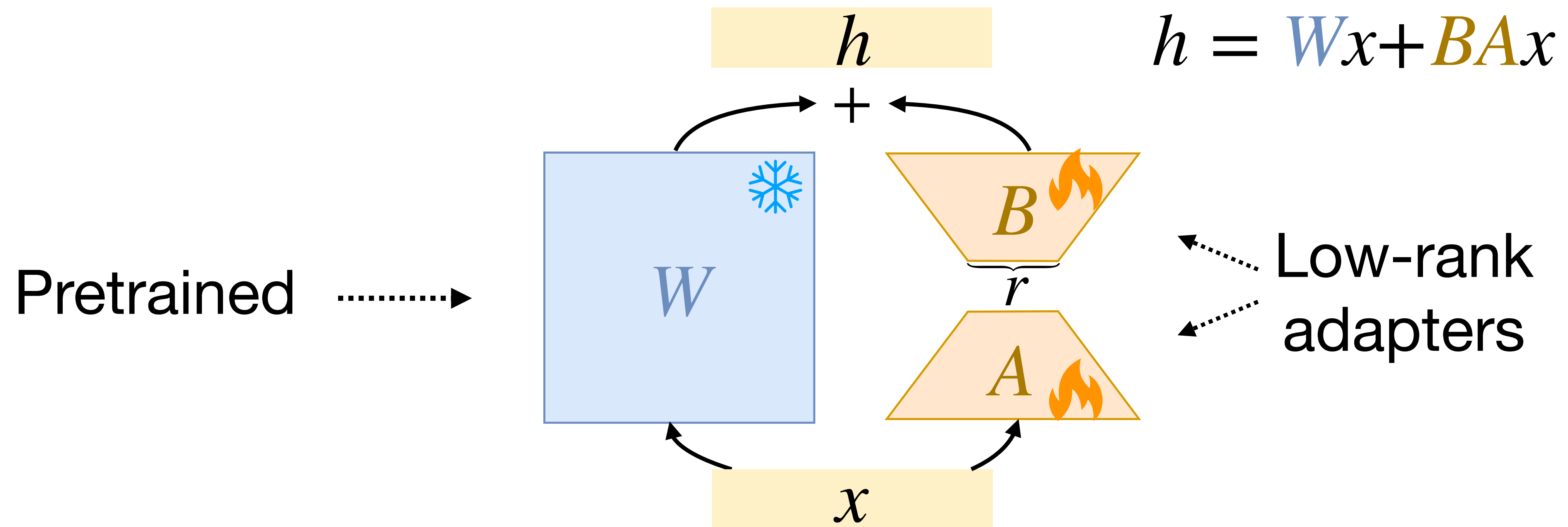
↓

$$\hat{G}_{t-1} \quad r \times M$$

LoRA - Low-Rank Adaption [3]

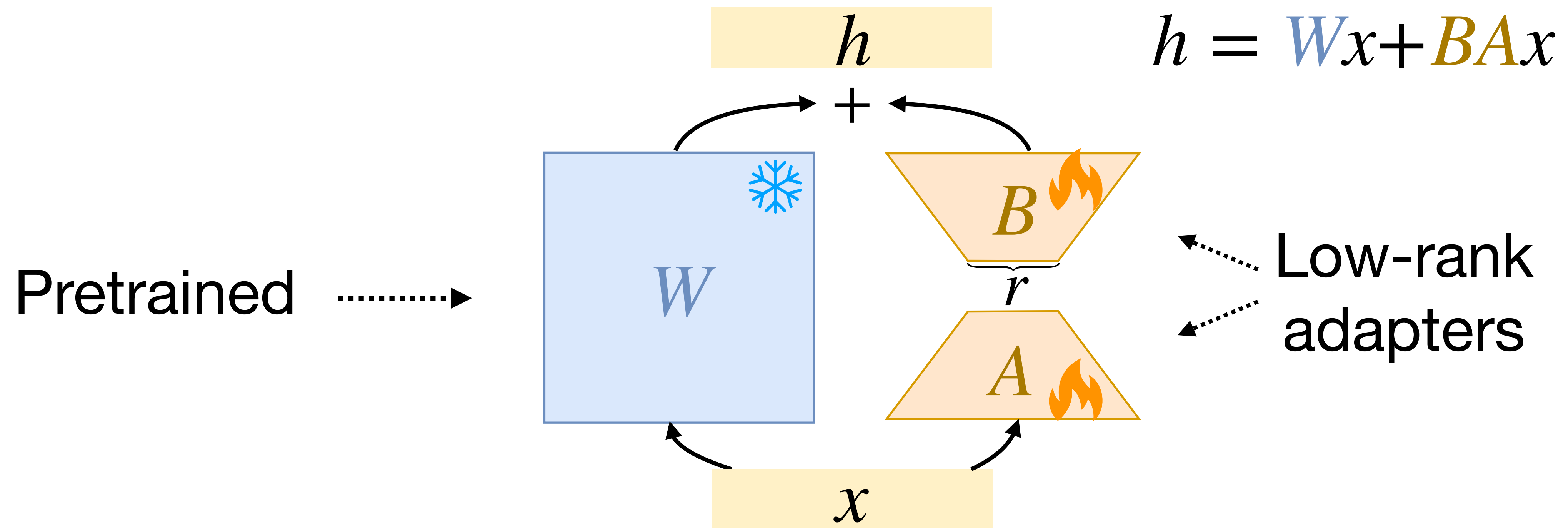


LoRA - Low-Rank Adaption [3]



Reduces memory for
gradients & optimizer-states

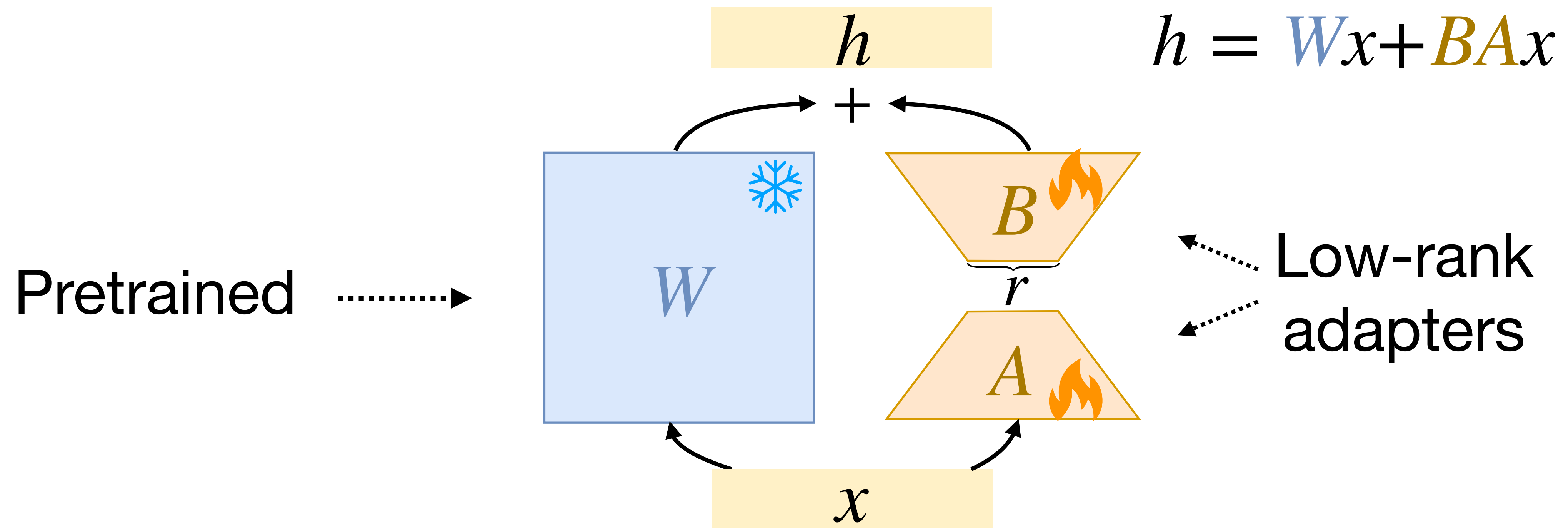
LoRA - Low-Rank Adaption [3]



Reduces memory for
gradients & optimizer-states

Quantizing W
works well [4]

LoRA - Low-Rank Adaption [3]



Reduces memory for
gradients & optimizer-states

Quantizing W
works well [4]

Does not work for
pretraining

LoQT: Low-Rank Quantized Pretraining

Key Components

- Pretrain from scratch with low-rank adapters (LoRA)

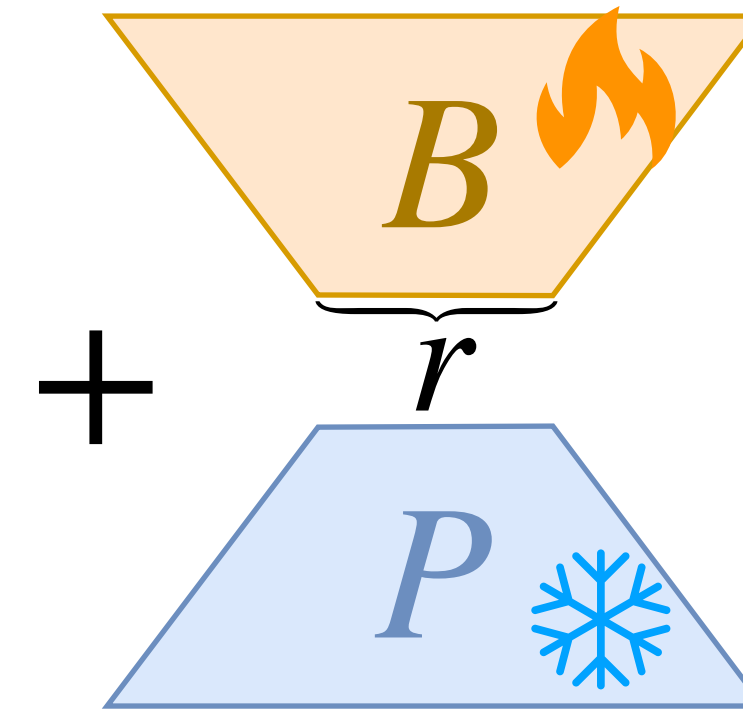
$$\hat{W} = W + PB$$

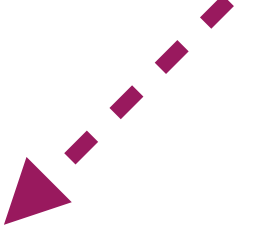
LoQT: Low-Rank Quantized Pretraining

Key Components

- Pretrain from scratch with low-rank adapters (LoRA)

Randomly
initialized W 



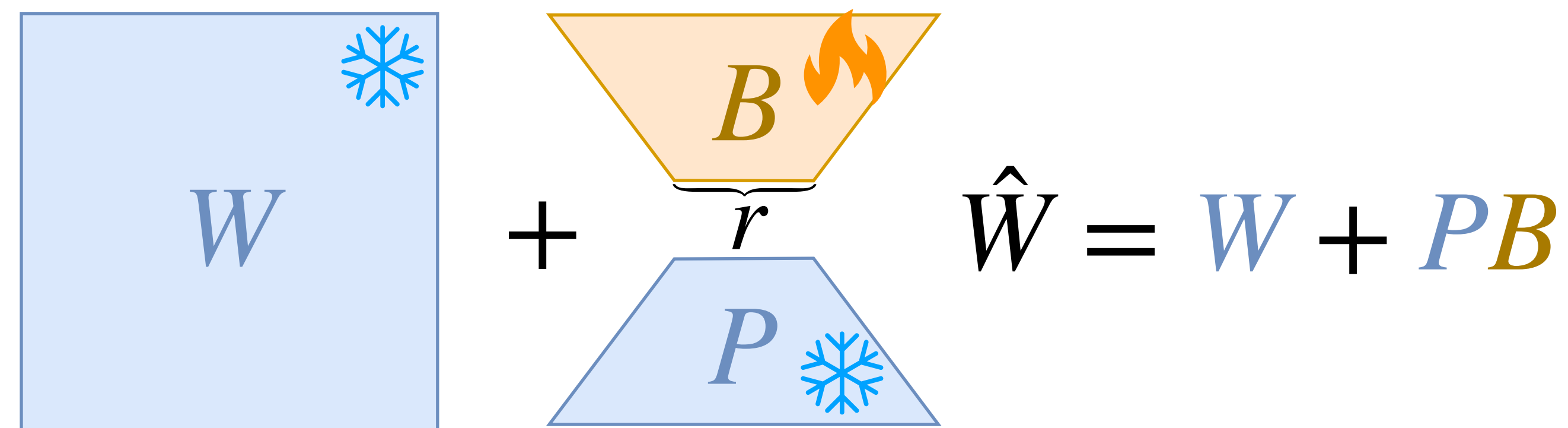
Only optimize a
single adapter 

$$\hat{W} = W + PB$$

LoQT: Low-Rank Quantized Pretraining

Key Components

- Pretrain from scratch with low-rank adapters (LoRA)
- Initialize P using ∇W



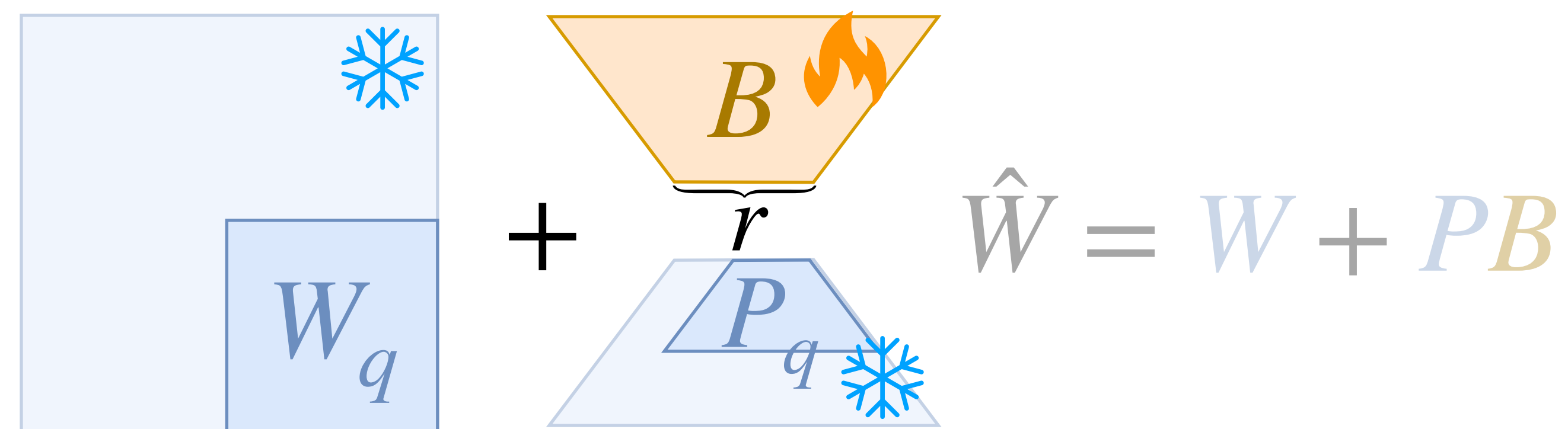
$$\text{SVD}(\nabla W) = U\Sigma V^{\top}$$

$$P = U[:, r]$$

LoQT: Low-Rank Quantized Pretraining

Key Components

- Pretrain from scratch with low-rank adapters (LoRA)
- Initialize P using ∇W
- Quantize P and W to 4-bit



$$\text{SVD}(\nabla W) = U \Sigma V^T$$

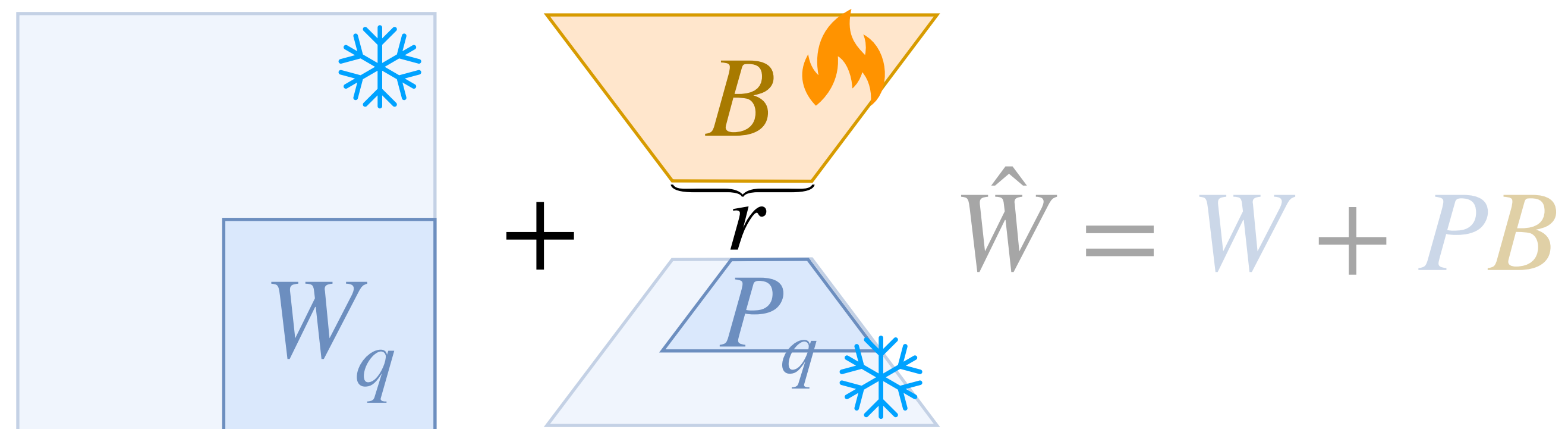
$$P = U[:, r]$$

LoQT: Low-Rank Quantized Pretraining

Key Components

- Pretrain from scratch with low-rank adapters (LoRA)
- Initialize P using ∇W
- Quantize P and W to 4-bit
- Initialize B using a *quantization error compensation* term.

