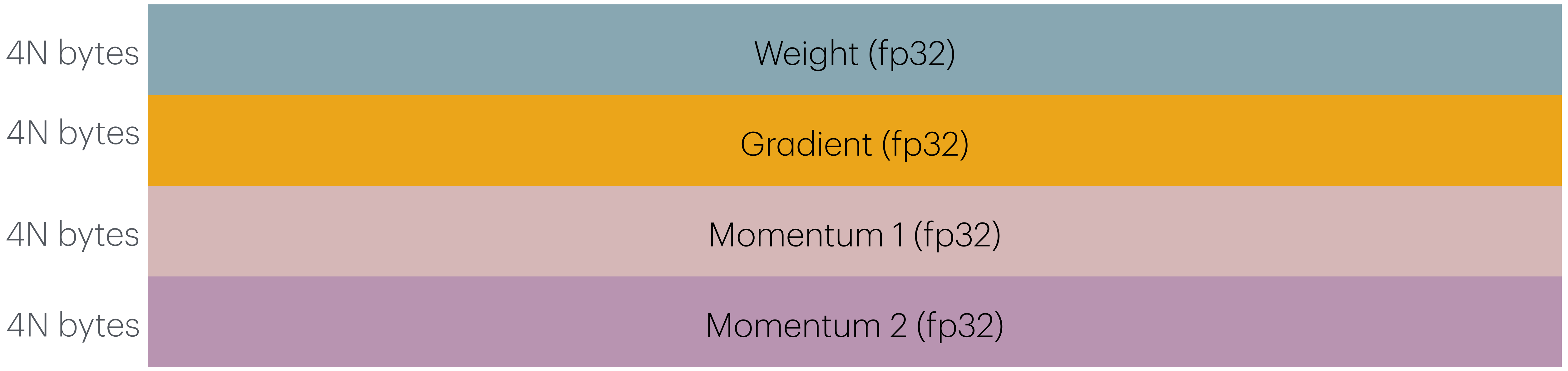
Low-rank projections

Philipp Krähenbühl, UT Austin

Training large models

Memory requirements

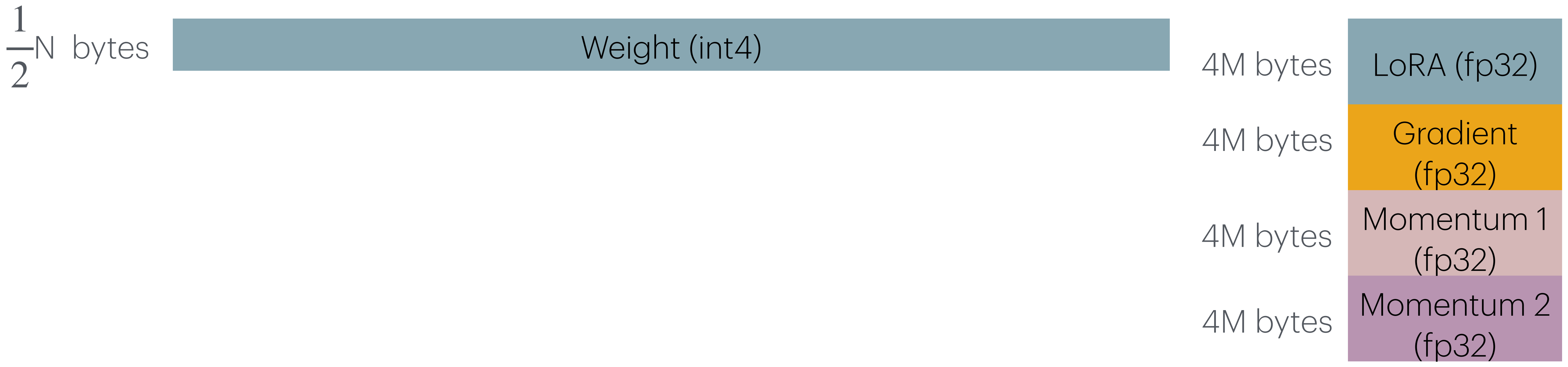
- Without optimization:
  - Model parameters: N
  - Weights: N floats
  - Gradients: N floats
  - Momentum: N floats
  - 2nd momentum (ADAM): N floats
- 16N bytes without counting activations



Training QLoRA models

Memory requirements

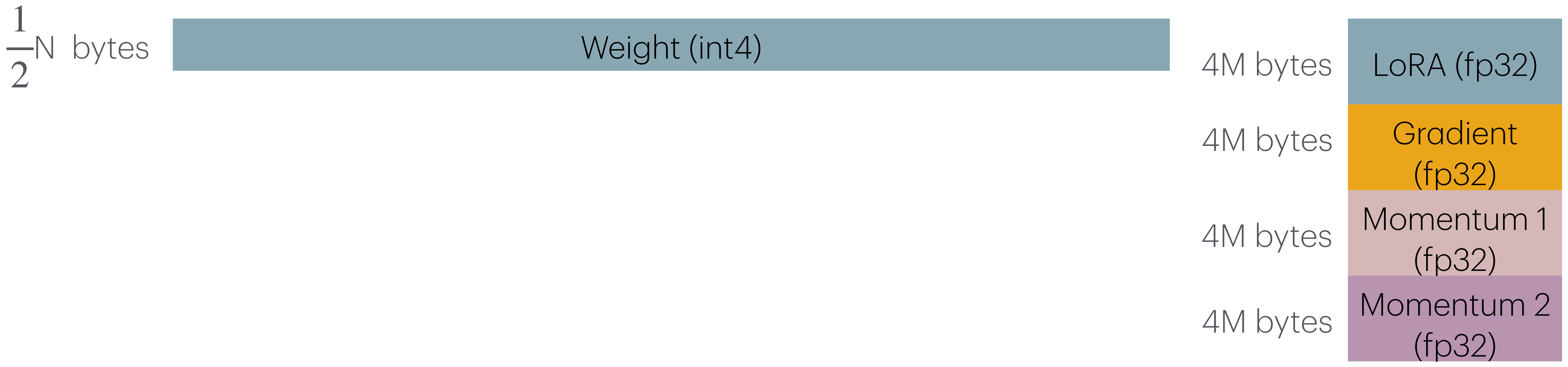
- QLoRA
  - Model parameters: N, LoRA param M
  - Weights: N int4, M floats
  - Gradients: M floats
  - Momentum: M floats
  - 2nd momentum (ADAM): M floats
- $\frac{1}{2}$N+16M bytes without activations
- M often ~1-5% of N



QLoRA

Tradeoffs

- Advantage
  - Extremely low optimizer memory
- Disadvantages
  - **Fine-tuning only (no pre-training)**
  - **Task dependent**
    - May require large rank R

