Enabling Sparsity in AI Workloads

Maryam Mehri Dehnavi

University of Toronto

NVIDIA Research

Outline

Model Compression in Pretraining

- **SLoPe**: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs Lazy Low-rank Adapters [ICLR'25]

Model Compression at Inference

- **SLiM**: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression [ICML'25]

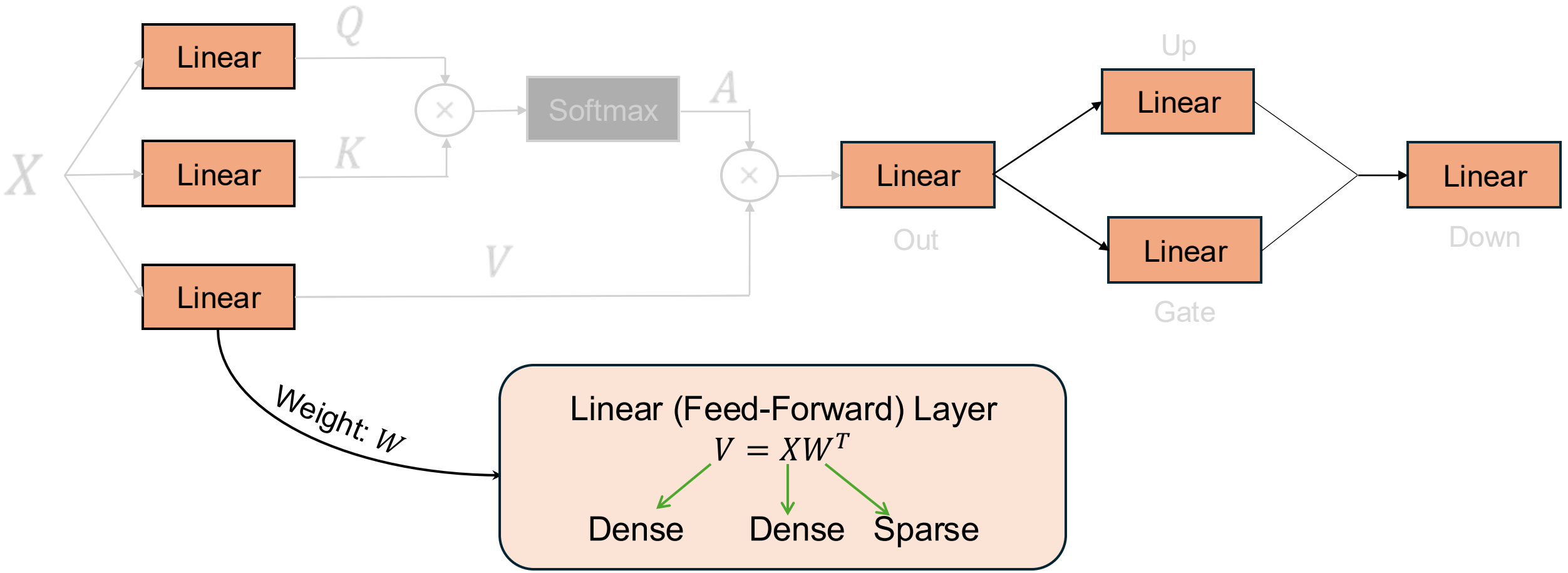
Compression Trinity Webpage

A Compiler for Rapid Prototyping of Sparsity and Quantization of Models on GPUs!

- **STOICC**: A Compiler for Tile-based Sparsity

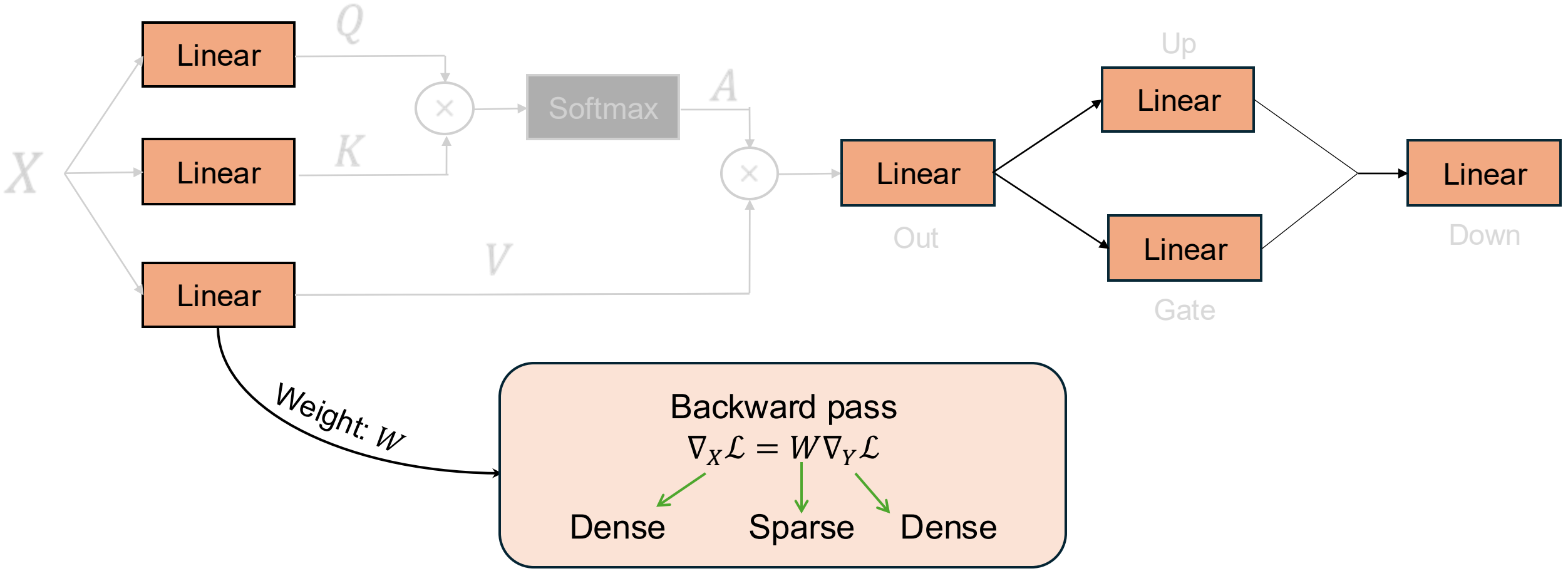
STOICC

LLM Compute Graph | Weight Sparsity

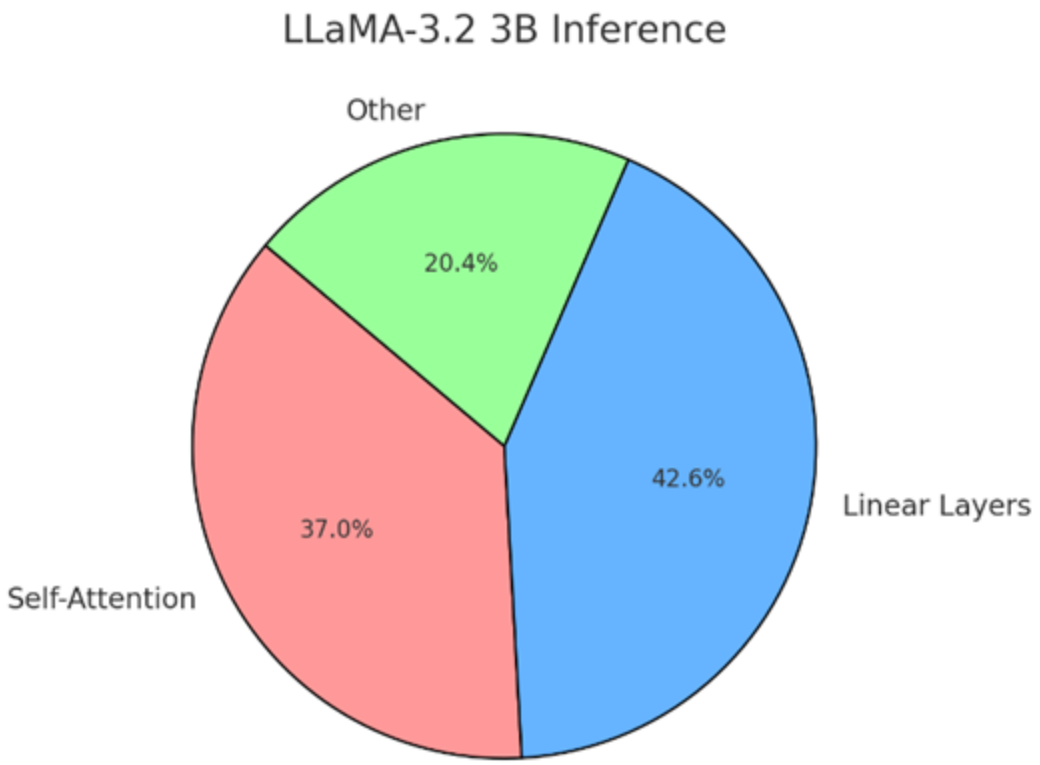


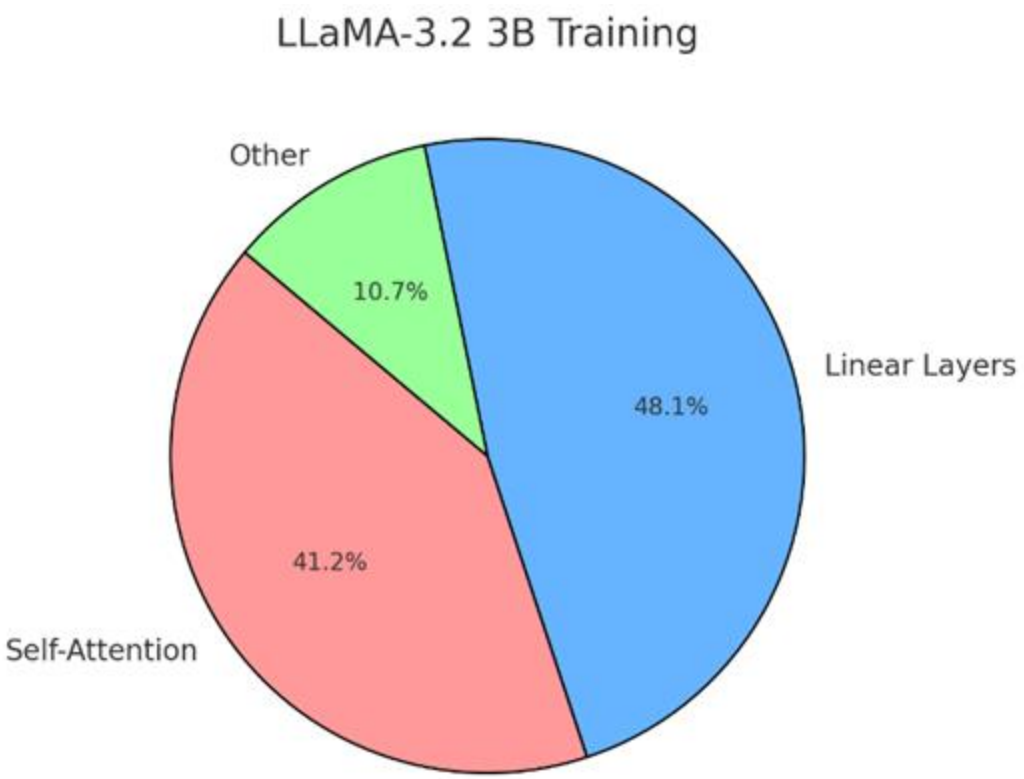
LLM Compute Graph | Weight Sparsity

Y: Output
X: Input
W: Weight
∇: Gradient
$\mathcal{L}$: Loss



LLM Compute Graph | Time Breakdown





Linear Layers are the bottleneck in both training and inference.