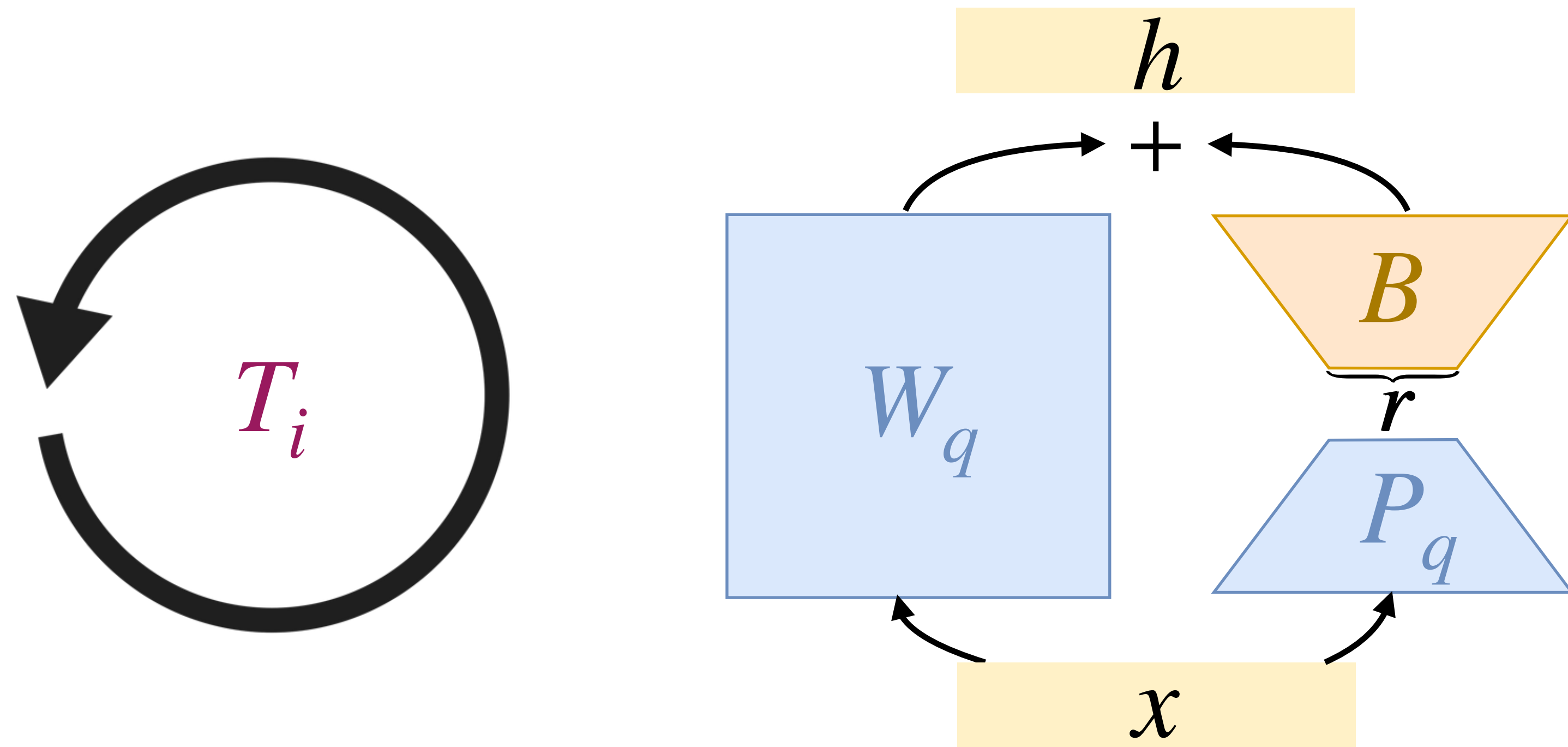
$$B = P^{-1}(W_q - W)$$



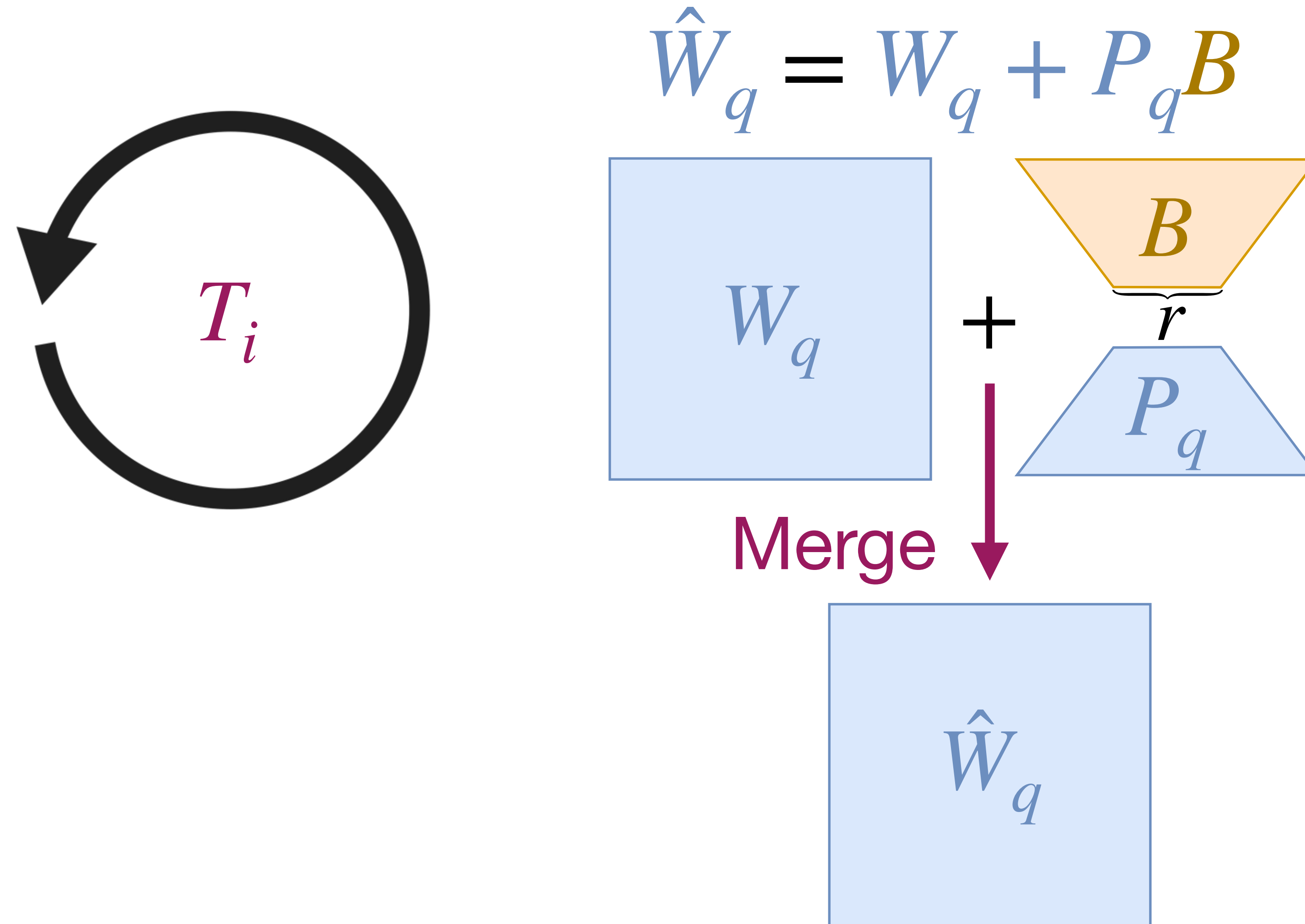
$$\text{SVD}(\nabla W) = U \Sigma V^T$$

$$P = U[:, r]$$

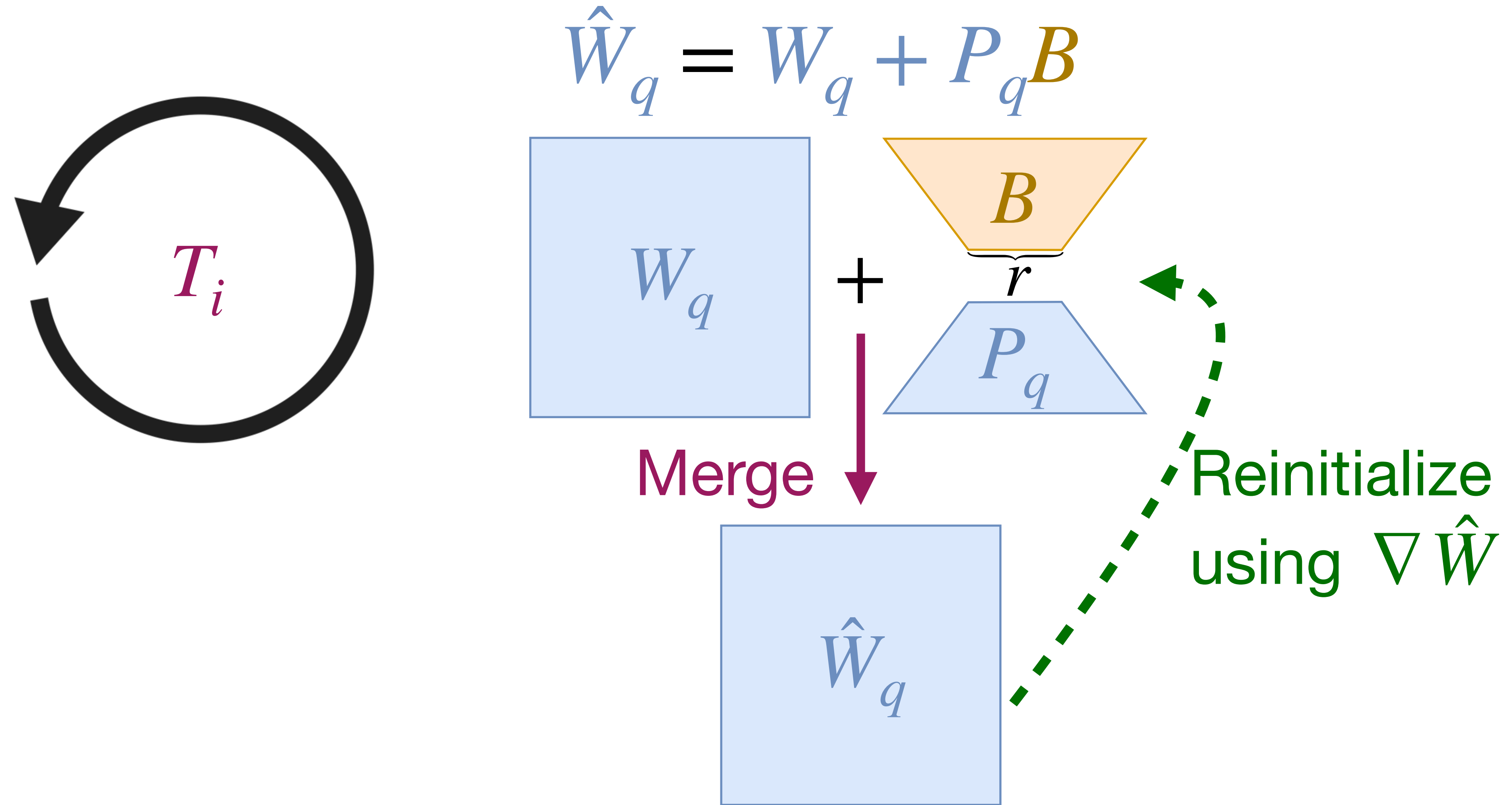
LoQT - Training



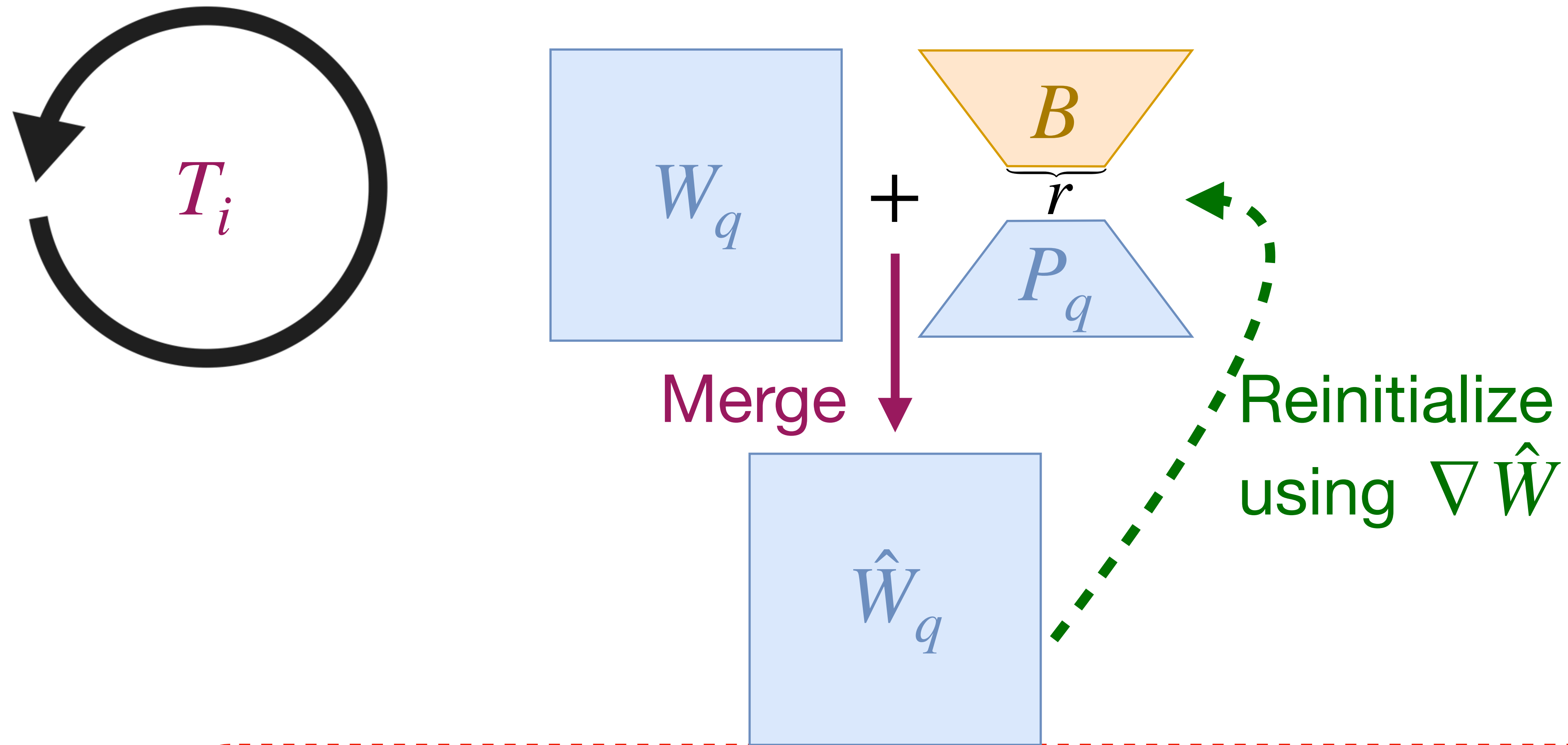
LoQT - Merge and Reinitialize



LoQT - Merge and Reinitialize



LoQT - Merge and Reinitialize



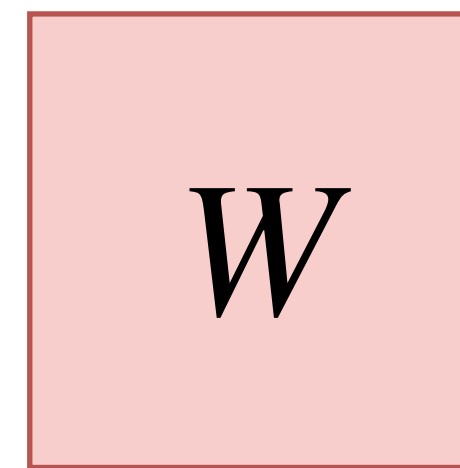
$$W_N = W_0 + P_0 B_0 + P_1 B_1 + \dots + P_N B_N$$

Memory Requirements with Adam

Regular Training

$$h = Wx$$

Weights



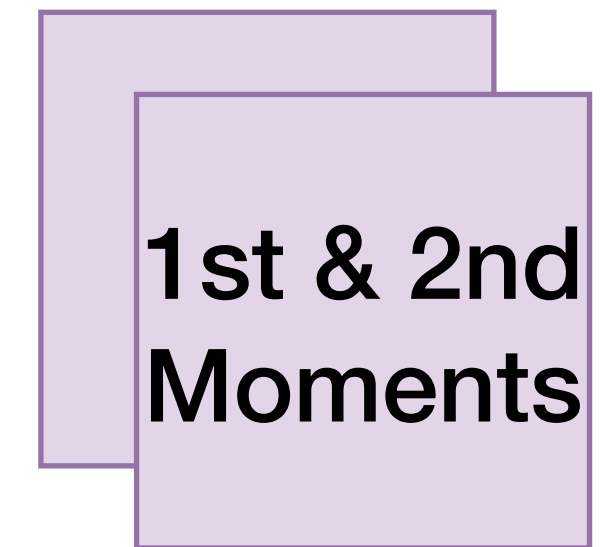
mn

Gradients



mn

Optimizer States



1st & 2nd Moments

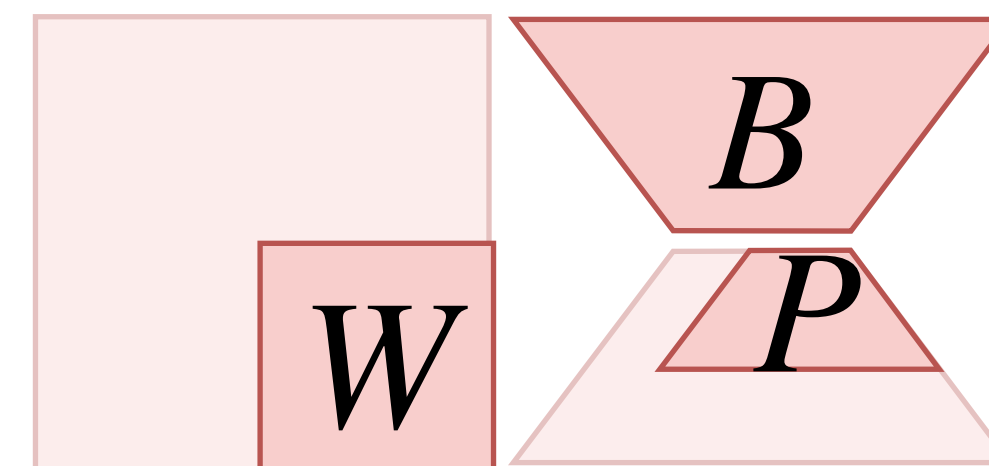
$2mn$

LoQT

$$h = Wx + PBx$$

Good

if $r \ll \min(m, n)$

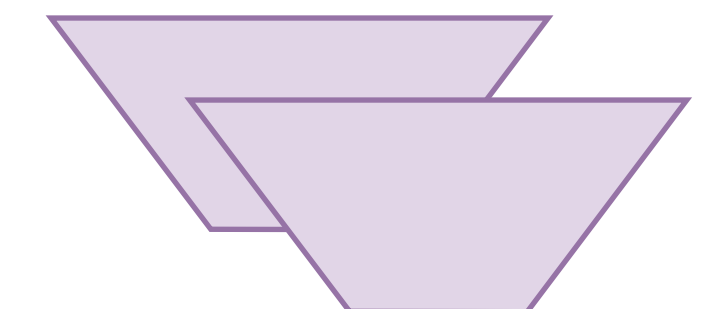


$m_q n_q + nr + m_q r$



nr

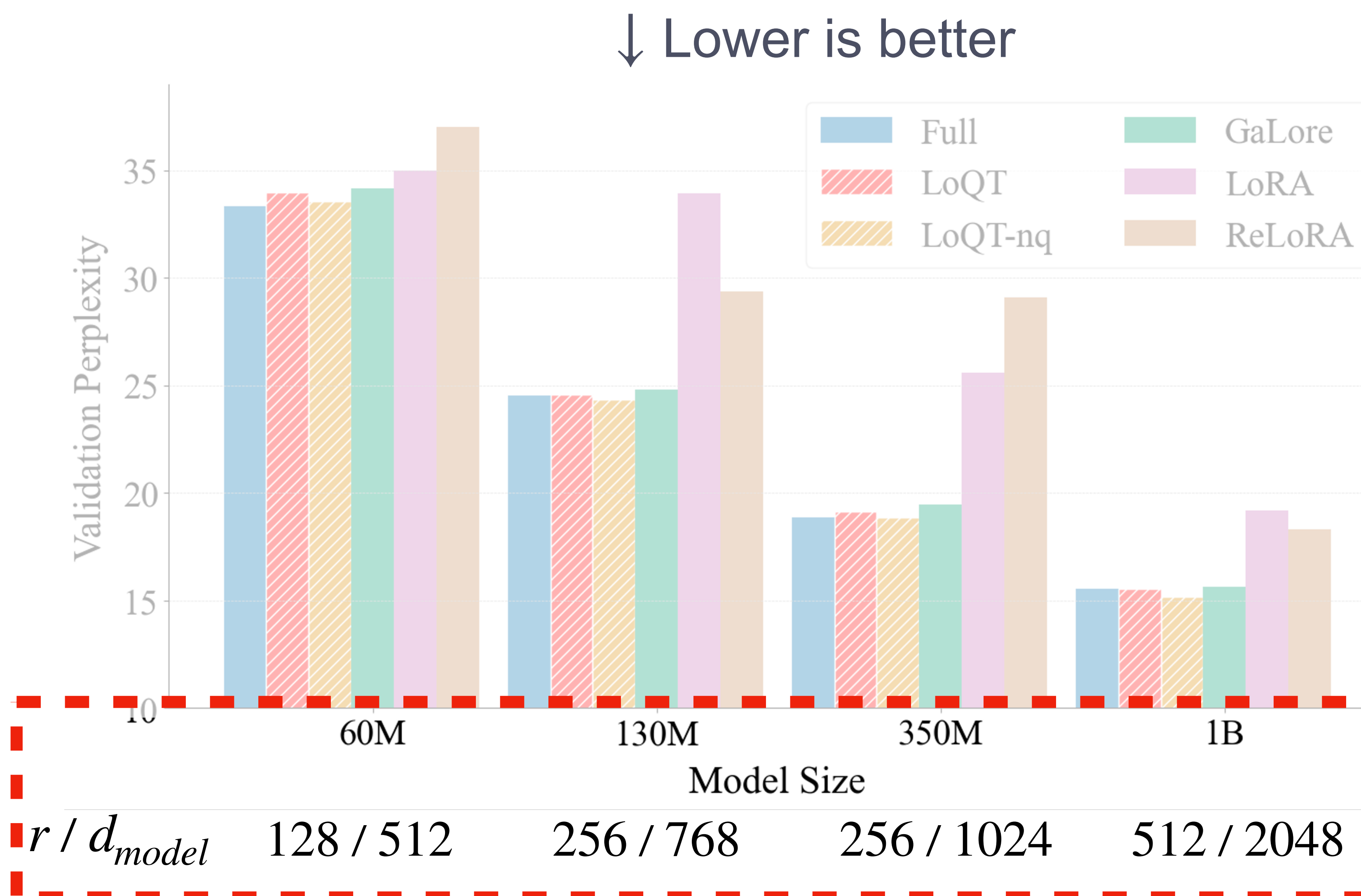
$\frac{m}{r}$



$2nr$

Key Results

Pretraining LLaMA2 Models on the C4 Dataset



Pretraining LLaMA2 Models on the C4 Dataset

Takeaways

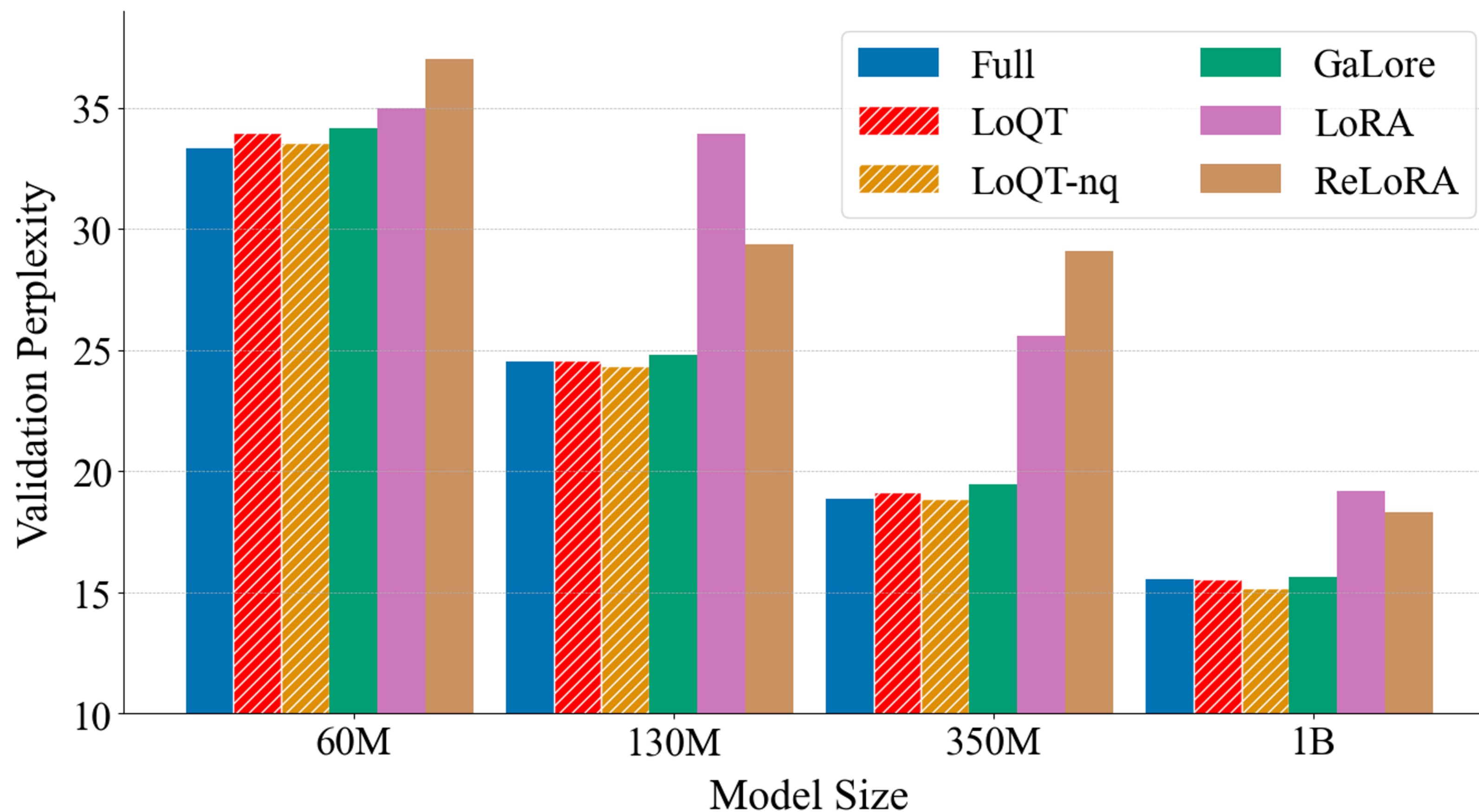
- **LoQT** is comparable to **Full** training and **GaLore**

LoQT - 1B model

61% less memory than **Full**

41% less memory than **GaLore**

↓ Lower is better



r / d_{model}

128 / 512

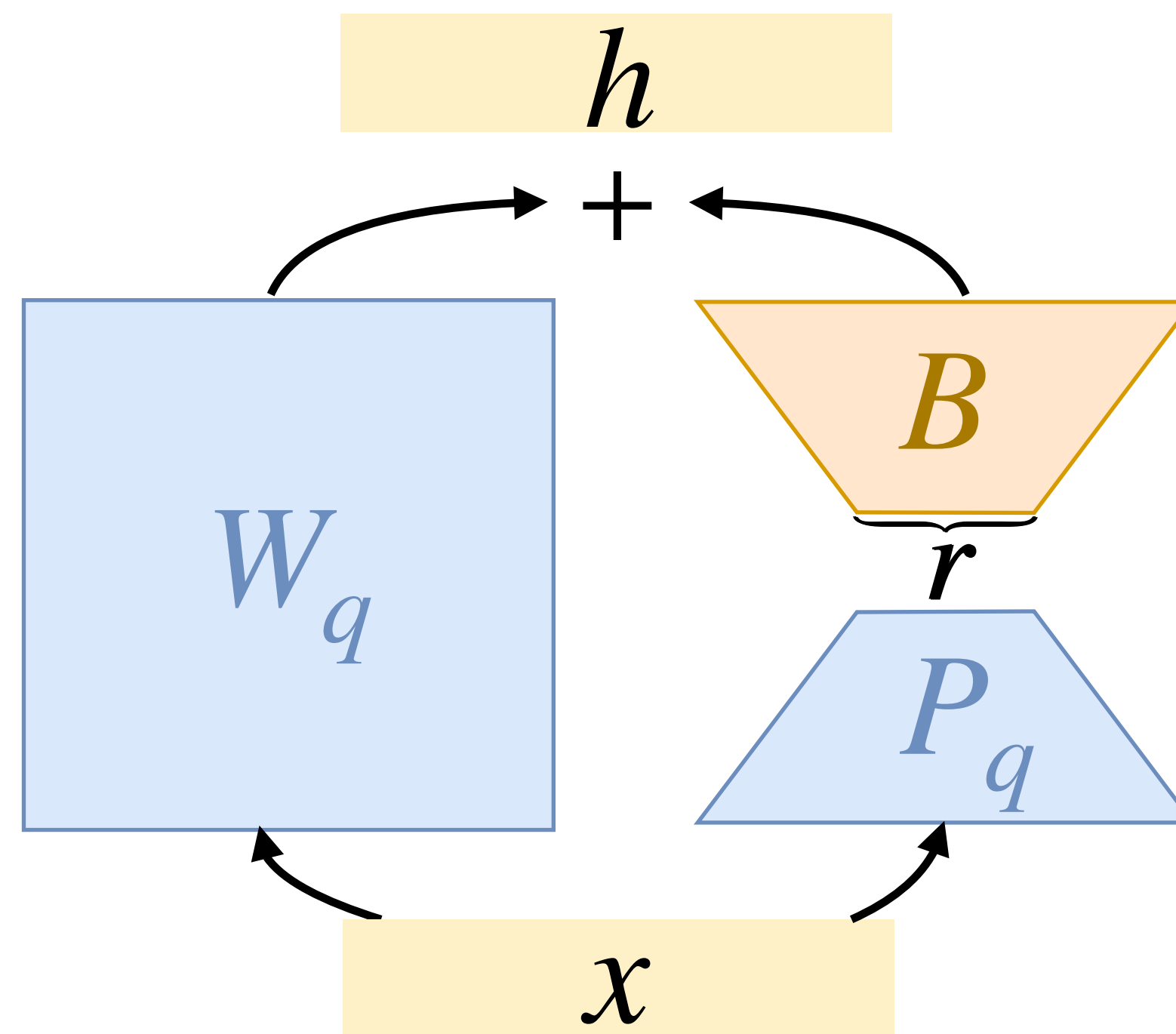
256 / 768

256 / 1024

512 / 2048

Limitations and Future Work

- **Speed:** LoQT is 16% slower than BF16 training.



Limitations and Future Work

- **Speed:** LoQT is 16% slower than BF16 training.
- **Alternative Architectures and Quantization.**

Key Takeaways

- **Pretraining** from scratch with **low-rank updates** and **4-bit quantization** can match BF16 training.

Key Takeaways

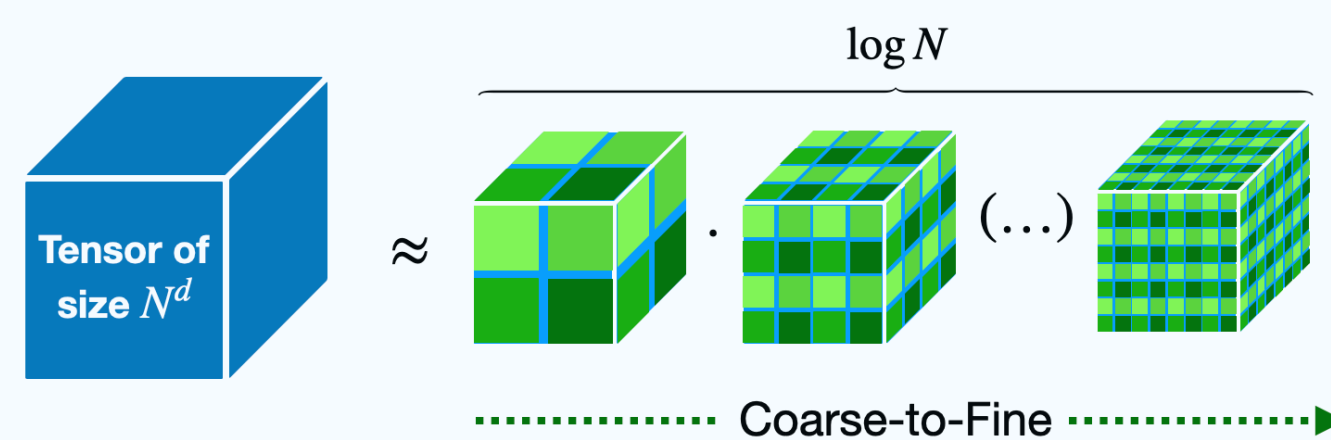
- **Pretraining** from scratch with **low-rank updates** and **4-bit quantization** can match BF16 training.
- **Low Memory:** LoQT enables training on consumer hardware.

