Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization

Paras Jain, Ajay Jain, Ani Nrusimha, Amir Gholami,
Pieter Abbeel, Kurt Keutzer, Joseph Gonzalez, Ion Stoica

riselab BAIR

Code and paper: checkmateai.github.io

Email me: parasj@berkeley.edu

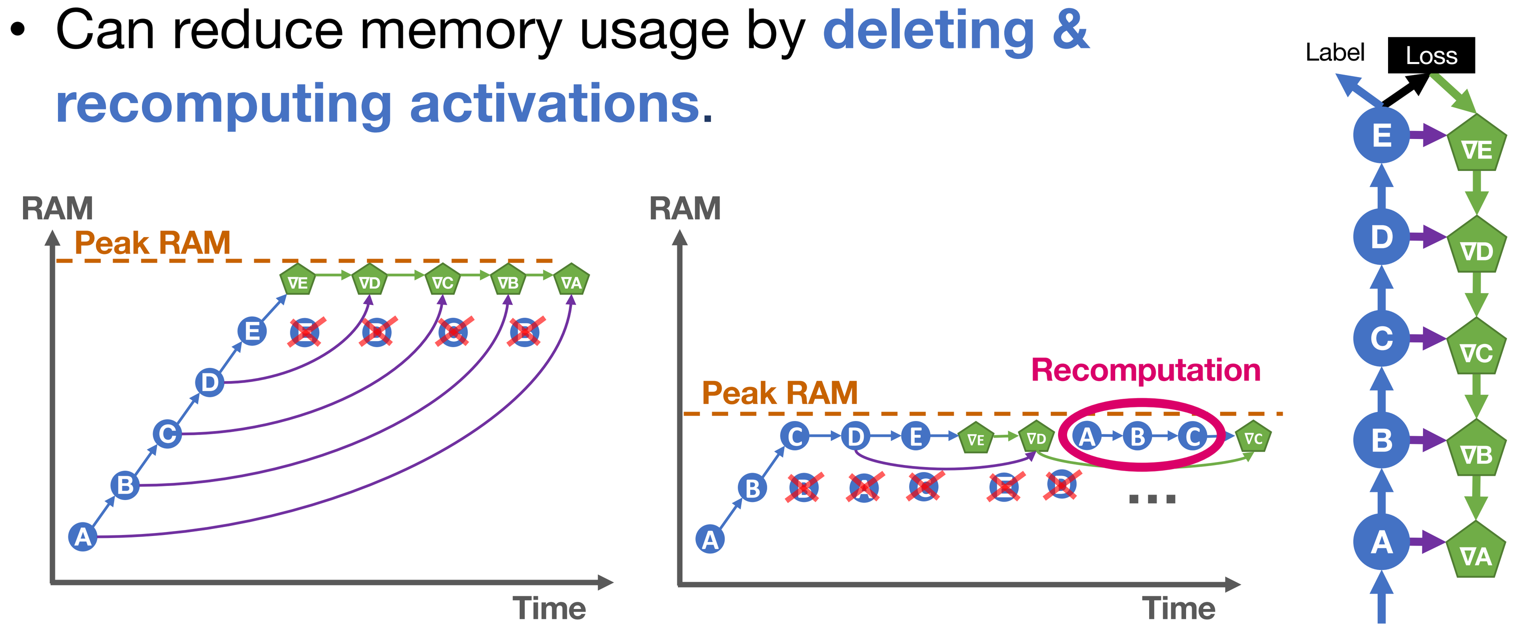
To appear at MLSys 2020

Overview

- **Problem:** Limited memory prevents the development of new deep learning models, but compute is growing quickly.
- We tradeoff memory and compute with an **optimal strategy for arbitrary DNN memory checkpointing.**
- Formulation supports **arbitrary DAGs** and is both **hardware-aware** and **memory-aware.**
- **Up to 5x higher batch sizes, 1.2x speedups.**
- Integration with just **one line of code.**

Backprop space-time tradeoff

- Most memory is used by activations, not parameters.
- Can reduce memory usage by **deleting & recomputing activations.**



- **This work:** How to minimize recomputation while using less than the GPU memory budget?

