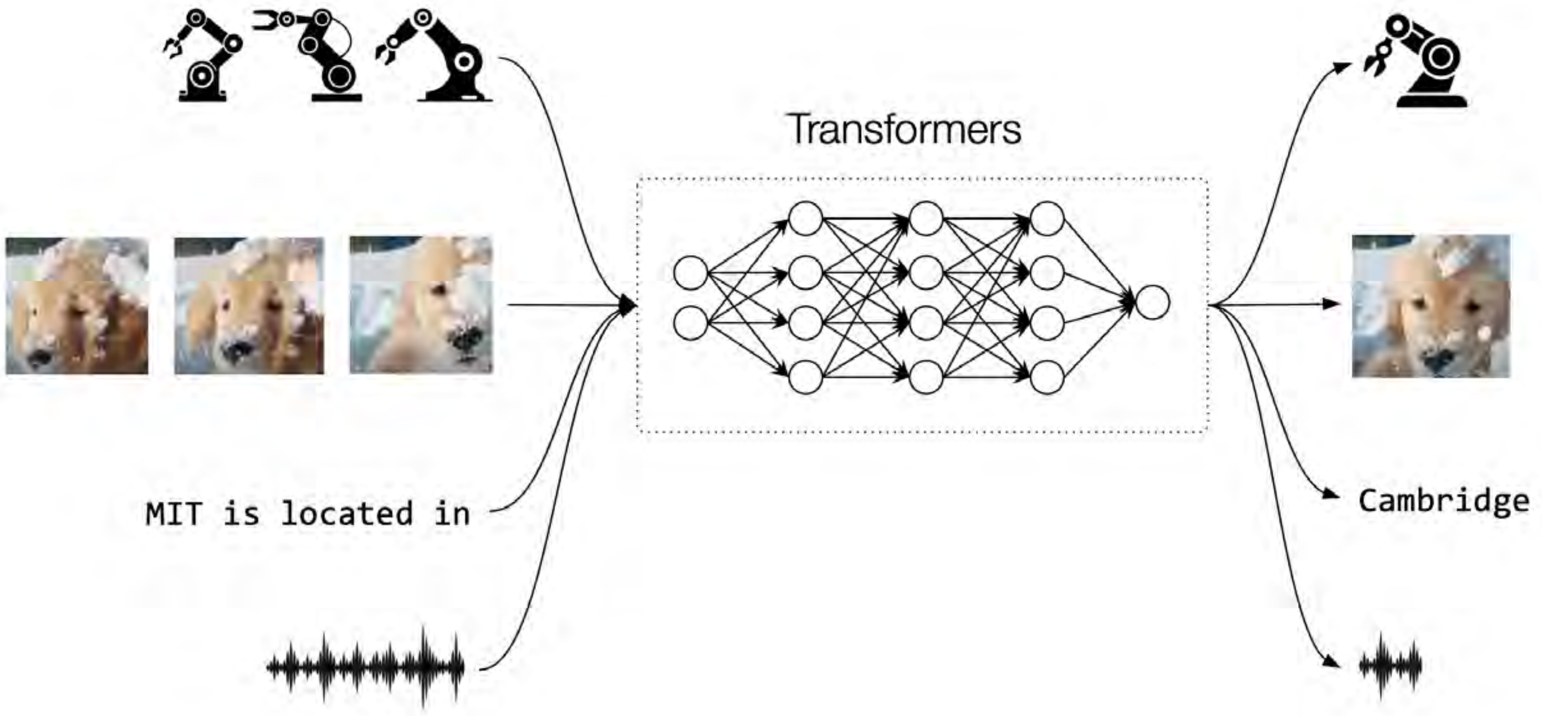
Hardware-efficient Neural Architectures for Language Modeling