Structured Learning Under Memory Constraints

Sebastian Loeschcke

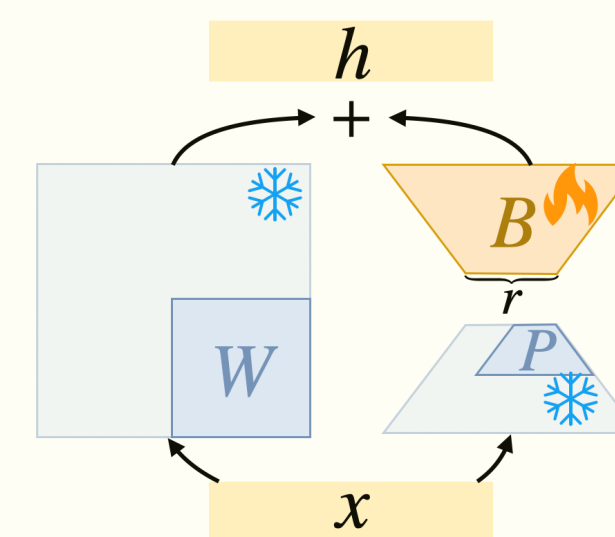
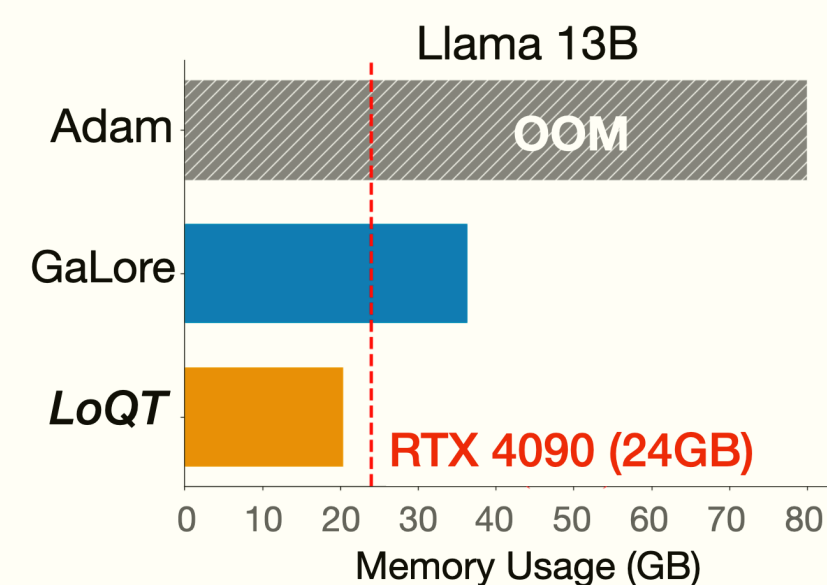
ICML 2024

Coarse-To-Fine Tensor Trains for Compact Visual Representations



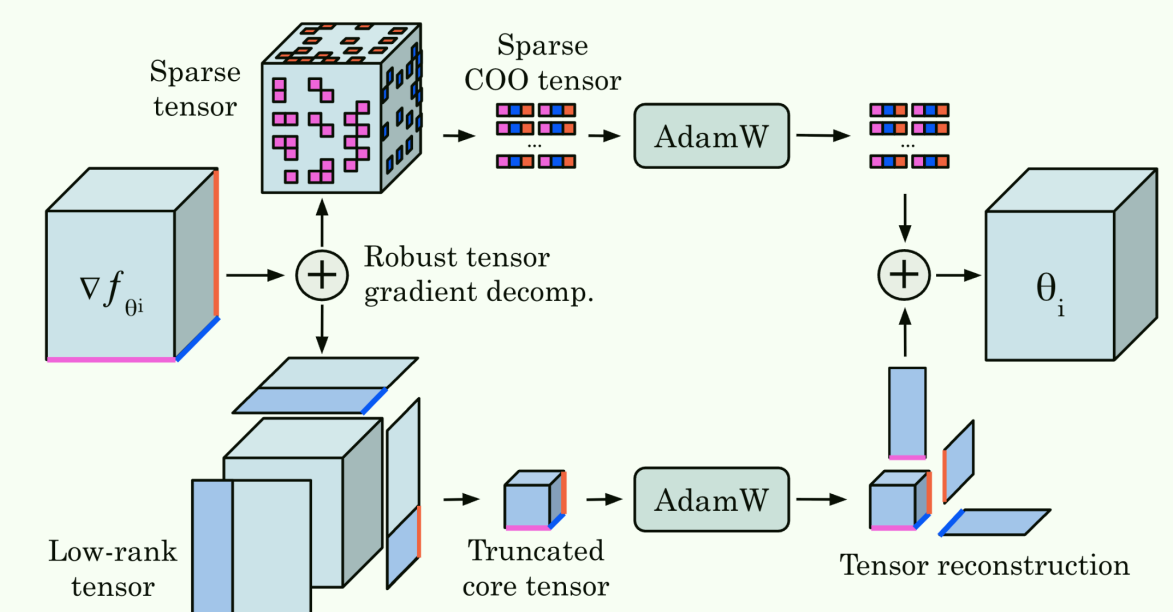
NeurIPS 2024

LoQT: Low Rank Adapters for Quantized Pre-Training



Under review

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



Conclusion: Structured Learning Under Memory Constraints

- Modern model training is increasingly limited by memory, not compute.
- Weights, gradients, and optimizer states have low-rank structures during training
- Exploiting this structure enables scalable, memory-efficient training without sacrificing performance.

Ongoing work

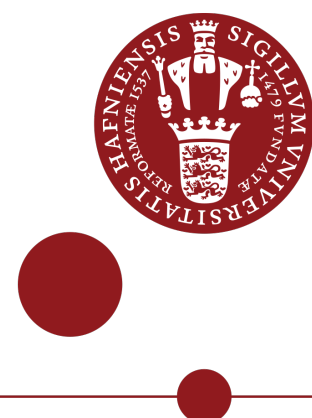
- Lattice based quantization aware training (Qualcomm project)
- Ongoing Caltech projects I'm advising:
 - Low-precision training using exponent and mantissa separation
 - Adaptive Sign-Based Optimizer for Communication-Efficient Distributed Training
 - Self-Scaled Adaptive Preconditioning with Applications to Matrix Optimization and Physics-Informed Machine Learning

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

Research Visit at Caltech

UNIVERSITY OF
COPENHAGEN



Caltech



Meta

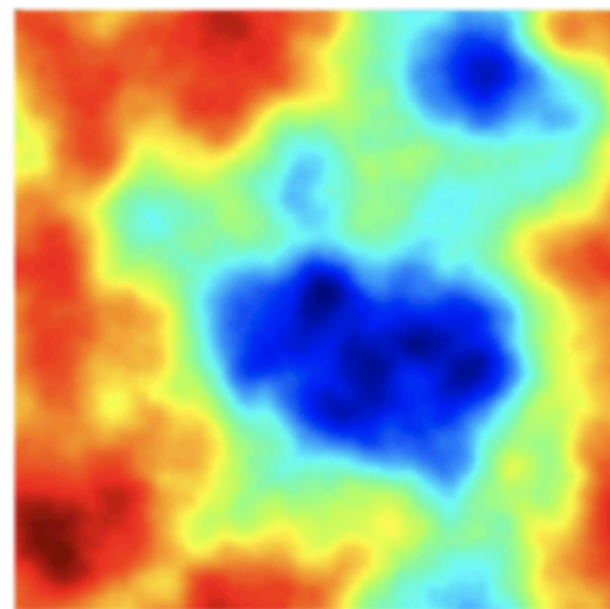


nVIDIA®

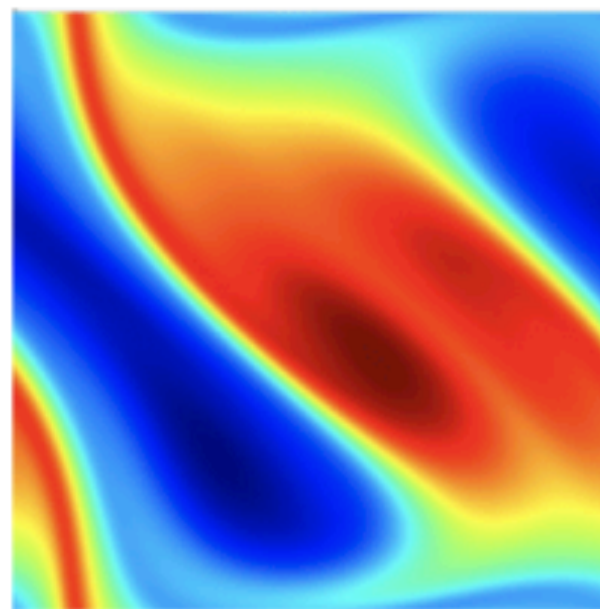
Learning Operators from Initial Conditions

- Given an initial condition of a physical system, we aim to learn an operator that maps it to future states.

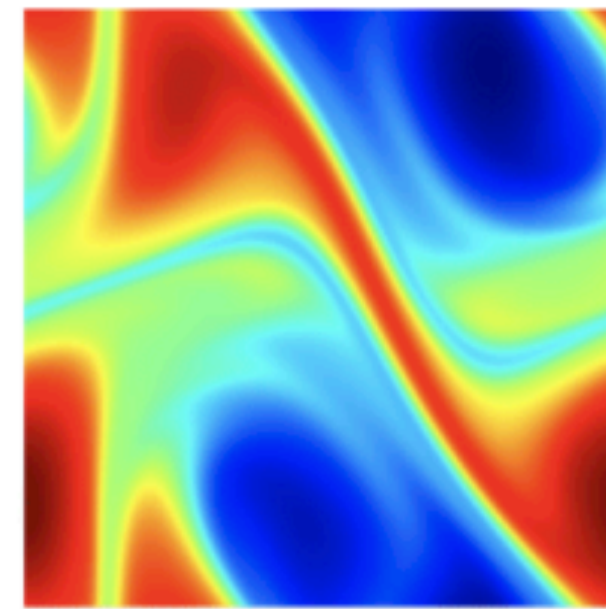
Initial Vorticity



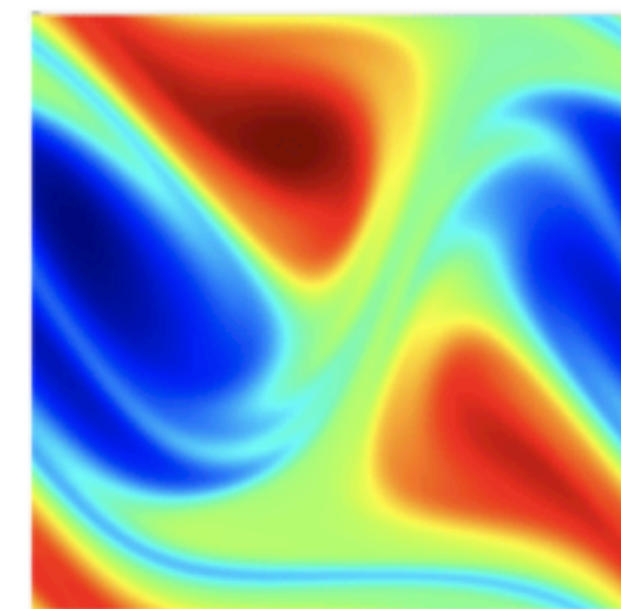
$t=15$



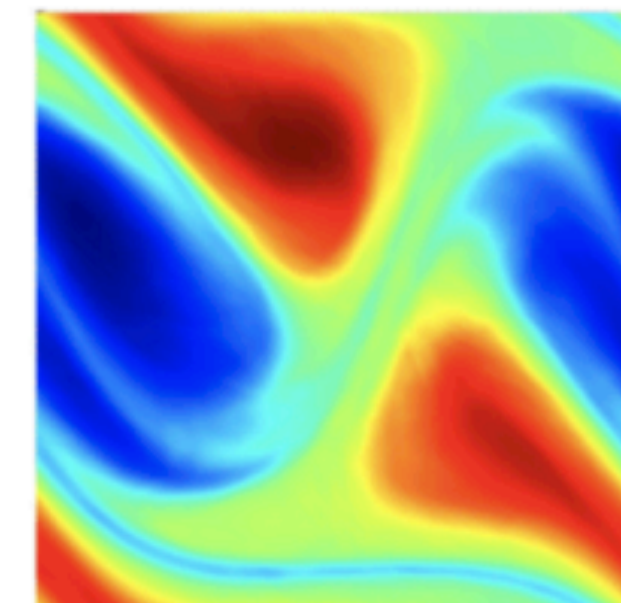
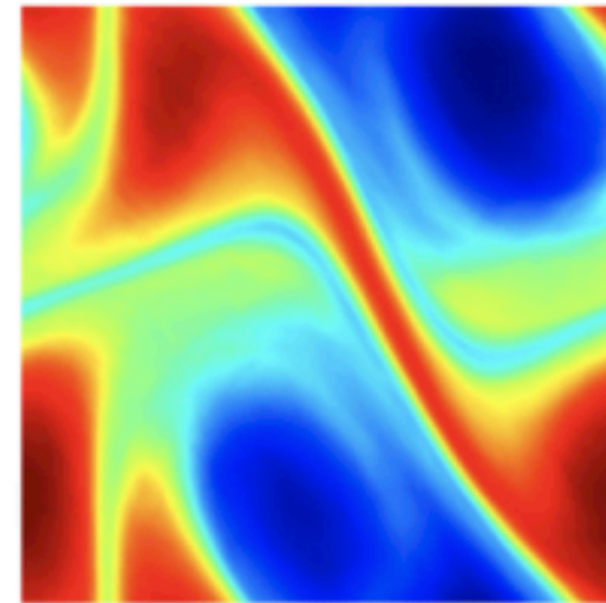
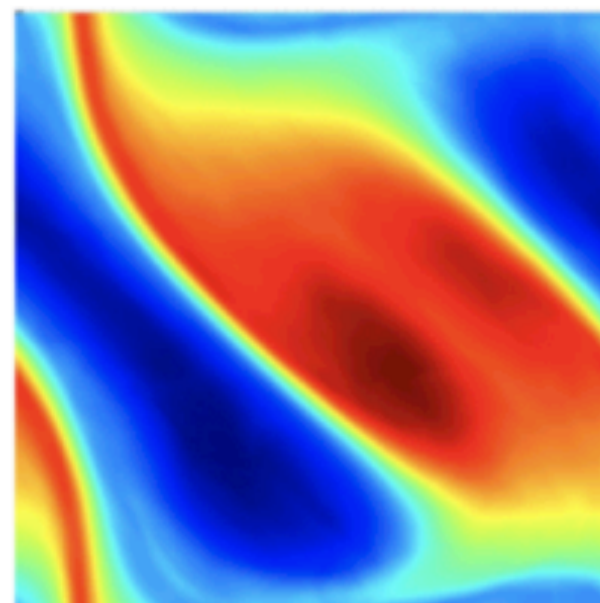
$t=20$



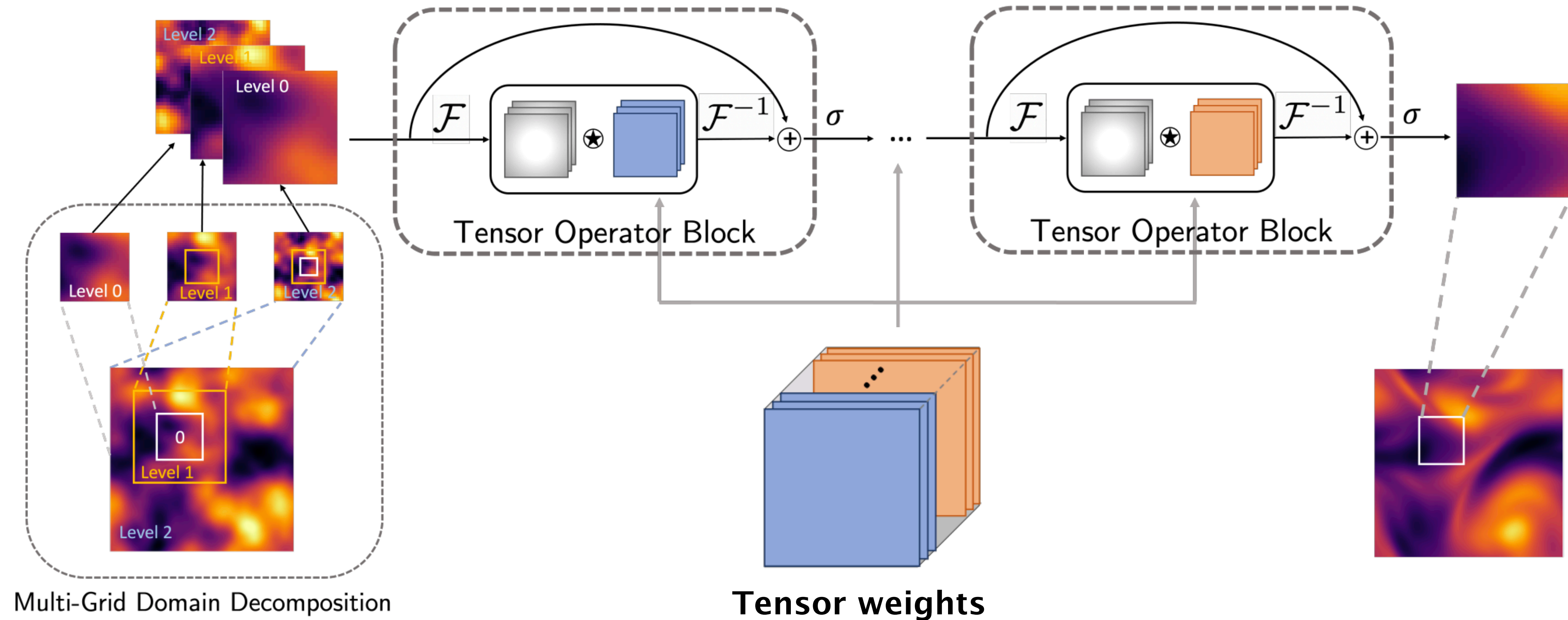
$t=25$



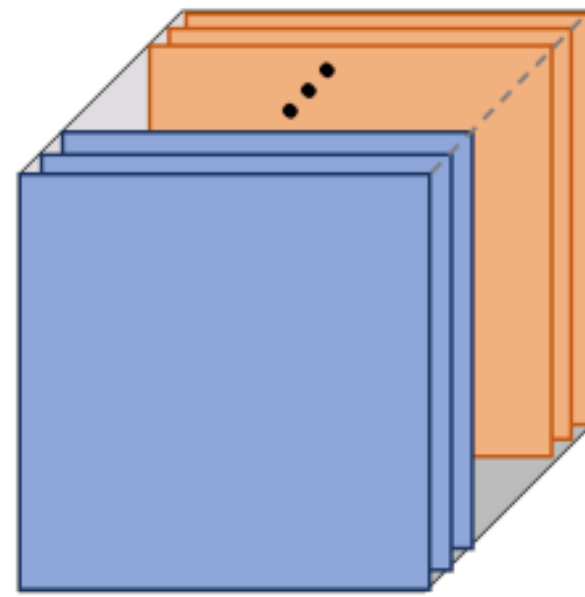
Prediction



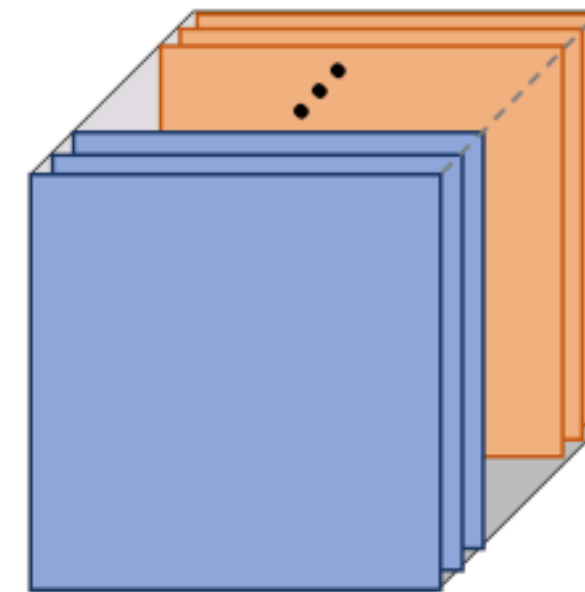
State of the art use Tensorized weights to learn operators on multidimensional inputs



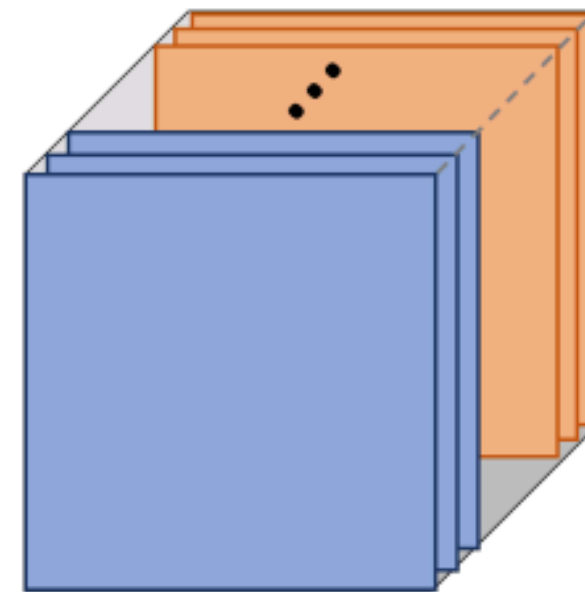
Tensor weights make Neural Operator Learning memory intensive



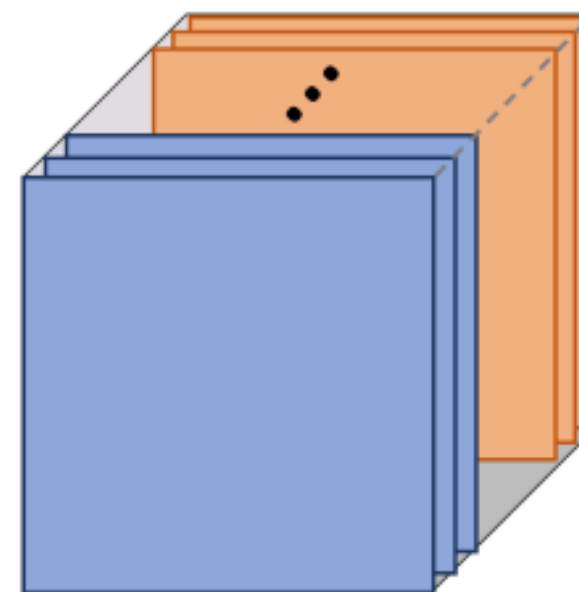
Tensor weights



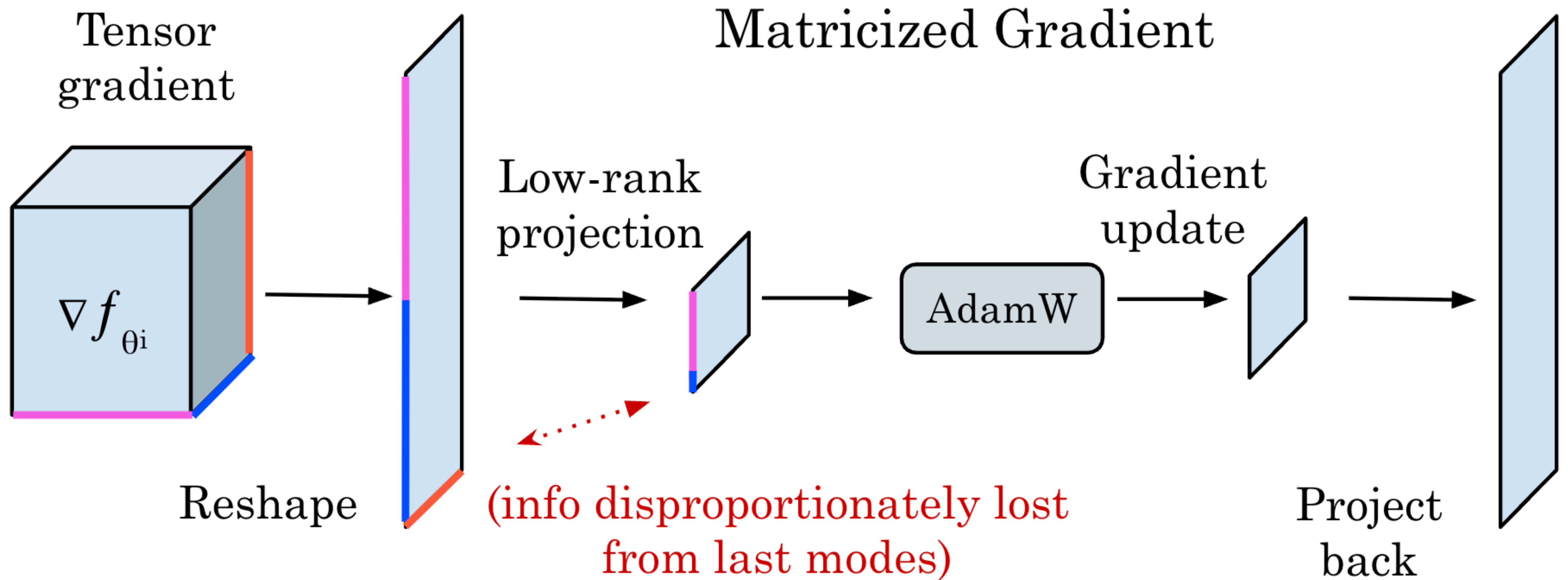
Tensor Gradients



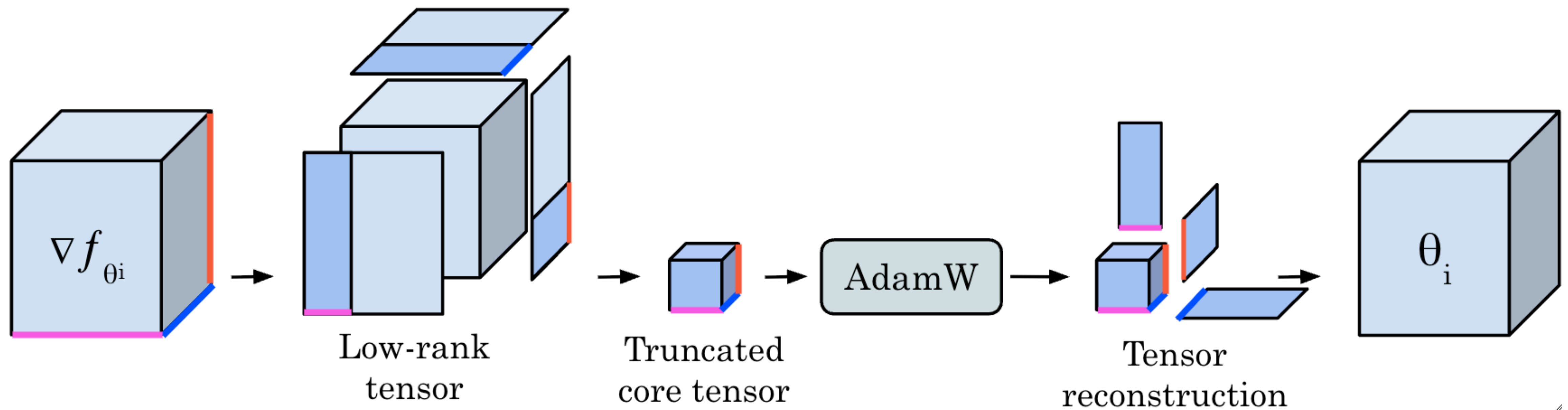
Tensor Adam Statistics



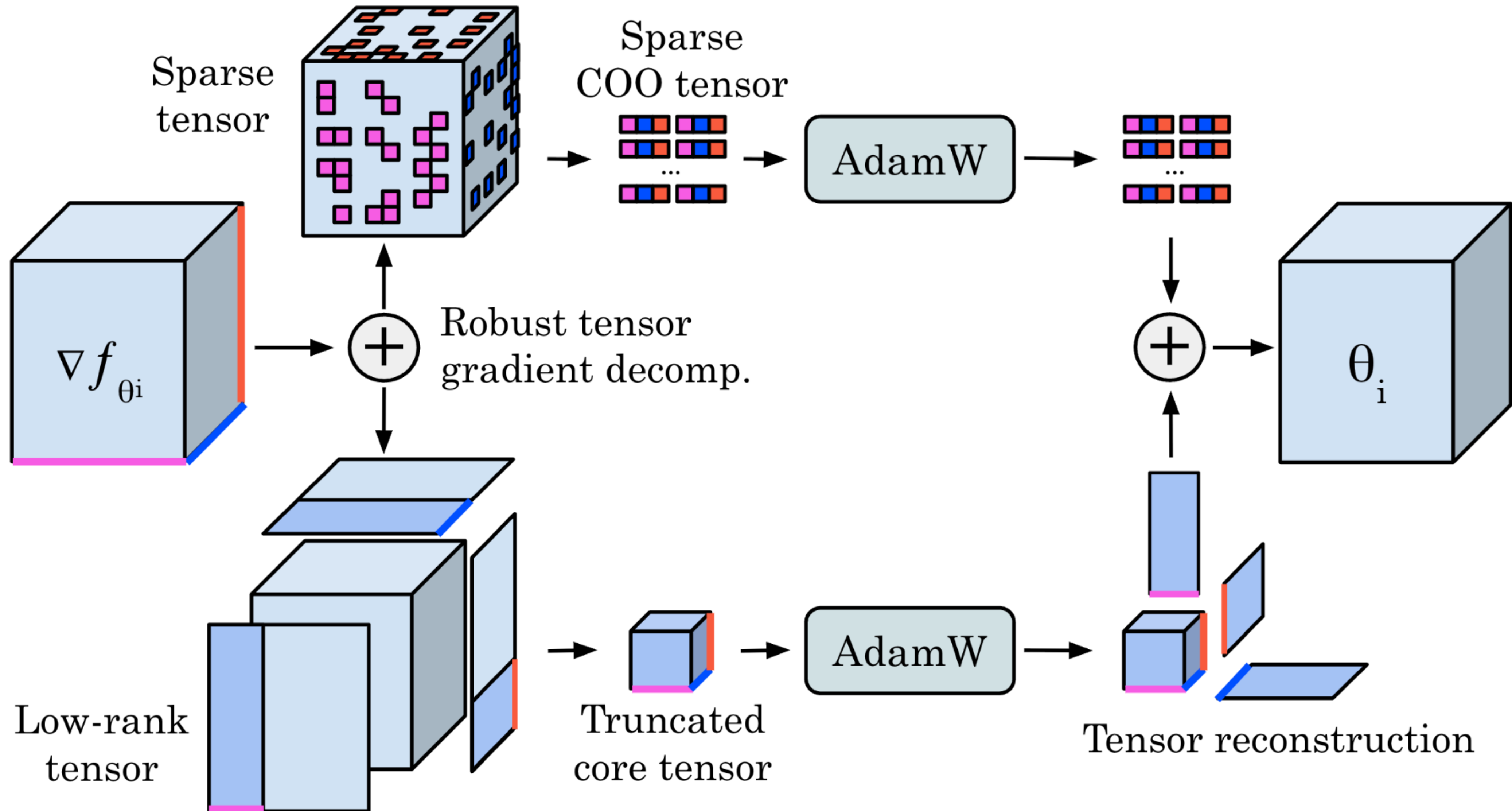
Can we use same gradients decomposition approaches for tensors?



