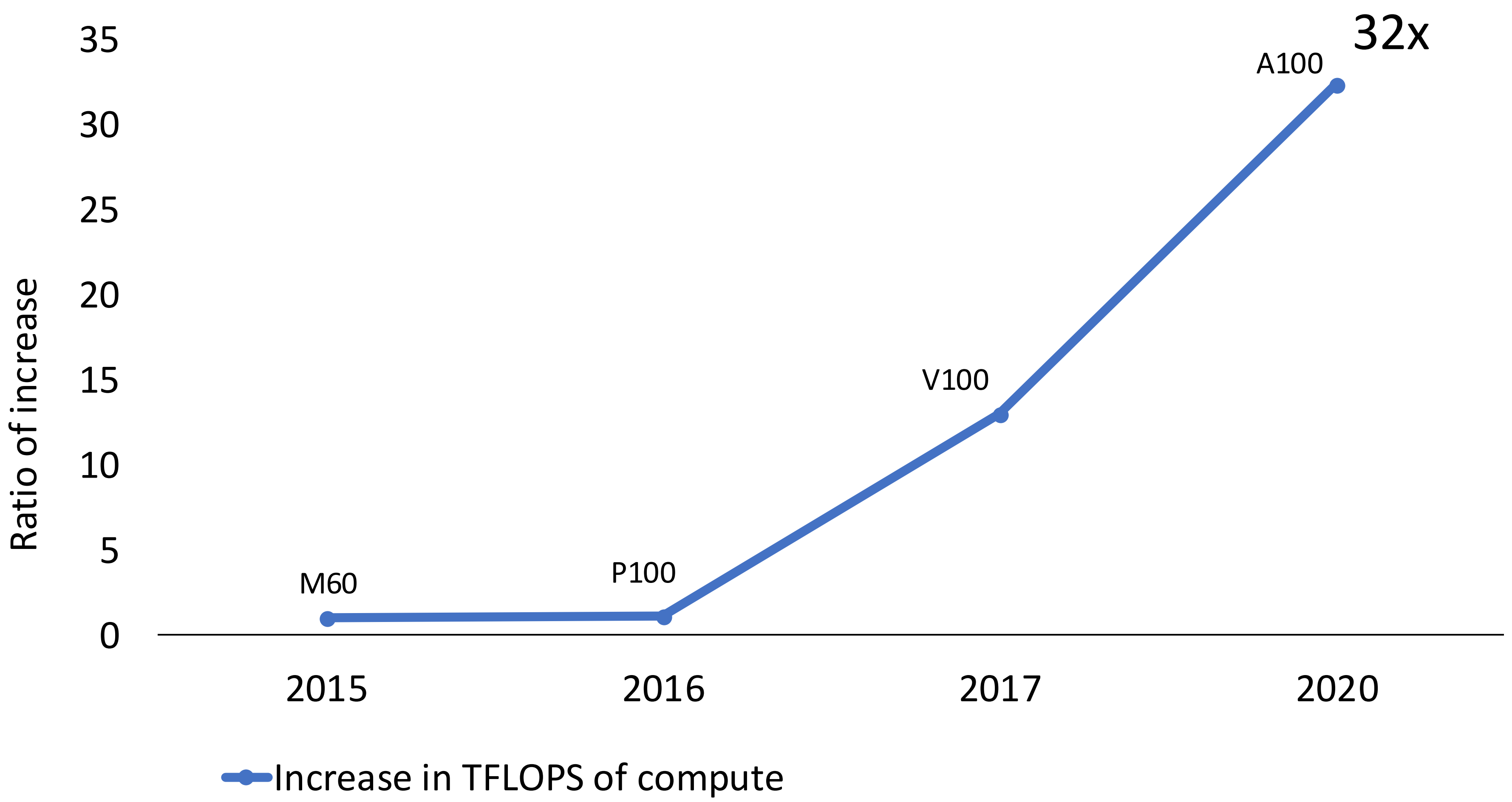
Memory Optimization for Deep Networks

Aashaka Shah[1], Chao-Yuan Wu[1], Jayashree Mohan[1],
Vijay Chidambaram[1,2], Philipp Krähenbühl[1]

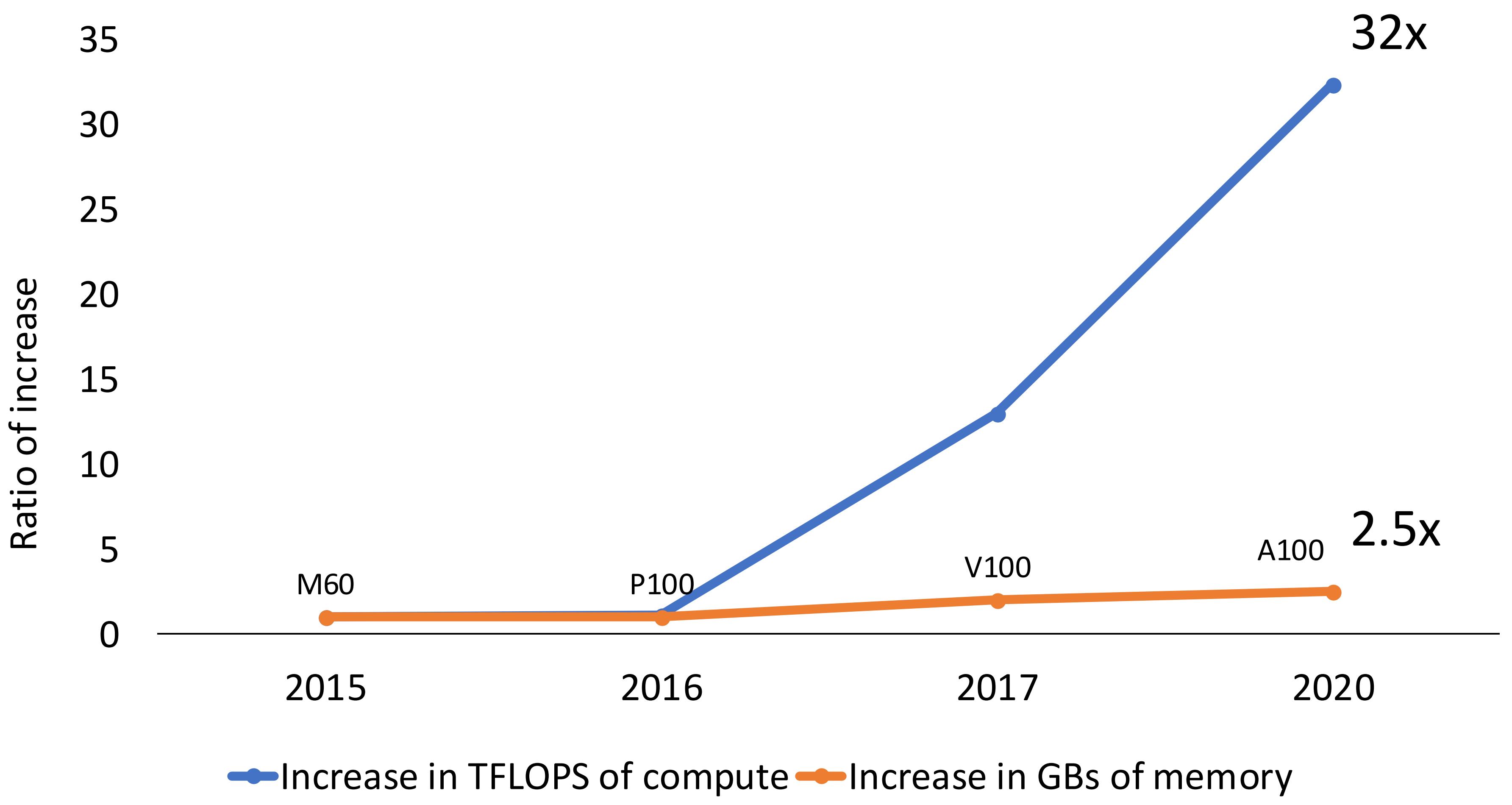
1 TEXAS The University of Texas at Austin