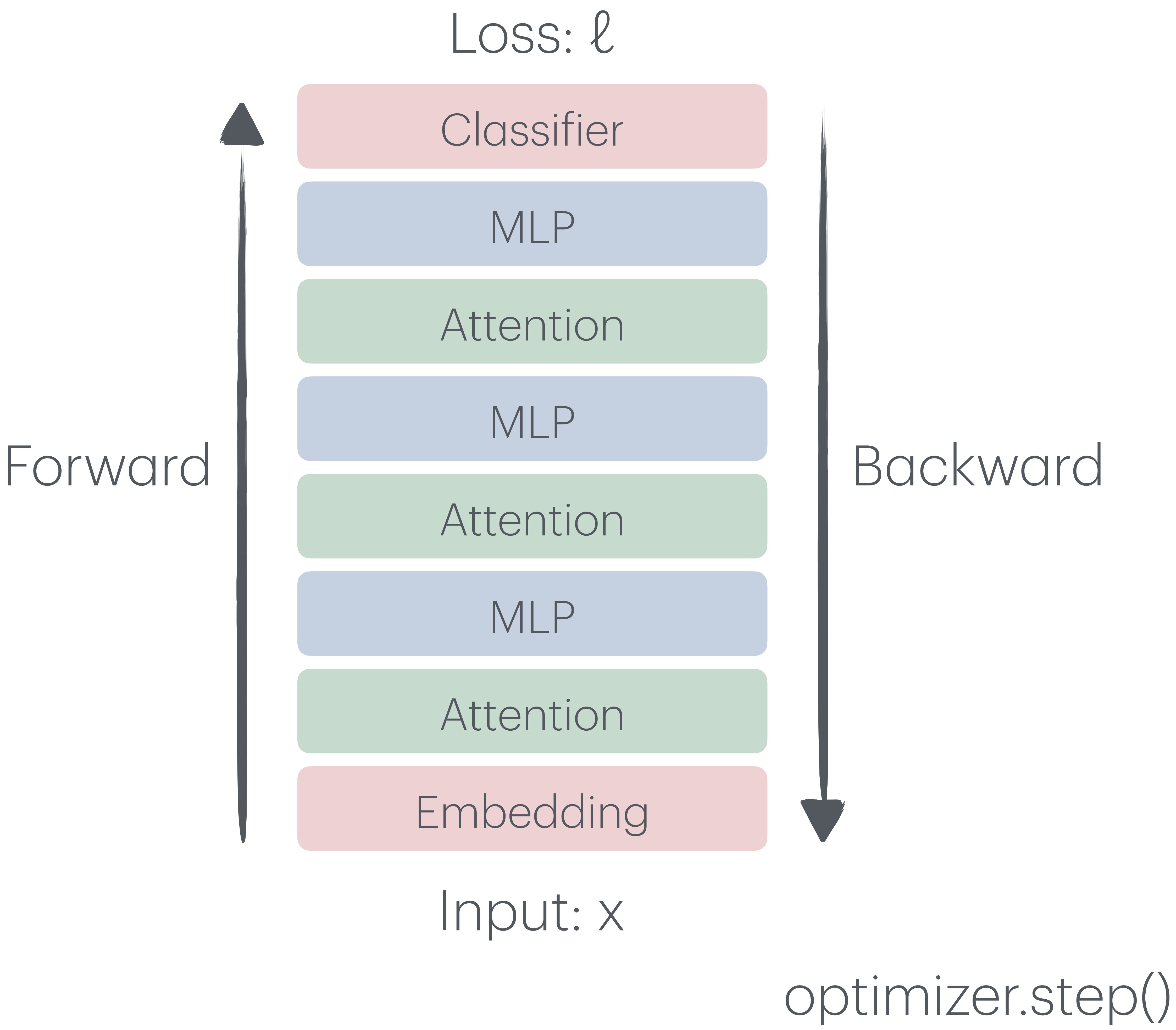


[1] Tim Dettmers, et al. QLoRA: Efficient Finetuning of Quantized LLMs. 2023

Backpropagation

A closer look

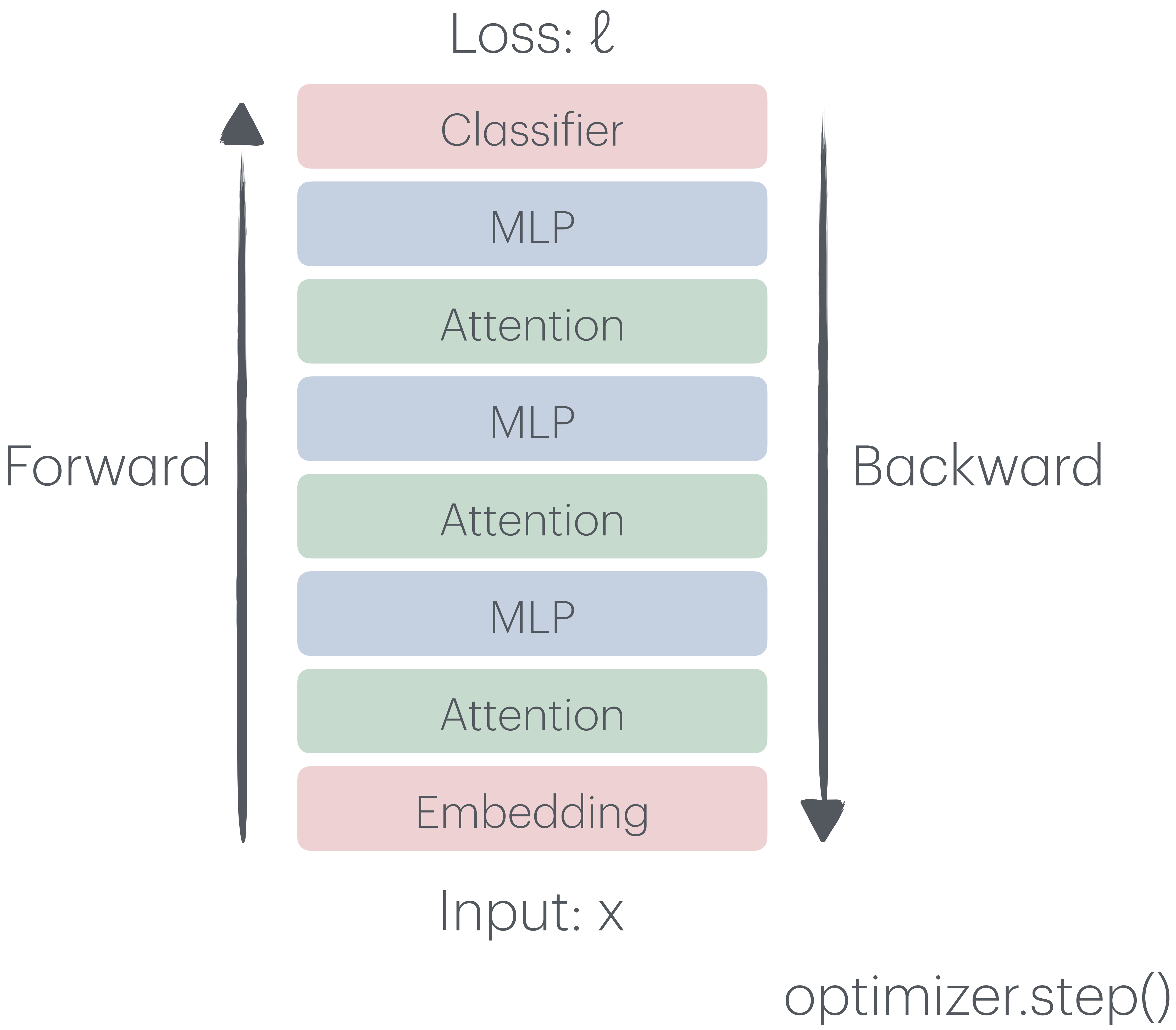
- Linear layer $y = W^\top x$
  - Gradient: $\frac{\partial}{\partial W} y = x y^\top$
  - Backprop: $\frac{\partial}{\partial x} y = W^\top$
- Nonlinear layer $y = f(x)$
  - Backprop: $\frac{\partial}{\partial x} y = \nabla f(x)$