Why are heuristics suboptimal?

1. Layer runtimes vary

In VGG, 10^7x difference in early and late layer FLOPS.

Checkmate optimizes the evaluation plan using a **per-operation cost model, profiled on the target GPU.**

2. Layer RAM usages vary

Layers significantly differ in memory usage.

Our linear program accounts for & **constrains peak memory usage** at all points in time, using statically known memory consumptions.

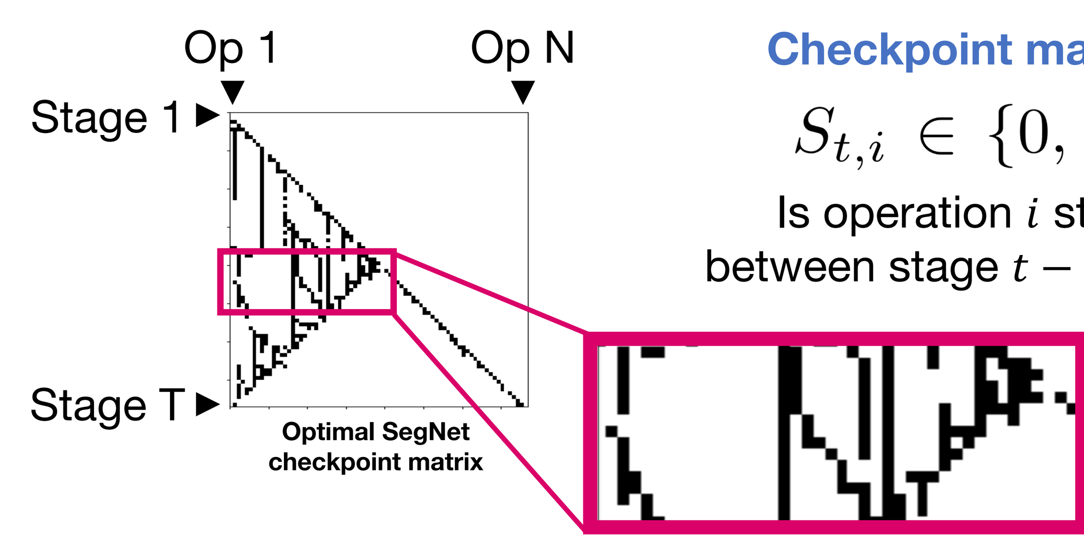
3. Real DNNs are non-linear

What to checkpoint with skip connections, multi-tower architectures etc?

Checkmate **traces fwd & bwd graph** and **constructs optimization problem using graph structure + flexible search space.**

Representing a schedule

For flexibility, unroll schedule into *stages.*
Separately model checkpoints (S) and computations (R).



Checkpoint matrix

$S_{t,i} \in \{0,1\}$

Is operation i stored between stage $t-1$ and t?

Prior work: Inflexible single stage, checkpoint for life

Checkmate: Delete & recreate checkpoints up to T−1 times

Computation matrix: Is operation i computed in stage t?

Space-time schedule repr. generalizes checkpointing.

→ Fine-grained control of evaluation + GC.

Rematerialization ILP

$$\underset{R,S,U,\text{FREE}}{\arg\min} \sum_{t=1}^{n}\sum_{i=1}^{t} C_i R_{t,i}$$ ← **Find the lowest cost schedule**

subject to

$$R_{t,j} \le R_{t,i} + S_{t,i}$$
$$S_{t+1,i} \le R_{t,i} + S_{t,i}$$ ← **which is valid (dependencies resident),**

$$R_{t,t} = 1$$

$$U_{t,i} \le \text{budget}$$ ← **and has constrained memory usage.**

$$R,S,U \in \{0,1\}^{n\times n}$$

For tractability, each stage is frontier-advancing:
→ Op i evaluated in stage i for the first time.
→ From 9 hr to 1.18 sec for certifiable optimality.