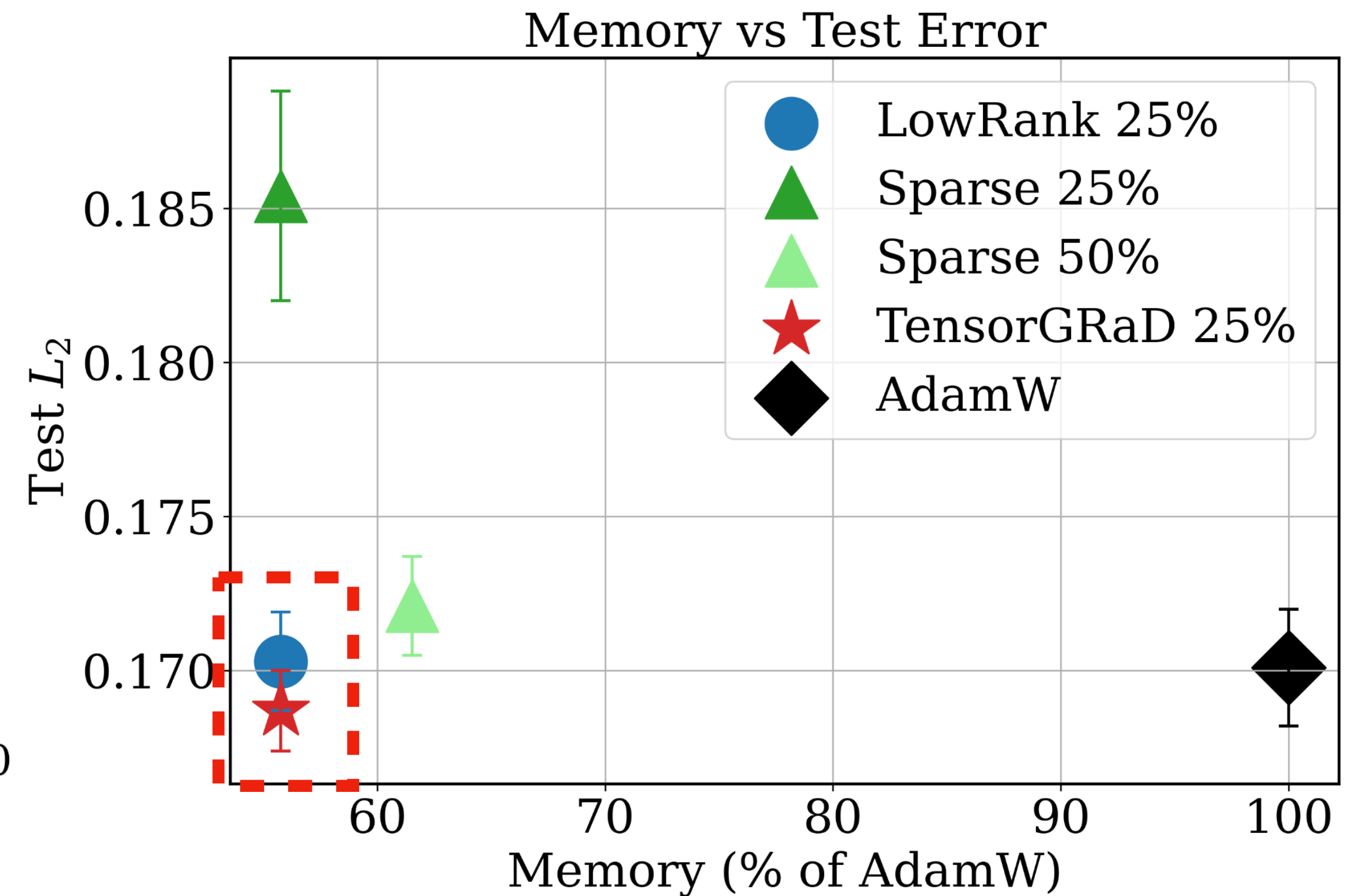
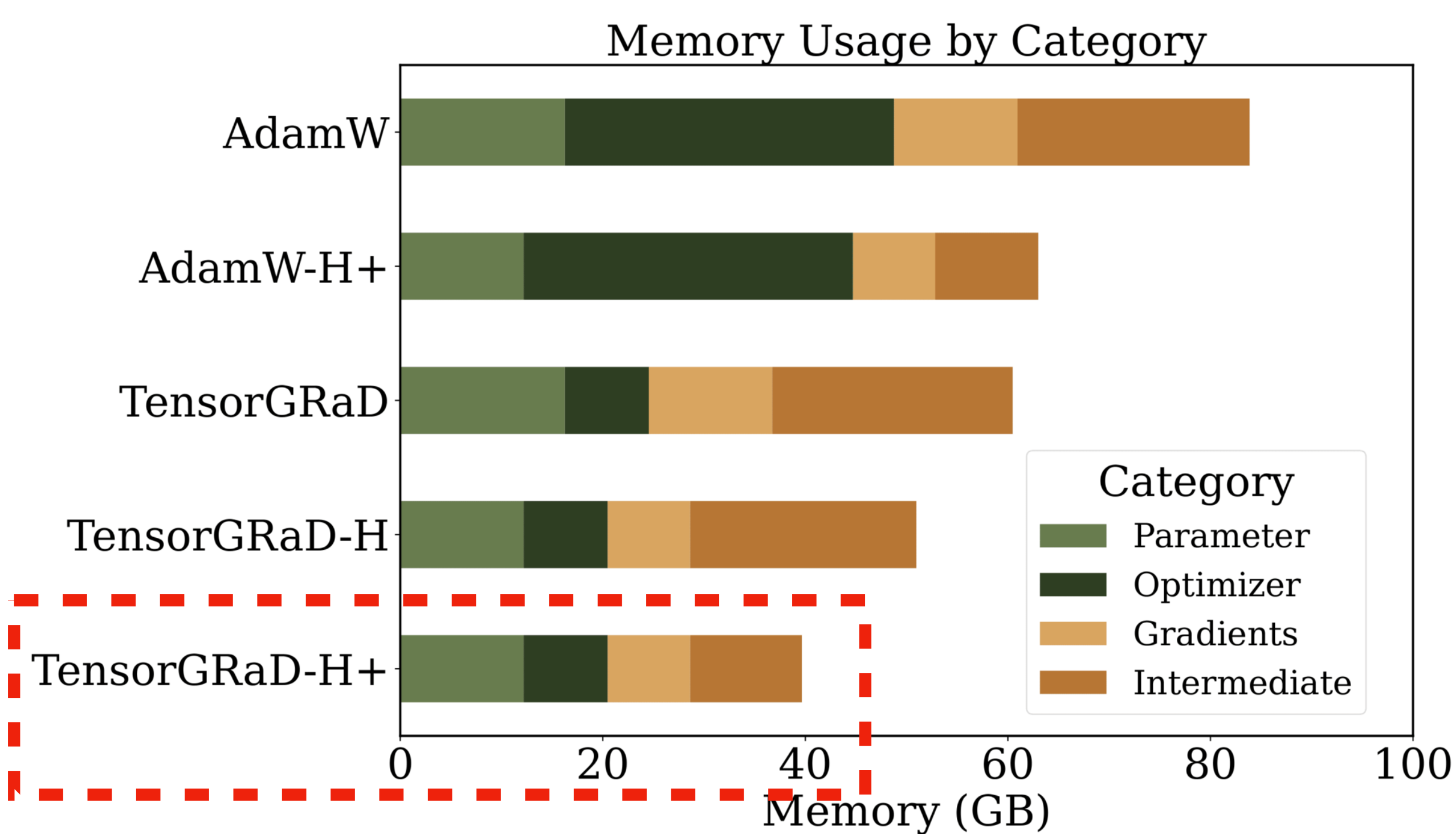
Projects tensor gradients using low-rank tensor factorization



Low-rank + Sparse



Matches performance of Adam using half the memory



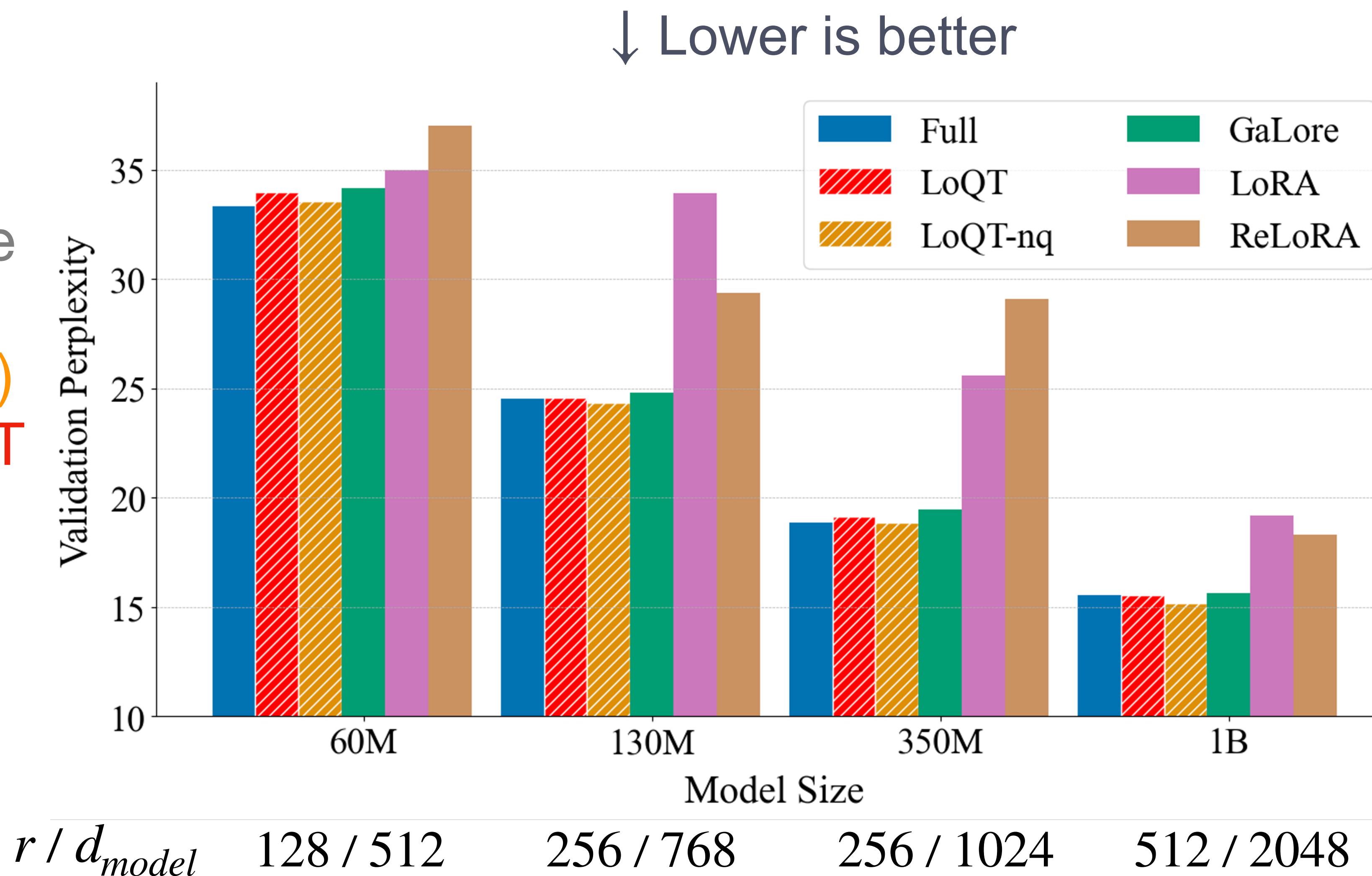
Key Takeaways

- Neural operator training is memory-limited due to large tensor-valued gradients.
- TensorGRaD exploits structure in these gradients via low-rank + sparse tensor decomposition.
- This enables memory-efficient training without sacrificing accuracy in scientific ML tasks.

Pretraining LLaMA2 Models on the C4 Dataset

Takeaways

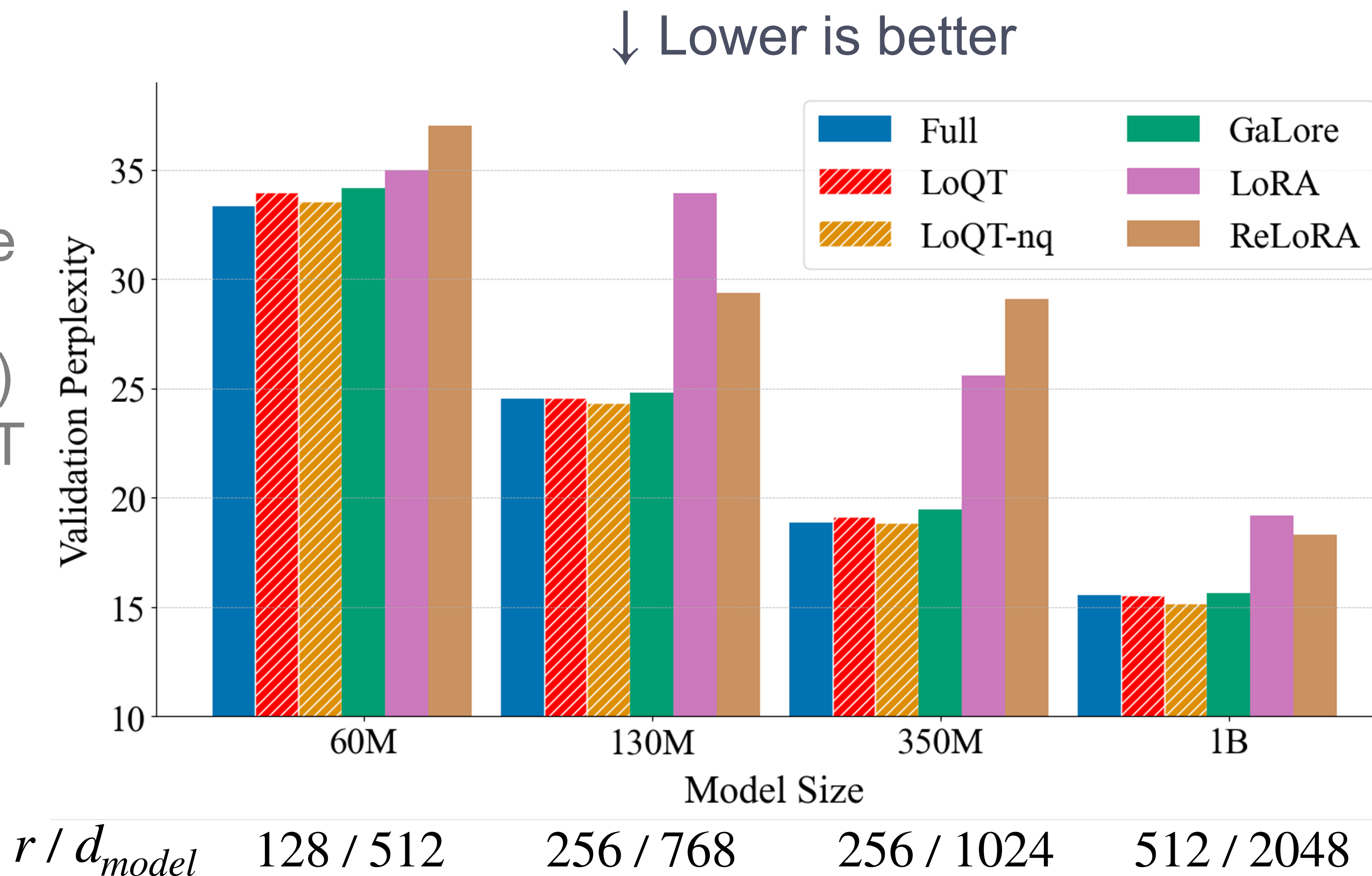
- LoQT is very comparable to Full training and GaLore
- No-quantization (LoQT-nq) is slightly better than LoQT



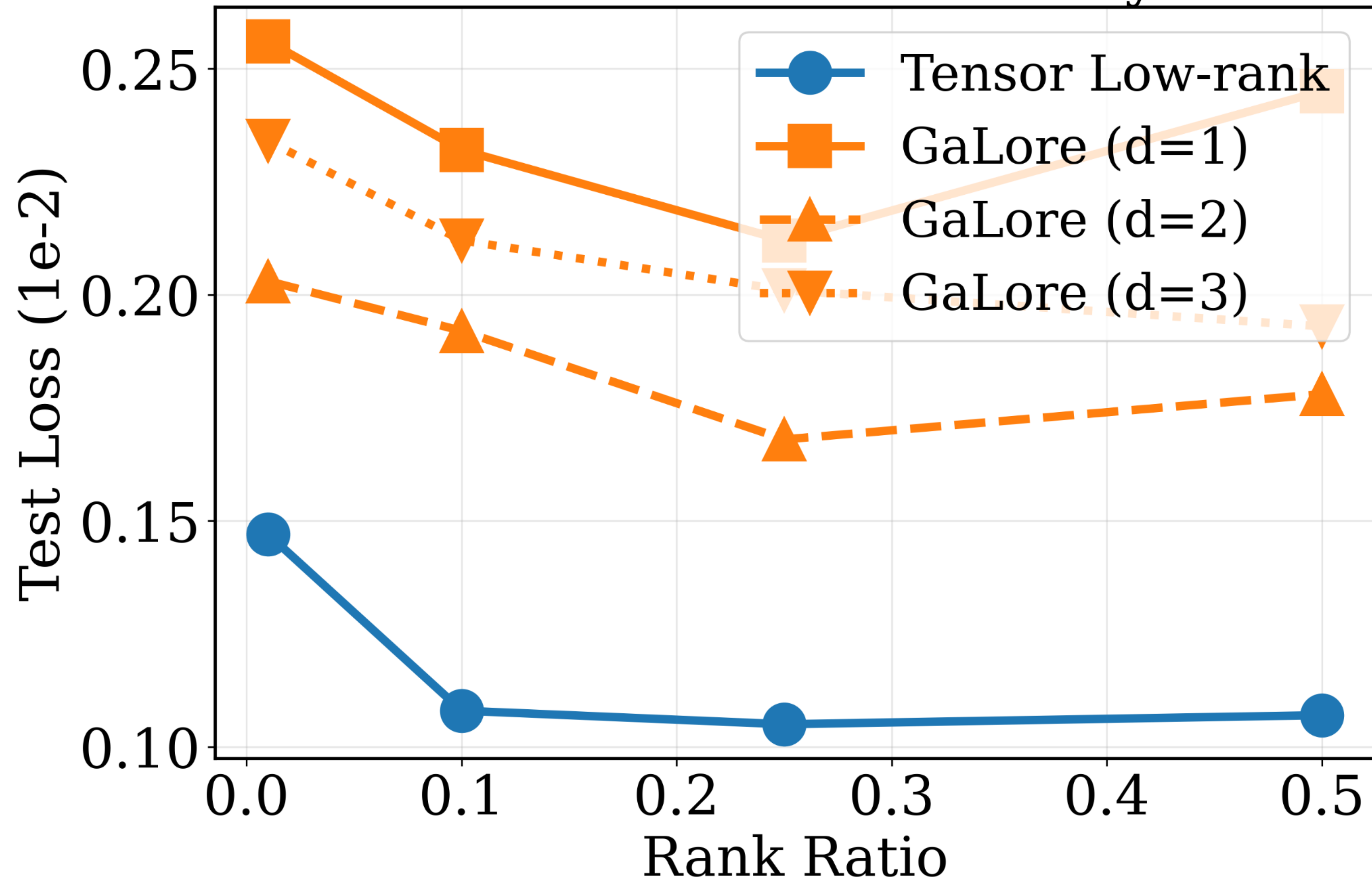
Pretraining LLaMA2 Models on the C4 Dataset

Takeaways

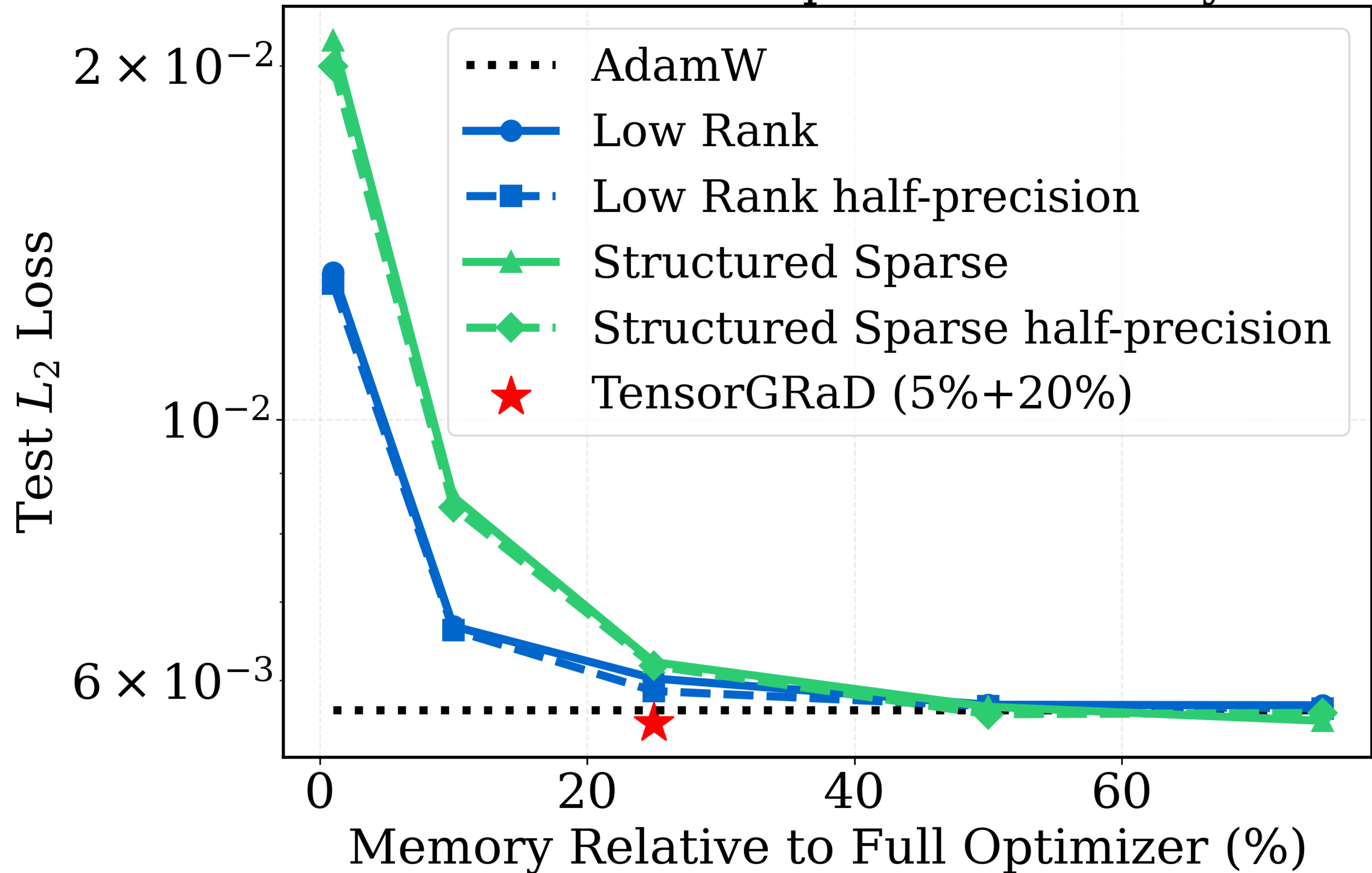
- LoQT is very comparable to Full training and GaLore
- No-quantization (LoQT-nq) is slightly better than LoQT
- **LoQT** is much better than **LoRA** and **ReLoRA**