Backpropagation

A closer look

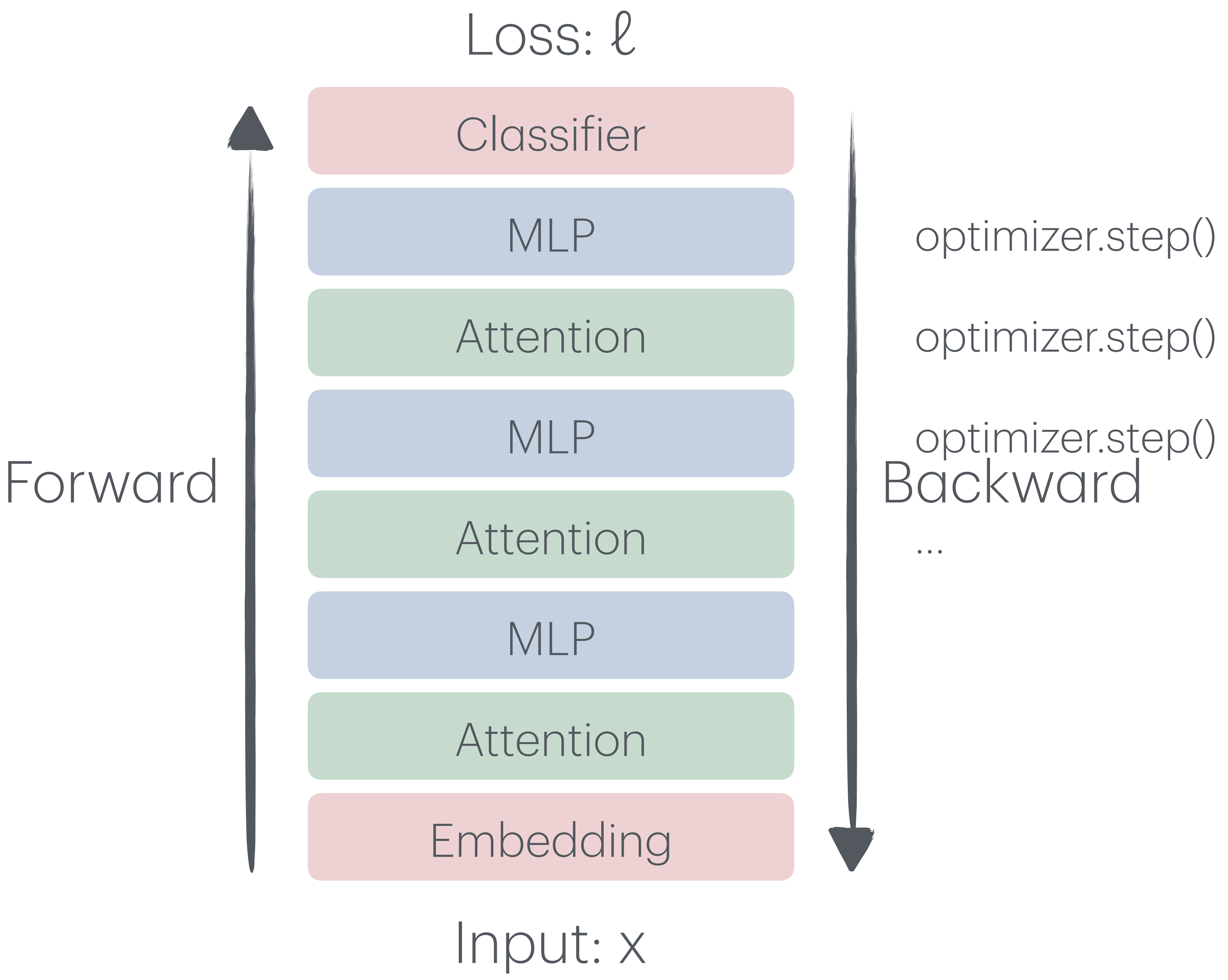
- Forward
  - Store activation
    - Backprop of non-linear layers
    - Weight gradient of linear layers
- Backward
  - Compute gradient (allocate memory)
  - Discard activation (free memory)



