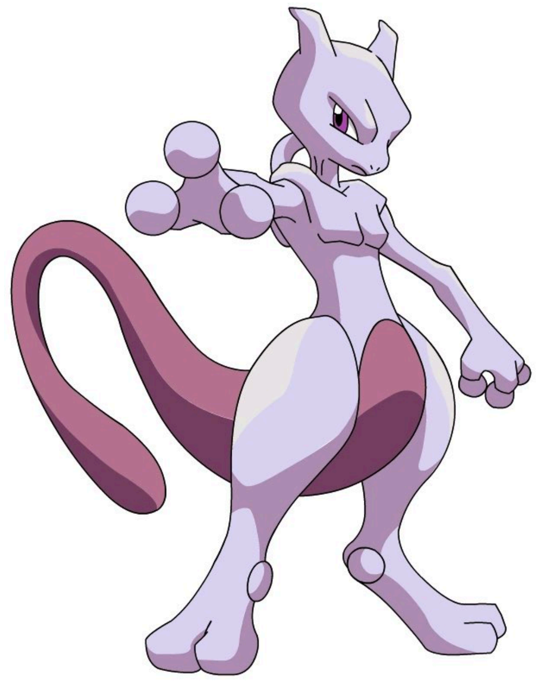


μ-TWO: 3x Faster Multi-model Training with Orchestration and Memory Optimization

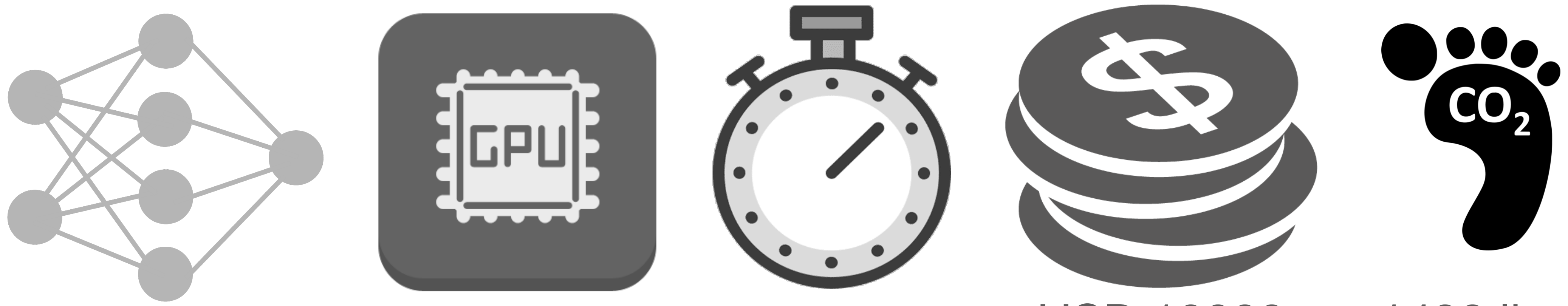
Sanket Purandare
Abdul Wasay
Animesh Jain
Stratos Idreos

@ Harvard SEAS

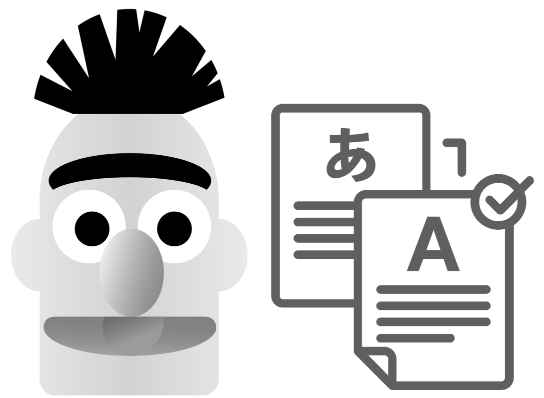
Deep Learning Training

↓

High Computational Cost



200M | 64 GPUs | 79 hrs | USD 12000 | 1438 lbs



BERT

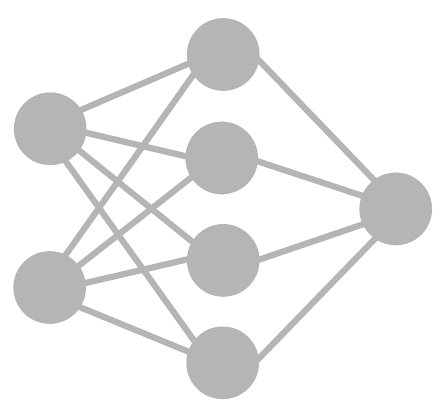
Natural Language Processing

Strubell et al., Energy and policy considerations for deep learning in NLP, 2019

Deep Learning Training

↓

High Computational Cost

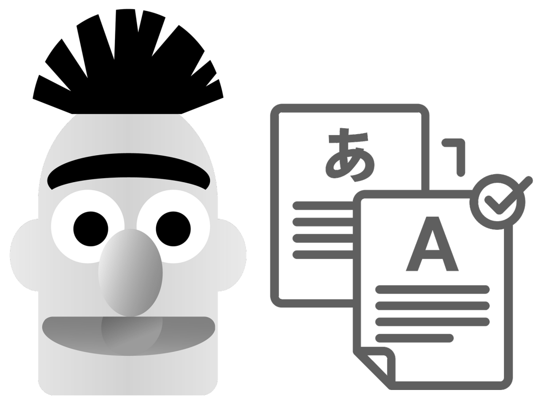

200M

64 GPUs

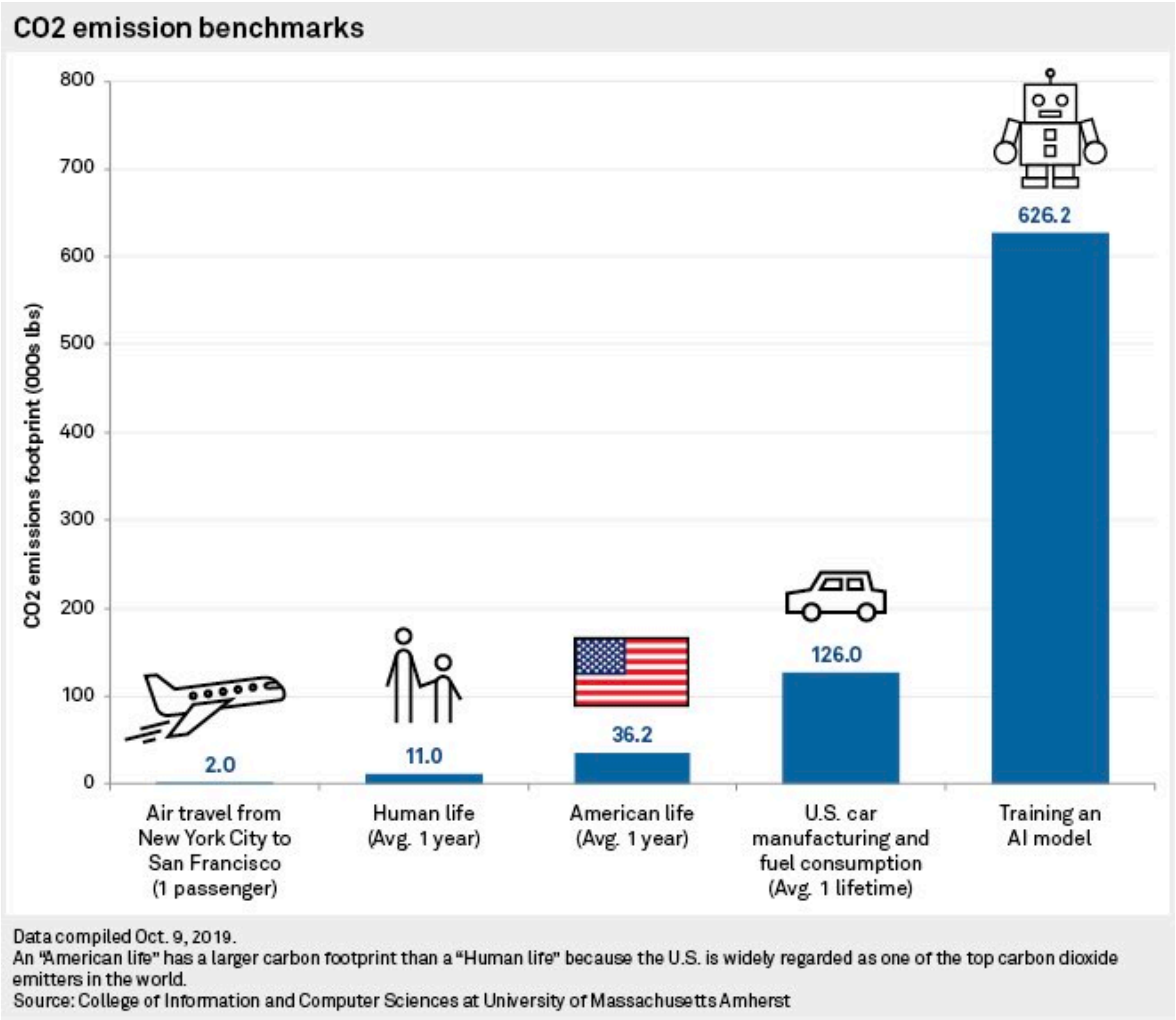

79 hrs


USD 12000

1438 lbs



Google BERT



Strubell et al., Energy and policy considerations for deep learning in NLP, 2019

DASlab @ Harvard SEAS

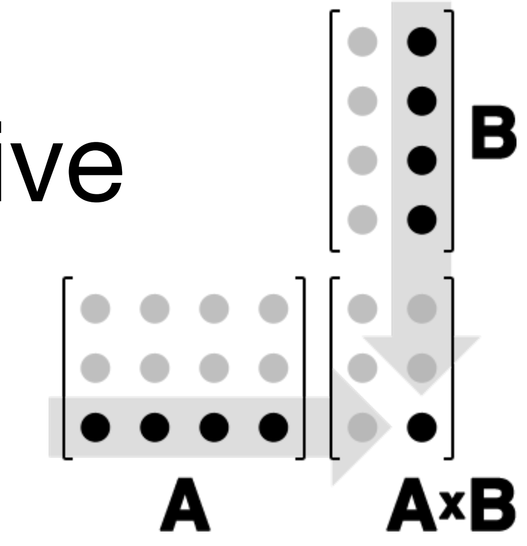
Deep Learning Training

↓

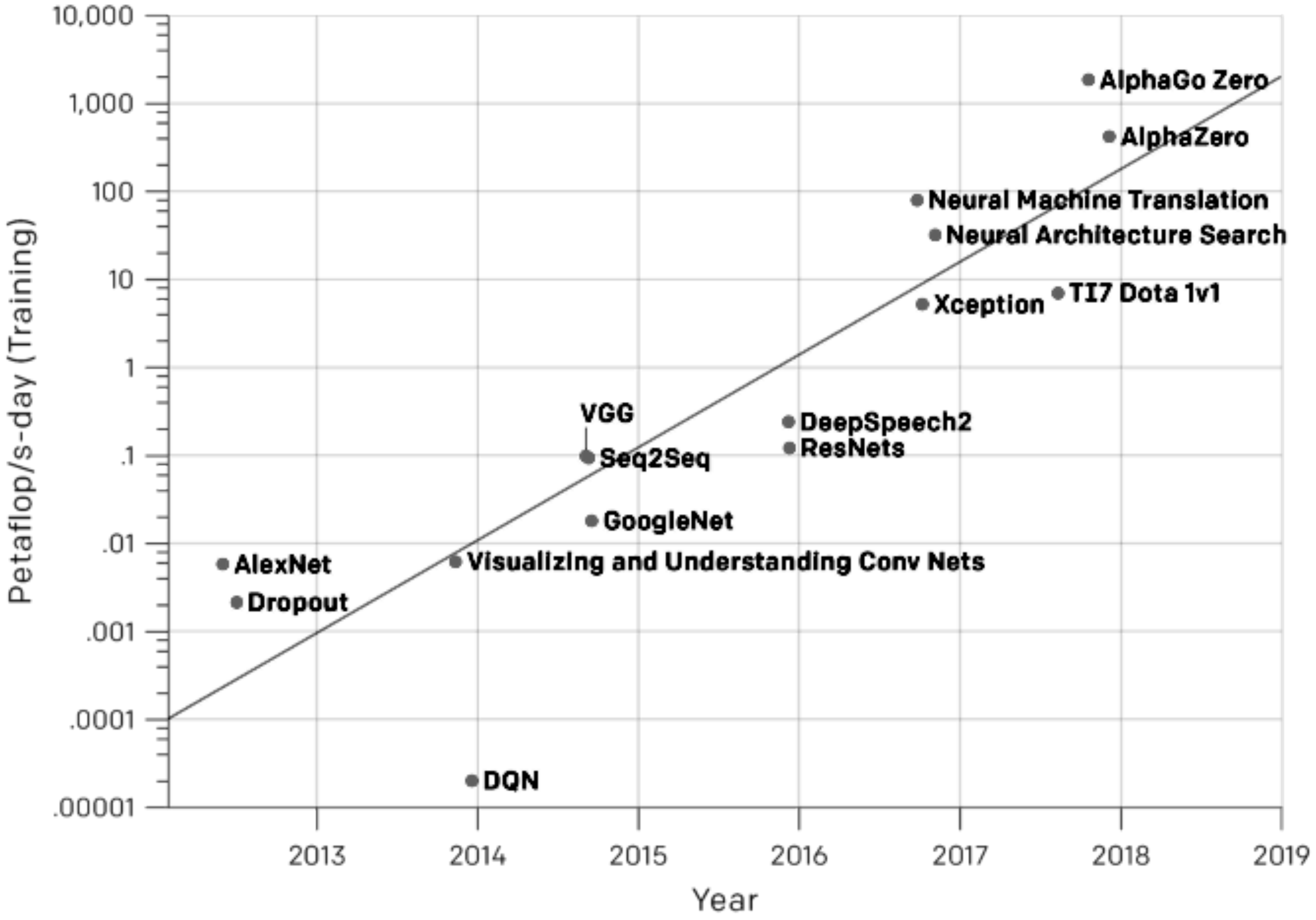
High Computational Cost

Sources?

Compute Intensive



300,000x Increase in compute



Open AI: https://openai.com/blog/ai-and-compute/

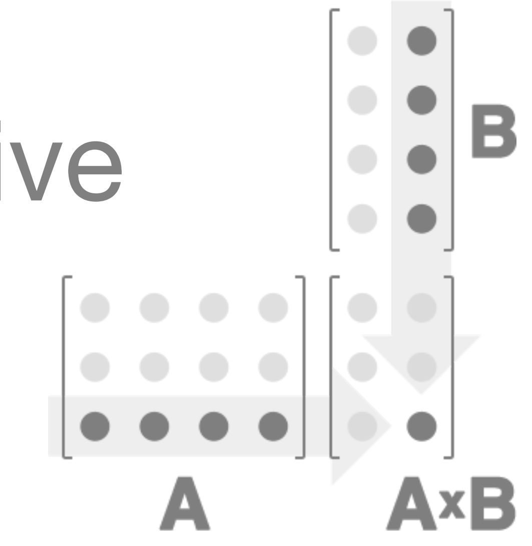
DASlab @ Harvard SEAS

Deep Learning Training

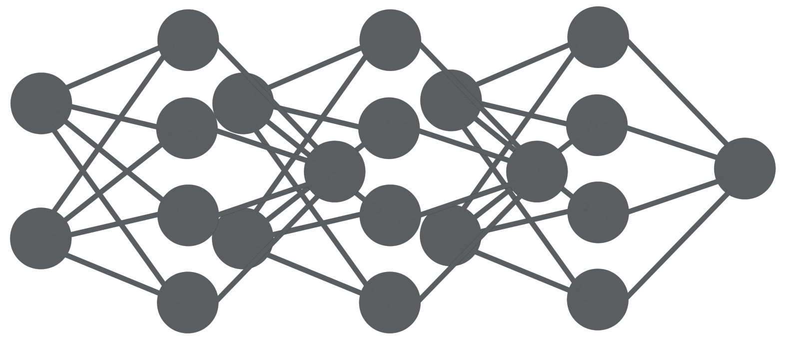
High Computational Cost

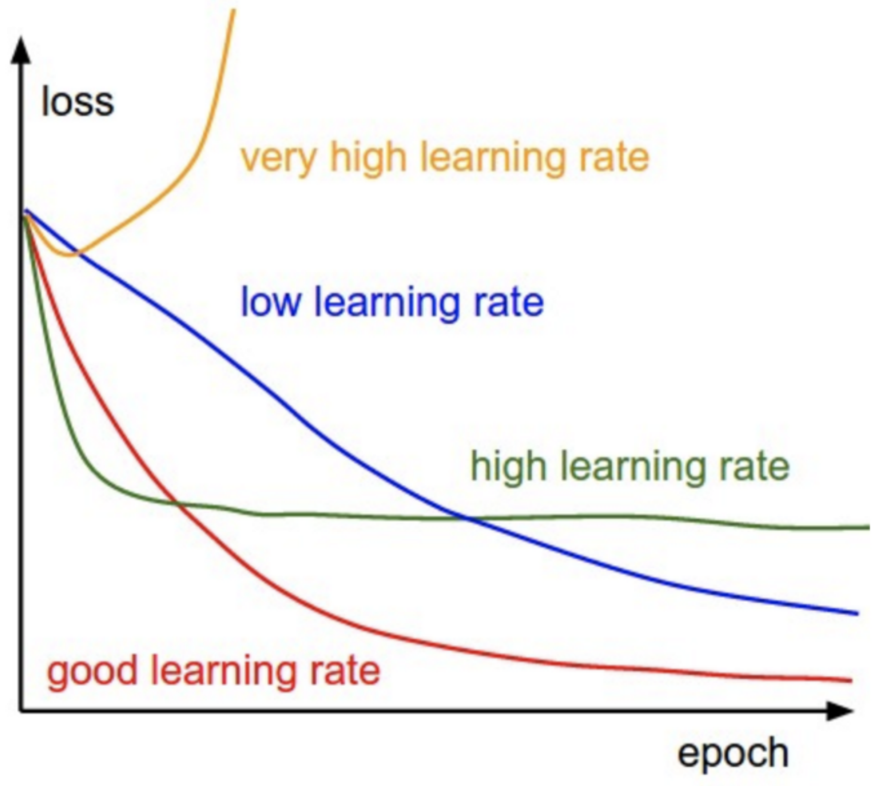
Sources?

Compute Intensive



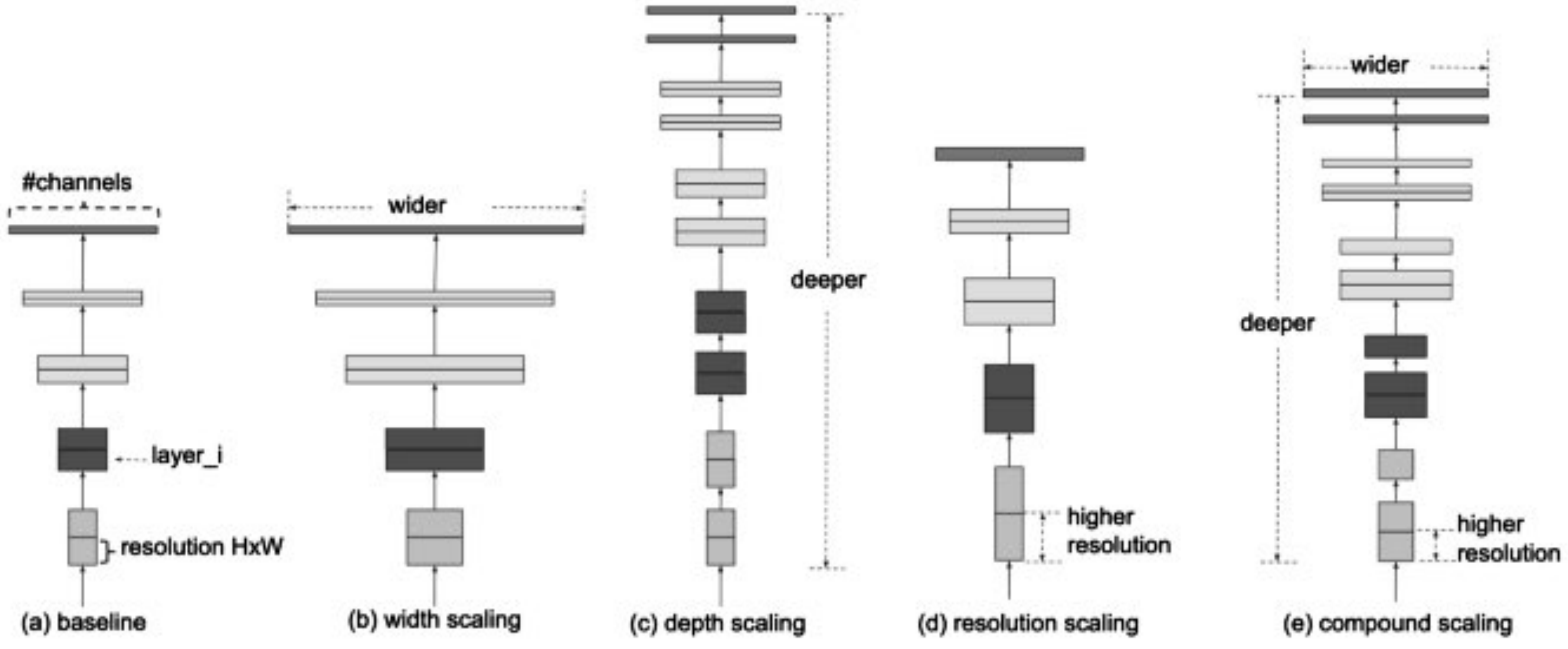
Training
Several
Networks





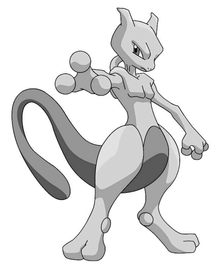
Bergstra, J., Algorithms for hyper-parameter optimization, NeurIPS 2011.

Hyperparameter
Tuning



T. Elsken et al, Neural Architecture Search: A Survey, JMLR 2019

Neural
Architecture
Search



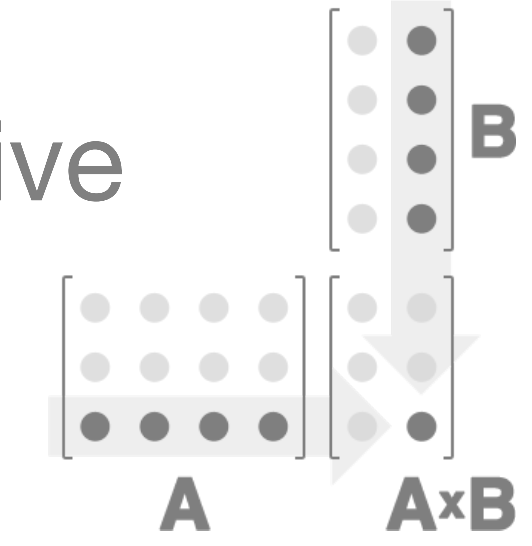
DASlab
@ Harvard SEAS

Deep Learning Training

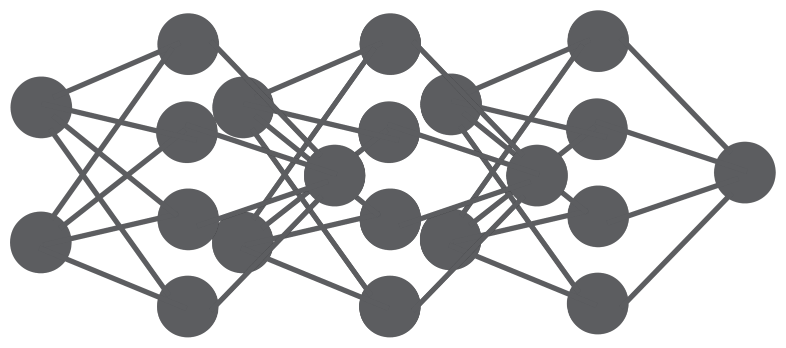
High Computational Cost

Sources?