2 vmware

Memory and Compute Trends in GPUs

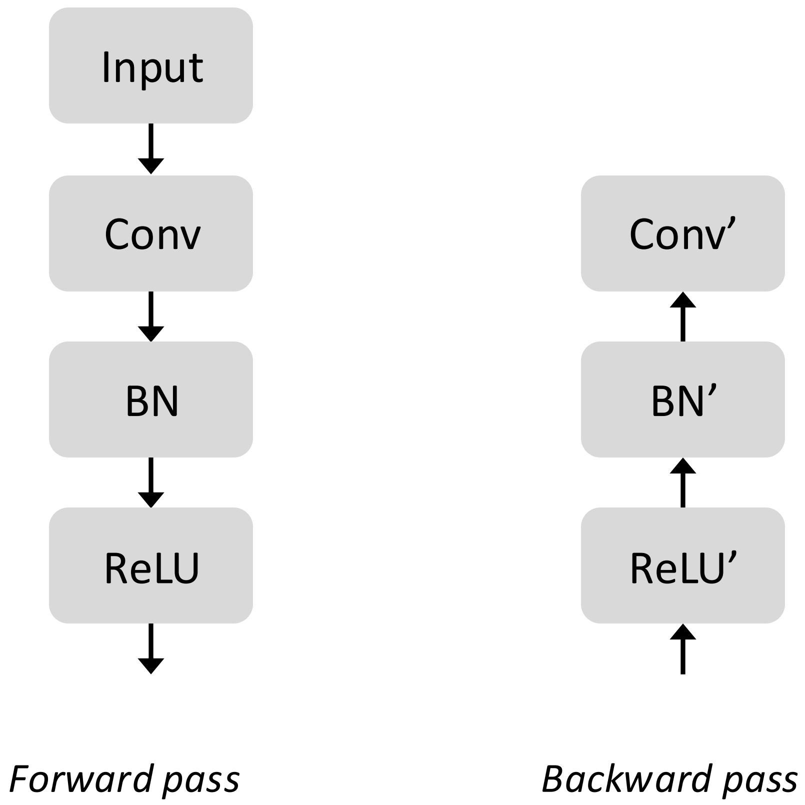


Memory and Compute Trends in GPUs

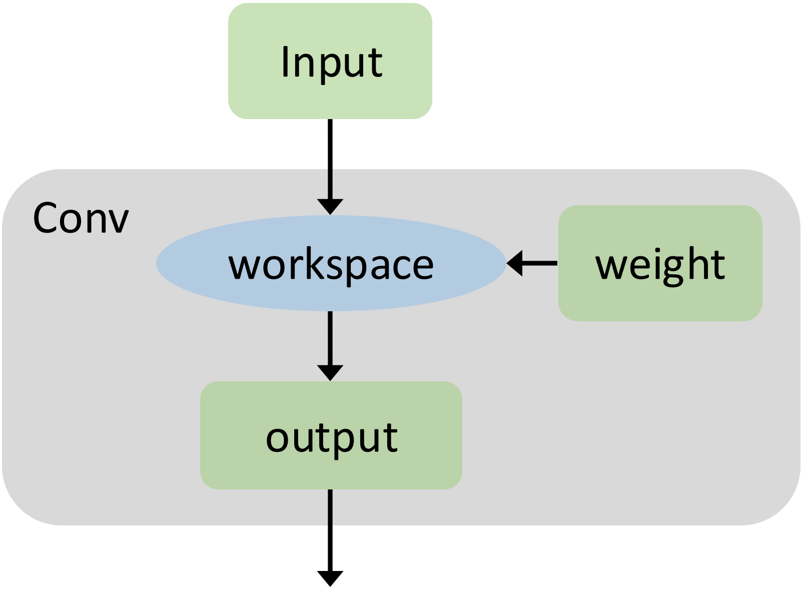


```
(pytorch_env) cc@gpu:~$
```

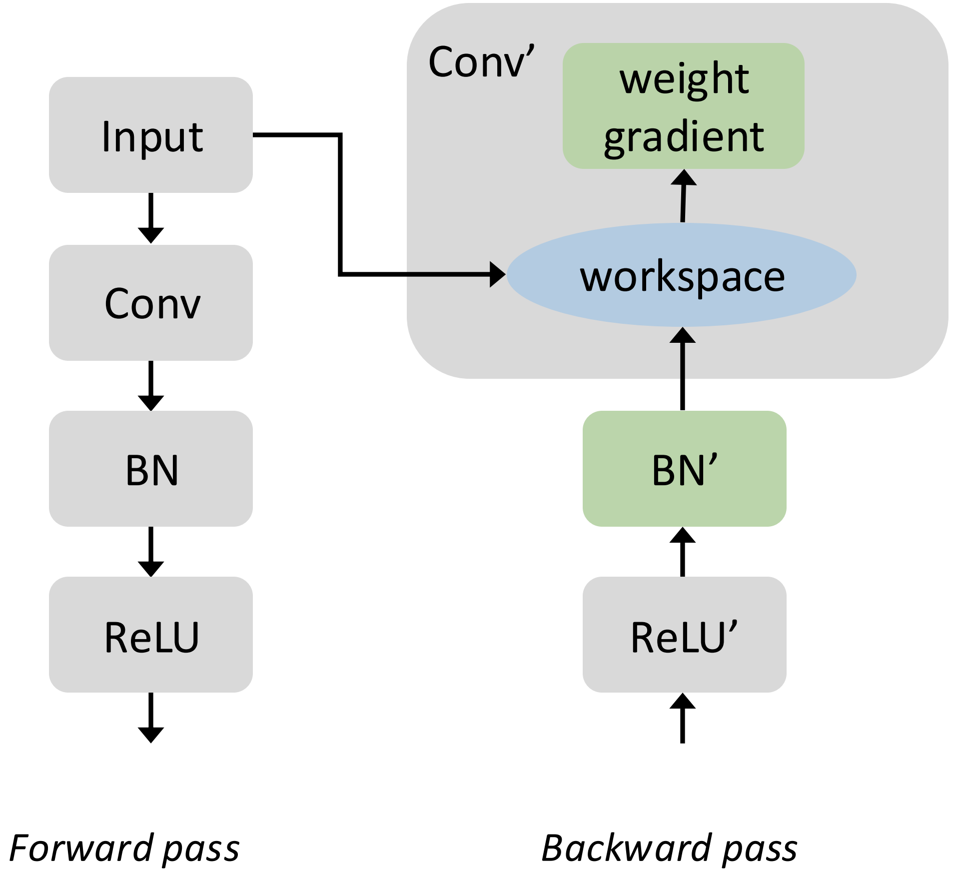
Memory usage in deep networks



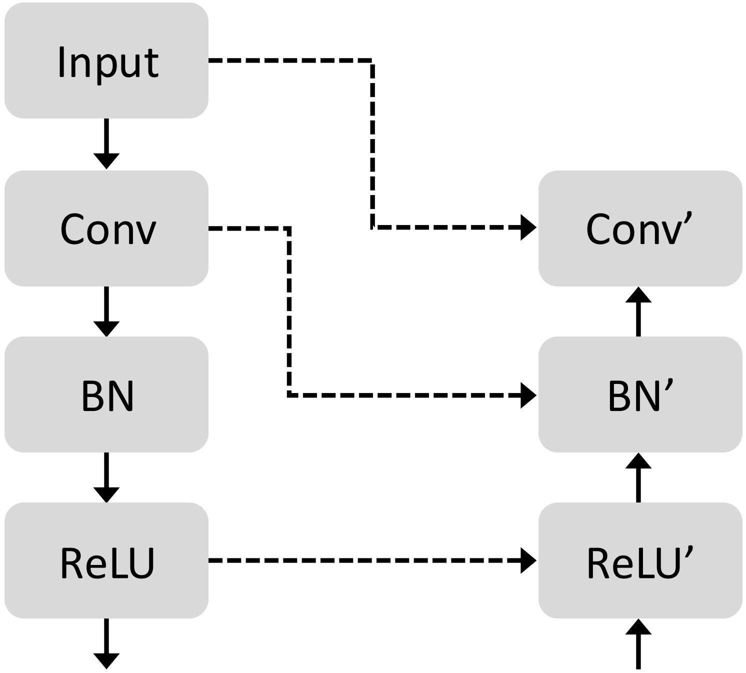
Memory usage in deep networks



Memory usage in deep networks



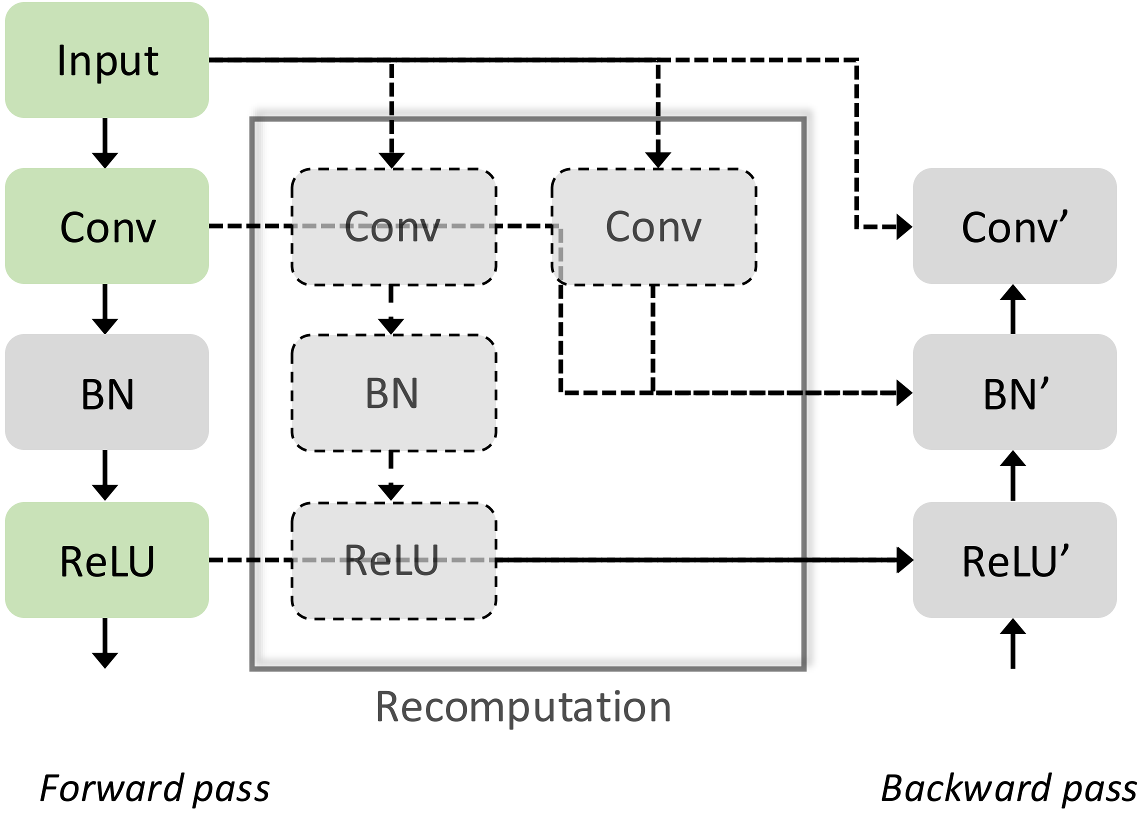
Memory usage in deep networks



Forward pass

Backward pass

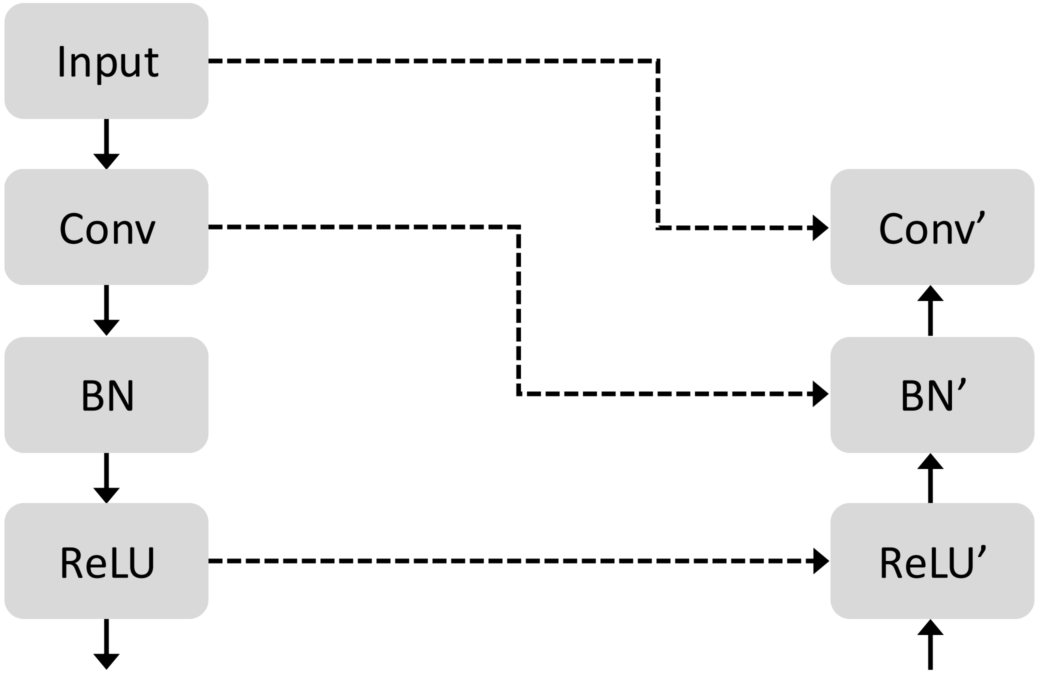
Checkpointing



[1] Jain, Paras, et al. "Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization." *MLSys. 2020.*
[2] Chen, Tianqi, et al. "Training deep nets with sublinear memory cost." *arXiv preprint arXiv:1604.06174* (2016).