Backpropagation

Memory efficient backprop

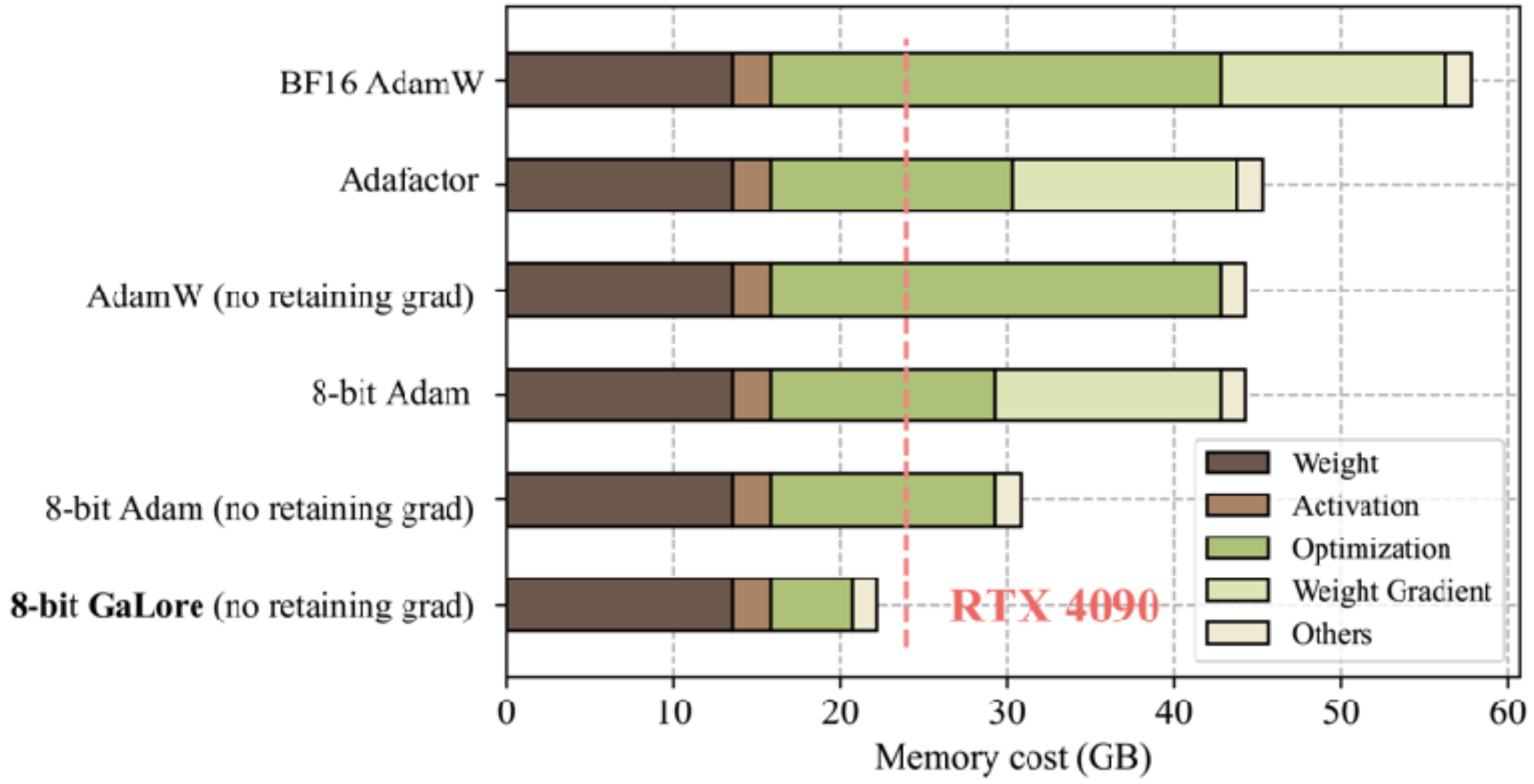
- Backward
  - Compute gradient (allocate memory)
  - Discard activation (free memory)
  - Optimizer.step
  - Discard gradient (free memory)



[1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024

Backpropagation

Memory efficient backprop



- Advantage
  - No memory cost to store gradient
- Restrictions
  - Single GPU
  - No gradient accumulation
  - Fairly hacky

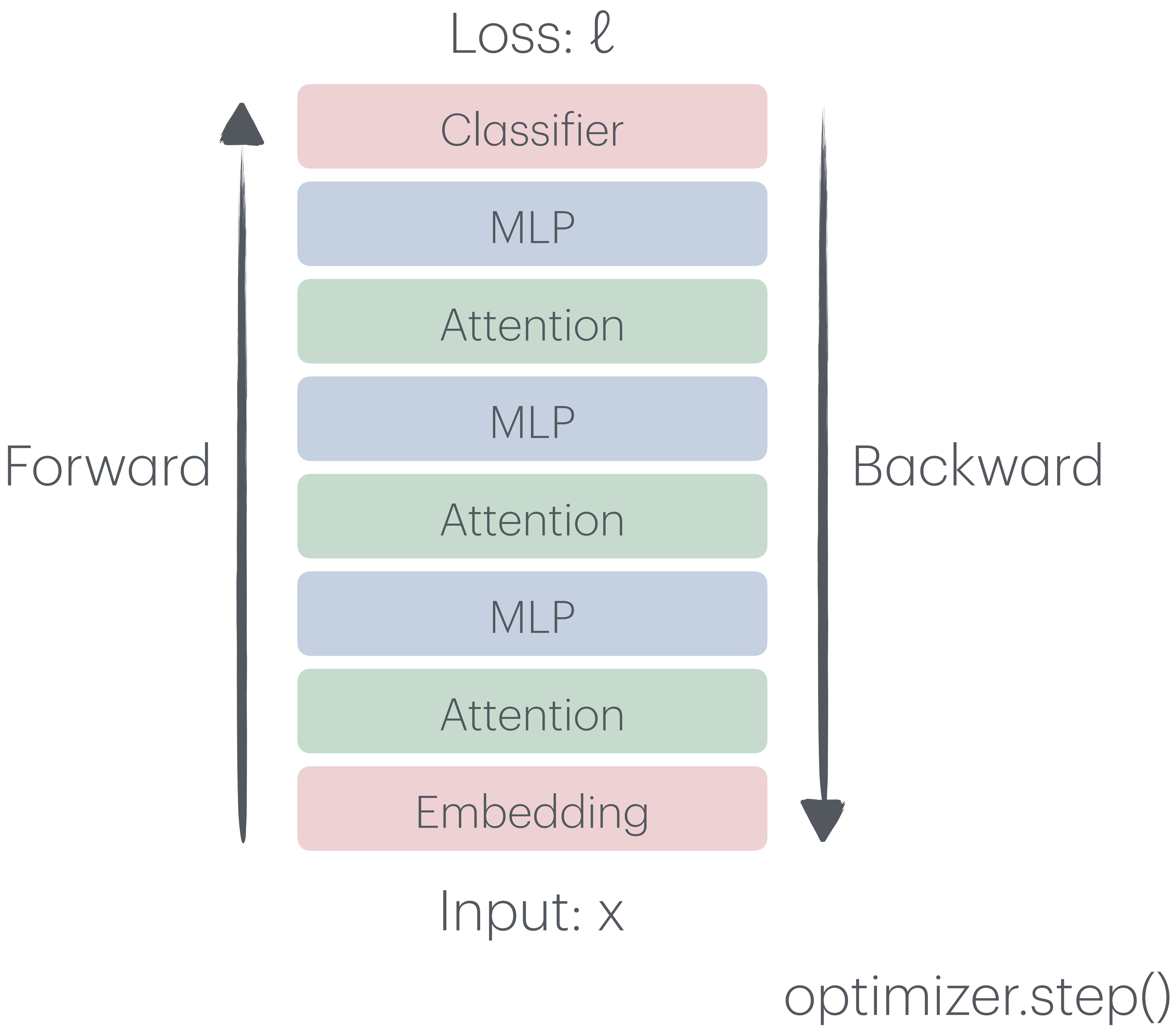
```
optimizer_dict = {}

def optimizer_hook(p):
    if p.grad is None:
        return
    optimizer_dict[p].step()
    optimizer_dict[p].zero_grad()

for p in model.parameters():
    if p.requires_grad:
        optimizer_dict[p] = AdamW([p], lr=…)
        p.register_post_accumulate_grad_hook(optimizer_hook)
```

