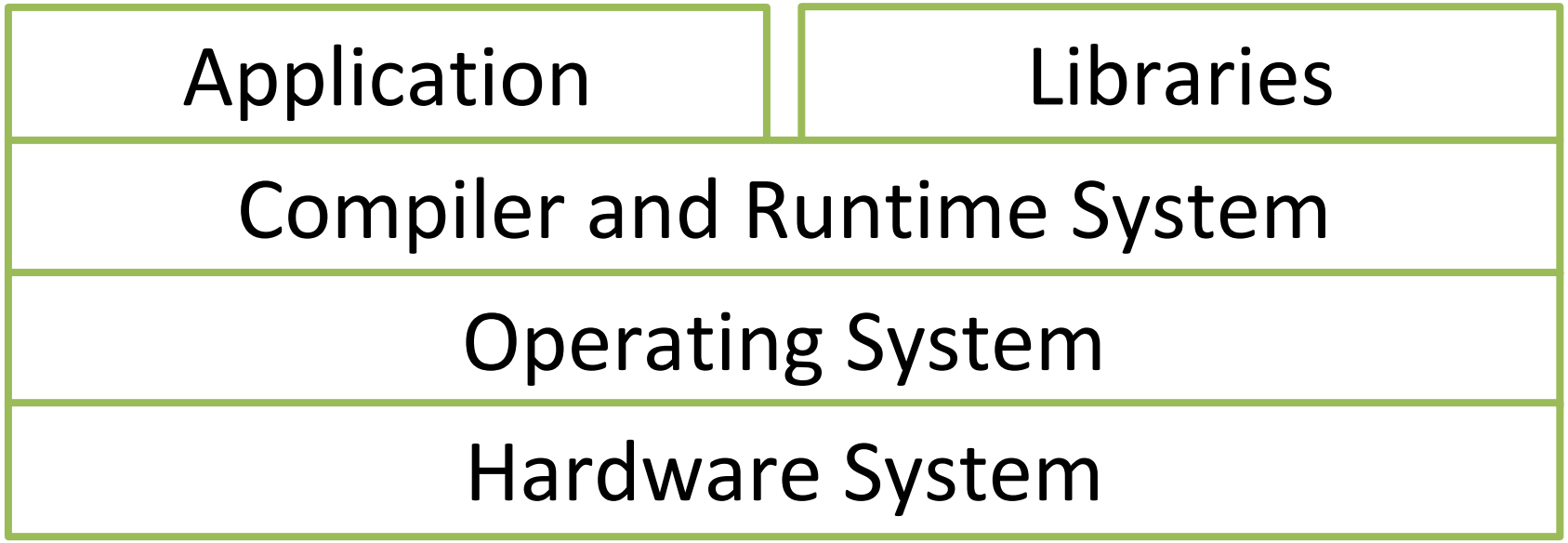
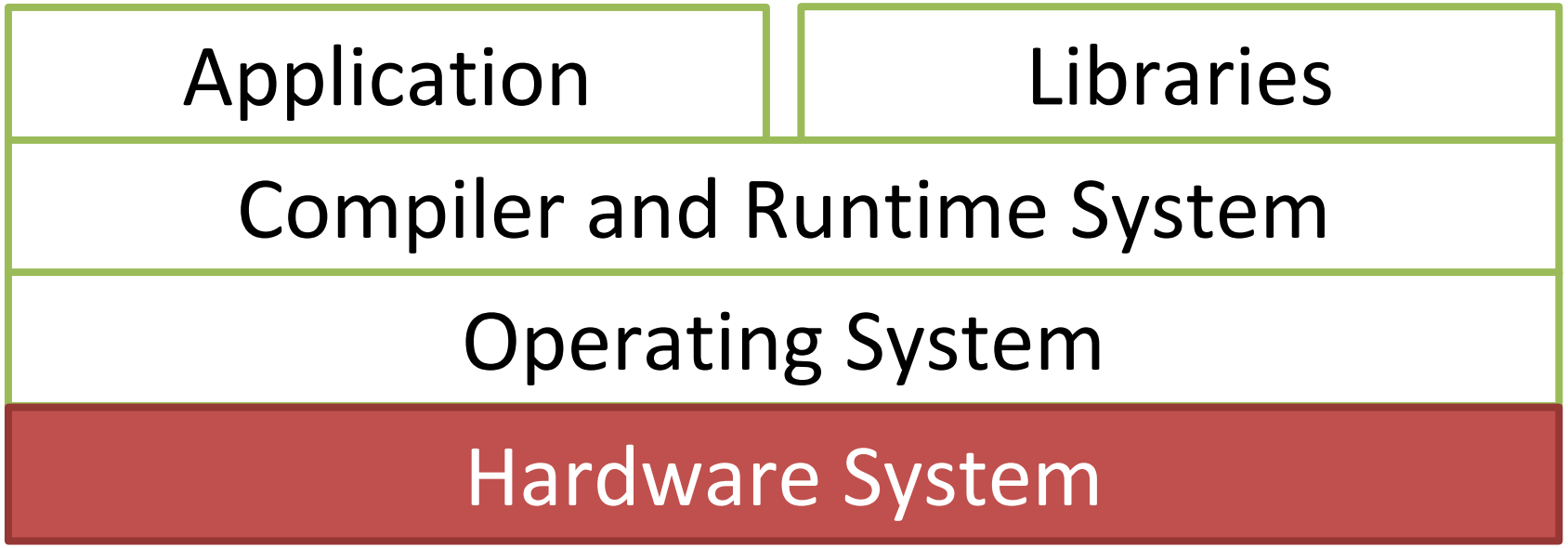
# Engineering Approximate Computations

Michael Carbin, MIT EECS & CSAIL

# Computing Stack



# Hardware Errors



# Examples

~175 Issues in Current Errata for Intel 6th Generation (Sept. 2018)

SKL057: Cache Performance Monitoring Events May be Inaccurate

SKL082: Processor May Hang or Cause Unpredictable Behavior

> “Under complex microarchitecture conditions, processor may **hang** with an internal timeout error ... or cause **unpredictable system behavior**.”

# In Practice

**Consumer**: "Intel Skylake bug causes PCs to freeze during complex workloads: Bug discovered while using Prime95 to find Mersenne primes."

http://arstechnica.com/gadgets/2016/01/intel-skylake-bug-causes-pcs-to-freeze-during-complex-workloads/

**Supercomputer**: "Researchers performed a study in 2010 on the then most powerful supercomputer, called Jaguar. The study found that [uncorrectable] errors occurred about once every 24 hours in Jaguar's 360 TB of memory."

http://spectrum.ieee.org/computing/hardware/how-to-kill-a-supercomputer-dirty-power-cosmic-rays-and-bad-solder

# In Practice

**Consumer**: "Intel Skylake bug causes PCs to freeze during complex workloads: Bug discovered while using Prime95 to find Mersenne primes."

http://arstechnica.com/gadgets/2016/01/intel-skylake-bug-causes-pcs-to-freeze-during-complex-workloads/

**Supercomputer**: "Researchers performed a study in 2010 on the then most powerful supercomputer, called Jaguar. The study found that [uncorrectable] errors occurred about once every 24 hours in Jaguar's 360 TB of memory."

http://spectrum.ieee.org/computing/hardware/how-to-kill-a-supercomputer-dirty-power-cosmic-rays-and-bad-solder

# Why?

- Logic: complex workload with multiple architectural events happening concurrently triggers corner case in design.
- Electrical: increasingly smaller transistors are more sensitive to physical variation in fabrication process.
- Environmental:  cosmic-rays, heat, aging, etc. …

# How do people cope with this problem?

Transient Errors: i.e., nondeterministic errors.

# How do people cope with this problem?

# What do you do?

Replication:

```
z = x + y;
```

```
do {
    z  = x + y;
    z' = x + y;
} while (z != z');
```

Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. 1956

# What do you do?

Replication:

```
z = x + y;
```

```
do {
    z  = x + y;
    z' = x + y;
} while (z != z');
```

Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. 1956

Checkable Computations:

```
x = newton_method(f, guess);
```

```
do {
    x = newton_method(f, guess);
} while (abs(f(x)) > eps);
```

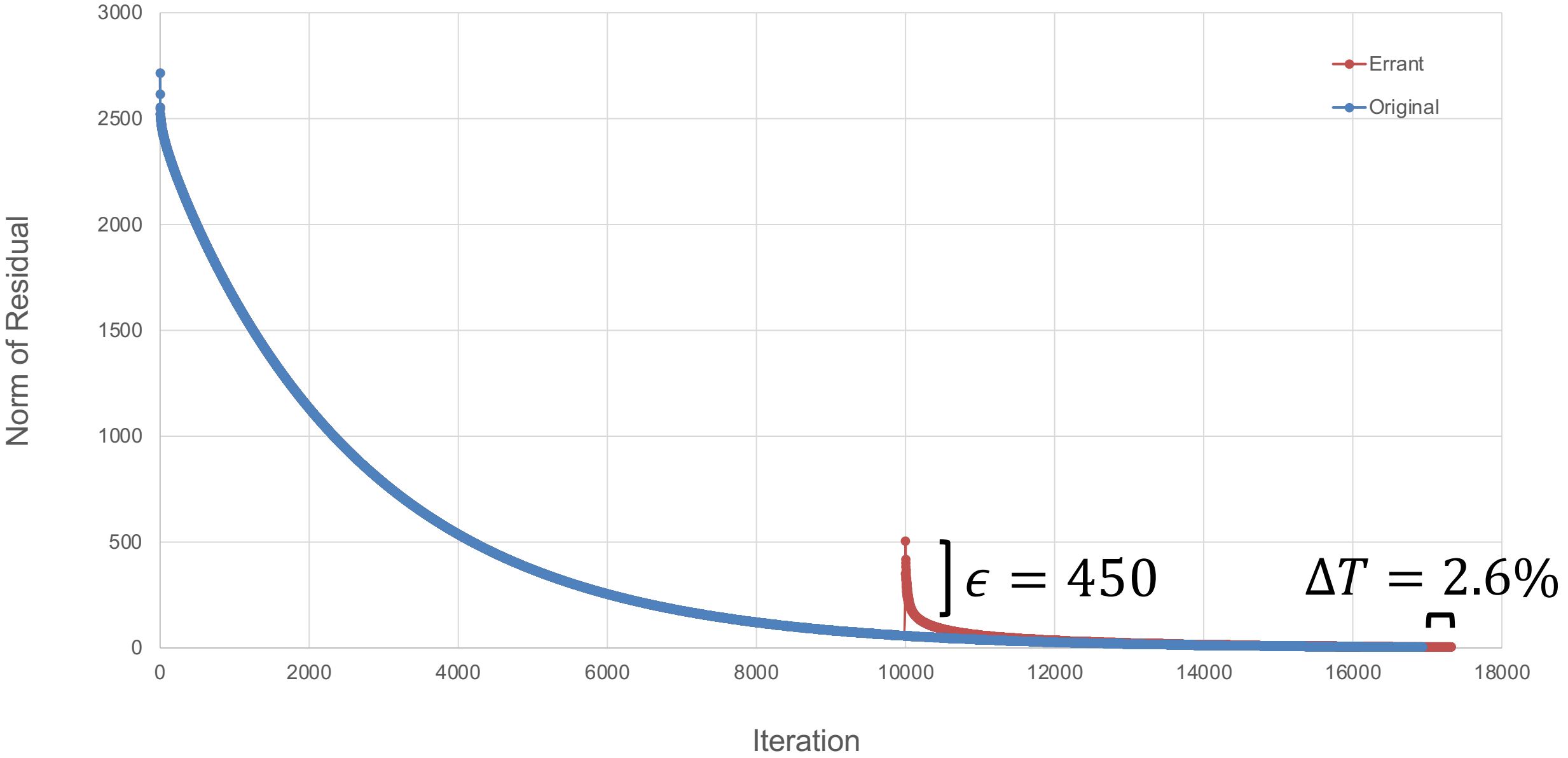
Huang and Abraham. Algorithm-based fault tolerance for matrix operations. 1984.

# What if we just let errors *happen*?

# Jacobi Iterative Method

```
while(<convergence condition>) {
  for (int i = 0; i < x.length; ++i) {
    float sigma = 0;
    for(int j = 0; j < x.length; ++j) {
       if (j != i) {
            float delta = A[i][j] * last_x[j];
            sigma = sigma + delta;
       }
    }
    float num = b[i] -. sigma;
    x[i] = num /. A[i][i];
  }
}
```

# Jacobi Iterative Method (with Error)



# Opportunity in Jacobi

Option 1: Replicate Computation

- 50% overhead with clever tricks and hardware support.

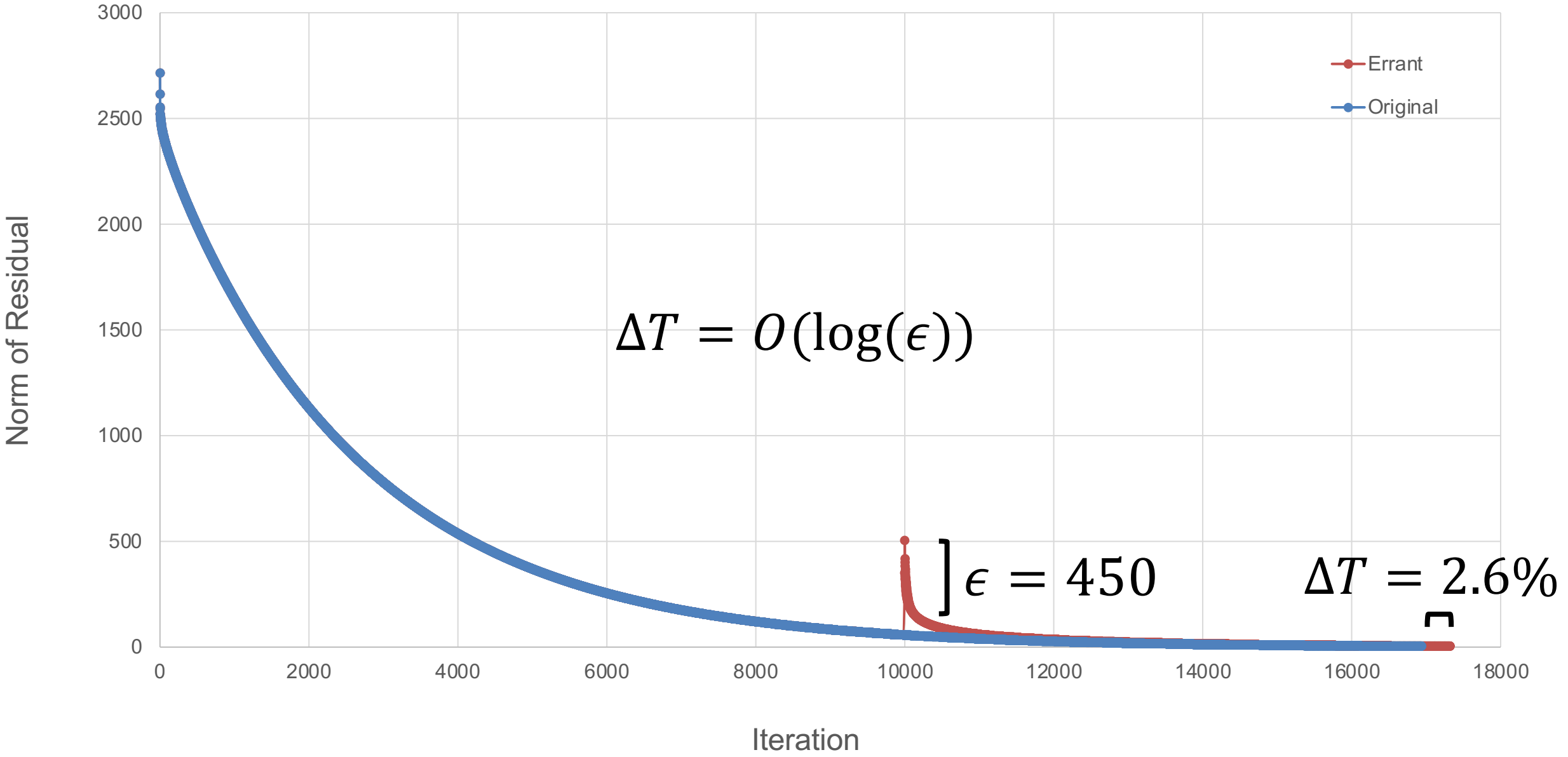
Option 2: Check Computation

- 30 % overhead with checks on linear algebra

Option 3: Approximate Computation (let errors happen)

- 2.6% overhead – if you can **prove** error is acceptable

# Jacobi Iterative Method (with Error)



# Expanded Approaches

Replication:

```
z = x + y;
```

```
do {
     z  = x + y;
     z’ = x + y;
} while (z != z’);
```

Checking:

```
x = newton_method(f, guess);
```

```
do {
     x = newton_method(f, guess);
} while (abs(f(x)) > eps);
```

Approximation:

```
z = x + y;
```

```
z = x + y;
assert(|z<o> - z<r>| < e);
```

# Research Hypothesis

We as systems researchers should think about approximation.

New opportunities:

- **Resilience**

# Research Hypothesis

We as systems researchers should think about approximation.

New opportunities:

- **Resilience**
- Performance
- Capability

# Loop Perforation

```
for (uint i = 0; i < n; ++i) {...}
```

```
for (uint i = 0; i < n/2; ++i) {...}
```

**What will happen to your program?**

Original

Perforated
(2x performance)

# Loop Perforation Results

## Applications

**Media Processing**

**Computer Vision**

**Machine Learning**

**Search**

**Finance**

## Framework

- Developer specifies maximum acceptable error using metric
- Automatically identifies loops perforations with acceptable error

## Performance improvement

- Typically over a factor of two
- Up to a factor of seven

## Quality Impact

- < 10% change in output

[1] Misailovic ., ISCE ’10, [2] Hoffman., ASPLOS ’11, [3] Lou, SIGGRAPH ‘16

# Research Hypothesis

We as systems researchers should think about approximation.

New opportunities:

- Resilience
- **Performance**
- Capability

# Research Challenges

Approximate Computation:

```
z = x + y;
```

```
z = x + y;
assert(|z<o> - z<r>| < e);
```

What is the methodology for verifying this `assert` in a program?

- Refers to two executions of the program
  - One without errors and one with errors
- How can a system help automate verification?

Relaxed Programming (PLDI '12), Rely (OOPSLA '13), Chisel (OOPSLA '14), Leto (OOPSLA '18)

# Research Hypothesis

We as systems researchers should think about approximation.

New opportunities:

- Resilience
- Performance
- Capability

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

- Embracing approximation yields improved capability

Systems Building Challenges:

- Performance
- Semantics (Real-valued, Differentiable, Probabilistic, Programs)
- Correctness

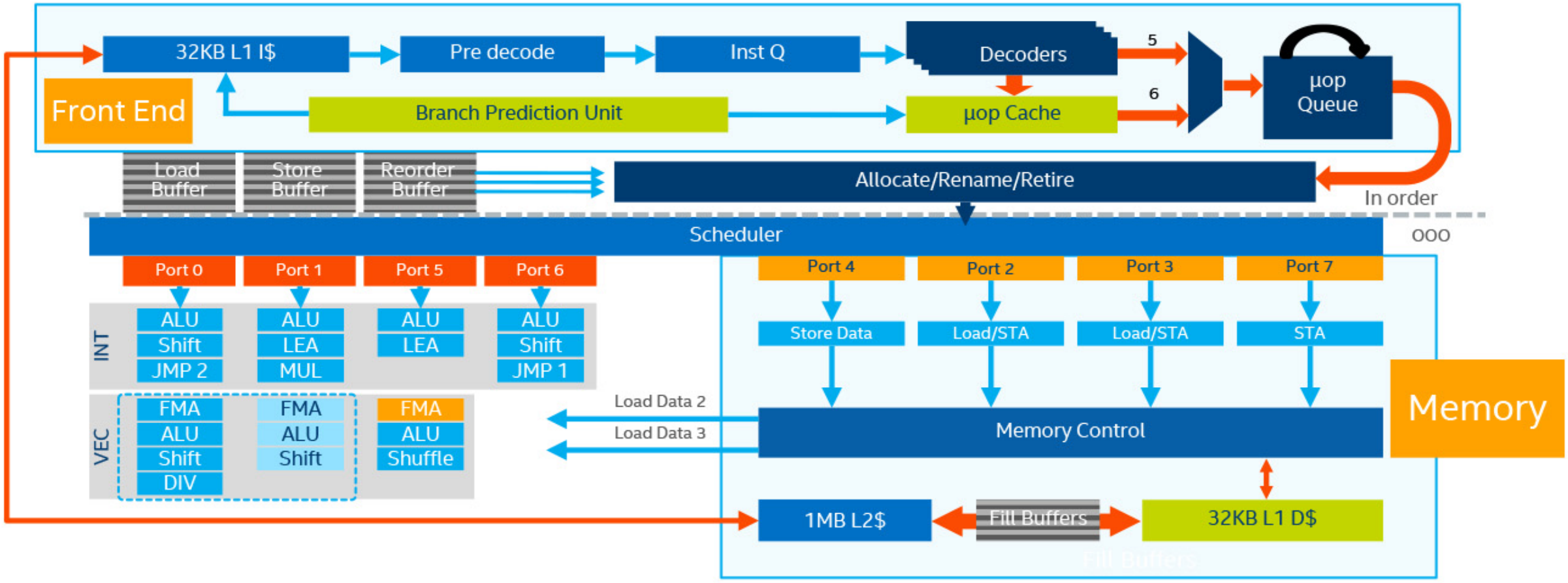
# Case Study: Performance Modeling

Critical for compilation, performance engineering, and debugging.

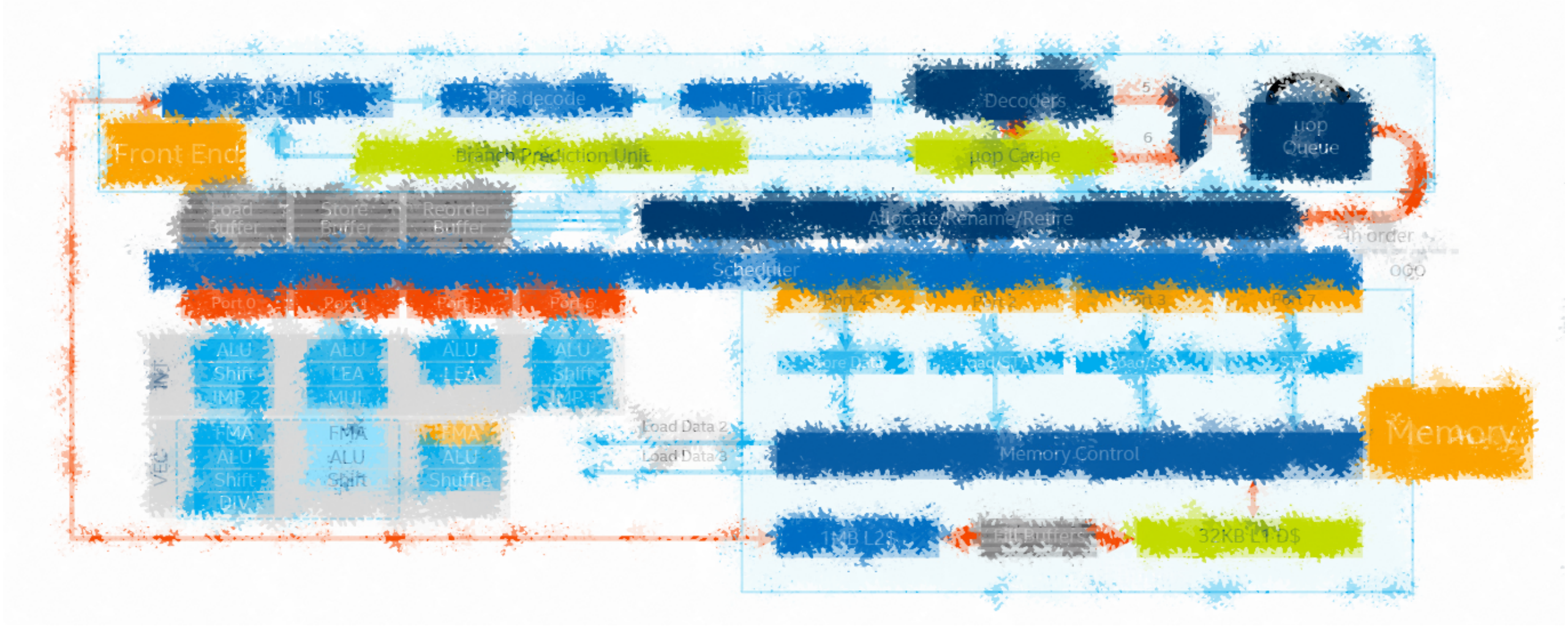
$$f\left(\begin{array}{l}\texttt{mov ebx, [ecx]}\\ \texttt{add ecx, ebx}\\ \texttt{shl eax, 0x02}\end{array}\right) = 2$$

The *throughput* of a sequence of instructions—the number processor clock cycles taken to execute the sequence when looped in steady state.