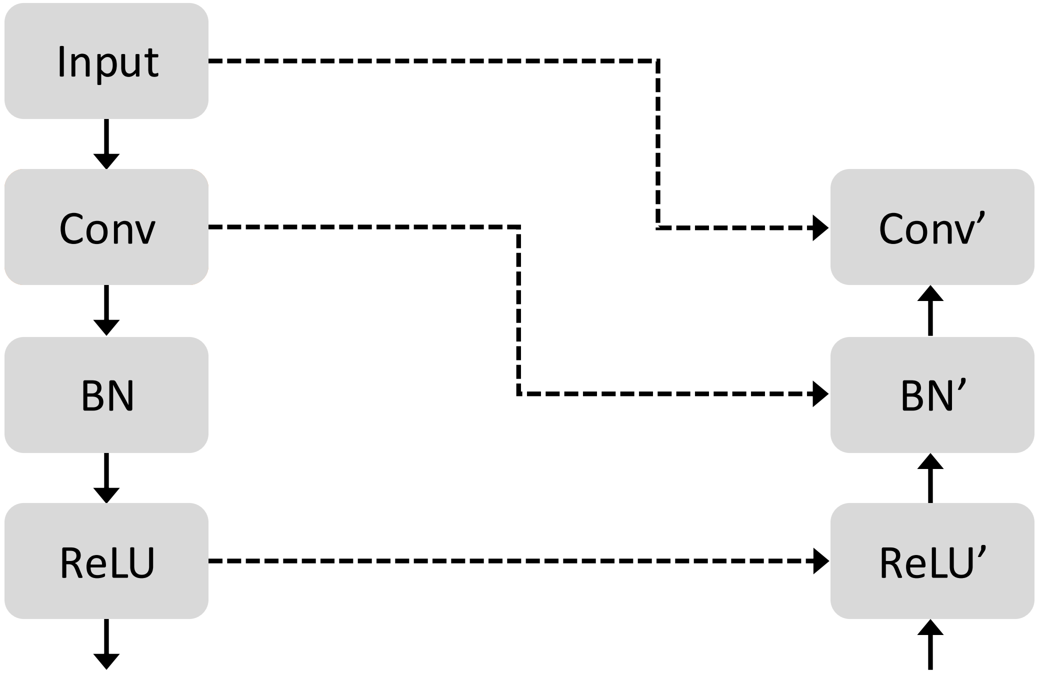
Operator optimizations

Convolution algorithm selection



Operator optimizations

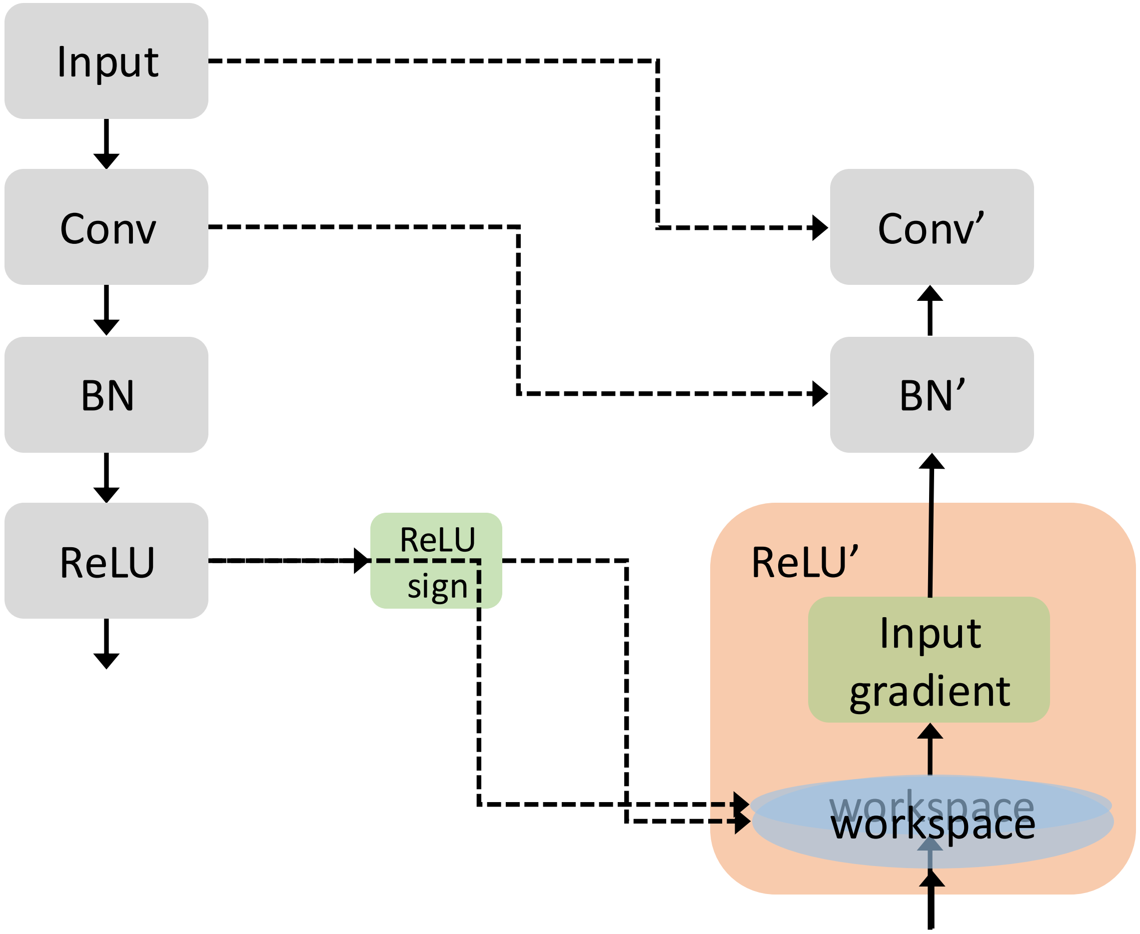
Intermediate-activated ReLU



[1] Jain, Animesh, et al. "Gist: Efficient data encoding for deep neural network training." *ISCA* 2018.

Operator optimizations

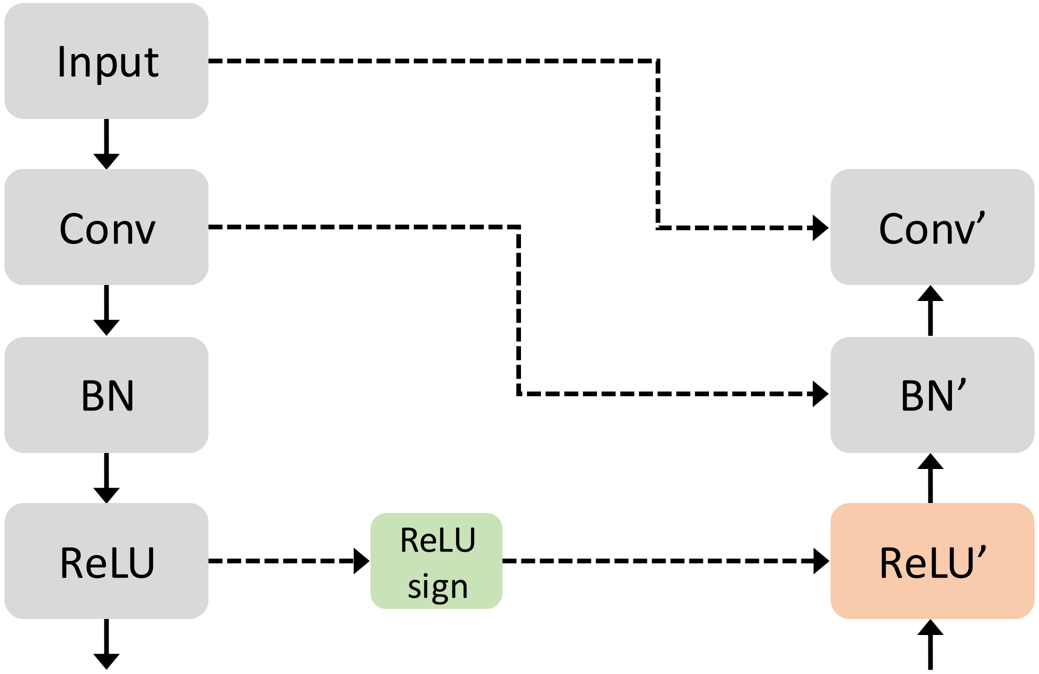
Intermediate-activated ReLU



[1] Jain, Animesh, et al. "Gist: Efficient data encoding for deep neural network training." *ISCA* 2018.

Operator optimizations

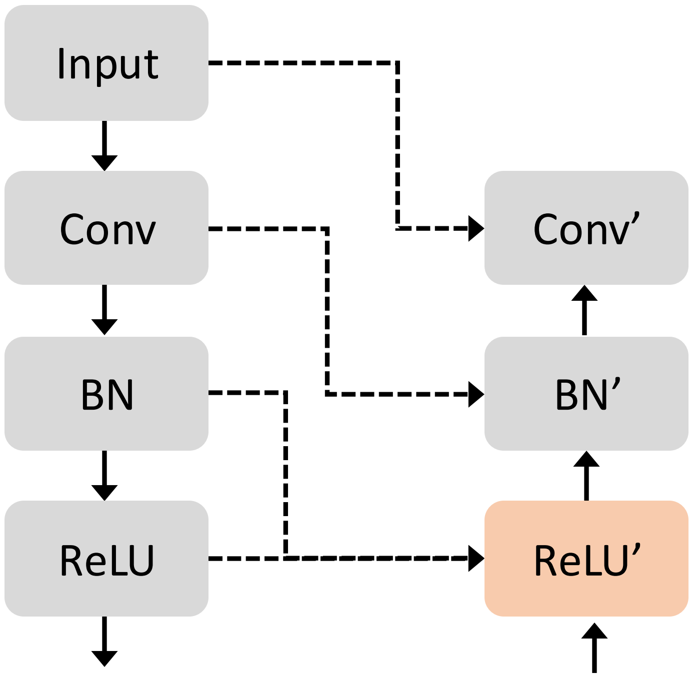
Intermediate-activated ReLU



[1] Jain, Animesh, et al. "Gist: Efficient data encoding for deep neural network training." *ISCA* 2018.

Operator optimizations

Changing backward dependencies

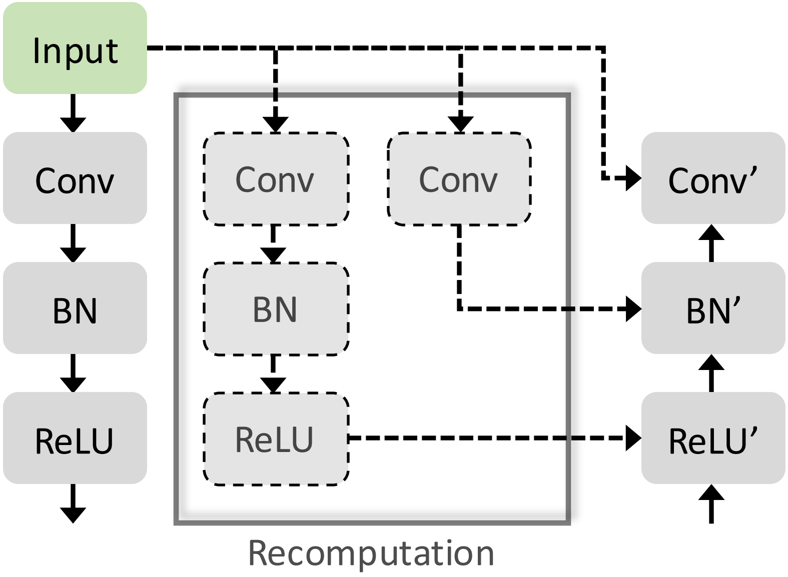


[1] Bulo, Samuel Rota, Lorenzo Porzi, and Peter Kontschieder. "In-place activated batchnorm for memory-optimized training of dnns." *CVPR* 2018.