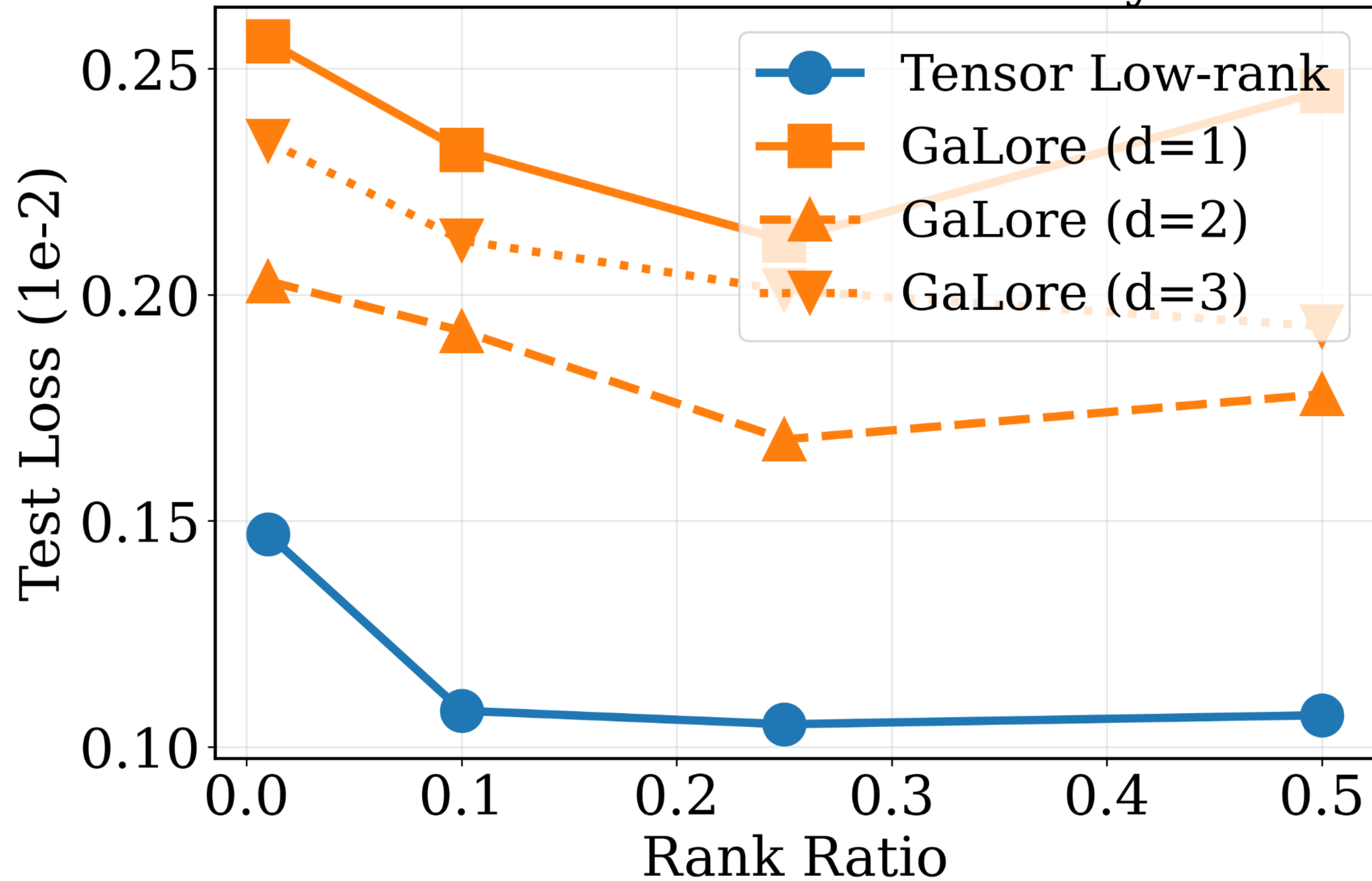
Model Performance on Darcy-flow

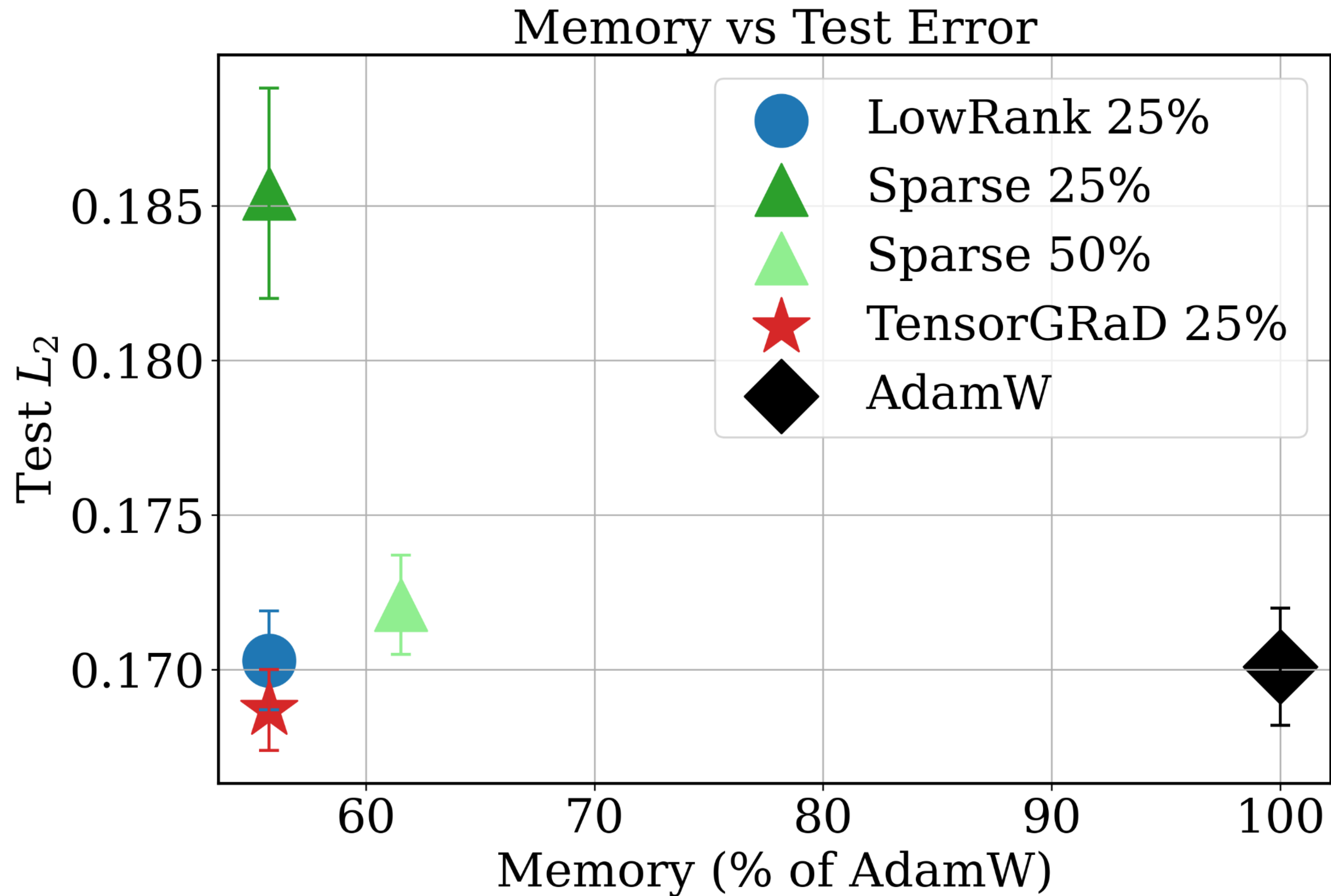


Test Error vs Optimizer Memory

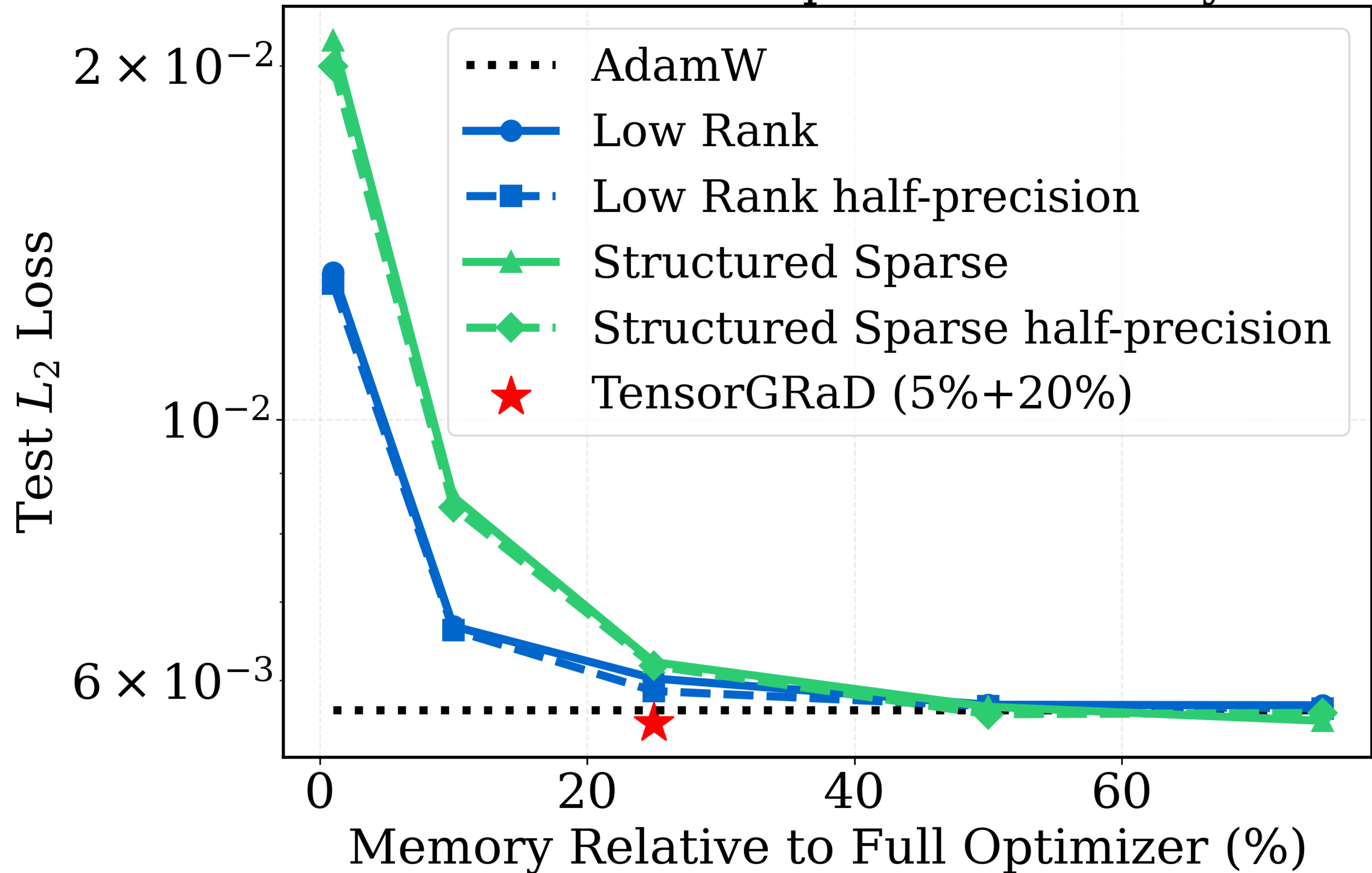


Model Performance on Darcy-flow





Test Error vs Optimizer Memory

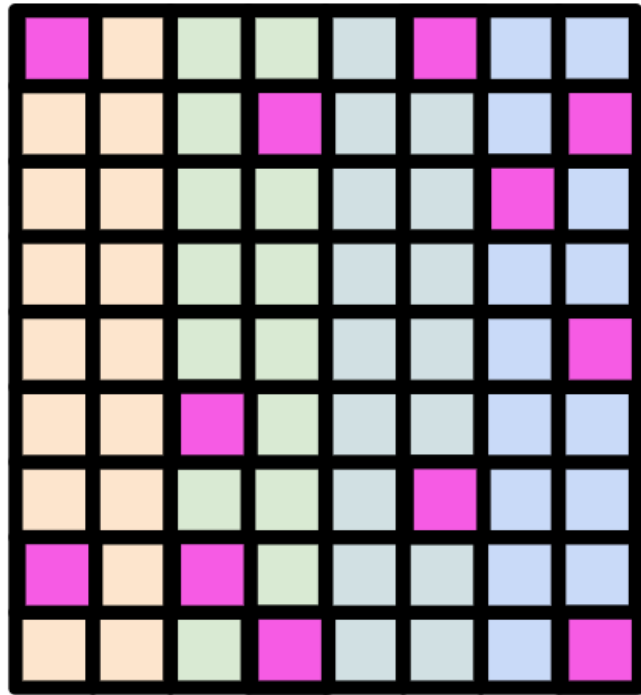
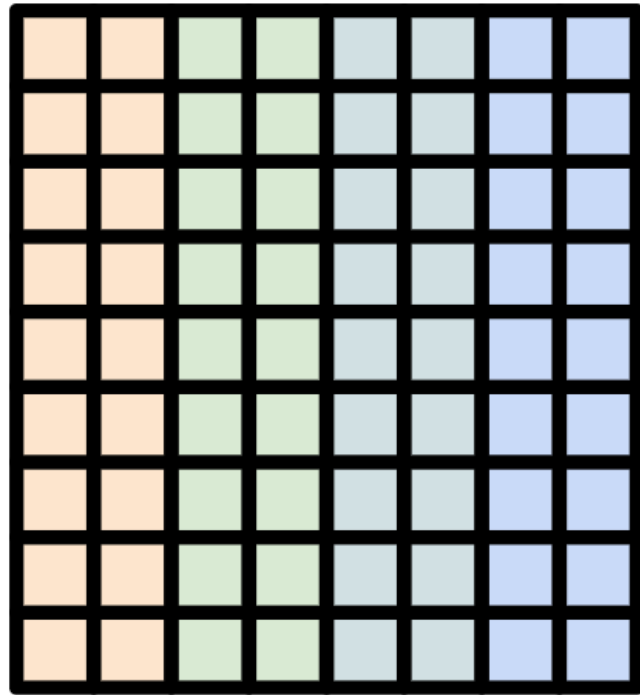
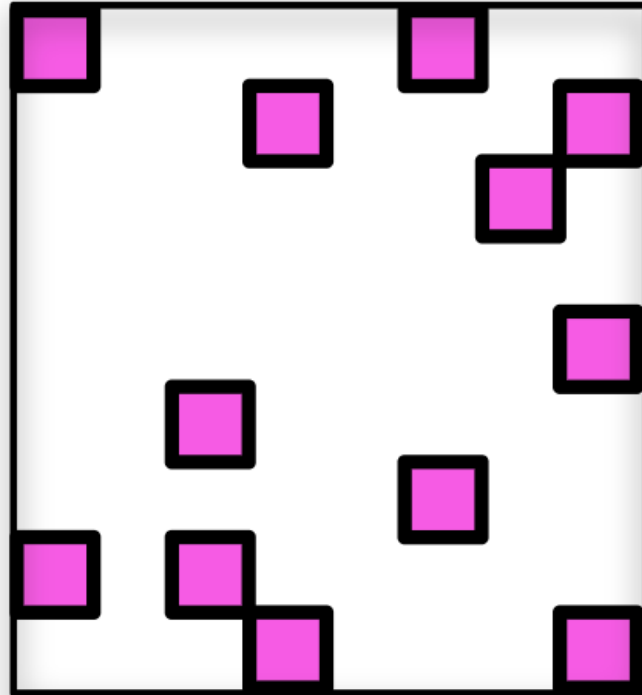


Structure of Gradient Matrices

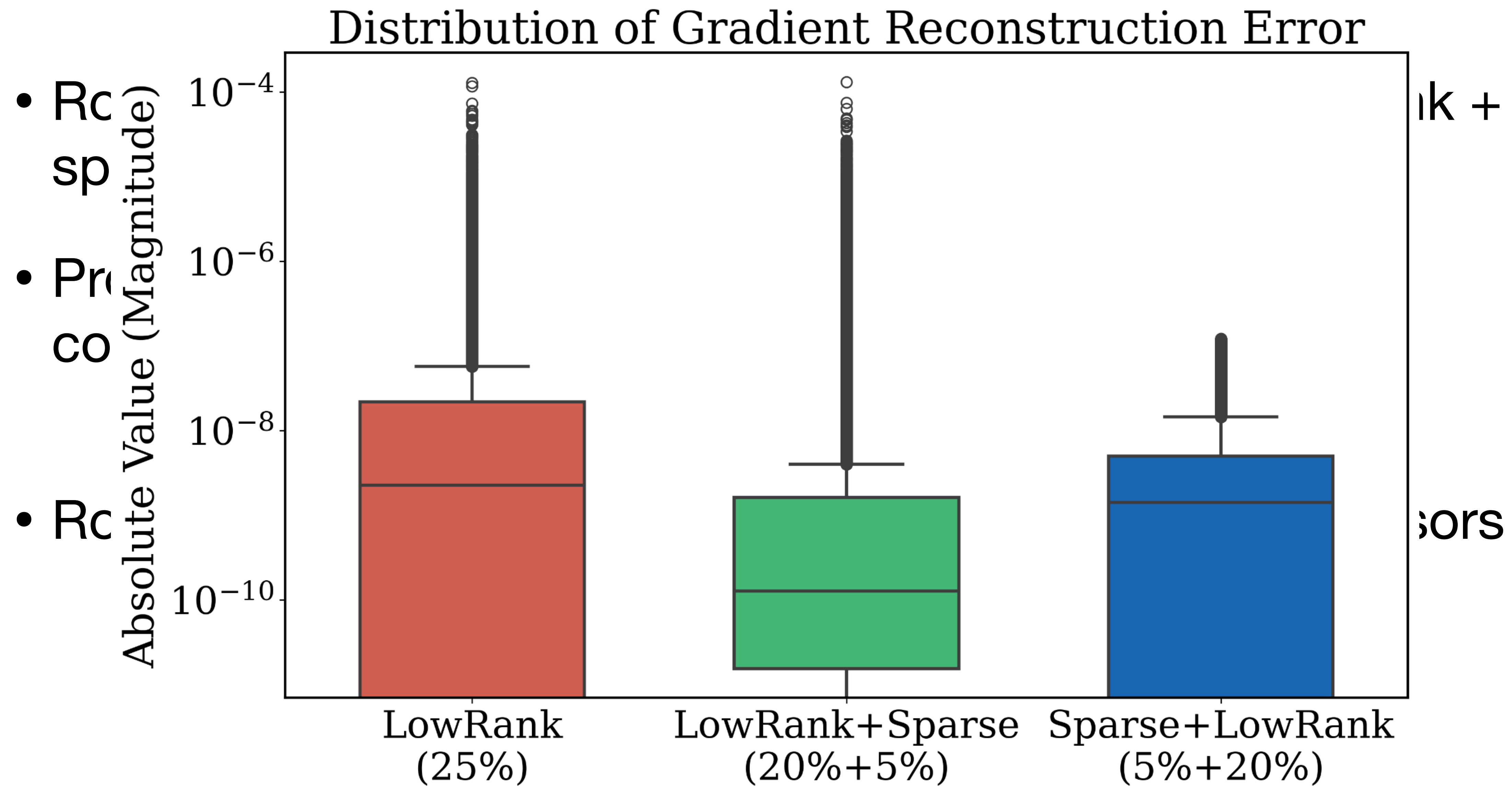
- **GaLore:** “Gradient becomes low-rank during training” [1]
- **InRank:** “Cumulative weight updates follow an incremental low-rank trajectory” [2]
- This **implies** that gradients **can be well-approximated by low-rank factorizations.**

Low-rank + Sparse

- Robust PCA [2] decomposes a matrix $M = L + S$ (low-rank + sparse outliers).
- Provably recovers the true subspace even under gross corruptions.

- Robust tensor c  =  $\oplus$  tensors

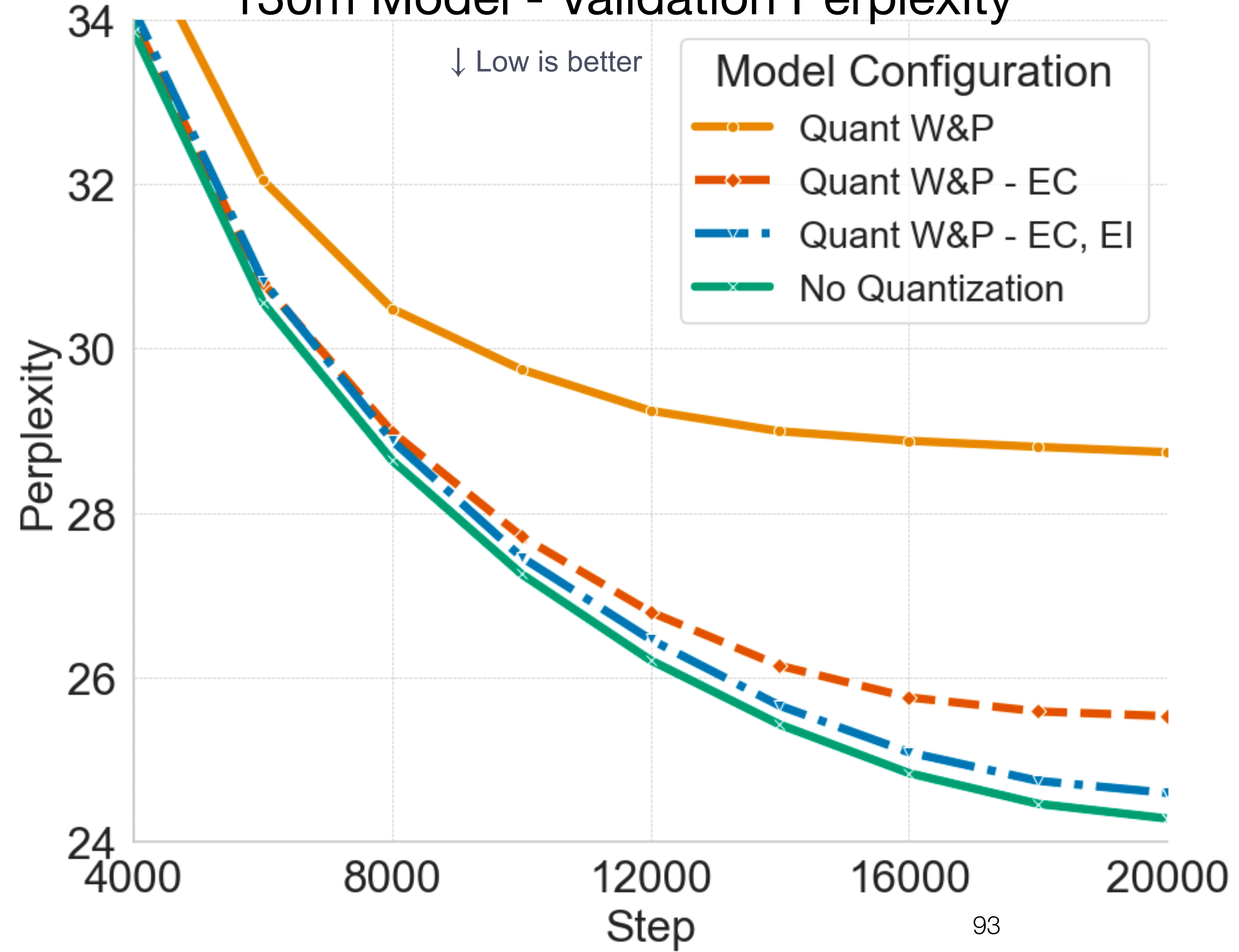
Sparse + low-rank can better capture outliers in the gradient



Extra

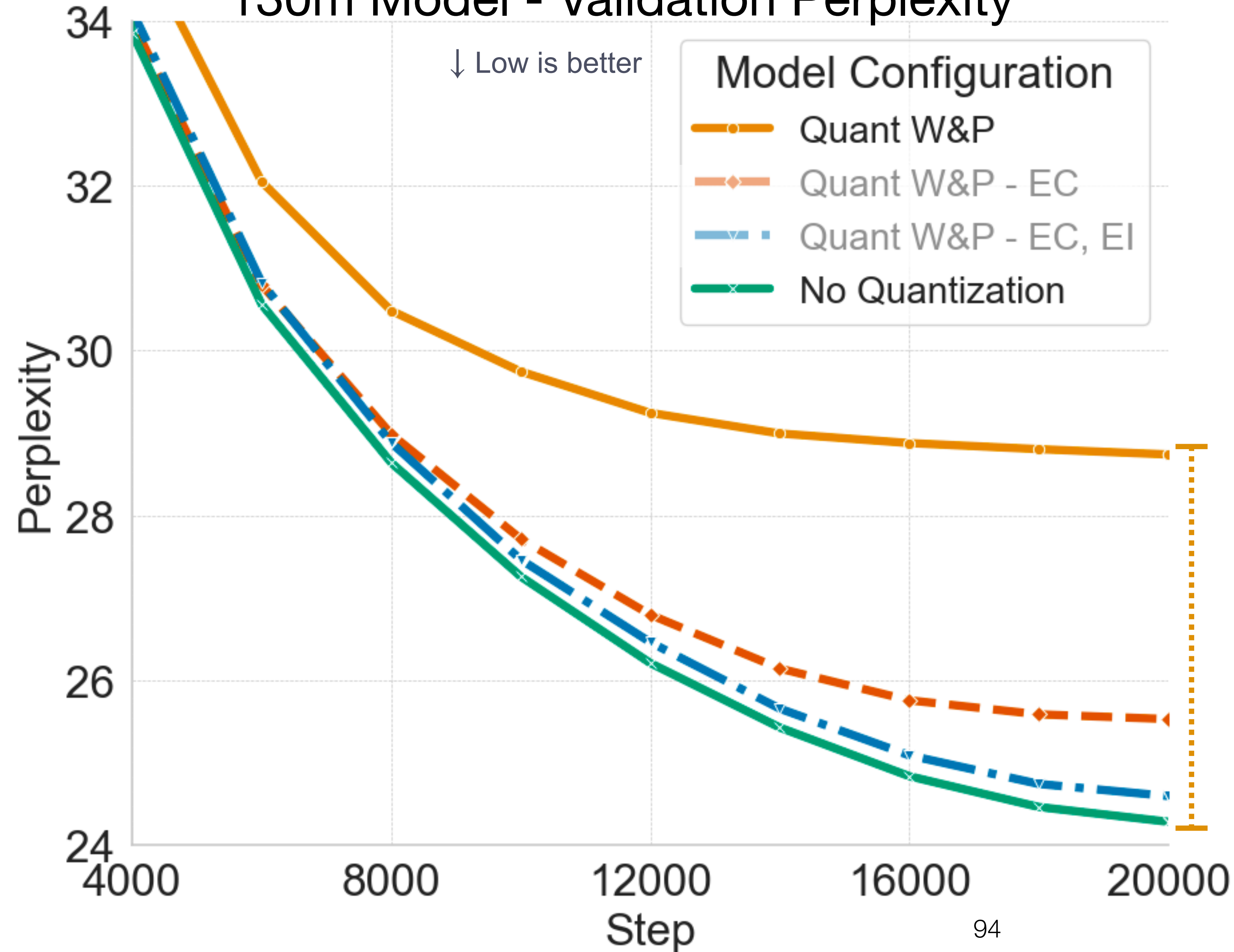
Component Ablations

130m Model - Validation Perplexity



Component Ablations

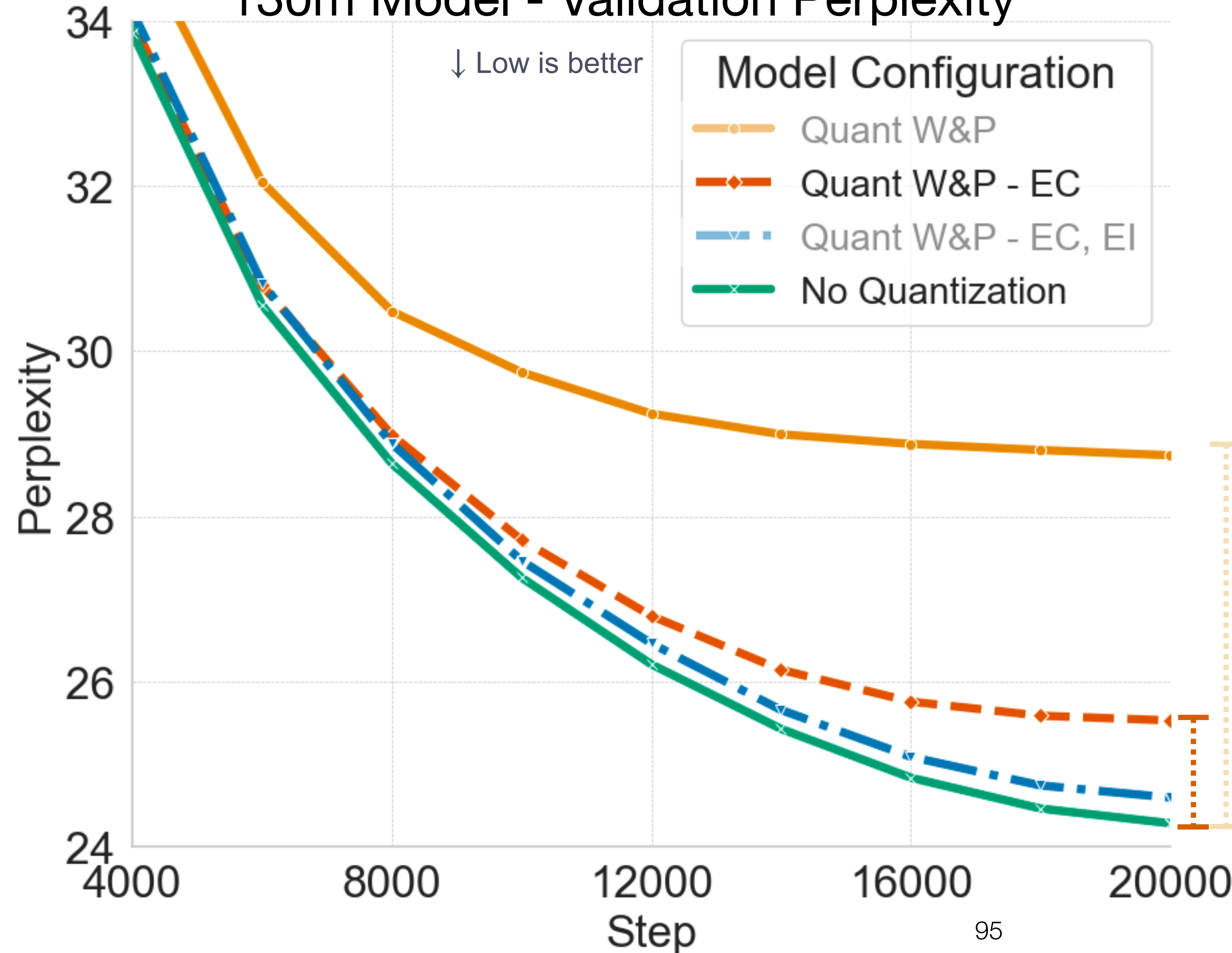
130m Model - Validation Perplexity



- **Simple 4-bit Quantization of W and P :** Flattens early.

