**Checkmate**: Breaking the Memory Wall with Optimal Tensor Rematerialization

Paras Jain

*Joint work with:* Ajay Jain, Ani Nrusimha, Amir Gholami, Pieter Abbeel, Kurt Keutzer, Ion Stoica, Joseph Gonzalez
BigGAN (2018)
Image generation
Brock et al. 2019

VideoBERT (2019)
Video generation
Sun et al. 2019

GPT-2 (2019)
Text generation
Radford et al. 2019

SYSTEM PROMPT (HUMAN-WRITTEN)
In a shocking finding, scientists discovered a herd of unicorns living in a remote, previously unexplored valley in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)
The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several
Emerging trend:

Rapid growth in model size
State-of-the-art models have hit a **memory capacity wall**.

Limited GPU memory is slowing progress in new deep learning models!

Cited memory as limiting factor:
- Chen et al. 2016
- Gomez et al. 2017
- Pohlen et al. 2017
- Liu et al. 2019
- Dai et al. 2019
- Child et al. 2019
Problem:

How do we efficiently train large models beyond memory limits?

State-of-the-art models have hit a memory capacity wall.

Limited GPU memory is slowing progress in new deep learning models!
Compute is outstripping DRAM capacity growth
Backprop is optimized for **compute efficiency**, not **RAM usage**
**Ideal:** scalable algorithm for backprop that adapts to RAM constraints
This work: optimal space-time tradeoff for backpropagation

Checkmate explores optimal trade-off
5x larger inputs w/ 2x cost

RAM

Compute

Compute-optimized backprop

RAM-optimized backprop
RAM-hungry backprop policy

Keep all layers in RAM

RAM

Compute-optimized backprop

Compute
RAM-hungry backpropagation policy

Keep all layers in RAM
RAM-hungry backpropagation policy

Keep all layers in RAM
RAM-hungry backpropagation policy

Keep all layers in RAM

Forward Pass

Backward Pass

RAM used

Time

RAM-hungry backpropagation policy

Keep all layers in RAM
RAM-hungry backpropagation policy
Keep all layers in RAM

Forward Pass
A → B → C → D → E

Backward Pass
\(\nabla A\) → \(\nabla B\) → \(\nabla C\) → \(\nabla D\) → \(\nabla E\)

RAM used

Peak RAM

Time
RAM-optimized backpropagation policy
Recompute all layers as needed
How can we use less memory?
Free early & recomputate
RAM-optimized backpropagation policy
Recompute **all layers**

How can we use less memory?
**Free early & recompute**
RAM-optimized backpropagation policy
Recompute all layers

How can we use less memory?
Free early & recompute
RAM-optimized backpropagation policy

Recompute **all layers**

**How can we use less memory?**

**Free early & recompute**
How to choose which layers to recompute?
How to choose which layers to recompute?
How to choose which layers to recompute?

Forward Pass

Backward Pass

**Compute:** $O(n)$ additional overhead

**RAM:** $O(\sqrt{n})$ RAM usage
Challenges of heuristics:

1. Variable runtime per layer

$10^6 \times$ slower
Challenges of heuristics:

1. Variable runtime per layer

2. Variable RAM usage per layer

$10^3 \times$ more RAM
Challenges of heuristics:

1. Variable runtime per layer
2. Variable RAM usage per layer
3. Real DNNs are non-linear
Prior work is suboptimal in general setting!

**Greedy heuristic**
[Chen 2016]
[XLA authors 2017, 2020]

**Divide-and-conquer heuristic**
[Griewank 2000]
[Kowarz 2006]
[Siskind 2018]
[Kumar 2019]

**Optimal for specific architecture**
[Gruslys 2016]
[Feng 2018]
[Beaumont 2019]

**Challenges:**

1. Variable runtime per layer
2. Variable RAM usage per layer
3. Real DNNs are non-linear
Can we optimally trade-off RAM for compute?