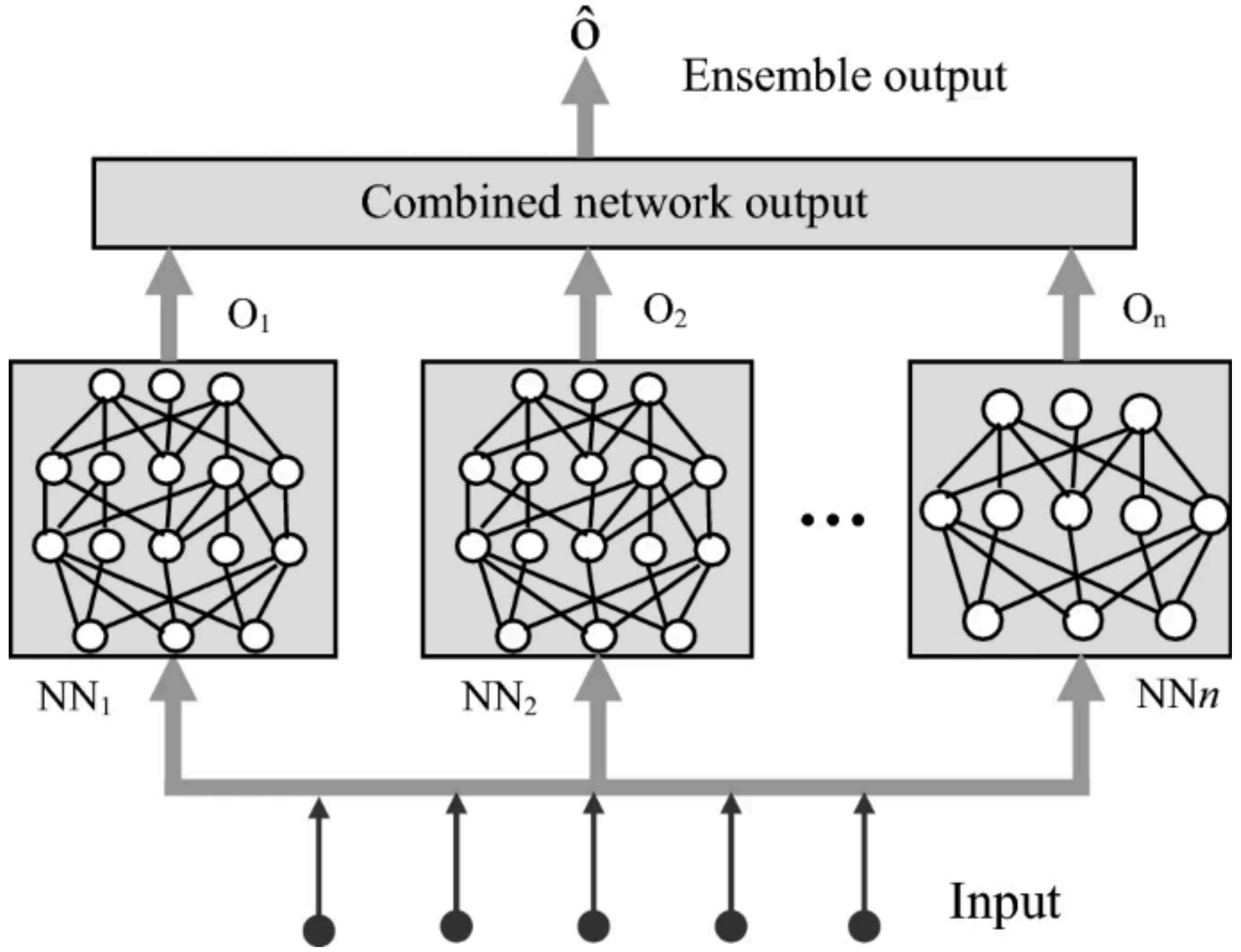
Compute Intensive



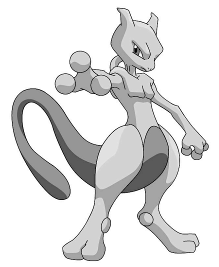
Training
Several
Networks



Ensemble Training



Ganaie MA., Ensemble deep learning: A review, 2021



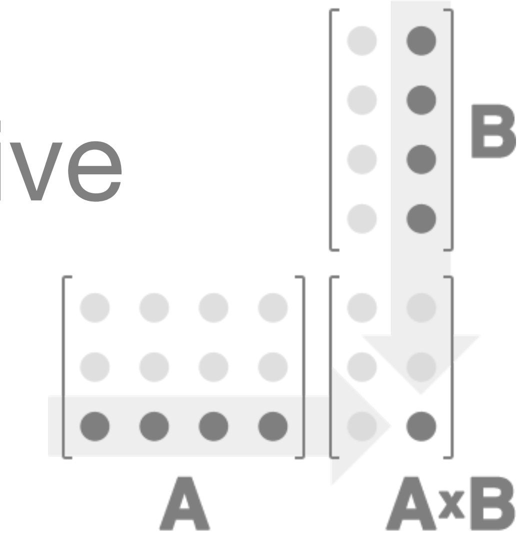
DASlab @ Harvard SEAS

Deep Learning Training

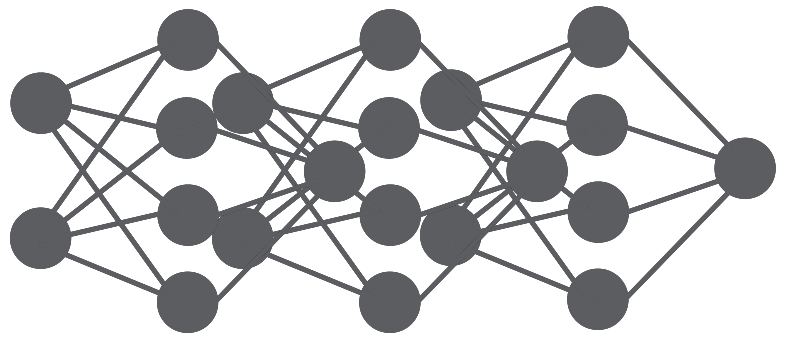
High Computational Cost

Sources?

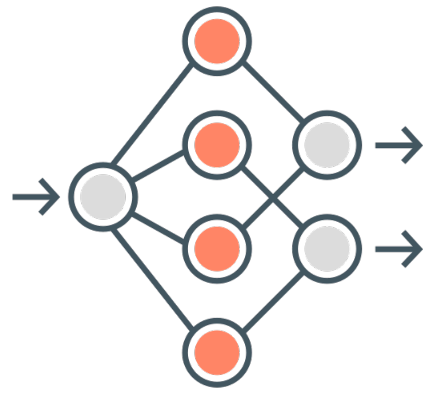
Compute Intensive



Training Several Networks

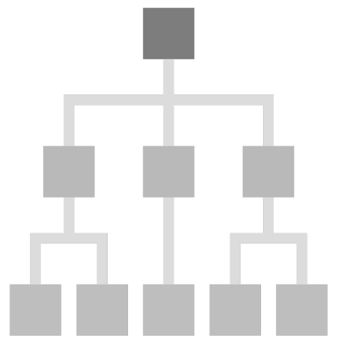


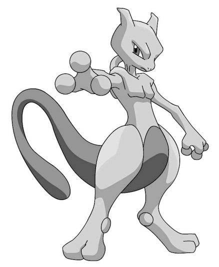
Interpretability Studies: Class Representations



Adversarial ML: Data poisoning

ML for Systems: Indexing





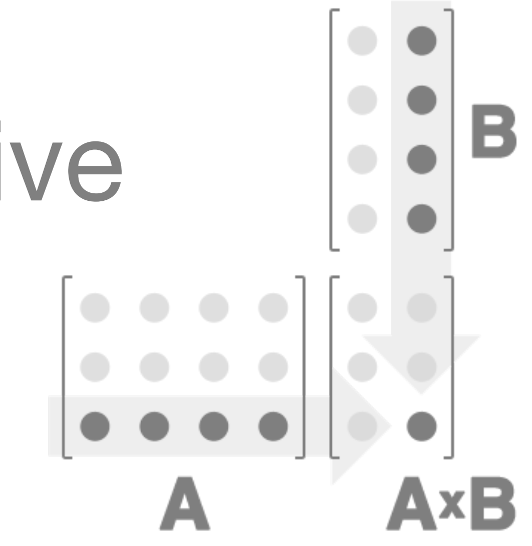
DASlab @ Harvard SEAS

Deep Learning Training

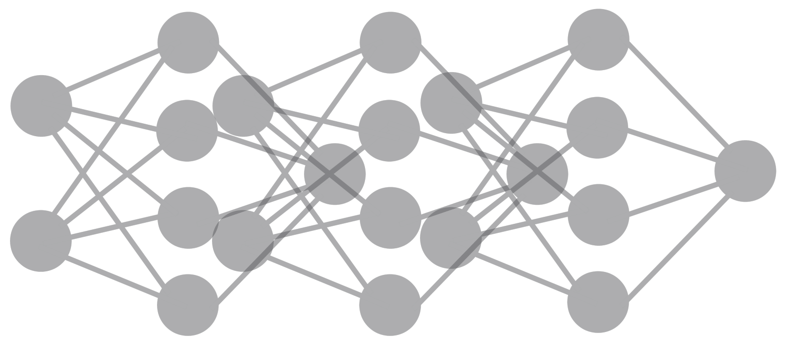
High Computational Cost

Sources?

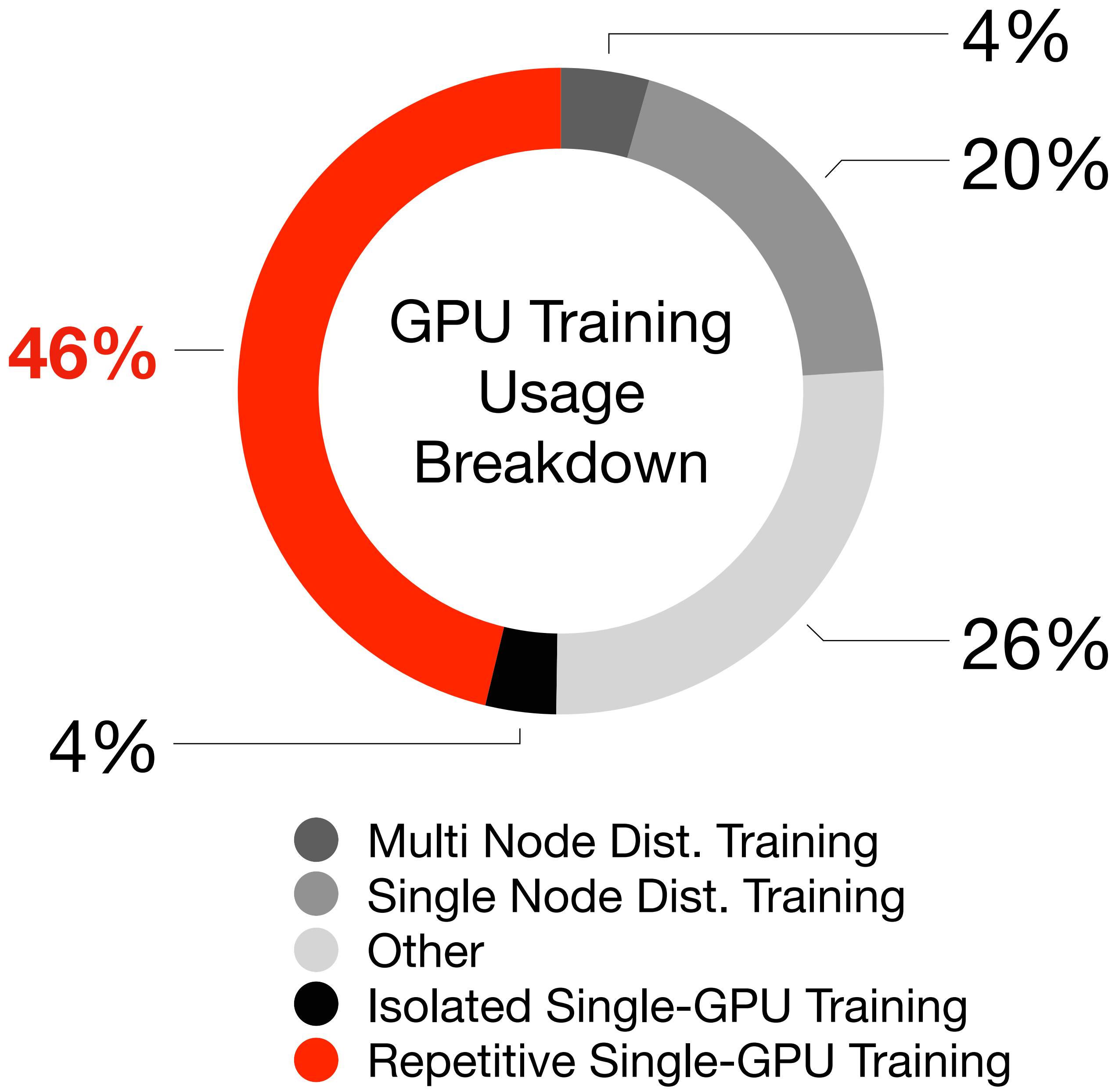
Compute Intensive

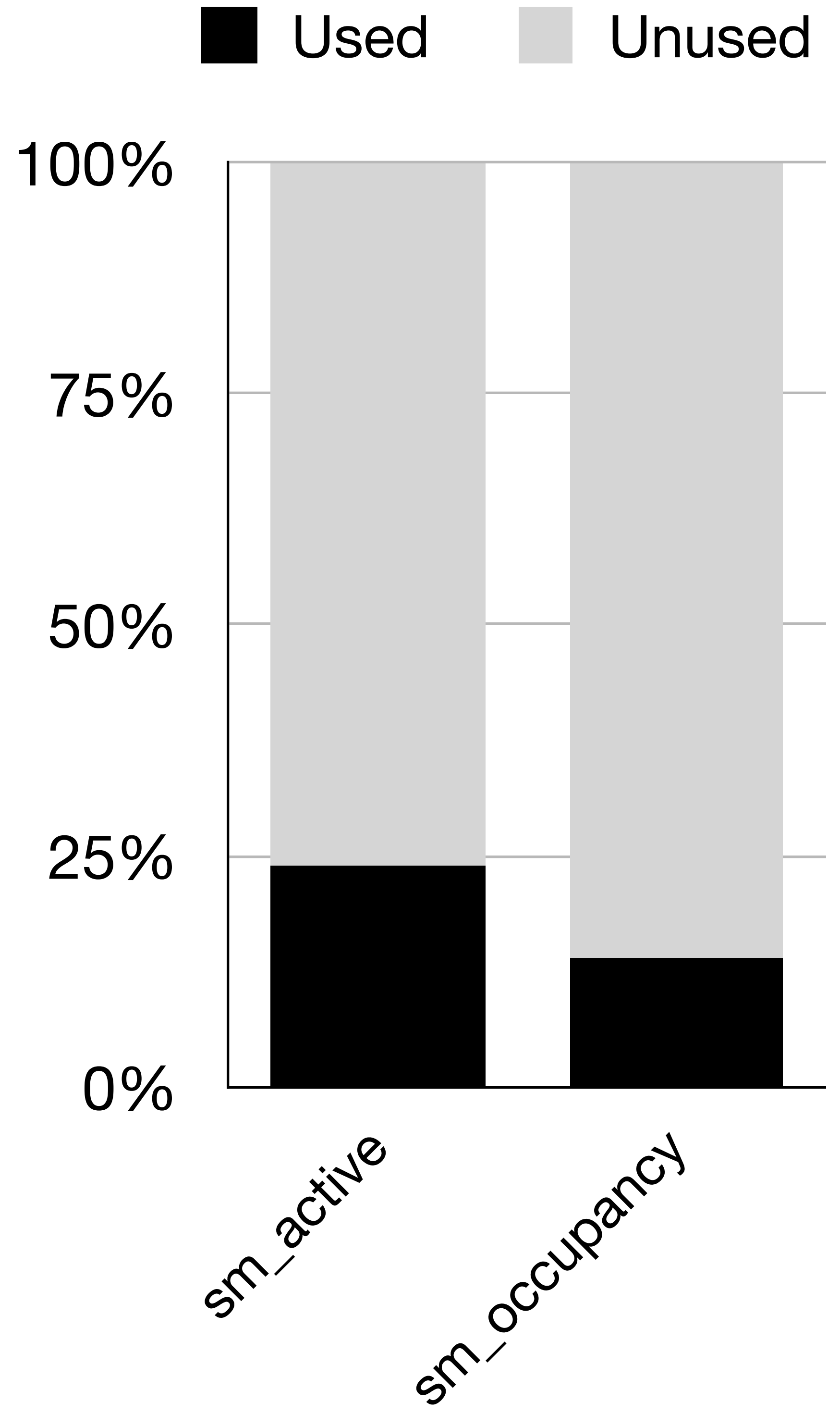


Training Several Networks

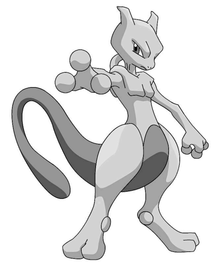


Sub-optimal Hardware Utilization





GPU Performance Counters



Wang et al, Horizontally Fused Training Array (HFTA), MLSys 2021

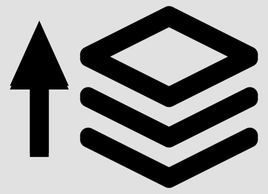
DASlab @ Harvard SEAS

Deep Learning Training

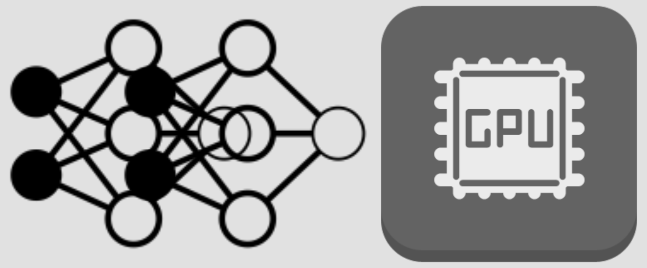
High Computational Cost

Sub-optimal
Hardware Utilization

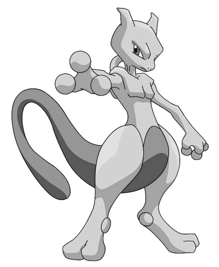
existing h/w utilization techniques

1 

Increasing mini-batch size
(*data parallelism*)

2 

Concurrent-training
(*resource sharing*)



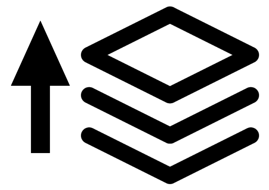
Deep Learning Training

High Computational Cost

Sub-optimal
Hardware Utilization

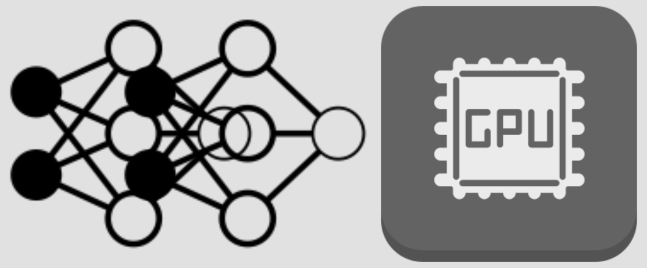
existing h/w utilization techniques

1

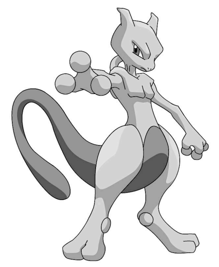


Increasing mini-batch size
(*data parallelism*)

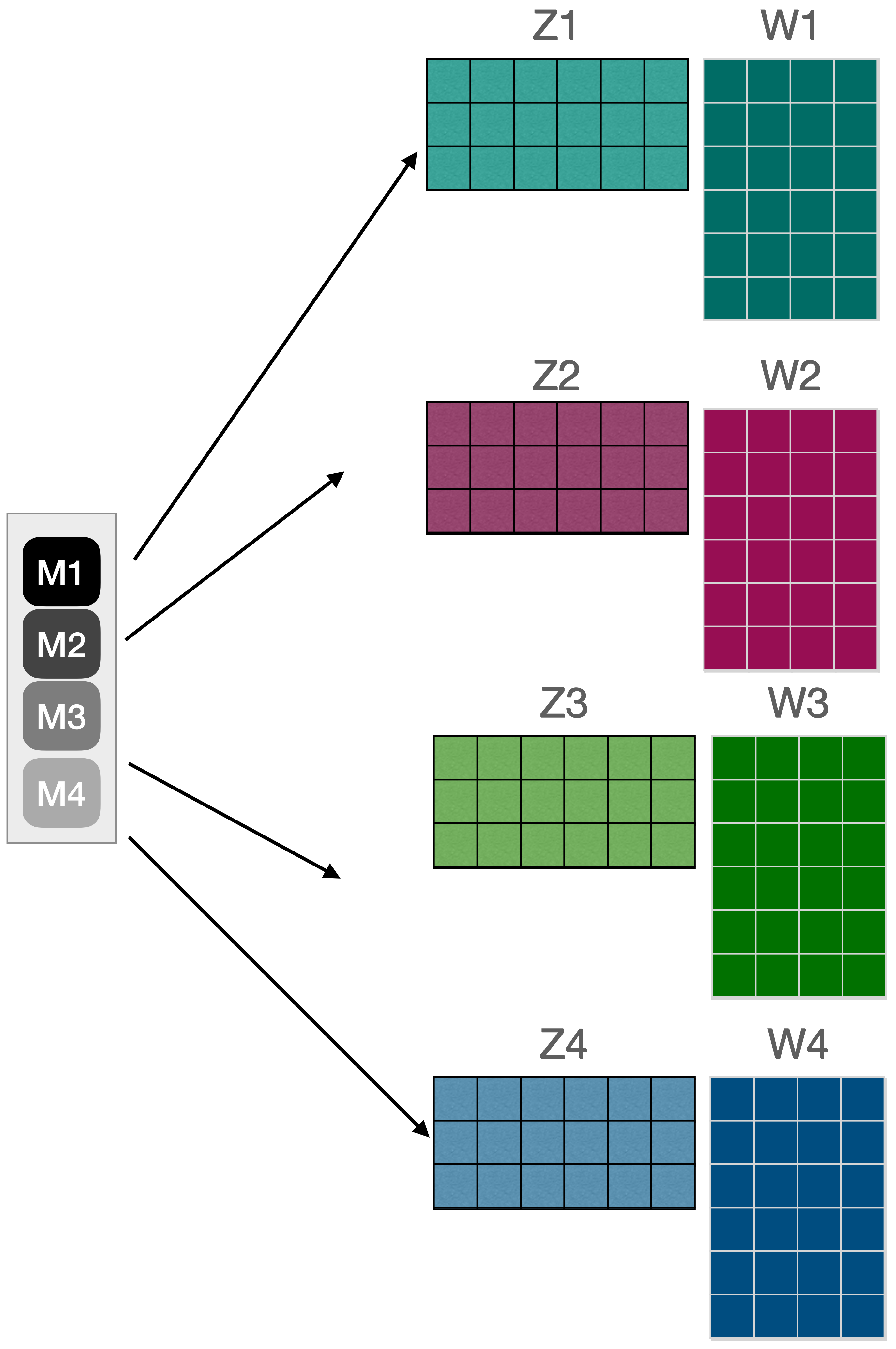
2



Concurrent-training
(*resource sharing*)



Horizontal Fusion



Matrix Multiply Operator

Deep Learning Training

High Computational Cost

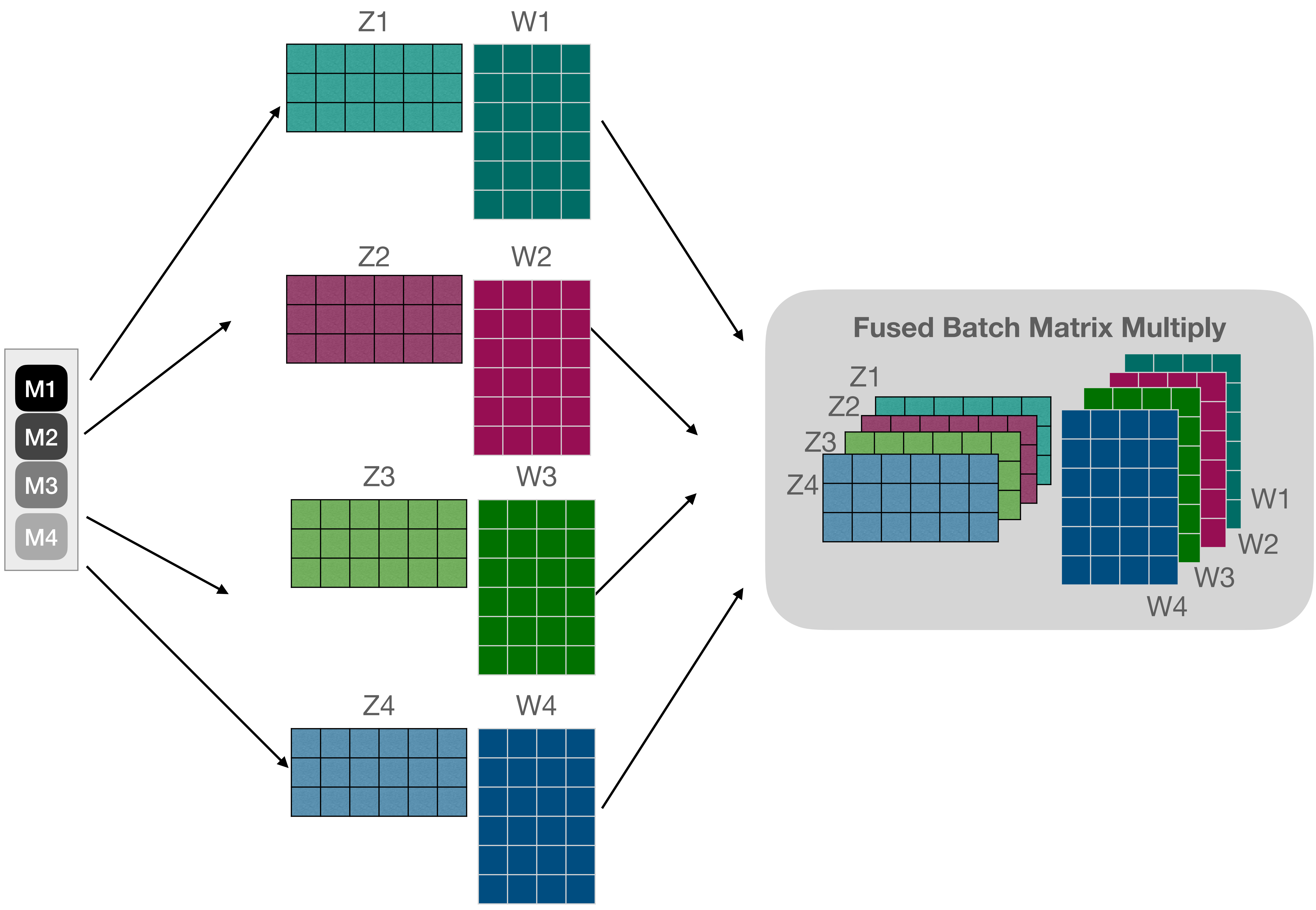
Sub-optimal
Hardware Utilization

existing h/w utilization techniques

1 Increasing mini-batch size
(*data parallelism*)

2 Concurrent-training
(*resource sharing*)

Horizontal Fusion



Matrix Multiply Operator

Wang et al, Horizontally Fused Training Array (HFTA), MLSys 2021

DASlab @ Harvard SEAS

Deep Learning Training

High Computational Cost

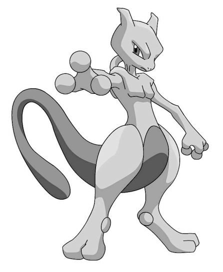
Sub-optimal
Hardware Utilization

existing h/w utilization techniques

1 Increasing mini-batch size (*data parallelism*)

2 Concurrent-training (*resource sharing*)

Added Memory Pressure



DASlab @ Harvard SEAS

Deep Learning Training

High Computational Cost

Sub-optimal
Hardware Utilization

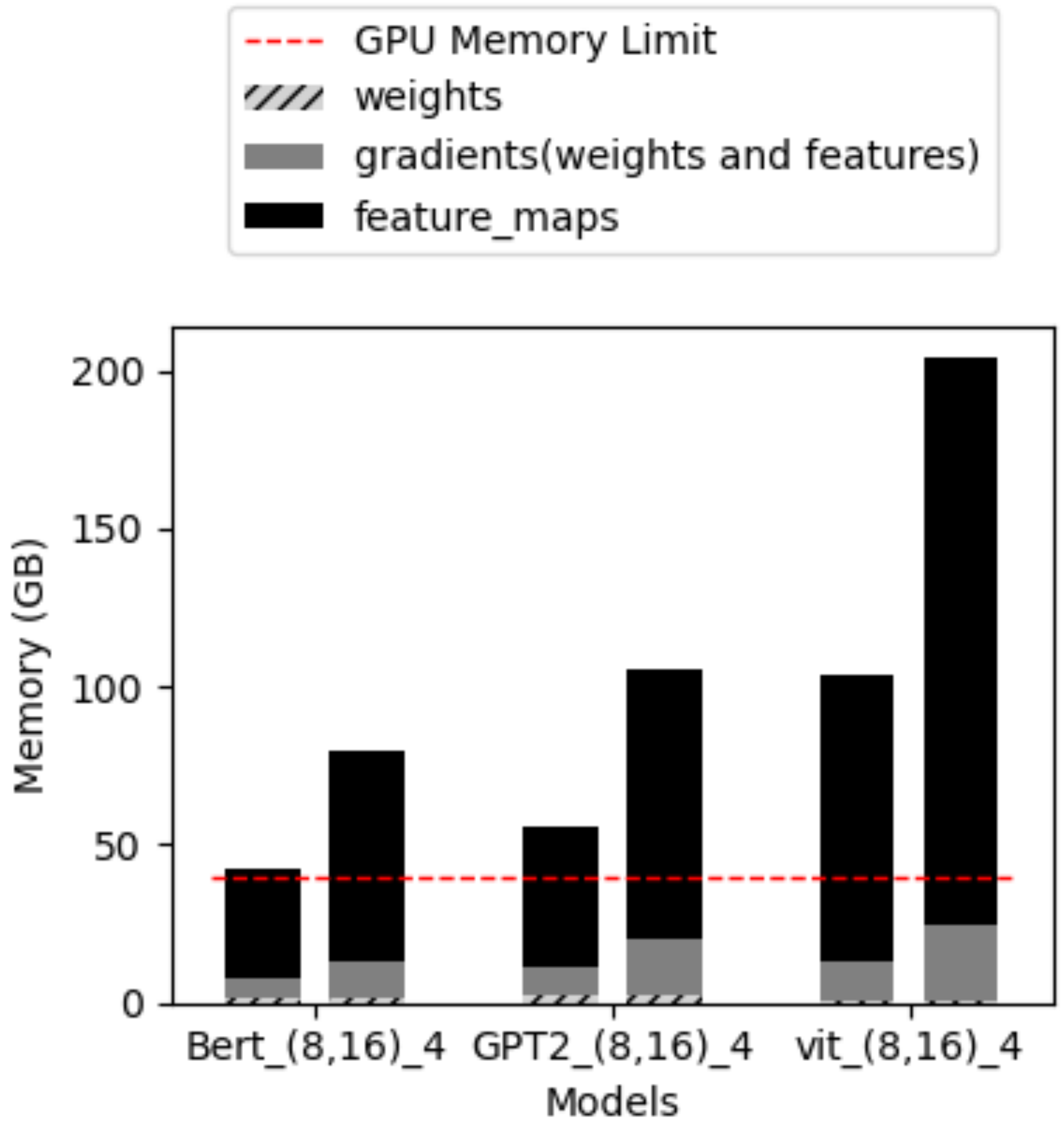
existing h/w utilization techniques

1 Increasing mini-batch size
(*data parallelism*)

2 Concurrent-training
(*resource sharing*)

Memory Oversubscription -> Limits Scaling

- A Large model sizes
- B Greater number of models
- C Large training memory footprint
- D Limited GPU memory capacity