Outline

Model Compression in Pretraining

- **SLoPe**: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs Lazy Low-rank Adapters [ICLR'25]

Model Compression at Inference

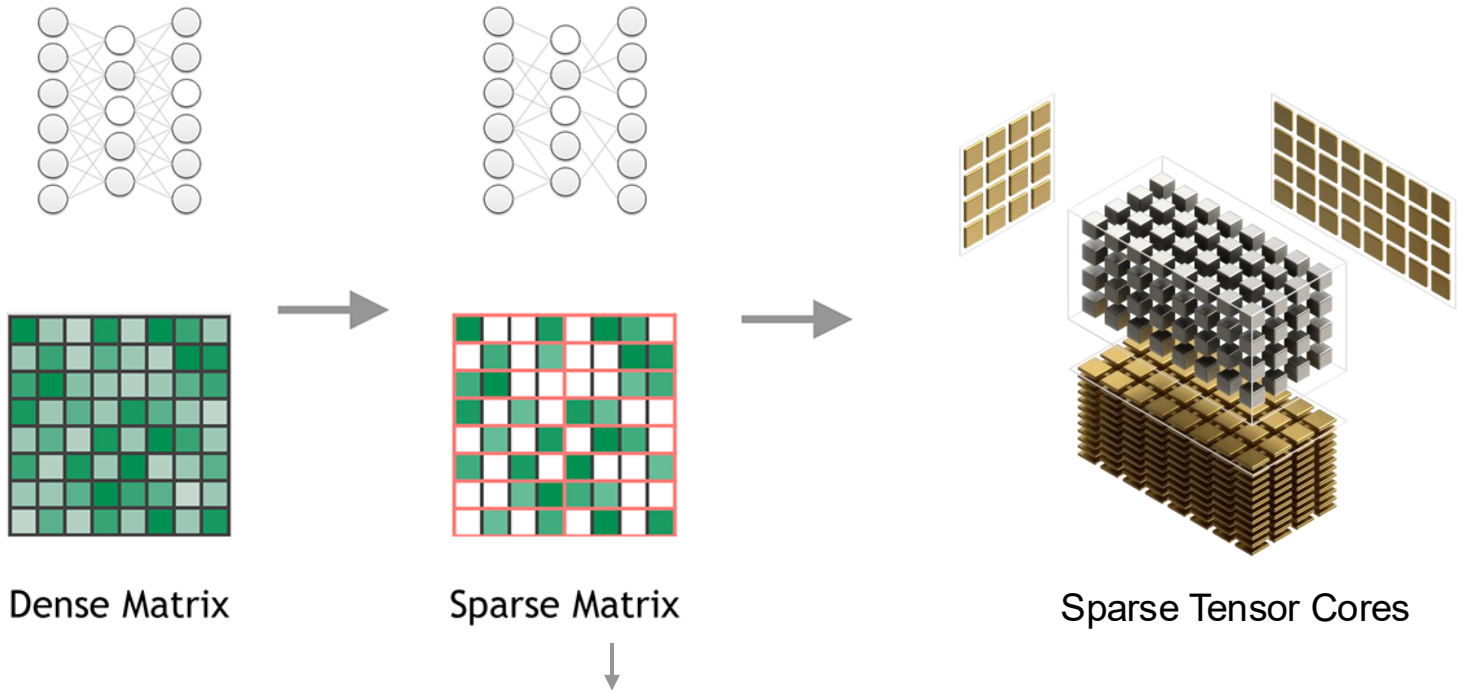
- **SLiM**: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression [ICML'25]

A Compiler for Rapid Prototyping of Sparsity and Quantization of Models n on GPUs!

- **STOICC**: A Compiler for Tile-based Sparsity

Weight Sparsity | Sparse Tensor Cores

We focus on 2:4 sparsity because NVIDIA sparse tensor cores!



2:4 Sparsity → 2 out of 4 consecutive elements are zero

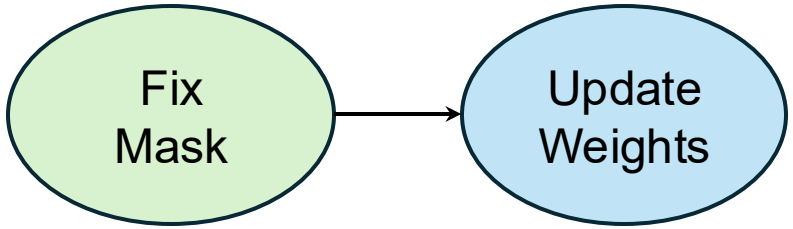
Static vs. Dynamic Sparsity in Pretraining

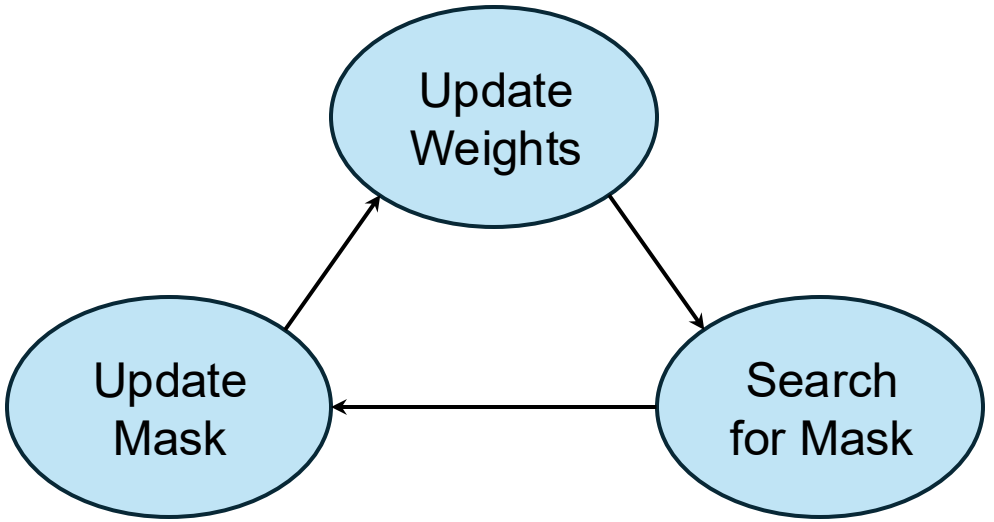
Static Sparsity[1]

- ☑ Memory Reduction
- ☑ No compute overhead

Dynamic Sparsity[2]

- ☒ Requires storing dense weights, gradients, and optimizer states
- ☒ Mask search and update overhead

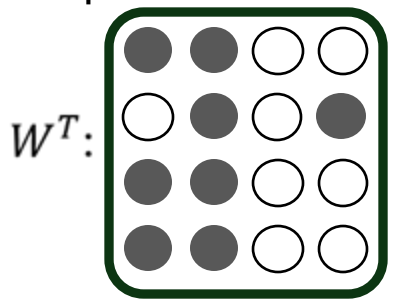




[1] Thangasara et al., SPDF: Sparse Pre-training and Dense Fine-tuning for Large Language Models (UAI23)
[2] Hu et al., Accelerating Transformer Pre-training with 2:4 Sparsity (ICML24)

Sparse Training | Forward Pass

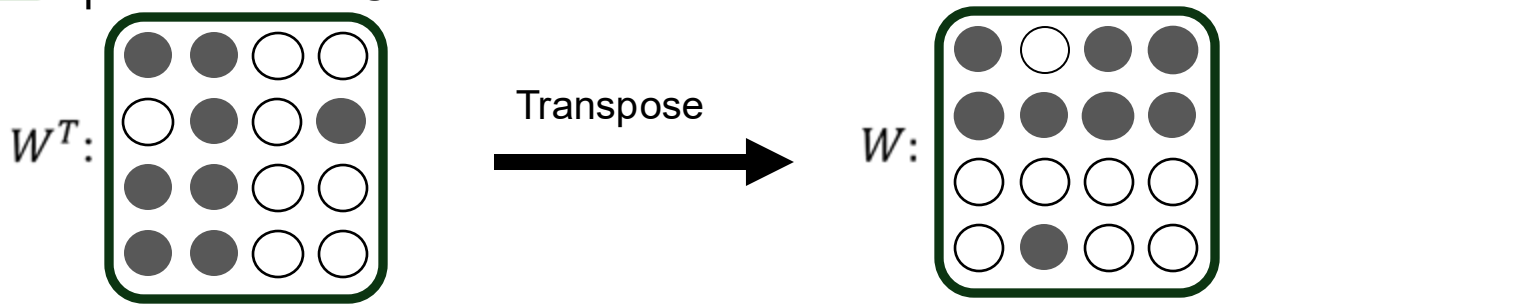
☑ Sparse Tensor Cores



Forward Pass: $Y = XW^T$

Sparse Training | Backward Pass

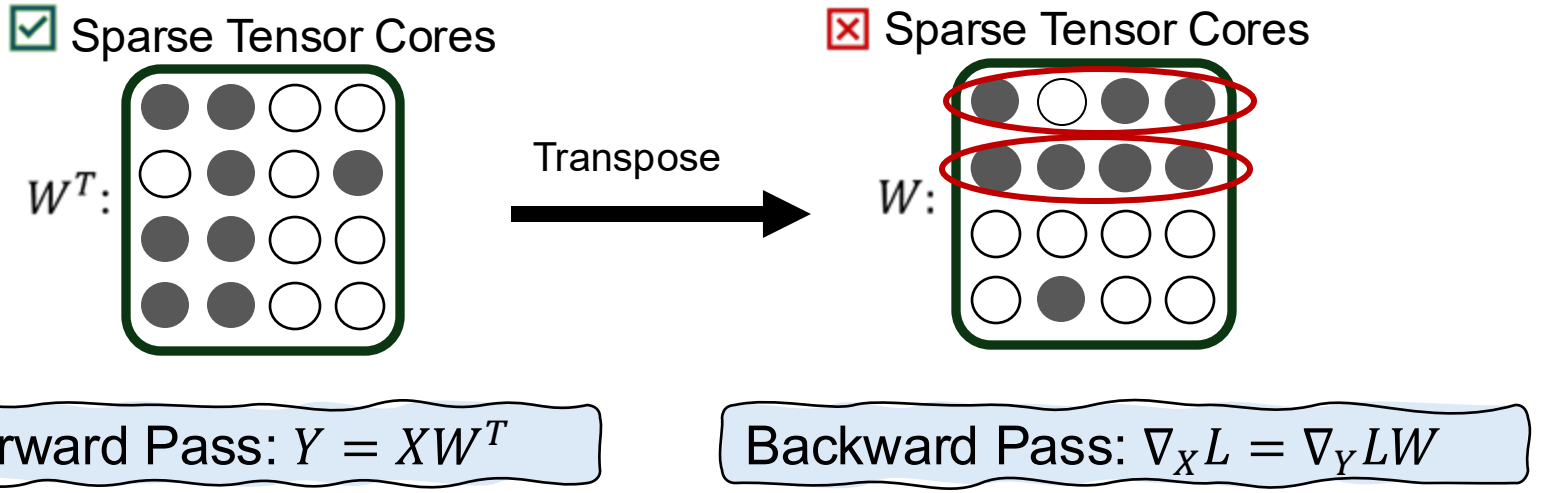
☑ Sparse Tensor Cores



Forward Pass: $Y = XW^T$

Backward Pass: $\nabla_X L = \nabla_Y LW$

Sparse Training | Backward Pass | Challenges



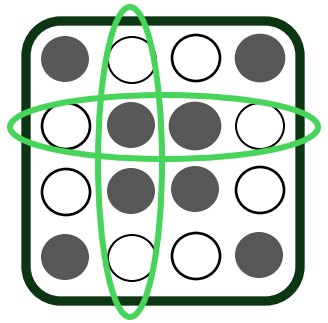
Sparse Training | Backward Pass | Prior Solutions