Component Ablations

130m Model - Validation Perplexity

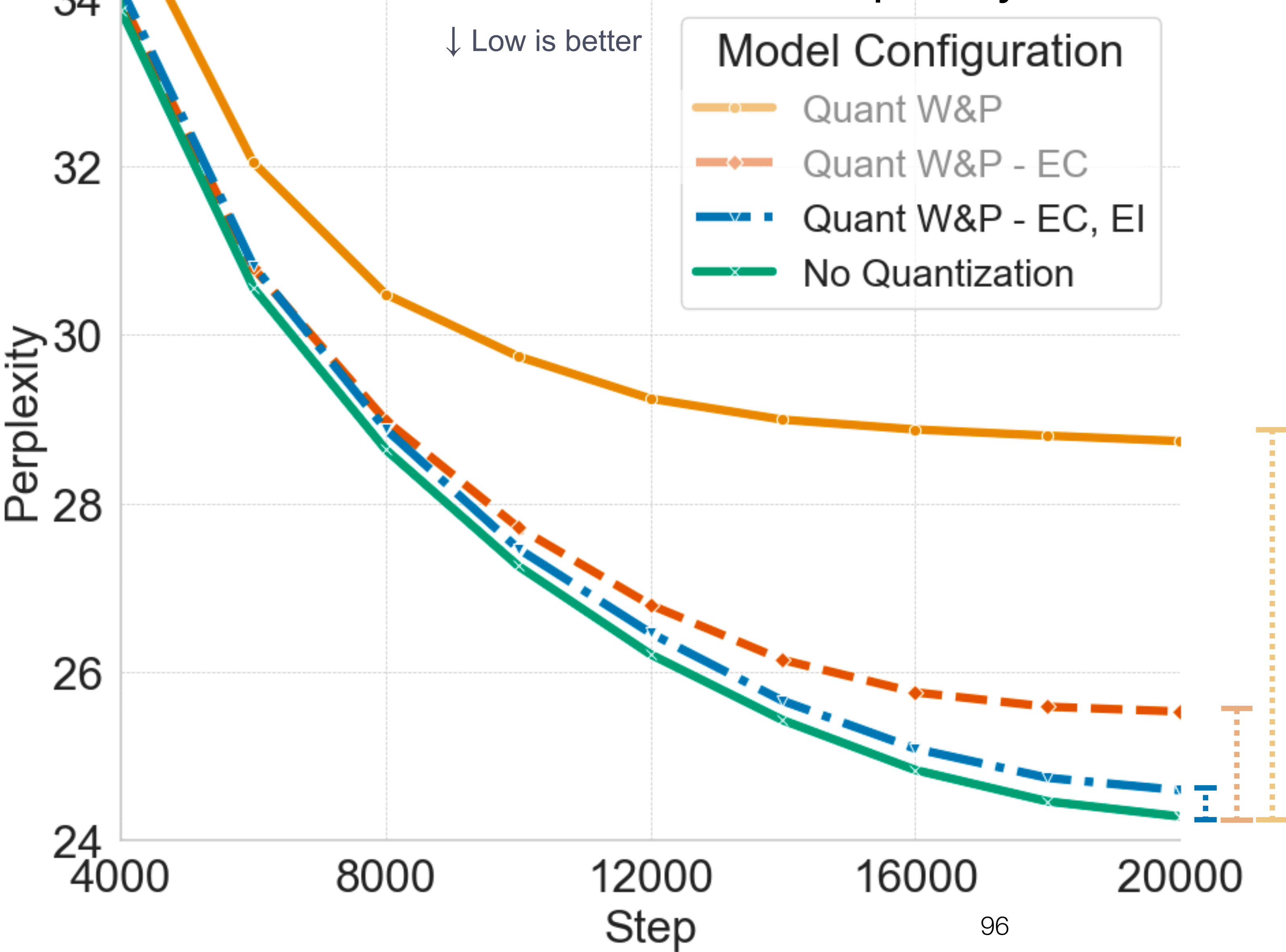


$$B = P^{-1}(W_q - W)$$

- **Simple 4-bit Quantization of W and P :** Flattens early.
- **Error Compensation (EC):** This gap is reduced by 72%.

Component Ablations

130m Model - Validation Perplexity



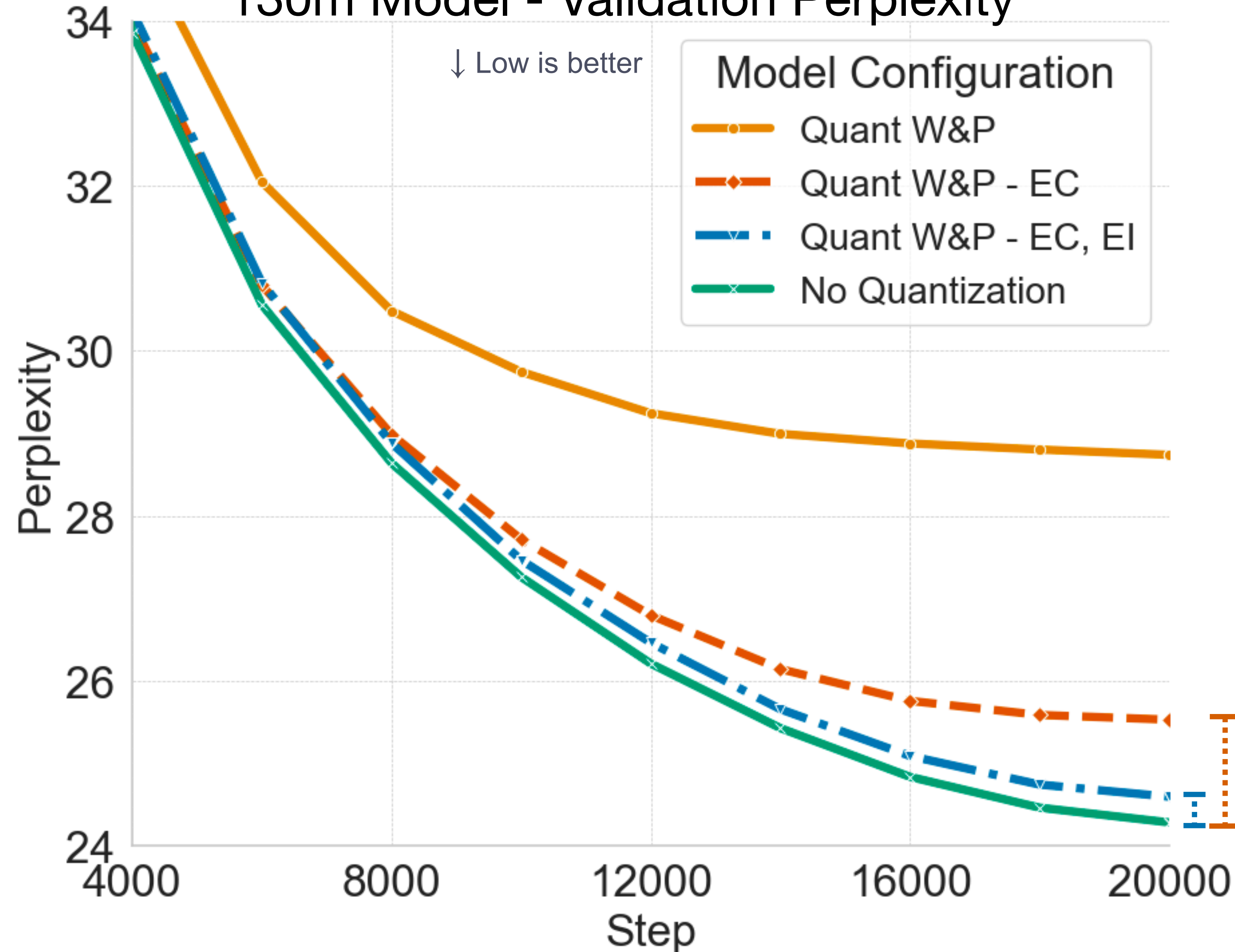
$$B = P^{-1}(W_q - W)$$

$$T_i = \tau + \psi^i$$

- **Simple 4-bit Quantization of W and P :** Flattens early.
- **Error Compensation (EC):** This gap is reduced by 72 %.
- **Exponentially increasing update intervals (EI) and EC:** the gap is reduced by 93 %.

Component Ablations

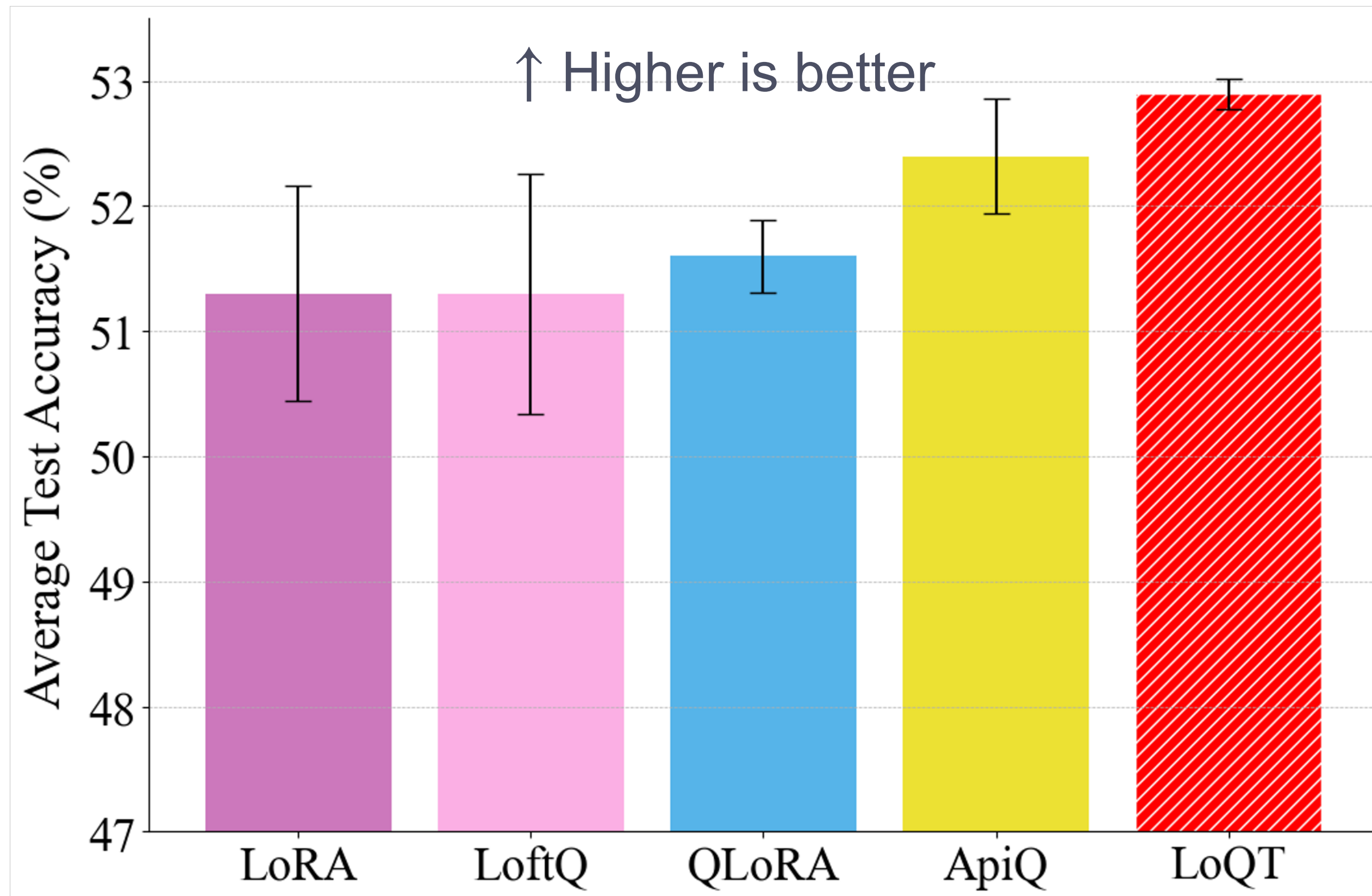
130m Model - Validation Perplexity



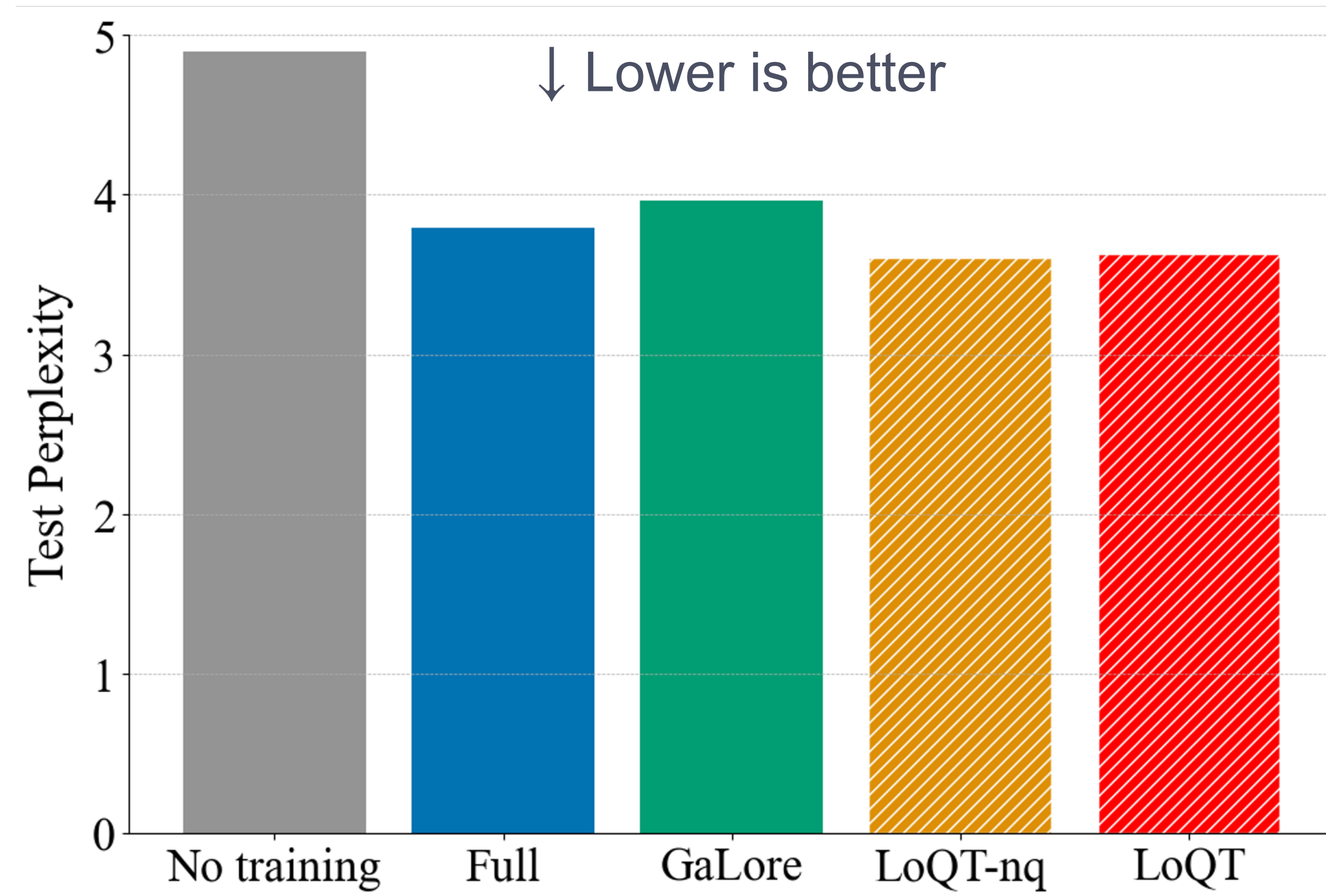
- **Simple 4-bit Quantization of W and P :** Flattens early.
- **Error Compensation (EC):** This gap is reduced by 72%.
- **Exponentially increasing update intervals (EI) and EC:** the gap is reduced by 93%.

Fine-tuning and Continued Pretraining

Fine-tuning
Llama-13B on GSM8K



Continued Pretraining
Llama 7B on Icelandic



Memory requirements by LLMs

- **Gradient Descent**

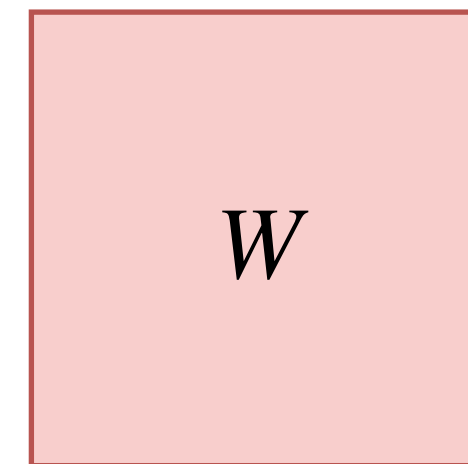
$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(G_t)$$

Optimizer

Learning Rate

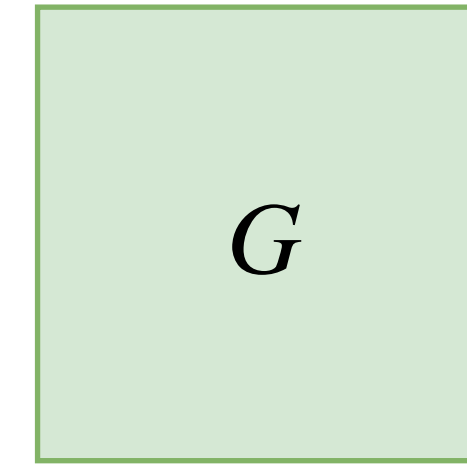
$$G_t = -\nabla_W \varphi_t(W_t) \in \mathbb{R}^{m \times n}$$

Weights



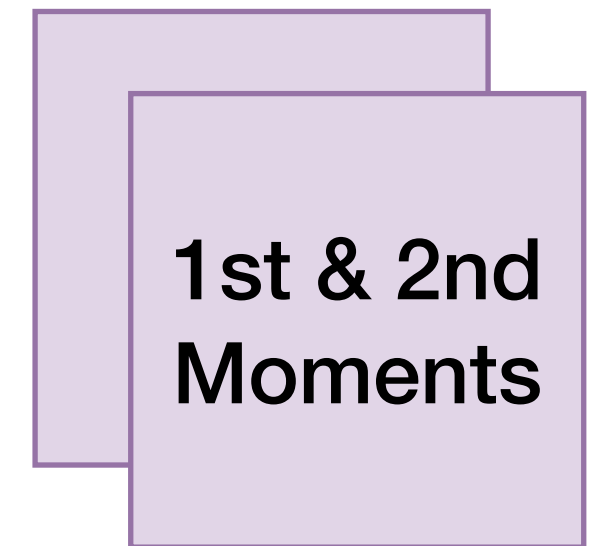
mn

Gradients



mn

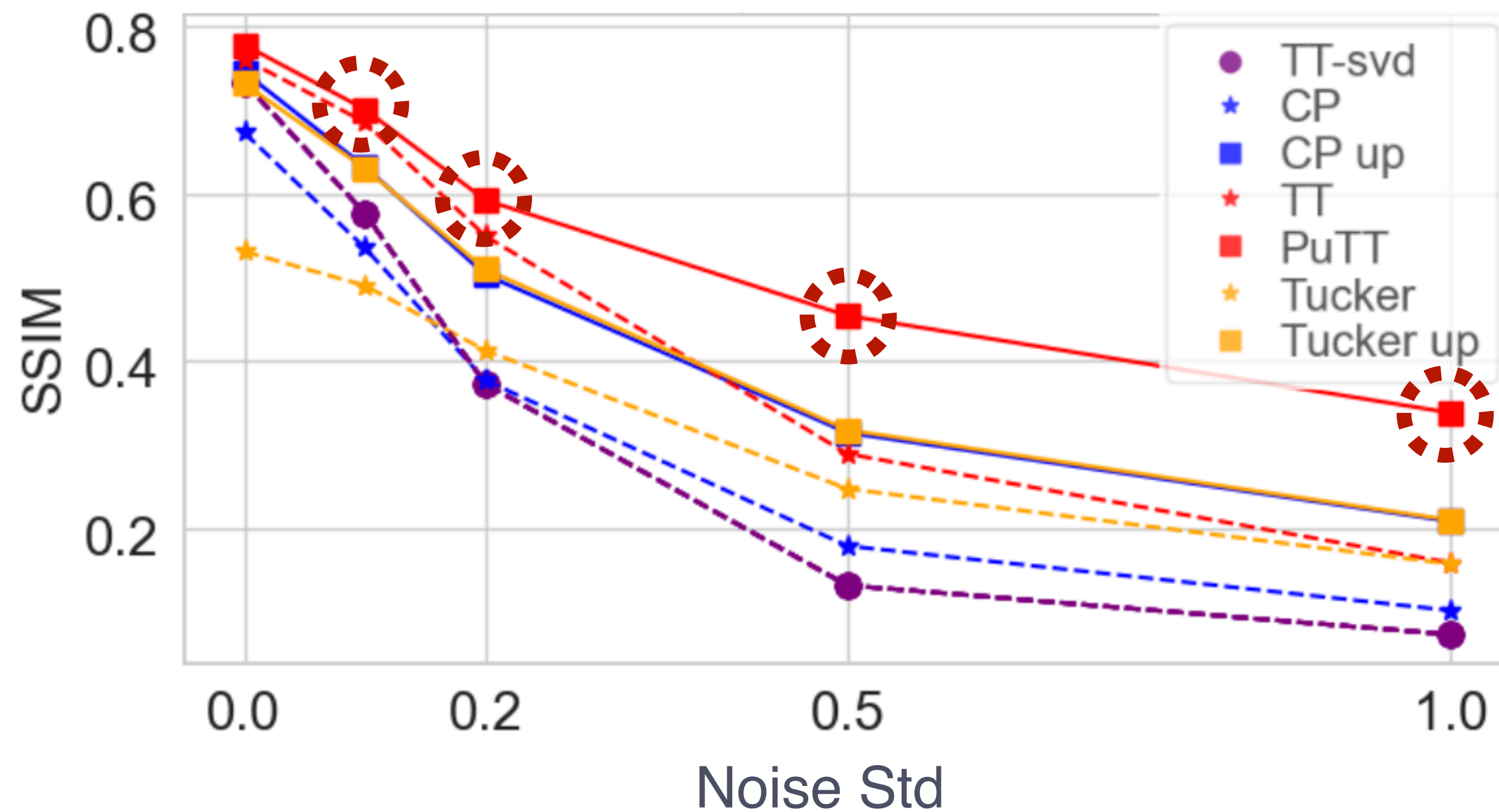
Optimizer States



$2mn$

Noisy and Incomplete data

Noisy Data



Incomplete Data



Memory requirements by LLMs

- Gradient Descent

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(G_t)$$

Weights



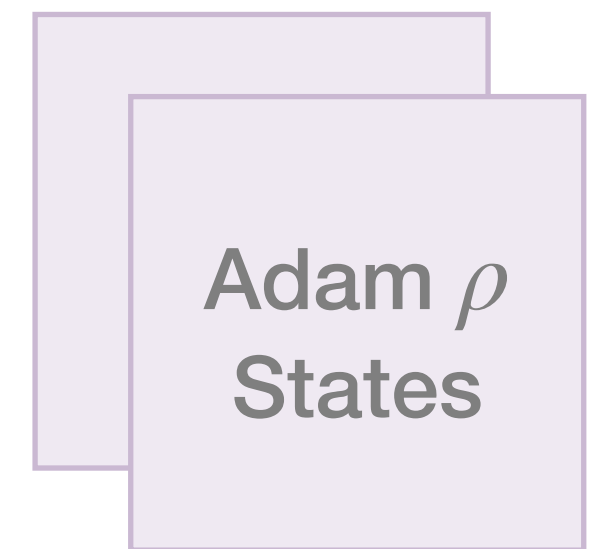
mn

Gradients



mn

Optimizer States

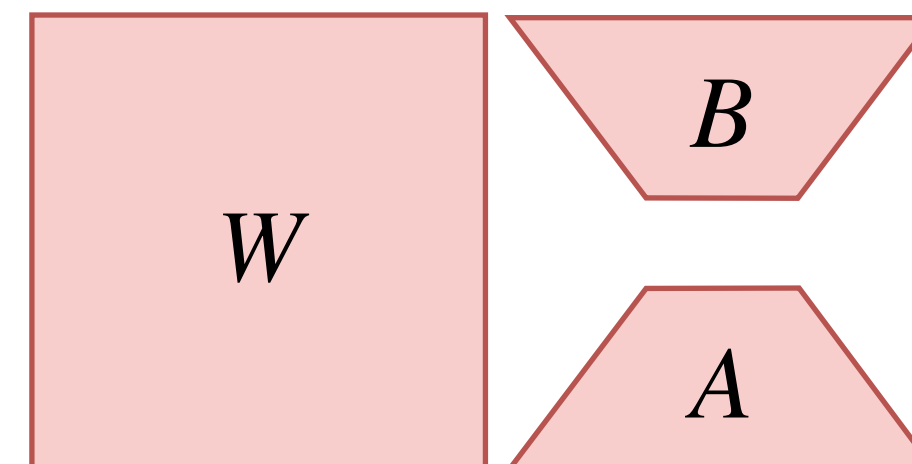


$2mn$

- LoRA (Hu et. al. 2021)