@ Harvard SEAS

Deep Learning Training

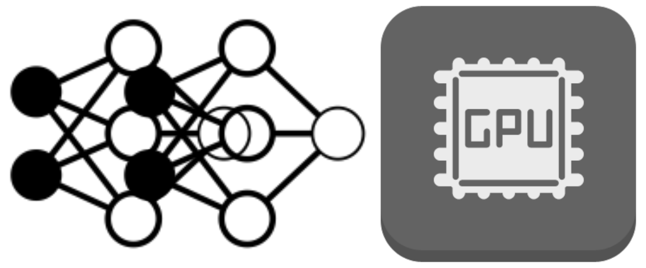
High Computational Cost

Sub-optimal
Hardware Utilization

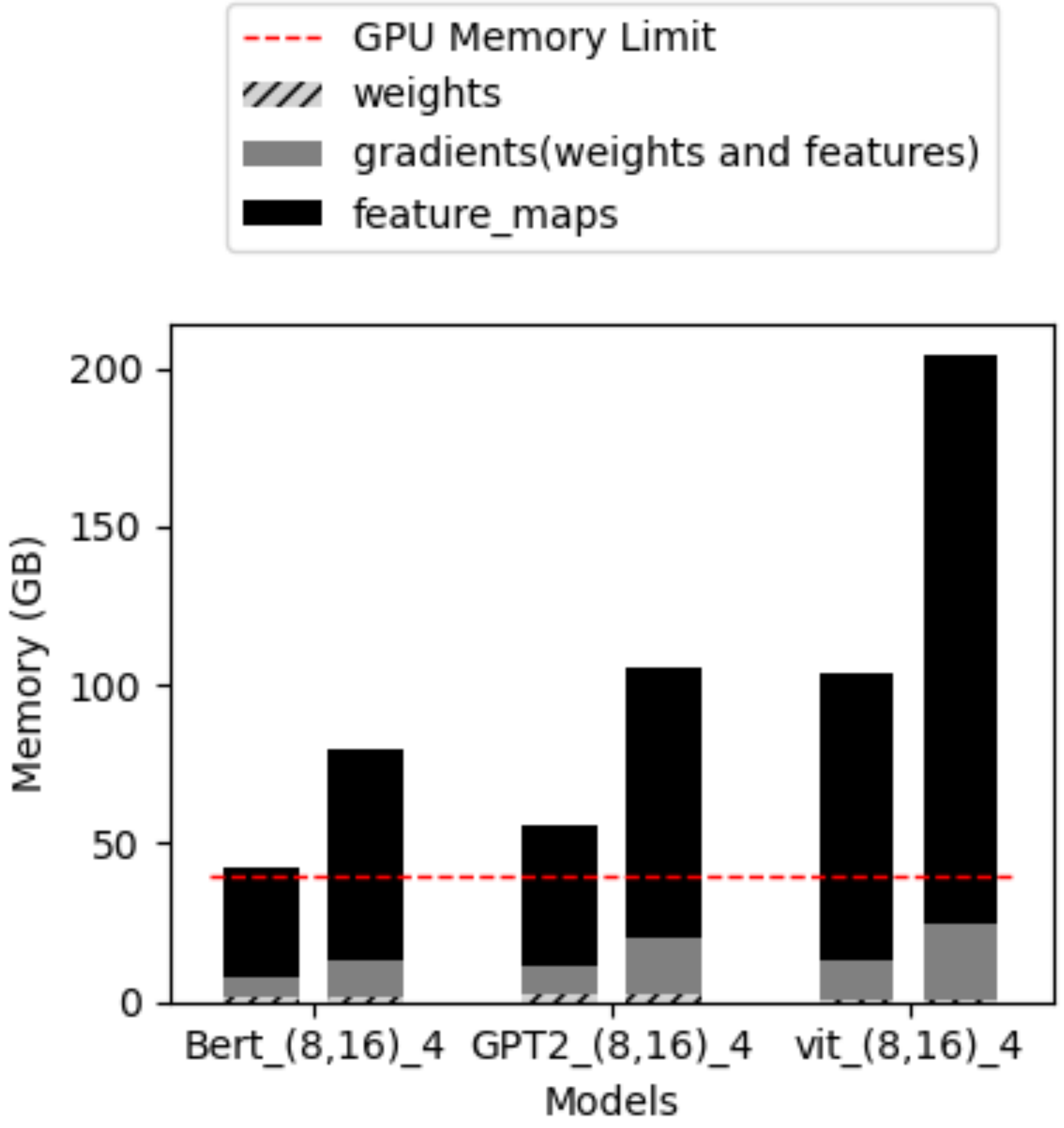
existing h/w utilization techniques

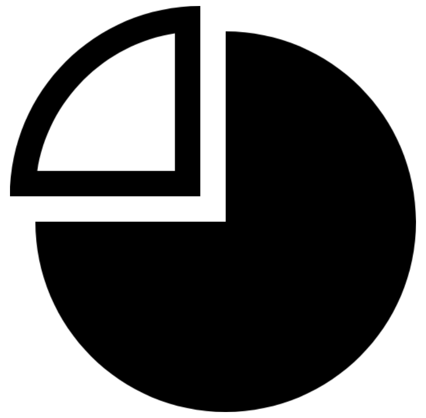
1 Increasing mini-batch size (*data parallelism*)

2 Concurrent-training (*resource sharing*)

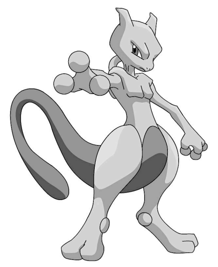


Addressing Memory Oversubscription





Feature Maps have the largest and most significant share in memory consumption.



DASlab @ Harvard SEAS

Deep Learning Training

↓

High Computational Cost

↓

Sub-optimal Hardware Utilization

↓

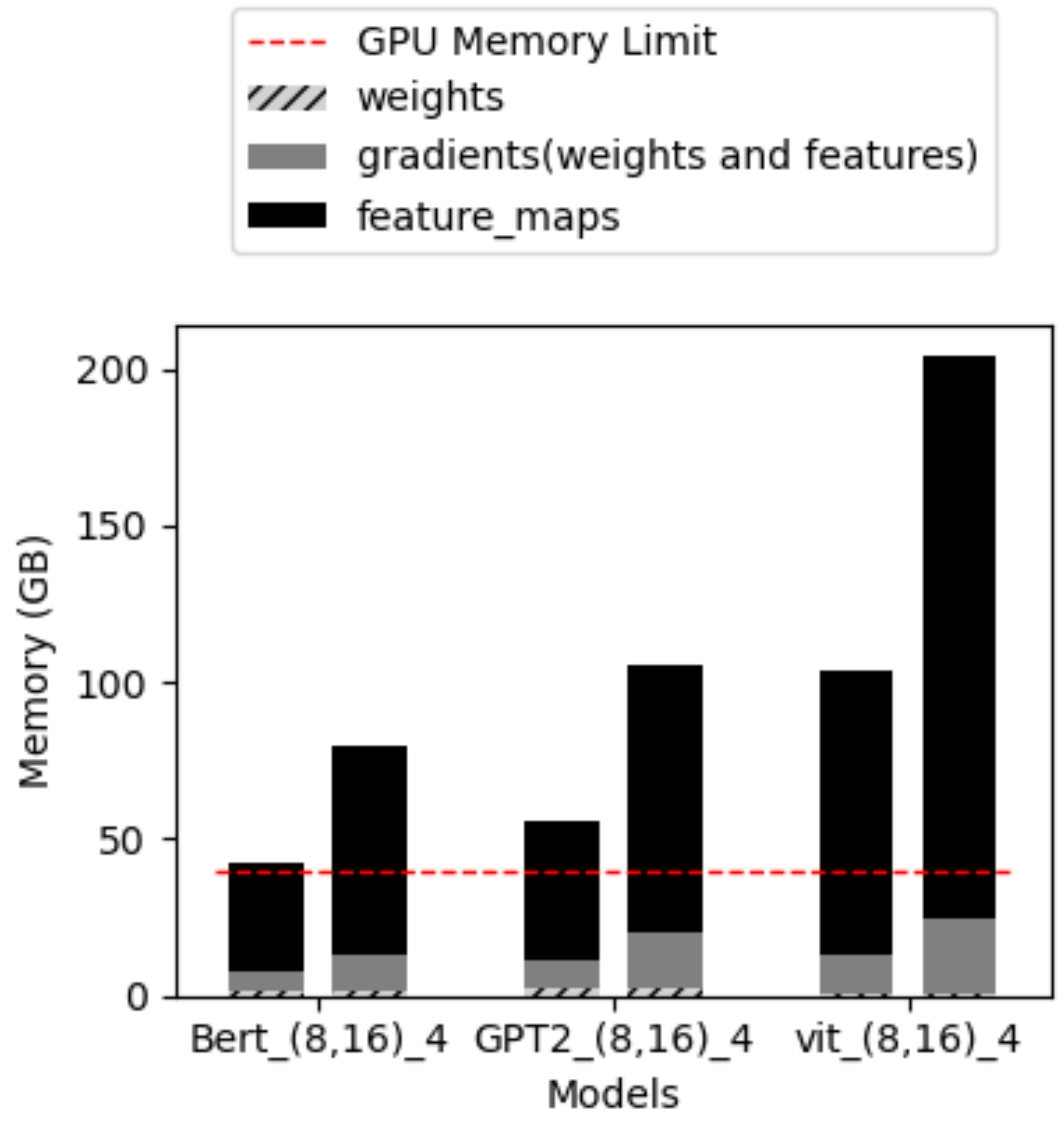
existing h/w utilization techniques

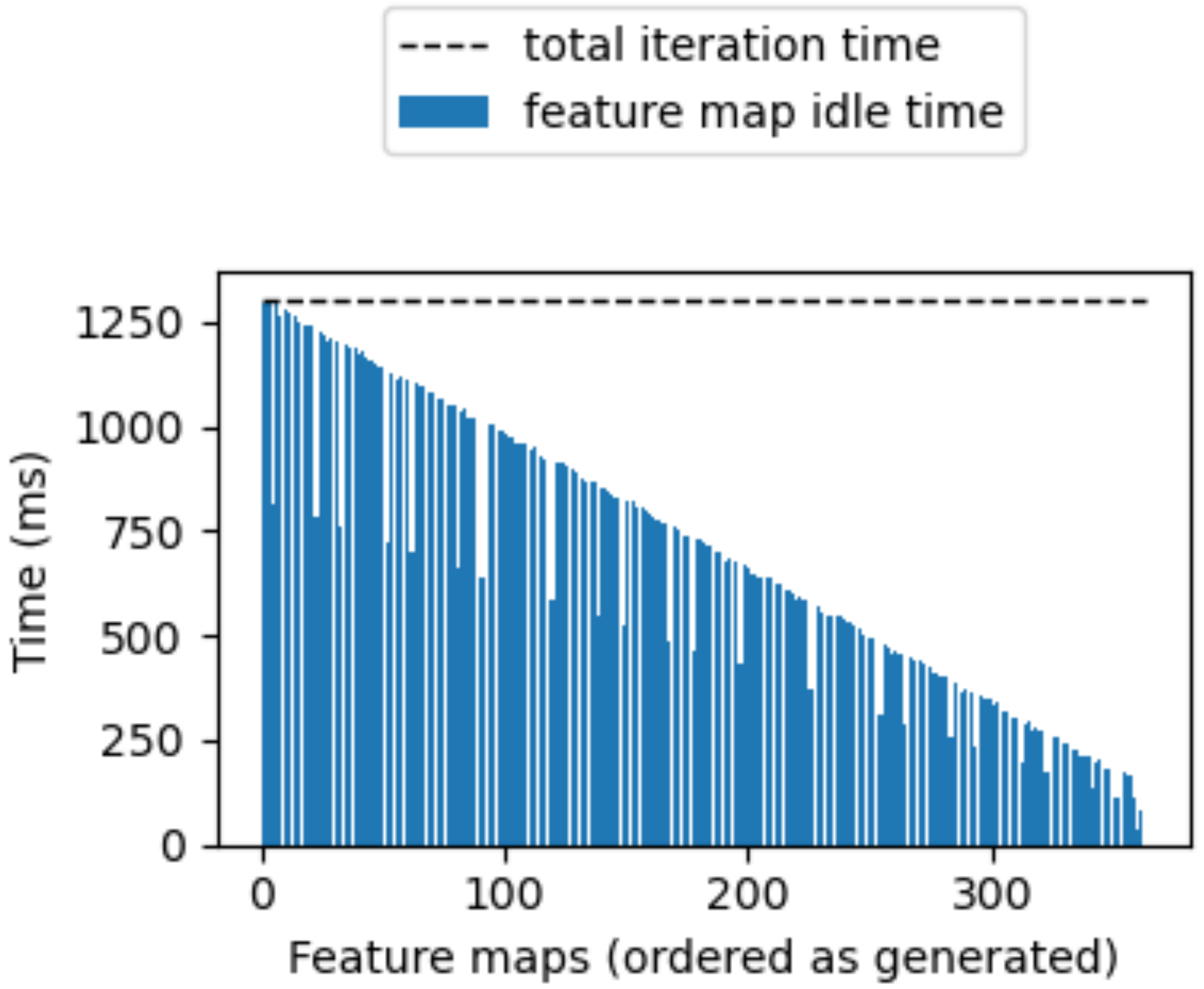
1 Increasing mini-batch size (*data parallelism*)

2 Concurrent-training (*resource sharing*)

Addressing Memory Oversubscription

Feature Maps have the largest and most significant share in memory consumption.





Feature Maps lie idle in GPU memory between forward and backward pass

Deep Learning Training

↓

High Computational Cost

↓

Sub-optimal
Hardware Utilization

↓

existing h/w utilization techniques

1. Increasing mini-batch size (*data parallelism*)

2. Concurrent-training (*resource sharing*)

Addressing Memory Oversubscription

Recomputation

Discard feature maps after use in forward pass

Recompute when needed during backward pass

Tradeoff Compute for Memory

Superfluous Compute Overhead

Deep Learning Training

High Computational Cost

Sub-optimal
Hardware Utilization

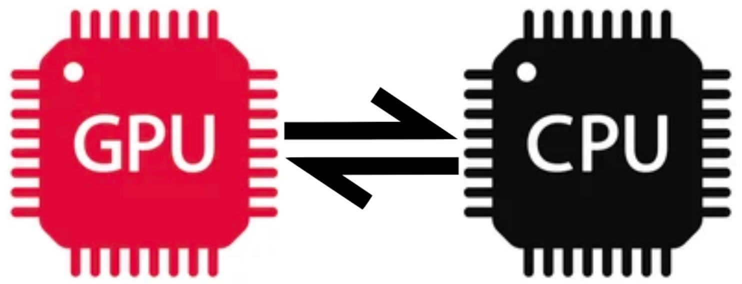
existing h/w utilization techniques

1 Increasing mini-batch size (*data parallelism*)

2 Concurrent-training (*resource sharing*)

Addressing Memory Oversubscription

Swapping



Offload feature maps after use in forward pass to larger host memory

Fetch feature maps to GPU memory when needed during backward pass

Tradeoff stall time for Memory

Stalling Overhead

Deep Learning Training

High Computational Cost

Sub-optimal
Hardware Utilization

existing h/w utilization techniques

1

Increasing mini-batch size
(*data parallelism*)

2

Concurrent-training
(*resource sharing*)

Addressing Memory Oversubscription

Recomputation

Swapping

GPU ⇌ CPU

Direct Application

Multi-Model Training

Significant Overheads

50% Slowdown

Problem Space

Deep Learning Training

↓

High Computational Cost

↓

Sub-optimal Hardware Utilization

↓

Addressing Underutilization

Mini-batch size

Concurrent-training

Number of Models | **Model Size** | **Mini-batch size**

Do not scale

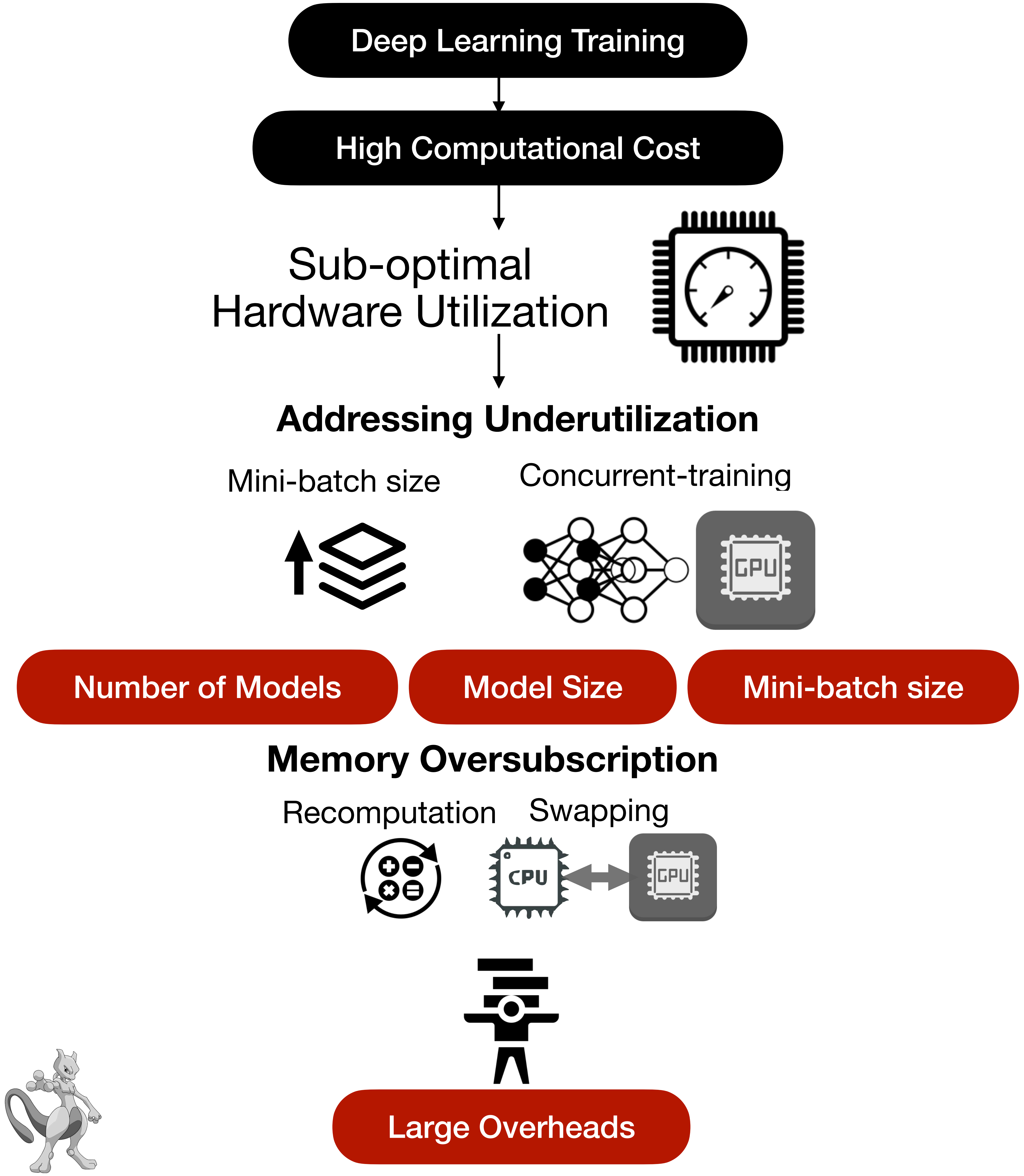
Memory Oversubscription

Recomputation

Swapping

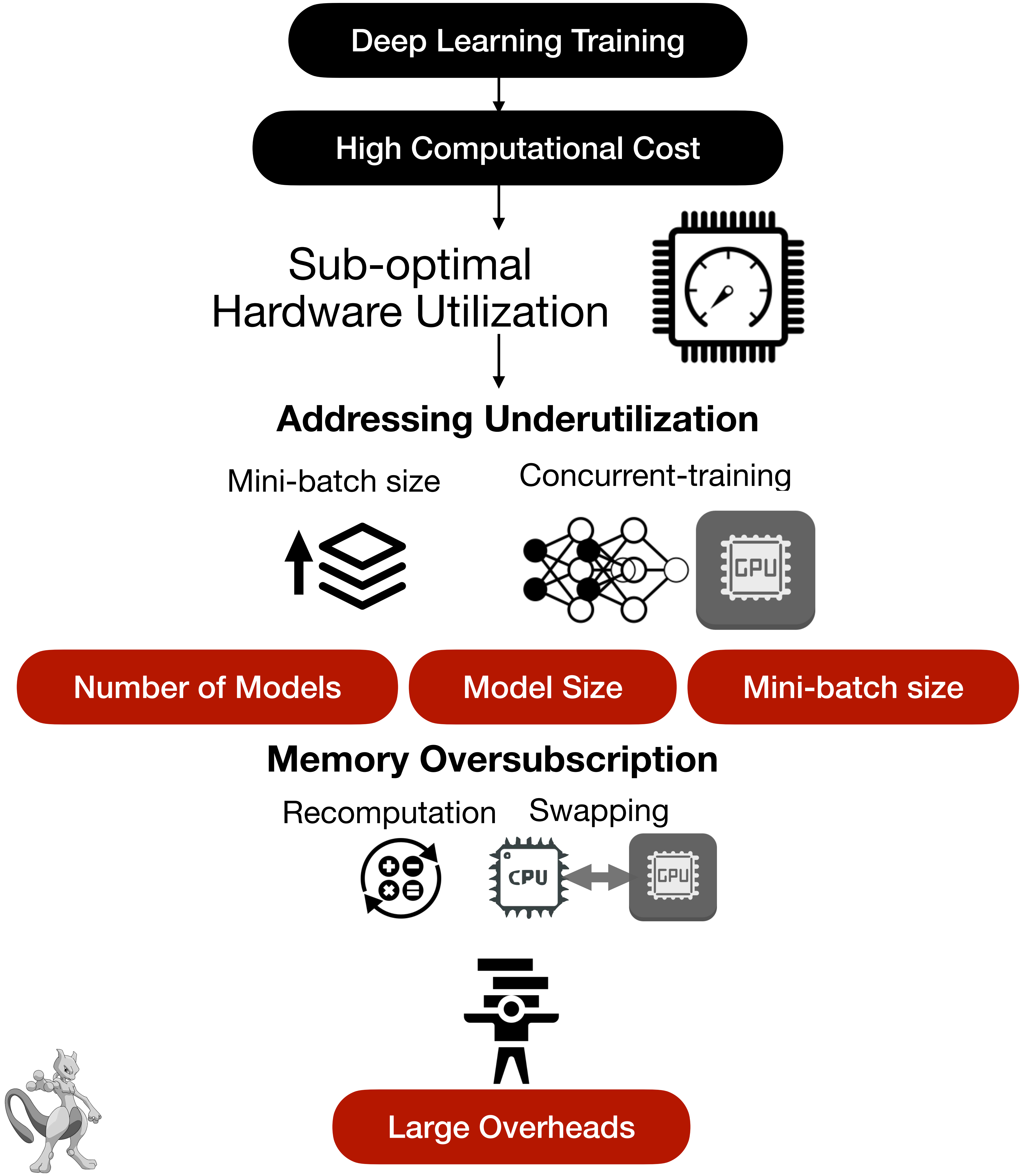
Large Overheads

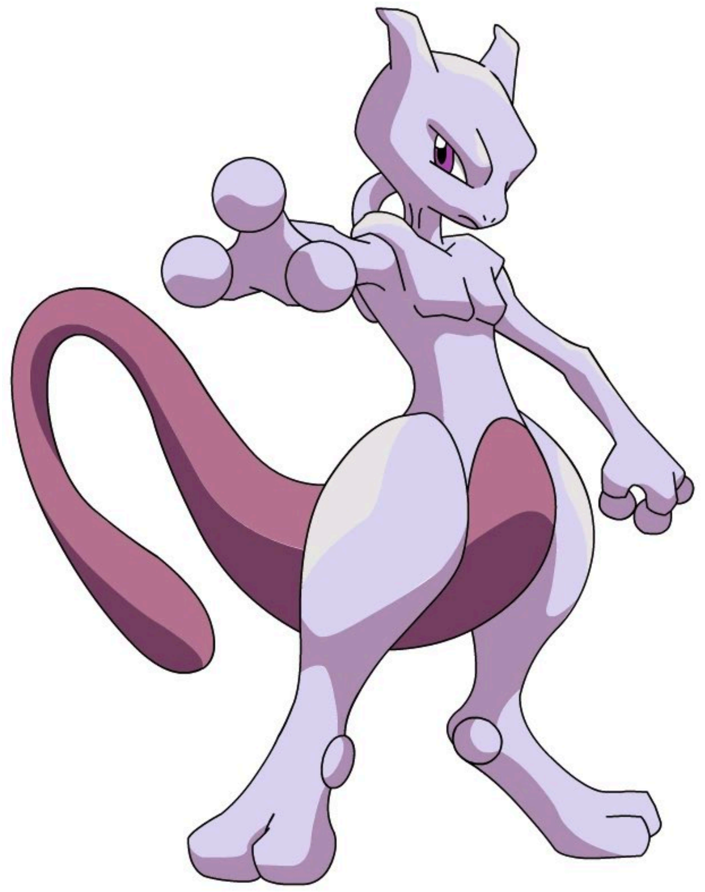
Problem Definition



How to scale concurrent multi-model training as models grow and peak memory requirement surpasses the available GPU memory capacity?