Lucas Torroba-Hennigen, PhD Candidate, MIT CSAIL

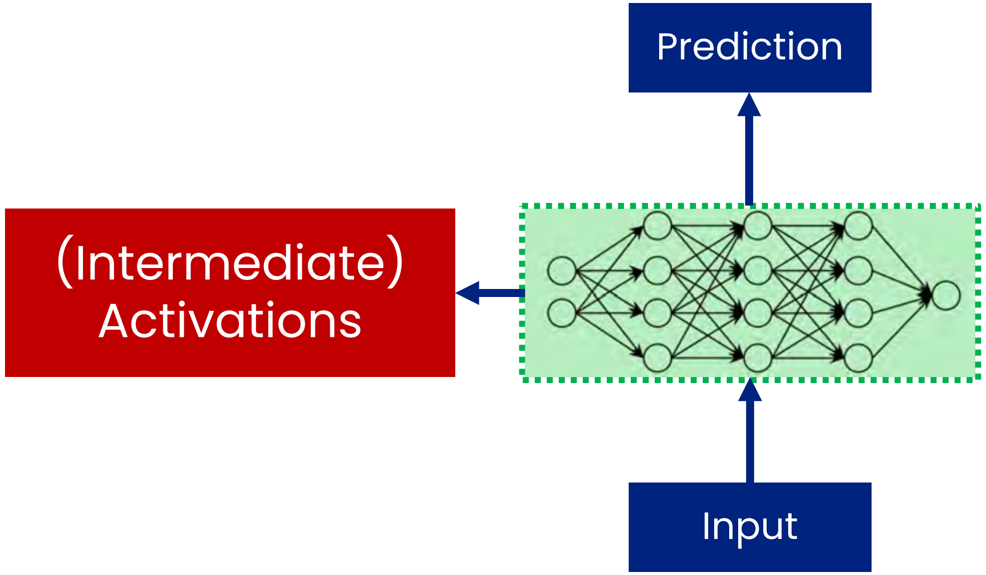
Transformers are ubiquitous



LLM training process

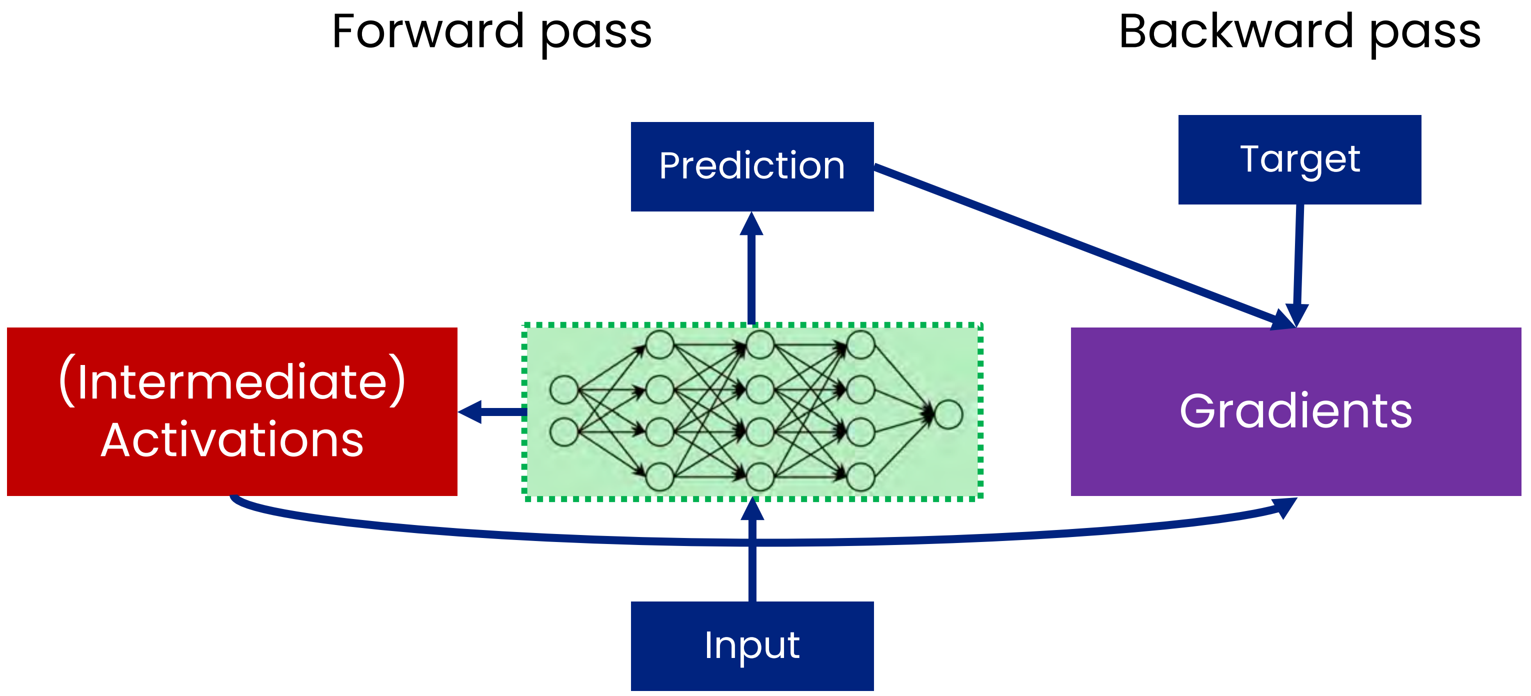
Training LLMs is done using an iterative process, based on gradient descent