Prior work[1,2] uses transposable masks → Both W and W^T are 2:4 sparse

- Challenge: The sparsity is limited to very few 2:4 patterns → Accuracy Loss

In SLoPe we allow the forward pass to have an arbitrary 2:4 pattern

- Double Pruned Backward Pass will solve the issue

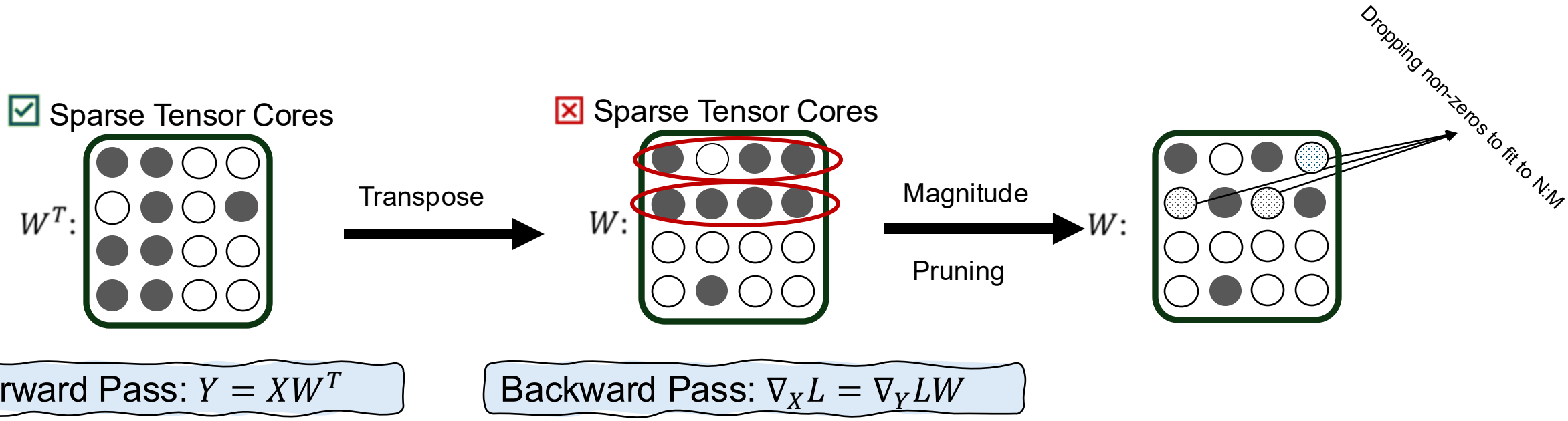


Both rows and columns have 2:4 sparsity

[1] Hubara et al., Accelerated Sparse Neural Training: A Provable and Efficient Method to Find N:M Transposable Masks (NeurIPS21)
[2] Zhang et al., Bi-directional Masks for Efficient N:M Sparse Training(ICML23)

Sparse Training | Double Pruned Backward Pass

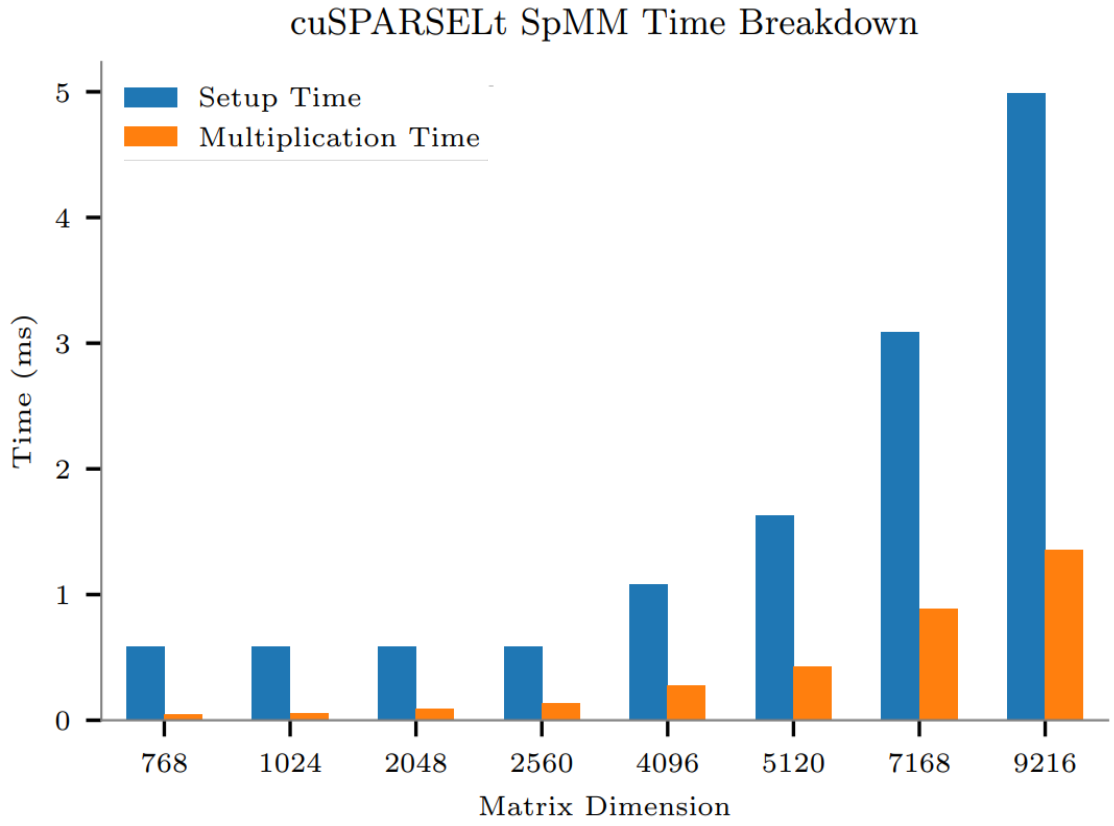
Prune the transposed weight again in the backward pass



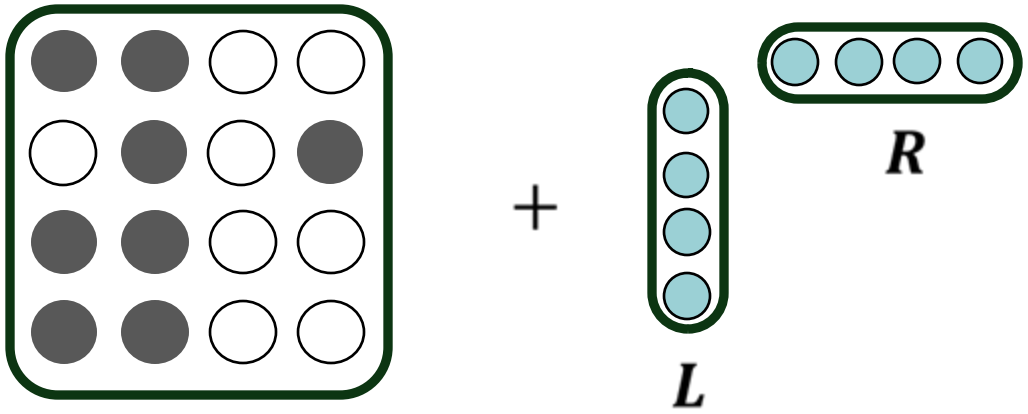
SLoPe | SpMM Setup Overhead

Pruning and preparing the weights in every iteration is expensive!

- SLoPe prunes and prepares W in the backward pass every 100 iterations



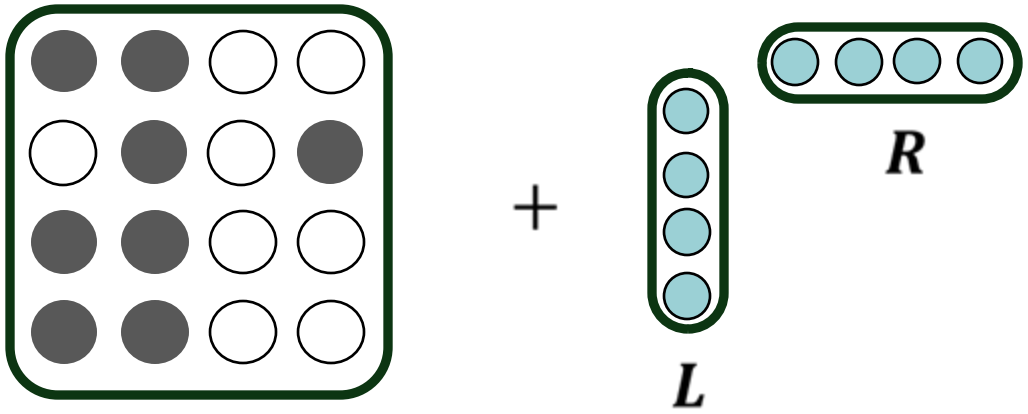
SLoPe | Lazy Low-rank Adapters (LoRA)



$$W = W_{sparse} + LR$$

SLoPe | Lazy Low-rank Adapters (LoRA)

d: Hidden Dimension
r: Adapter Rank
b: Batch Size

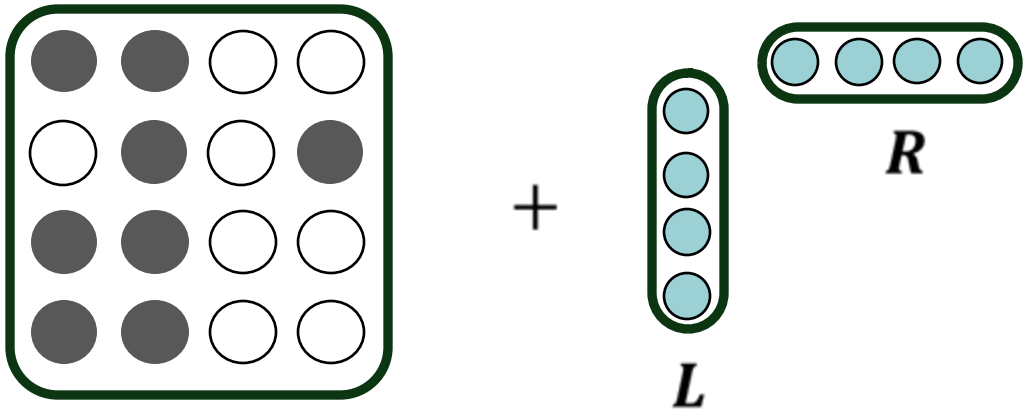


$$W = W_{sparse} + LR$$

Complexity	Sparse + Low-rank	Dense
Memory	$\frac{d^2}{2} + 2rd$	d^2
Compute	$\frac{bd^2}{2} + 2brd$	bd^2