Solution





μ-TWO

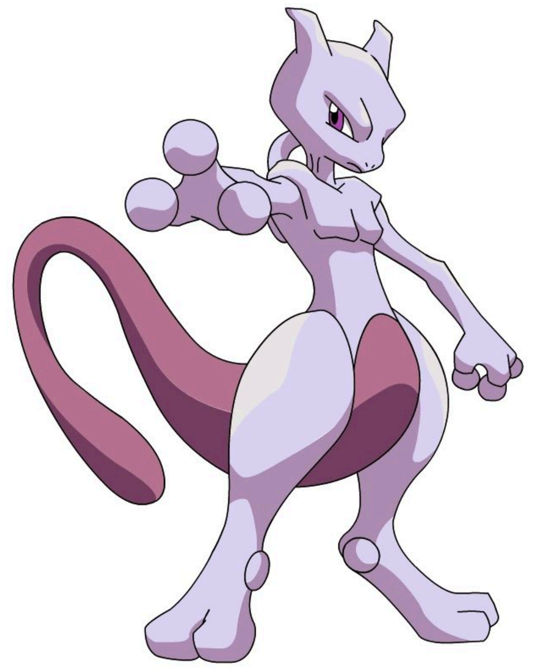
Multi-model training compiler

3x latency speed-up

3-5x more models

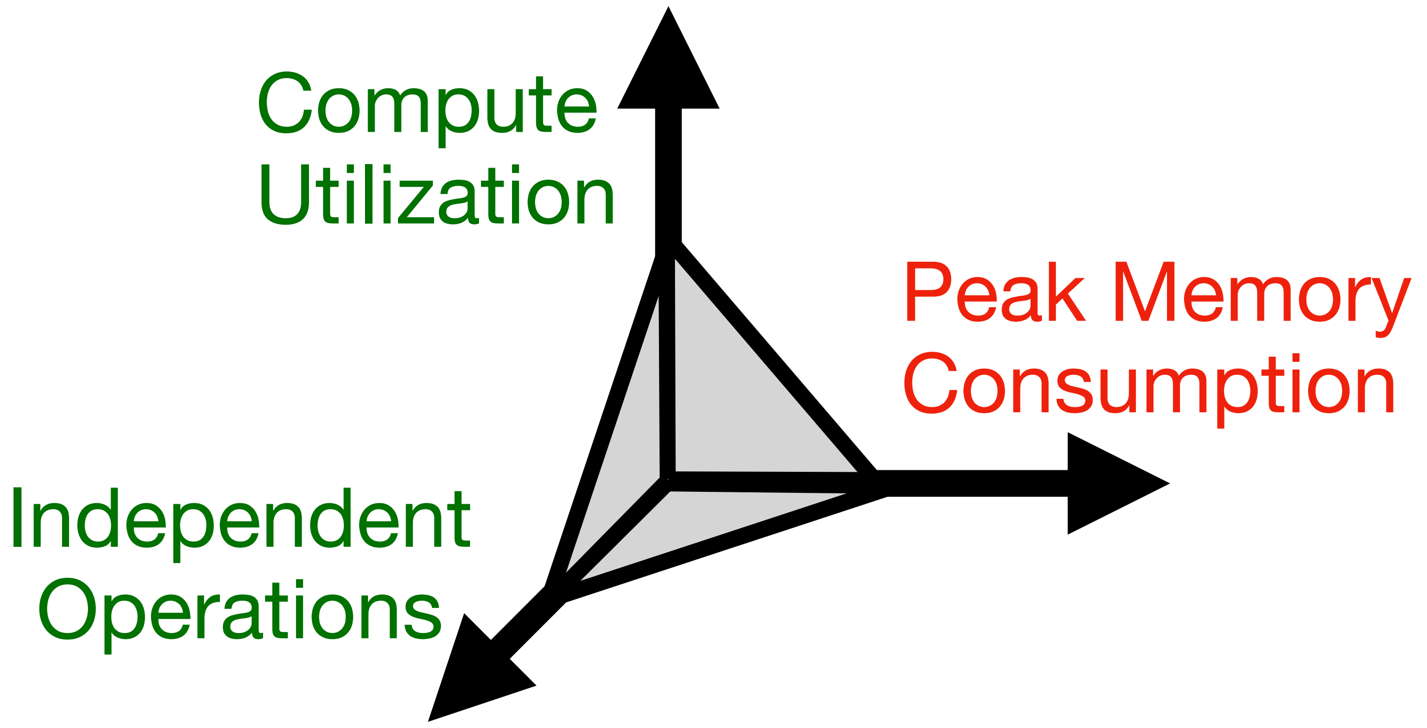
6x the GPU memory size

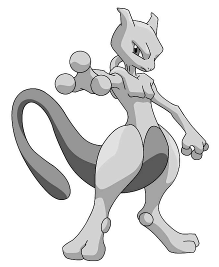
DASlab @ Harvard SEAS

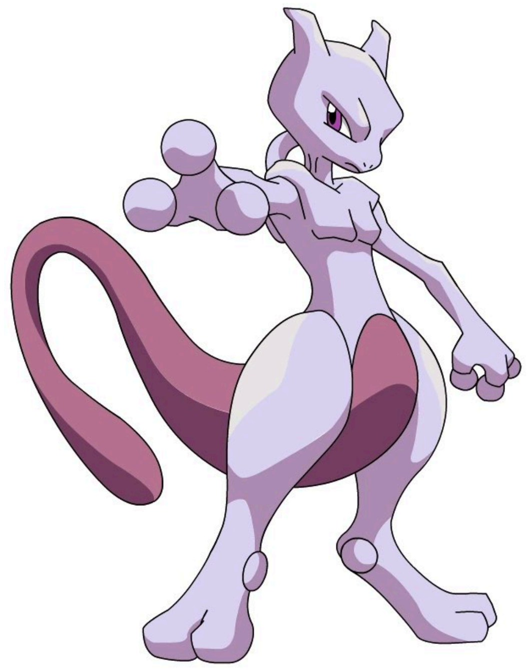


μ-TWO : Multi-Model Training with Orchestration

Latency of a multi-model training schedule

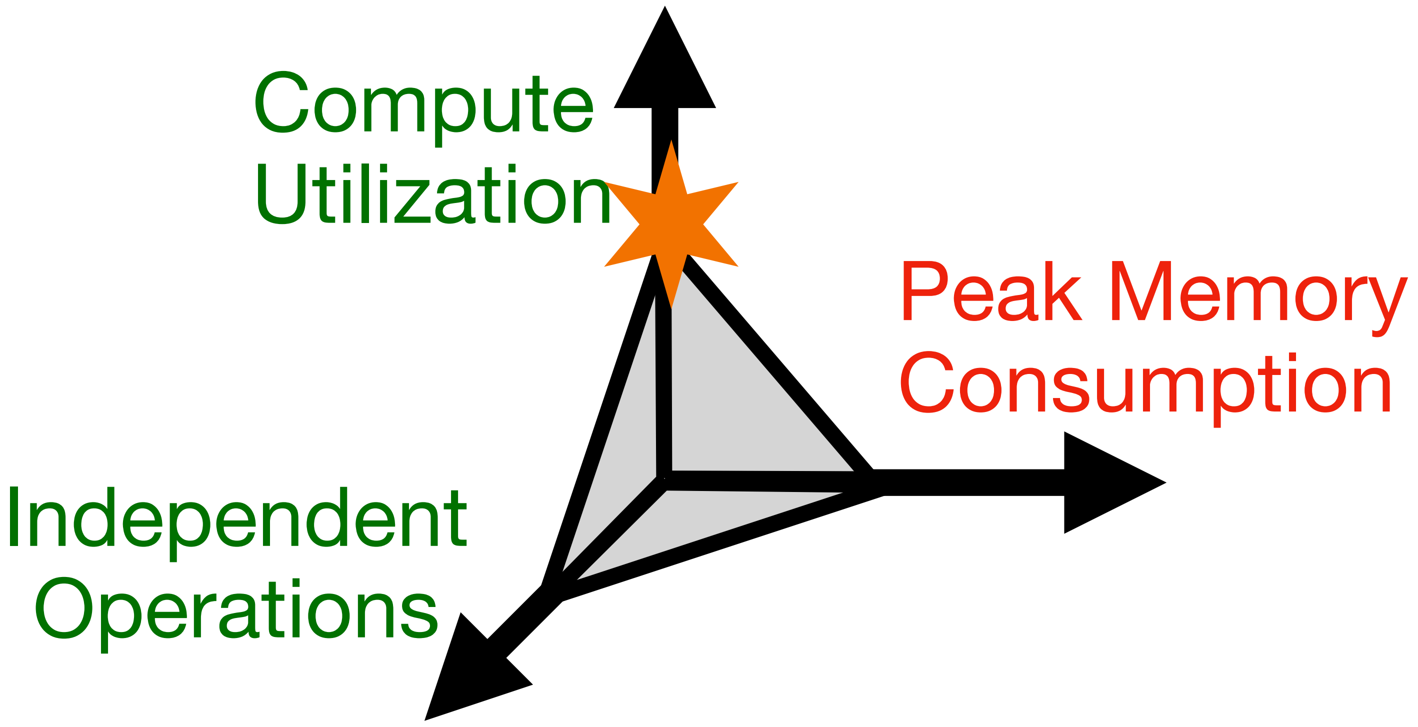




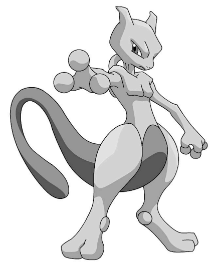


μ-TWO : Multi-Model Training with Orchestration

Latency of a multi-model training schedule



Efficiently navigates the trade-off for a given set of models and target GPU and finds a sweet spot

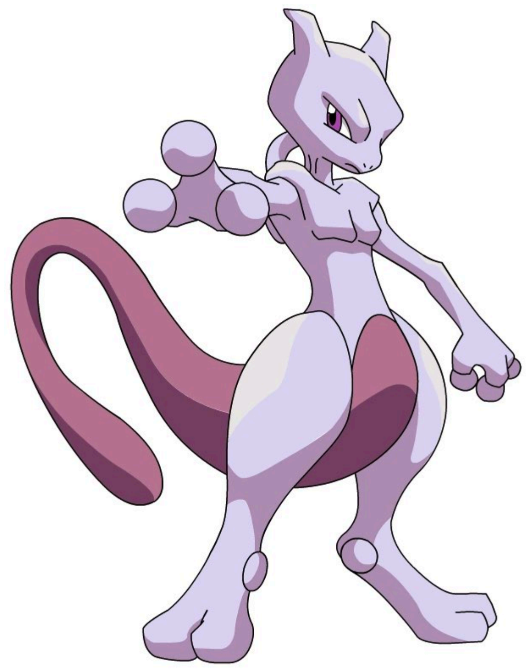


μ-TWO : Multi-Model Training with Orchestration

For a given set of models and target GPU

Lightweight Profiling

Static Analysis

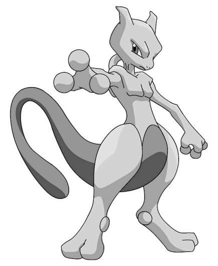


Operators to fuse
saturate compute

Tailored strategy

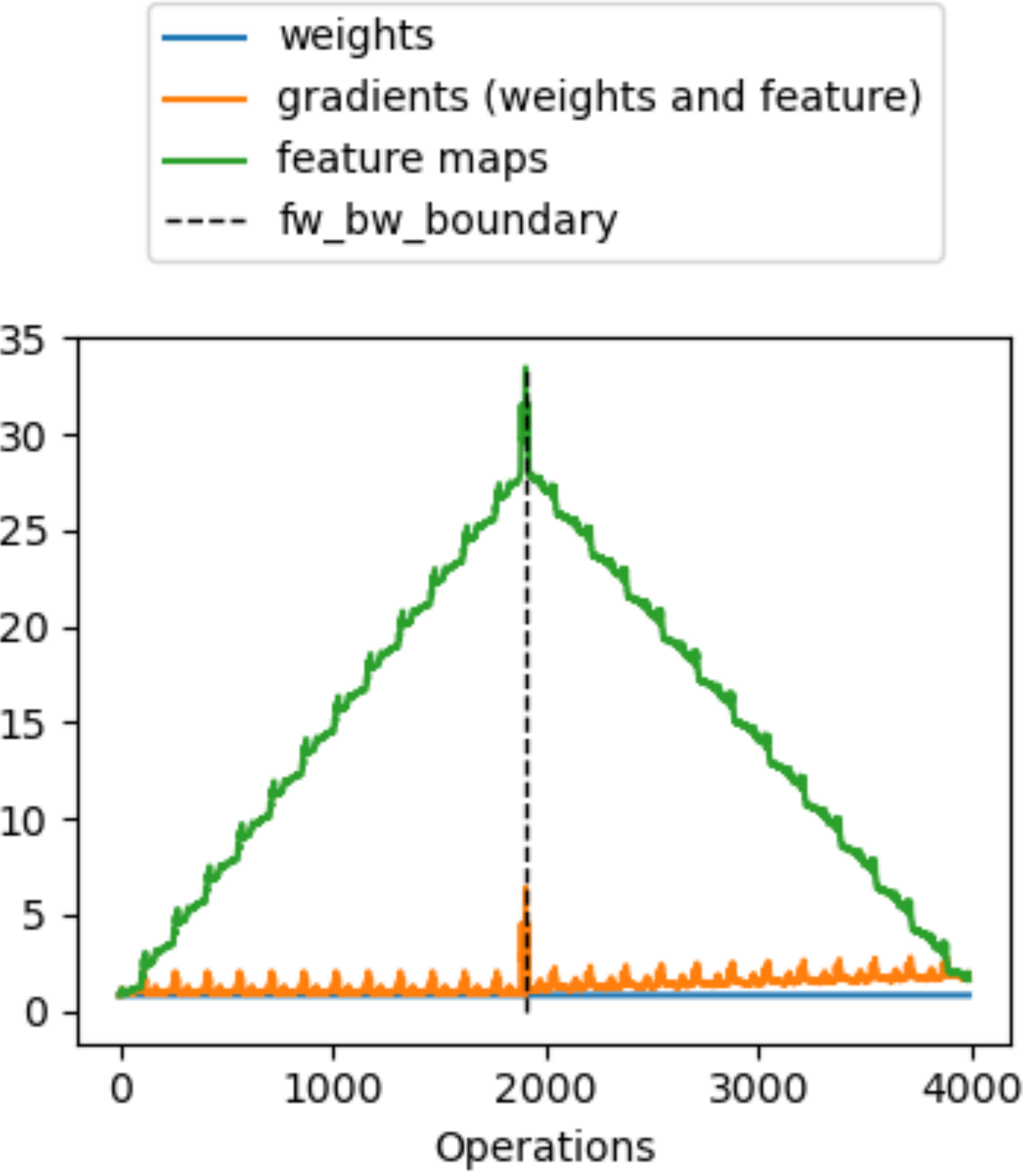
Swap/ recompute
optimize memory usage

Maximally overlapped swapping
eliminate stalling

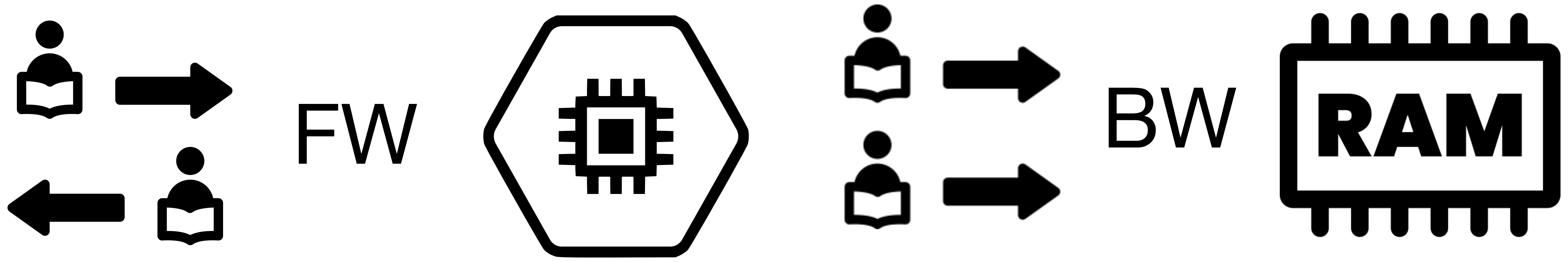


DASlab @ Harvard SEAS

Insights for μ-TWO



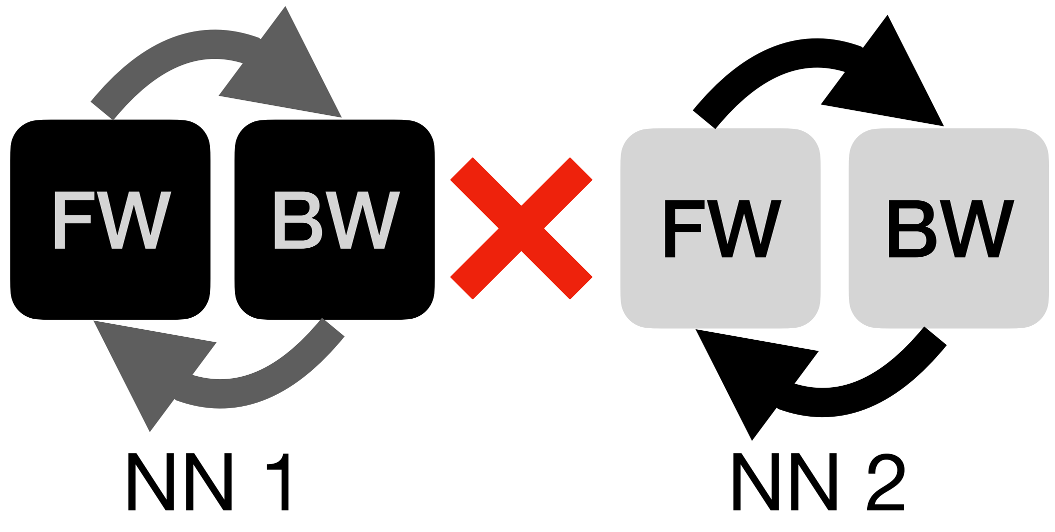
1 CHANGED NATURE OF FORWARD AND BACKWARD PASS DURING SWAPPING



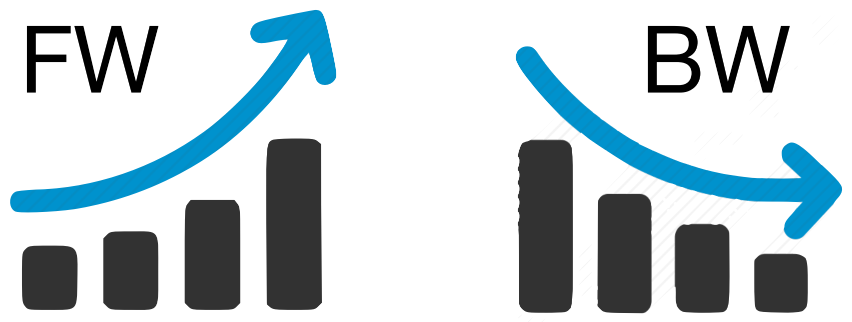
Not Sensitive to IO

Sensitive to IO

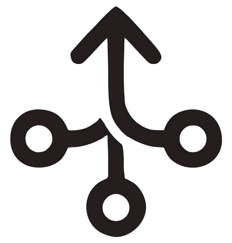
2 NO INTER-NETWORK DEPENDENCY

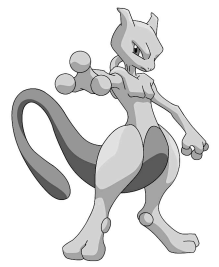


3 PEAK MEMORY OCCUPANCY



4 HORIZONTAL FUSION IS ESSENTIAL FOR COMPUTE UTILIZATION

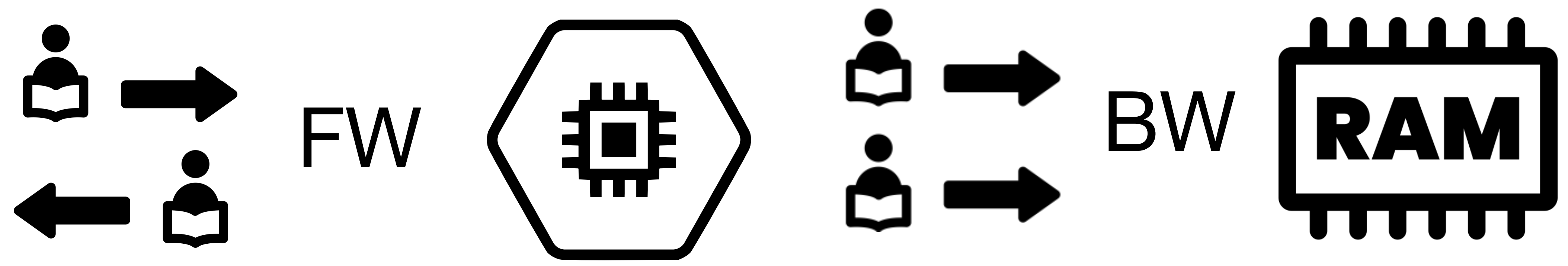




DASlab @ Harvard SEAS

Design Implications for μ-TWO

1 CHANGED NATURE OF FORWARD AND BACKWARD PASS DURING SWAPPING

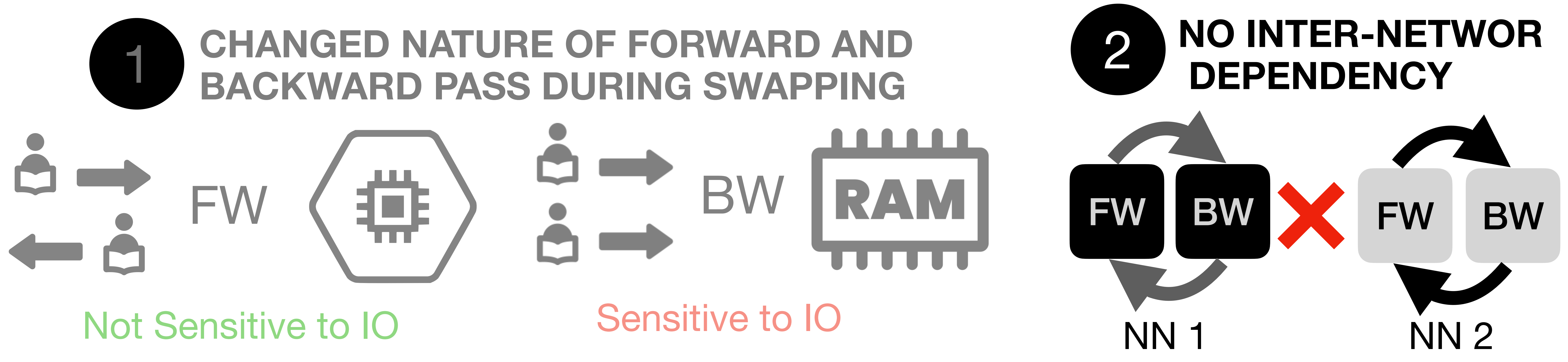


Not Sensitive to IO

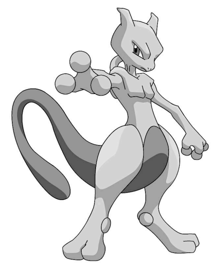
Sensitive to IO

1. Conservatively schedule Swapping, overlap with compute as much as possible

Design Implications for μ-TWO

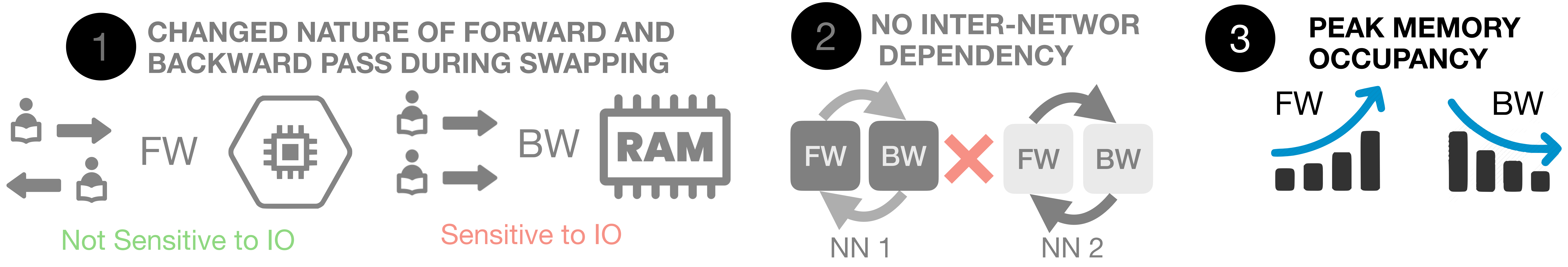


1. Conservatively schedule Swapping, overlap with compute as much as possible
2. Operations from one model can be used to overlap IO operations of other models