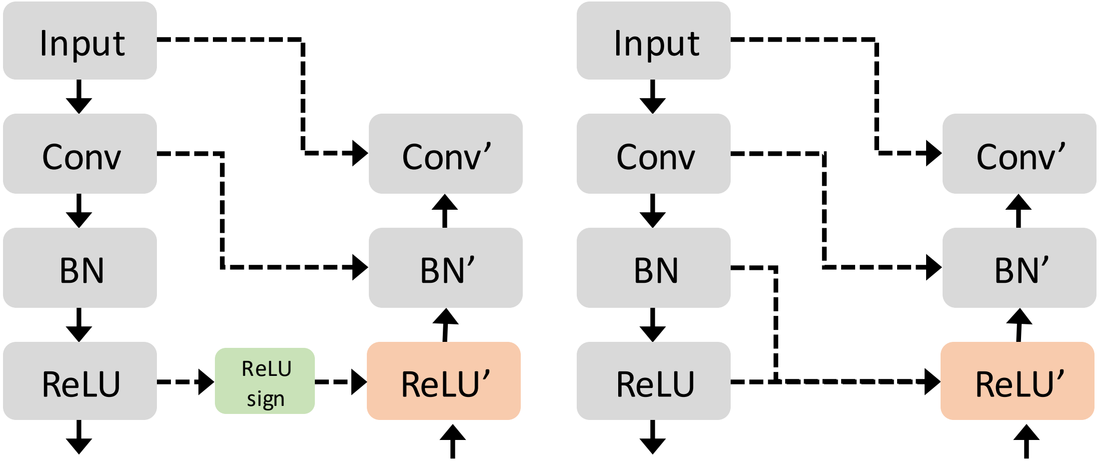
MONeT

Optimizes for minimal memory v/s compute tradeoff

Reduces memory usage by 3x while incurring 9–16% compute overhead over PyTorch for ResNet-50, GoogleNet, etc.

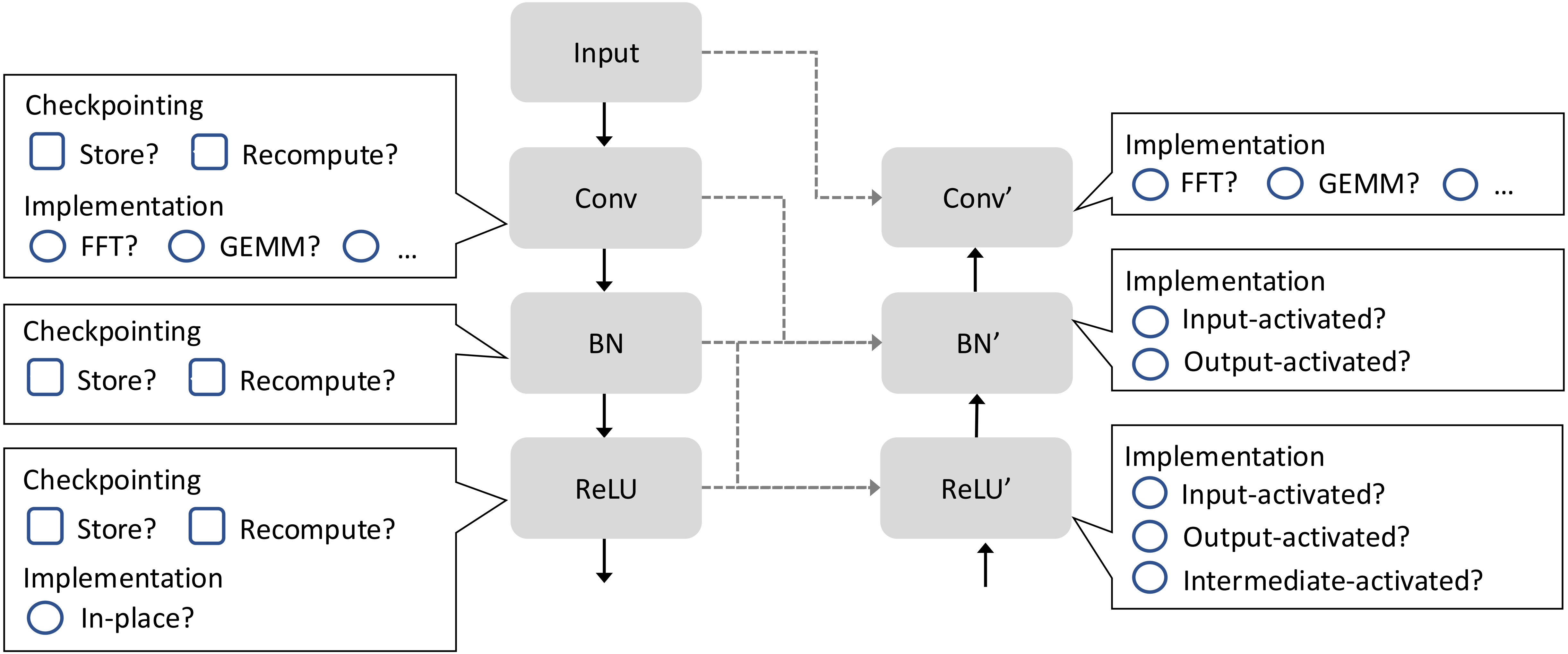


Checkpointing



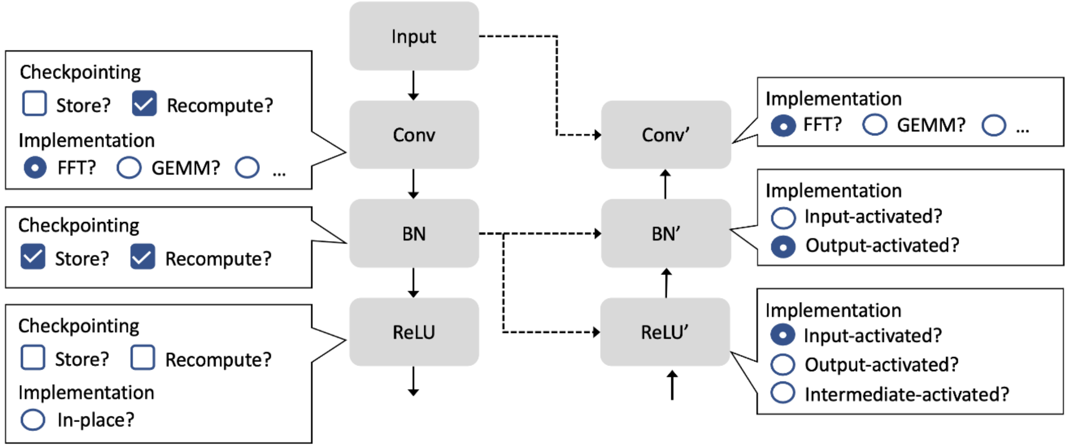
Operator optimizations

MONeT


Input
Conv
BN
ReLU
Conv'
BN'
ReLU'
Checkpointing
Store?
Recompute?
Implementation
FFT?
GEMM?
...
Checkpointing
Store?
Recompute?
Checkpointing
Store?
Recompute?
Implementation
In-place?
Implementation
FFT?
GEMM?
...
Implementation
Input-activated?
Output-activated?
Implementation
Input-activated?
Output-activated?
Intermediate-activated?

MONeT

MONeT formulation



Optimization problem
Execution schedule

Boolean linear programming problem

Objective: Minimize computation time

Memory constraints

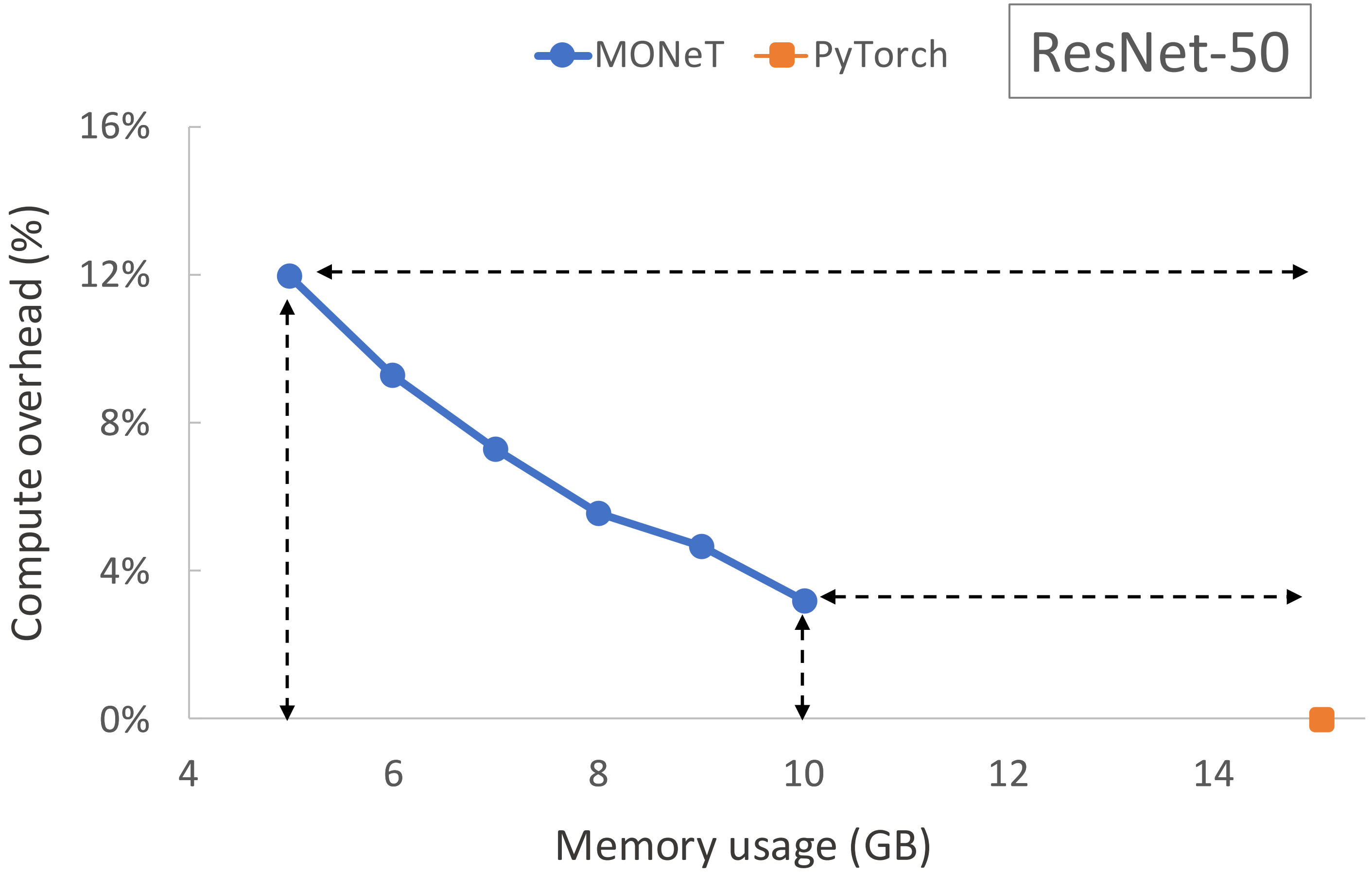
Peak forward memory ≤ Memory budget
Peak backward memory ≤ Memory budget
Peak recomputation memory ≤ Memory budget