SLoPe | Lazy Low-rank Adapters (LoRA)

d: Hidden Dimension
r: Adapter Rank
b: Batch Size



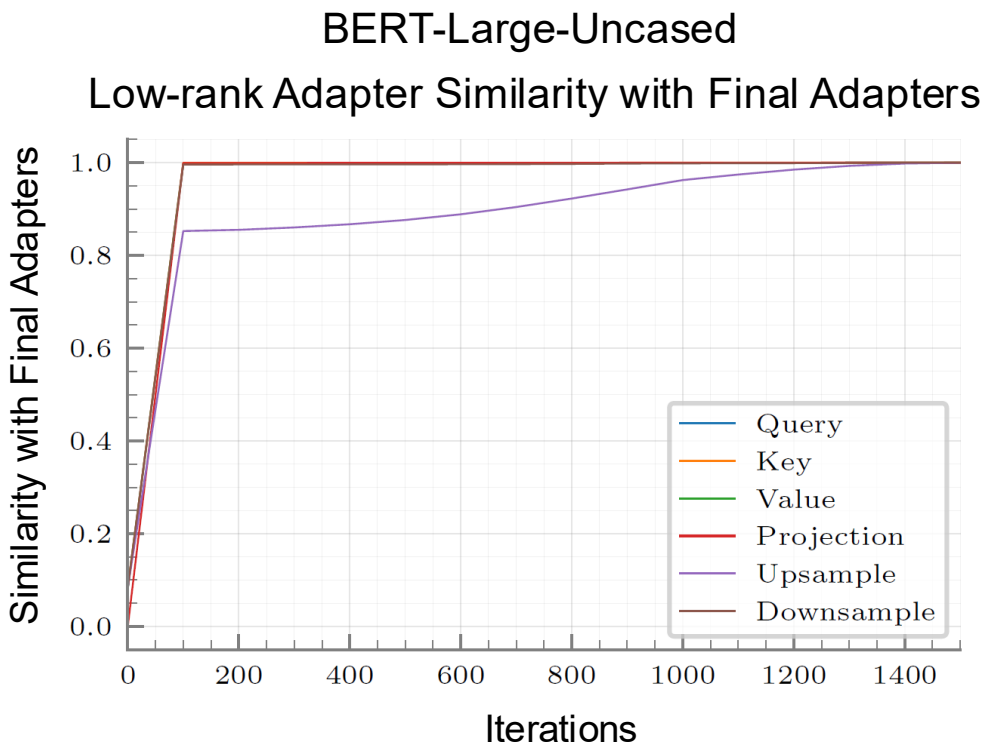
$$W = W_{sparse} + LR$$

Complexity	Sparse + Low-rank	Dense
Memory	$\frac{d^2}{2} + 2rd$	d^2
Compute	$\frac{bd^2}{2} + 2brd$	bd^2

Since $r \ll d$, the memory and compute overhead is negligible.

SLoPe | Lazy Low-rank Adapters | Convergence Rate

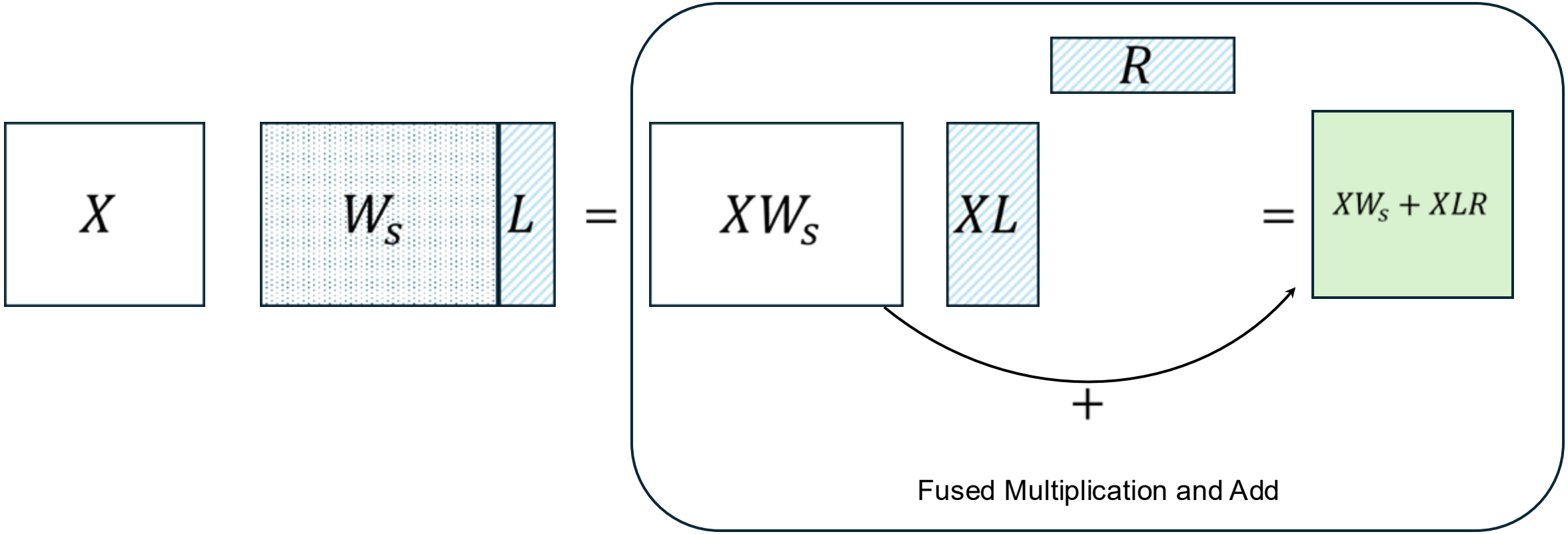
LoRA has a fast convergence rate!



SLoPe adds LoRA in the last 1% training iterations!
(Lazy Low-rank Adapters )

SLoPe | Combined SpMM and Low-rank Adapters

- Low arithmetic intensity in low-rank adapters
  - **Solution:** Combines SpMM and Low-rank adapters



SLoPe | Results | GPT2 Zero-shot Accuracy

Comparison: We compare SLoPe against state of the art 2:4 sparse pretraining work (Ext. SR-STE[1,2])

SLoPe outperforms prior work in 6 out of 8 tasks!

GPT2 Small

Dataset	Dense	Ext. SR-STE	SLoPe
MMLU	22.9	**24.2**	23.0
Arc Challenge	20.7	18.4	**19.4**
OpenBookQA	16.2	14.2	**16.4**
WinoGrande	50.6	47.5	**50.8**
HellaSwag	28.5	26.9	**27.5**
MathQA	21.8	**21.4**	20.8
PIQA	59.8	55.2	**57.6**
RACE	28.4	24.2	**27.2**

[1] Zhou et al., Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch (ICLR21)
[2] Hu et al., Accelerating Transformer Pre-training with 2:4 Sparsity (ICML24)

SLoPe | Results | Speedup and Memory Reduction

Speedup (vs. Dense)

Training: 1.25×
Inference: 1.54×

Memory Reduction (vs. Dense)

Training: 0.63×
Inference: 0.51×

Outline

Model Compression in Pretraining

- **SLoPe**: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs Lazy Low-rank Adapters [ICLR'25]

Model Compression at Inference

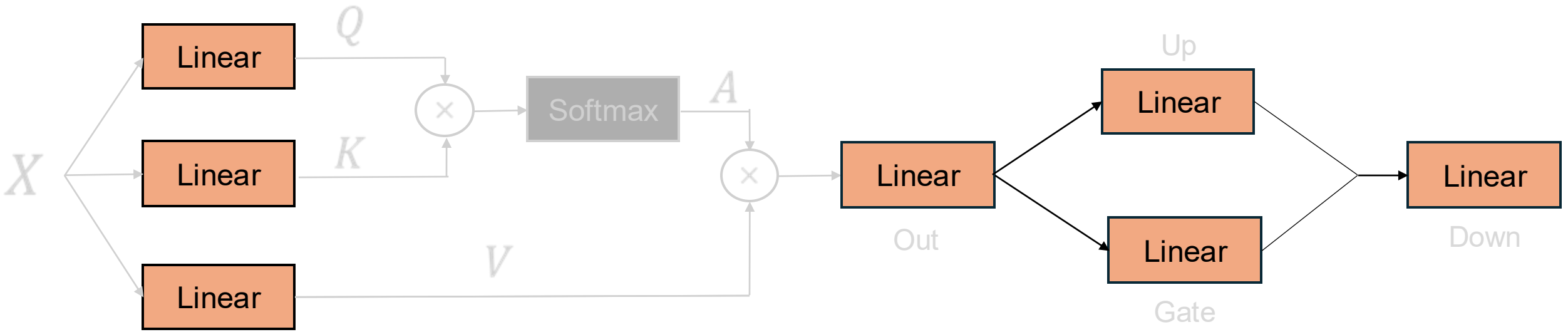
- **SLiM**: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression [ICML'25]

Compression Trinity Webpage

A Compiler for Rapid Prototyping Sparsity and Quantization of Models on GPUs!

- **STOICC**: A Compiler for Tile-based Sparsity

LLM Compute Graph | Sparse Inference

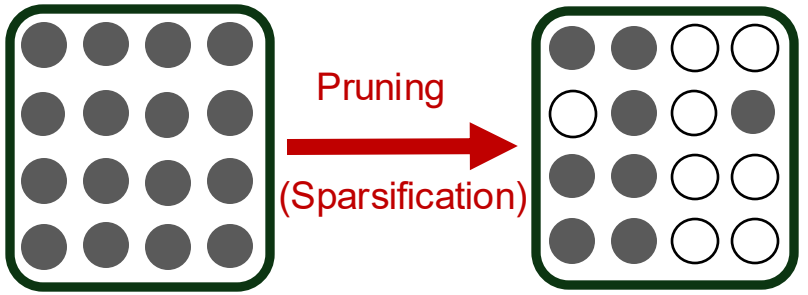


For large models, it might not be feasible to train models for sparsity, SLiM focuses on introducing sparsity at post training!