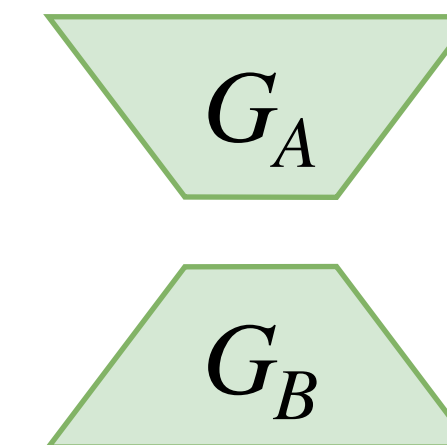
$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(A)\rho(B)$$

↓ Increase



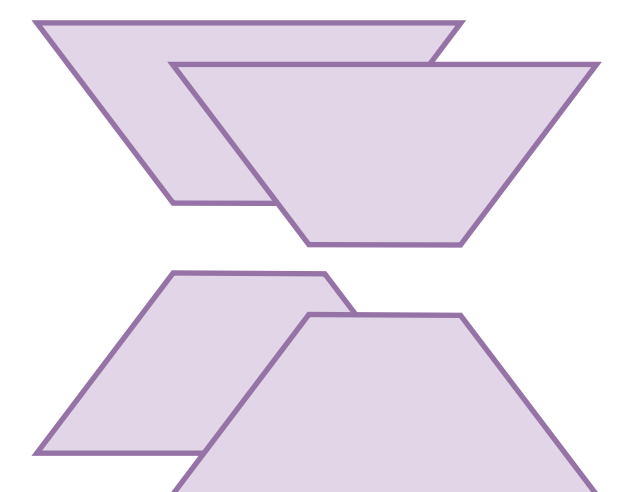
$mn + nr + mr$

↓ Decrease



$nr + mr$

↓ Decrease



$2nr + 2mr$

Low-Rank Projection of Gradient

- **GaLore** (Zhao et. al., 2024)

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\in \mathbb{R}^{m \times r}} \underbrace{\rho(P_t^\top G_t)}_{\in \mathbb{R}^{r \times n}}$$

Gradient
Descent

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(G_t)$$

LoRA

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(A)\rho(B)$$

Low-Rank Projection of Gradient

- **GaLore** (Zhao et. al., 2024)

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\in \mathbb{R}^{m \times r}} \underbrace{\rho(P_t^\top G_t)}_{\in \mathbb{R}^{r \times m}}$$

$$\text{SVD}(G_t) = U \Sigma V^\top, \quad P_t = U[:, :r]$$

Weights



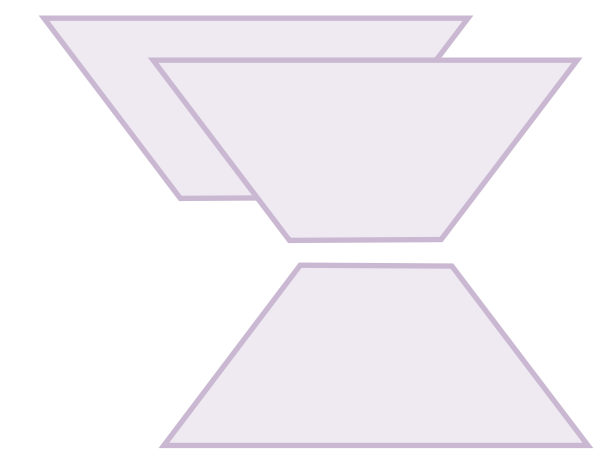
mn

Gradients



mn

Optimizer States



$mr + 2nr$

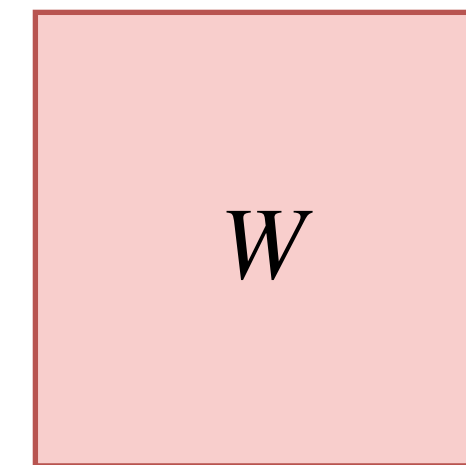
Low-Rank Projection of Gradient

- **GaLore** (Zhao et. al., 2024)

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\in \mathbb{R}^{m \times r}} \underbrace{\rho(P_t^\top G_t)}_{\in \mathbb{R}^{r \times n}}$$

$$SVD(G_t) = U \Sigma V^\top, P_t = U[:, :r]$$

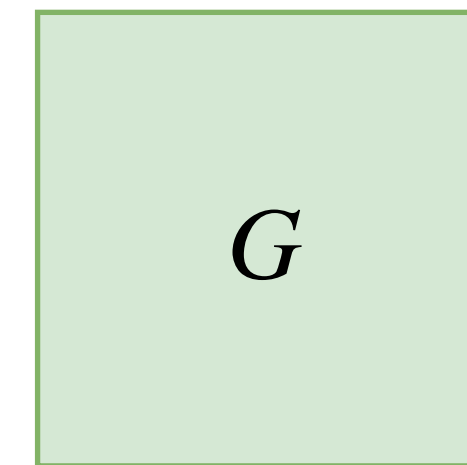
Weights



W

mn

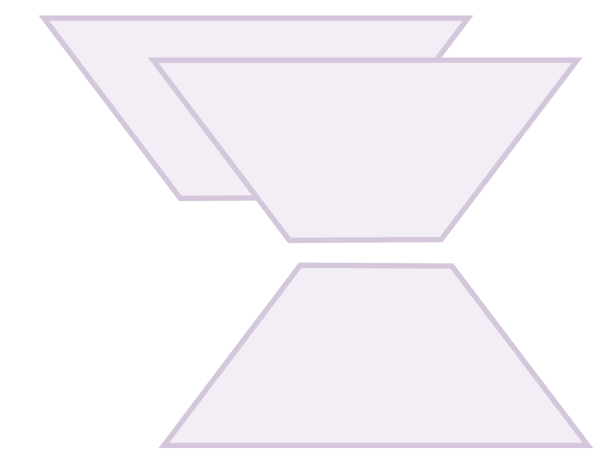
Gradients



G

mn

Optimizer States



$mr + 2nr$

LoQT - Low Rank Adapters for Quantized Pre-Training

- **GaLore** (Zhao et. al., 2024)

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\in \mathbb{R}^{m \times r}} \underbrace{\rho(P_t^\top G_t)}_{\in \mathbb{R}^{r \times n}}$$

Weights



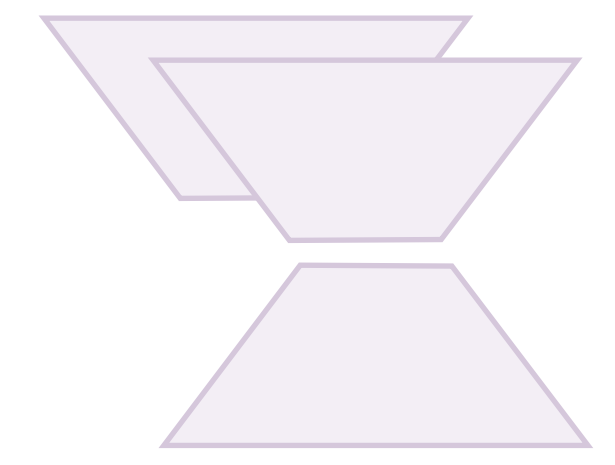
mn

Gradients



mn

Optimizer States



$mr + 2nr$

- **LoQT**

LoRA factors: $\swarrow$ A $\searrow$ B

LoRA: $W_T = W_0 + \eta \sum_{t=0}^{T-1} \rho(A) \rho(B)$

LoQT - Low Rank Adapters for Quantized Pre-Training

- **GaLore** (Zhao et. al., 2024)

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\in \mathbb{R}^{m \times r}} \underbrace{\rho(P_t^\top G_t)}_{\in \mathbb{R}^{r \times n}}$$

- **LoQT**

LoRA factors:

$$W_T = \underbrace{W_0}_{\text{Frozen}} + \eta \sum_{t=0}^{T-1} \underbrace{P_t}_{\text{Optimize}} \rho(\underbrace{B_t}_{\text{Optimize}})$$

Weights



mn

↓ Increase

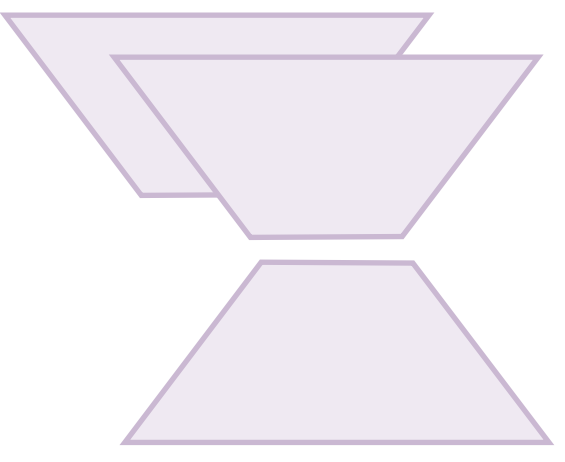
Gradients



mn

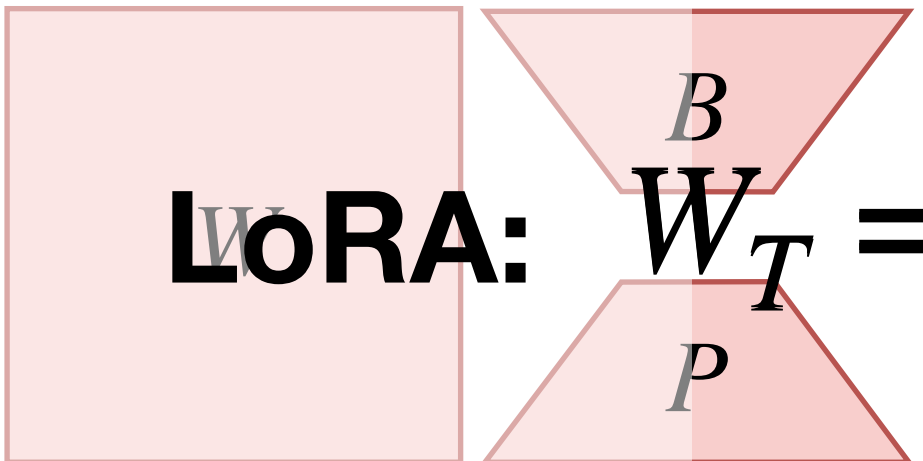
↓ Decrease

Optimizer States



$mr + 2nr$

↓ Decrease



$mn + nr + mr$

LoRA:

$$W_T = W_0 + \eta \sum_{t=0}^{T-1} \underbrace{\rho(G_B)}_{\text{Gradients}} \underbrace{\rho(A)}_{\text{Weights}} \underbrace{\rho(B)}_{\text{Optimizer States}}$$

nr

$2nr$

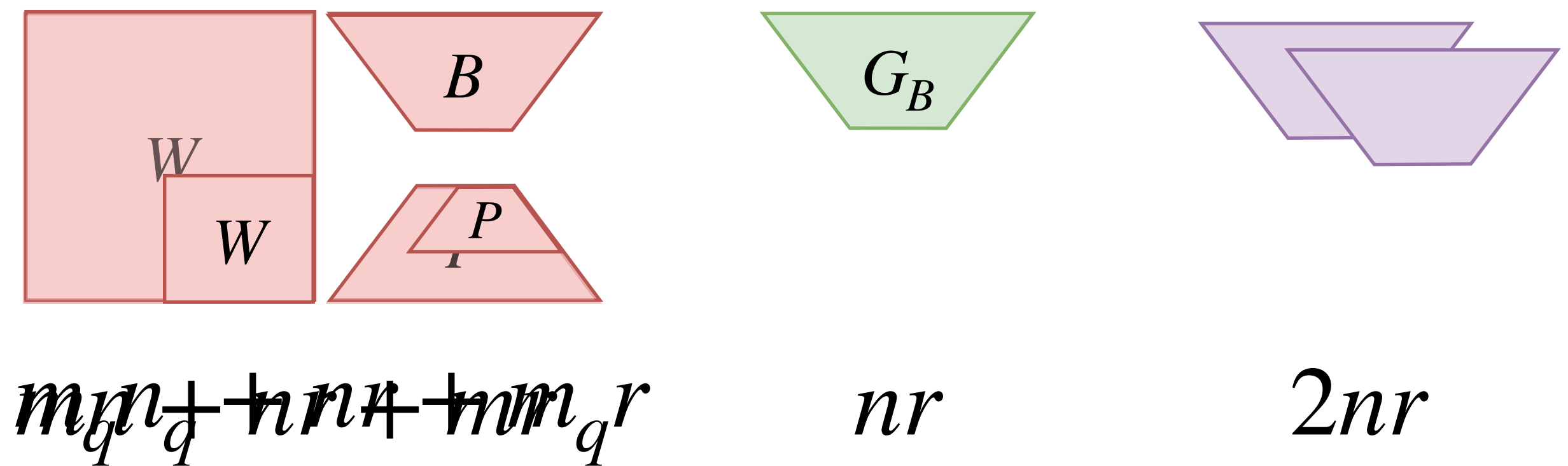
❄️ Frozen

🔥 Optimize

LoQT - Quantization

Compress weights with **4-bit** Normal Float quantization - Dettmers et al. (2023)

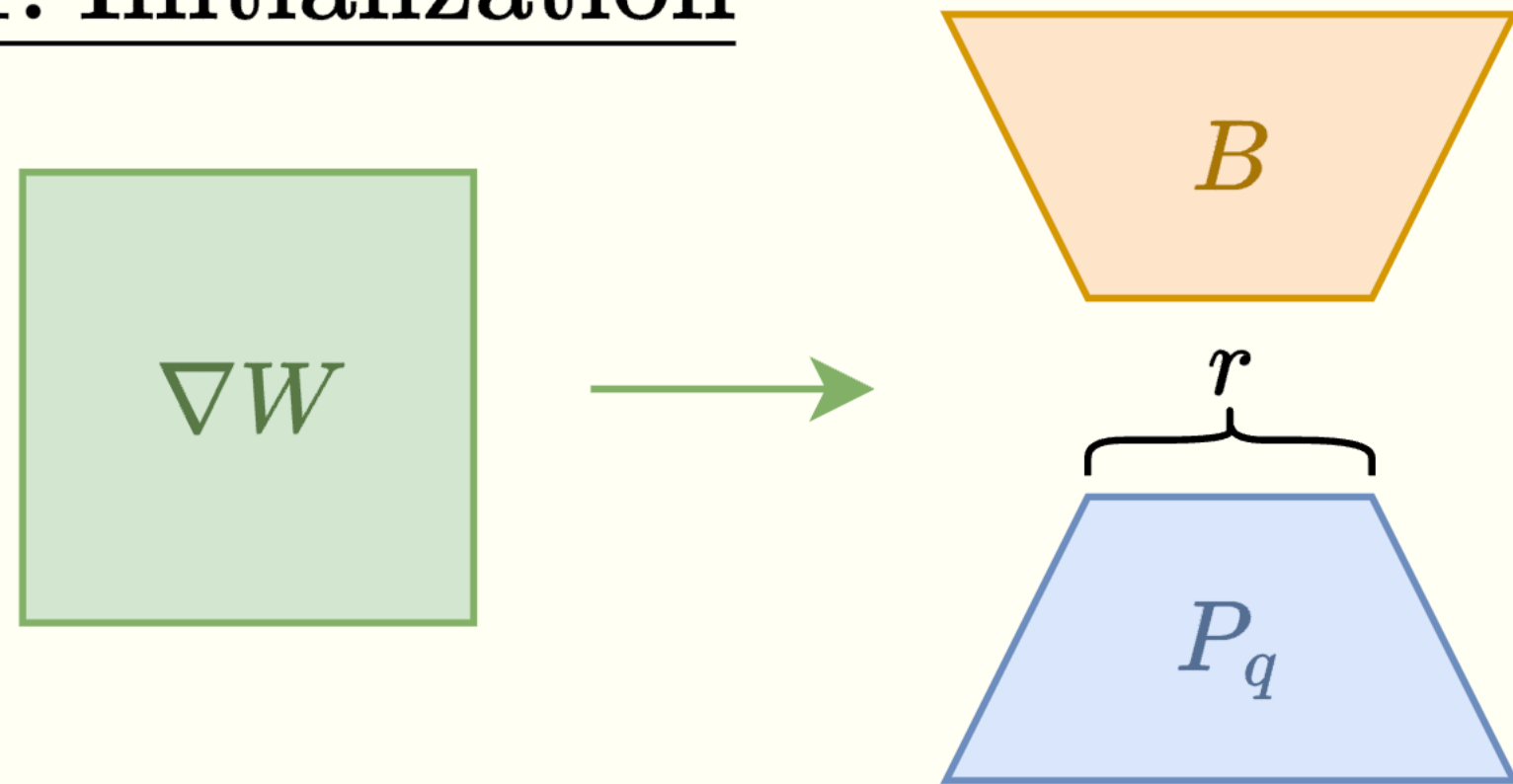
$$W_T = W_\theta^q + \eta \sum_{t=0}^{T-1} P_t^q p(B_t)$$



75 % less memory for W and P compared to FP16

LoQT

1. Initialization



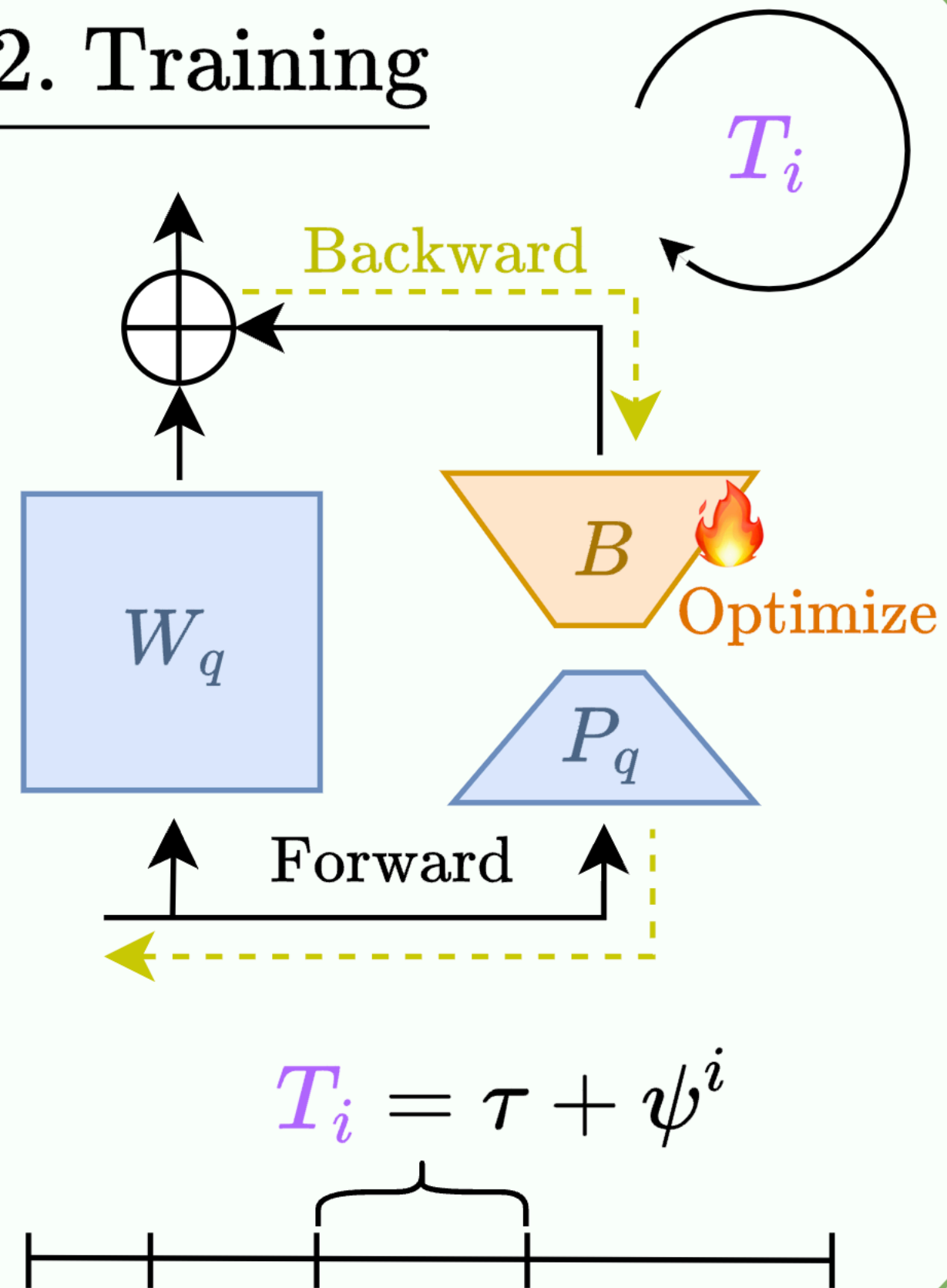
$$\text{SVD}(\nabla W) = U\Sigma V^\top$$