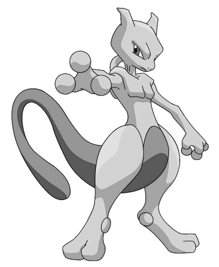


DASlab @ Harvard SEAS

Design Implications for μ-TWO

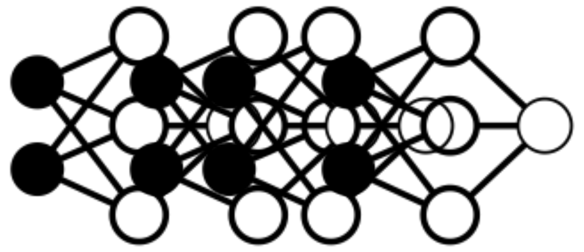


1. Conservatively schedule Swapping, overlap with compute as much as possible
2. Operations from one model can be used to overlap IO operations of other models
3. Only forward pass operations should be used for overlapping backward pass IO operations



DASlab @ Harvard SEAS

Design Trade-Offs for μ-TWO



Multiple Models

M1 M2 M3 M4 M5 M6 M7 M8

FW BW

Monolithic inseparable forward and backward pass operations (Minimal opportunity to overlap)

High Compute Utilization due to maximal fusion

High Peak memory consumption due to fusion

M1 M2 M3 M4 M5 M6 M7 M8

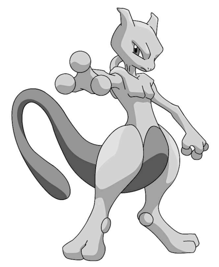
FW1 FW2 FW3 FW4 FW5 FW6 FW7 FW8

BW1 BW2 BW3 BW4 BW5 BW6 BW7 BW8

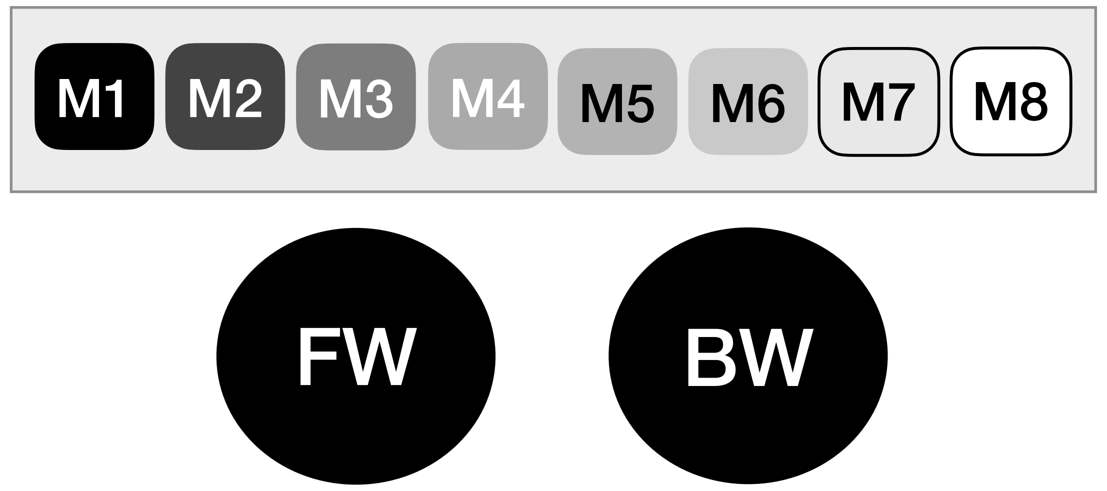
Separate forward and backward pass operations (Maximum opportunity to overlap)

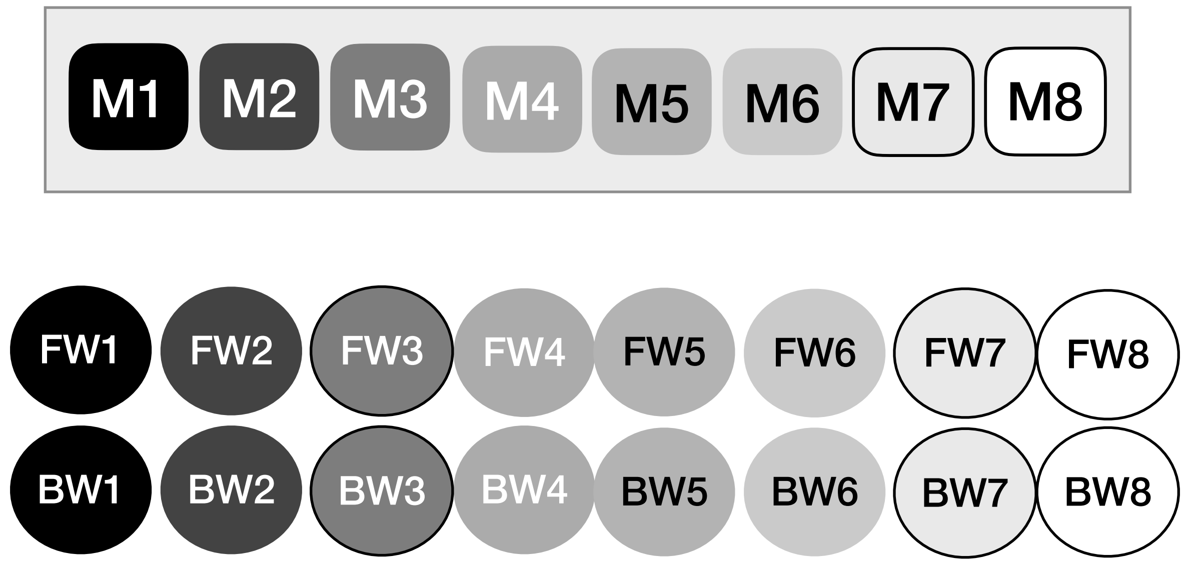
Low Compute Utilization due to no fusion

Lowest Peak memory consumption due to absence of fusion

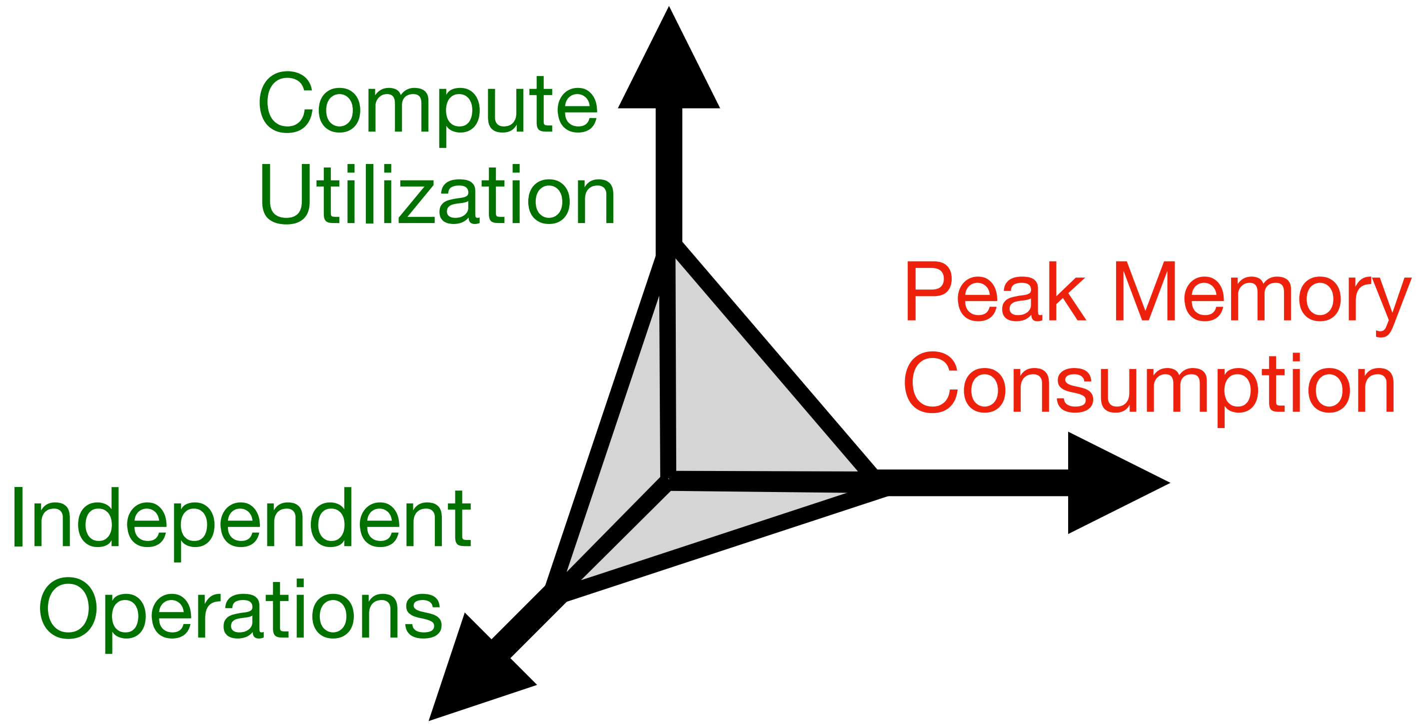


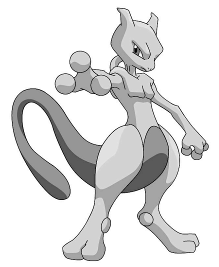
Design Trade-Offs for μ-TWO



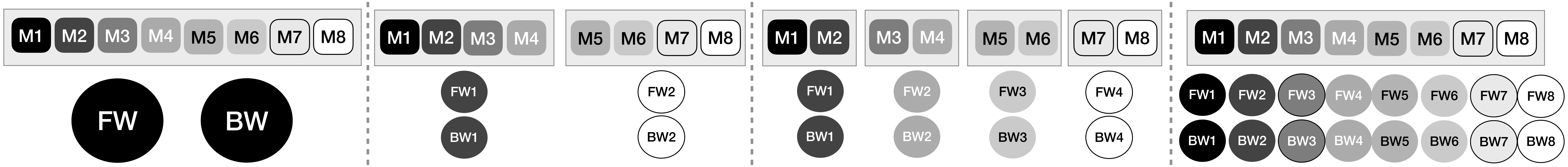


Peak Memory Consumption	Highest	Lowest
Compute Utilization	Highest	Lowest
Overlap Opportunity	Lowest	Highest

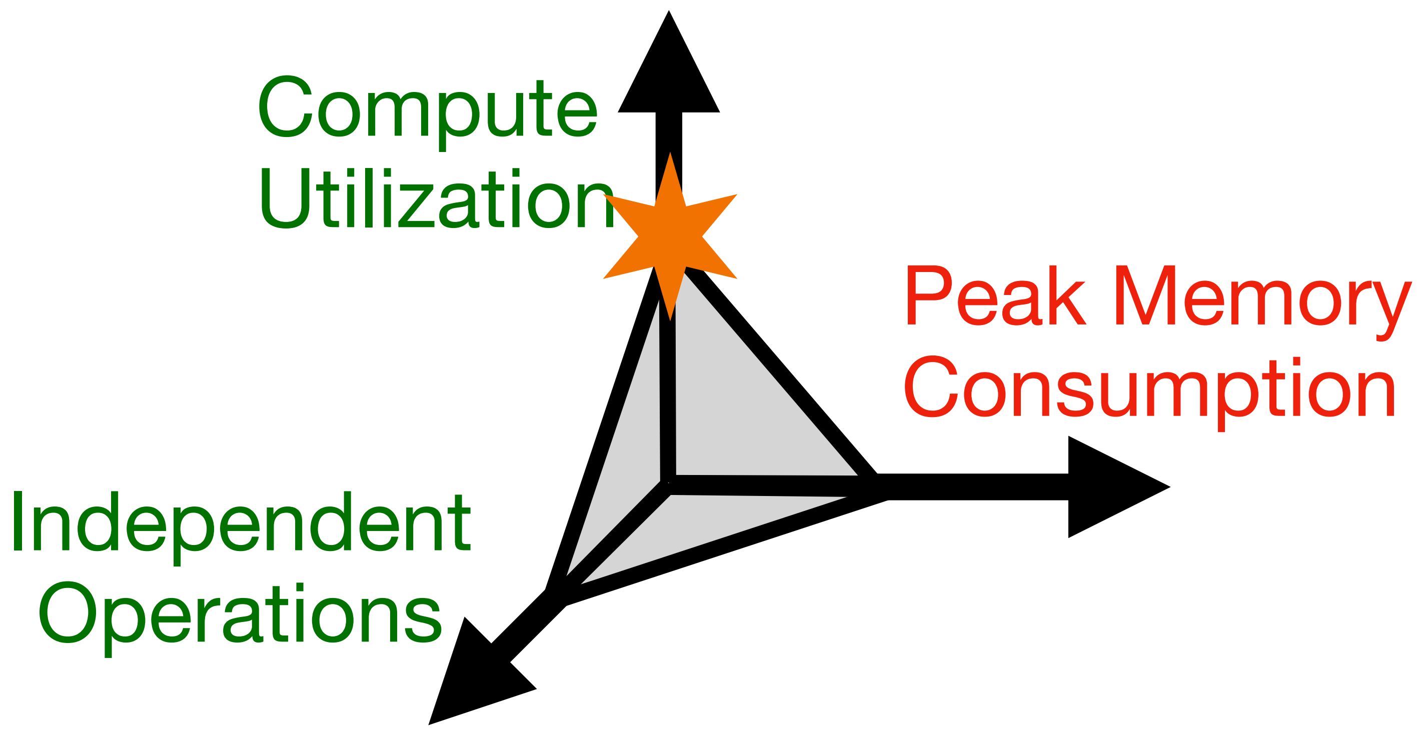




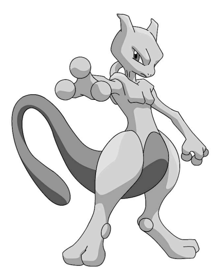
Design Trade-Offs for μ-TWO



Peak Memory Consumption	Highest	High	Low	Lowest
Compute Utilization	Highest	High	Moderate	Lowest
Overlap Opportunity	Lowest	Moderate	High	Highest

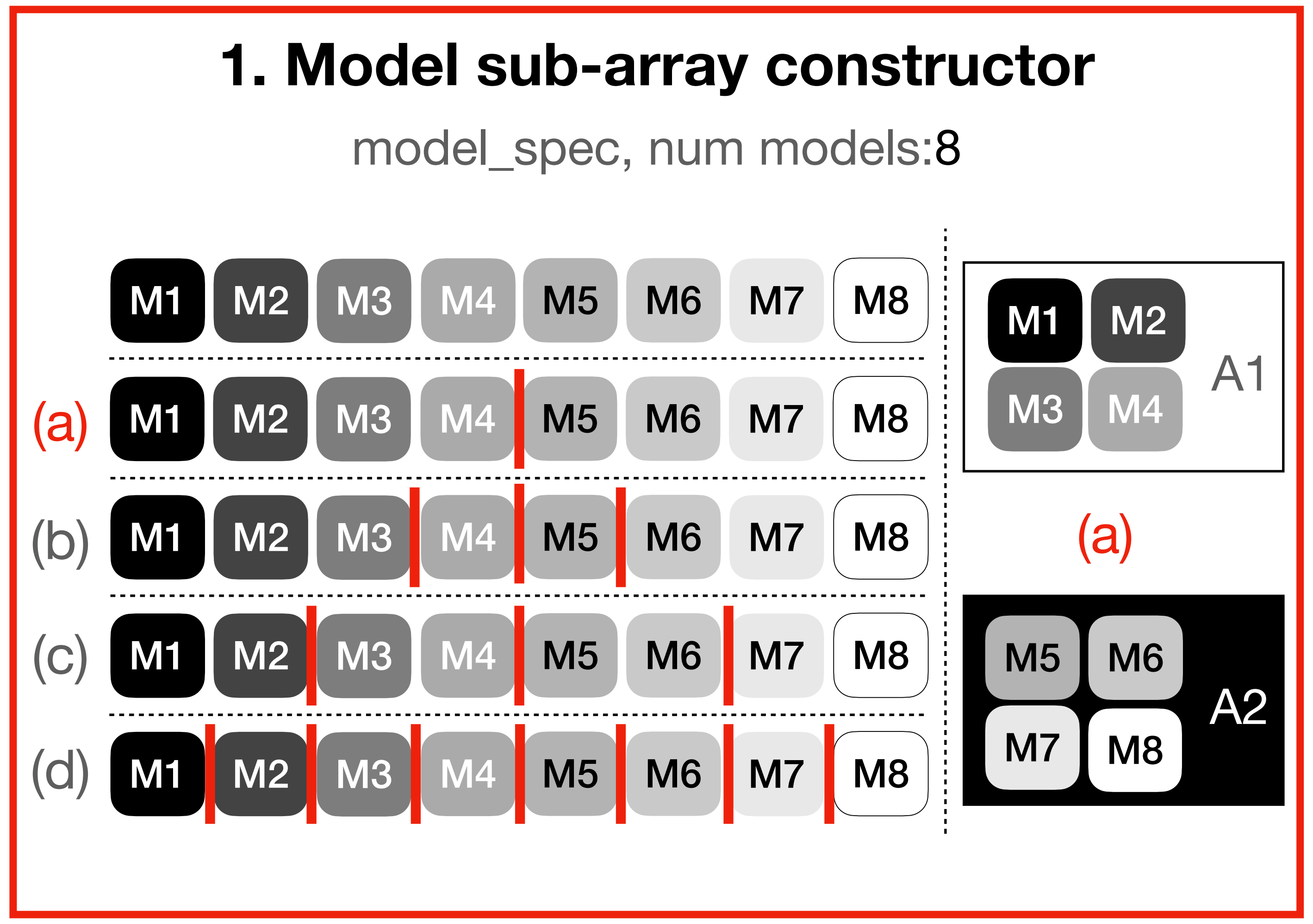


Use fusion granularity to navigate the trade-off



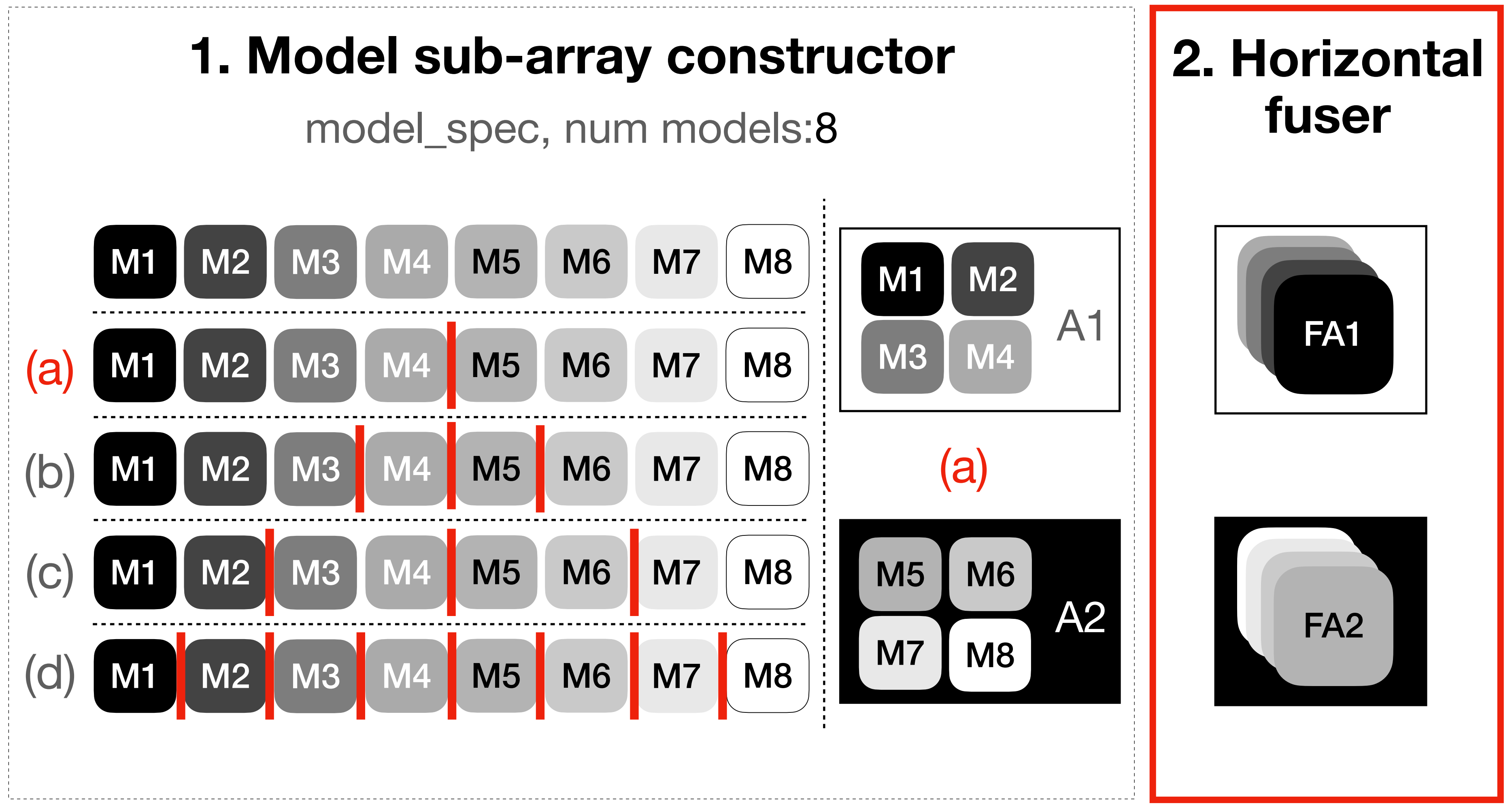
DASlab @ Harvard SEAS

μ-TWO System Overview



Enumerate possible sub-array partitions of models

μ-TWO System Overview



Horizontally fuse models within each sub-array

Models in each sub-array must have same architecture to fuse kernels

Can have different hyper parameters

Loss function

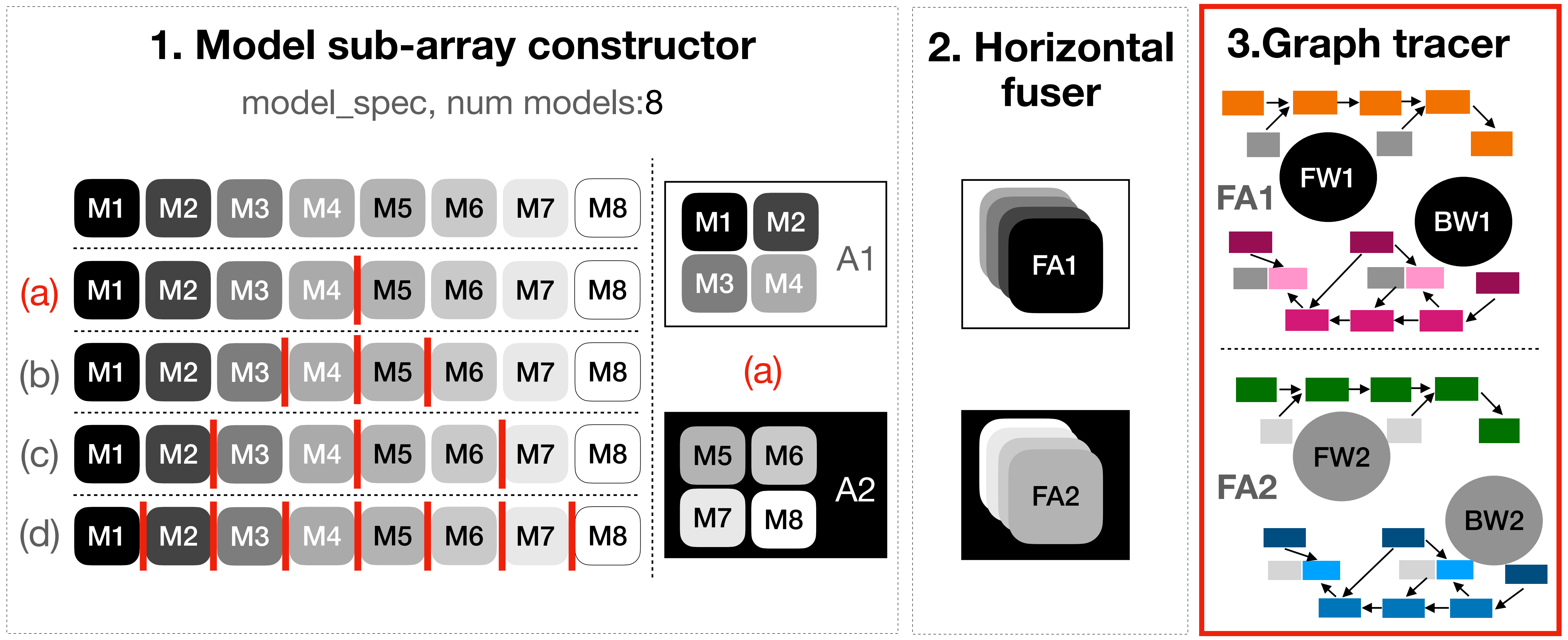
Weight initialization

Learning rate

Momentum

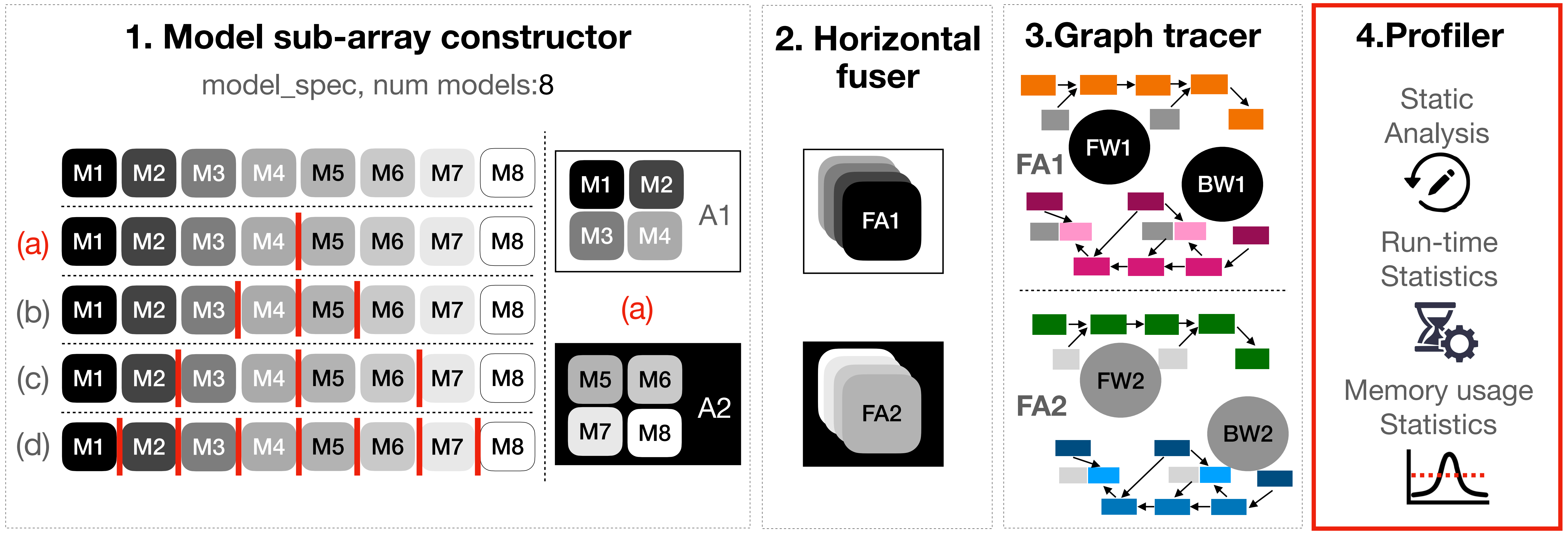
Can train on different data set/partitions

μ-TWO System Overview



Trace forward and backward pass graphs for each fused sub-array

μ-TWO System Overview



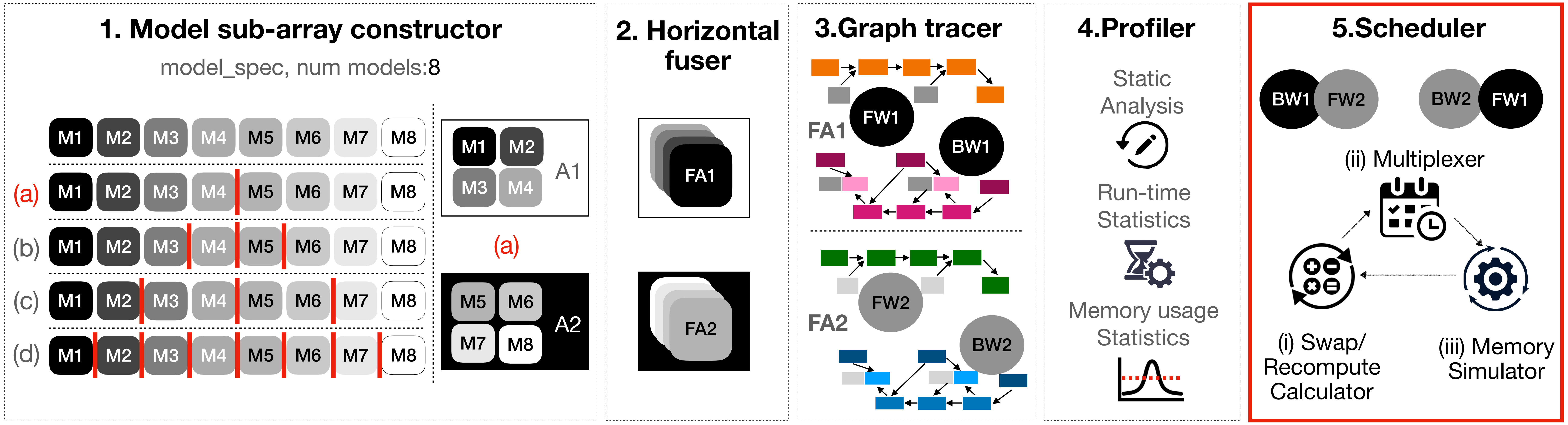
Calculate the run-time of every operation in graph

Measure the swap-time of all intermediate tensors

Measure the peak memory usage of each node

To be able to profile graphs larger than GPU memory

μ-TWO System Overview



Choose whether an intermediate tensor should be

Recomputed

Sources to recompute the tensor?