Checkpointing constraints

Only use operator output if present in memory

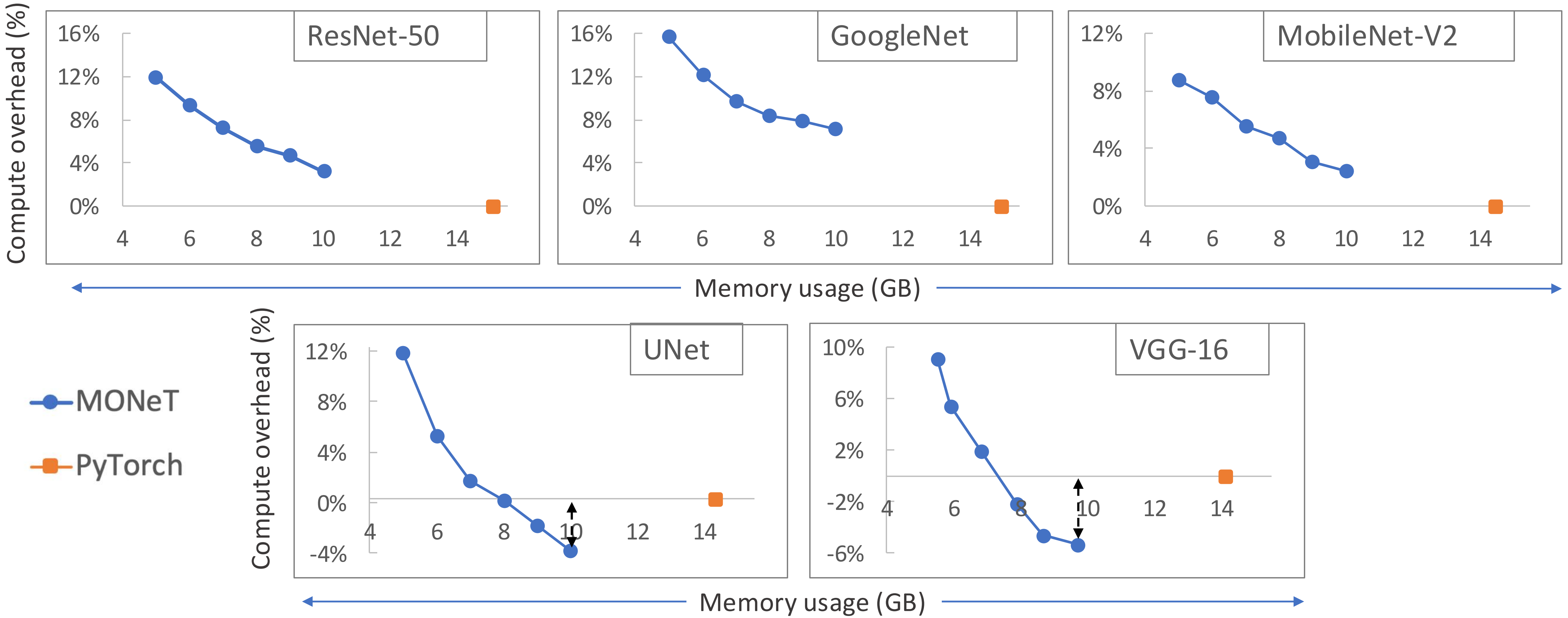
Off-the-shelf solver

Evaluation



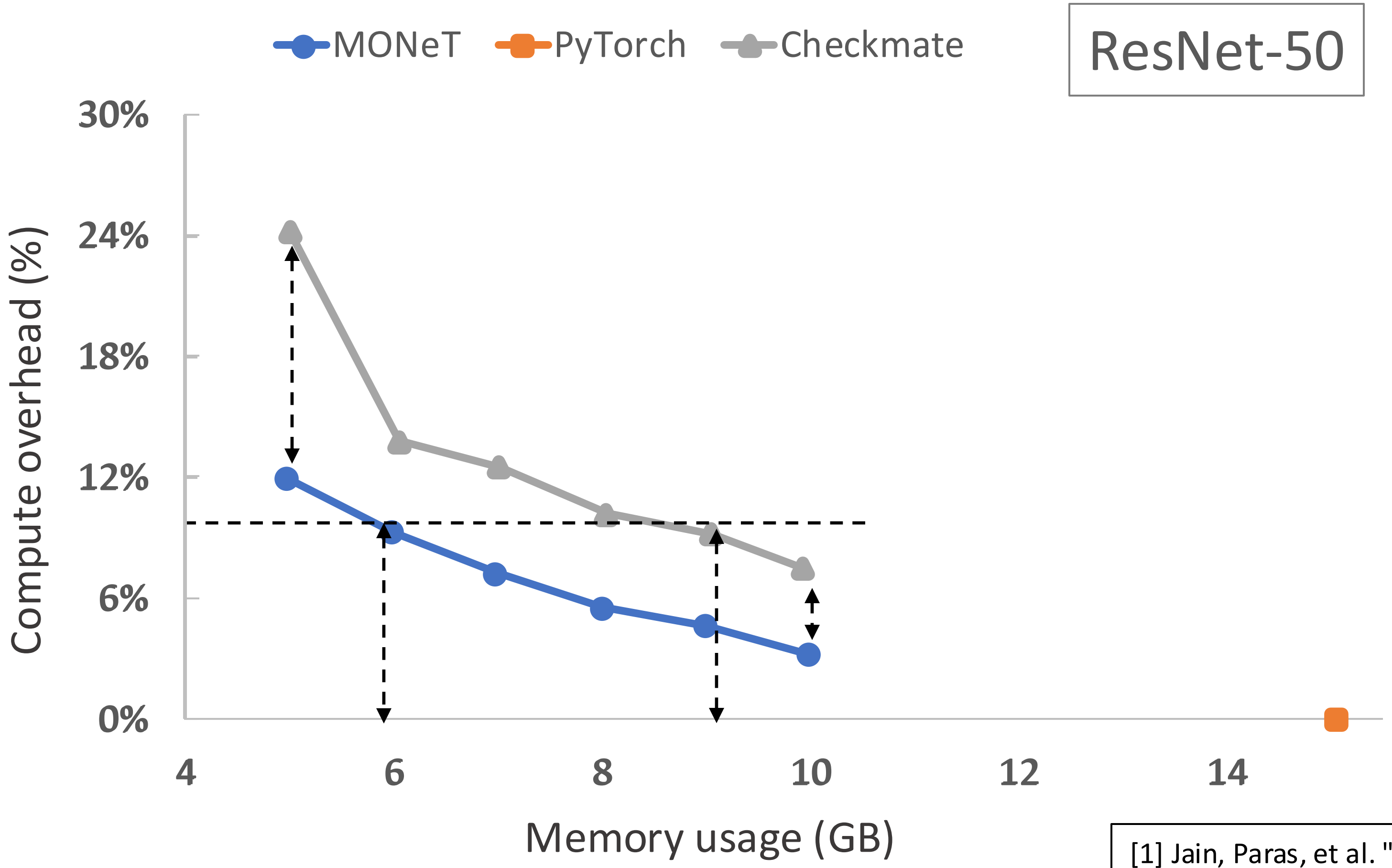
MONeT
PyTorch
ResNet-50
16%
12%
8%
4%
0%
Compute overhead (%)
4
6
8
10
12
14
Memory usage (GB)

Evaluation



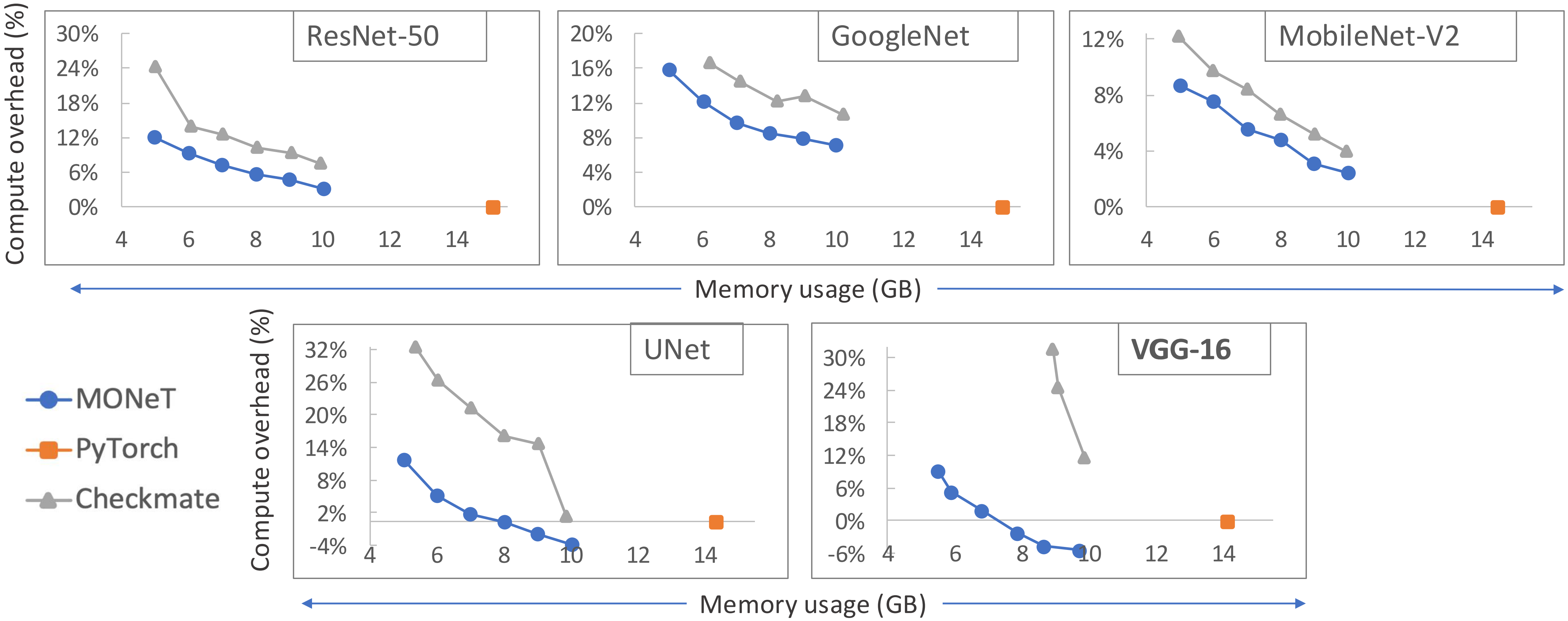
Compute overhead (%)
ResNet-50
16%
12%
8%
4%
0%
4
6
8
10
12
14
GoogleNet
16%
12%
8%
4%
0%
4
6
8
10
12
14
MobileNet-V2
12%
8%
4%
0%
4
6
8
10
12
14
Memory usage (GB)
MONeT
PyTorch
Compute overhead (%)
UNet
12%
8%
4%
0%
-4%
4
6
8
10
12
14
VGG-16
10%
6%
2%
-2%
-6%
4
6
8
10
12
14
Memory usage (GB)