Post-training Compression Methods

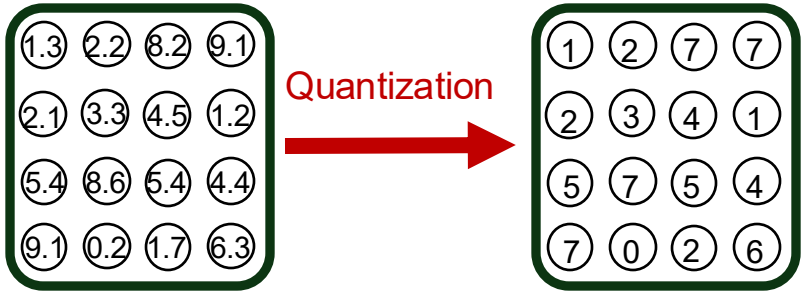
Sparsity

Set non-important weights to zero



Quantization

Reduce the precision of numbers



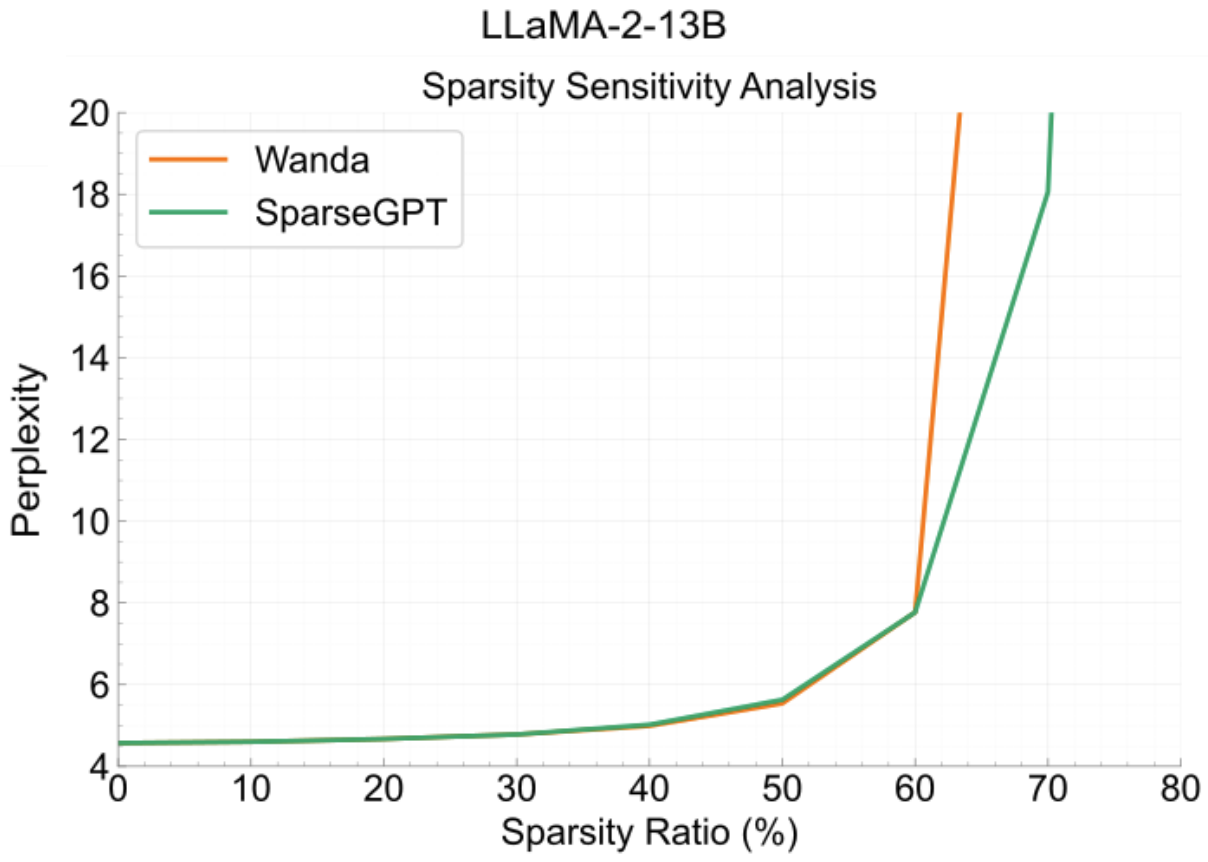
3-bit Quantization:

Round to the closest integer
Clip the data larger than 7

Sparsity Challenges

The perplexity of models becomes too big below 50% sparsity!

- Maximum 2 × reduction in model size

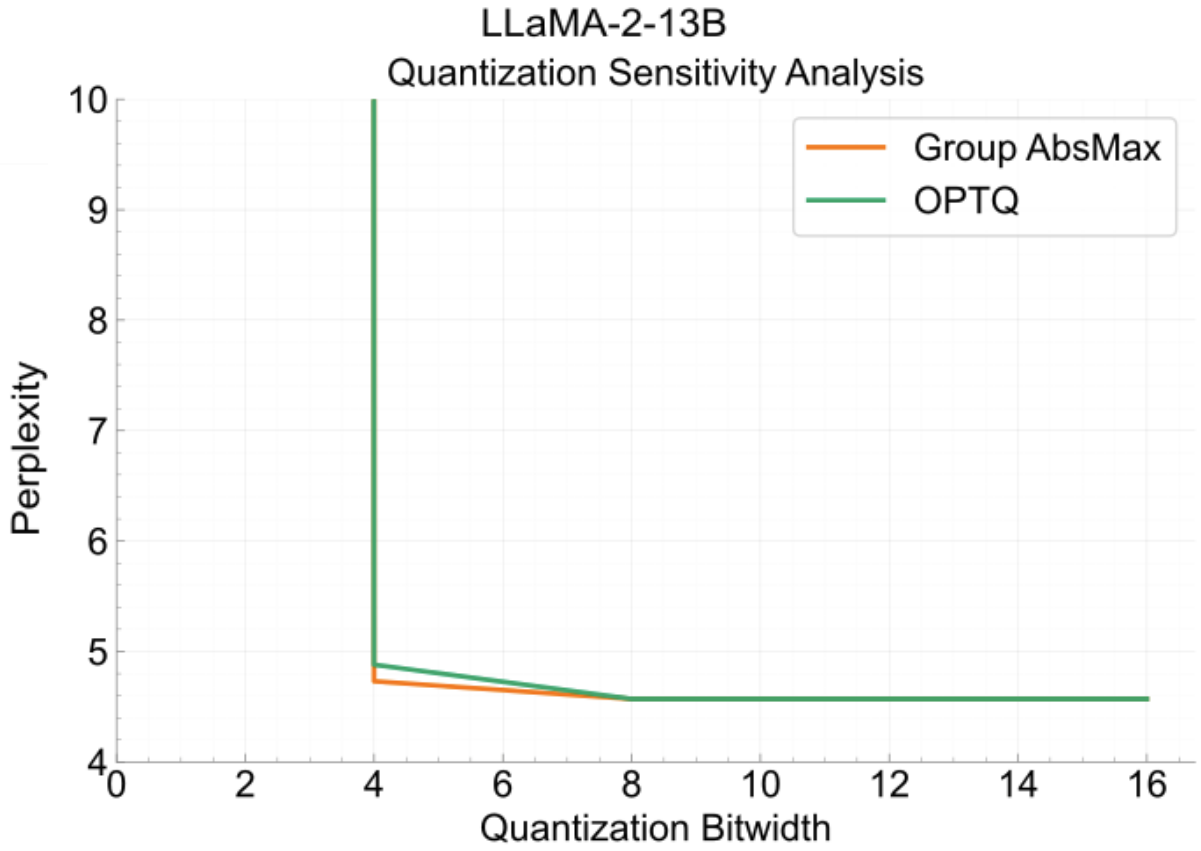


↓ indicates better performance.

Quantization Challenges

The perplexity of models becomes too big below 4-bit quantization!

- Maximum $4\times$ reduction in model size



A lower quantization bit width is desired

Higher Compression Ratios

8 × Compression ratio case study:

Average Accuracy on 6 LM Harness Tasks*

Method	LLaMA-2-7B	LLaMA-2-13B
Dense	56.6%	60.8%
87.5% Sparse**	31.06%	31.59%
2-bit Quantization***	31.81%	31.68%
4-bit Quantization + 50% Unstructured Sparsity	53.62%	57.00%
4-bit Quantization + 2:4 Sparsity	45.49%	51.05%

Combining sparsity and quantization gives better accuracy vs quantization or sparsity alone!

*The tasks include MMLU, PIQA, ARC-Easy, ARC-Challenge, WINOGRANDE, and OpenBookQA
**Best method among Wanda and SparseGPT
***Best method among AbsMax and OPTQ

Higher Compression Ratios

$8\times$ Compression ratio case study:

Average Accuracy on 6 LM Harness Tasks*

Method	LLaMA-2-7B	LLaMA-2-13B
Dense	56.6%	60.8%
87.5% Sparse**	31.06%	31.59%
2-bit Quantization***	31.81%	31.68%
4-bit Quantization + 50% Unstructured Sparsity	53.62%	57.00%
4-bit Quantization + 2:4 Sparsity	45.49%	51.05%

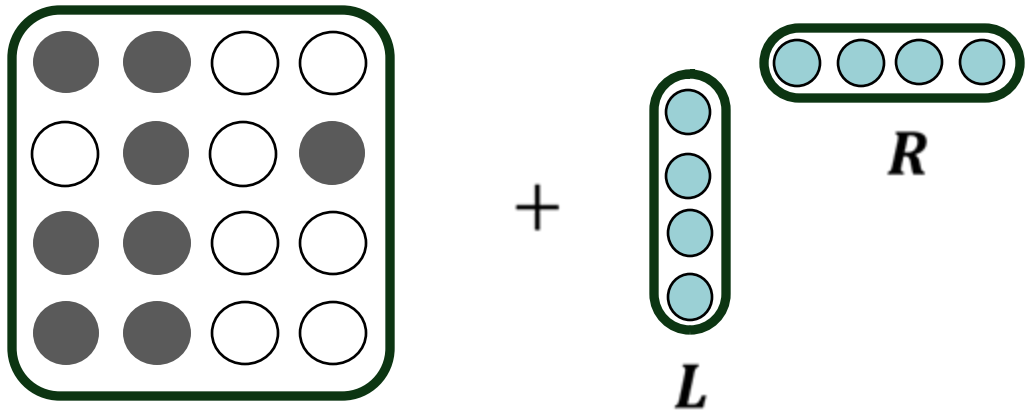
However, the accuracy gap between compressed and dense models is large!

*The tasks include MMLU, PIQA, ARC-Easy, ARC-Challenge, WINOGRANDE, and OpenBookQA
**Best method among Wanda and SparseGPT
***Best method among AbsMax and OPTQ

Accuracy Recovery with Low-rank Adapters

Low-rank adapters can help recover the accuracy of the models[1,2]

- Challenge: They require millions of tokens to train if acquired from training
- Solution: One-shot Low-rank Adapters compute LR mathematically with no training need

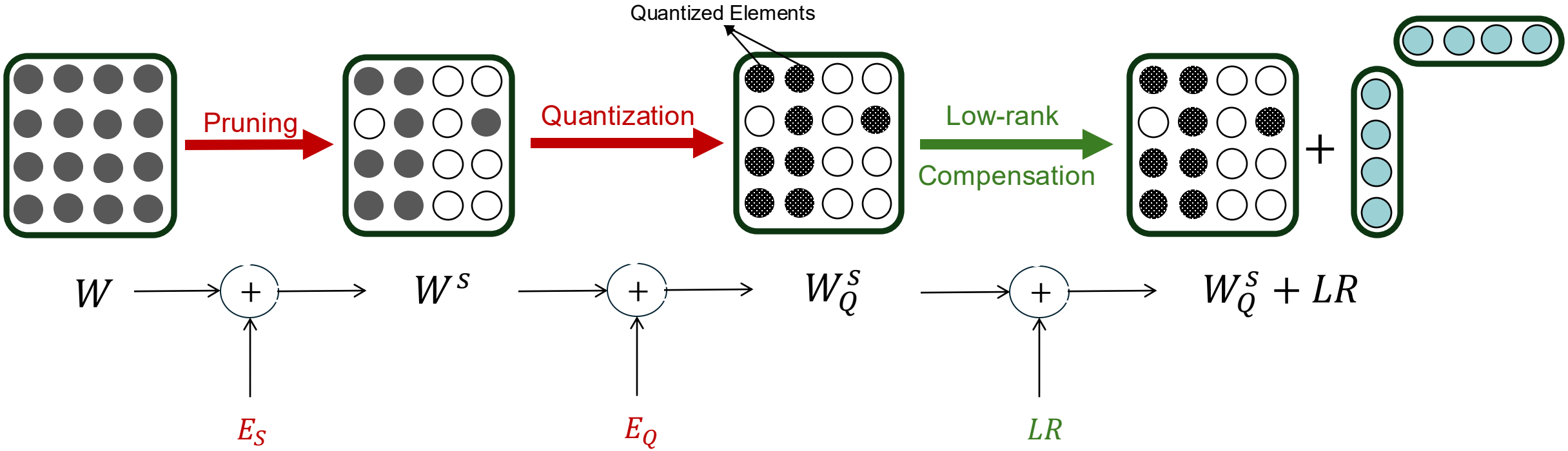


$$W = W^C + LR$$

[1] Dettmers et al., QLoRA: Efficient Finetuning of Quantized LLMs (NeurIPS23)
[2] Nikdan et al., RoSA: Accurate Parameter-Efficient Fine-Tuning via Robust Adaptation (ICML24)