Let’s be:

1. Hardware-aware
2. RAM-aware
3. DAG flexibility

Checkpoint every node
Recompute all layers
Checkmate
A system for **optimal** tensor rematerialization

- Hardware + RAM aware
- Solve for 10s-1hr
  - Train for 1mo
- GPU, CPU, TPU support

- Accurate cost model
- Optimal solver
  - Integer Linear Program
- Near-optimal solver
  - Two phase rounding
- Flexible search space
- Graph rewrite

GPU, CPU, TPU support

Solve for 10s-1hr
- Train for 1mo

Hardware + RAM aware
- Accurate cost model
- Flexible search space
- Graph rewrite

Optimal solver
- Integer Linear Program
- Near-optimal solver
  - Two phase rounding
A system for **optimal** tensor rematerialization

- Accurate cost model
- Flexible search space
- Optimal solver
  - Integer Linear Program
- Near-optimal solver
  - Two phase rounding
- Graph rewrite
Checkmate
A system for **optimal** tensor rematerialization
Checkmate
A system for **optimal** tensor rematerialization

Stage

Layer

$R_{t,i} \in \{0, 1\}$
Checkmate

A system for **optimal** tensor rematerialization
Checkmate
A system for **optimal** tensor rematerialization

Stage

Layer

$t=1$

$t=2$

$t=3$

$t=4$

$t=5$

$t=6$

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

$R_{t,i} \in \{0, 1\}$
Checkmate
A system for **optimal** tensor rematerialization

Stage

Layer

$t=1$

$t=2$

$t=3$

$t=4$

$t=5$

$t=6$

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

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

S = What is in memory?

R = What is computed?
A system for optimal tensor rematerialization

\[ R \in \{0, 1\} \]

Stage

Layer

Example of optimal “S” (SegNet)
A system for optimal tensor rematerialization

Checkmate

Optimal solver
Integer Linear Program

Accurate cost model

Flexible search space

Near-optimal solver
Two phase rounding

Graph rewrite
Checkmate
A system for **optimal** tensor rematerialization

Use R matrix to create linear objective

\[
\min_{S,R,U} \sum \sum C_i R_{t,i}
\]

Minimize forward + backward cost

**Decision variables**

- \( S_{t,i} \in \{0, 1\} \) Layer \( i \) stored for stage \( t \)
- \( R_{t,i} \in \{0, 1\} \) Layer \( i \) (re)computed in stage \( t \)
Checkmate

A system for **optimal** tensor rematerialization

Minimize forward + backward cost

$$\min_{S,R,U} \sum \sum C_i R_{t,i}$$

Correctness

- $R_{t,j} \leq R_{t,i} + S_{t,i}$
- $S_{t,i} \leq R_{t-1,i} + S_{t-1,i}$

Decision variables

- $S_{t,i} \in \{0, 1\}$ Layer $i$ stored for stage $t$
- $R_{t,i} \in \{0, 1\}$ Layer $i$ (re)computed in stage $t$

“A layer’s dependencies must be computed before evaluation”

“A layer must be computed before it can be stored in RAM”
A system for **optimal** tensor rematerialization

Minimize forward + backward cost

\[
\min_{S, R, U} \sum \sum C_i R_{t,i}
\]

Decision variables

- \( S_{t,i} \in \{0, 1\} \): Layer \( i \) stored for stage \( t \)
- \( R_{t,i} \in \{0, 1\} \): Layer \( i \) (re)computed in stage \( t \)
- \( U_{t,i} \in \mathbb{R}_+ \): Memory usage in stage \( t \)

Correctness

- \( R_{t,j} \leq R_{t,i} + S_{t,i} \)
- \( S_{t,i} \leq R_{t-1,i} + S_{t-1,i} \)

Memory limit

- \( U_{t,k} \leq \text{budget, ...} \)

Constrain memory via an implicit variable to model memory usage at each stage
A system for optimal tensor rematerialization

Minimize forward + backward cost

\[
\min_{S,R,U} \sum \sum C_i R_{t,i}
\]

Correctness

\[
R_{t,j} \leq R_{t,i} + S_{t,i}
\]

\[
S_{t,i} \leq R_{t-1,i} + S_{t-1,i}
\]

Memory limit

\[
U_{t,k} \leq \text{budget}, \ldots
\]

Constrain memory at all execution steps

\[
U_{t,k} \leq \text{budget}
\]

More checkpoints \(\rightarrow\) more initial memory

\[
U_{t,0} = \sum M_i S_{t,i}
\]

\[
U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}
\]

Memory accounting details in paper

Memory limit

Correctness

Constrain memory to model memory usage at each stage
Checkmate
A system for **optimal** tensor rematerialization

Minimize forward + backward cost

$$\min_{S,R,U} \sum \sum C_i R_{t,i}$$

Correctness

$$R_{t,j} \leq R_{t,i} + S_{t,i}$$
$$S_{t,i} \leq R_{t-1,i} + S_{t-1,i}$$

Memory limit

$$U_{t,k} \leq \text{budget}, \ldots$$

How long is the solve time?