# Modern Microarchitecture (Skylake)



# Modern Microarchitecture (Skylake)

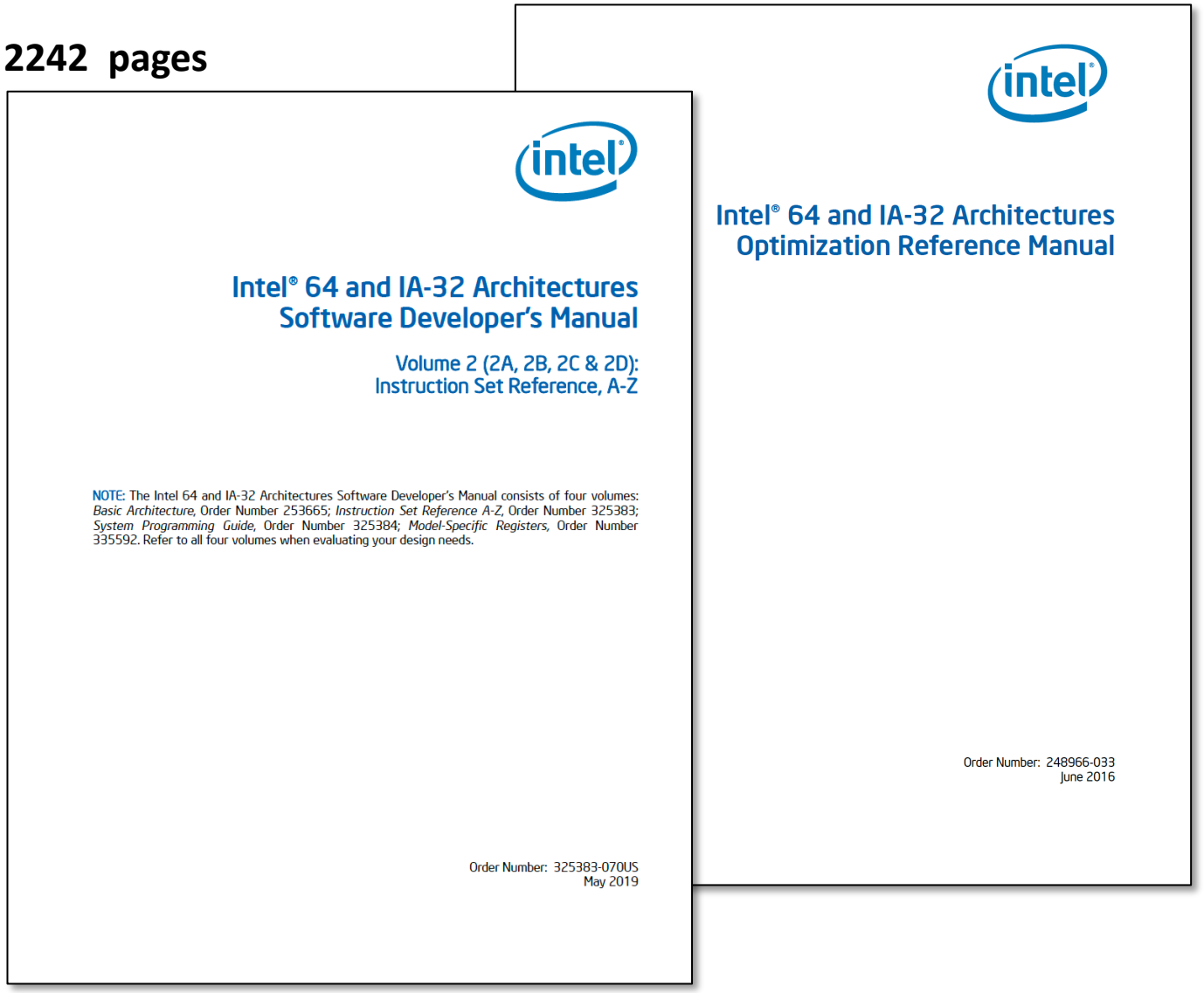


# Design Documents

- Intel Manual
  - Instruction specifications
  - Optimizations
  - CPU Parameters
  - Instruction Throughputs

**672 pages**

**2242 pages**



# Instruction Tables

Table C-17. General Purpose Instructions (Contd.)

| Instruction | Latency[1] | | | | Throughput | | | |
|---|---|---|---|---|---|---|---|---|
| CPUID | 06_4E,06_5E | 06_3D/47/56 | 06_3C/45/46/3F | 06_3A, 06_3E | 06_4E,06_5E | 06_3D/47/56 | 06_3C/45/46/3F | 06_3A, 06_3E |
| MUL r64[11] | 4, 5 | 3, 4 | 3, 4 | 3, 4 | 1 | 1 | 1 | 1 |
| NEG/NOT | 1 | 2 | 2 | 2 | 0.25 | 0.25 | 0.25 | 0.33 |
| PAUSE | | | | | ~140 | ~10 | ~10 | ~10 |
| RCL/RCR reg, 1 | 2 | 2 | 2 | 2 | 2 | 1.5 | 1.5 | 1.5 |
| RCL/RCR | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| RDTSC | | | | | ~13 | ~10 | ~10 | ~20 |
| RDTSCP | | | | | ~20 | ~30 | ~30 | ~30 |
| ROL/ROR reg 1 | 1 (2 flg) | 1 (2 flg) | 1 (2 flg) | 1 (2 flg) | 1 | 1 | 1 | 1 |
| ROL/ROR reg imm | 1 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| ROL/ROR reg, cl | 2 | 2 | 2 | 2 | 1.5 | 1.5 | 1.5 | 1.5 |
| LAHF/SAHF | 3 | 2 | 2 | 2 | | | | |
| SAL/SAR/SHL/SHR reg, imm | 1 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| SAL/SAR/SHL/SHR reg, cl | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 | 1.5 |
| SETBE | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| SETE | 1 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| SHLD/RD reg, reg, cl | 6 | 4 | 4 | 2 (4 flg) | 1.5 | 1 | 1 | 1.5 |
| SHLD/RD reg, reg, imm | 3 | 3 | 3 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| XSAVE[12] | | | | | ~98 | ~100 | ~100 | ~100 |
| XSAVEOPT[12] | | | | | ~86 | ~90 | ~90 | ~90 |
| XADD | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| XCHG reg, reg | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| XCHG reg, mem | 22 | 19 | 19 | 19 | 22 | 19 | 19 | 19 |

# Instruction Tables

Table C-17. General Purpose Instructions (Contd.)

| Instruction | Latency[1] | | | | Throughput | | | |
|---|---|---|---|---|---|---|---|---|
| CPUID | 06_4E,06_5E | 06_3D/47/56 | 06_3C/45/46/3F | 06_3A,06_3E | 06_4E,06_5E | 06_3D/47/56 | 06_3C/45/46/3F | 06_3A,06_3E |
| MUL r64[11] | 4, 5 | 3, 4 | 3, 4 | 3, 4 | 1 | 1 | 1 | 1 |
| NEG/NOT | 1 | 2 | 2 | 2 | 0.25 | 0.25 | 0.25 | 0.33 |
| PAUSE | | | | | ~140 | ~10 | ~10 | ~10 |

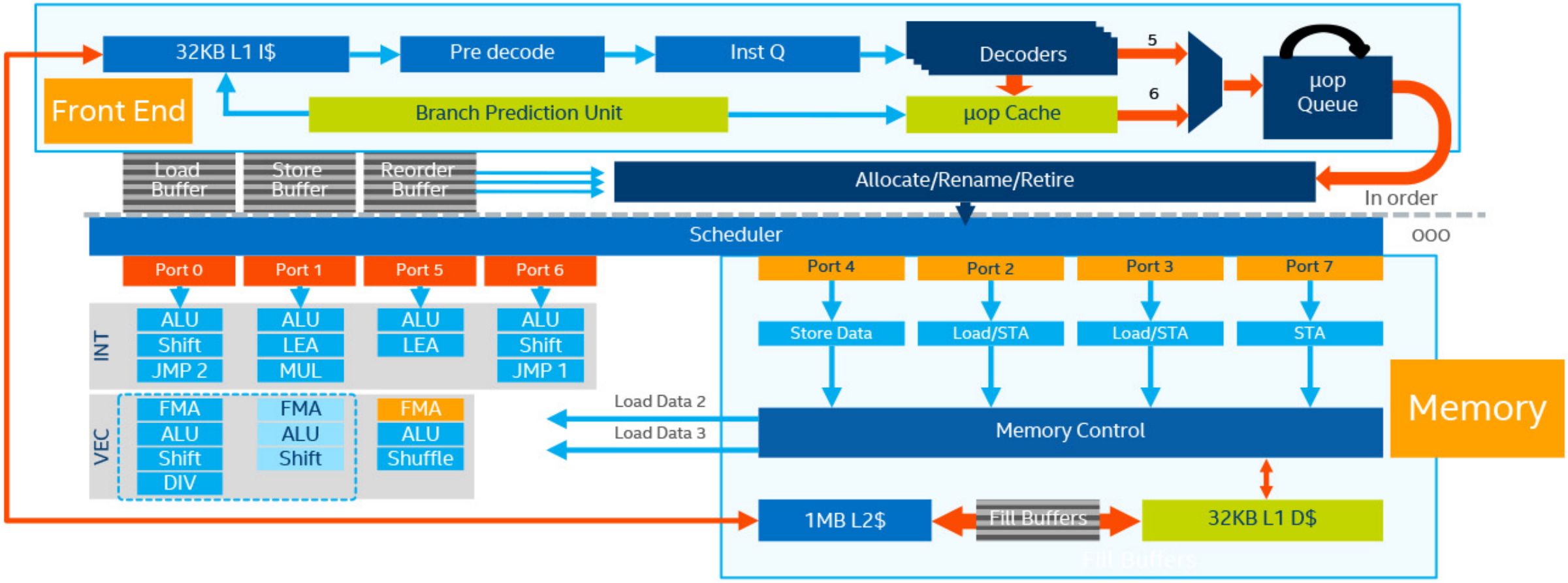
## C.3.2 Table Footnotes

The following footnotes refer to all tables in this appendix.

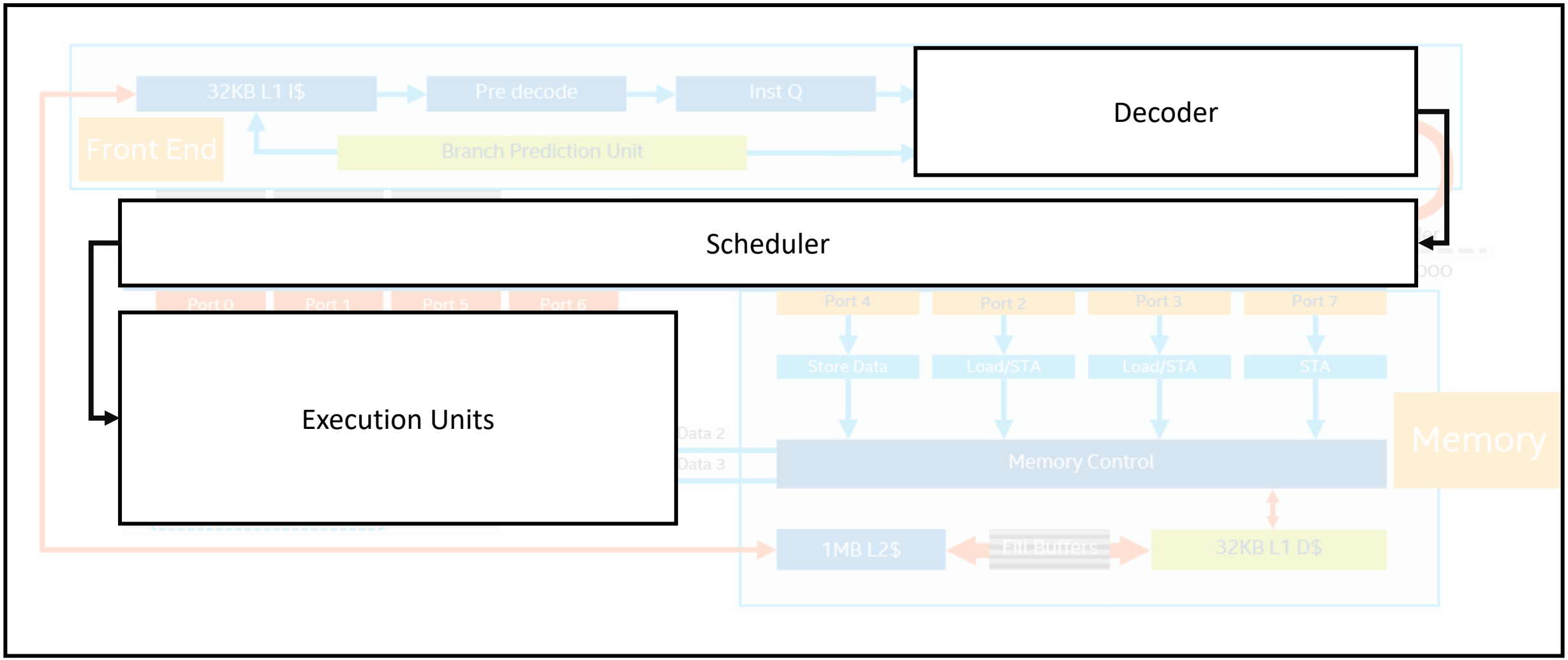
1. Latency information for many instructions that are complex (> 4 μops) are estimates based on conservative (worst-case) estimates. Actual performance of these instructions by the out-of-order core execution unit can range from somewhat faster to significantly faster than the latency data shown in these tables.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| SETE | 1 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| SHLD/RD reg, reg, cl | 6 | 4 | 4 | 2 (4 flg) | 1.5 | 1 | 1 | 1.5 |
| SHLD/RD reg, reg, imm | 3 | 3 | 3 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| XSAVE[12] | | | | | ~98 | ~100 | ~100 | ~100 |
| XSAVEOPT[12] | | | | | ~86 | ~90 | ~90 | ~90 |
| XADD | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
| XCHG reg, reg | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| XCHG reg, mem | 22 | 19 | 19 | 19 | 22 | 19 | 19 | 19 |

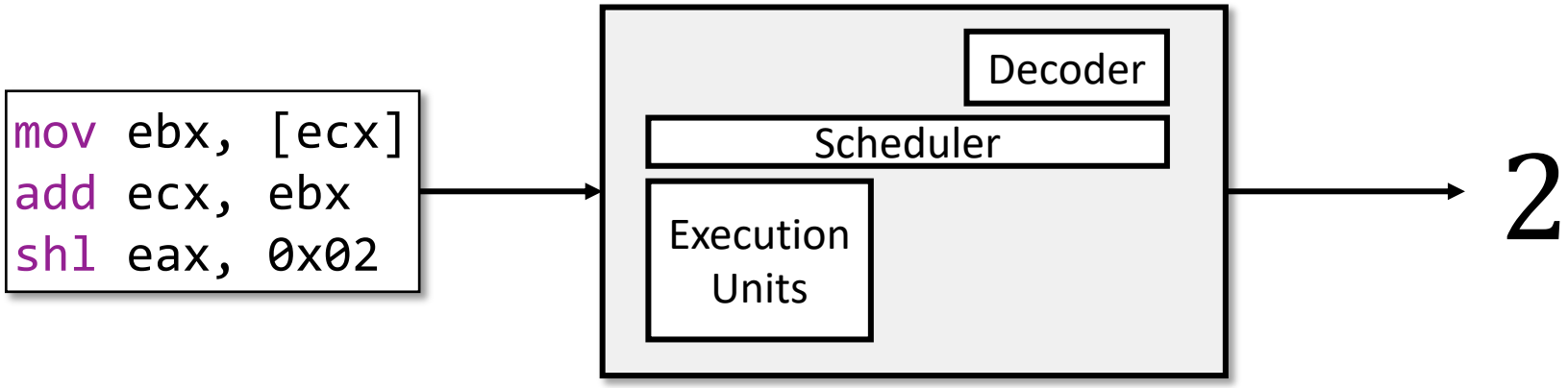
# Modern Microarchitecture



# Modern Performance Model



# Modern Performance Model



# Example

How many cycles to compute 100 iterations of:

```
vxorps xmm0 xmm0, xmm0
```

# Example

How many cycles to compute 100 iterations of:

```
vxorps xmm0 xmm0, xmm0
```

Intel Instruction Set Reference Page 1955 of 2242:

| | | | | |
|---|---|---|---|---|
| VEX.128.0F.WIG 57 /r<br>VXORPS xmm1,xmm2, xmm3/m128 | B | V/V | AVX | Return the bitwise logical XOR of packed single-precision floating-point values in xmm2 and xmm3/mem. |

# Example

How many cycles to compute 100 iterations of:

r1 = r1 xor r1

Intel Instruction Set Reference Page 1955 of 2242:

| | | | | |
|---|---|---|---|---|
| VEX.128.0F.WIG 57 /r<br>VXORPS xmm1,xmm2, xmm3/m128 | B | V/V | AVX | Return the bitwise logical XOR of packed single-precision floating-point values in xmm2 and xmm3/mem. |

# Example

How many cycles to compute 100 iterations of:

```
r1 = r1 xor r1
```

Intel Instruction Set Reference Page 1955 of 2242:

| | | | | |
|---|---|---|---|---|
| VEX.128.0F.WIG 57 /r<br>VXORPS xmm1,xmm2, xmm3/m128 | B | V/V | AVX | Return the bitwise logical XOR of packed single-precision floating-point values in xmm2 and xmm3/mem. |

Intel Architectures Optimization Reference Manual 655 of 672:

| Instruction | Latency | Throughput |
|---|---|---|
| `r1 = r2 xor r3` | 1 | 0.33 |

# Example

How many cycles to compute 100 iterations of:

```
r1 = r1 xor r1
```

Intel Instruction Set Reference Page 1955 of 2242:

| | | | | |
|---|---|---|---|---|
| VEX.128.0F.WIG 57 /r<br>VXORPS xmm1,xmm2, xmm3/m128 | B | V/V | AVX | Return the bitwise logical XOR of packed single-precision floating-point values in xmm2 and xmm3/mem. |

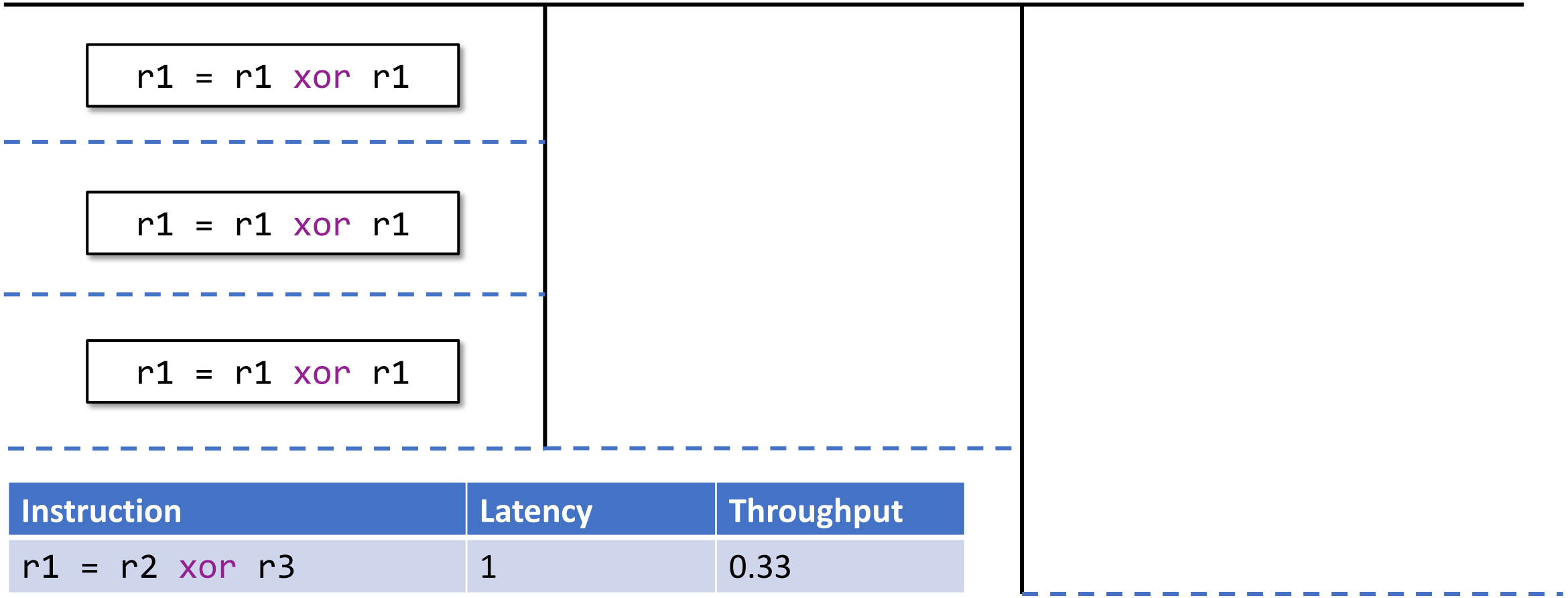
Intel Architectures Optimization Reference Manual 655 of 672:

| Instruction | Latency | Throughput |
|---|---|---|
| `r1 = r2 xor r3` | 1 | 0.33 |

?