Recompute overhead

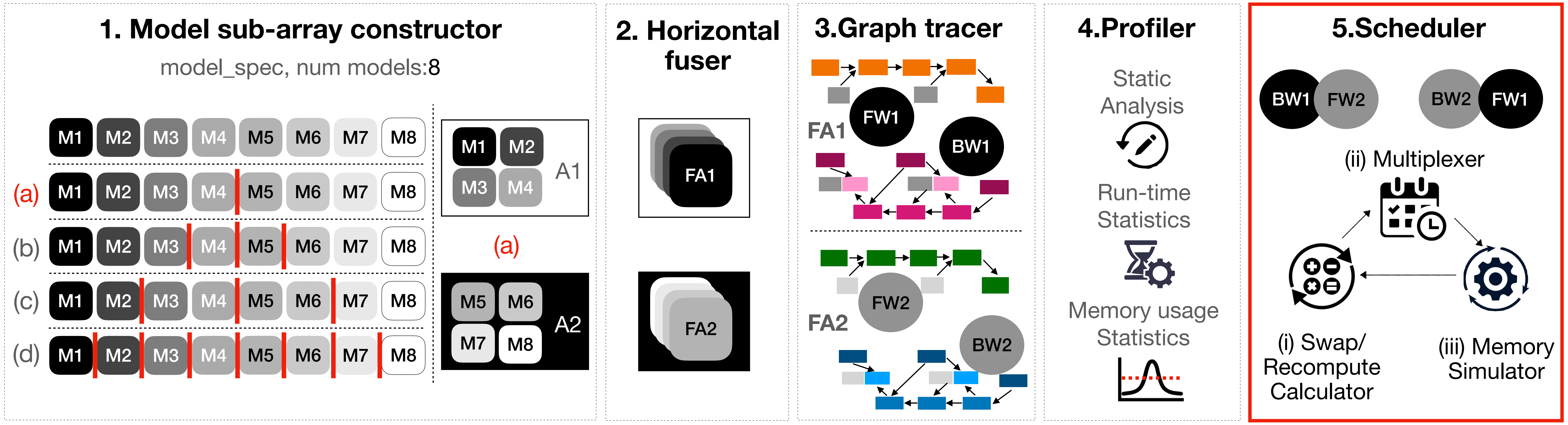
Swapped

When to prefetch/offload

STALL

Swap overhead

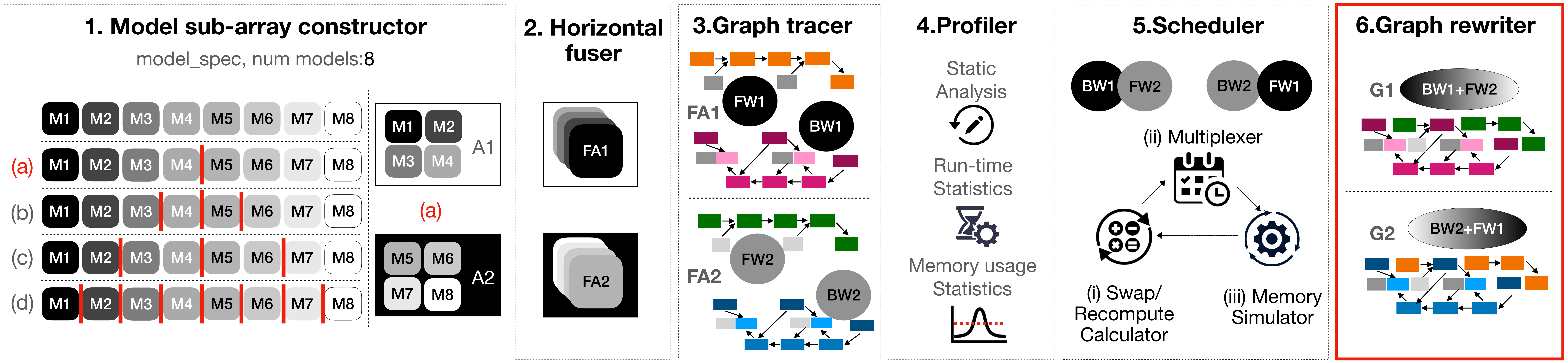
μ-TWO System Overview



Multiplexer overlaps all the swaps in the backward pass of one fused sub-array with forward pass of another

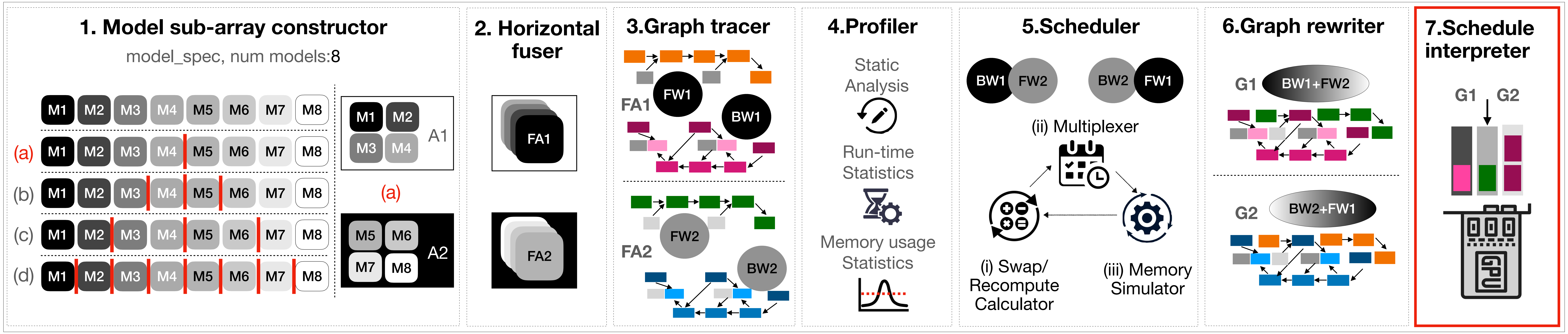
Memory simulator ensures that the peak memory consumption post the reordering pass is within the GPU memory limit

μ-TWO System Overview



Embed the scheduling information in the graphs

μ-TWO System Overview

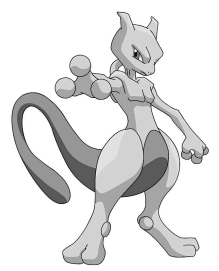


Enqueue the appropriate operations across different execution queues

Add synchronization markers for coordination across queues

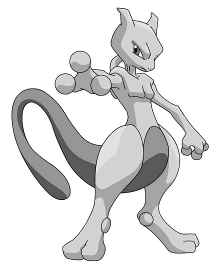
Experimental Setup

Instance	Nvidia GPU Version	GPU Mem (GB)	Tensor Cores	CPU-GPU Link	CPUs	CPU Mem
AWS p4d24-large	A-100	40	Yes	PCI-e Gen 4 x16 (32GB/s)	16	1152
Dell Claudron DSS 8440	Tesla V-100	32	Yes	PCI-e Gen 4 x16 (32GB/s)	16	384



Workload

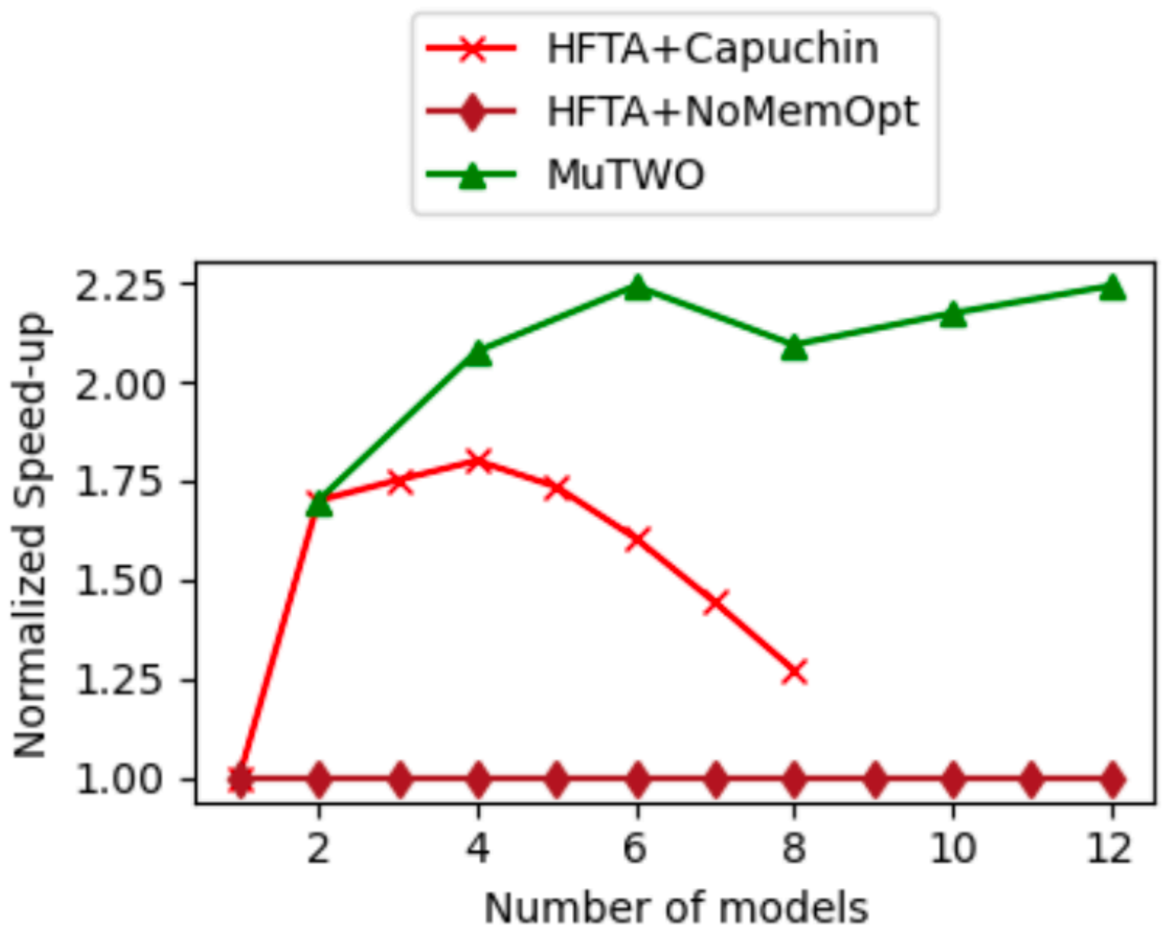
Application	Model Name	Functionality	Architectural Features	Params	Batch Sizes
Vision	Vision Transformer	Image Classification, Image Segmentation, Action Recognition	Positional image embeddings, transformers	60M	8 16
	Mobilenet v3 large		Depthwise separable convolutions	5.4M	64 128
	Resnet101		Convolutions, Skip Connections	44.5M	48 64
Natural Language Processing	Bert	Predict Next Sentence	Transformer Encoders	100M	16 24
	GPT2	Predict Next Token	Transformer Decoders	124M	8 16
Recomm-der Systems	NVIDIA DLRM	Item Recomm-endation	Encoders, Decoders, sparse embeddings	40M	512 1024



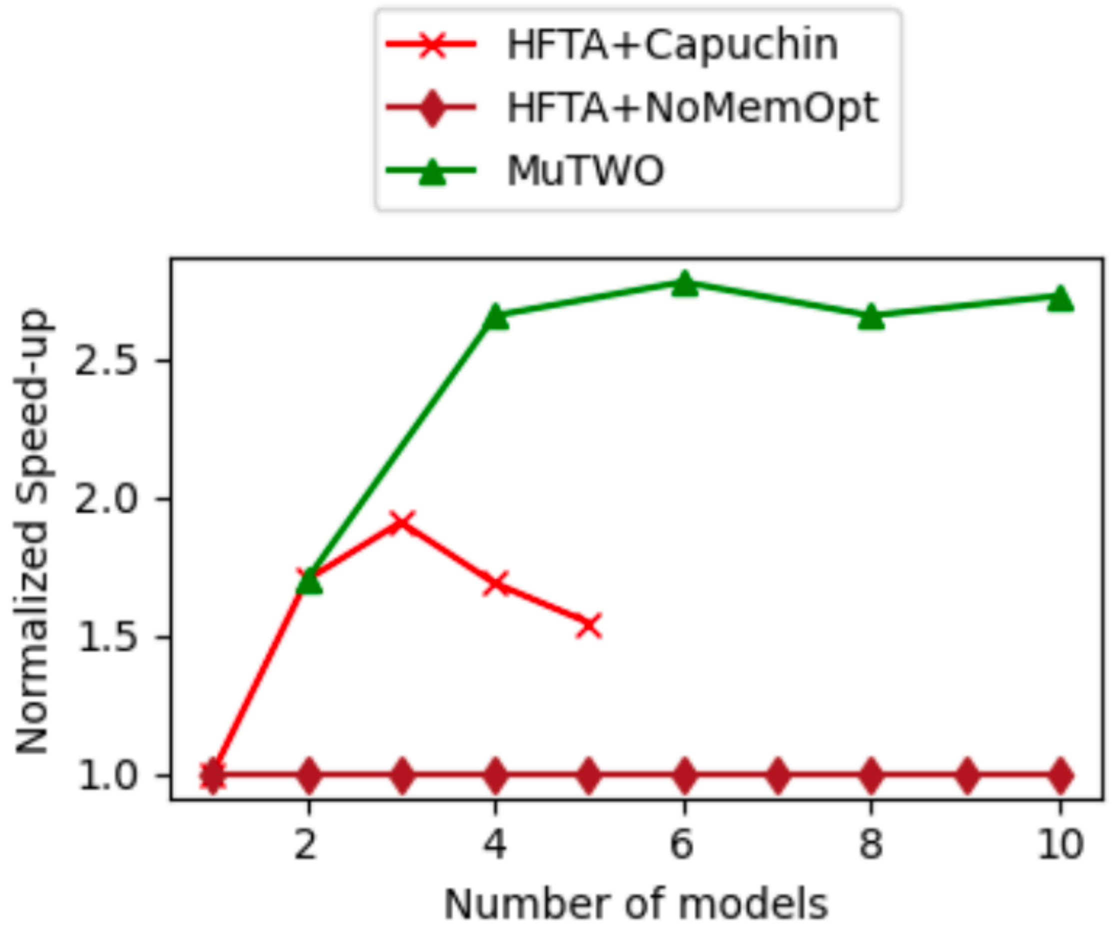
Baselines

- HFTA-NoMemOpt - Horizontal Fusion only with no memory optimization
- HFTA-Capuchin - HFTA with Capuchin Algorithm applied directly
- μ-TWO - Multi-model training with orchestration and memory optimization

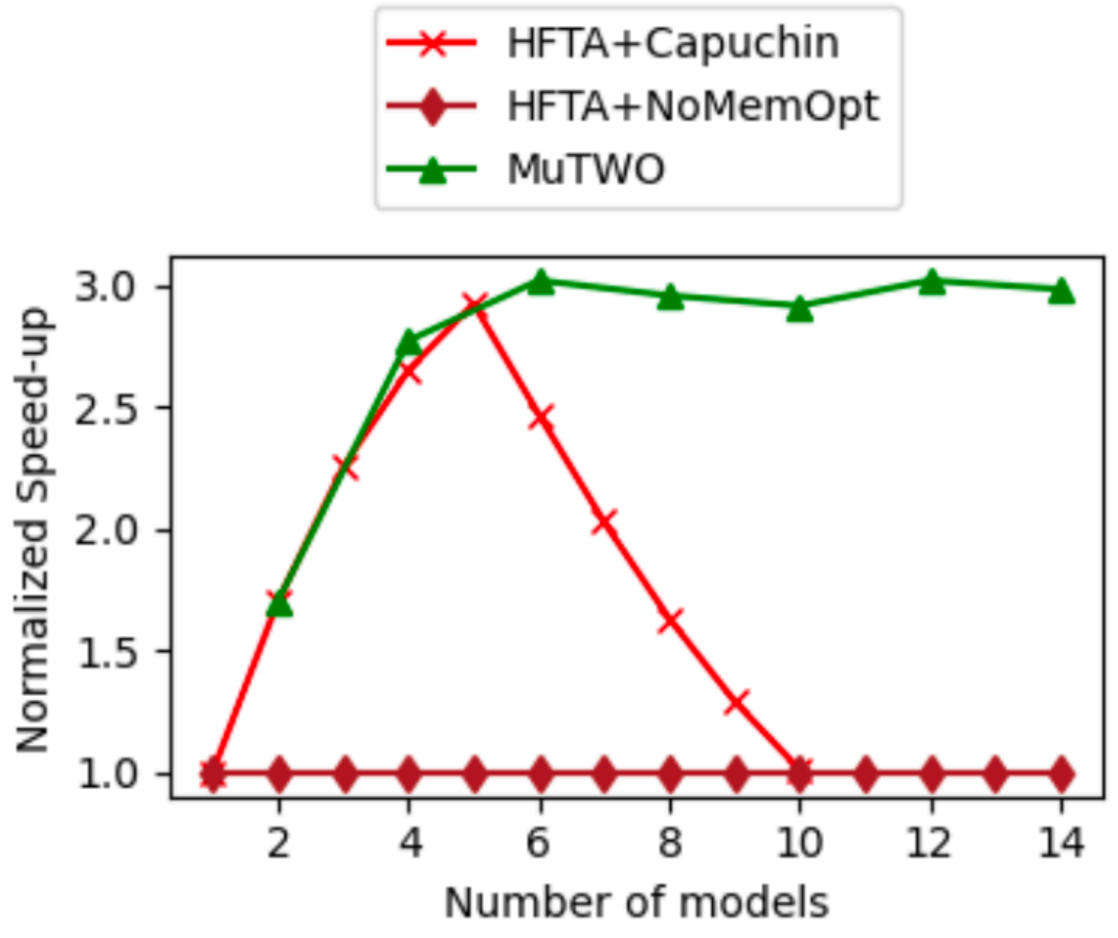
μ-TWO achieves upto 3x Speed-up



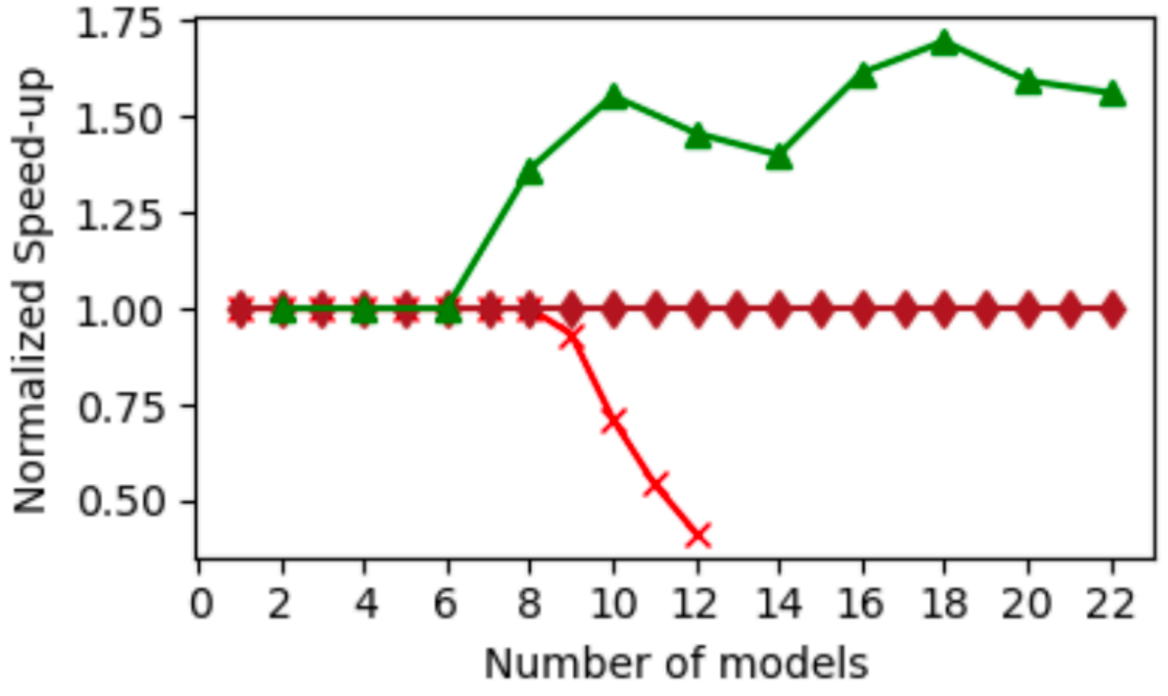
(a) Bert (Batch size: 24)



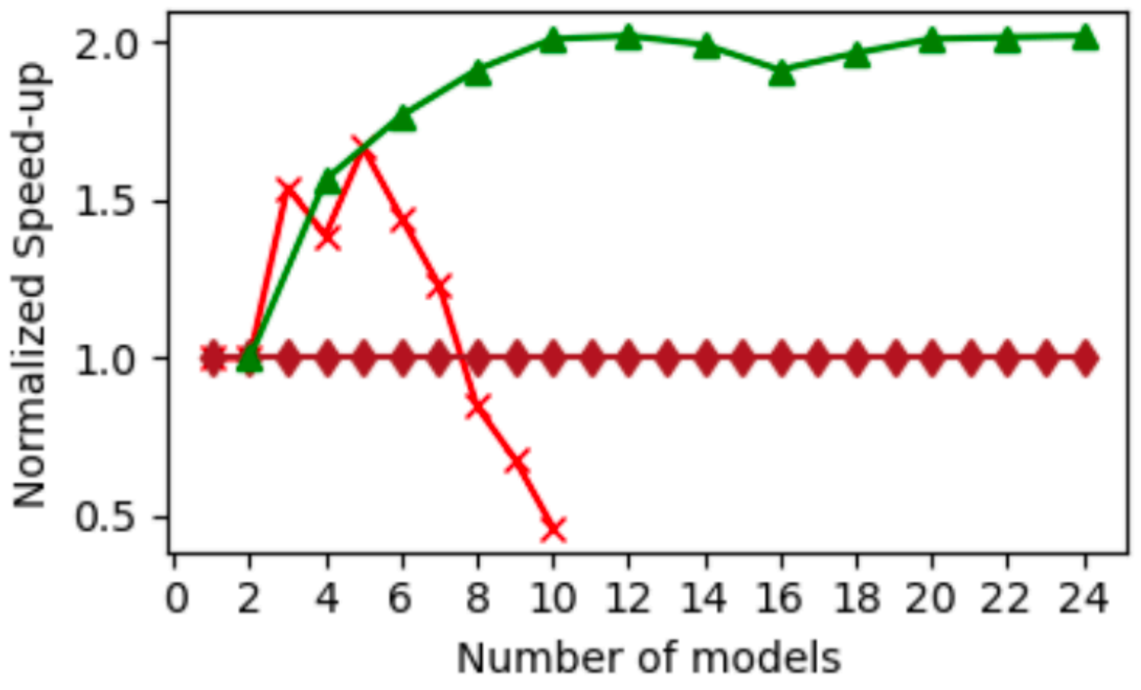
(b) GPT2 (Batch size: 16)