[1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024

Backpropagation

A closer look

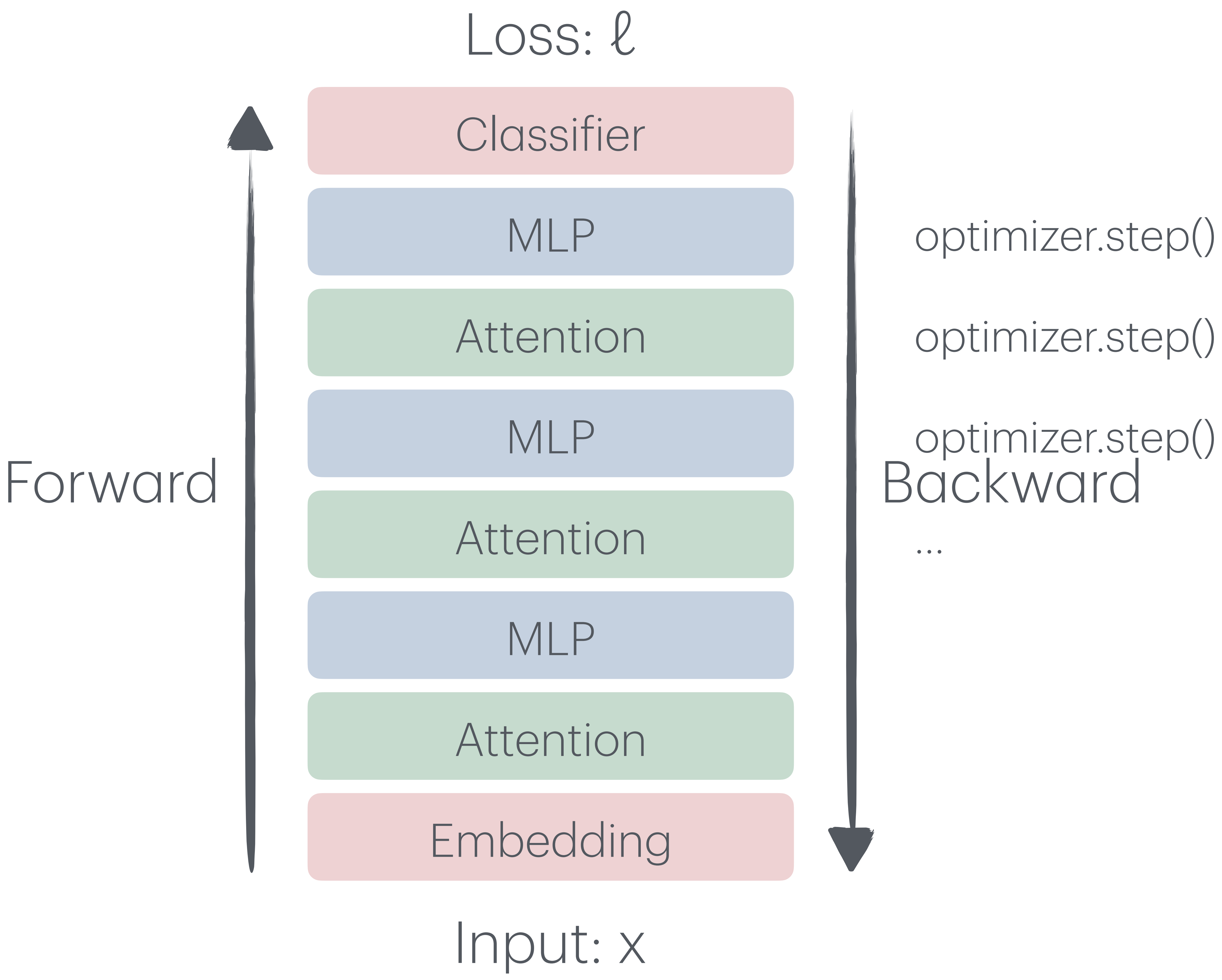
- Linear layer $y = W^\top x$
  - Gradient: $\frac{\partial}{\partial W} y = x y^\top$
  - Empirically, gradient is low-rank
- Galore
  - Can we update low-rank projection of gradient instead?