# Modeling Execution

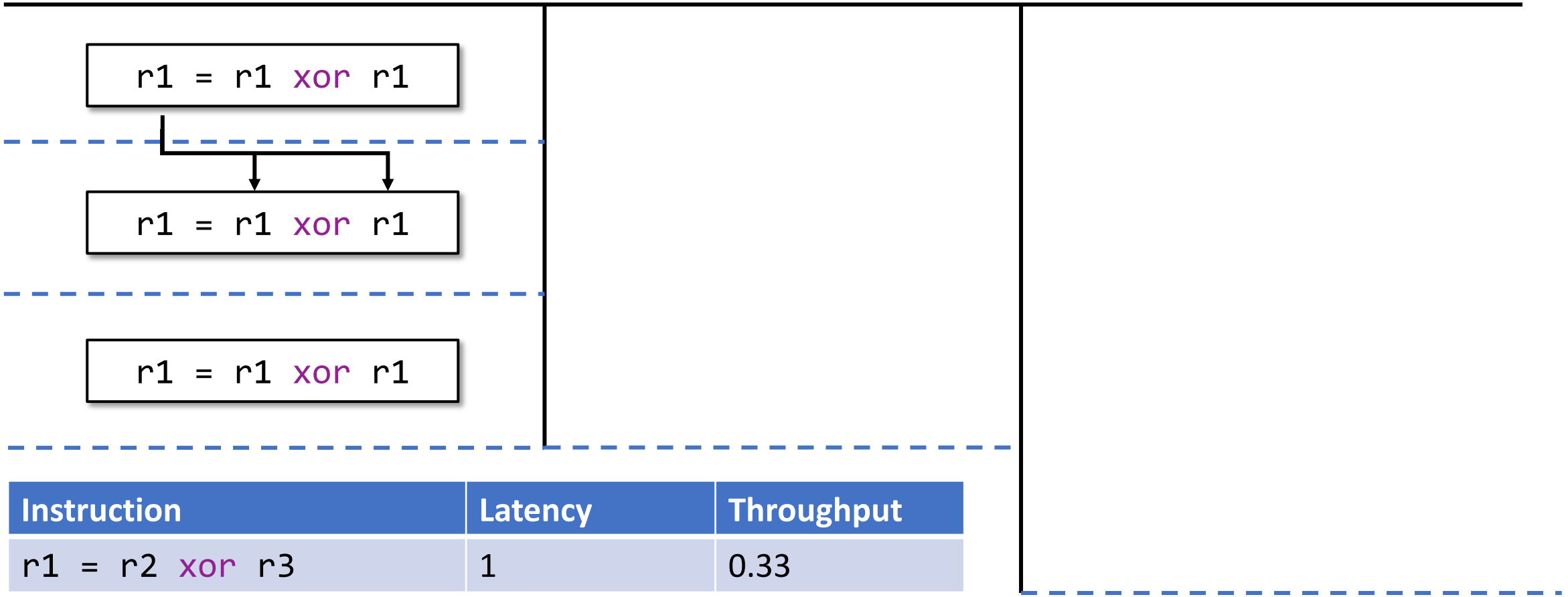
LLVM: 100




| Instruction | Latency | Throughput |
|---|---|---|
| `r1 = r2 xor r3` | 1 | 0.33 |

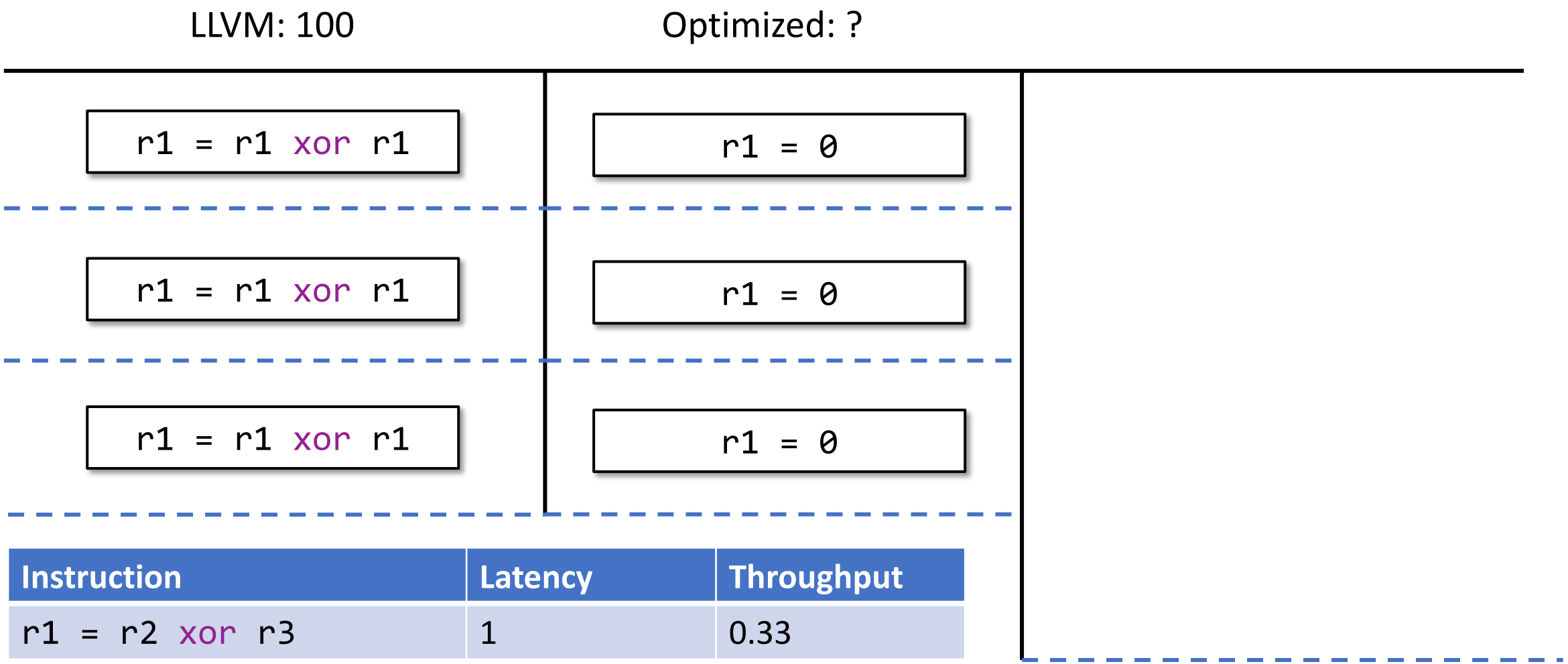
# Modeling Execution

LLVM: 100




| Instruction | Latency | Throughput |
|---|---|---|
| `r1 = r2 xor r3` | 1 | 0.33 |

# Modeling Execution




| Instruction | Latency | Throughput |
|---|---|---|
| r1 = r2 xor r3 | 1 | 0.33 |

# Modeling Execution

**Dependency Breaking Idioms**

Instruction parallelism can be improved by using common instructions to clear register contents to zero. The renamer can detect them on the zero evaluation of the destination register.

Use one of these dependency breaking idioms to clear a register when possible.

- XOR REG,REG
- SUB REG,REG
- PXOR/VPXOR XMMREG,XMMREG
- PSUBB/W/D/Q XMMREG,XMMREG
- VPSUBB/W/D/Q XMMREG,XMMREG
- XORPS/PD XMMREG,XMMREG
- VXORPS/PD YMMREG, YMMREG

Since zero idioms are detected and removed by the renamer, they have no execution latency.

| Instruction | Latency | Throughput |
| --- | --- | --- |
| `r1 = r2 xor r3` | 1 | 0.33 |

```
r1 = 0
```

# Modeling Execution

**Dependency Breaking Idioms**

Instruction parallelism can be improved by using common instructions to clear register contents to zero. The renamer can detect them on the zero evaluation of the destination register.

Use one of these dependency breaking idioms to clear a register when possible.

- XOR REG,REG
- SUB REG,REG
- PXOR/VPXOR XMMREG,XMMREG
- PSUBB/W/D/Q XMMREG,XMMREG
- VPSUBB/W/D/Q XMMREG,XMMREG
- XORPS/PD XMMREG,XMMREG
- VXORPS/PD YMMREG, YMMREG

Since zero idioms are detected and removed by the renamer, they have no execution latency.

| Instruction | Latency | Throughput |
| --- | --- | --- |
| `r1 = r2 xor r3` | 1 | 0.33 |

```
r1 = 0
```

# Modeling Execution

**Dependency Breaking Idioms**

Instruction parallelism can be improved by using common instructions to clear register contents to zero. The renamer can detect them on the zero evaluation of the destination register.

Use one of these dependency breaking idioms to clear a register when possible.

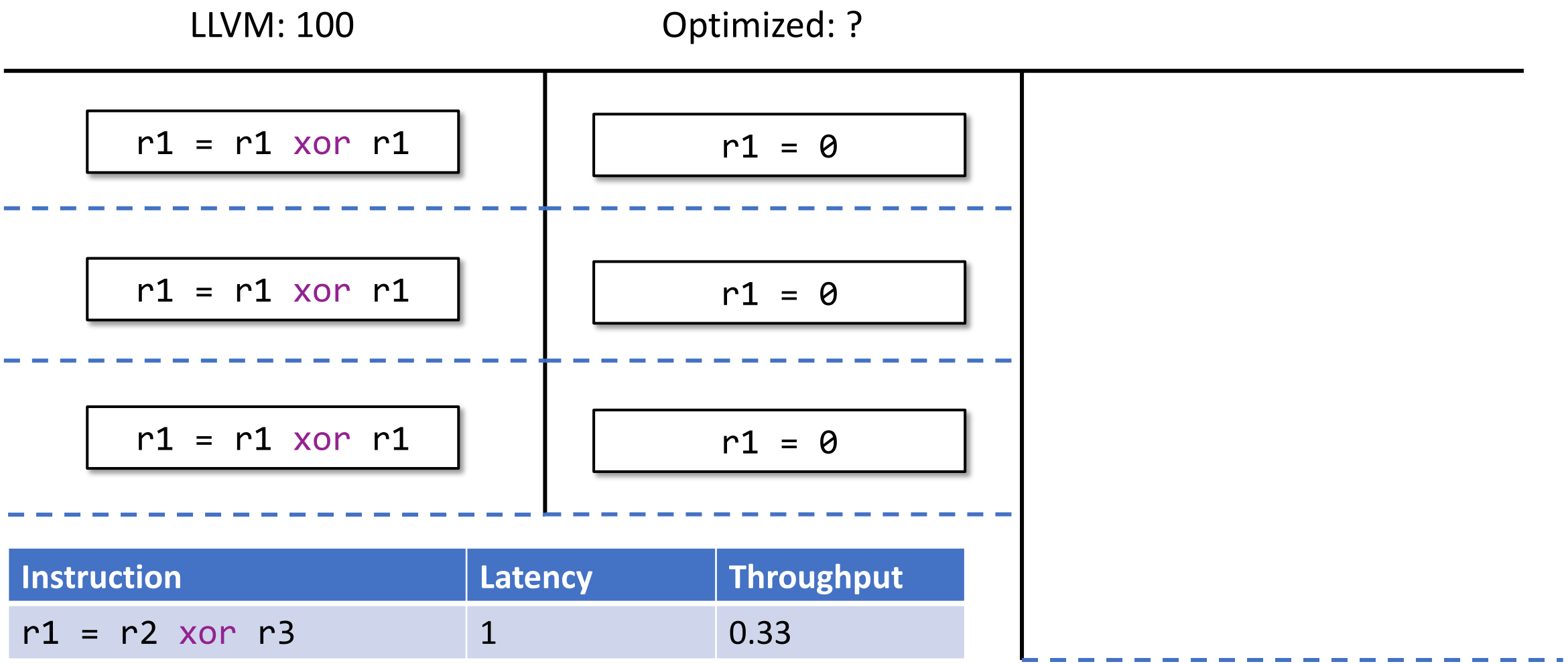
- XOR REG,REG
- SUB REG,REG
- PXOR/VPXOR XMMREG,XMMREG
- PSUBB/W/D/Q XMMREG,XMMREG
- VPSUBB/W/D/Q XMMREG,XMMREG
- XORPS/PD XMMREG,XMMREG
- VXORPS/PD YMMREG, YMMREG

Since zero idioms are detected and removed by the renamer, they have no execution latency.

| Instruction | Latency | Throughput |
| --- | --- | --- |
| r1 = r2 xor r3 | 1 | 0.33 |

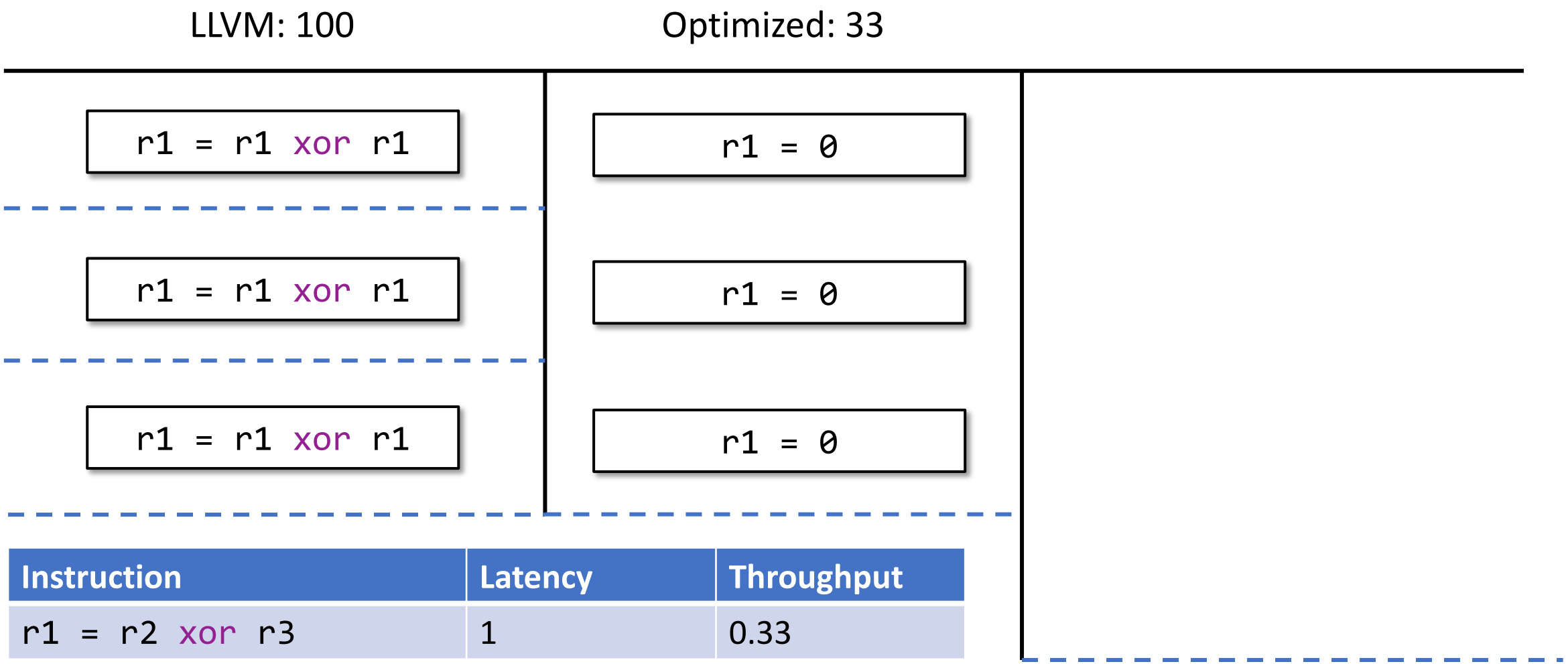
```
r1 = 0
```

# Modeling Execution



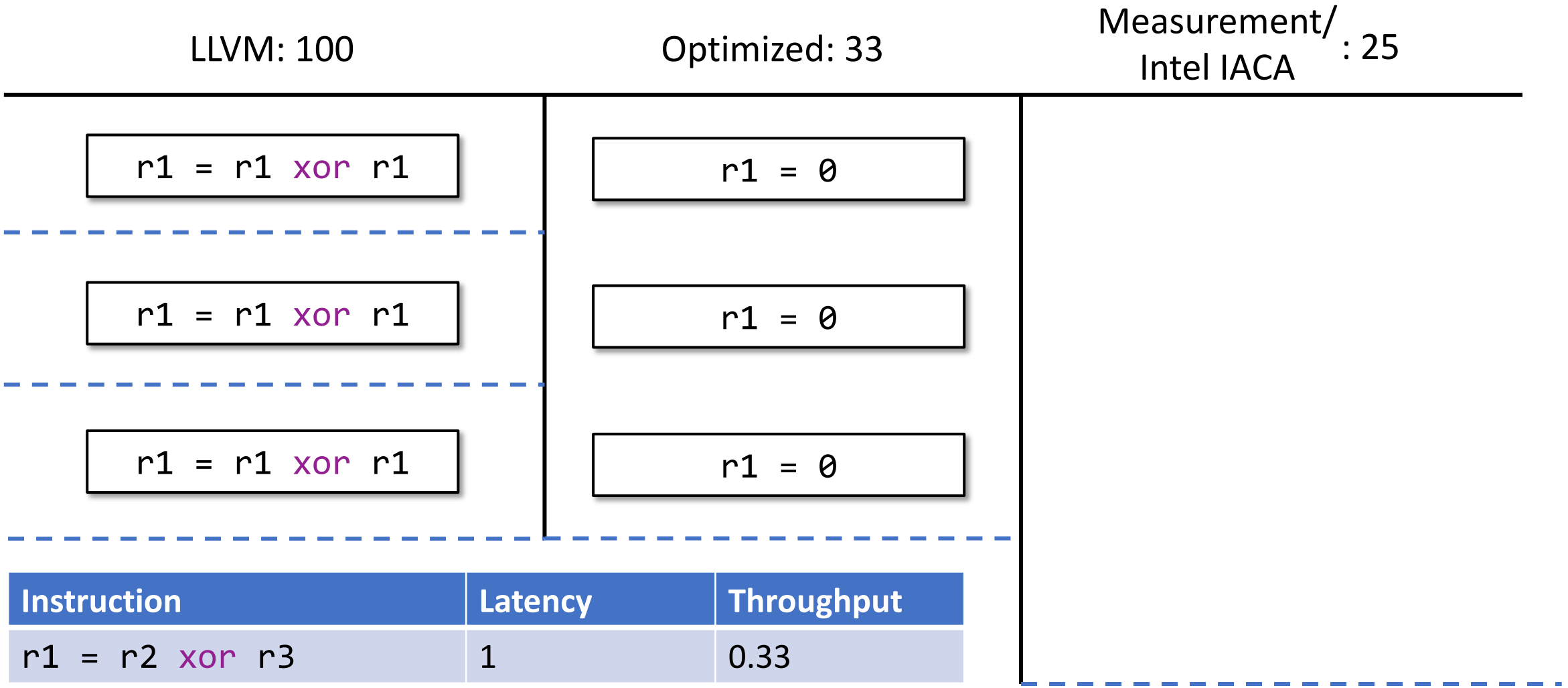

| Instruction | Latency | Throughput |
|---|---|---|
| r1 = r2 xor r3 | 1 | 0.33 |

# Modeling Execution



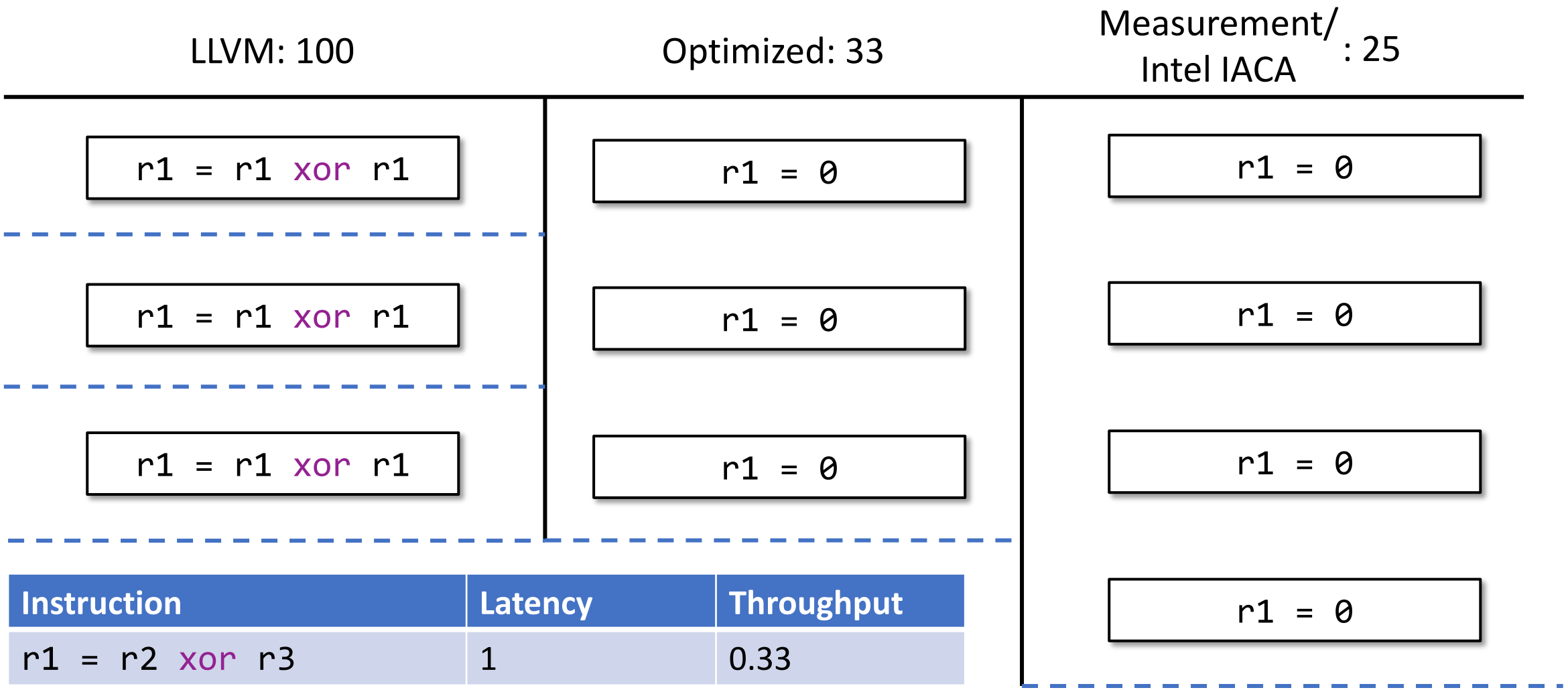

| Instruction | Latency | Throughput |
|---|---|---|
| r1 = r2 xor r3 | 1 | 0.33 |

# Modeling Execution



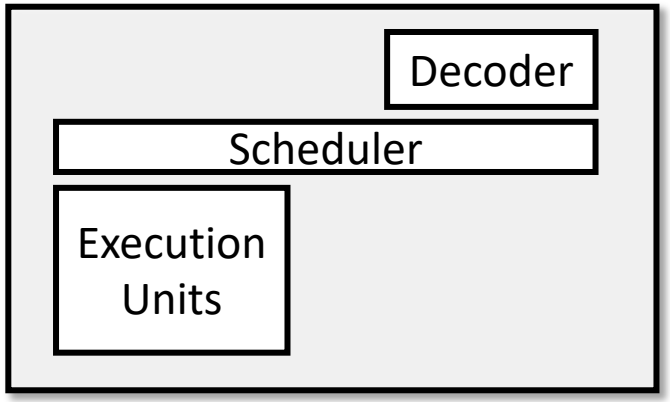

| Instruction | Latency | Throughput |
|---|---|---|
| r1 = r2 xor r3 | 1 | 0.33 |

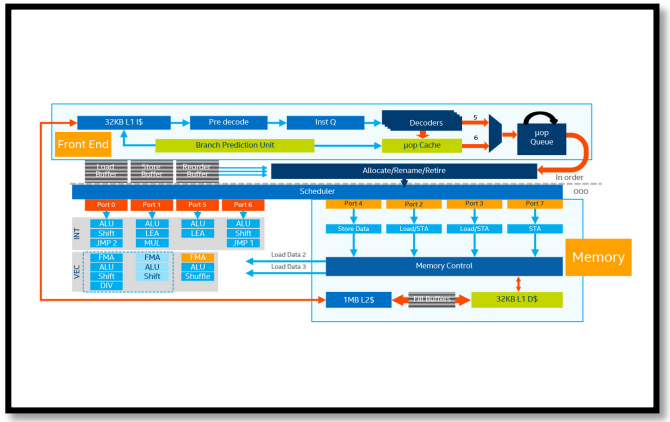
# Modeling Execution




| Instruction | Latency | Throughput |
|---|---|---|
| r1 = r2 xor r3 | 1 | 0.33 |