Forward pass



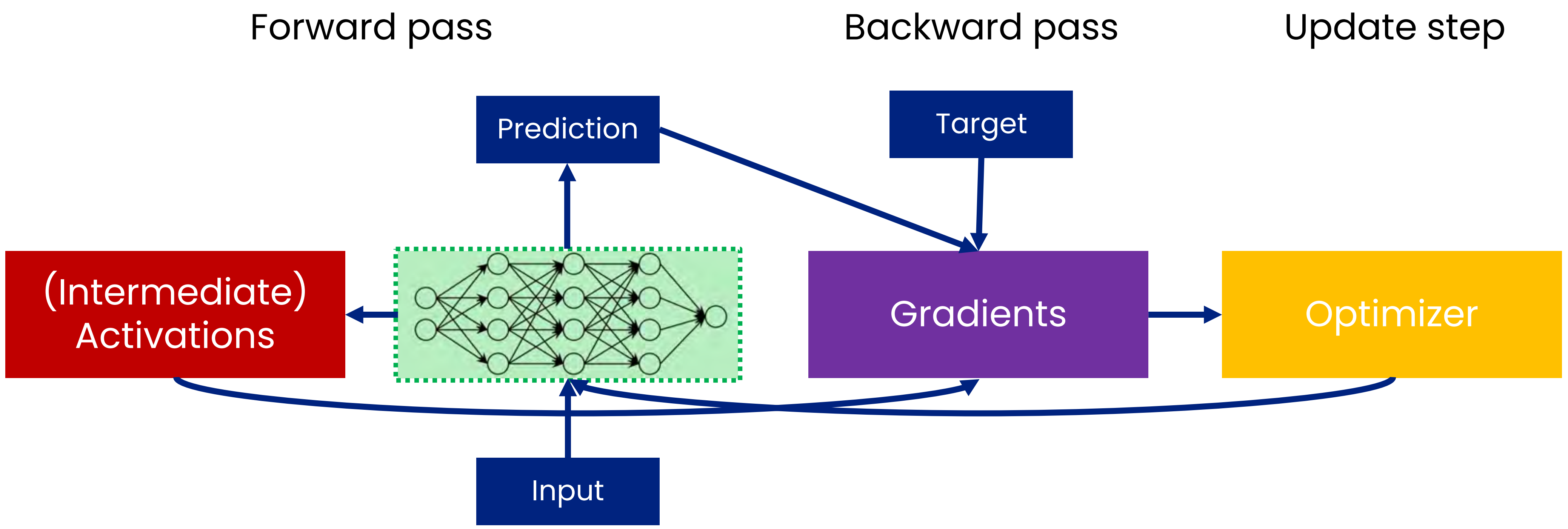
LLM training process

Training LLMs is done using an iterative process, based on gradient descent



LLM training process

Training LLMs is done using an iterative process, based on gradient descent



Memory bottlenecks

Memory usage adds up, even for moderate-sized LLMs:

Activations	Model	Gradients	Optimizer states
varies	1 float/parameter 140 GB (70B)	1 float/parameter 140 GB (70B)	~2 float/parameter 280 GB (70B)

Memory bottlenecks

Memory usage adds up, even for moderate-sized LLMs:

Activations	Model	Gradients	Optimizer states
varies	1 float/parameter 140 GB (70B)	1 float/parameter 140 GB (70B)	~2 float/parameter 280 GB (70B)

So as models become larger, we are increasingly bottlenecked by memory, a precious resource in GPUs

A100/H100 (~80GB) H200 (~140GB) B200 (~190GB)

Memory bottlenecks

Memory usage adds up, even for moderate-sized LLMs:

Activations	Model	Gradients	Optimizer states
varies	1 float/parameter 140 GB (70B)	1 float/parameter 140 GB (70B)	~2 float/parameter 280 GB (70B)

So as models become larger, we are increasingly bottlenecked by memory, a precious resource in GPUs

A100/H100 (~80GB) H200 (~140GB) B200 (~190GB)

This has motivated a range of work in memory-efficient training methods that decouple model size from the hardware used to train them

Relating memory-efficient methods

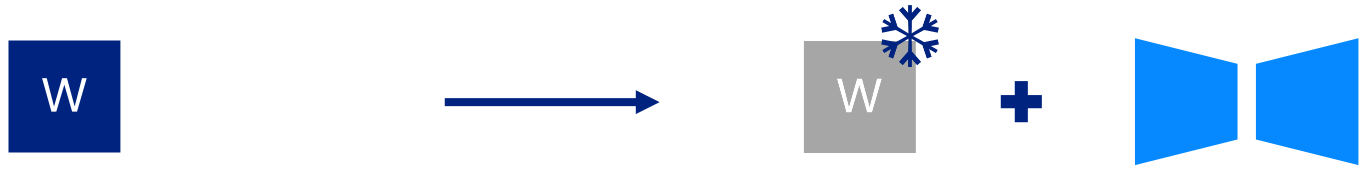
In a recent preprint (Torroba-Hennigen et al., 2025), we develop an equivalence between two classes of memory-efficient training methods:

Relating memory-efficient methods

In a recent preprint (Torroba-Hennigen et al., 2025), we develop an equivalence between two classes of memory-efficient training methods:

Adapter-based
(change model)

e.g., LoRA (Hu et al., 2021)

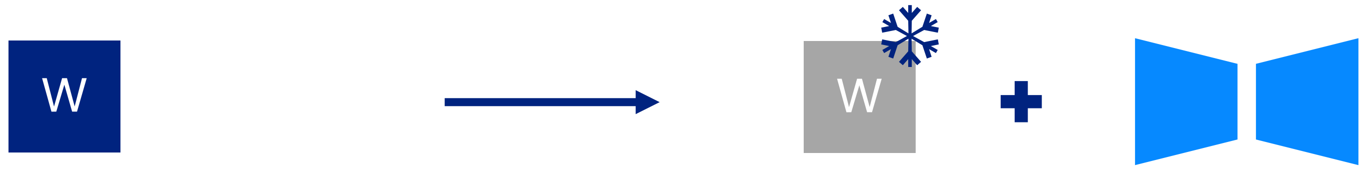


Relating memory-efficient methods

In a recent preprint (Torroba-Hennigen et al., 2025), we develop an equivalence between two classes of memory-efficient training methods:

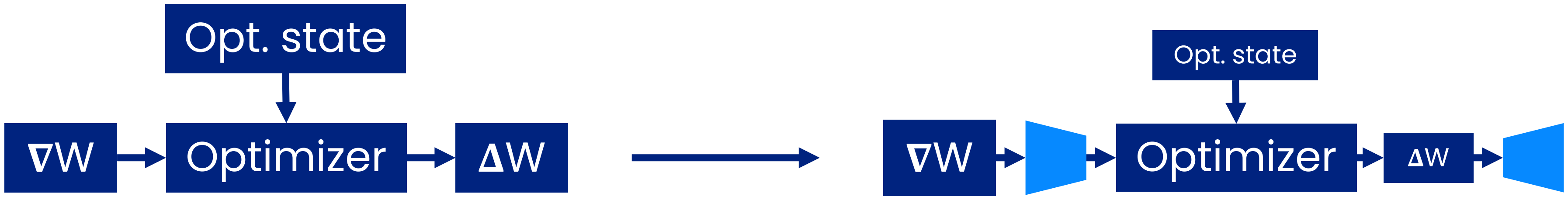
Adapter-based
(change model)

e.g., LoRA (Hu et al., 2021)



Gradient-based
(change optimizer)

e.g., GaLore (Zhao et al., 2024)



Relating memory-efficient methods

In a recent preprint (Torroba-Hennigen et al., 2025), we develop an equivalence between two classes of memory-efficient training methods:

Adapter-based
(change model)

e.g., LoRA (Hu et al., 2021)

$$W \to W + AS^{\top}$$

$$\Leftrightarrow$$

Gradient-based
(change optimizer)

e.g., GaLore (Zhao et al., 2024)

$$\eta(\overline{W}) \to \eta(\overline{W}S)S^{T}$$

Relating memory-efficient methods

In a recent preprint (Torroba-Hennigen et al., 2025), we develop an equivalence between two classes of memory-efficient training methods:

Adapter-based
(change model)

e.g., LoRA (Hu et al., 2021)