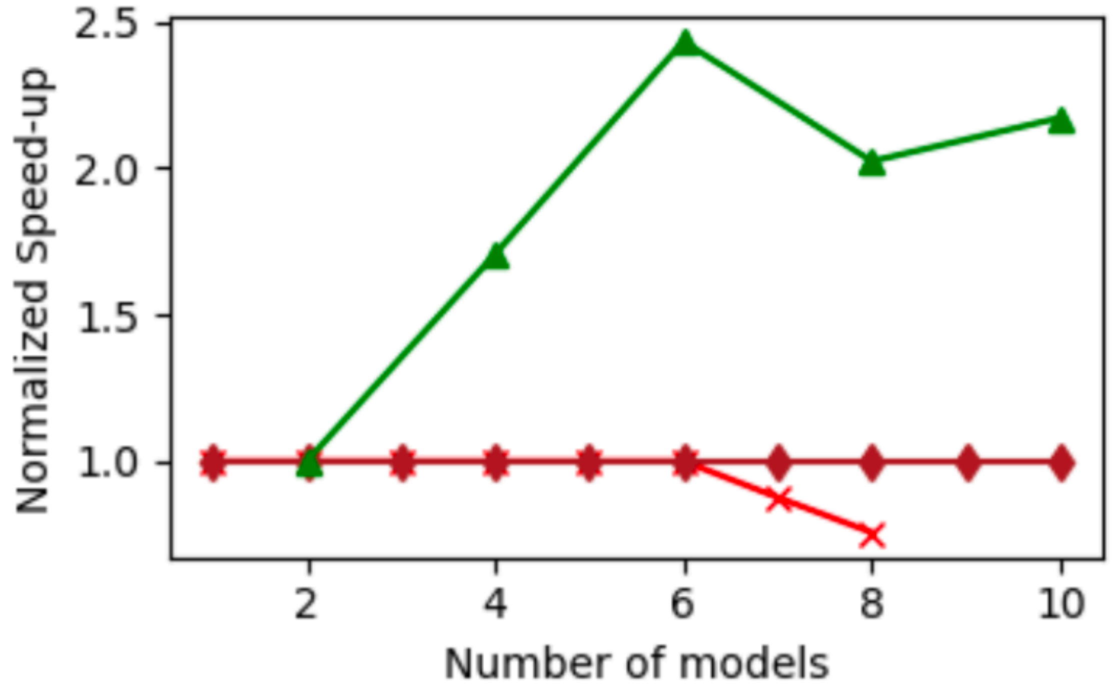
(c) Vision Transformer (Batch size: 8)



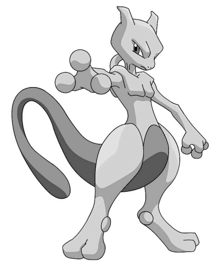
(d) Mobilenet v3 large (Batch size: 64)



(e) Resnet101 (Batch size: 48)



(f) NV DLRM (Batch size: 1024)



DASlab
@ Harvard SEAS

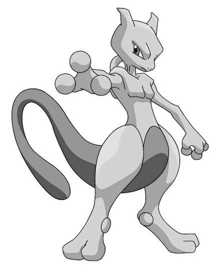
Performance Breakdown

Useful Compute: Computation time spent in necessary operations

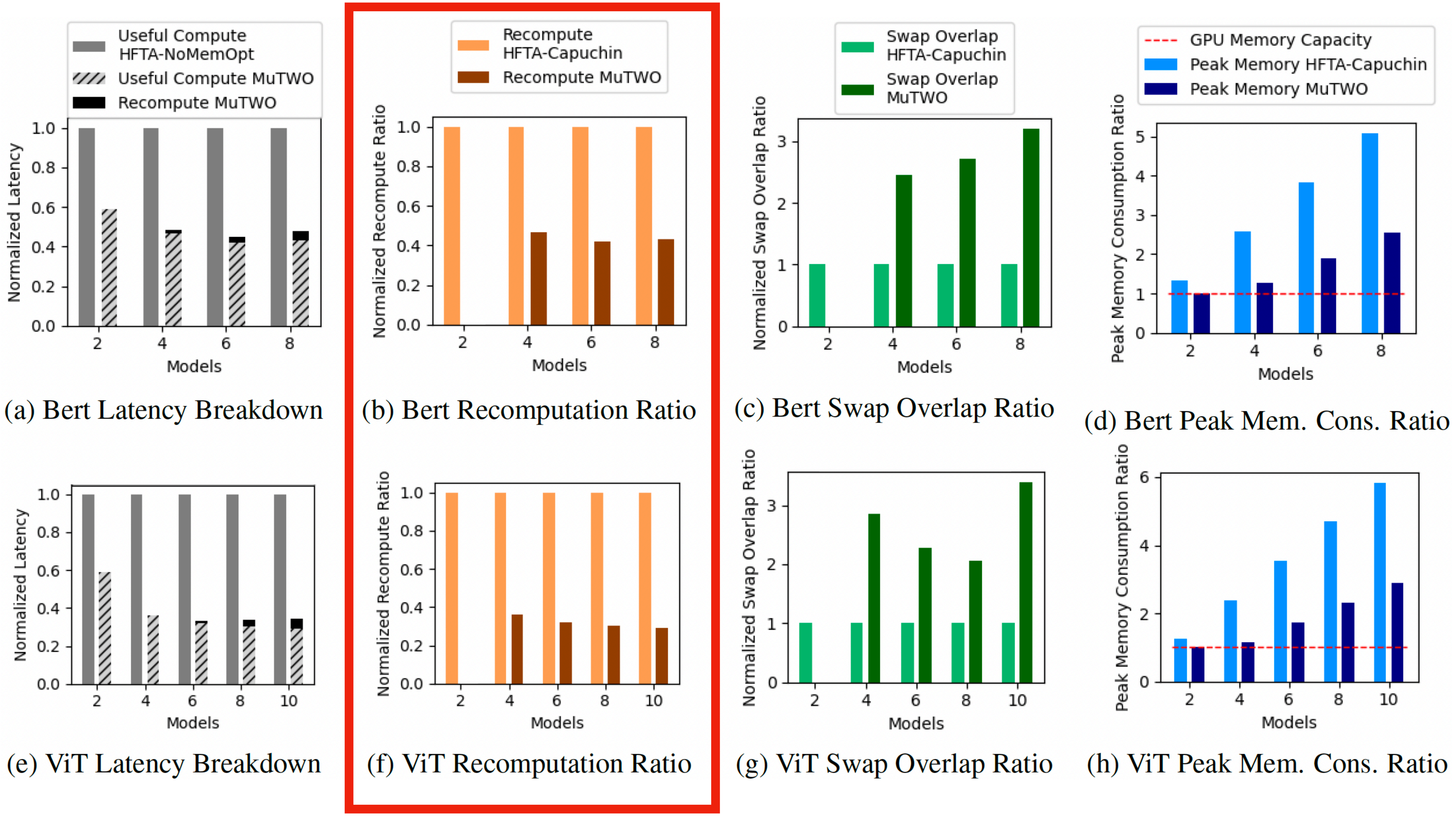
Recomputation: Computation time spent in recompute operations

Swap Overlap: Successful overlap with compute operations

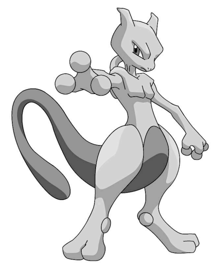
Peak memory Consumption: Maximum memory consumed at any point during the entire iteration



Performance Breakdown

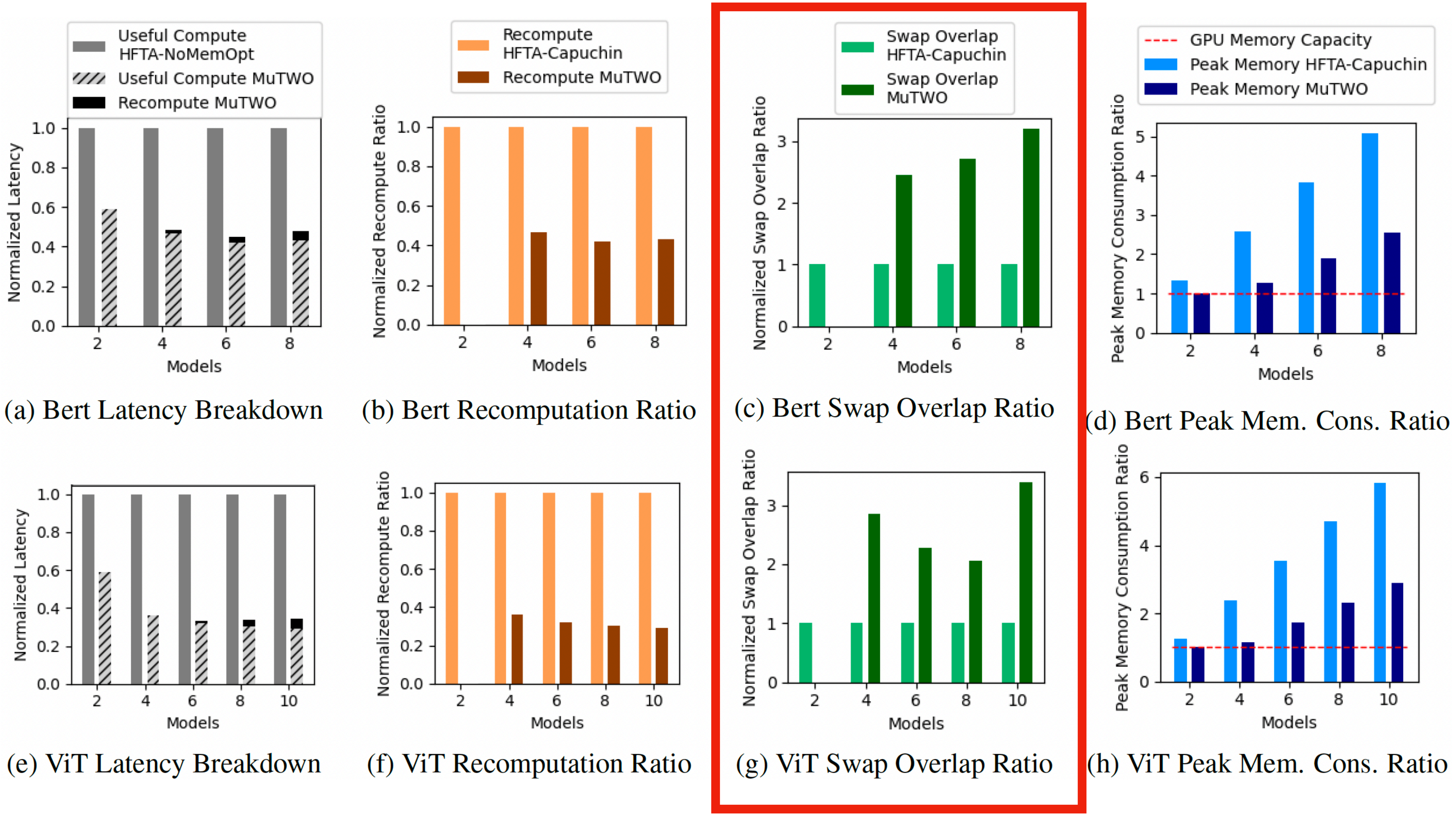


Less than 50% Recomputation



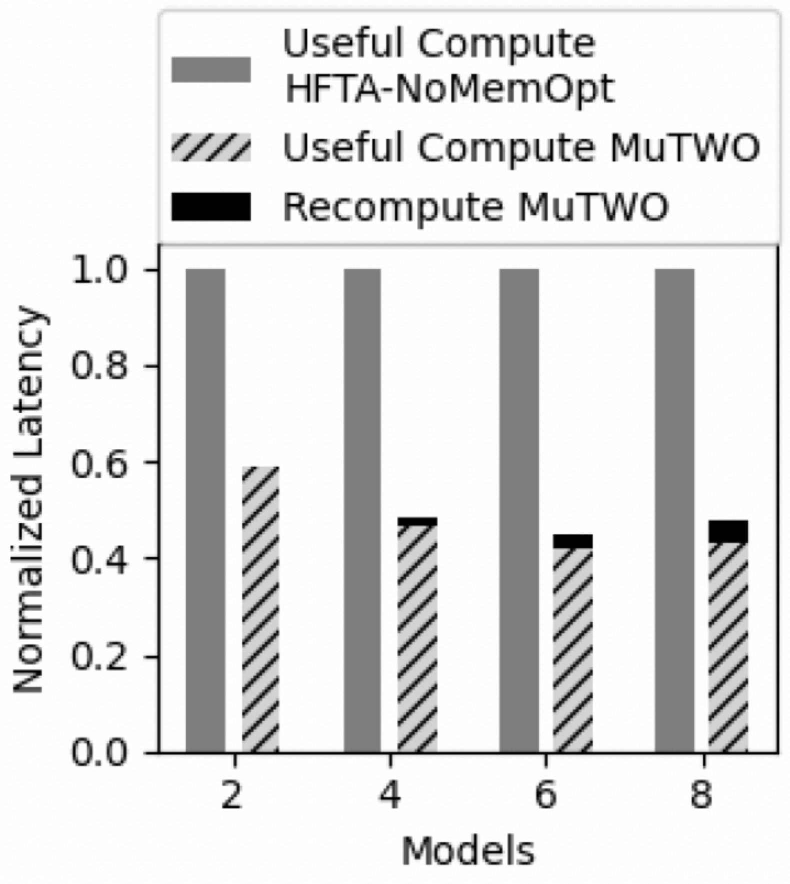
DASlab @ Harvard SEAS

Performance Breakdown

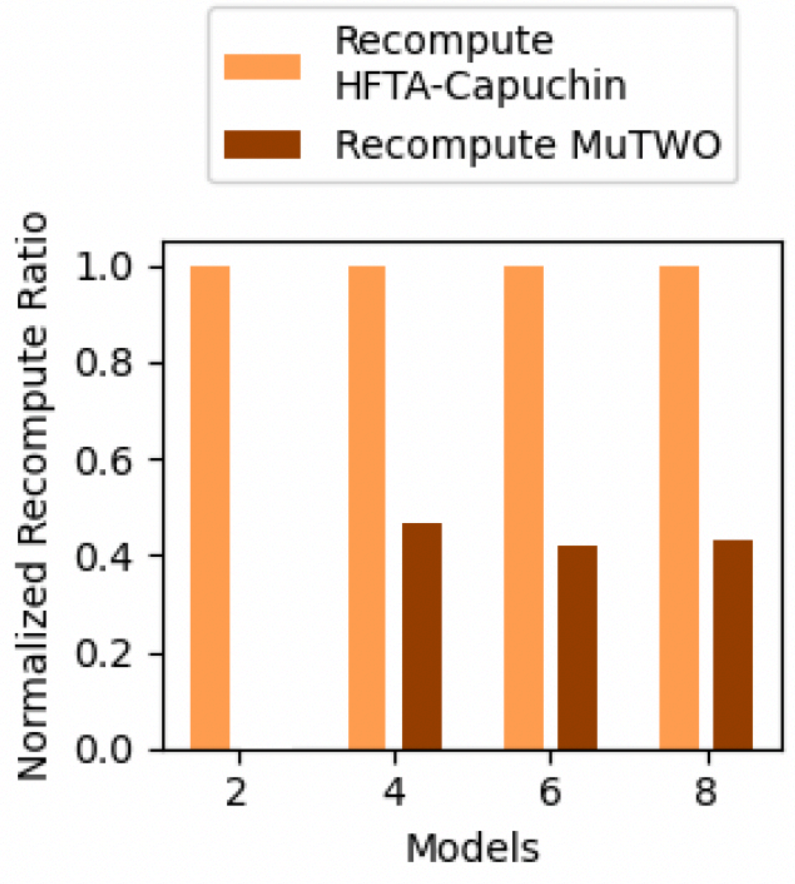


3x more Swap Overlap

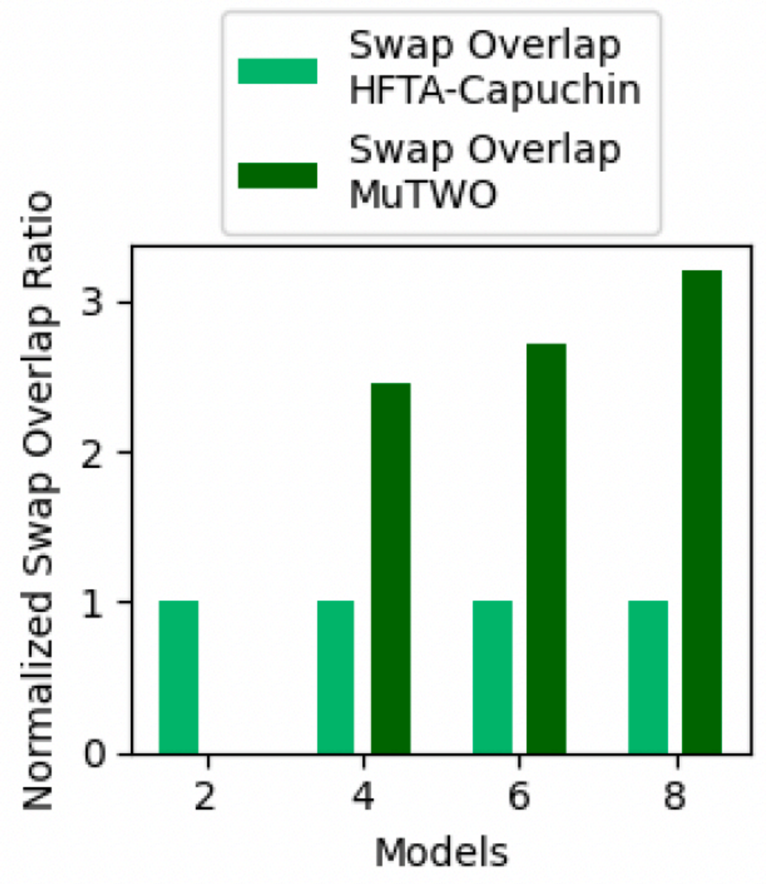
Performance Breakdown



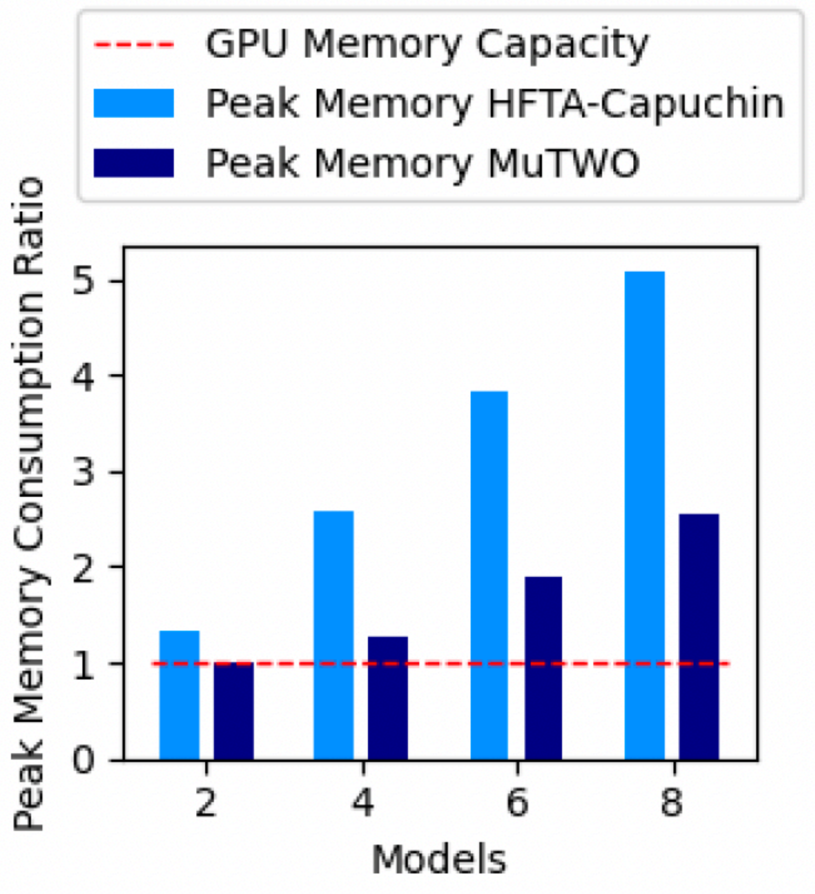
(a) Bert Latency Breakdown



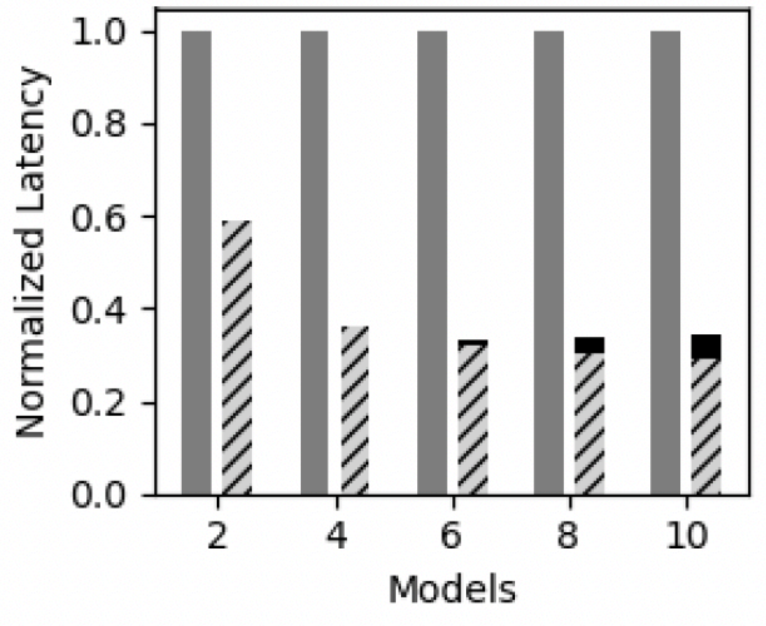
(b) Bert Recomputation Ratio



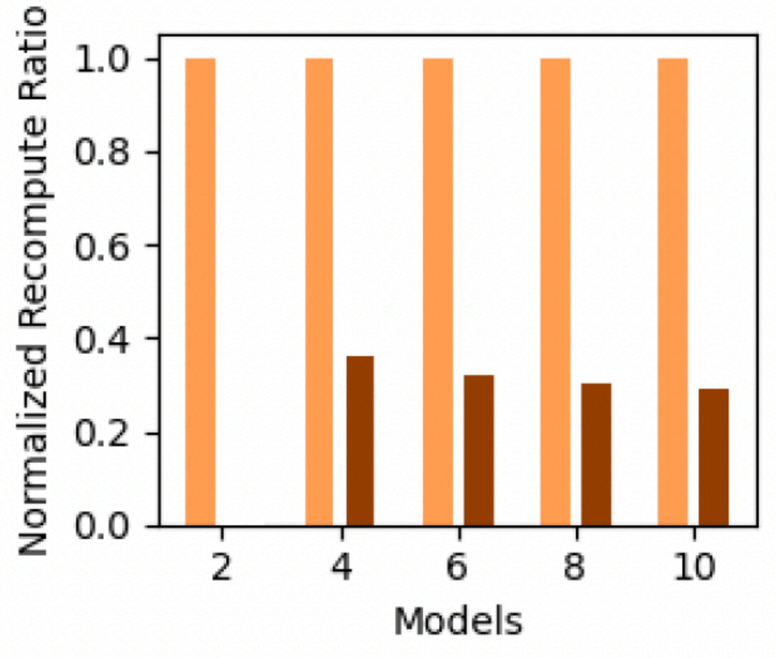
(c) Bert Swap Overlap Ratio



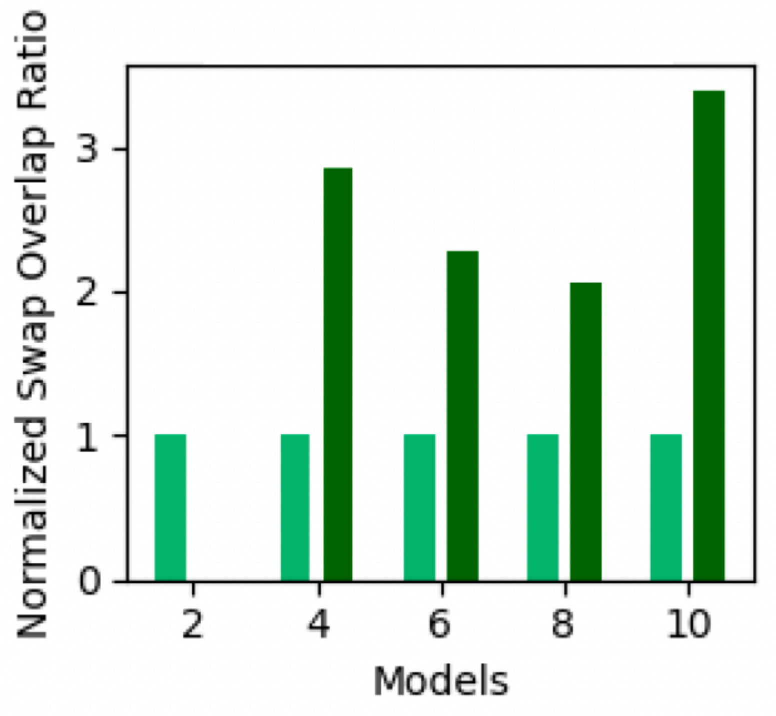
(d) Bert Peak Mem. Cons. Ratio



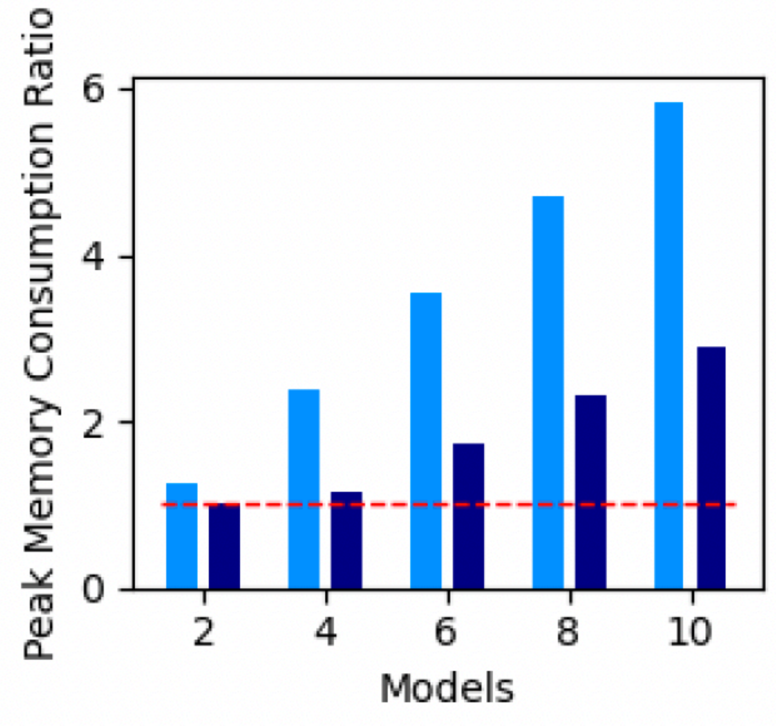
(e) ViT Latency Breakdown



(f) ViT Recomputation Ratio



(g) ViT Swap Overlap Ratio



(h) ViT Peak Mem. Cons. Ratio

Less than 50% peak memory consumption

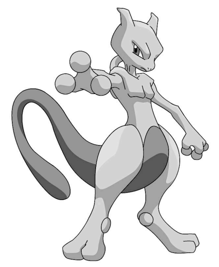
@ Harvard SEAS

Implementation Details

- Profiling: PyTorch Profiler
- Parallel Compute and Data Operations: CUDA Streams and Events
- Operator Fusion: PyTorch VMap
- Computational Model Graphs: PyTorch AOT Autograd with FakeTensors
- Runtime Overhead Reduction: CUDA Graphs