# Development Methodology

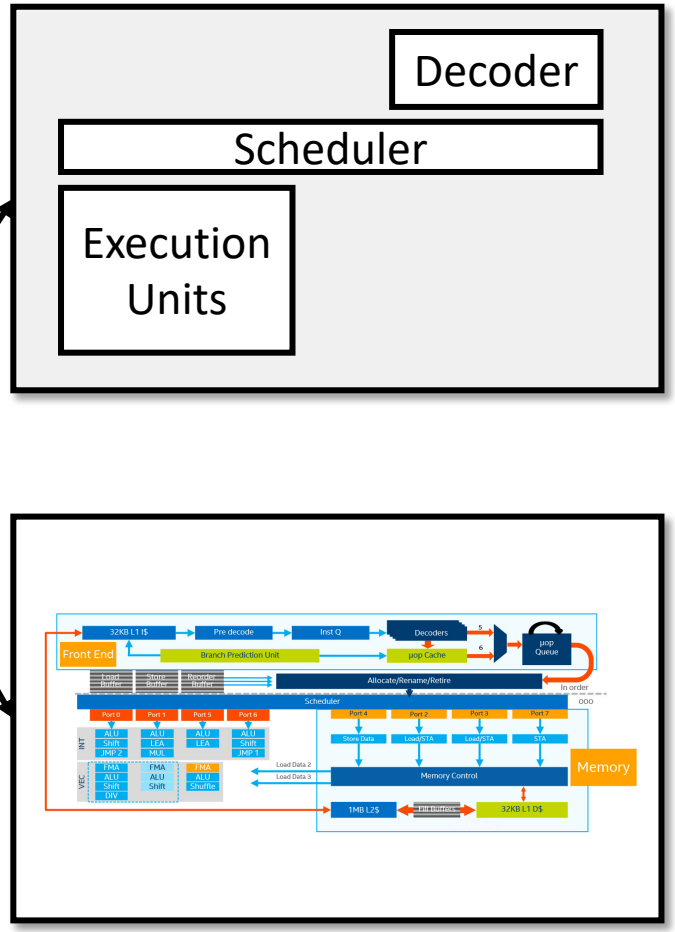


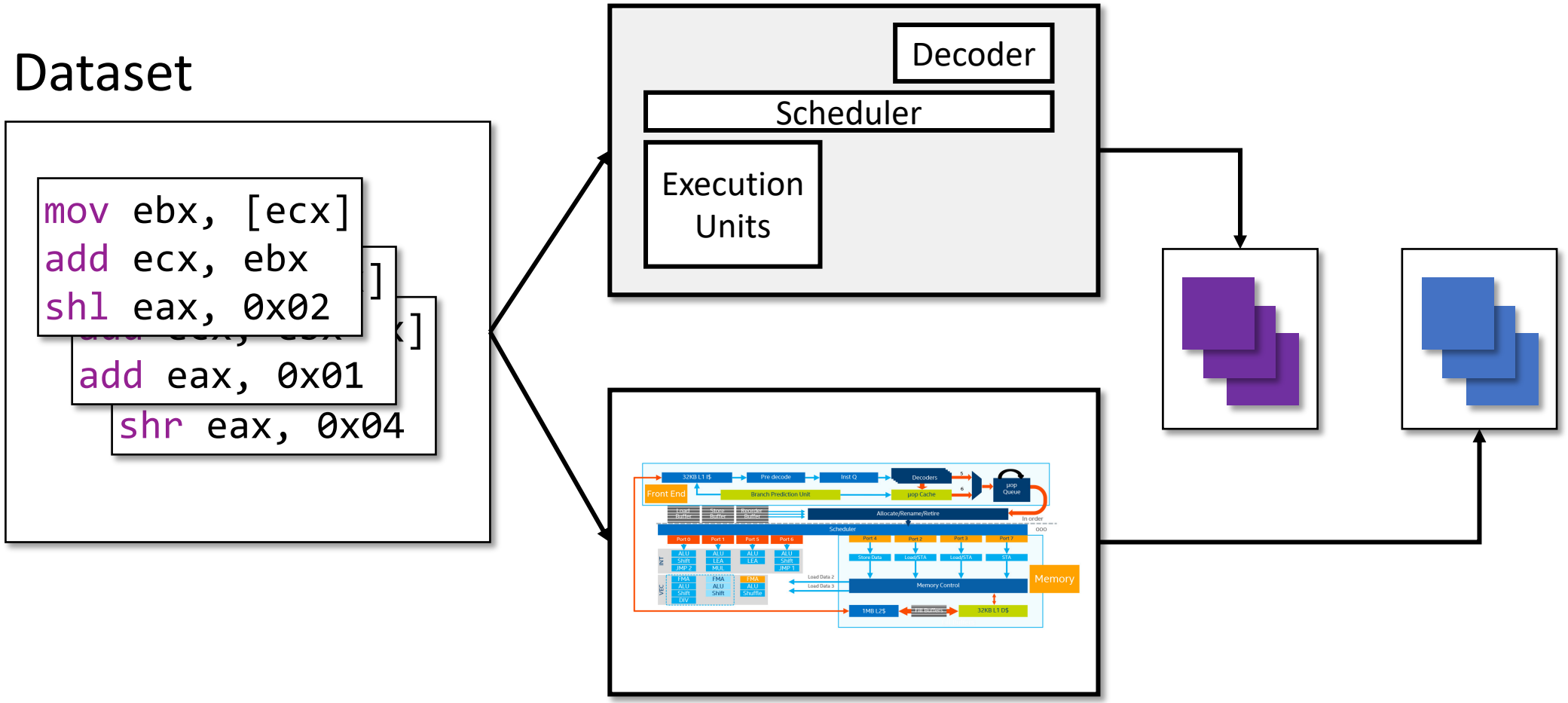
# Development Methodology

Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```



# Development Methodology



# Development Methodology

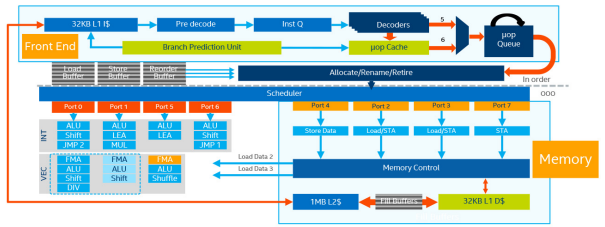
Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```

Decoder

Scheduler

Execution Units



# Development Methodology

Refine

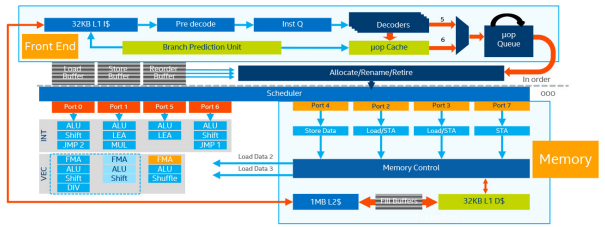
Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```

Decoder

Scheduler

Execution Units

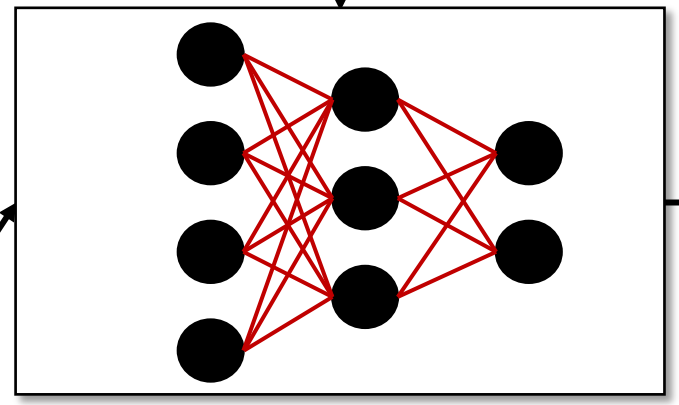


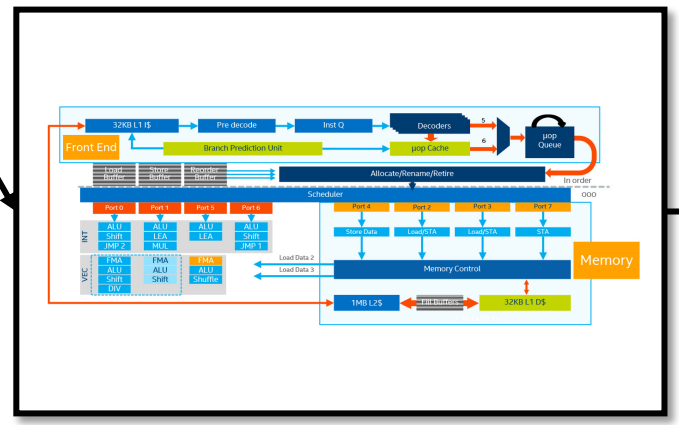
# Development Methodology

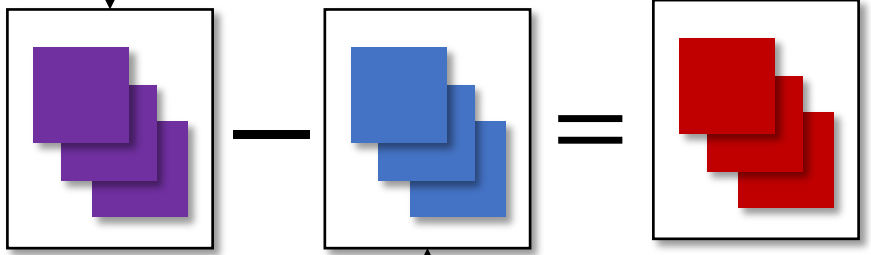
Refine

Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```





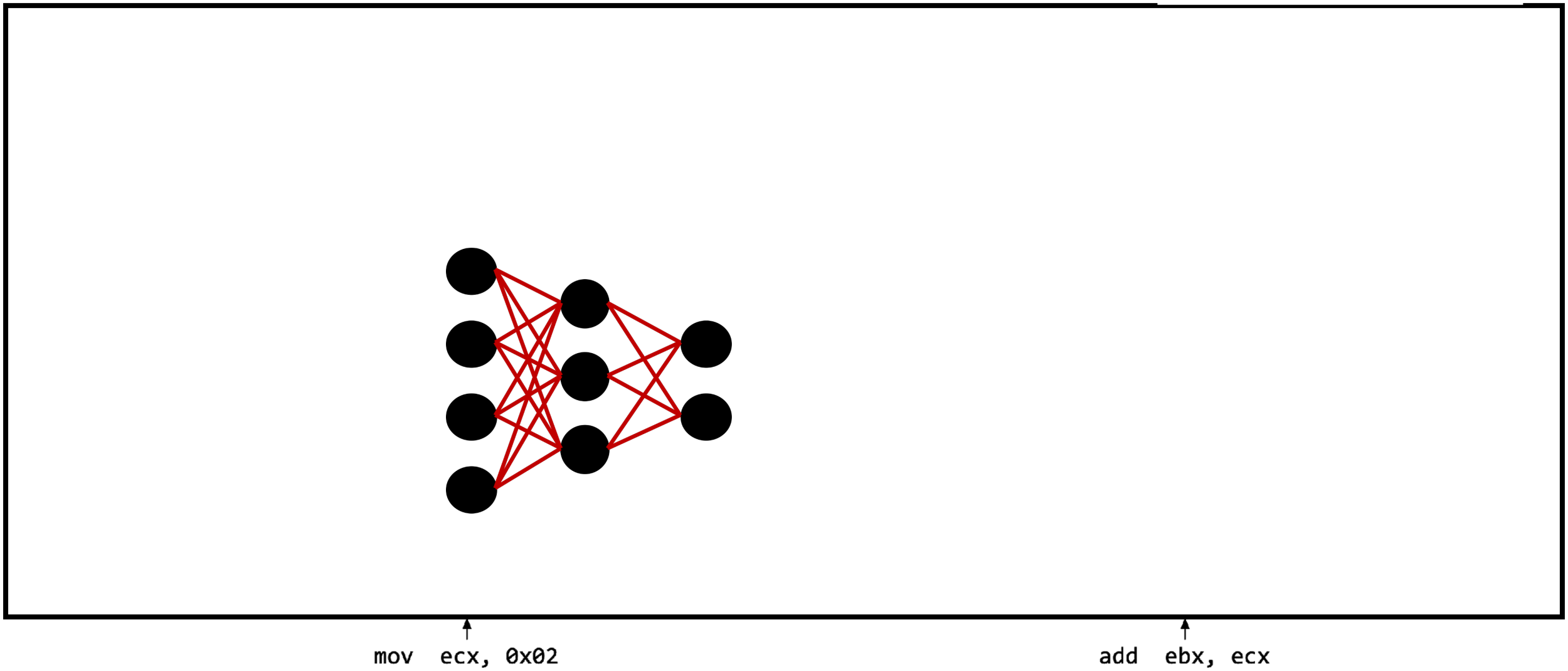
# Ithemal

- State-of-the-art performance
  - Half the error of LLVM and IACA (Intel's own tool)
- Hierarchical Recurrent Neural Network
- Novel throughput measurement system
  - Invariants: no cache misses, preemptions, etc.
- Dataset
  - 1.4M basic blocks: linux shared libraries, SPEC2006, SPEC2006, polybench, NAS, clang, python 2.7, python 3.5, firefox, rhythmbox,

| Micro-architecture | Method | Error |
|---|---|---|
| Ivy Bridge | llvm-mca | 0.181 |
| | Ithemal | **0.089** |
| Haswell | llvm-mca | 0.200 |
| | IACA | 0.209 |
| | Ithemal | **0.089** |
| Skylake | llvm-mca | 0.239 |
| | IACA | 0.167 |
| | Ithemal | **0.079** |

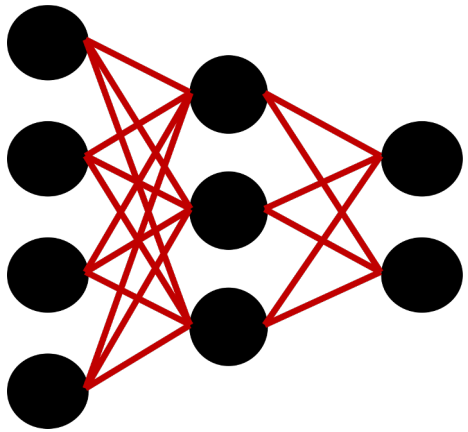
Mendis, Renda, Amarasinghe, and Carbin. Ithemal: Throughput Prediction with DNNs. ICML 2019
Chen et al., BHive: A Benchmark Suite and Measurement Framework for x86-64 Basic Blocks. IISWC 2019

# Ithemal



# Ithemal

Throughput Prediction 87.35

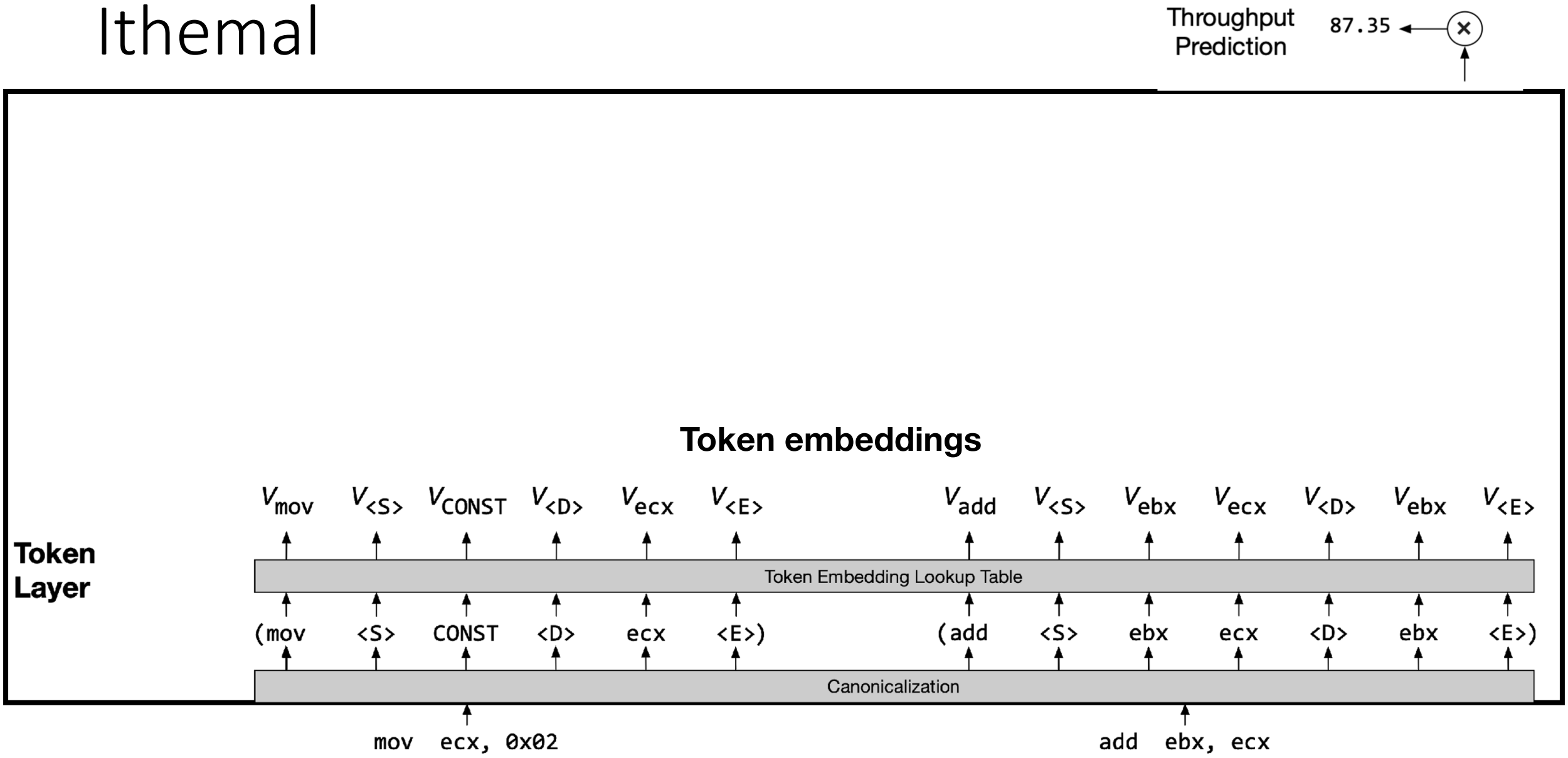


**No featurization required**

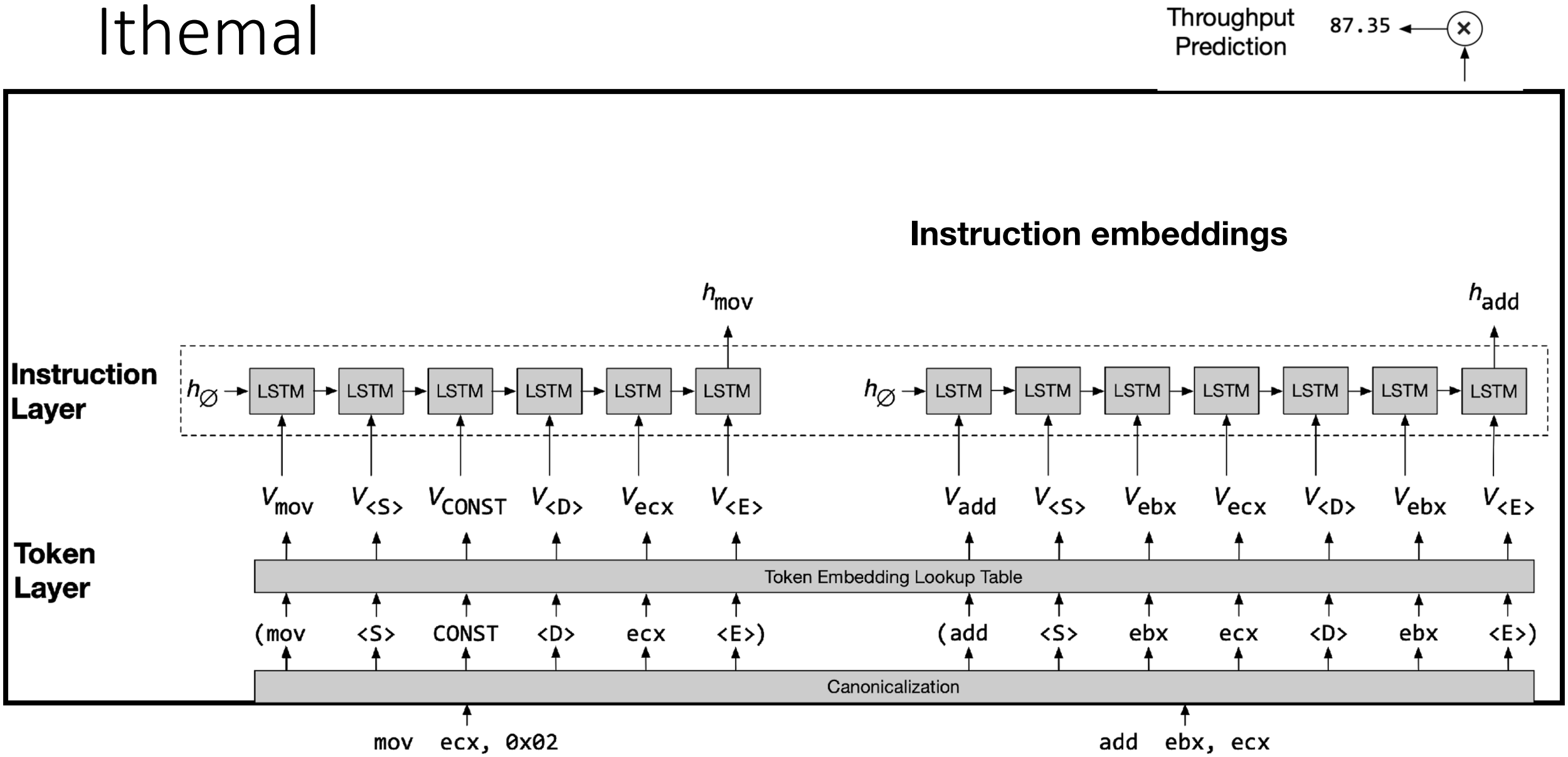
```
mov  ecx, 0x02
```

```
add  ebx, ecx
```