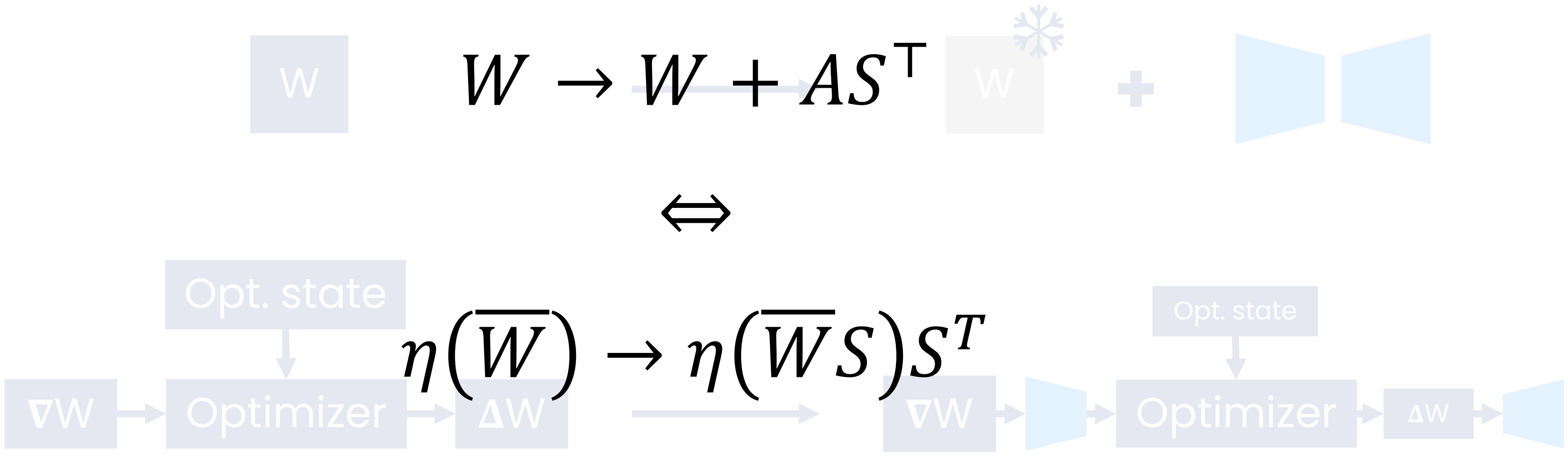
$$W \to W + AS^{\top}$$

$$\Leftrightarrow$$

Gradient-based
(change optimizer)

e.g., GaLore (Zhao et al., 2024)

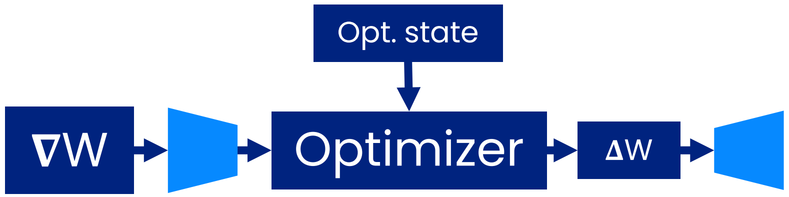
$$\eta(\overline{W}) \to \eta(\overline{W}S)S^{T}$$



This duality allows us to use insights from one method to improve the other

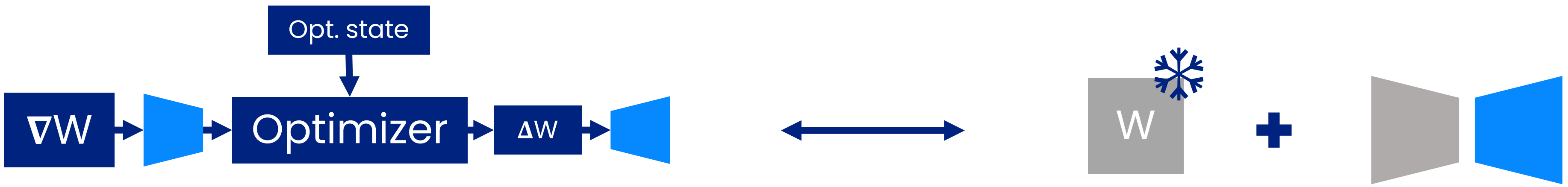
Improvements via quantization

Drawing from the adapter literature (e.g., QLoRA; Dettmers et al., 2023), we can improve gradient-based approaches by adding quantization



