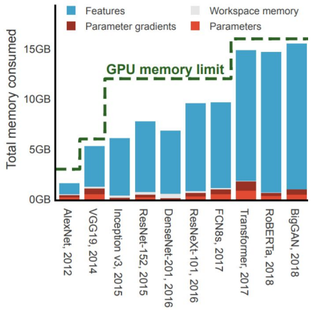
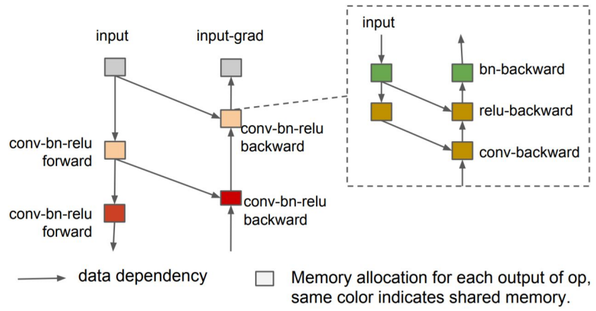
Dynamic Tensor Rematerialization

Marisa Kirisame*, Steven Lyubomirsky*, Altan Haan*, Jennifer Brennan, Mike He, Jared Roesch, Tianqi Chen, Zachary Tatlock

Checkpointing: Trade Time for Space

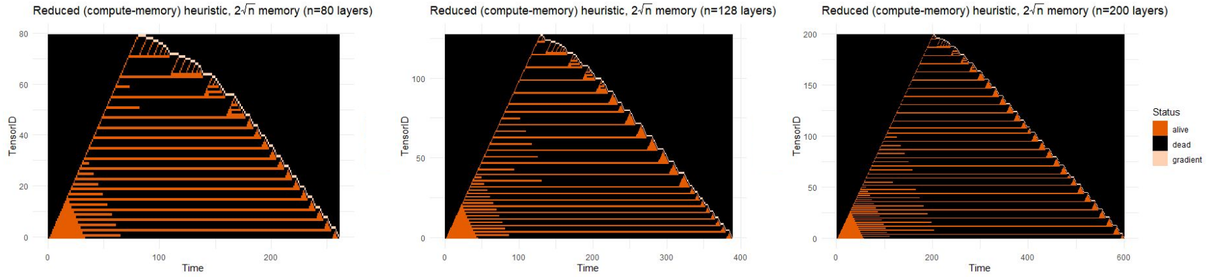


"The memory wall" (Jain *et al.*, 2020)



Past approaches: Static plan (Chen *et al.*, 2016)

Theoretical Results



Train N-layer FF network in Ω(√N) memory and O(N) operations! No static planning!

DTR: Checkpointing is Caching

```
PerformOp(op, args):
  Note: Performs op(args), rematerializing any
    evicted arguments. Wraps every operator
    invocation.
  Exclude members of args from eviction
  for any evicted arg in args:
      Rematerialize(arg)
  buf := AllocateBuffer(size(op(args)))
  res := call op(args), store into buf
  Permit eviction for members of args again
  Update metadata for args and res
  return res
```

```
PerformEviction():
  Free the tensor chosen by the heuristic
```

```
Rematerialize(t):
  op, args := operator and arguments that
    produced t (from metadata)
  return PerformOp(op, args)
```

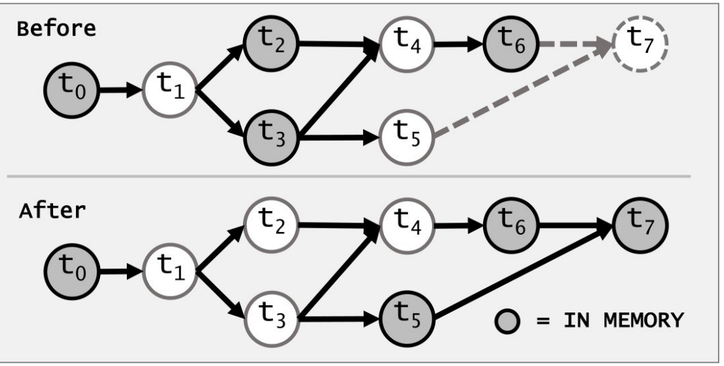
```
AllocateBuffer(b):
  Note: Wraps every memory allocation.
  while available memory < b:
      PerformEviction()
  return new buffer of size b
```

```
Deallocate(t):
  Note: Wraps every tensor deallocation.
  Heuristic decides policy for t (e.g., free
    permanently or simply evict)
```

Execution Trace

Computing t_7 with memory budget 4:

```
t7 = PerformOp(op7, t5, t6)
[t5, t6 become unevictable]
Rematerialize(t5)
t5 = PerformOp(op5, t3)
[t3 becomes unevictable]
AllocateBuffer(t5.size)
PerformEviction() #eg, t2
[t3 becomes evictable]
AllocateBuffer(t7.size)
PerformEviction() #eg, t3
```

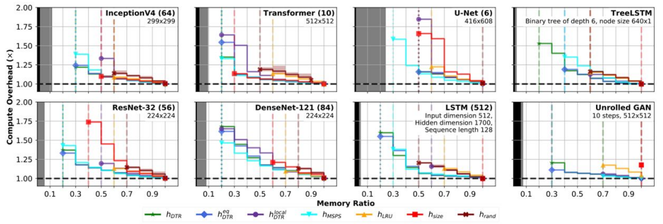


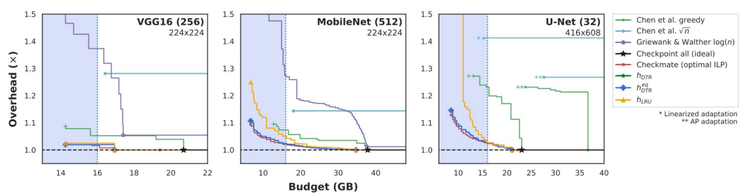
Heuristics: The Brains of DTR

Cost over dependencies

$$h_{\text{DTR}} \stackrel{\text{def}}{=} \frac{c_0(t) + \sum_{t' \in e^*(t)} c_0(t')}{m(t) \cdot s(t)}$$

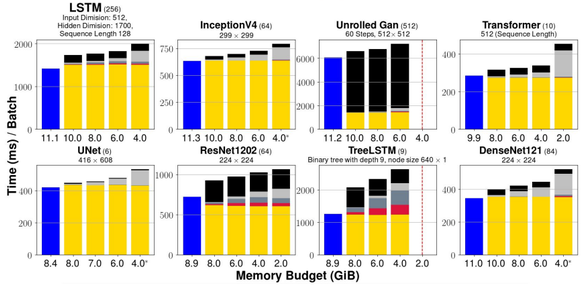
Size · Staleness





Near-optimal performance in simulation!

Prototype



Few hundred lines in PT, limited overhead

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING · sampl · PLSE · ada Applications Driving Architectures