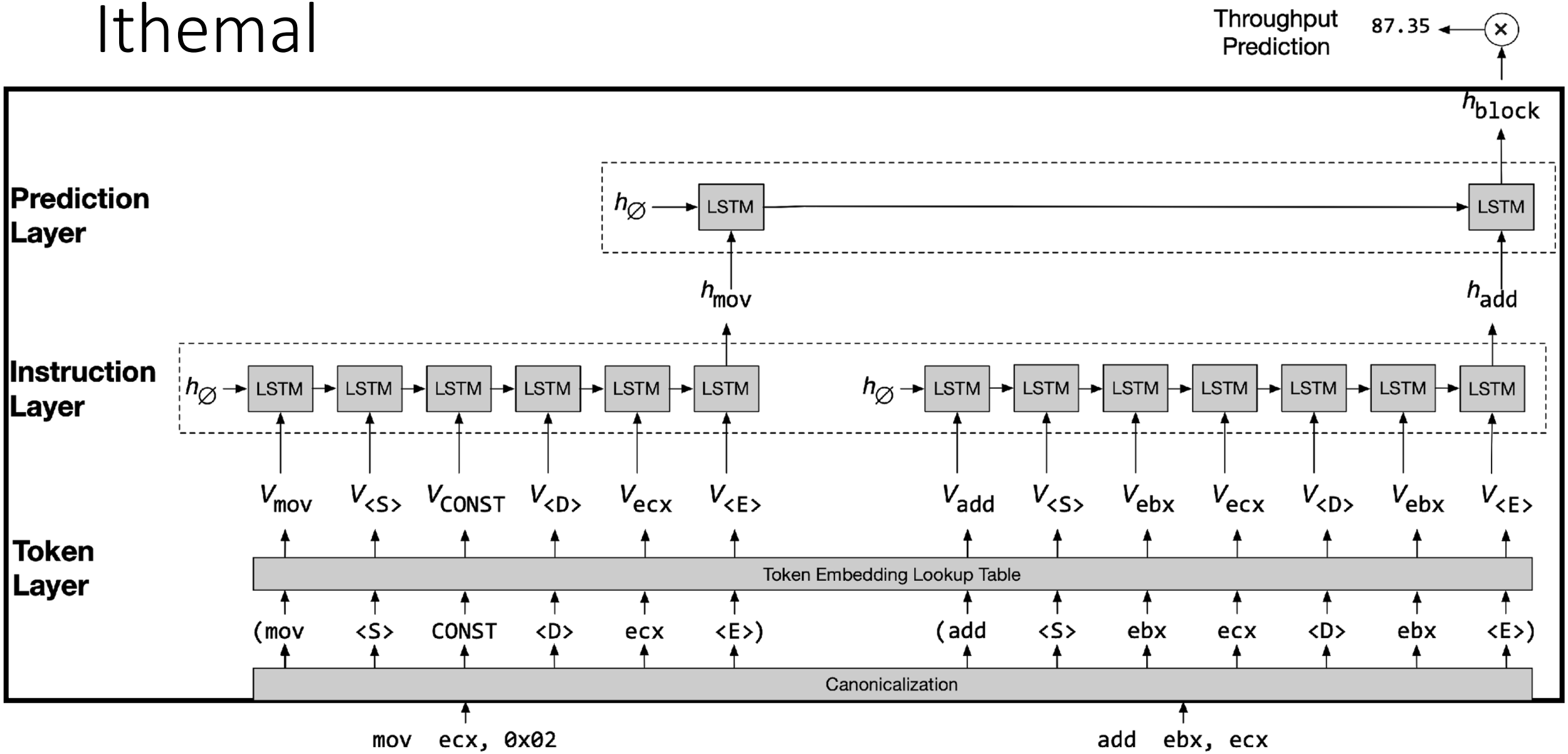
# Ithemal


Throughput Prediction
87.35
Token embeddings
Token Layer
$V_{mov}$ $V_{<S>}$ $V_{CONST}$ $V_{<D>}$ $V_{ecx}$ $V_{<E>}$ $V_{add}$ $V_{<S>}$ $V_{ebx}$ $V_{ecx}$ $V_{<D>}$ $V_{ebx}$ $V_{<E>}$
Token Embedding Lookup Table
(mov <S> CONST <D> ecx <E>) (add <S> ebx ecx <D> ebx <E>)
Canonicalization
mov ecx, 0x02
add ebx, ecx

# Ithemal


Throughput Prediction
87.35
Instruction embeddings
Instruction Layer
Token Layer
LSTM
Token Embedding Lookup Table
Canonicalization
(mov <S> CONST <D> ecx <E>)
(add <S> ebx ecx <D> ebx <E>)
mov ecx, 0x02
add ebx, ecx

# Ithemal

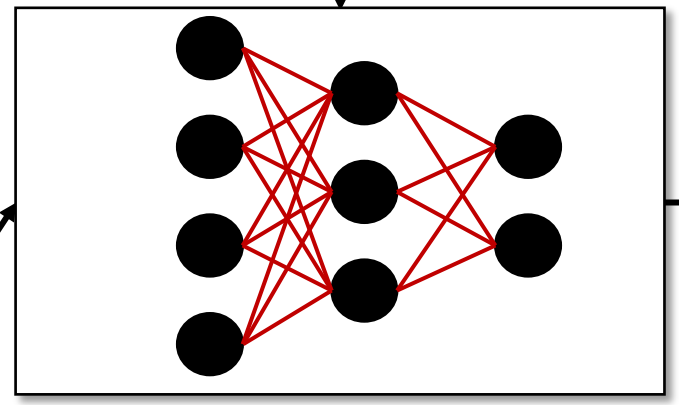

Throughput Prediction
87.35
×
$h_{block}$
Prediction Layer
$h_\varnothing$
LSTM
LSTM
$h_{mov}$
$h_{add}$
Instruction Layer
$h_\varnothing$
LSTM LSTM LSTM LSTM LSTM LSTM
$h_\varnothing$
LSTM LSTM LSTM LSTM LSTM LSTM LSTM
$V_{mov}$ $V_{<S>}$ $V_{CONST}$ $V_{<D>}$ $V_{ecx}$ $V_{<E>}$
$V_{add}$ $V_{<S>}$ $V_{ebx}$ $V_{ecx}$ $V_{<D>}$ $V_{ebx}$ $V_{<E>}$
Token Layer
Token Embedding Lookup Table
(mov <S> CONST <D> ecx <E>)
(add <S> ebx ecx <D> ebx <E>)
Canonicalization
mov ecx, 0x02
add ebx, ecx

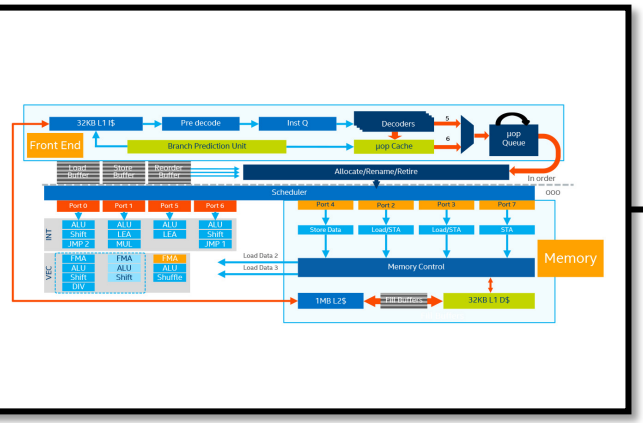
# Ithemal

Refine

Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```





# Differentiable Programming

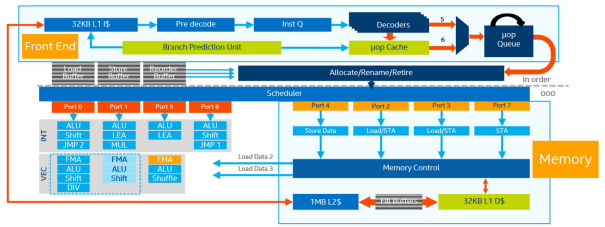
Refine

Dataset

```
mov ebx, [ecx]
add ecx, ebx
shl eax, 0x02
add eax, 0x01
shr eax, 0x04
```

Decoder

Scheduler



# Differentiable Programming

Use **search/optimization** to assist development of **data-integrated** software components that have **approximate** correctness specifications.

# Research Hypothesis

We as systems researchers should think about approximation.

New opportunities:

- Resilience
- Performance
- **Capability**

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

Systems Building Challenges:

- Performance
- Semantics (Real-valued, Differentiable, Probabilistic, Programs)
- Correctness

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

Systems Building Challenges:

- **Performance**
- Semantics (Real-valued, Differentiable, Probabilistic, Programs)
- Correctness

# Neural Networks are Large

# Neural Networks are Massive

# Neural Networks are Massive

**Language Models are Unsupervised Multitask Learners**

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested language modeling datasets in a zero-shot setting.

# Background: Network Pruning

prune /pru:n/ verb

To reduce the extent of [a neural network] by removing superfluous or unwanted parts.

(Oxford English Dictionary)

# Background: Network Pruning

*Optimal Brain Damage*

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

**ABSTRACT**

We have used information-theoretic ideas to derive a class of practical and nearly optimal schemes for adapting the size of a neural network. By removing unimportant weights from a network, several improvements can be expected: better generalization, fewer training examples required, and improved speed of learning and/or classification. The basic idea is to use second-derivative information to make a tradeoff between network complexity and training set error. Experiments confirm the usefulness of the methods on a real-world application.

# Background: Network Pruning

**Learning both Weights and Connections for Efficient Neural Networks**

**Song Han**
Stanford University
songhan@stanford.edu

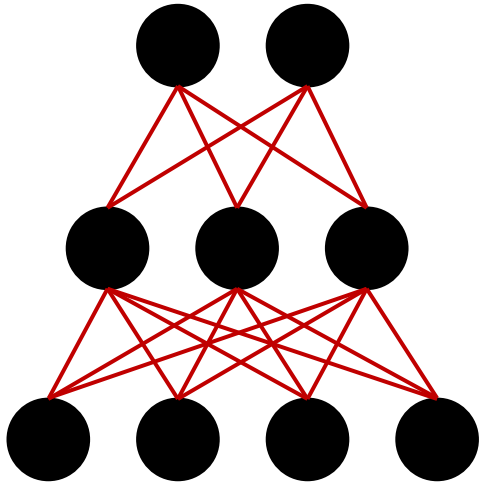
**Jeff Pool**
NVIDIA
jpool@nvidia.com

**John Tran**
NVIDIA
johntran@nvidia.com

**William J. Dally**
Stanford University
NVIDIA
dally@stanford.edu

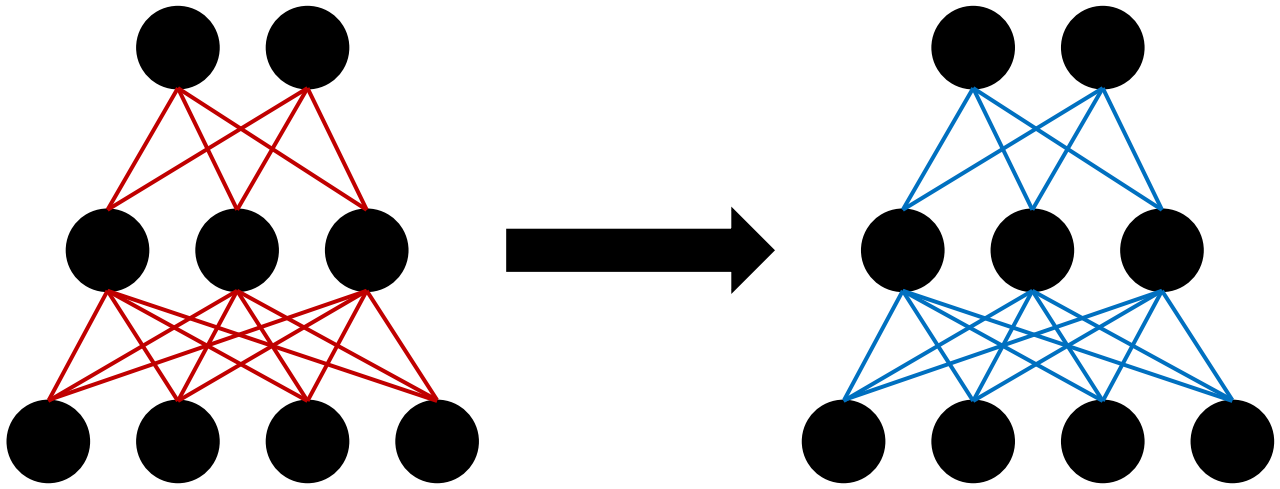
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



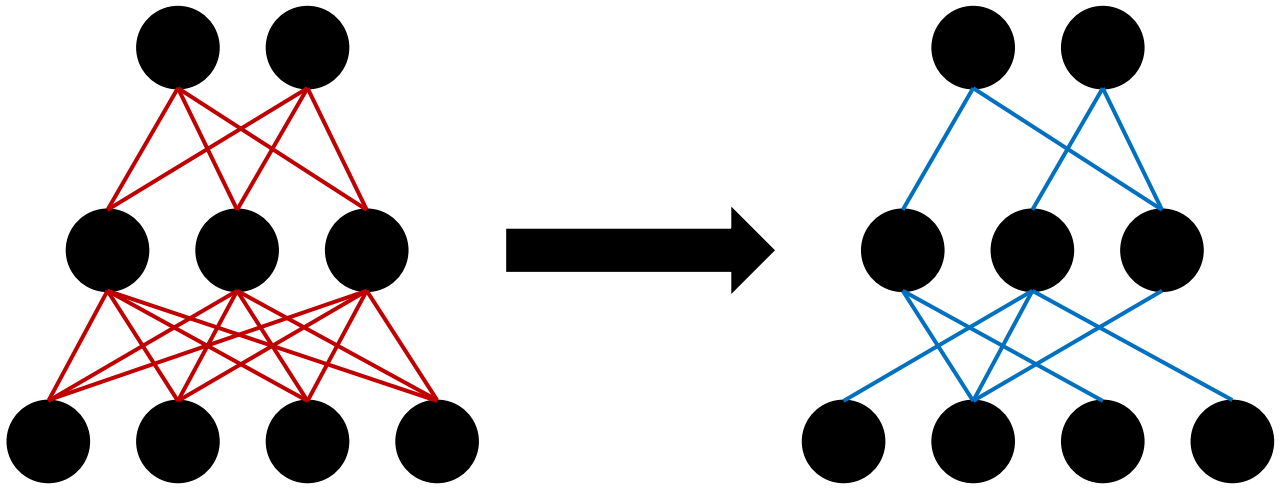
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



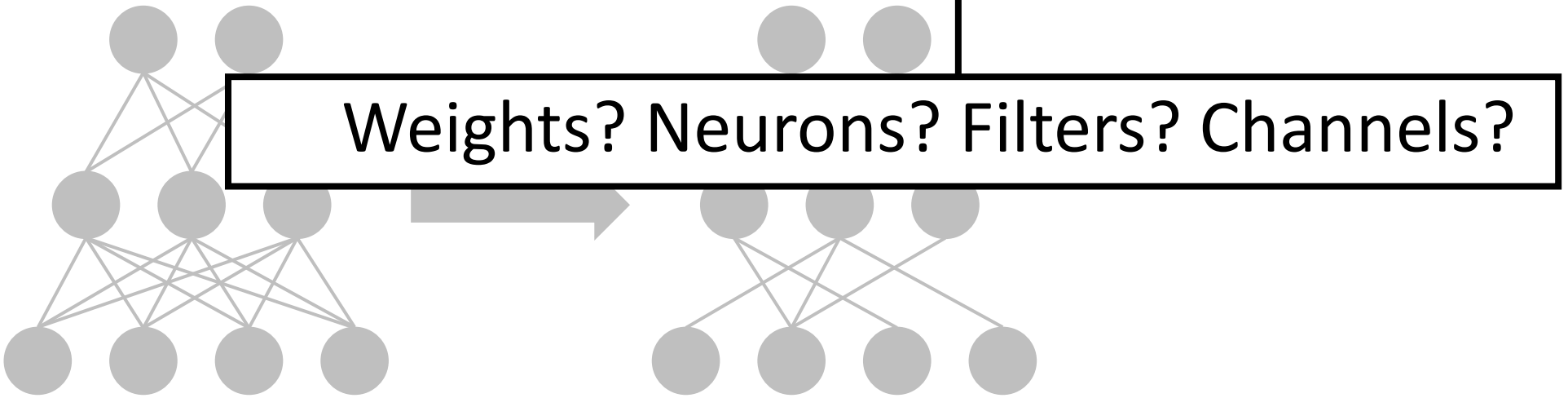
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



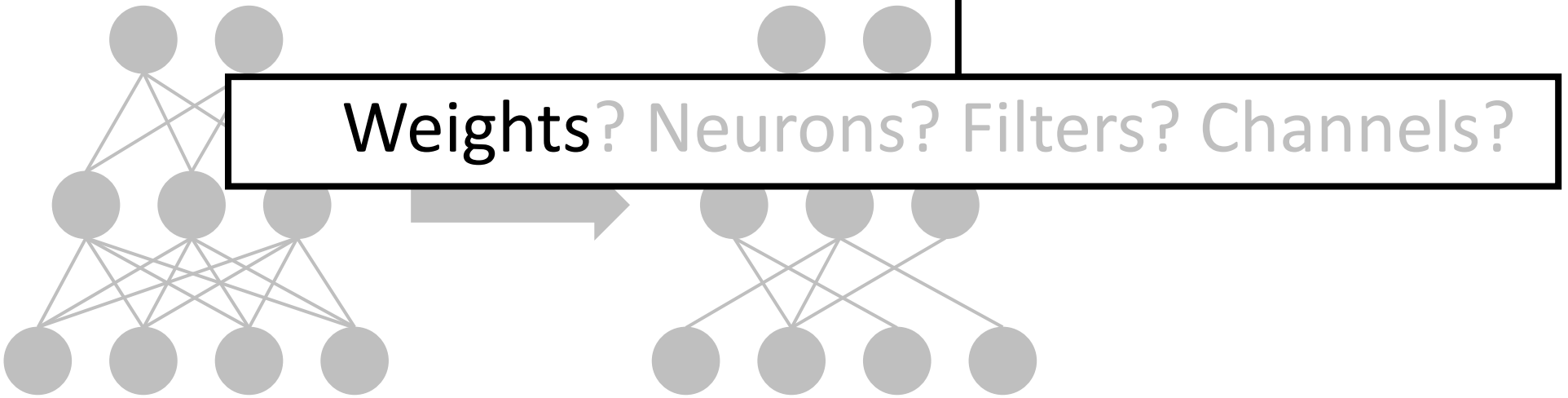
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



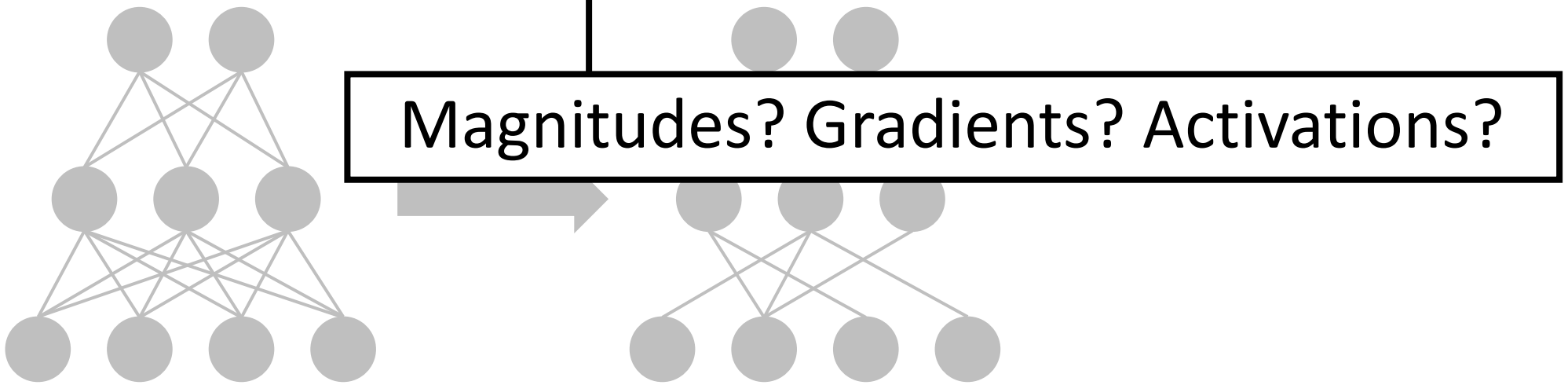
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



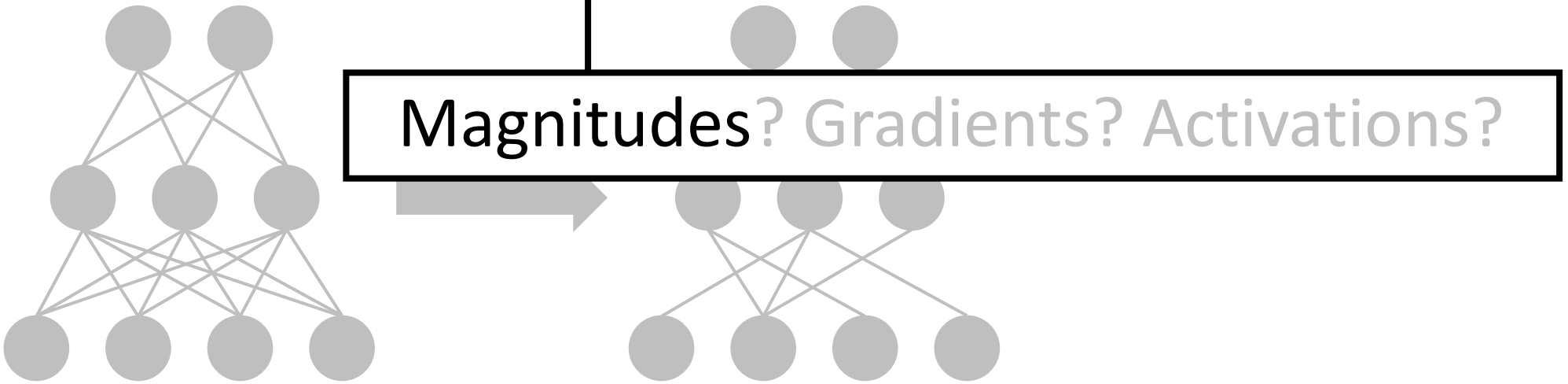
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



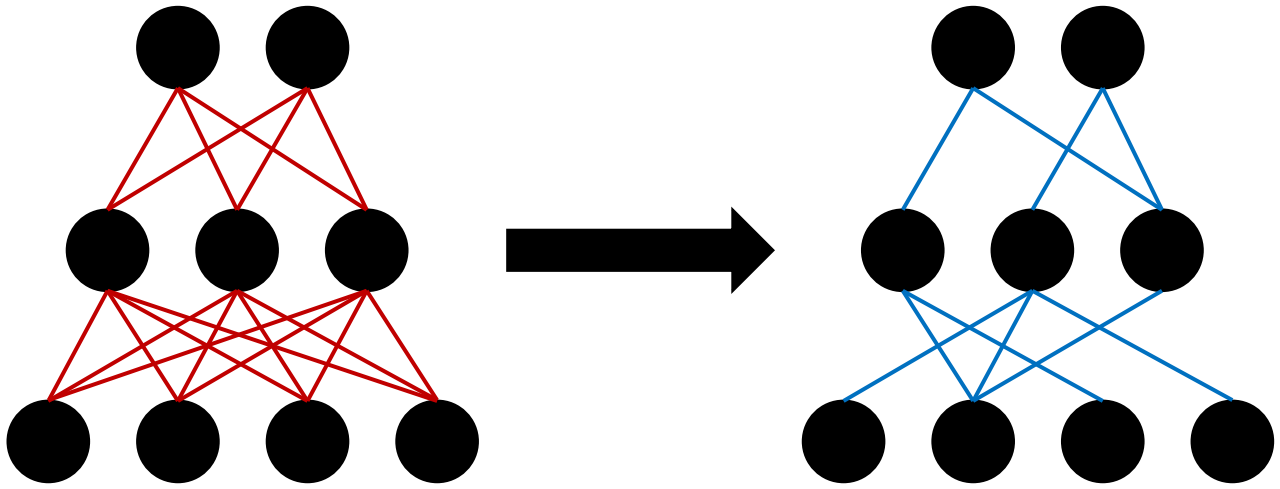
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



# Neural Network: as Code

```
float fully_connected_layer(float x[], float weights[][], float output[])
{
   for (uint i = 0; i < output.length; ++i)
   {
      for (uint j = 0; j < x.length; ++j)
      {

            output[i] += weights[i][j] * x[j];

      }
   }
}
```