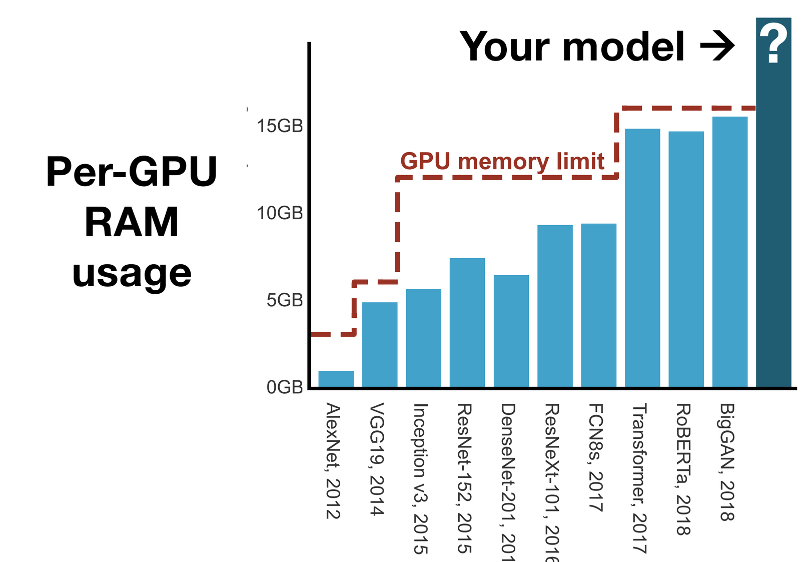
Model memory usage in each stage with recurrence.

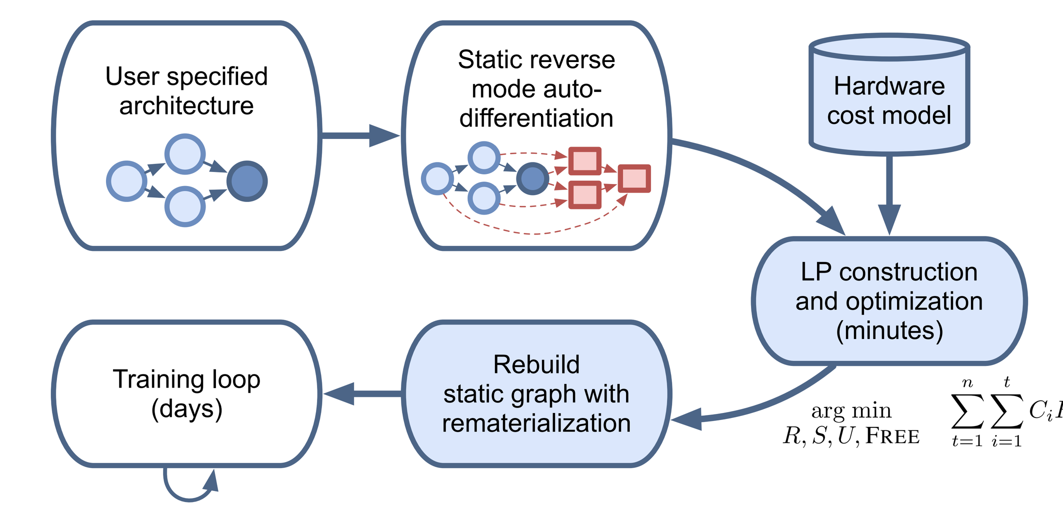
$$U_{t,0} = \boxed{\textstyle\sum_i M_i S_{t,i}}$$ **Start of stage: Checkpoints use memory**

$$U_{t,k+1} = U_{t,k} - \underbrace{\boxed{\textstyle\sum_i M_i * \text{FREE}_{t,i,k}}}_{\text{Garbage collection}} + \boxed{M_{k+1} R_{t,k+1}}$$ **Temporary value**

Optimal $\mathbf{R}$, $\mathbf{U}$, and FREE easy to compute given $\mathbf{S}$.
→ "Two-phase" rounding approximation works well.