Checkmate

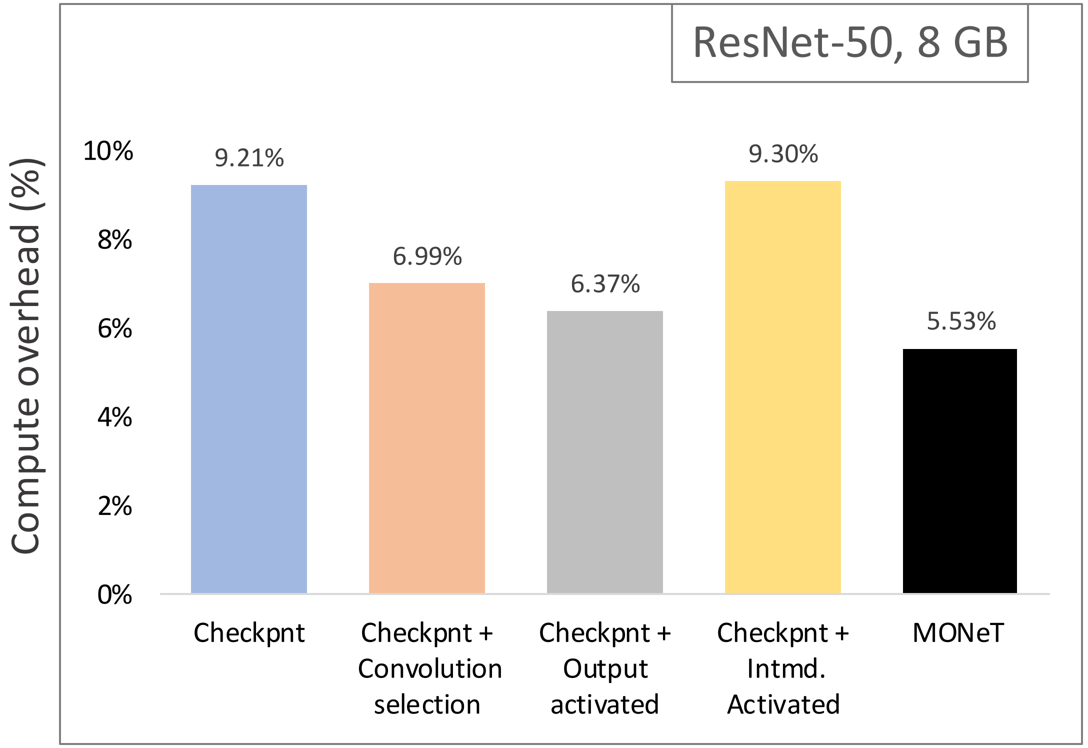


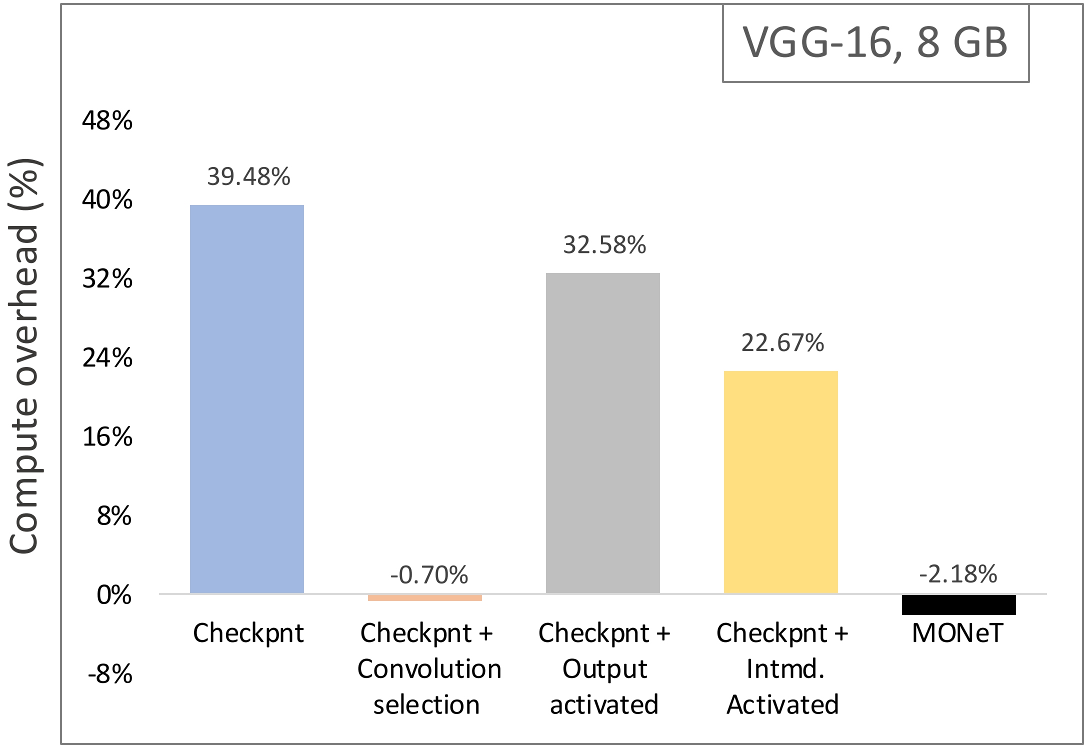
[1] Jain, Paras, et al. "Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization." *MLSys. 2020.*

Checkmate


ResNet-50
GoogleNet
MobileNet-V2
UNet
VGG-16
Compute overhead (%)
Memory usage (GB)
MONeT
PyTorch
Checkmate

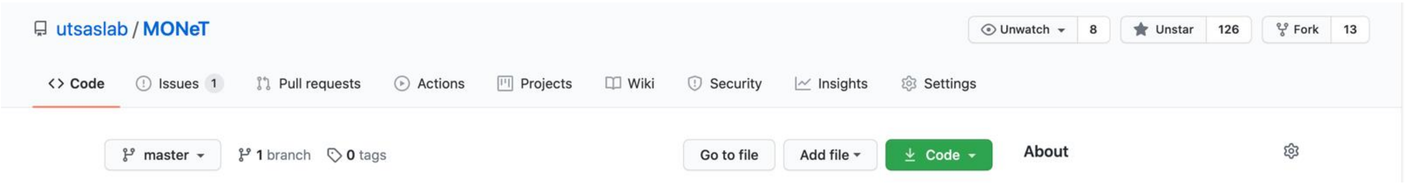
Ablation


ResNet-50, 8 GB
Compute overhead (%)
10%
8%
6%
4%
2%
0%
9.21%
6.99%
6.37%
9.30%
5.53%
Checkpnt
Checkpnt + Convolution selection
Checkpnt + Output activated
Checkpnt + Intmd. Activated
MONeT


VGG-16, 8 GB
Compute overhead (%)
48%
40%
32%
24%
16%
8%
0%
-8%
39.48%
-0.70%
32.58%
22.67%
-2.18%
Checkpnt
Checkpnt + Convolution selection
Checkpnt + Output activated
Checkpnt + Intmd. Activated
MONeT

Try MONeT!

https://github.com/utsaslab/MONeT



Works out-of-box with PyTorch 1.5.1

```
from monet.cvxpy_solver import Solution
from monet.monet_wrapper import MONeTWrapper

monet_model = MONeTWrapper(model, solution_file, input_shape)
```

No need to run a solver, plug in the pre-computed schedules ☺
https://github.com/utsaslab/monet-schedules

Conclusion

MONeT is a framework for memory optimization in training that is automated and creates schedules which jointly optimize global checkpointing and local operator implementations

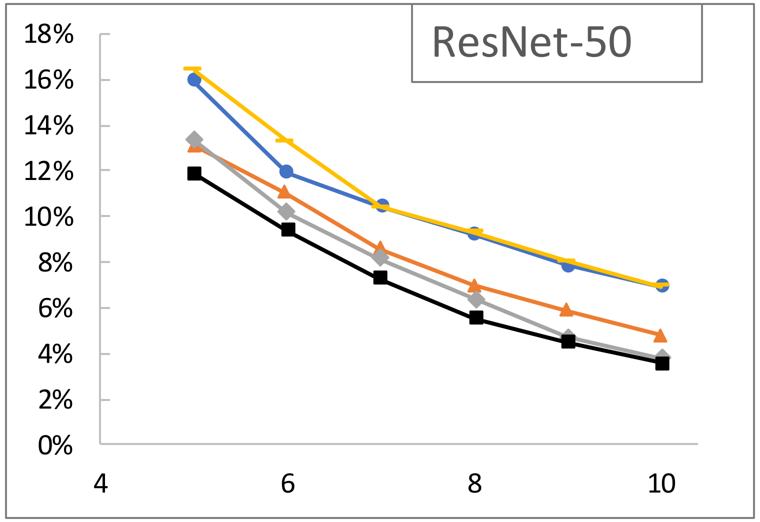
Reduces memory usage by 3x while incurring 9–16% compute overhead over PyTorch for ResNet-50, GoogleNet, etc.

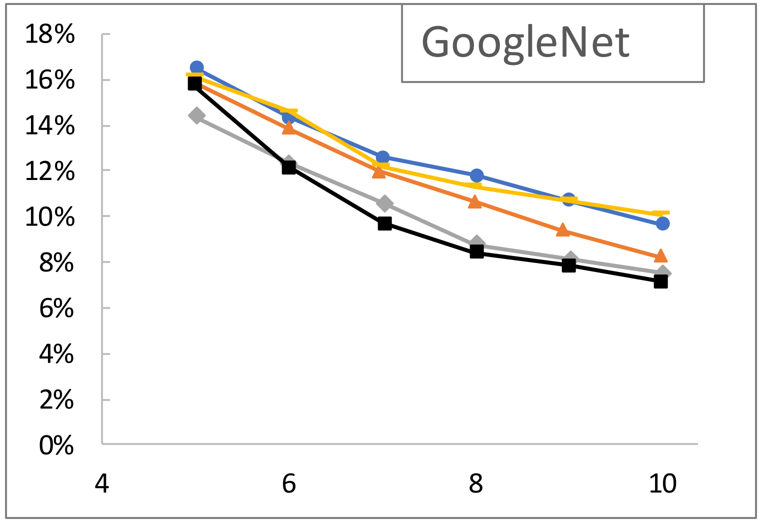
https://github.com/utsaslab/MONeT

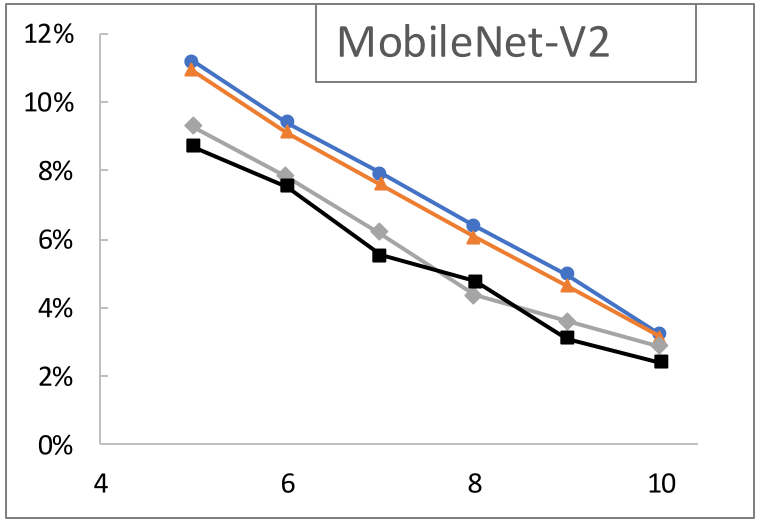
https://github.com/utsaslab/monet-schedules