# Neural Network: Pruning

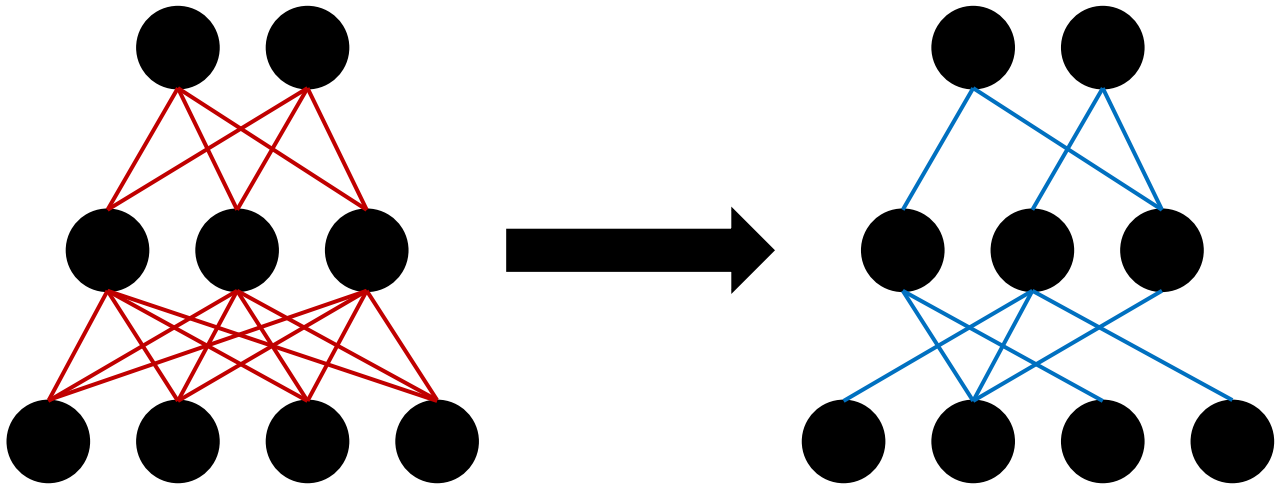
```
float fully_connected_layer(float x[], float weights[][], float output[])
{
   for (uint i = 0; i < output.length; ++i)
   {
      for (uint j = 0; j < x.length; ++j)
      {
          if (mask[i][j]) {
             output[i] += weights[i][j] * x[j];
          }
      }
   }
}
```

# Neural Network: Loop Perforation

```
float fully_connected_layer(float x[], float weights[][], float output[])
{
   for (uint i = 0; i < output.length; ++i)
   {
      for (uint j = 0; j < x.length; ++j)
      {
          if (mask[i][j]) {
             output[i] += weights[i][j] * x[j];
          }
      }
   }
}
```

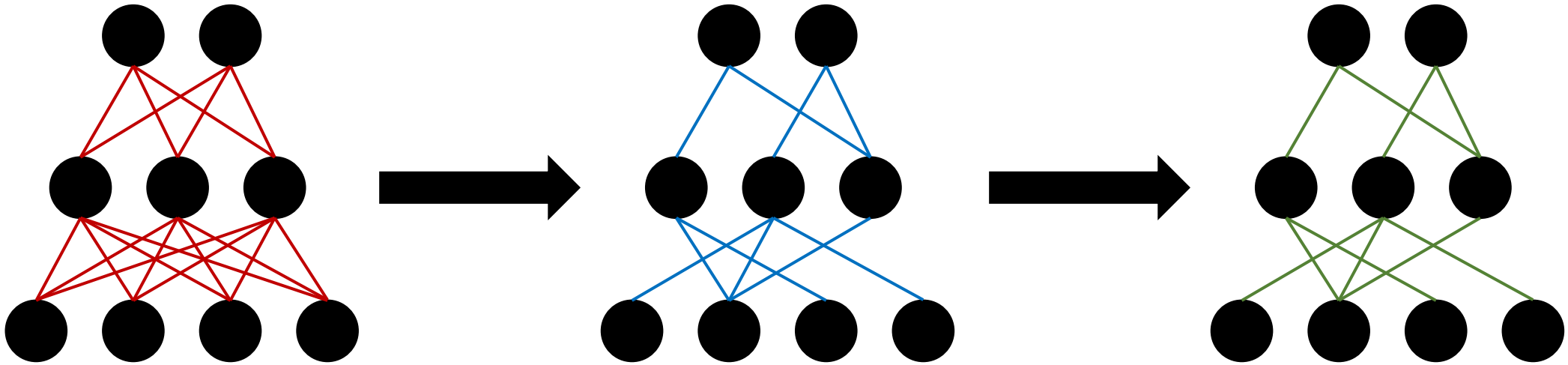
# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



# Background: Network Pruning

1) Train the network
2) Remove superfluous structure
3) Fine-tune the network



# Background: Network Pruning

**Learning both Weights and Connections for Efficient Neural Networks**

| Network | Top-1 Error | Top-5 Error | Parameters | Compression Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 267K | |
| LeNet-300-100 Pruned | 1.59% | - | **22K** | **12**× |
| LeNet-5 Ref | 0.80% | - | 431K | |
| LeNet-5 Pruned | 0.77% | - | **36K** | **12**× |
| AlexNet Ref | 42.78% | 19.73% | 61M | |
| AlexNet Pruned | 42.77% | 19.67% | **6.7M** | **9**× |
| VGG-16 Ref | 31.50% | 11.32% | 138M | |
| VGG-16 Pruned | 31.34% | 10.88% | **10.3M** | **13**× |

# Background: Network Pruning

**Learning both Weights and Connections for Efficient Neural Networks**

| Network | Top-1 Error | Top-5 Error | Parameters | Compression Rate |
|---|---|---|---|---|
| LeNet-300-100 Ref | 1.64% | - | 267K | |
| LeNet-300-100 Pruned | 1.59% | - | **22K** | **12**× |
| LeNet-5 Ref | 0.80% | - | 431K | |
| LeNet-5 Pruned | 0.77% | - | **36K** | **12**× |
| AlexNet Ref | 42.78% | 19.73% | 61M | |
| AlexNet Pruned | 42.77% | 19.67% | **6.7M** | **9**× |
| VGG-16 Ref | 31.50% | 11.32% | 138M | |
| VGG-16 Pruned | 31.34% | 10.88% | **10.3M** | **13**× |

# Training is Expensive

# Training is Really Expensive

| TPU type (v2) | v2 cores | Total memory | Evaluation price (USD) | 1-year commitment price (USD) | 3-year commitment price (USD) |
|---|---|---|---|---|---|
| v2-32 **(Beta)** | 32 | 256 GiB | $24 / hour | $11,038 / month | $7,884 / month |
| v2-128 **(Beta)** | 128 | 1 TiB | $96 / hour | $44,151 / month | $31,536 / month |
| v2-256 **(Beta)** | 256 | 2 TiB | $192 / hour | $88,301 / month | $63,072 / month |
| v2-512 **(Beta)** | 512 | 4 TiB | $384 / hour | $176,602 / month | $126,144 / month |

# Training is Really Expensive

| TPU type (v2) | v2 cores | Total memory | Evaluation price (USD) | 1-year commitment price (USD) | 3-year commitment price (USD) |
|---|---|---|---|---|---|
| v2-32 (Beta) | 32 | 256 GiB | $24 / hour | $11,038 / month | $7,884 / month |
| v2-128 (Beta) | 128 | 1 TiB | $96 / hour | $44,151 / month | $31,536 / month |
| v2-256 (Beta) | 256 | 2 TiB | $192 / hour | $88,301 / month | $63,072 / month |
| v2-512 (Beta) | 512 | 4 TiB | $384 / hour | $176,602 / month | $126,144 / month |

# Training is Really Expensive

**Energy and Policy Considerations for Deep Learning in NLP**

**Emma Strubell** **Ananya Ganesh** **Andrew McCallum**
College of Information and Computer Sciences
University of Massachusetts Amherst
{strubell, aganesh, mccallum}@cs.umass.edu

| Model | Hardware | Power (W) | Hours | kWh·PUE | $CO_2e$ | Cloud compute cost |
|---|---|---|---|---|---|---|
| $\text{Transformer}_{base}$ | P100x8 | 1415.78 | 12 | 27 | 26 | \$41–\$140 |
| $\text{Transformer}_{big}$ | P100x8 | 1515.43 | 84 | 201 | 192 | \$289–\$981 |
| ELMo | P100x3 | 517.66 | 336 | 275 | 262 | \$433–\$1472 |
| $\text{BERT}_{base}$ | V100x64 | 12,041.51 | 79 | 1507 | 1438 | \$3751–\$12,571 |
| $\text{BERT}_{base}$ | TPUv2x16 | — | 96 | — | — | \$2074–\$6912 |
| NAS | P100x8 | 1515.43 | 274,120 | 656,347 | 626,155 | \$942,973–\$3,201,722 |
| NAS | TPUv2x1 | — | 32,623 | — | — | \$44,055–\$146,848 |
| GPT-2 | TPUv3x32 | — | 168 | — | — | \$12,902–\$43,008 |

# Training is Really Expensive

**Energy and Policy Considerations for Deep Learning in NLP**

**Emma Strubell** **Ananya Ganesh** **Andrew McCallum**
College of Information and Computer Sciences
University of Massachusetts Amherst
{strubell, aganesh, mccallum}@cs.umass.edu

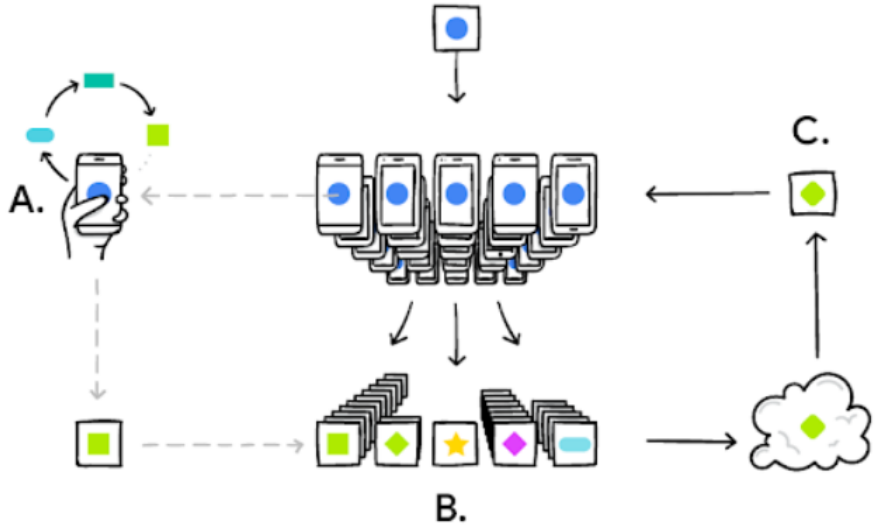
| Model | Hardware | Power (W) | Hours | kWh·PUE | $CO_2e$ | Cloud compute cost |
|---|---|---|---|---|---|---|
| $\text{Transformer}_{base}$ | P100x8 | 1415.78 | 12 | 27 | **26** | $41–$140 |
| $\text{Transformer}_{big}$ | P100x8 | 1515.43 | 84 | 201 | **192** | $289–$981 |
| ELMo | P100x3 | 517.66 | 336 | 275 | **262** | $433–$1472 |
| $\text{BERT}_{base}$ | V100x64 | 12,041.51 | 79 | 1507 | **1438** | $3751–$12,571 |
| $\text{BERT}_{base}$ | TPUv2x16 | — | 96 | — | — | $2074–$6912 |
| NAS | P100x8 | 1515.43 | 274,120 | 656,347 | **626,155** | $942,973–$3,201,722 |
| NAS | TPUv2x1 | — | 32,623 | — | — | $44,055–$146,848 |
| GPT-2 | TPUv3x32 | — | 168 | — | — | $12,902–$43,008 |

# Training is Really Expensive

Federated Learning: Collaborative Machine Learning without Centralized Training Data

Thursday, April 6, 2017

Posted by Brendan McMahan and Daniel Ramage, Research Scientists



# Motivation and Questions

We can prune models after training.

# Motivation and Questions

We can prune models after training.

Can we train smaller models to begin with?

# Motivation and Questions

The pruned network can represent a function as accurate as the unpruned network.

# Motivation and Questions

The pruned network can represent a function as accurate as the unpruned network.

**What if we train the pruned network from the start?**

# Motivation and Questions

1) Randomly initialize the pruned network
2) Train it to convergence

# Motivation and Questions

1) Randomly initialize the pruned network
2) Train it to convergence

**Does not reach the same accuracy**

# Motivation and Questions

PRUNING FILTERS FOR EFFICIENT CONVNETS

**Hao Li***
University of Maryland
haoli@cs.umd.edu

**Asim Kadav**
NEC Labs America
asim@nec-labs.com

**Igor Durdanovic**
NEC Labs America
igord@nec-labs.com

**Hanan Samet**†
University of Maryland
hjs@cs.umd.edu

**Hans Peter Graf**
NEC Labs America
hpg@nec-labs.com

Training a pruned model from scratch performs worse than retraining a pruned model, which may indicate the difficulty of training a network with small capacity.

# Motivation and Questions

PRUNING FILTERS FOR EFFICIENT CONVNETS

**Hao Li***
University of Maryland
haoli@cs.umd.edu

**Asim Kadav**
NEC Labs America
asim@nec-labs.com

**Igor Durdanovic**
NEC Labs America
igord@nec-labs.com

**Hanan Samet**†
University of Maryland
hjs@cs.umd.edu

**Hans Peter Graf**
NEC Labs America
hpg@nec-labs.com

**Learning both Weights and Connections for Efficient Neural Networks**

**Song Han**
Stanford University
songhan@stanford.edu

**Jeff Pool**
NVIDIA
jpool@nvidia.com

**John Tran**
NVIDIA
johntran@nvidia.com

**William J. Dally**
Stanford University
NVIDIA
dally@stanford.edu

Training a pruned model from scratch performs worse than retraining a pruned model, which may indicate the difficulty of training a network with small capacity.

During retraining, it is better to retain the weights from the initial training phase for the connections that survived pruning than it is to reinitialize the pruned layers.

# Motivation and Questions

Can we train (sparsely) pruned networks from scratch?

# Motivation and Questions

Can we train (sparsely) pruned networks from scratch?

Corollary: Do networks have to be so overparameterized to learn?

# Motivation and Questions

Can we train (sparsely) pruned networks from scratch?

**Yes**

Corollary: Do networks have to be so overparameterized to learn?

**No?**

The Lottery Ticket Hypothesis. Frankle and Carbin. ICLR ‘19 (Best Paper)

# Training Pruned Networks

TLDR: Weights pruned after training could have been pruned before training*

The Lottery Ticket Hypothesis. Frankle and Carbin. ICLR ‘19 (Best Paper)

# Training Pruned Networks

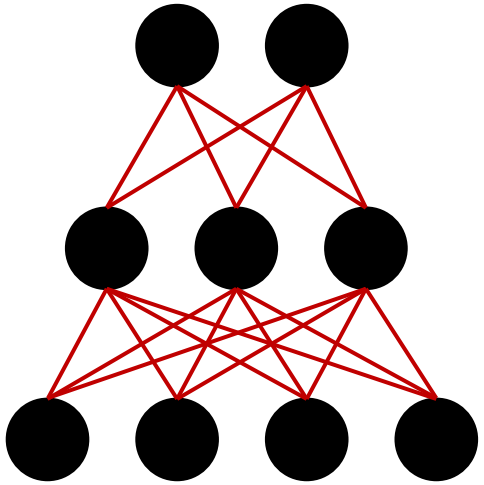
TLDR: Weights pruned after training could have been pruned before training*

* You need to use the same initializations

The Lottery Ticket Hypothesis. Frankle and Carbin. ICLR ‘19 (Best Paper)

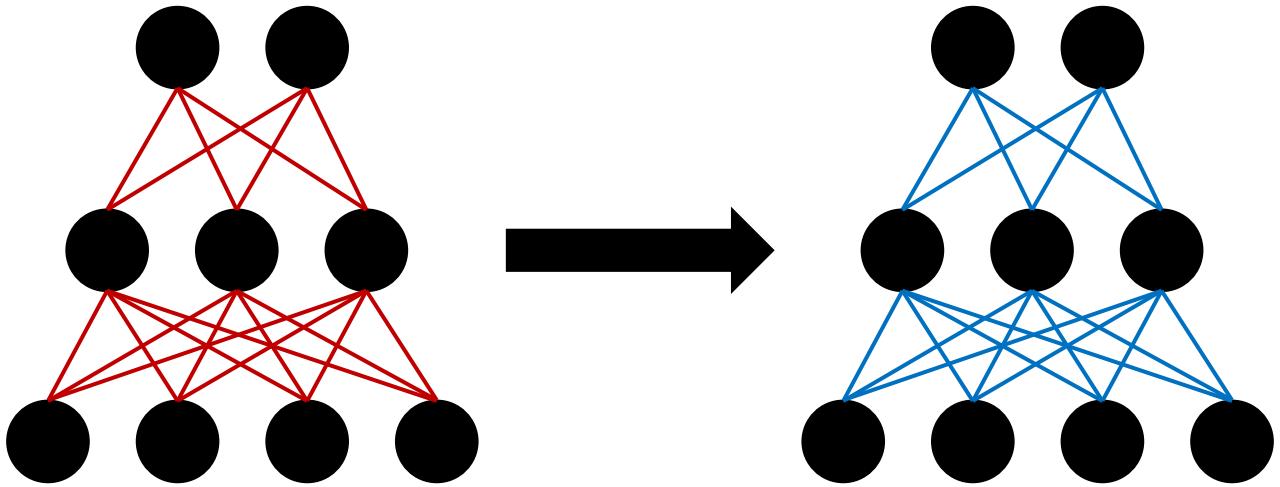
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1



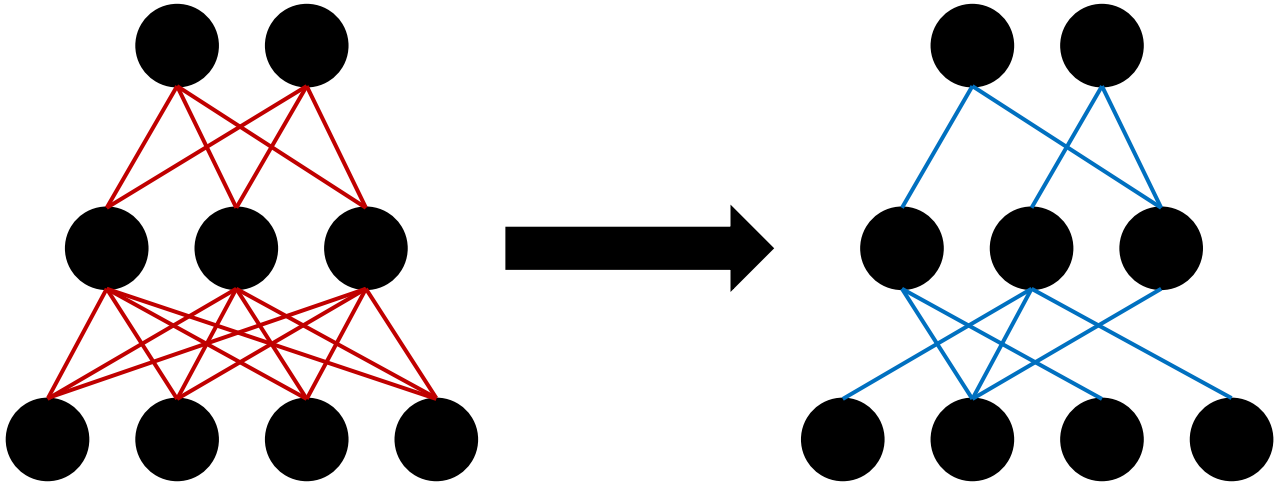
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1



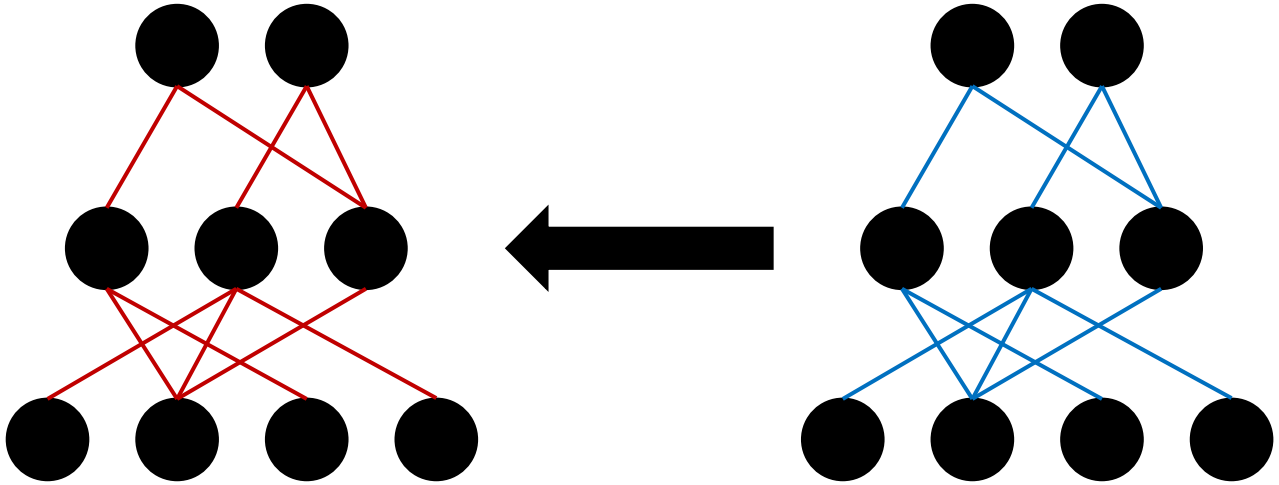
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1



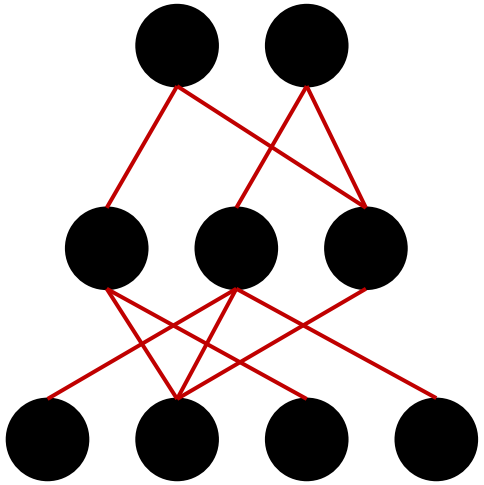
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1



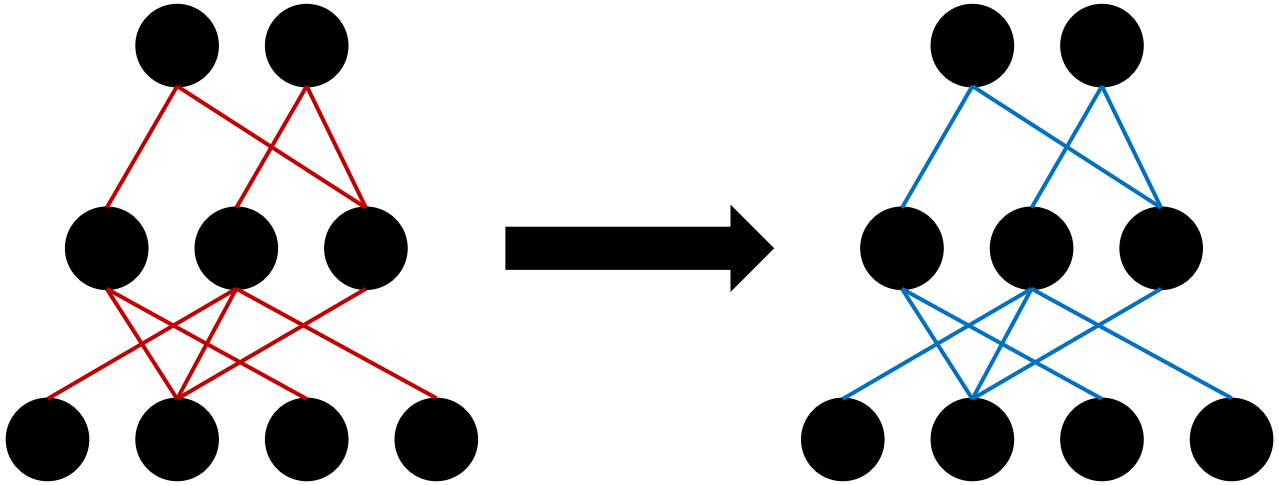
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1