Creating new applications with Checkmate



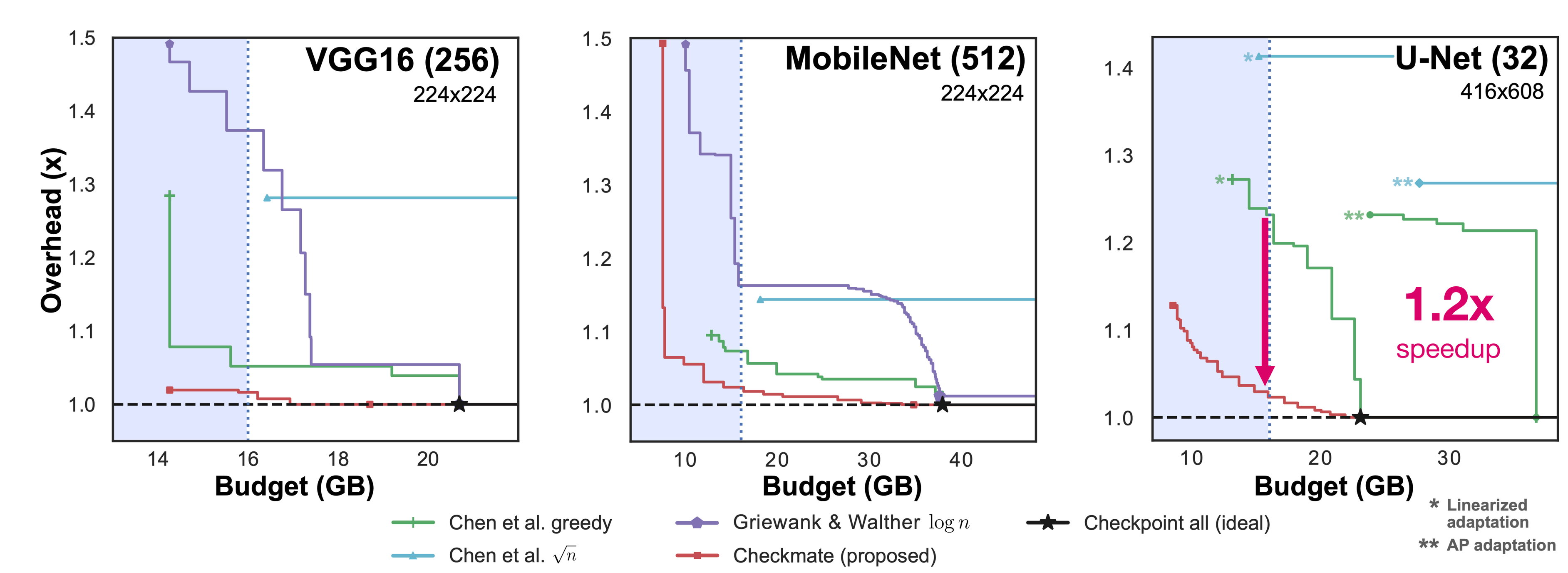


One line of code for memory-efficient deep learning!

```
train_iteration = checkmate.compile_tf2(
    model, loss, optimizer, input_shape, label_shape)
for epoch in range(100):
    for images, labels in dataset:
        predictions, loss = train_iteration(images, labels)
```

Evaluation

- TF 2.0 / Keras Image classification & semantic segmentation architectures.
- Checkmate achieves **up to 1.2x speedup** over our best baseline heuristic and finds schedules with the lowest memory usages.



- Maximize batch size as proxy for resolution, model depth etc.
- With +1x overhead cap, Checkmate supports **up to 5.1x larger batch sizes.**