9 hours 😳
A system for **optimal** tensor rematerialization

Minimize forward + backward cost

\[
\min_{S,R,U} \sum \sum C_i R_{t,i}
\]

**Correctness**

\[
R_{t,j} \leq R_{t,i} + S_{t,i}
\]

\[
S_{t,i} \leq R_{t-1,i} + S_{t-1,i}
\]

**Memory limit**

\[
U_{t,k} \leq \text{budget, ...}
\]

**Partition schedule into frontier-advancing stages**

9 hours → 0.2 seconds

**Tractability**

\[
R_{t,t} = 1
\]

\[
R, S, U \text{ lower triangular}
\]

**Prunes n! permutations of nodes**
Checkmate

A system for **optimal** tensor rematerialization

- Accurate cost model
- Flexible search space
- Optimal solver
  - Integer Linear Program
- Near-optimal solver
  - Two phase rounding
- Graph rewrite
ILP optimization is NP-hard (combinatorial search)

Polynomial-time approximation?

1. Relax boolean constraints
2. Solve LP
3. Round solution

How to maintain feasibility?

Insight: Given S, optimal R easy to compute

Proposed method: Two-Phase Rounding
Round S, solve other variables optimally
Evaluation: Questions

1. What is the memory vs compute trade-off?

2. How much can we increase batch/model size?

3. How well does two-phase rounding do?
Evaluation: What is the memory vs compute trade-off?

- **U-Net, batch size 32**
  - Best heuristic
  - Checkmate
  - 1.2x speedup!

- **MobileNet, batch size 512**
  - Best heuristic
  - Checkmate

Budget (GB)

GPU Memory available (GB)
Evaluation: How much can we increase batch size?

VGG19
224x224 images

No rematerialization
Batch size 167

Square root heuristic
Batch size 197

Checkmate
Batch size 289

1.18x larger!
1.73x larger! 10 sec solve
Evaluation: How much can we increase batch size?

- U-Net
- FCN8
- SegNet
- VGG19
- ResNet50
- MobileNet

Checkpoint all
AP $\sqrt{n}$
Lin. greedy
Checkmate (ours)

- 1.73x larger!
- 5.1x larger!
Evaluation: How much can we increase batch size?

*Ongoing work: BERT
2.3x larger batch size over TF2.0
Train BERT-Large w/o model parallelism
## Evaluation: How well does 2P rounding approximate ILP?

<table>
<thead>
<tr>
<th>Model</th>
<th>AP $\sqrt{n}$</th>
<th>AP greedy</th>
<th>Griewank log $n$</th>
<th>Two-phase LP rounding</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>1.14×</td>
<td>1.07×</td>
<td>7.07×</td>
<td>1.06×</td>
</tr>
<tr>
<td>VGG16</td>
<td>1.28×</td>
<td>1.06×</td>
<td>1.44×</td>
<td>1.01×</td>
</tr>
<tr>
<td>VGG19</td>
<td>1.54×</td>
<td>1.39×</td>
<td>1.75×</td>
<td>1.00×</td>
</tr>
<tr>
<td>U-Net</td>
<td>1.27×</td>
<td>1.23×</td>
<td>-</td>
<td>1.03×</td>
</tr>
<tr>
<td>ResNet50</td>
<td>1.20×</td>
<td>1.25×</td>
<td>-</td>
<td>1.05×</td>
</tr>
</tbody>
</table>

**Within 6% of optimal cost** (geomean)

- 43x speedup for ResNet50
- 440x speedup for MobileNet
Key ideas:

• GPU memory limits are preventing the development of new deep learning models.
• We present the first general solution for optimal & near-optimal graph rematerialization.
• Formulation supports arbitrary DAGs and is both hardware-aware and memory-aware.
• Integration with just one line of code:

```python
train_iteration = checkmate.compile_tf2(
    model, loss, optimizer, input_shape, label_shape)
```