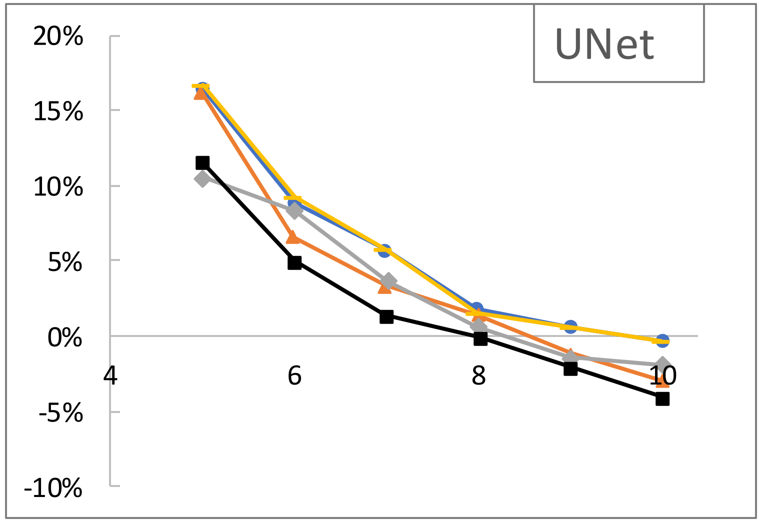
Backup slides

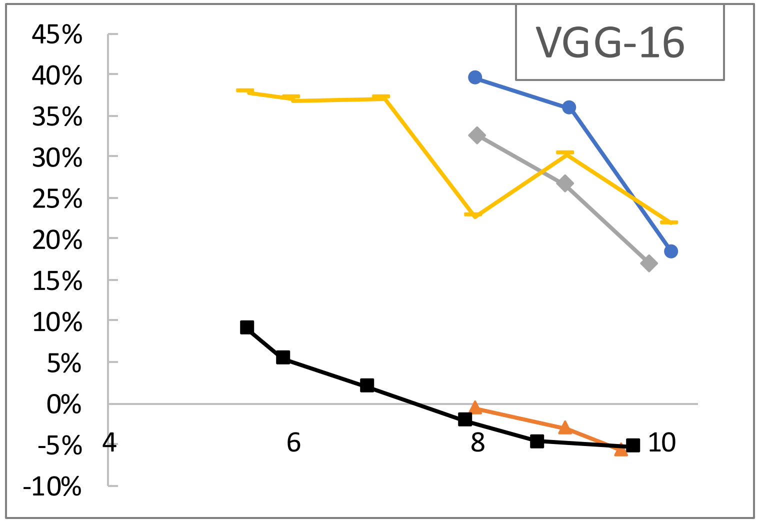
Ablation


ResNet-50
18%
16%
14%
12%
10%
8%
6%
4%
2%
0%
4
6
8
10

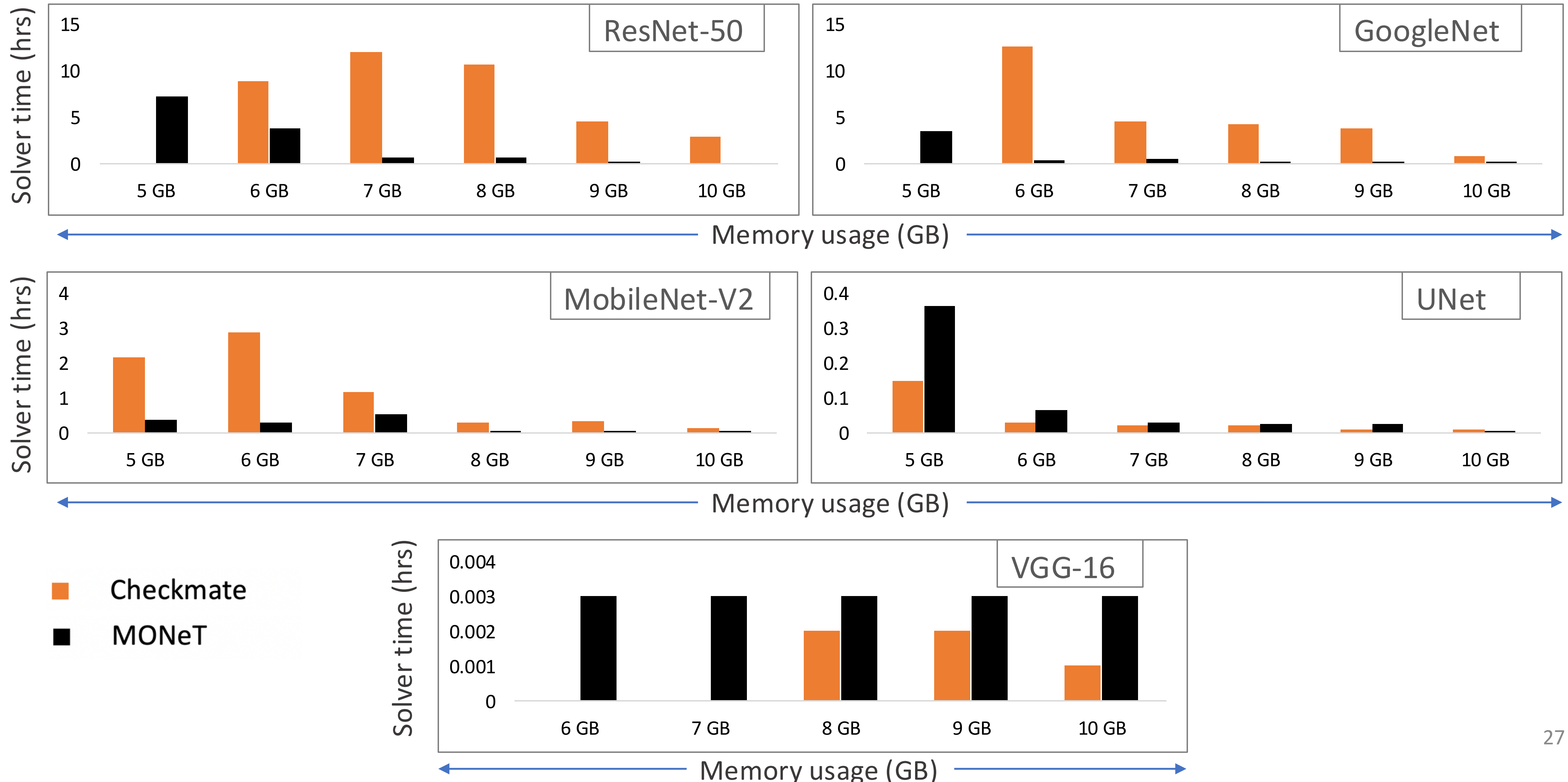

GoogleNet
18%
16%
14%
12%
10%
8%
6%
4%
2%
0%
4
6
8
10


MobileNet-V2
12%
10%
8%
6%
4%
2%
0%
4
6
8
10


UNet
20%
15%
10%
5%
0%
-5%
-10%
4
6
8
10


VGG-16
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
-5%
-10%
4
6
8
10

Solver time



ResNet-50
GoogleNet
MobileNet-V2
UNet
VGG-16
Solver time (hrs)
Memory usage (GB)
15
10
5
0
4
3
2
1
0.4
0.3
0.2
0.1
0.004
0.003
0.002
0.001
5 GB
6 GB
7 GB
8 GB
9 GB
10 GB
Checkmate
MONeT

Directly adding operator optimizations to current frameworks won't scale

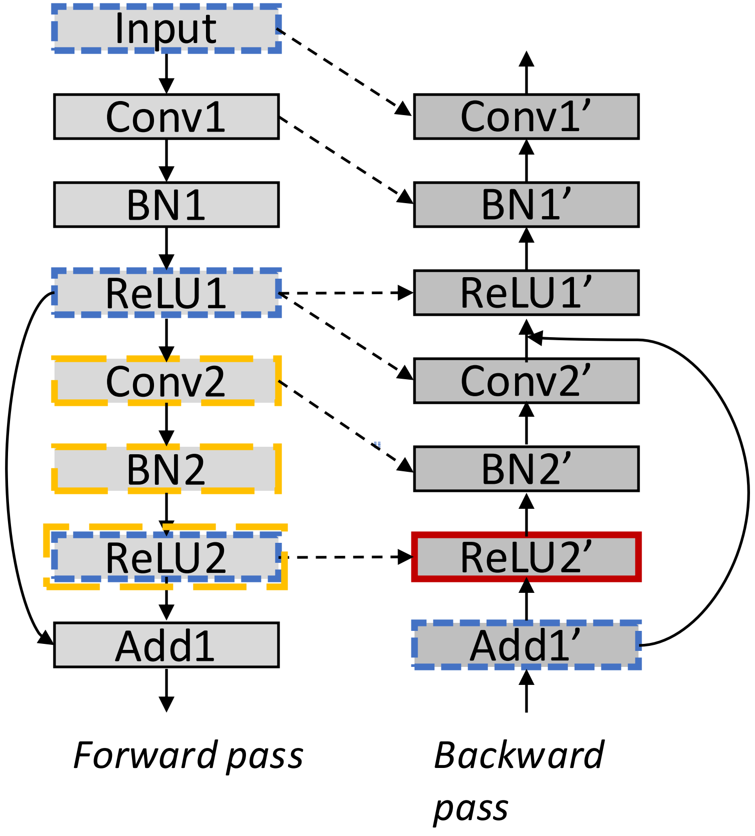
- Directly adding operator optimization in Checkmate formulation will lead to a linear blowup in the number of constraints, and number of auxiliary variables, leading to an at least quadratic expansion on computational costs.

Scope for operator optimization in memory saving

Certain neural network operators can have multiple implementations

1. A convolution can be implemented using at least 8 different algorithms like GEMM, Winograd, FFT, etc. Each algorithm has a different workspace memory requirement and compute usage

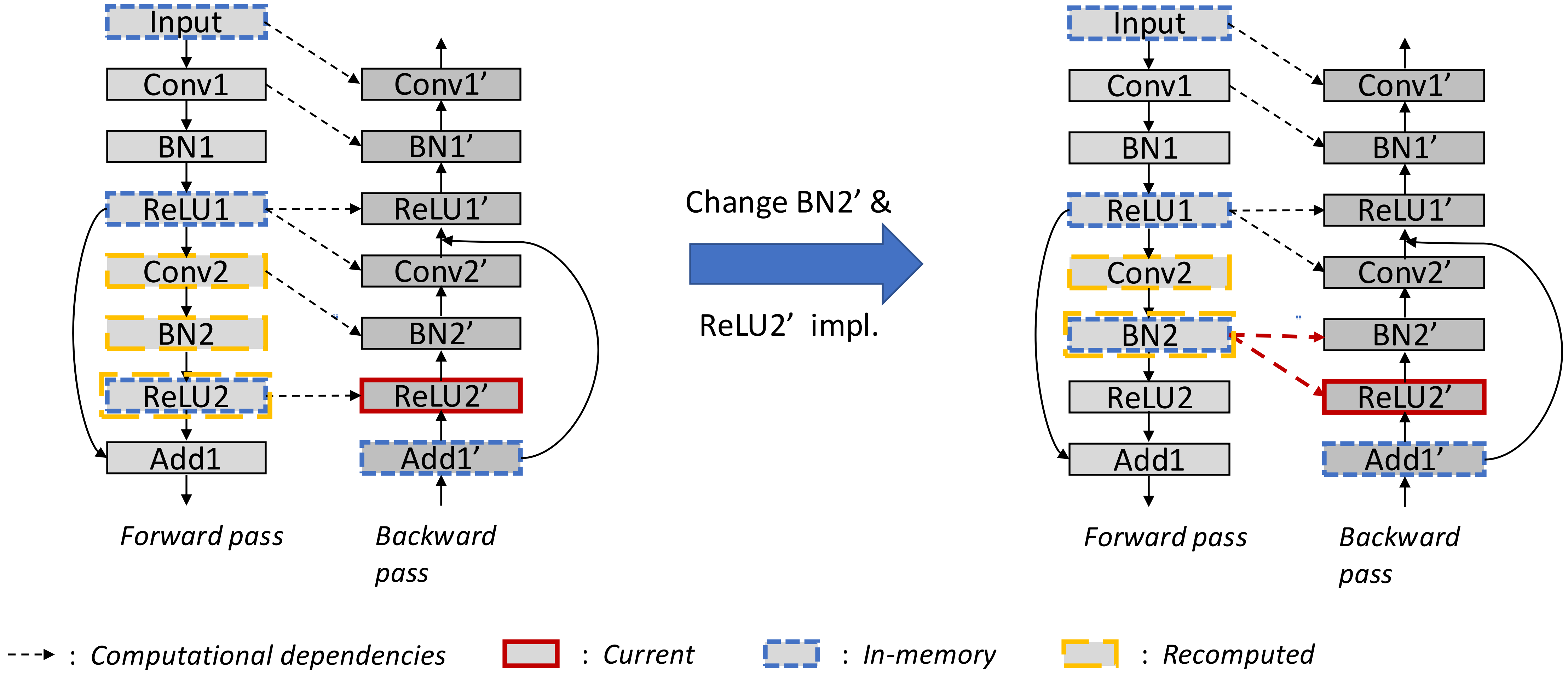
Operator optimizations affect checkpointing