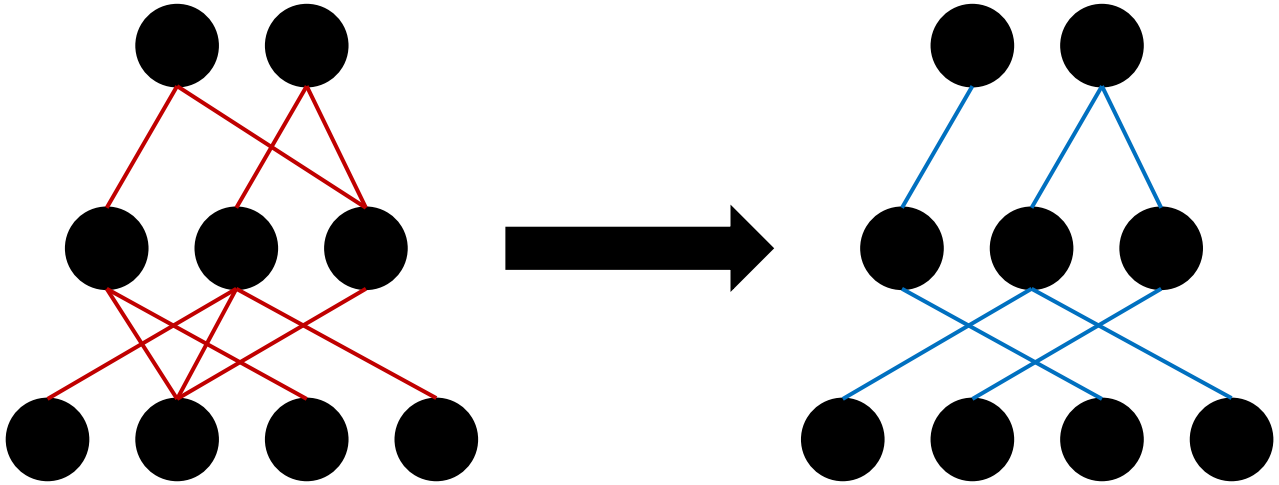
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1
4) Repeat steps 2-3 iteratively



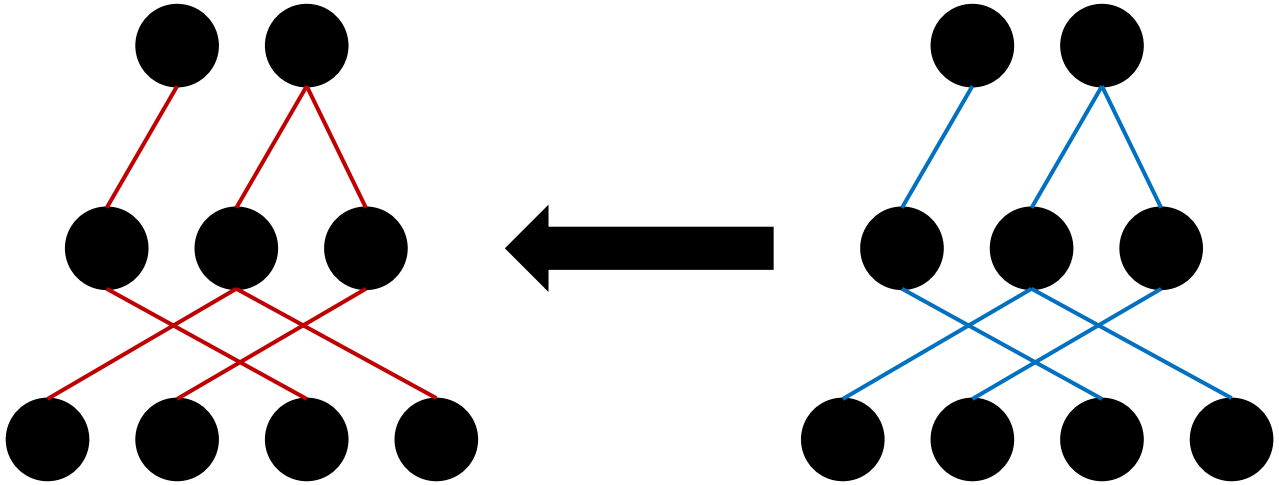
# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1
4) Repeat steps 2-3 iteratively

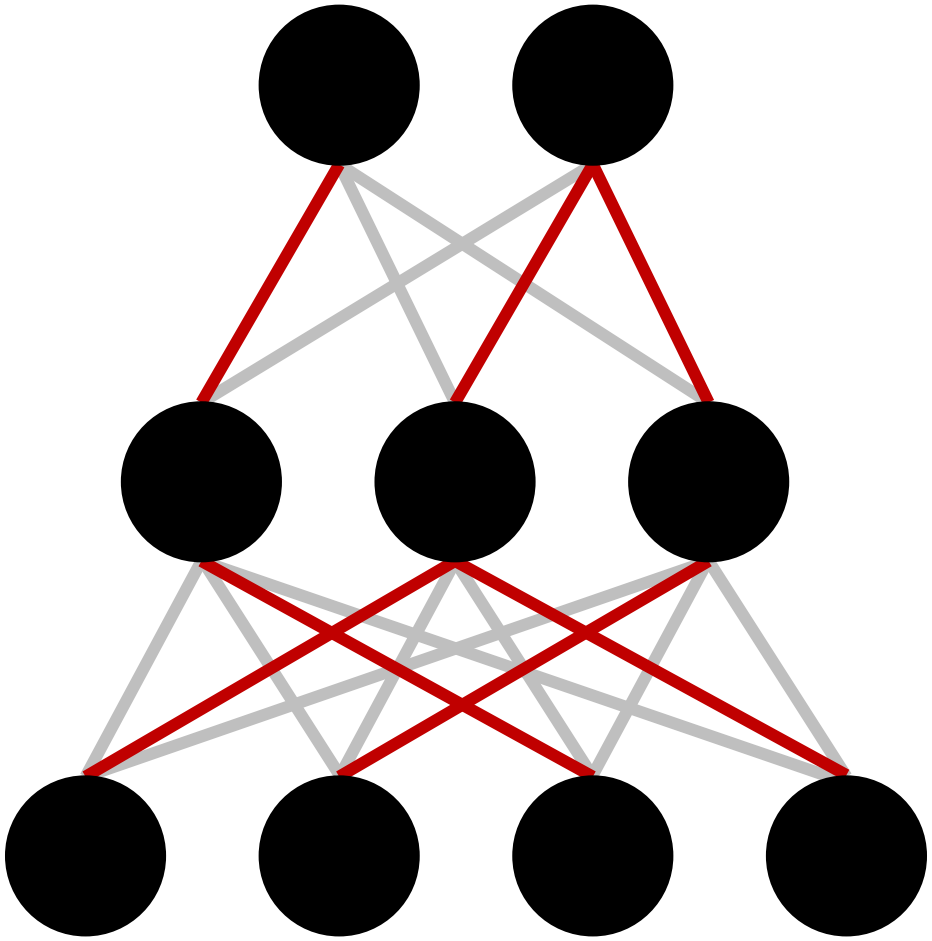


# Training Pruned Networks

1) Randomly initialize the full network
2) Train it and prune superfluous structure
3) Reset each remaining weight to its value from 1
4) Repeat steps 2-3 iteratively



# Training Pruned Networks



# Training Pruned Networks

We find sparse, trainable subnetworks for:

- A fully-connected network for MNIST
- Convolutional networks for CIFAR10
- Resnet-50 for ImageNet
- Dropout, weight decay, batchnorm, resnets, your favorite optimizer, etc. etc.

# Training Pruned Networks

We find sparse, trainable subnetworks for:

- A fully-connected network for MNIST
- Convolutional networks for CIFAR10
- Resnet-50 for ImageNet

**If you reinitialize, this doesn’t work**

# Training Pruned Networks

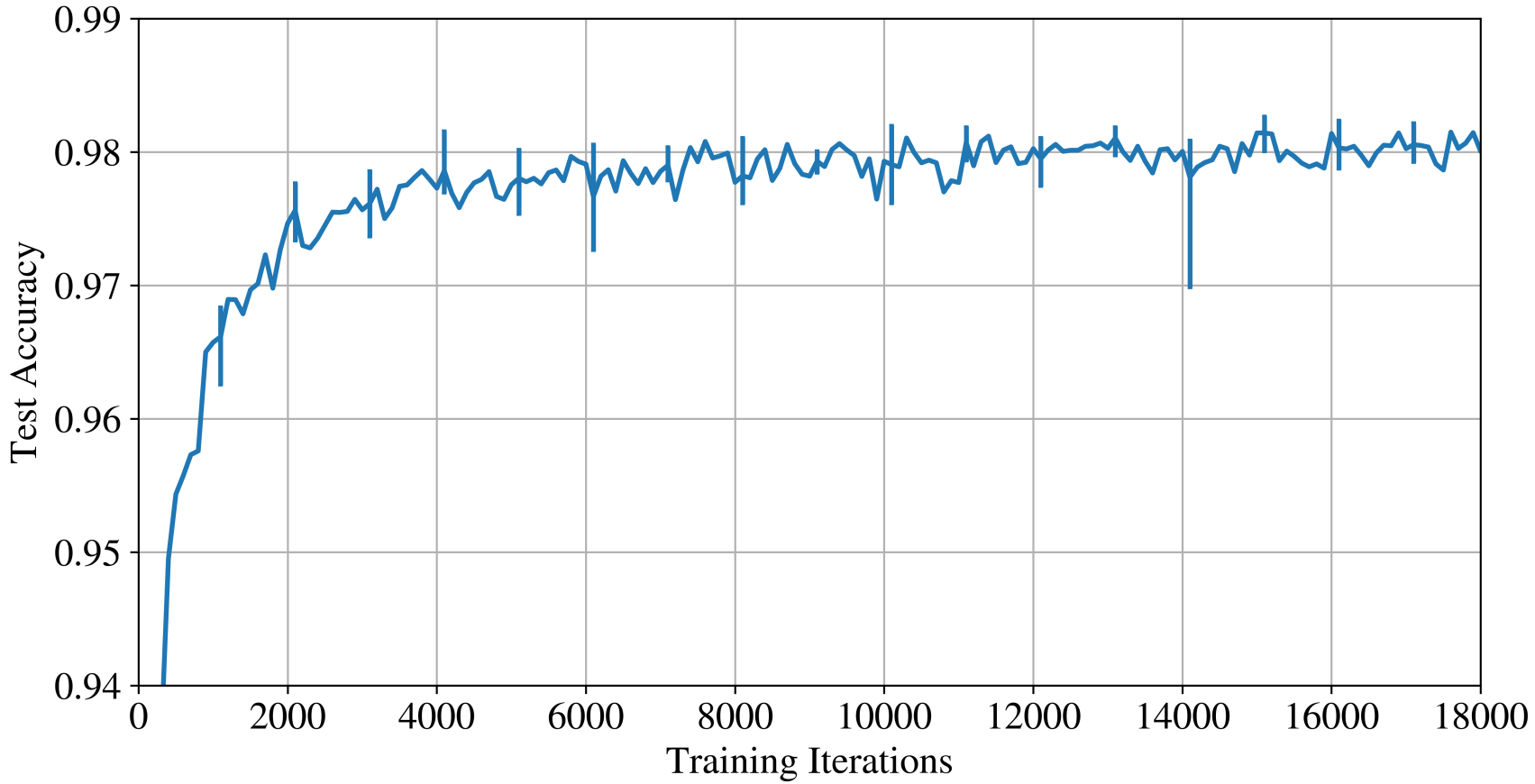
These subnetworks:

- Between 15% and 1% of the original size
- Learn faster than the original network
- Reach the same or higher test accuracy

Caveats:

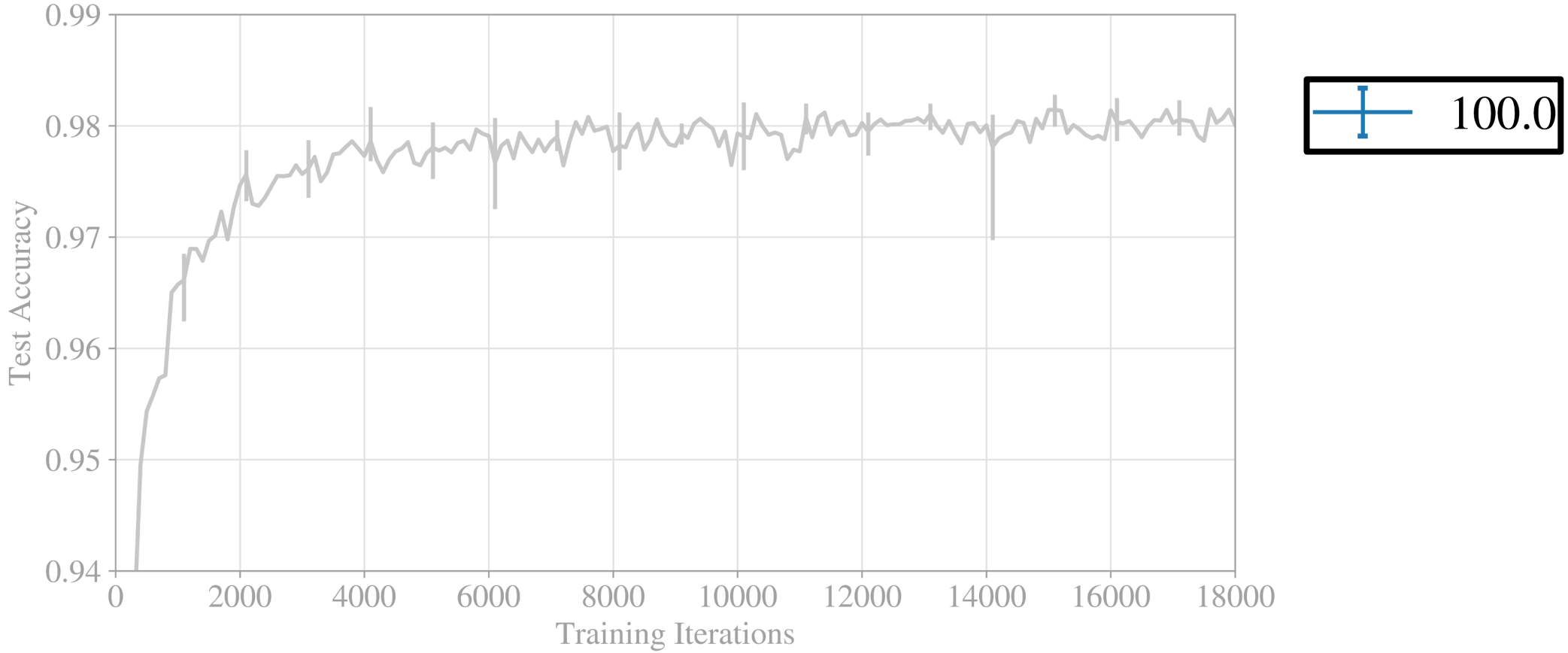
- Subnetworks are found retroactively
- Finding subnetworks is very expensive