$$P = U[:, :r]$$

$$P_q = q_{\text{NF4}}(P) \quad W_q = q_{\text{NF4}}(W)$$

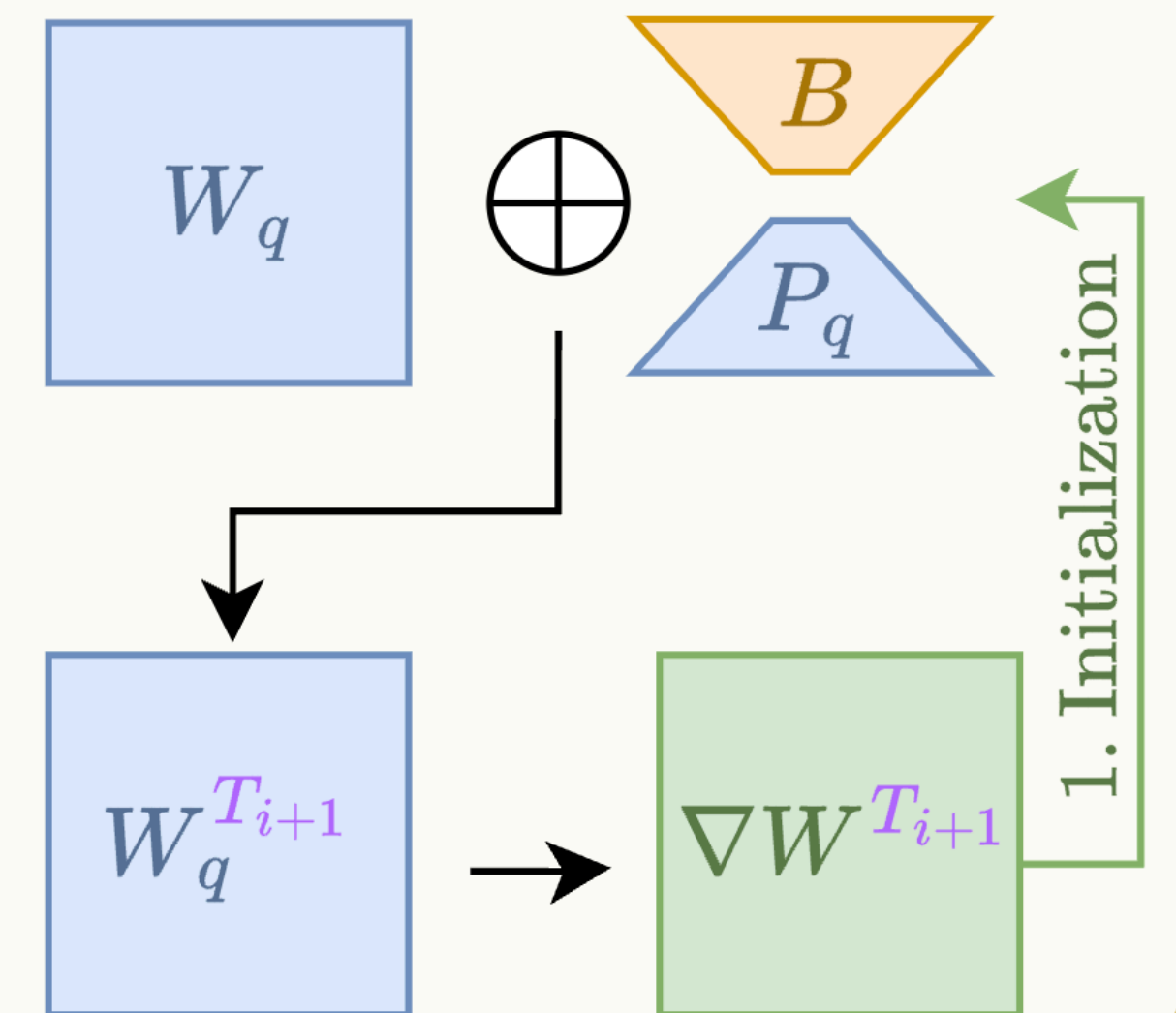
$$B = P^{-1}(W_q - W)$$

2. Training



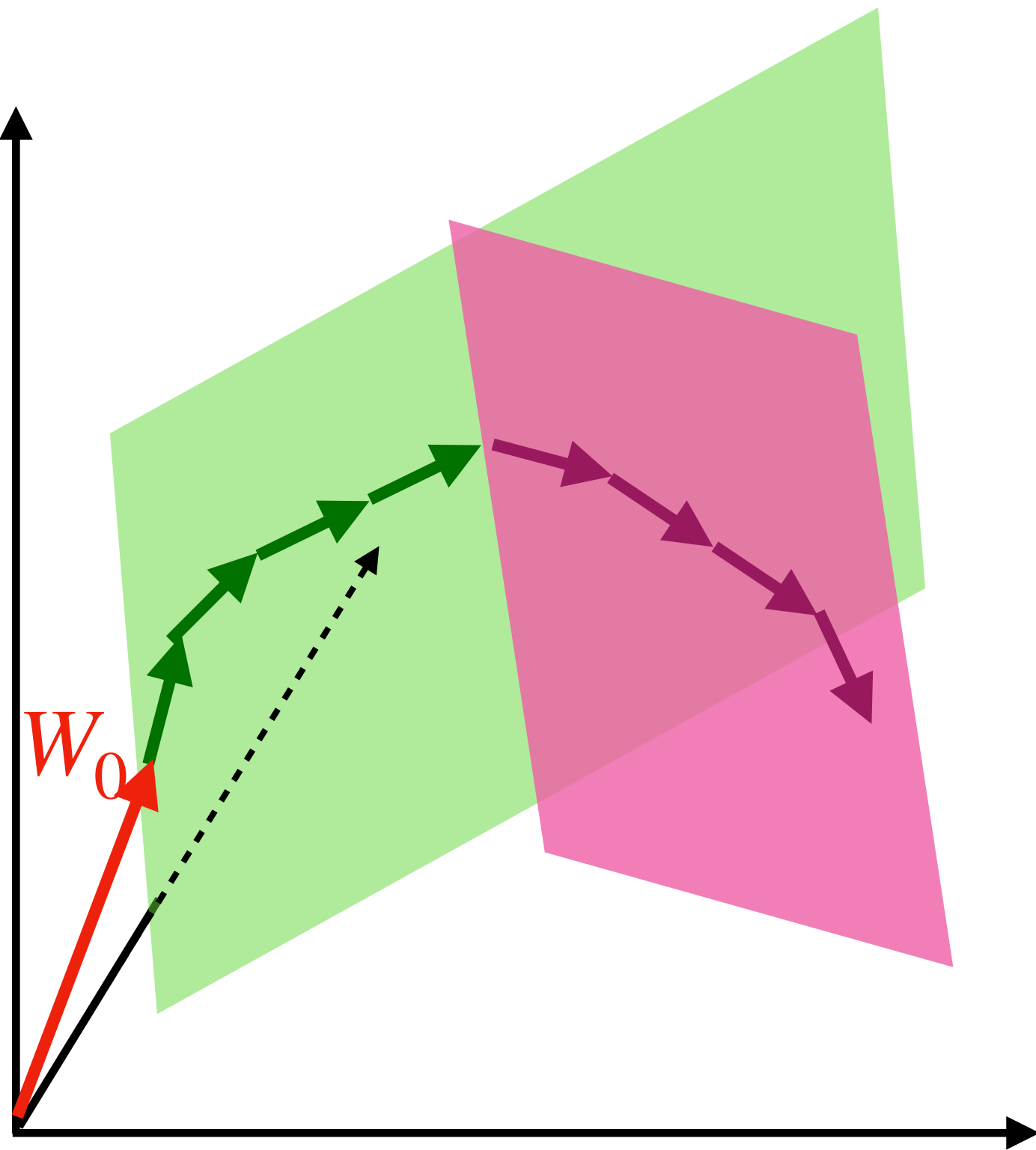
3. Merging

$$W_q^{T_{i+1}} = W_q^{T_i} + P_q^{T_i} B$$

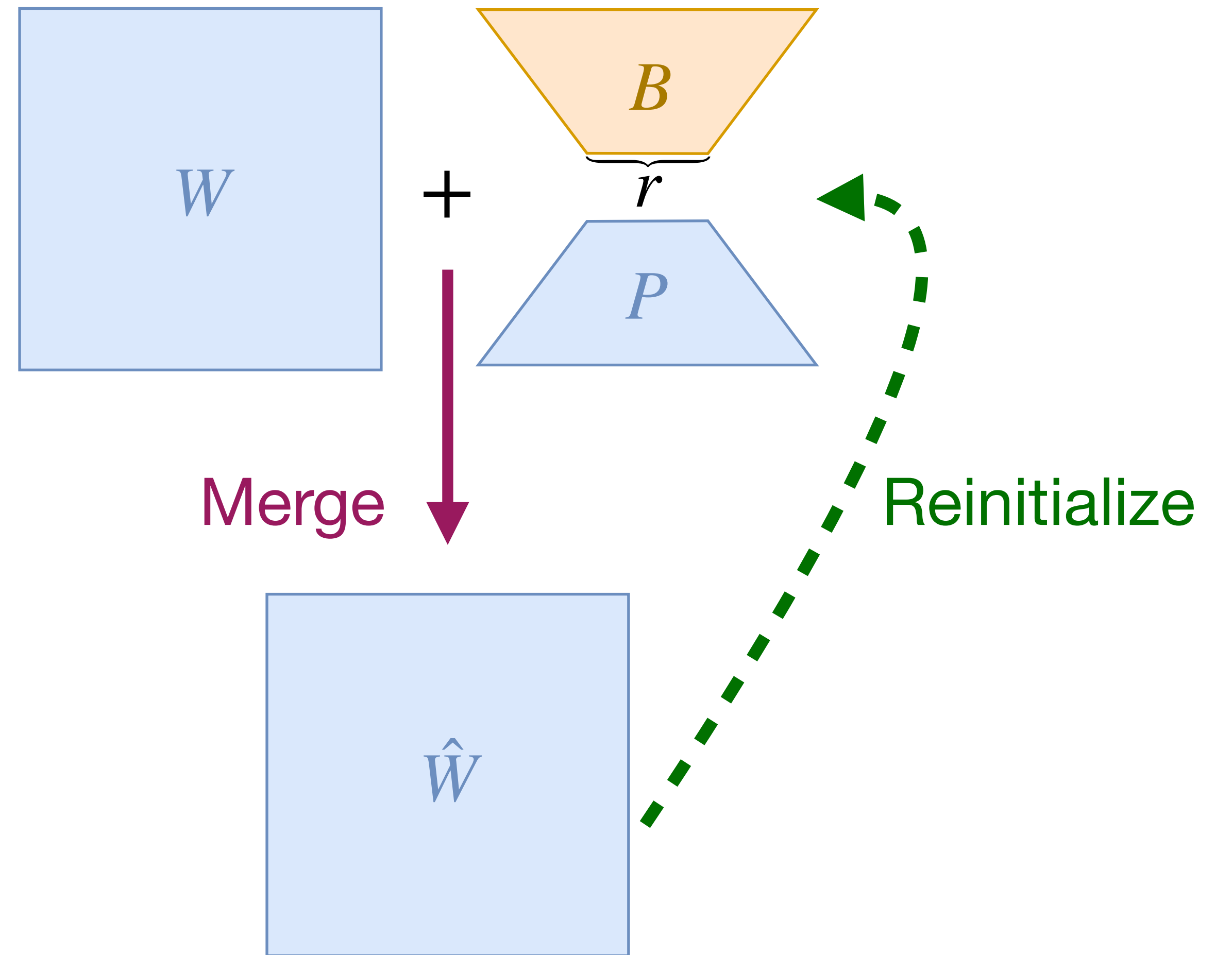


$$W_N = W_0 + P_0 B_0 + P_1 B_1 + \dots + P_N B_N$$

LoQT Updating W

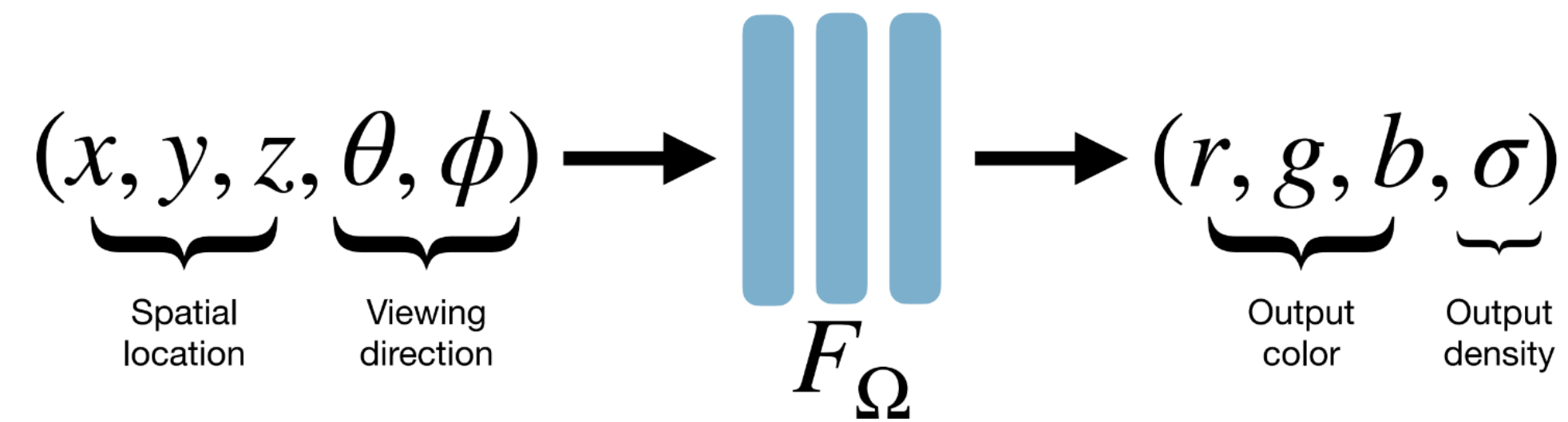


$$W_N = W_0 + P_0 B_0 + P_1 B_1 + \dots + P_N B_N$$



Feature Grids and Radiance Fields

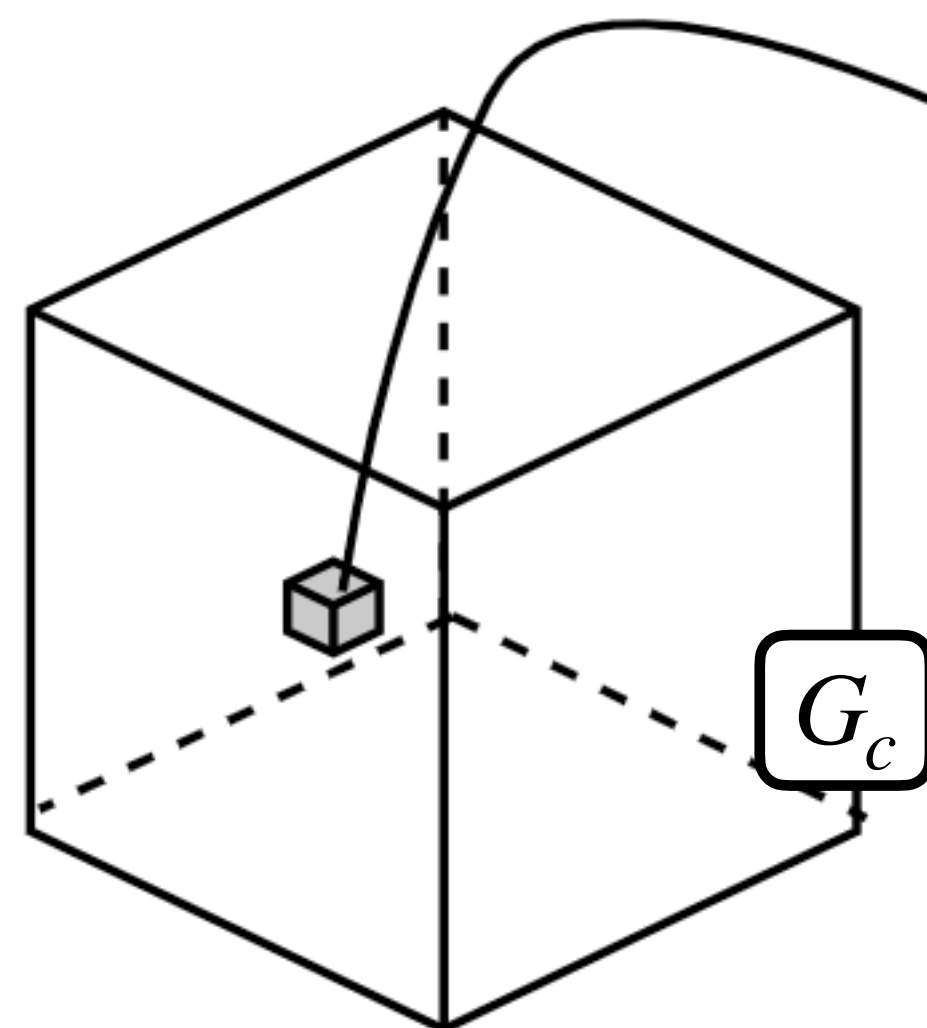
- Goal



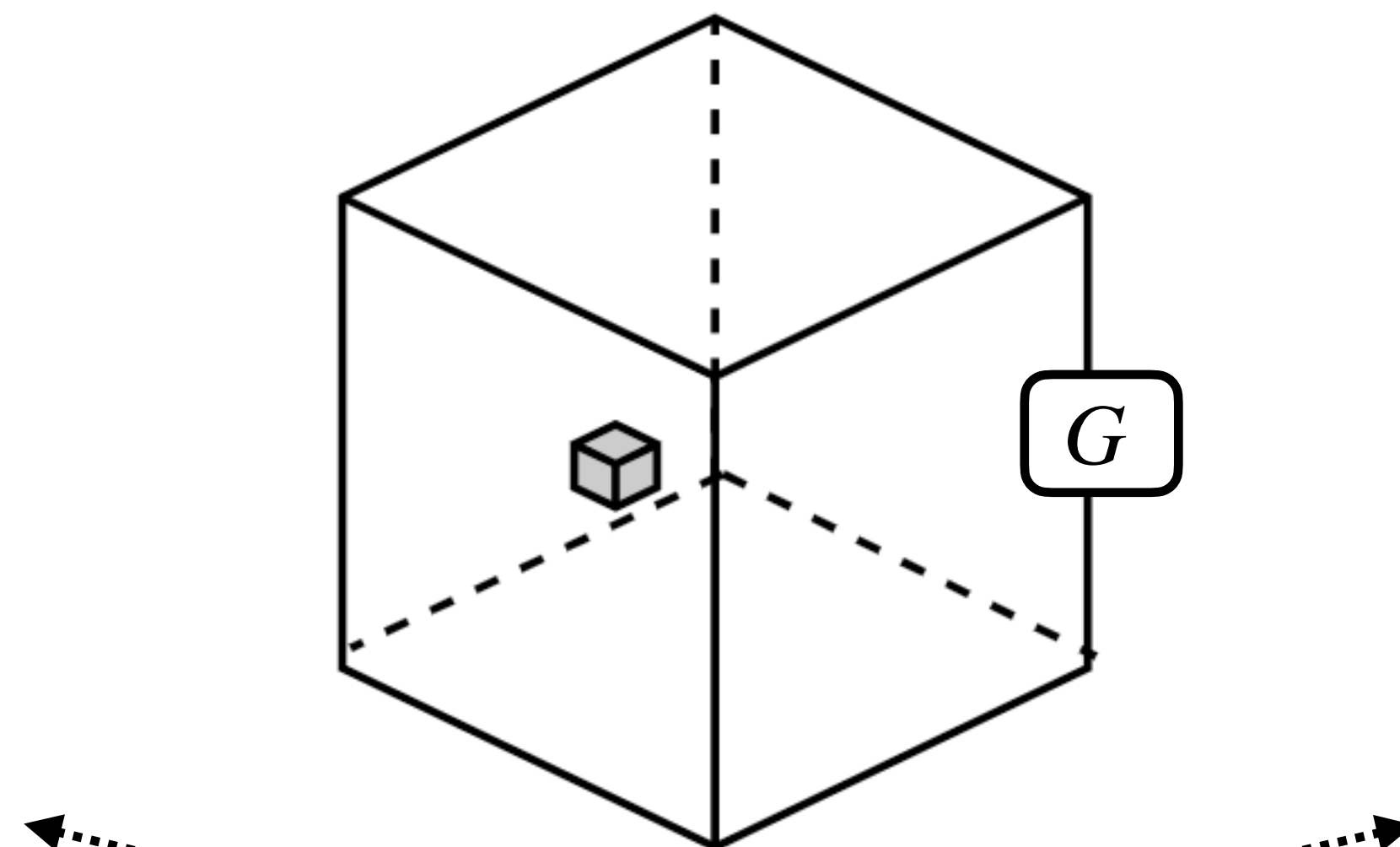
Shader



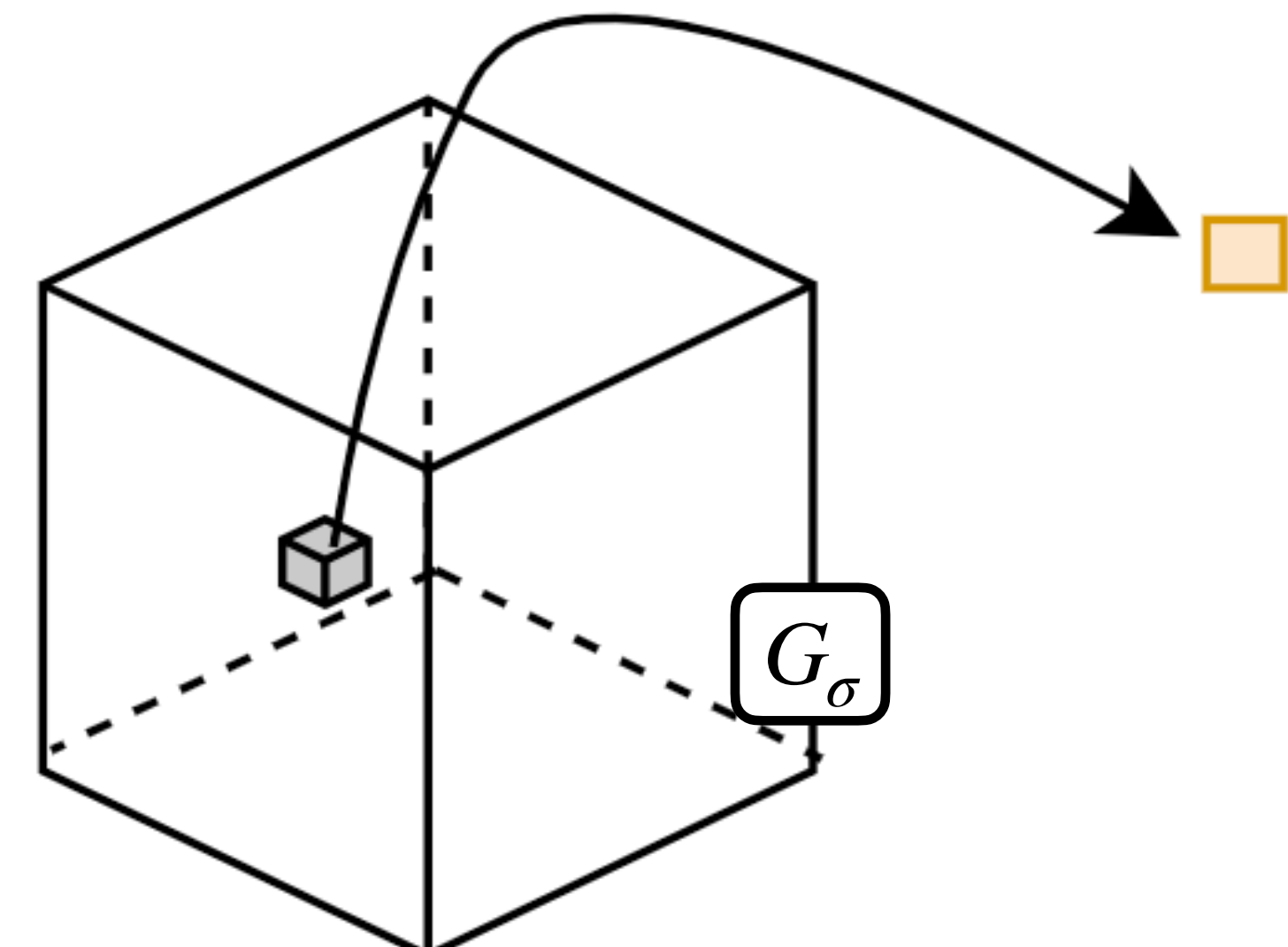
Factorize using QTT with D cores for a grid of size $(2^D)^3$



Appearance Grid

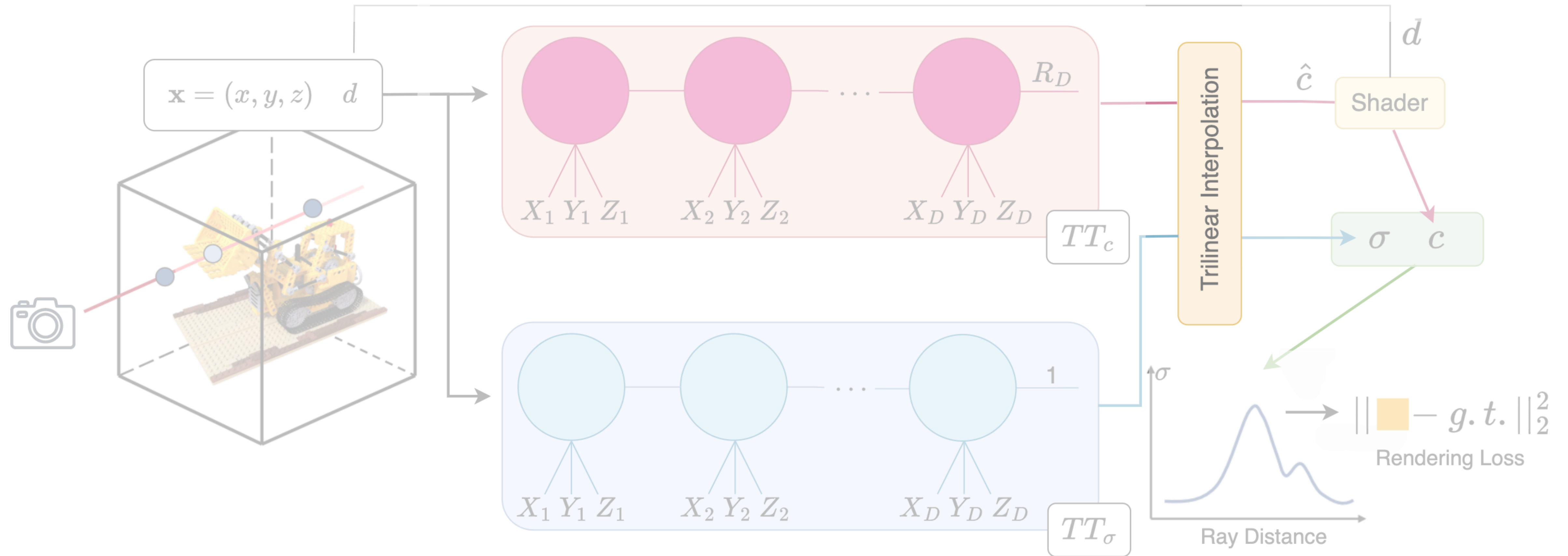


Split

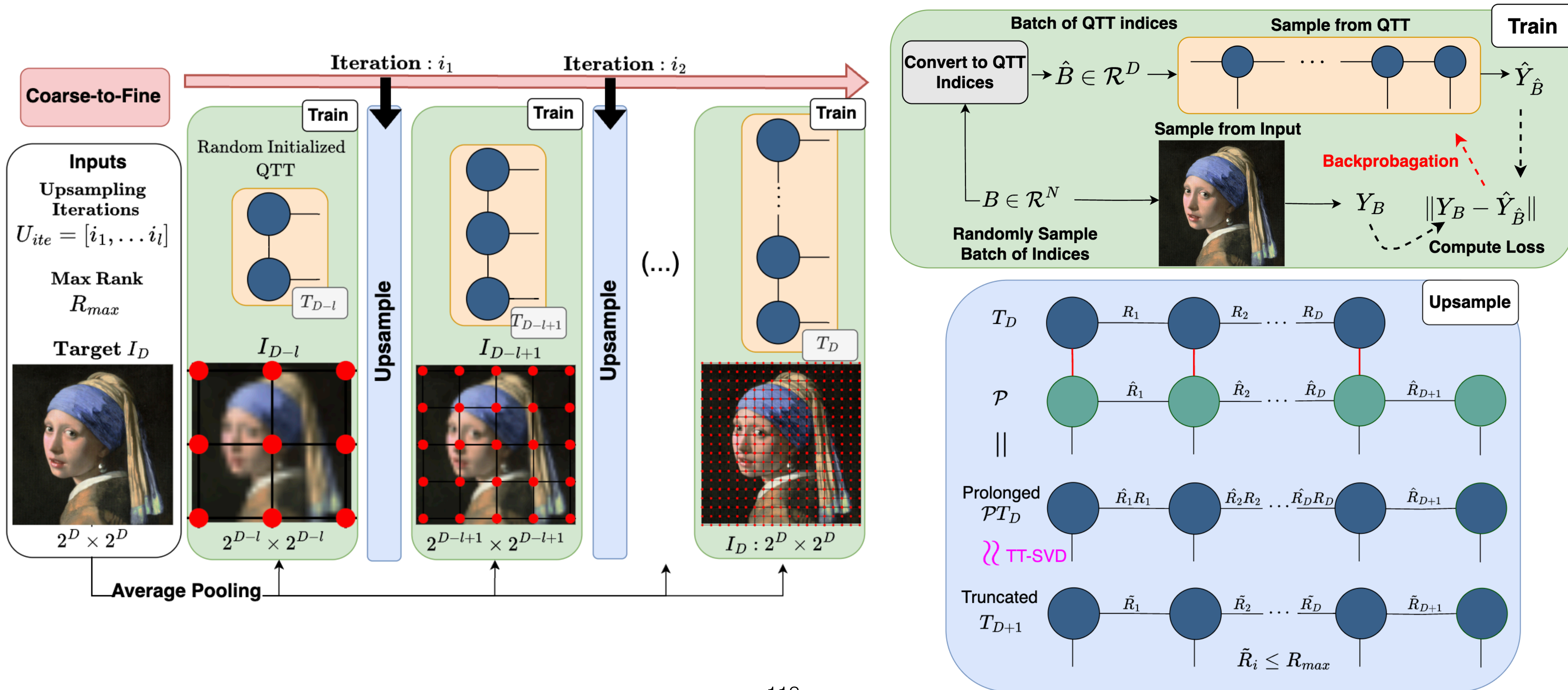


Geometry Grid

PuTT-NeRF Pipeline



Method Components



Method Components