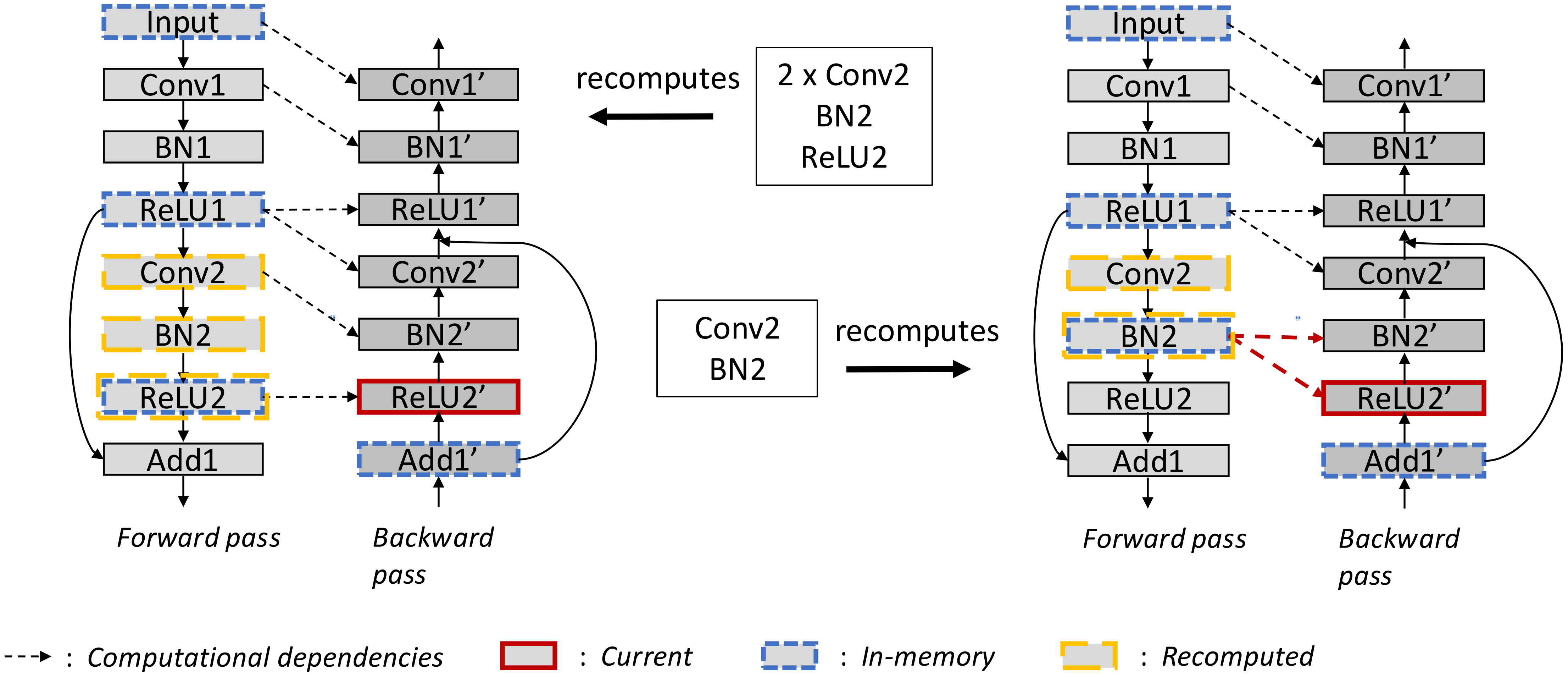
- ReLU2’ : recomputes Conv2, BN2, ReLU2
- BN2’ : recomputes Conv2

- - -▸ : *Computational dependencies* : *Current* : *In-memory* : *Recomputed*

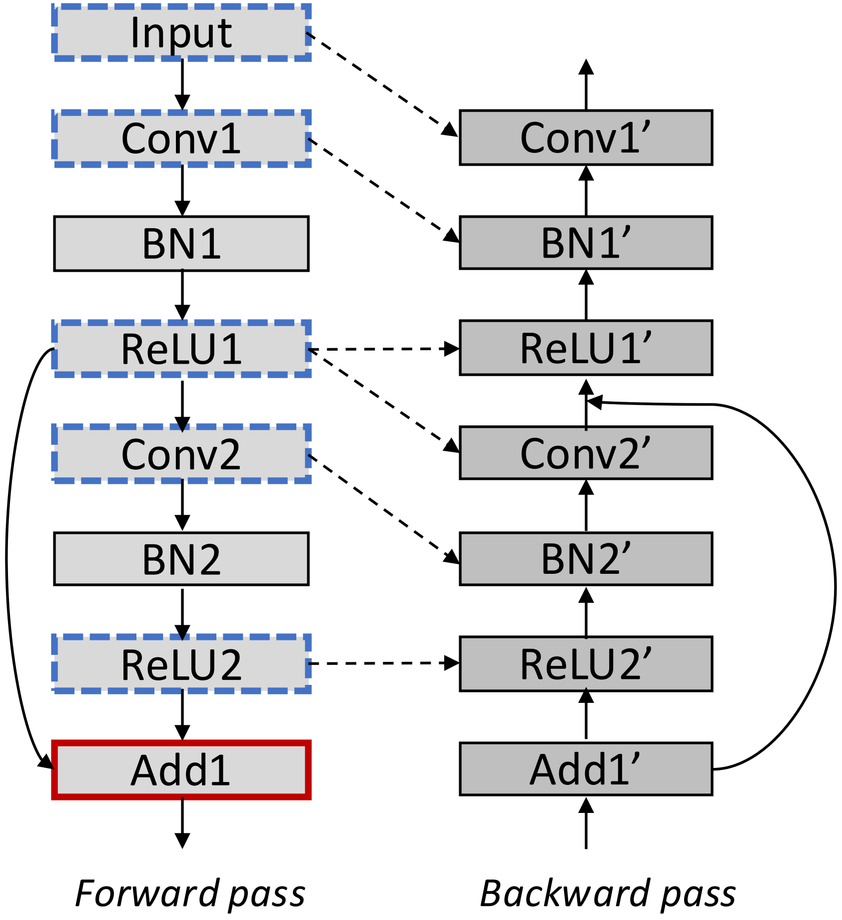
Operator optimizations affect checkpointing



Operator optimizations affect checkpointing



Memory usage during training



Local tensors (L)	ReLU1, ReLU2
Stored tensors (S)	Input, Conv1, ReLU1, Conv2, ReLU2

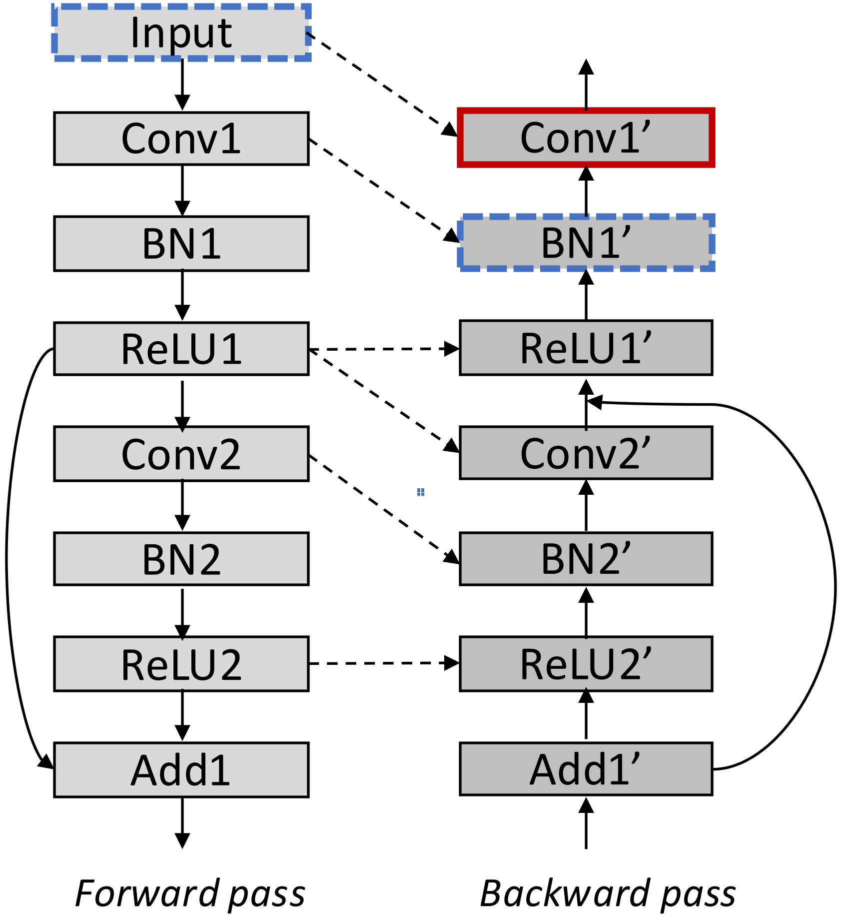
$$\text{Memory} = |\text{ReLU1}| + |ReLU2| + |Input| + |Conv1| + |Conv2| + |Add1| + \text{workspace(Add1)}$$

Forward pass memory of node i :

$$|L_i \cup S_i| + |x_i| + workspace_i$$

- - -▸ : *Computational dependencies*   [red box] : *Current*   [blue dashed box] : *In-memory*

Memory usage during training



Local tensors $(\hat{L})$	BN1'
Stored tensors (S)	Input
Forward deps (D)	Input

$$\text{Memory} = |BN1'| + |Input| + |Conv1'| \; + \text{workspace(Conv1')}$$

Backward pass memory of node k :

$$\left|\widehat{L_k} \cup S_k \cup D_k\right| + |y_k| + workspace_k$$

--→ : *Computational dependencies* ▭ : *Current* ▭ : *In-memory*