Galore

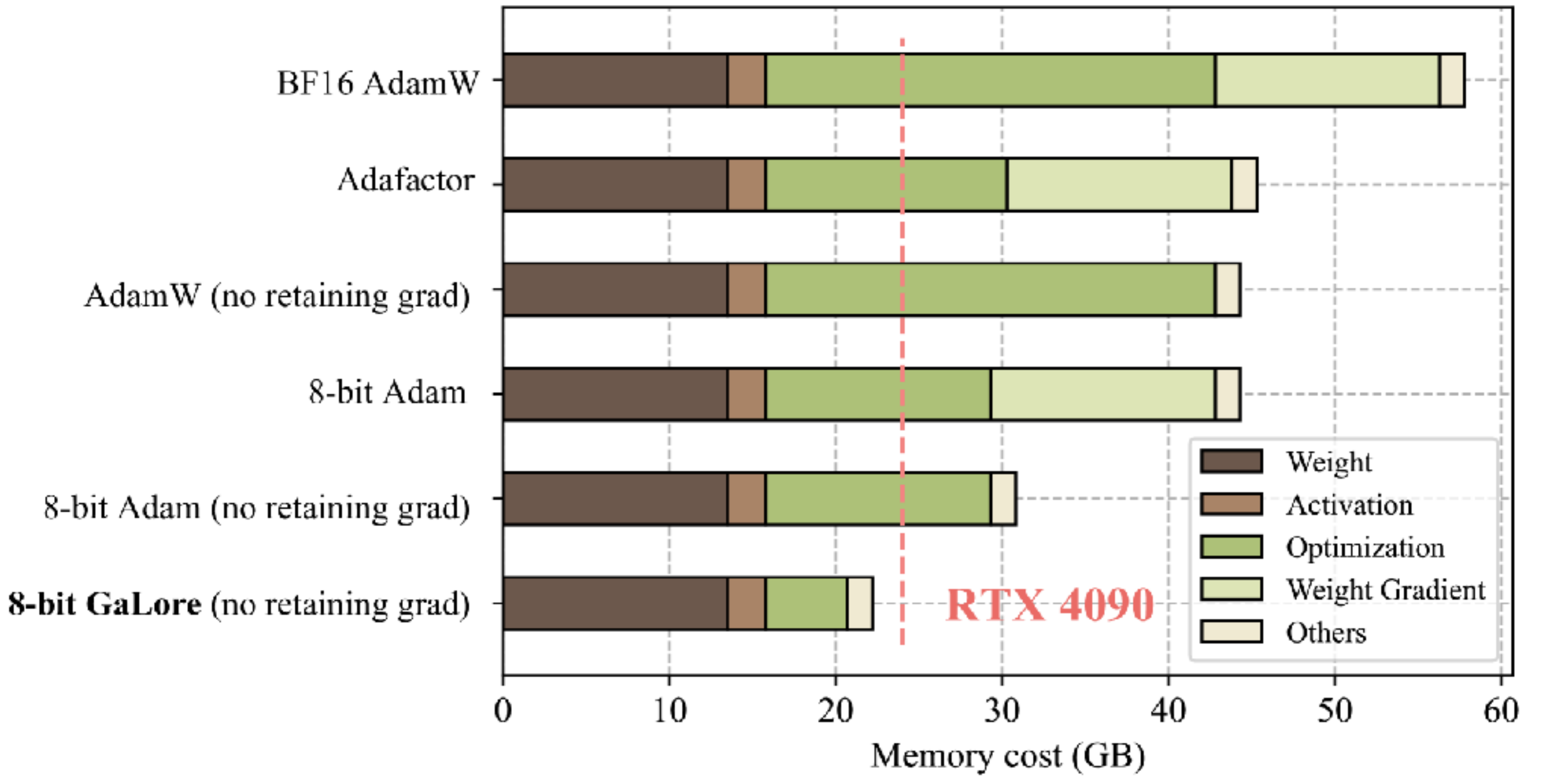
Low rank gradient

- Backward
    - Compute gradient (allocate memory)
    - Discard activation (free memory)
    - **Project gradient to low-rank subspace**
    - Discard gradient (free memory)
    - Optimizer.step **(low rank)**
    - Update weight **(full rank)**



[1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024

Galore

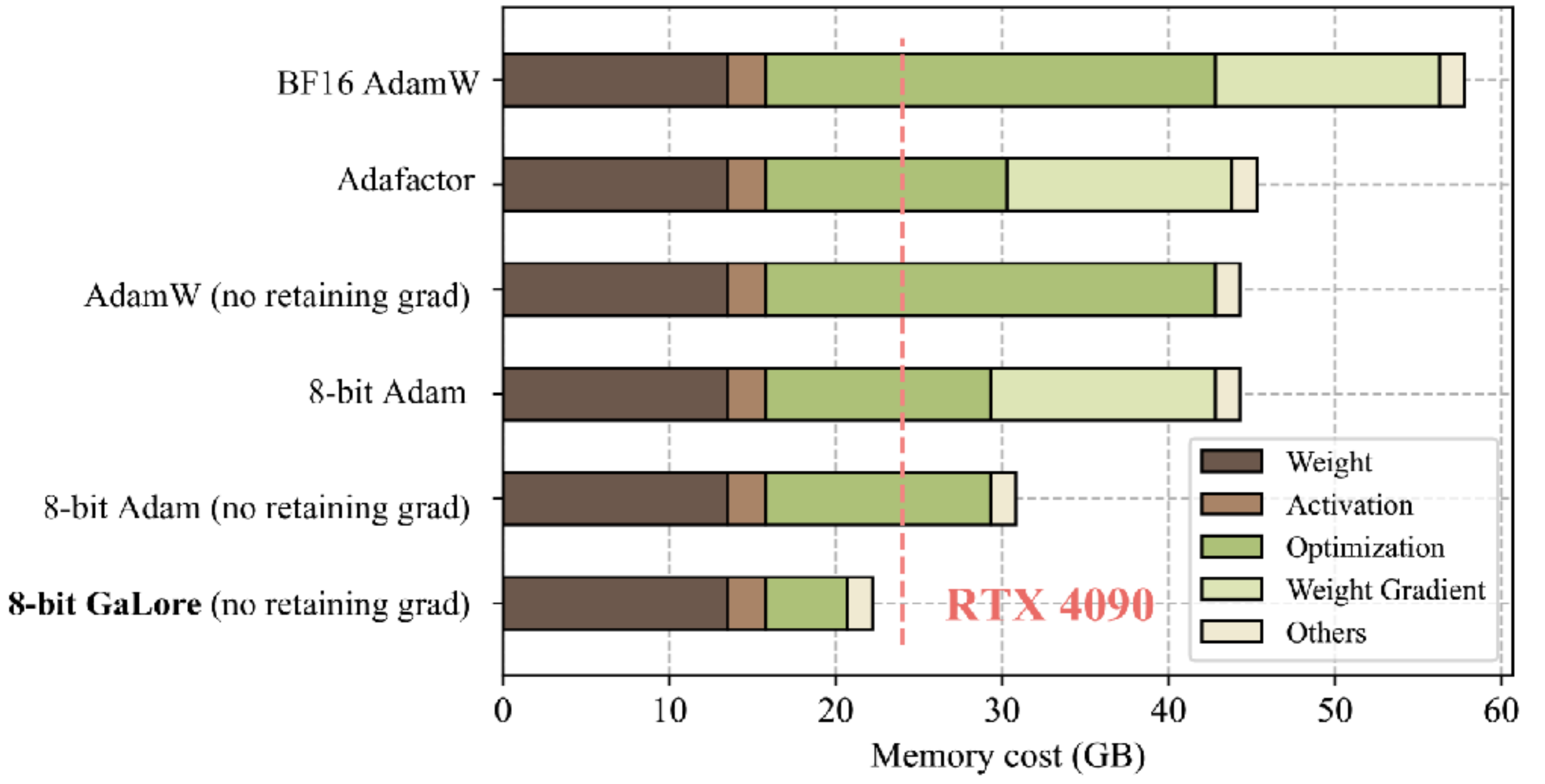


- Much smaller memory footprint for optimizer state
  - < 4N bytes of total memory
- Training full weights, not just LoRA adapter