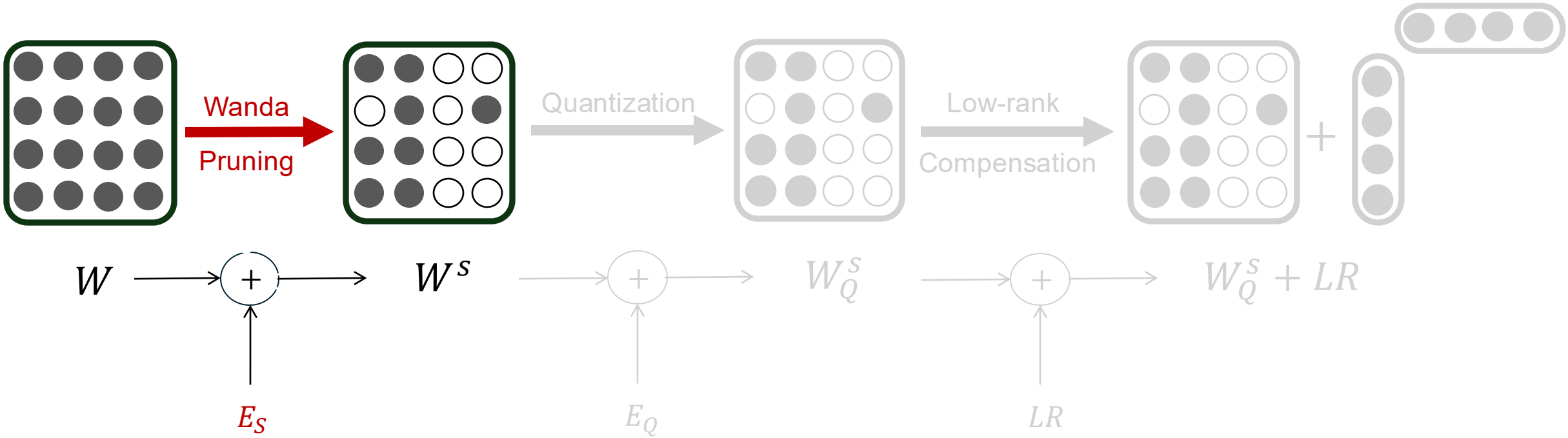
SLiM | Overview

E_S: Sparsity Error
E_Q: Quantization Error
L, R: Low-rank Adapters
W^S: Sparse Weight
W_Q^S: Sparse and Quantized Weight



SLiM | One-shot Pruning Method

E_S: Sparsity Error
E_Q: Quantization Error
L, R: Low-rank Adapters
W^S: Sparse Weight
W_Q^S: Sparse and Quantized Weight

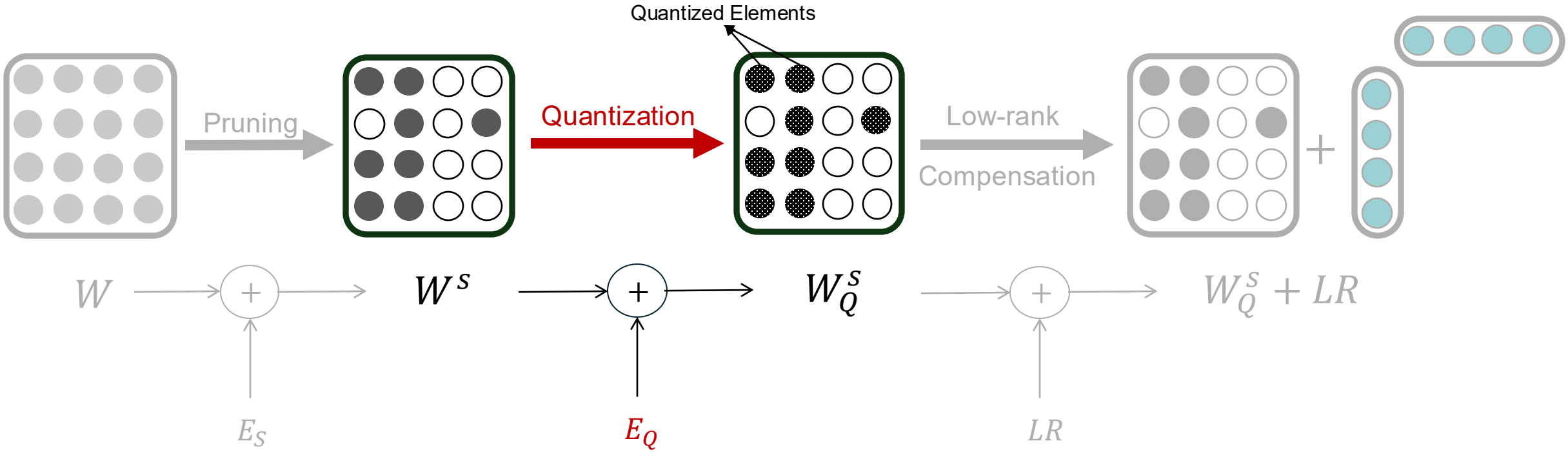


SLiM uses an off-the-shelf method (Wanda[1]) for one-shot pruning.

[1] Sun et al., A Simple and Effective Pruning Approach for Large Language Models

SLiM | Quantization

E_S: Sparsity Error
E_Q: Quantization Error
L, R: Low-rank Adapters
W^S: Sparse Weight
W_Q^S: Sparse and Quantized Weight



SLiM finds a tractable solution for minimizing the quantization error using novel a probabilistic approach.

Uniform Quantization

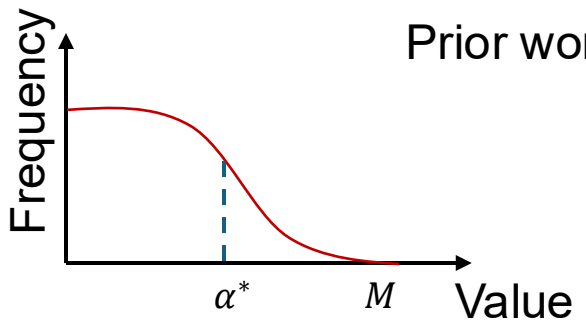
q: Quantization Bitwidth
Q: Quantization Function
$f(x)$: Weight PDF

Uniform quantization uses a single parameter per tensor to quantize the weight.

- The values larger than α^* get clipped: $W_Q = clip(\frac{W}{\alpha^*}, \pm 1) \times 2^{q-1}$
- Tuning Parameter α^* → Minimize the MSE of the quantization.

$$\alpha^* = \arg\min_{\alpha} \left| W - W_Q \right|^2$$

Non-convex np-hard problem!
Prior work[1] approximately solves it through exhaustive search.



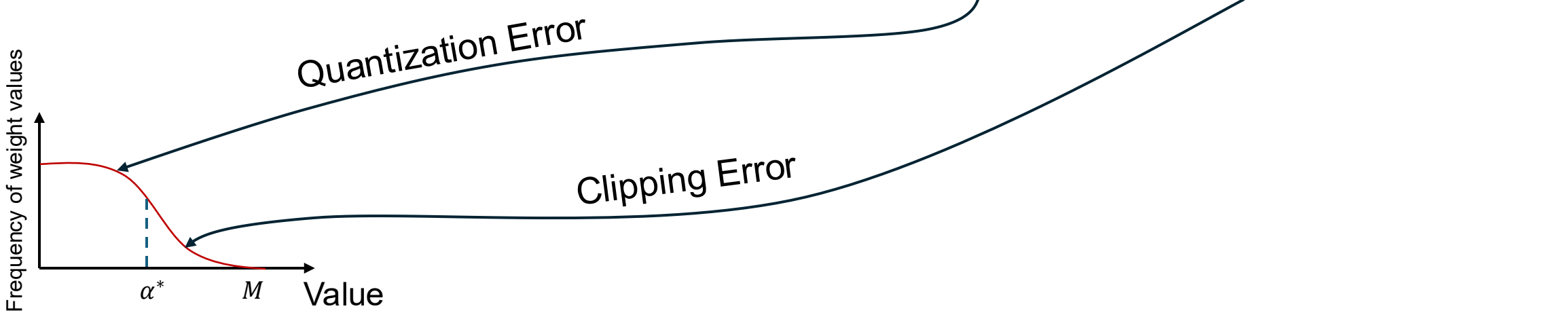
[1] Zhao et al., Atom: Low-bit Quantization for Efficient and Accurate LLM Serving

Uniform Quantization | SLiM-Quant

q: Quantization Bitwidth
Q: Quantization Function
$f(x)$: Weight PDF

SLiM-Quant uses a probabilistic approach to formulate the objective function in uniform quantization

$$\alpha^* = \arg\min_{\alpha} \left| W - W_Q \right|^2 \quad \Rightarrow \quad \alpha^* = \arg\min \int_0^{\alpha} Q(x) f(x) dx + \int_{\alpha}^{M} |x - \alpha|^2 f(x) dx$$

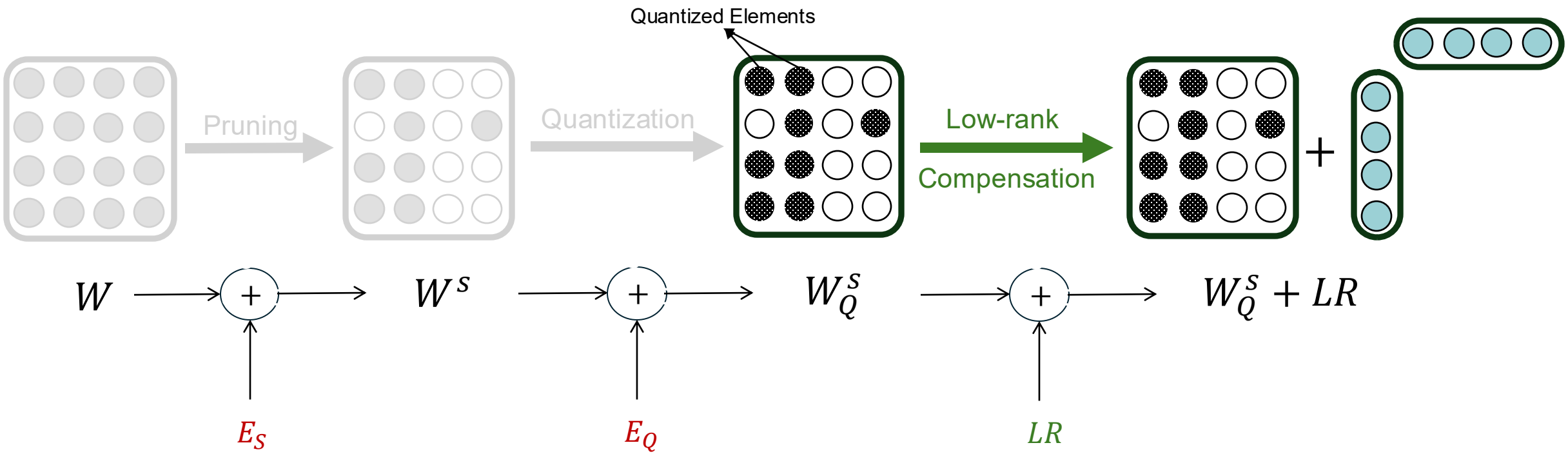


$f(x)$: Weight probability density function

Low-rank Adapters

E_S: Sparsity Error
E_Q: Quantization Error
L, R: Low-rank Adapters
W^S: Sparse Weight
W_Q^S: Sparse and Quantized Weight

Goal: Reduce the error added due to pruning and quantization.