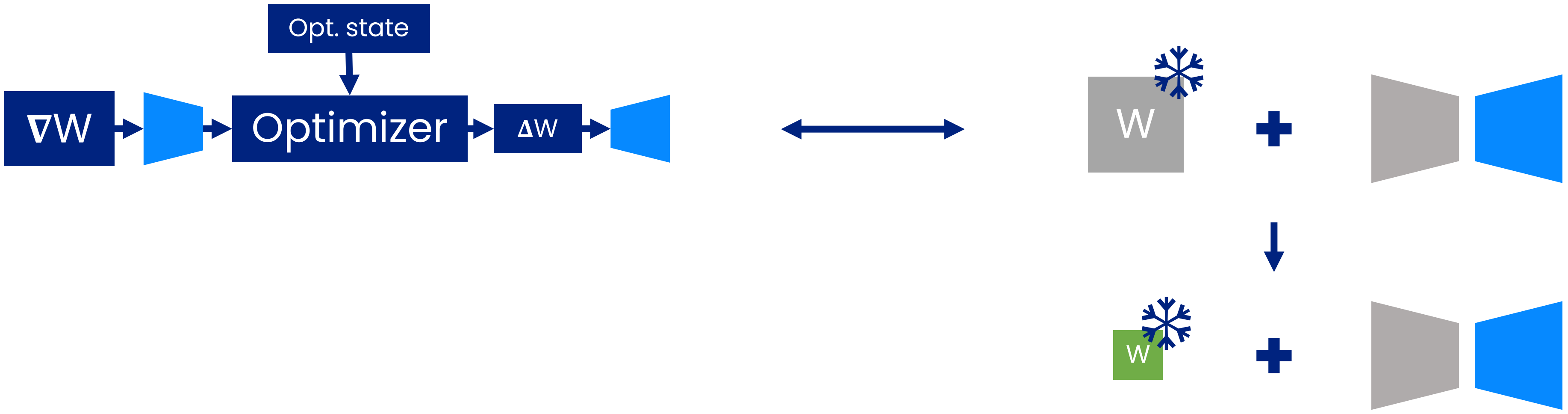
Improvements via quantization

Drawing from the adapter literature (e.g., QLoRA; Dettmers et al., 2023), we can improve gradient-based approaches by adding quantization



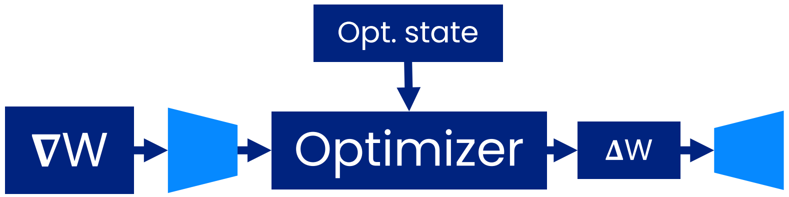
Improvements via quantization

Drawing from the adapter literature (e.g., QLoRA; Dettmers et al., 2023), we can improve gradient-based approaches by adding quantization



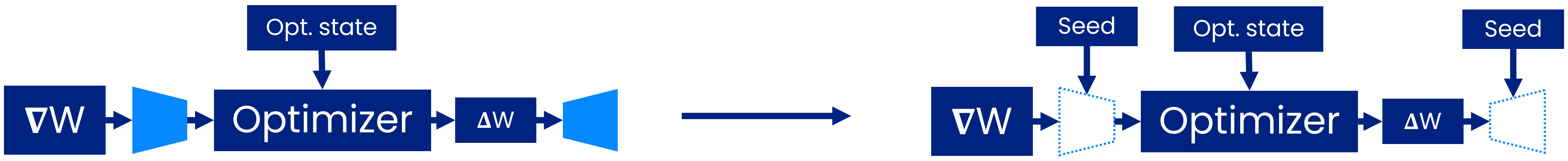
Improvements via rematerialization

Drawing from the gradient-based efficient training literature, we can consider other choices of projections that can be implemented better in hardware



Improvements via rematerialization

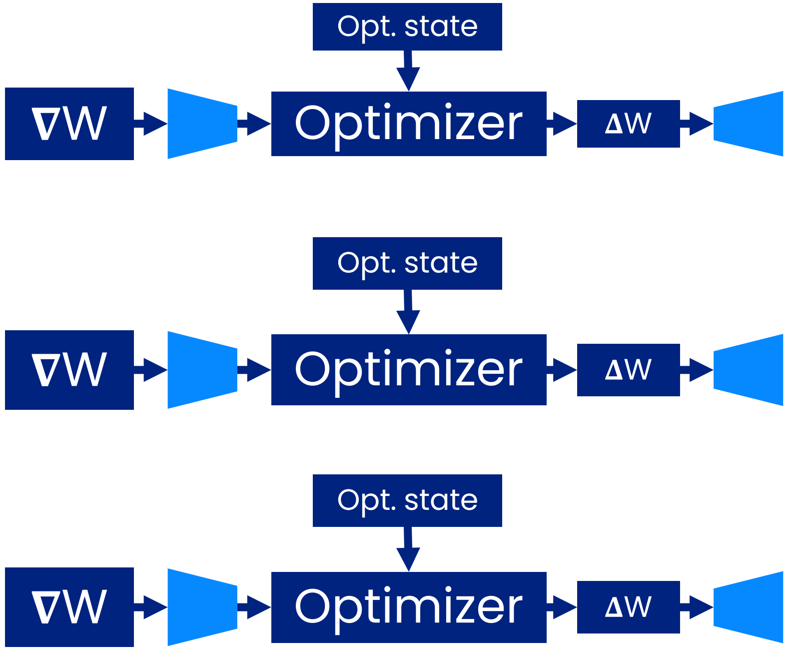
Drawing from the gradient-based efficient training literature, we can consider other choices of projections that can be implemented better in hardware