[1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024

Galore

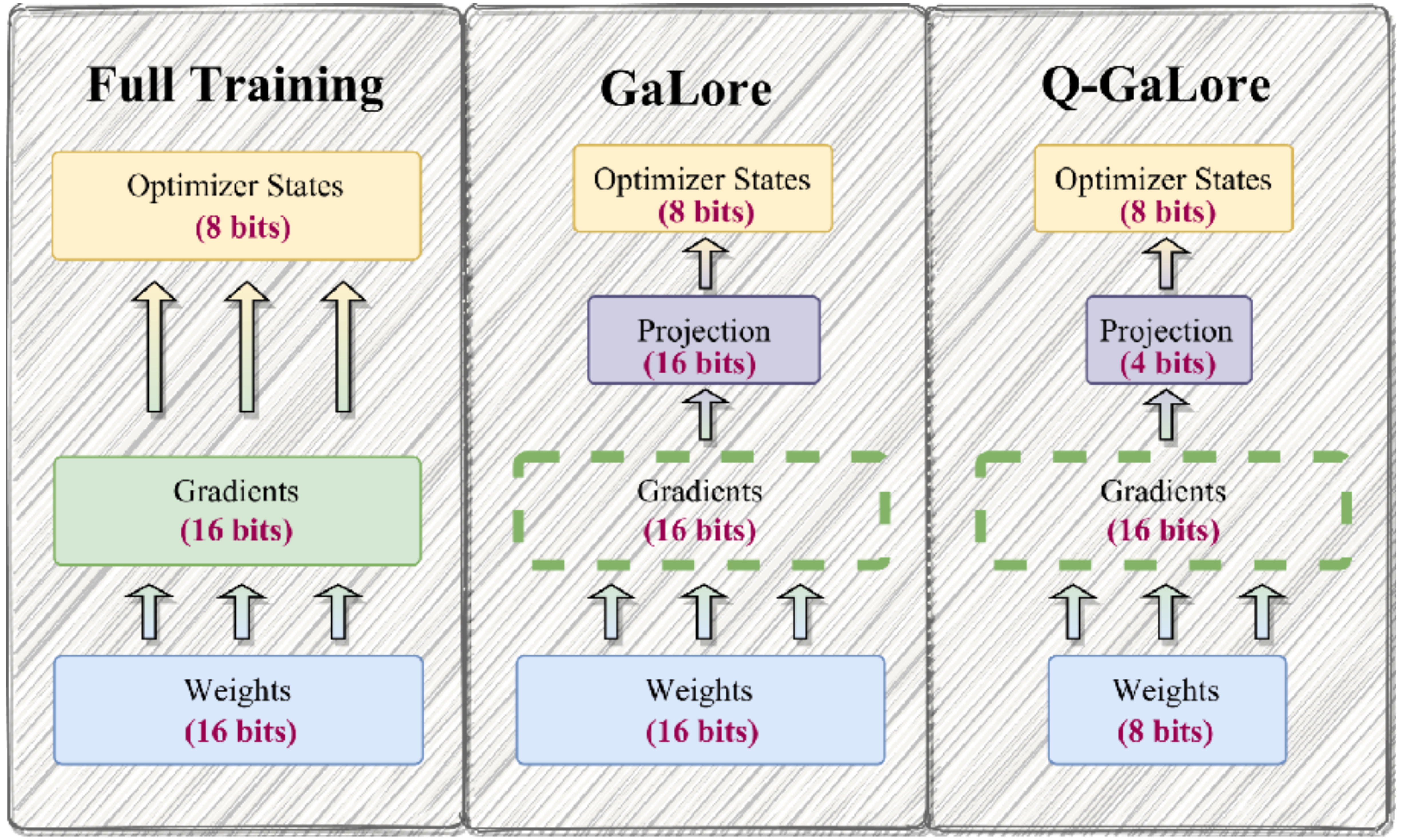
- How to we obtain low-rank projection?
  - SVD of gradient every K iterations
  - Fairly slow
- What happens to momentum after change in projection?
  - Nothing, we pretend we didn’t change anything



[1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024

Q-Galore

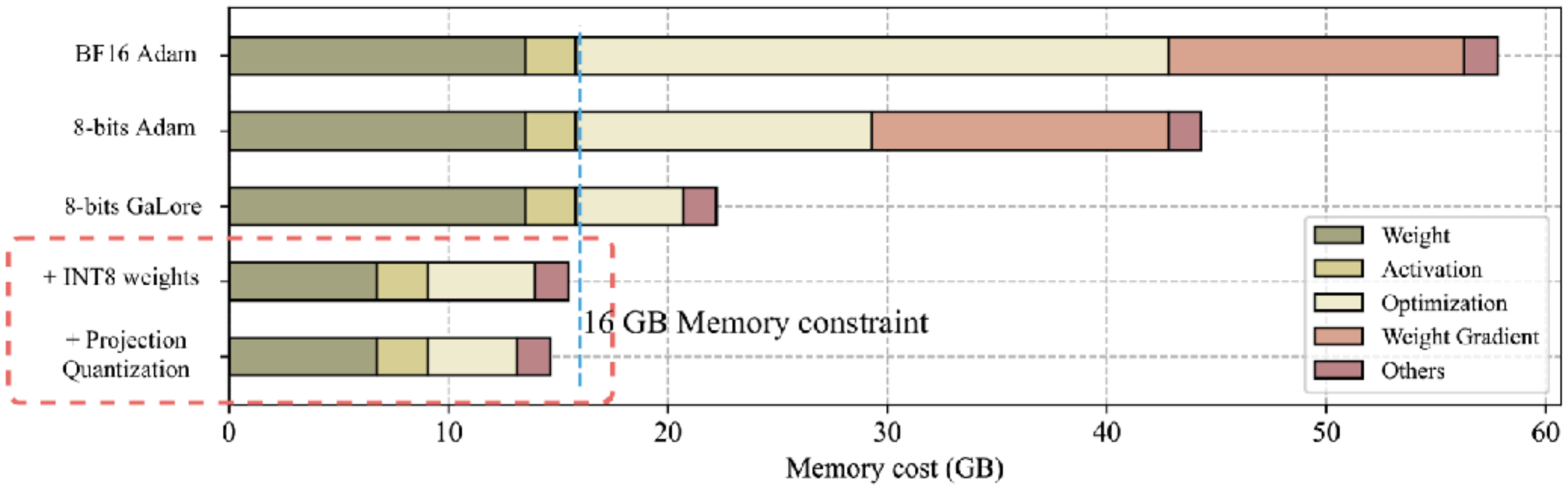
- Galore
  - Quantized weights + stochastic rounding
  - Quantized optimizer state
  - Quantized projection
  - Fewer SVD updates