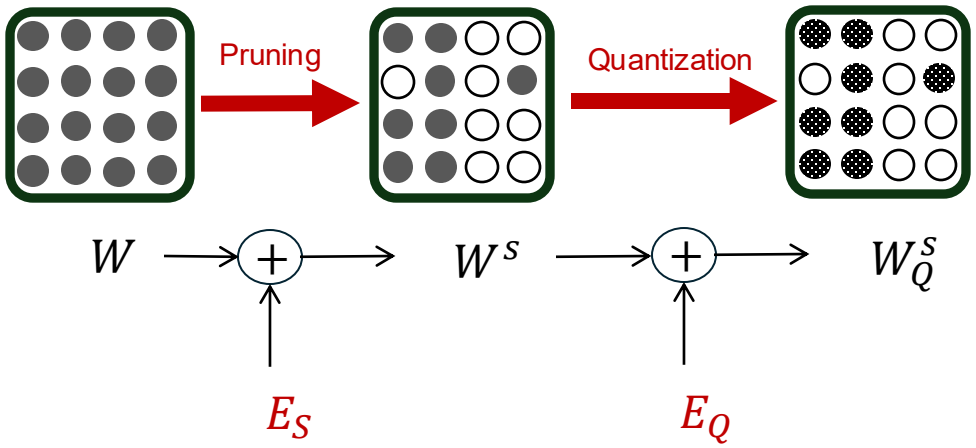
Low-rank Adapters | Naïve-LoRA

E_S: Pruning Error
E_Q: Quantization Error
F: Saliency Function

Error Norm Minimization

$$L^*, R^* = \arg\min \left| W - \left(W_Q^S + LR \right) \right|$$
$$L^*, R^* = \arg\min \left| E_S + E_Q - LR \right|$$

$$L^*, R^* = SVD\left(E_S + E_Q \right)$$



Error norm does not take the importance (saliency) of the weights into account.

Low-rank Adapters | SLiM-LoRA

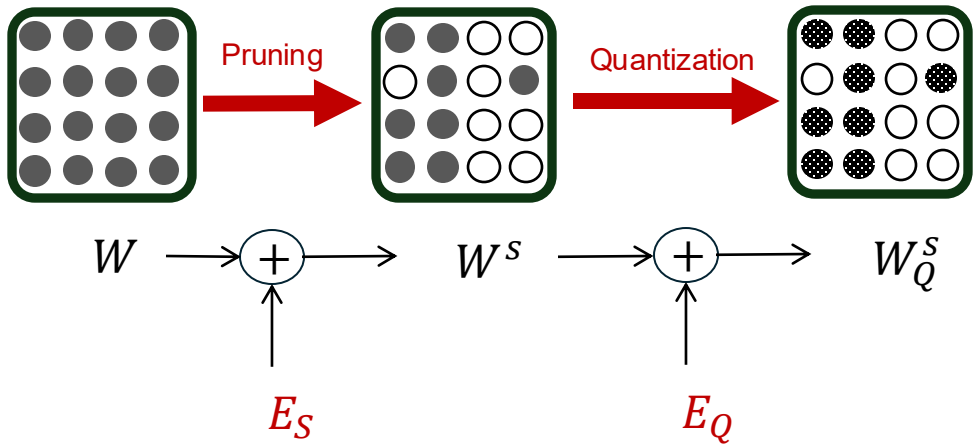
E_S: Pruning Error
E_Q: Quantization Error
F: Saliency Function
$\bar{x}$: Average Calibration Input

Error Saliency Minimization

$$L^*, R^* = \arg\min \left| F\left(W - \left(W_Q^S + LR\right)\right)\right|$$

$$L^*, R^* = \arg\min \left| F\left(E_S + E_Q - LR\right)\right|$$

Minimizing the saliency of the reconstruction error!



Low-rank Adapters | SLiM-LoRA

E_S: Pruning Error
E_Q: Quantization Error
F: Saliency Function
$\bar{x}$: Average Calibration Input

Error Saliency Minimization

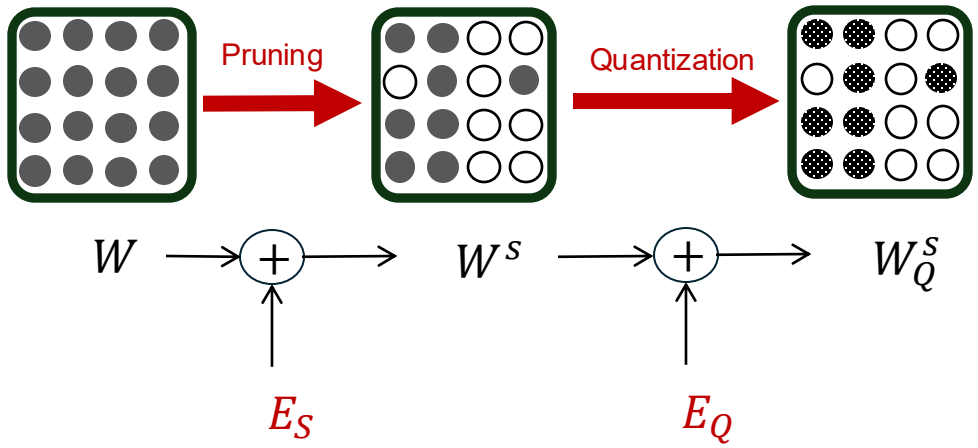
$$L^*, R^* = \arg\min \left| F\left(W - \left(W_Q^S + LR\right)\right)\right|$$

$$L^*, R^* = \arg\min \left| F\left(E_S + E_Q - LR\right)\right|$$

Minimizing the saliency of the reconstruction error!

Saliency Function : $F(W) = diag(\bar{x})W$

$$L^*, R^* = diag\left(\frac{1}{\bar{x}}\right)\left(SVD\left(diag(\bar{x})E_S + E_Q\right)\right)$$



The low-rank adapters in SLiM are further **quantized to 4-bits**!

SLiM | Zero-shot Accuracy Results

SLiM achieves up to 5.7% accuracy improvement over SOTA compression methods!

Average Accuracy over 6 Zero-shot tasks

2:4 Sparsity with 4-bit Weight Quantization

Method	OPT						LLaMA 2	
	125M	350M	1.3B	2.7B	6.7B	13B	7B	13B
SOTA*	33.70	33.38	38.75	40.15	44.32	45.64	45.49	51.05
Naïve-LoRA	34.28	33.38	38.36	41.21	44.91	45.25	48.45	51.94
SLiM-LoRA	**34.62**	**34.36**	**40.61**	**42.73**	**45.99**	**46.24**	**51.15**	**54.94**

Unstructured Sparsity with 4-bit Weight Quantization

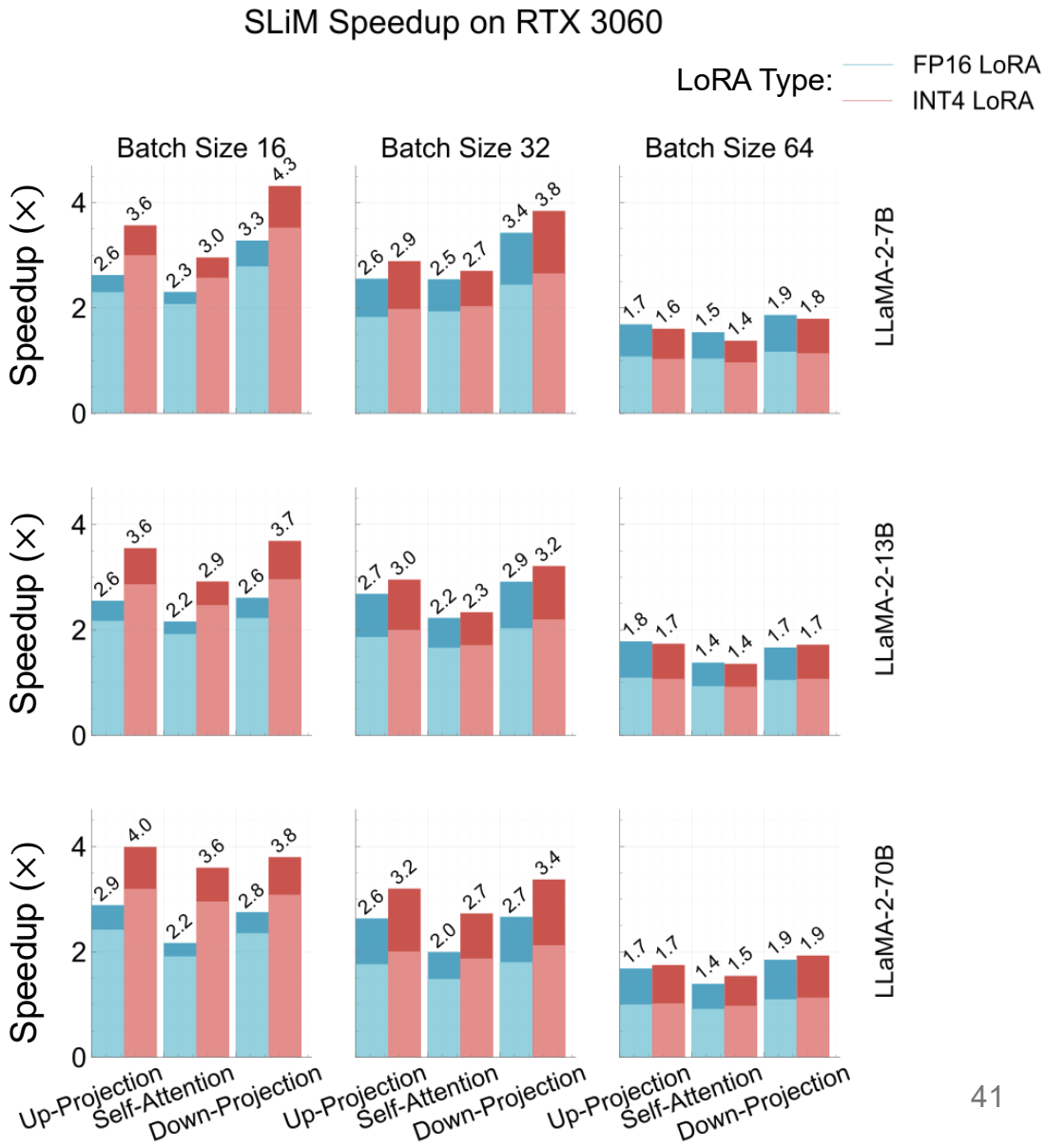
Method	OPT						LLaMA 2	
	125M	350M	1.3B	2.7B	6.7B	13B	7B	13B
SOTA*	35.11	35.16	41.02	43.43	46.97	47.38	53.62	57.00
Naïve-LoRA	34.77	34.23	40.40	43.37	46.64	47.30	51.52	55.33
SLiM-LoRA	**35.20**	**35.32**	**41.85**	**43.63**	**47.16**	**47.96**	**54.26**	**57.85**

*SOTA refers to the best accuracy among SparseGPT and Wanda for pruning and OPTQ, AWQ, AbsMax, OmniQuant, and AffineQuant for quantization.