This reduces memory consumption and (potentially) memory movement

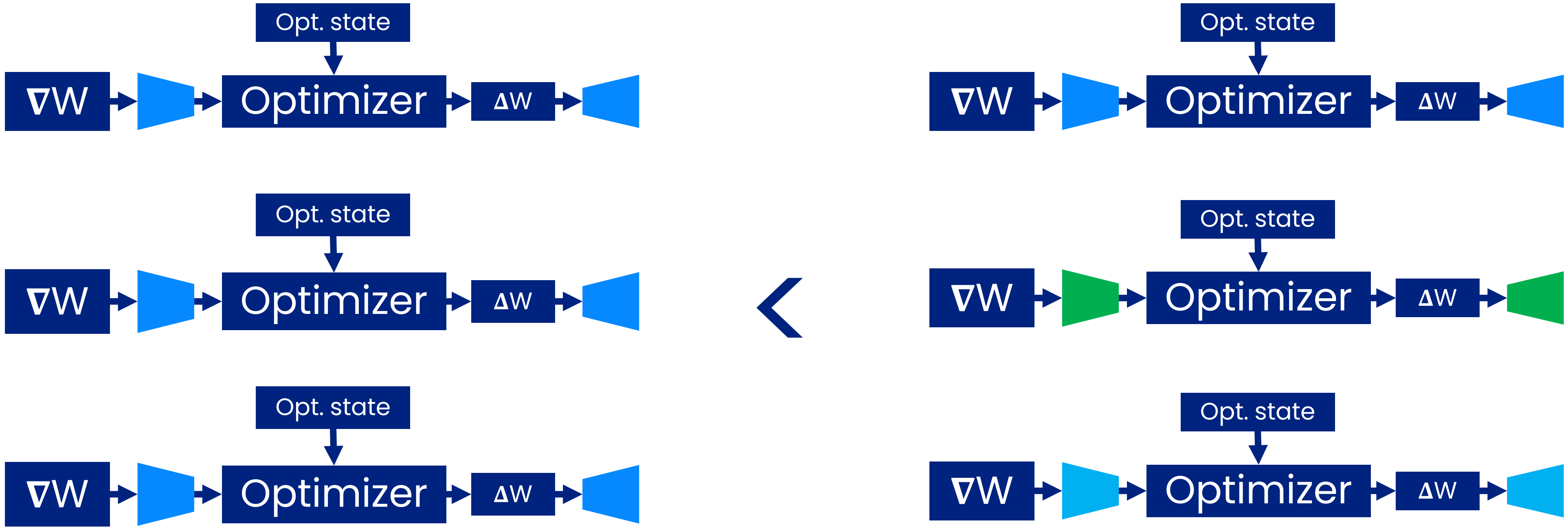
Improving distributed training

We also find that for distributed training with poorly connected, memory-constrained workers, choosing the right projections is important



Improving distributed training

We also find that for distributed training with poorly connected, memory-constrained workers, choosing the right projections is important



Future work

Some directions we are considering to pursue next:

1. Single-node setting
   - More complex projections
   - In-register rematerialization
2. Distributed setting
   - Reduce communication cost
   - Reduce stall time when synchronizing gradients

Thanks!