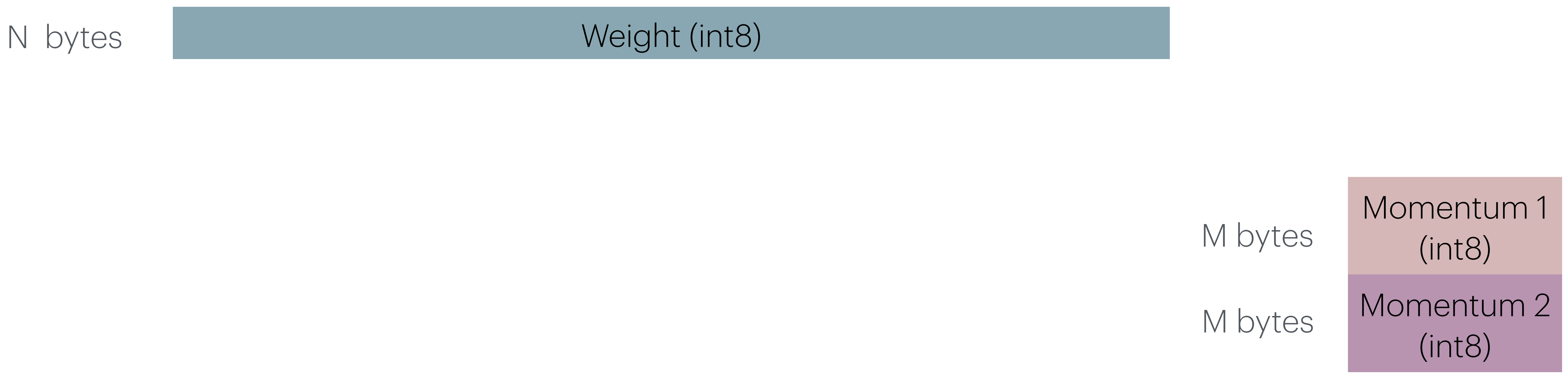
[1] Zhenyu Zhang, et al. Q-GaLore: Quantized GaLore with INT4 Projection and Layer-Adaptive Low-Rank Gradients. 2024

Training Q-Galore models

Memory requirements

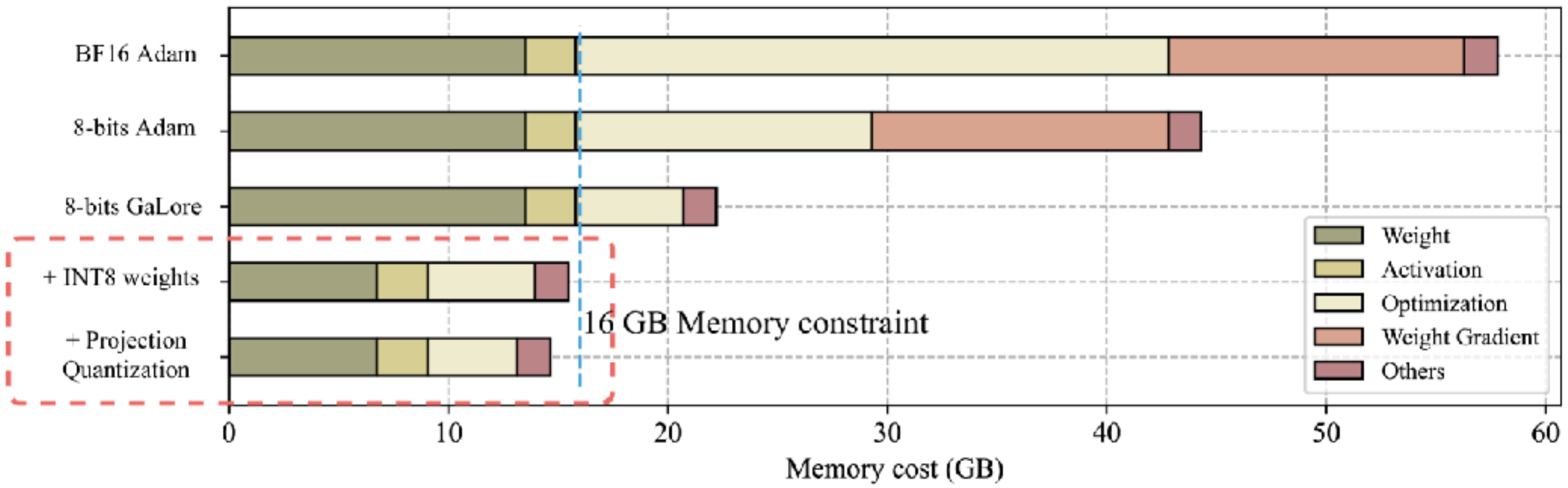


- Q-Galore
  - Model parameters: N, Galore parameters M
  - Weights: N int8
  - Momentum: M int8
  - 2nd momentum (ADAM): M int8
  - Projection: int4
- N+2M + |projection| bytes without activations



[1] Zhenyu Zhang, et al. Q-GaLore: Quantized GaLore with INT4 Projection and Layer-Adaptive Low-Rank Gradients. 2024

Galore discussion



Model	Methods	Memory	STEM	Social Sciences	Humanities	Other	Average
LLaMA-3-8B	**Full**	**48 GB**	54.27	75.66	59.08	72.80	64.85
	LoRA	**16 GB**	53.00	74.85	58.97	72.34	64.25
	GaLore	**16 GB**	54.40	75.56	58.35	71.19	64.24
	QLoRA	**8 GB**	53.63	73.44	58.59	71.62	63.79
	Q-GaLore	**8 GB**	53.27	75.37	58.57	71.96	64.20
Gemma-7B	**Full**	**51 GB**	30.03	37.16	34.08	35.47	34.21
	LoRA	**17 GB**	26.23	34.94	30.88	36.96	32.18
	GaLore	**17 GB**	27.33	36.74	30.82	37.90	33.20
	QLoRA	**9 GB**	24.83	27.54	28.09	33.40	28.49
	Q-GaLore	**9 GB**	27.73	36.80	32.54	37.89	33.68
Mistral-7B	**Full**	**43 GB**	52.40	72.95	55.16	69.05	61.67
	LoRA	**14 GB**	52.13	72.46	55.05	68.77	61.41
	GaLore	**14 GB**	51.50	73.02	55.03	69.49	61.55
	QLoRA	**7 GB**	50.00	71.29	55.84	67.66	60.70
	Q-GaLore	**7 GB**	52.23	72.82	55.01	69.30	61.62

- Able to train all weights (not just adapter)
- Not very stable
- Slower
- No gradient accumulation without changes to algorithm
    - Single GPU training

[1] Zhenyu Zhang, et al. Q-GaLore: Quantized GaLore with INT4 Projection and Layer-Adaptive Low-Rank Gradients. 2024

References

- [1] Jiawei Zhao, et al. GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection. 2024. (link)
- [2] Zhenyu Zhang, et al. Q-GaLore: Quantized GaLore with INT4 Projection and Layer-Adaptive Low-Rank Gradients. 2024. (link)