SLiM | Speedup and Memory Reduction

SLiM achieves up to $4.3\times$ speedup on RTX3060 GPUs and up to $3.8\times$ speedup on A100 GPUs vs dense on LLaMA-2 models!

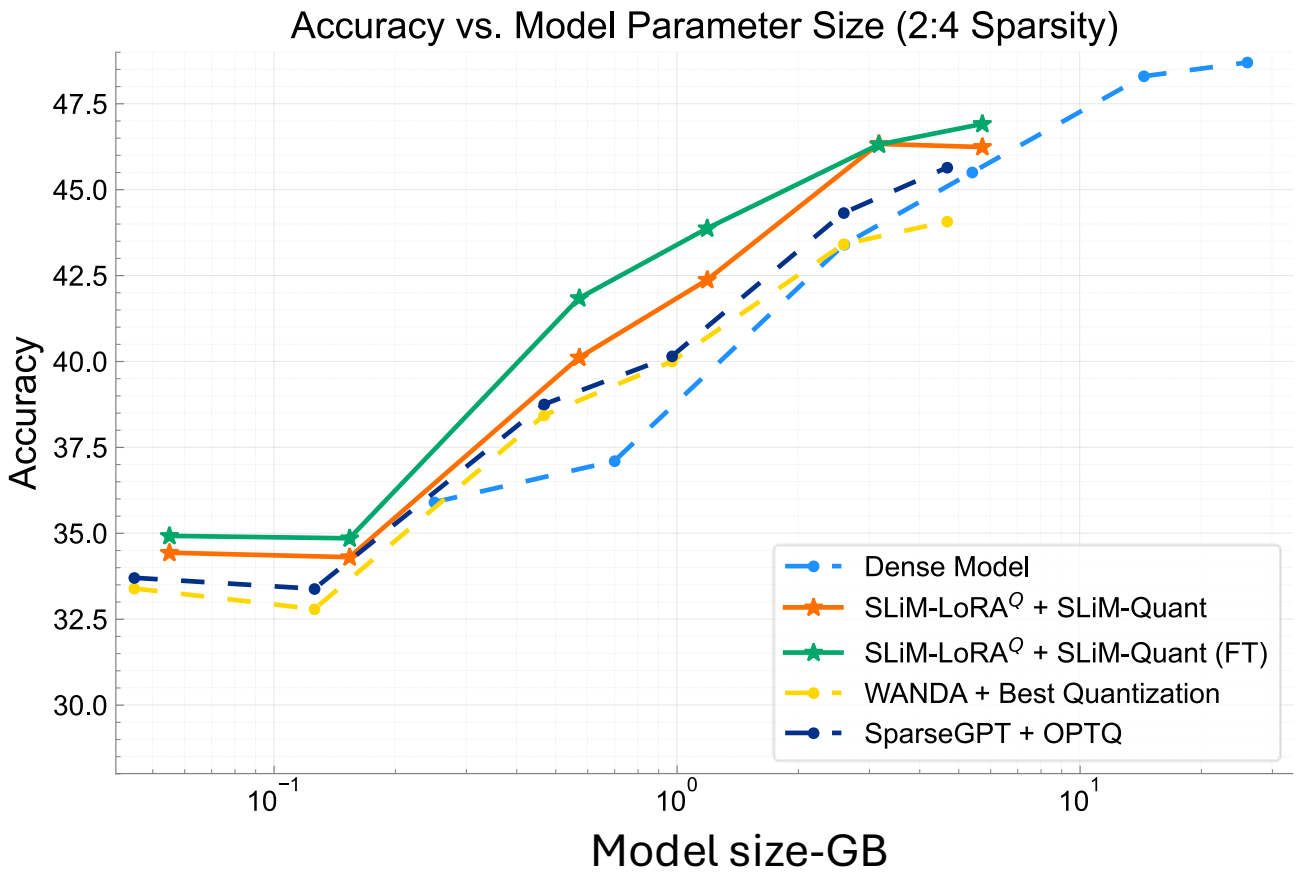
SLiM leads to consistent memory reduction on different models and GPUs!



SLiM | Larger Compressed vs. Smaller Dense

^{Q}: Quantized LoRA
FT: Fine-tuning

For a given parameter size budget, SLiM outperforms other methods, including dense models!



The accuracy results are from OPT family of models.

Outline

Model Compression in Pretraining

- **SLoPe**: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs Lazy Low-rank Adapters [ICLR'25]

Model Compression at Inference

- **SLiM**: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression [ICML'25]

Compression Trinity Webpage

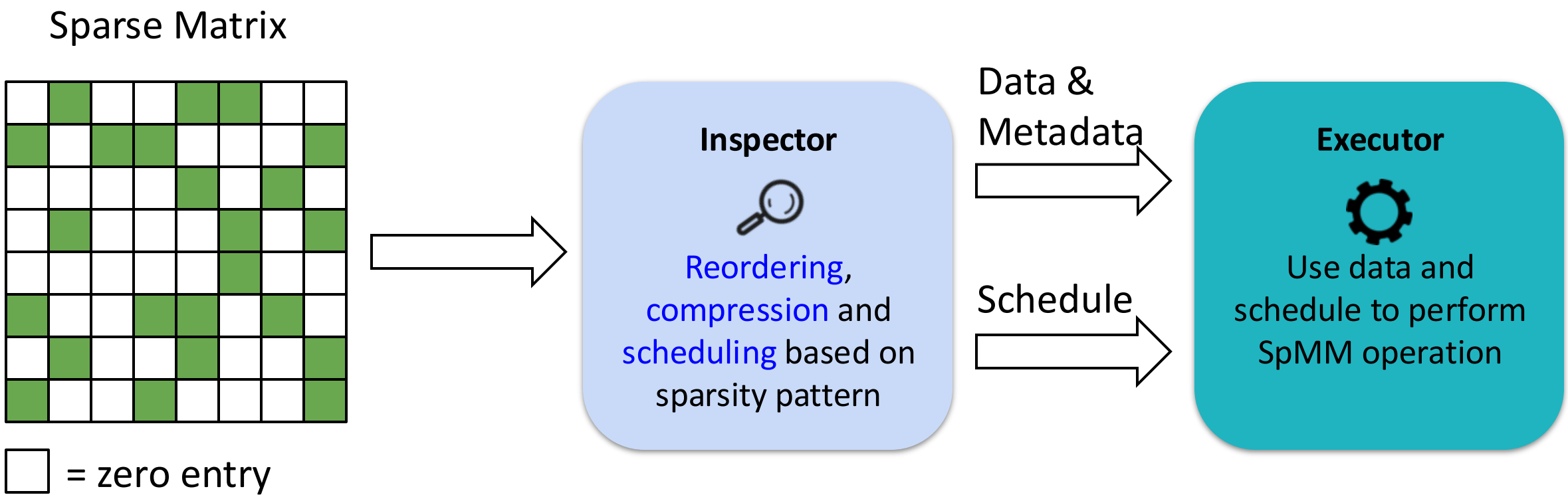
A Compiler for Rapid Prototyping of Sparsity and Quantization of Models on GPUs!

- **STOICC**: A Compiler for Tile-based Sparsity

STOICC

Inspector-Executor Tools for Sparsity

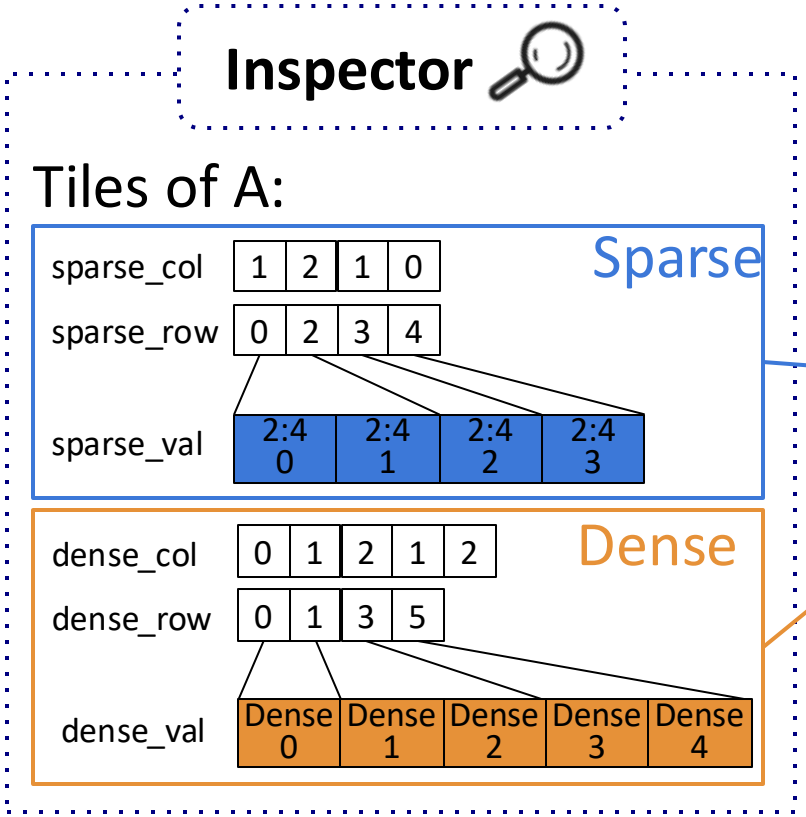
Compilers must optimize sparse matrix operations to achieve performance.



[PLDI'23] L. Wilkinson, K. Cheshmi, and M. M. Dehnavi, 'Register Tiling for Unstructured Sparsity in Neural Network Inference', *Proc. ACM Program. Lang.*, vol. 7, no. PLDI, Jun. 2023.

STOICC Executor: Built in Triton!

The executor performs the SpMM according to the schedule.



Launch grid with one Triton program per tile of output matrix D.

Each program iterates over a row of A and column of B according to the schedule.

Executor

```
for row, col in grid:
    acc_tile = 0
    for i in range(sparse_row[row], sparse_row[row + 1]):
        acc_tile += dot(sparse_val[i], B[sparse_col[i]][col])
    for i in range(dense_row[row], dense_row[row + 1]):
        acc_tile += dot(dense_val[i], B[dense_col[i]][col])
    D[row, col] = acc_tile
```

Summary

Model Compression

- **SLoPe**: Double-Pruned Sparse Plus Lazy Low-Rank Adapter Pretraining of LLMs Lazy Low-rank Adapters [ICLR'25]
  - **SLiM**: One-shot Quantization and Sparsity with Low-rank Approximation for LLM Weight Compression [ICML'25]

A Compiler for Rapid Prototyping of Sparsity and Quantization of Models on GPUs!

- **STOICC**: A Compiler for Tile-based Sparsity

Compression Trinity Webpage

STOICC

https://paramathic.github.io/stoicc-docs/
https://www.cs.toronto.edu/~mmozaffari/compression-trinity/index.html