State of the Art vs μ-TWO

Parameter	HFTA (MLSys'21)	Checkmate (MLSys'20)	Capuchin (ASPLOS'20)	μ-TWO
High Compute Utilization	Yes	No	No	Yes
High Memory Utilization	No	Yes	Yes	Yes
Large Number of Models	Yes	No	No	Yes
Large Model Size	No	Yes	Yes	Yes
Large Mini-batch Size	No	Yes	Yes	Yes
Stalls	NA	NA	Low	Low
Compute Overhead	NA	High	Moderate	Low



DASlab @ Harvard SEAS

Thank You

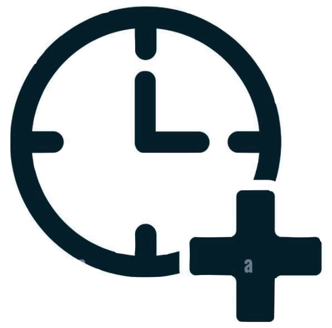
Choose whether an intermediate tensor should be

Recomputed **Swapped**

Combinatorial in nature

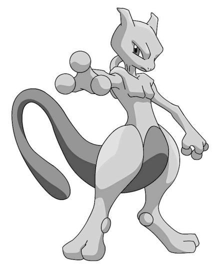
NP-HARD

Jain Paras et al.,Checkmate: Breaking the memory wall with optimal tensor rematerialization, *MLSys 2020*

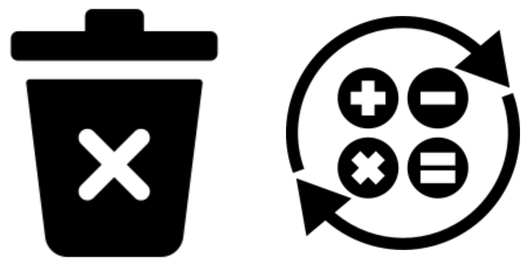


ILP Solvers
Approximation algorithms

GREEDY



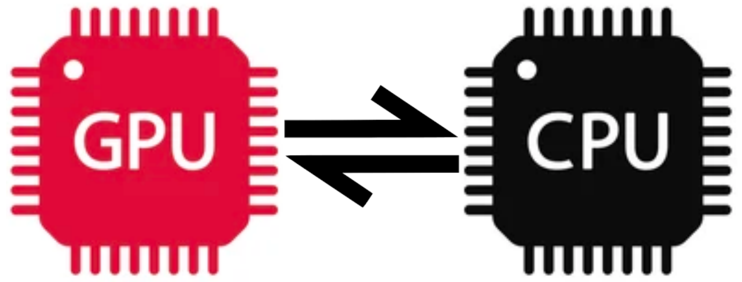
Recomputation



$$recompute\_ratio = \frac{memory\_size}{recomp\_time}$$

Memory Savings Per Second

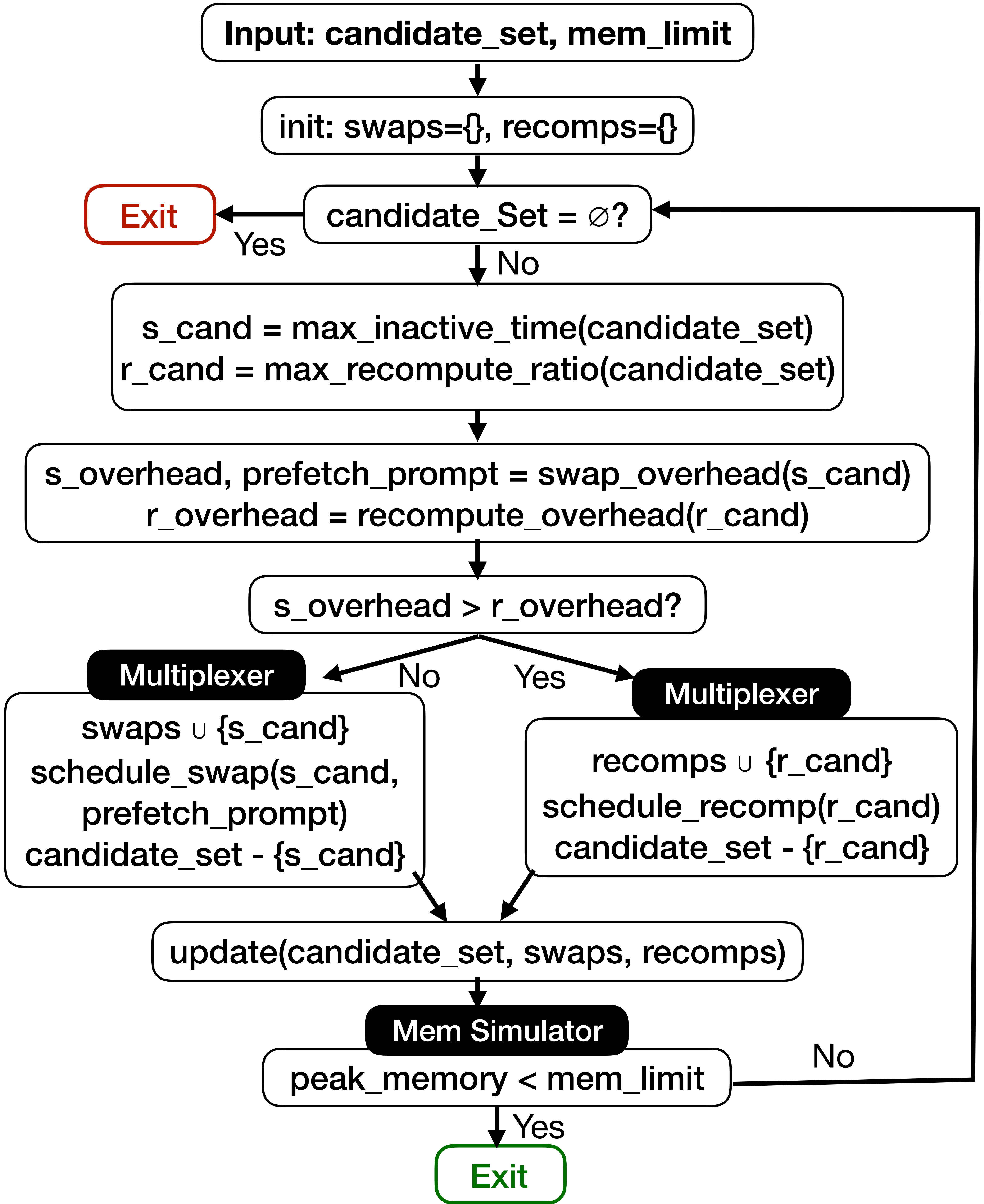
Swapping



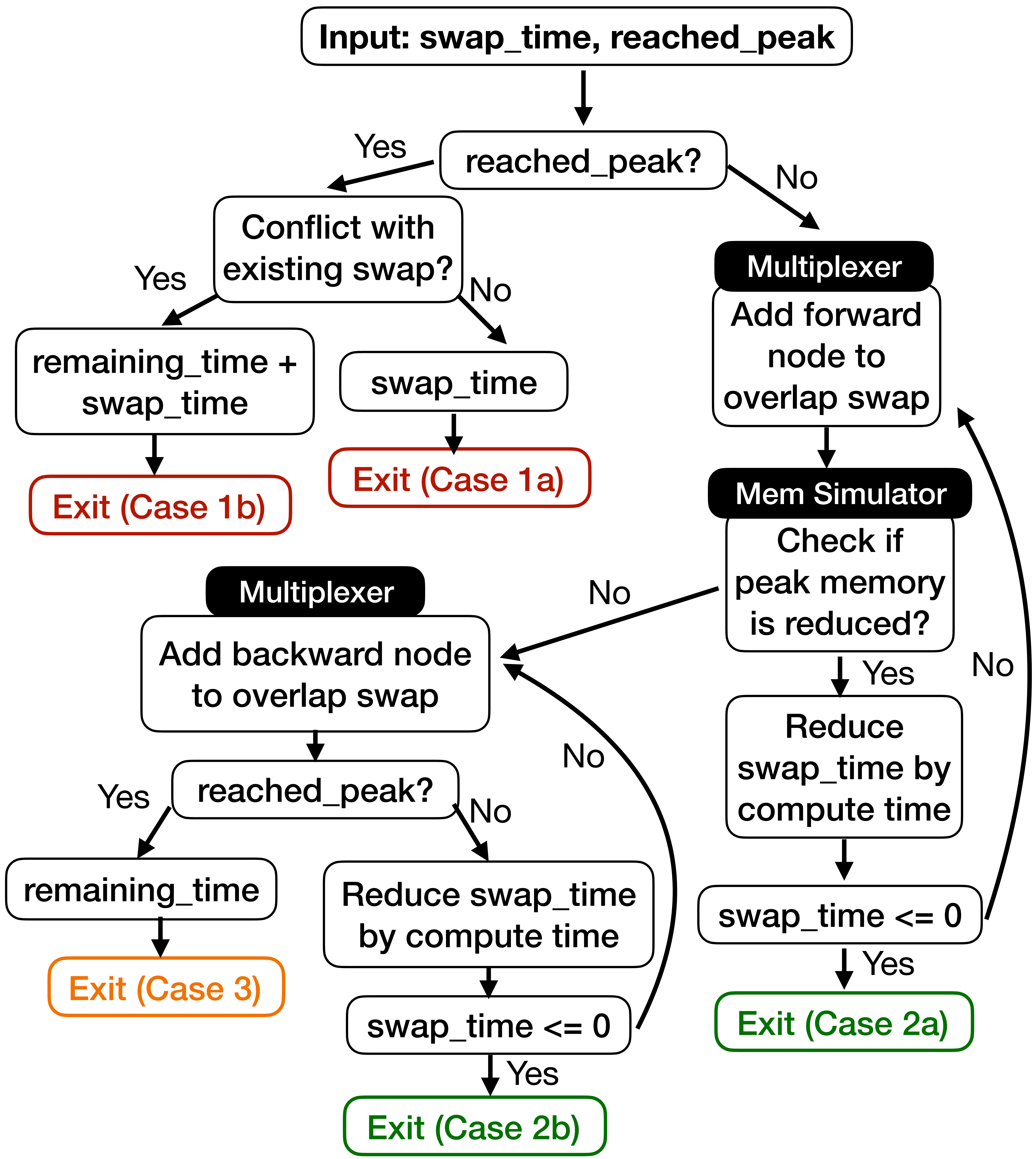
Inactive Time: Last Use (Forward Pass) -> First Use (Backward Pass)

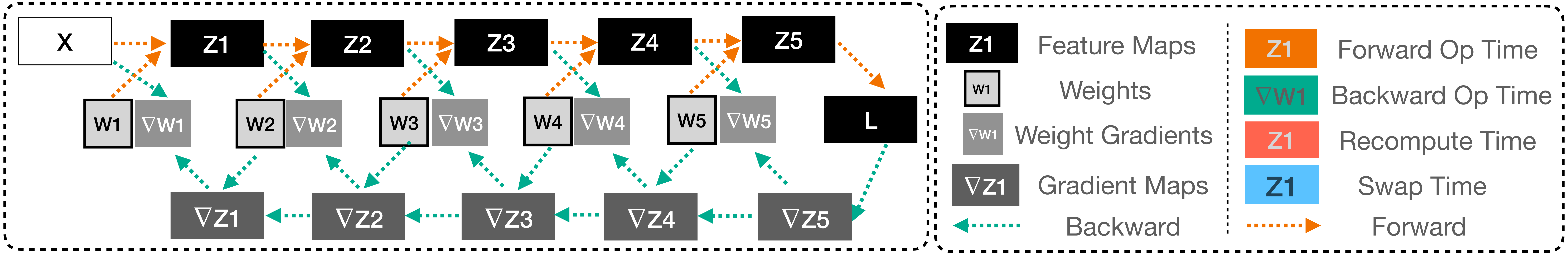
Better opportunity to hide swapping latency

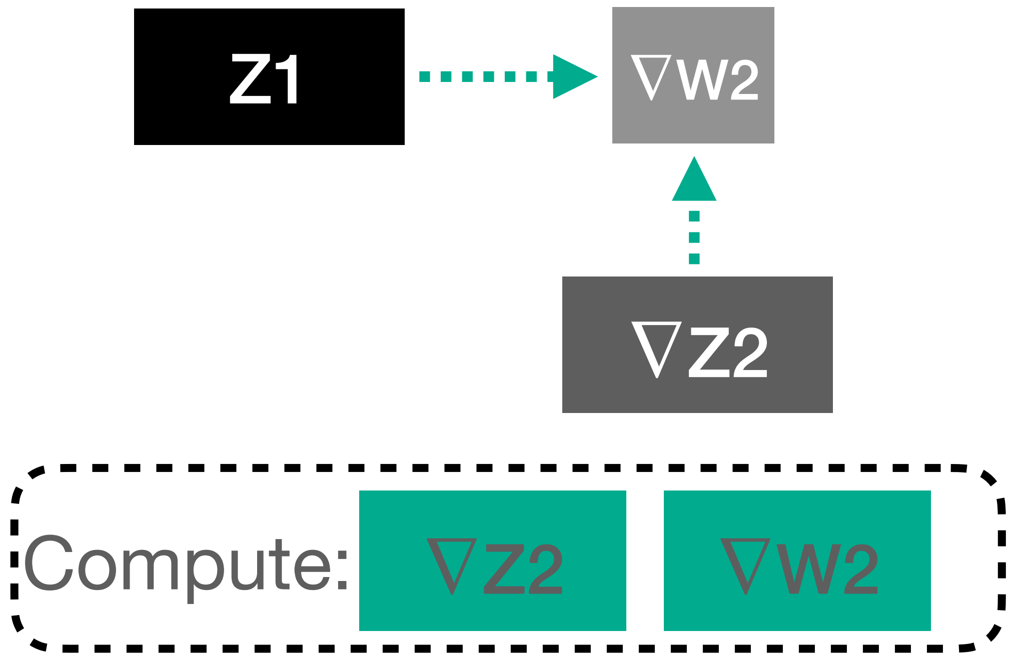
Scheduling Policy

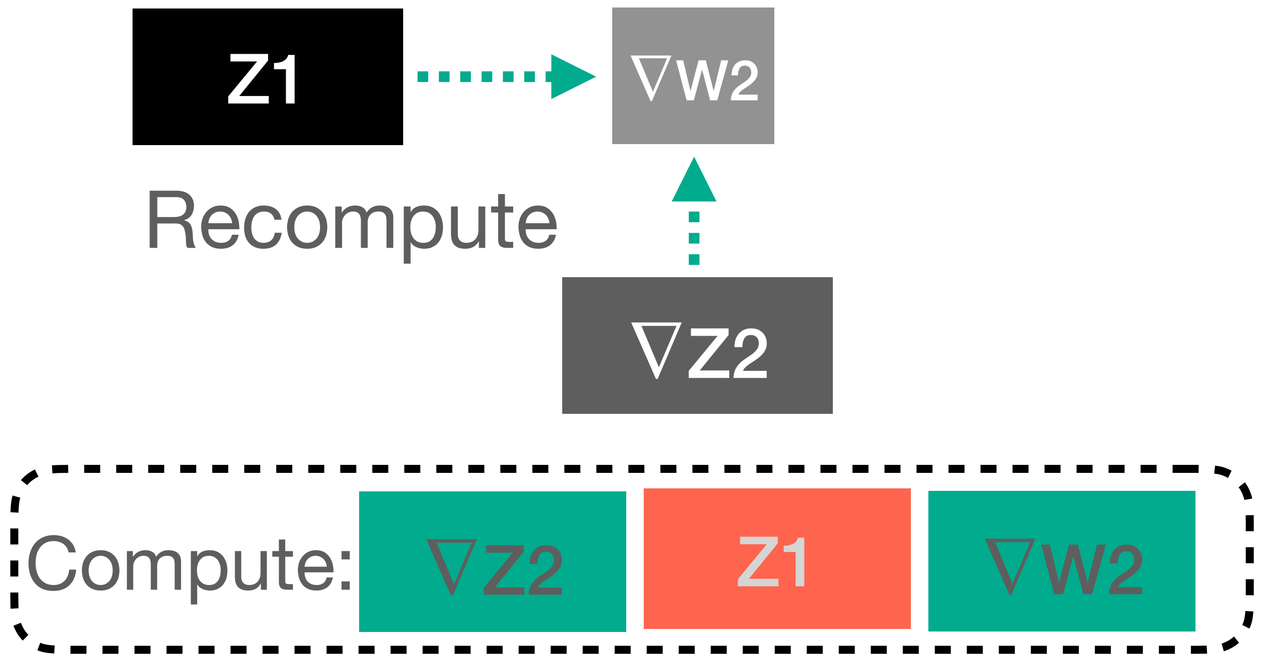


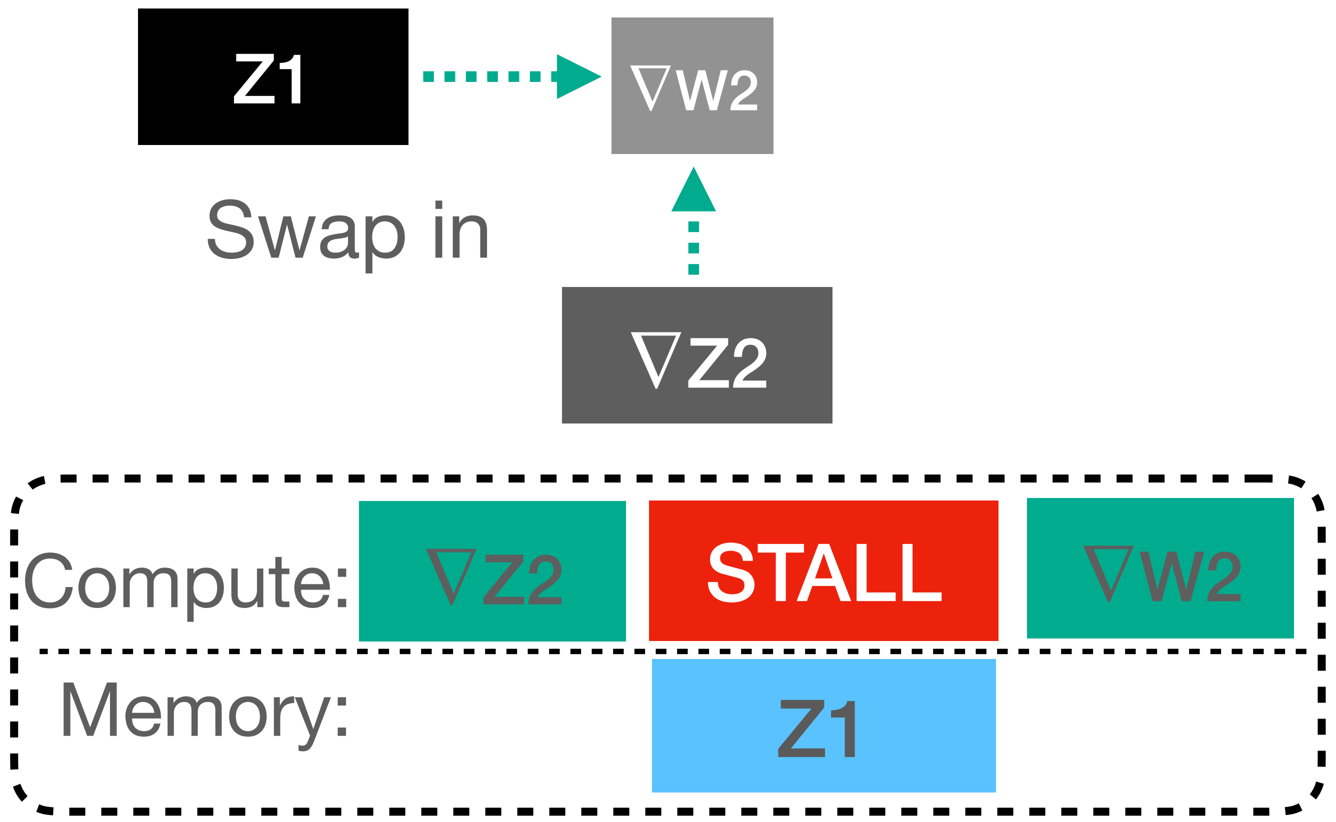
Swap Overhead Calculation



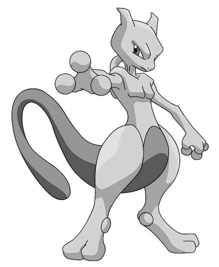




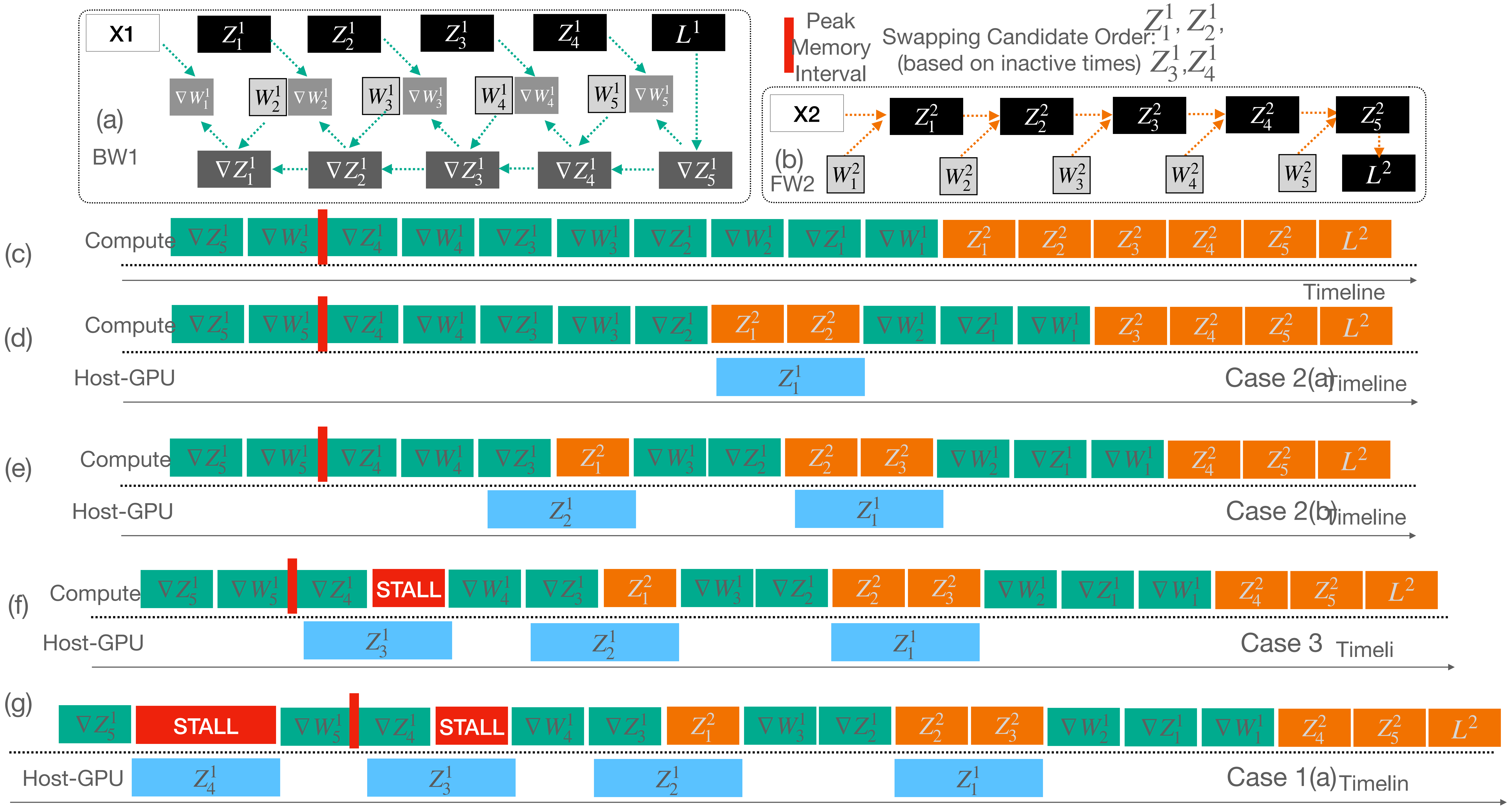




Operation Timelines



DASlab @ Harvard SEAS



X1
Z_1^1 Z_2^1 Z_3^1 Z_4^1 L^1
∇W_1^1 W_2^1 ∇W_2^1 W_3^1 ∇W_3^1 W_4^1 ∇W_4^1 W_5^1 ∇W_5^1
(a)
BW1
∇Z_1^1 ∇Z_2^1 ∇Z_3^1 ∇Z_4^1 ∇Z_5^1
Peak Memory Interval
Swapping Candidate Order: $Z_1^1, Z_2^1,$
(based on inactive times) Z_3^1, Z_4^1
X2
Z_1^2 Z_2^2 Z_3^2 Z_4^2 Z_5^2
(b)
FW2
W_1^2 W_2^2 W_3^2 W_4^2 W_5^2 L^2
(c)
Compute
Timeline
(d)
Compute
Host-GPU
Case 2(a)
Timeline
(e)
Compute
Host-GPU
Case 2(b)
Timeline
(f)
Compute
STALL
Host-GPU
Case 3
Timeli
(g)
STALL
STALL
Host-GPU
Case 1(a)
Timelin