# Results



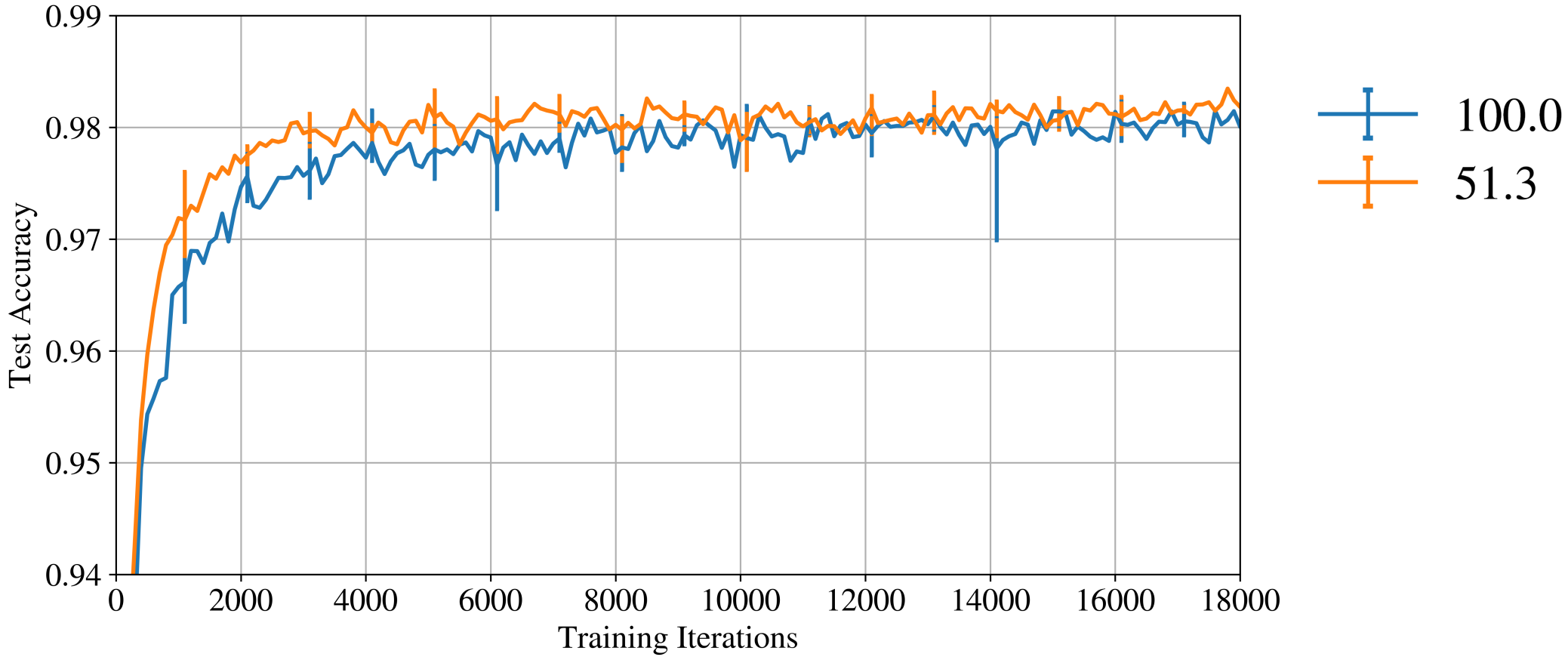

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



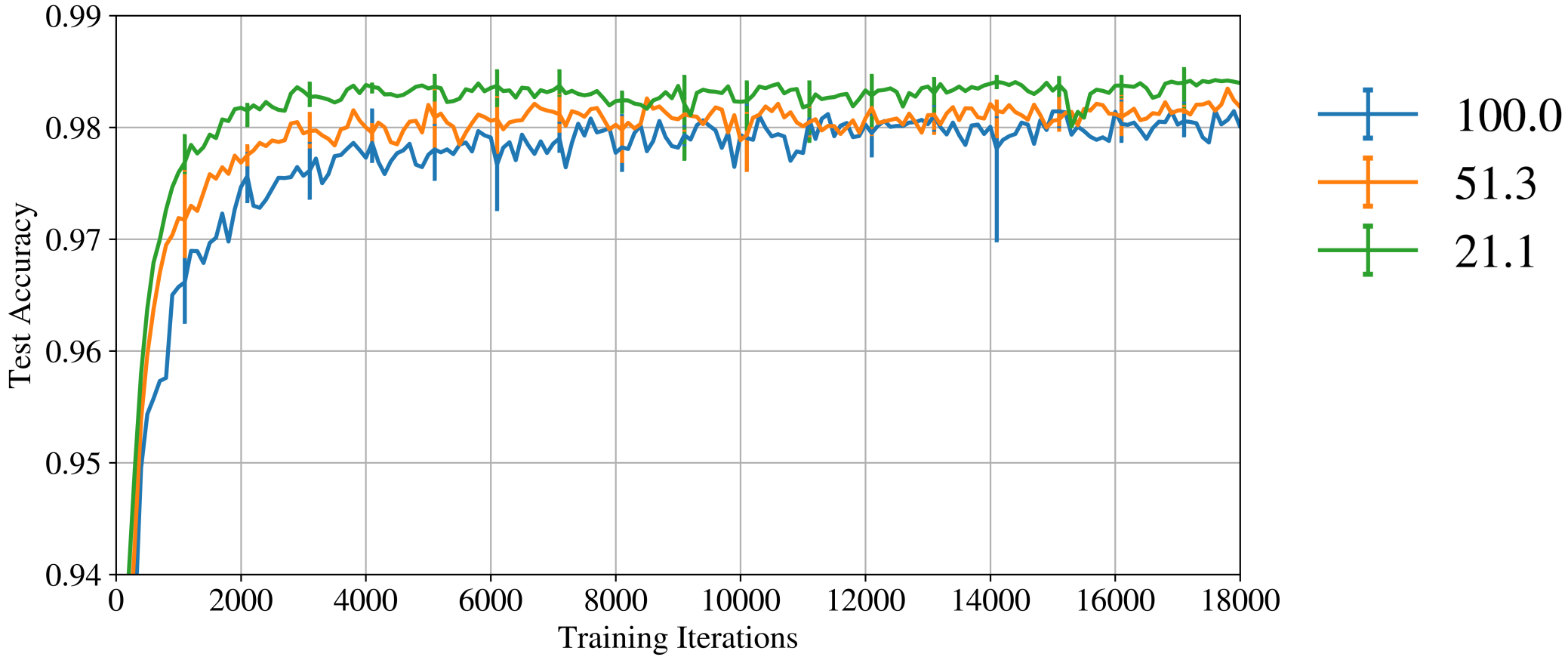

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



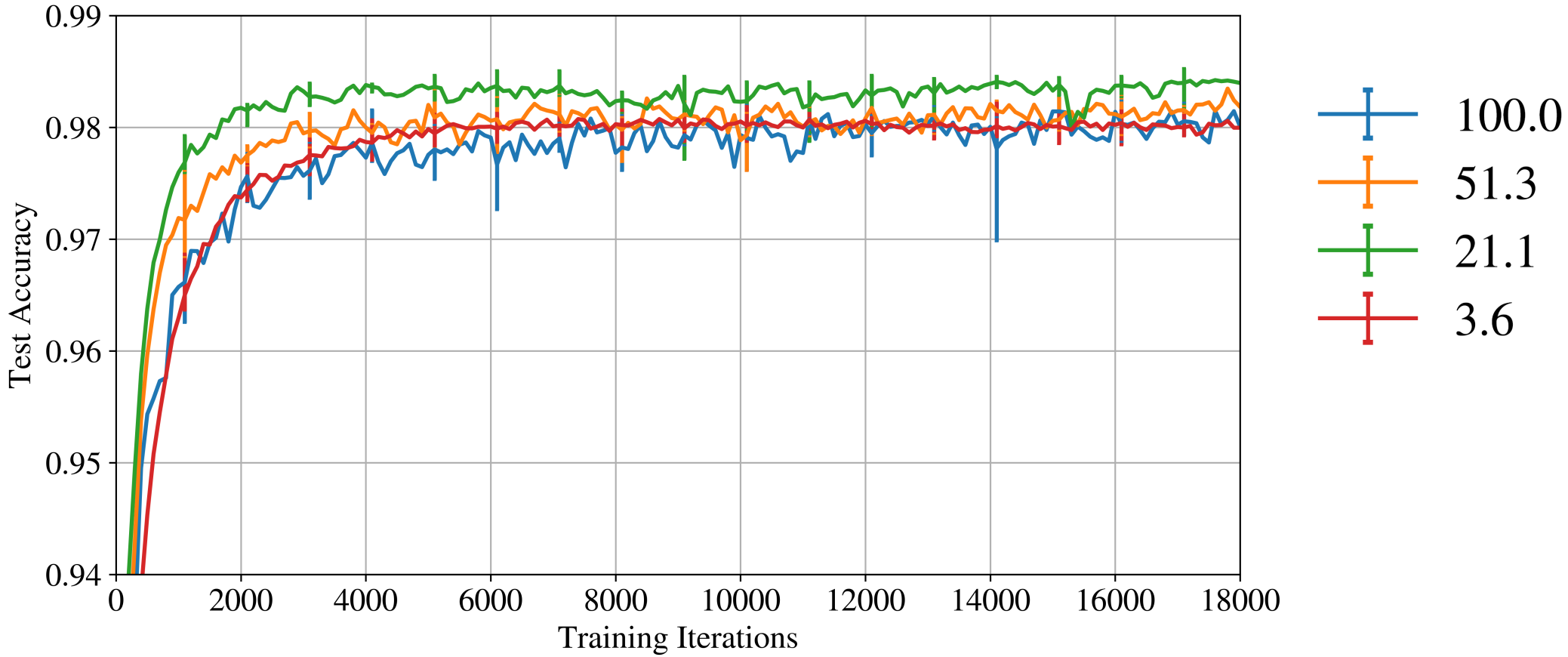

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



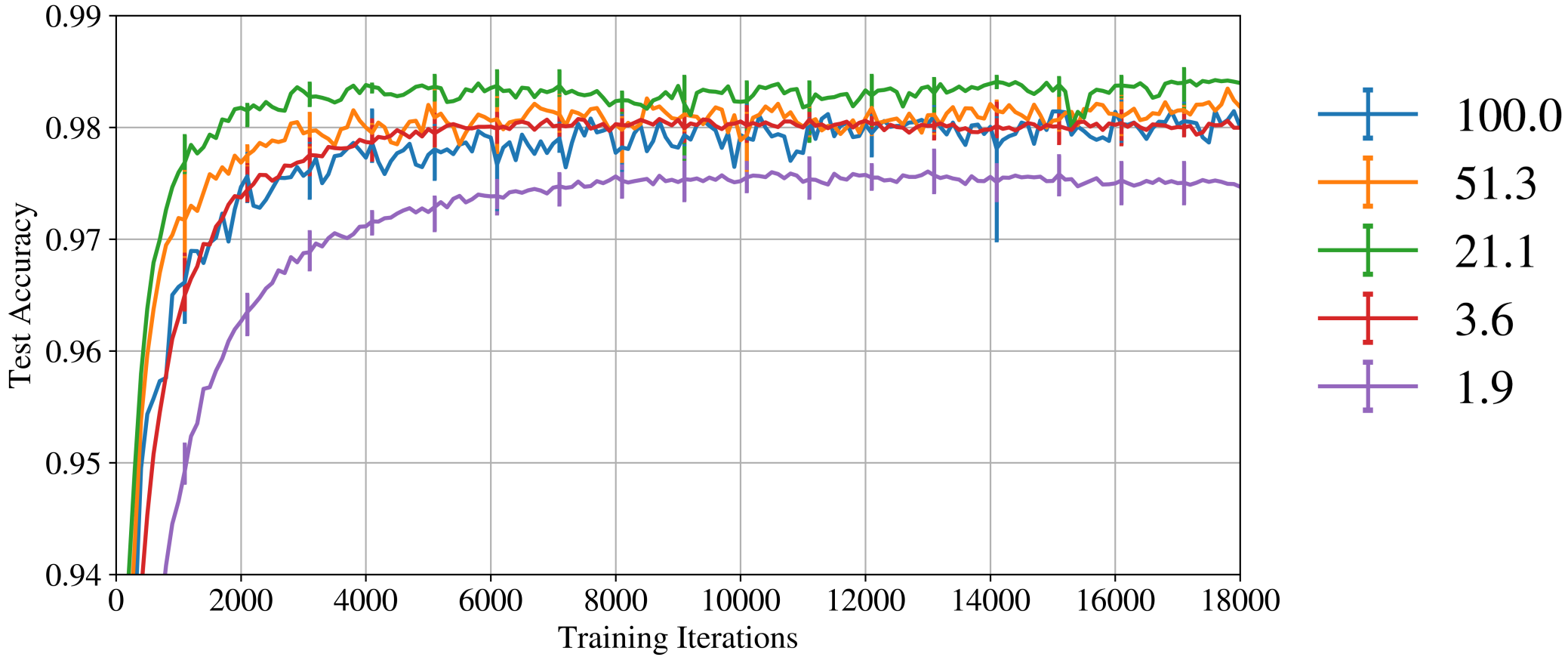

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



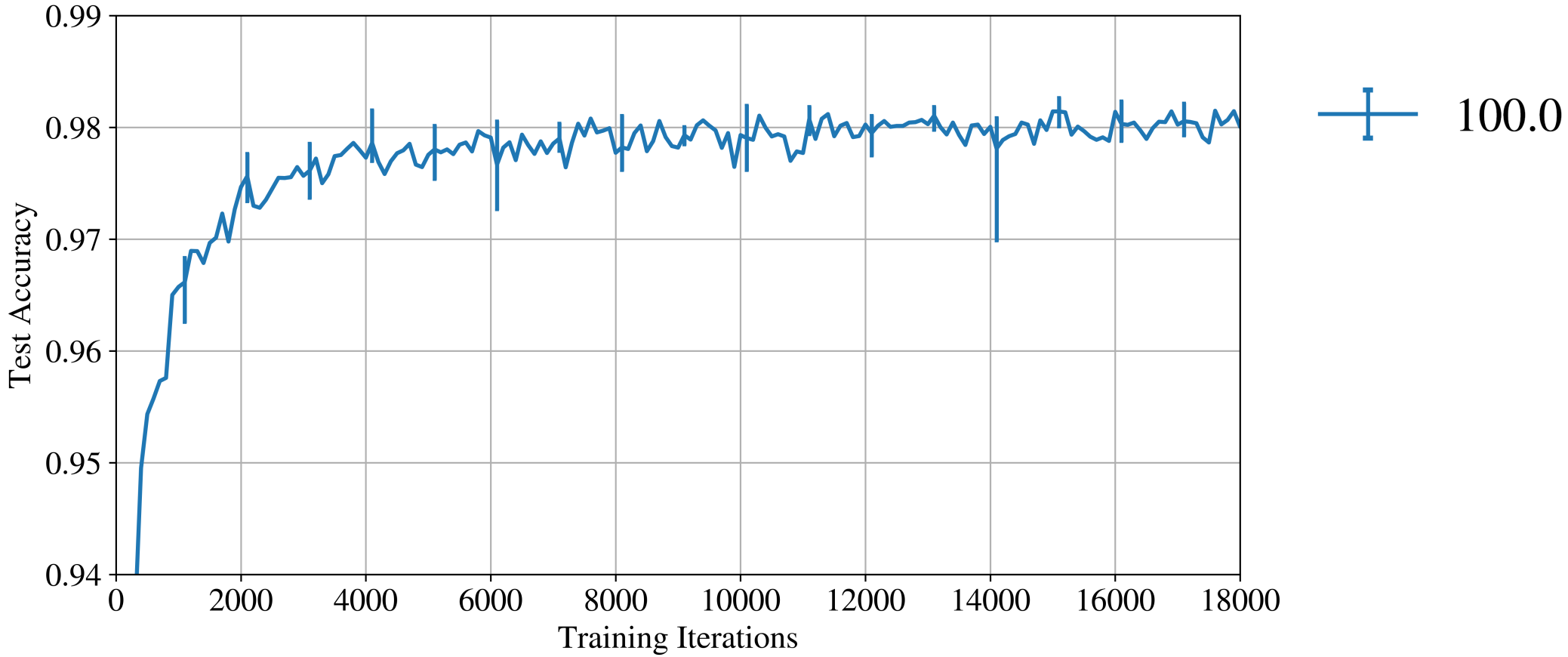

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



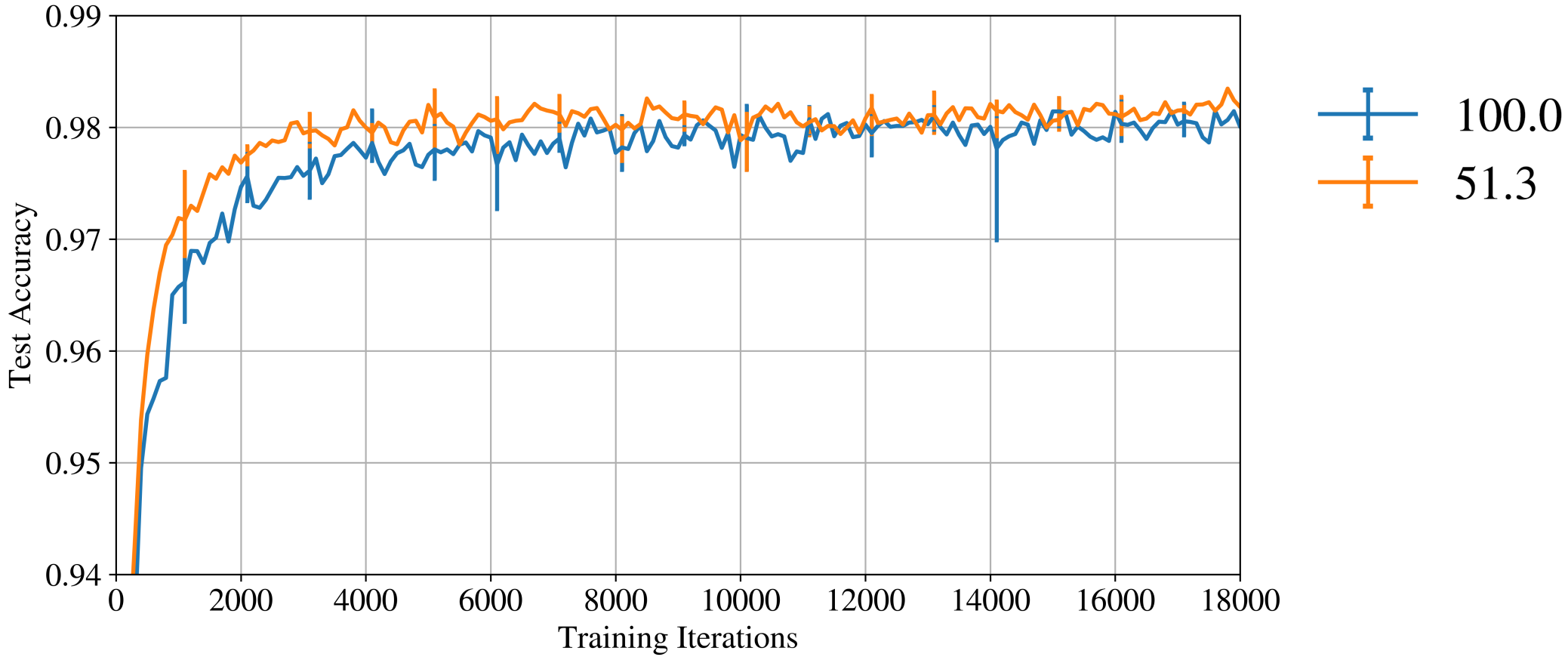

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



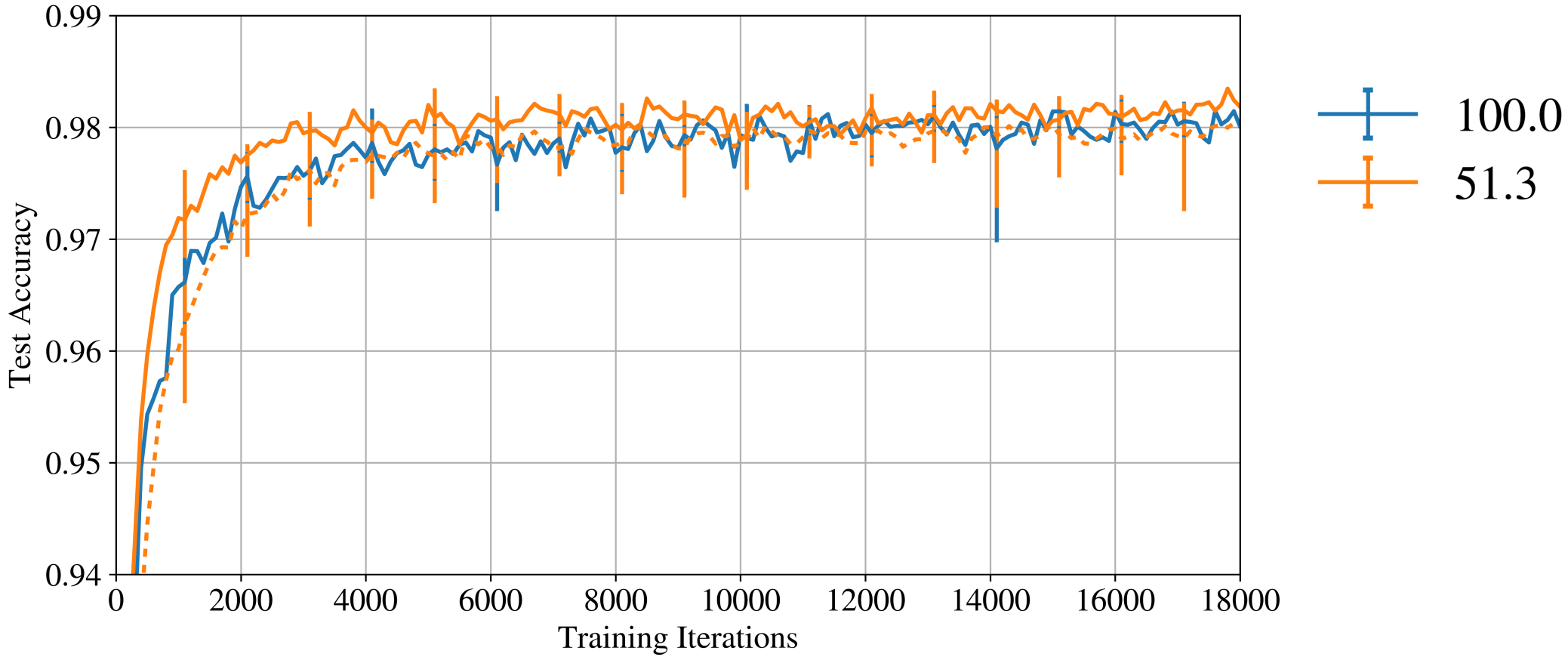

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



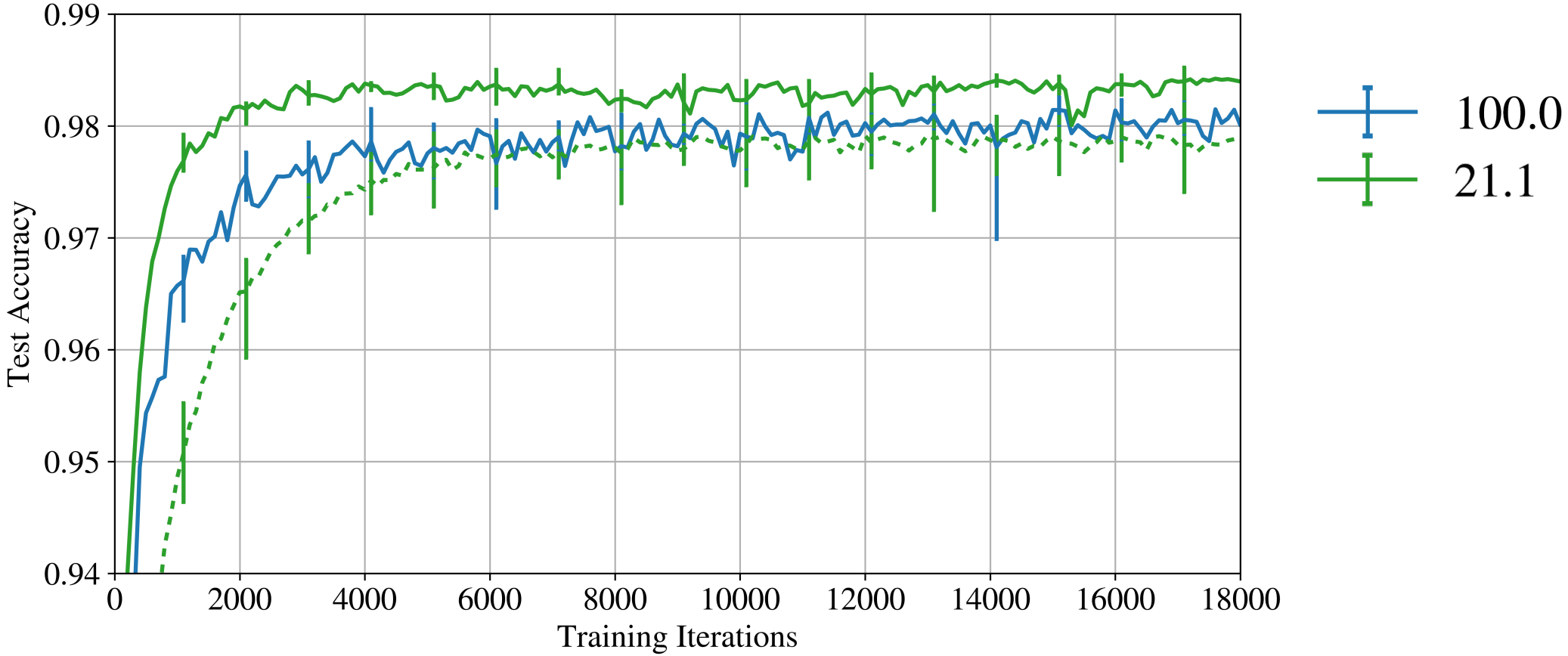

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



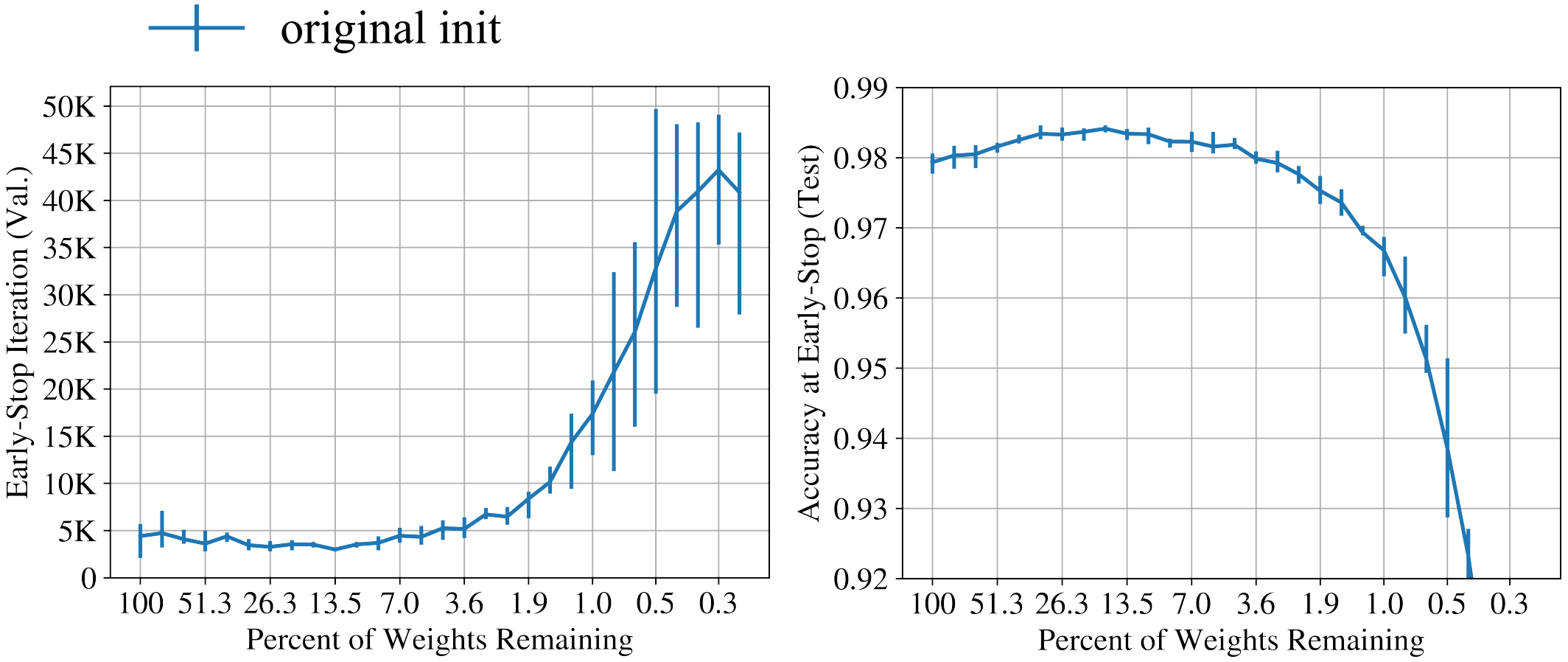

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



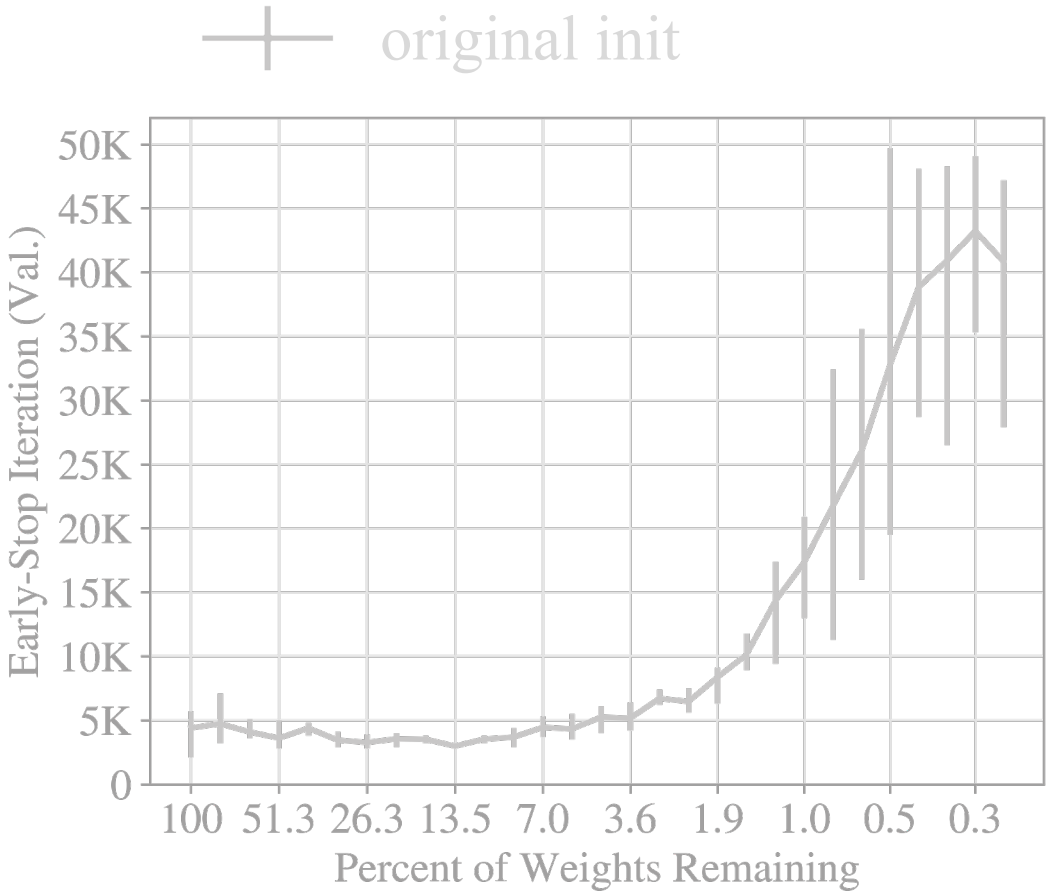

LeNet 300-100-10 for MNIST fully-connected 300K parameters

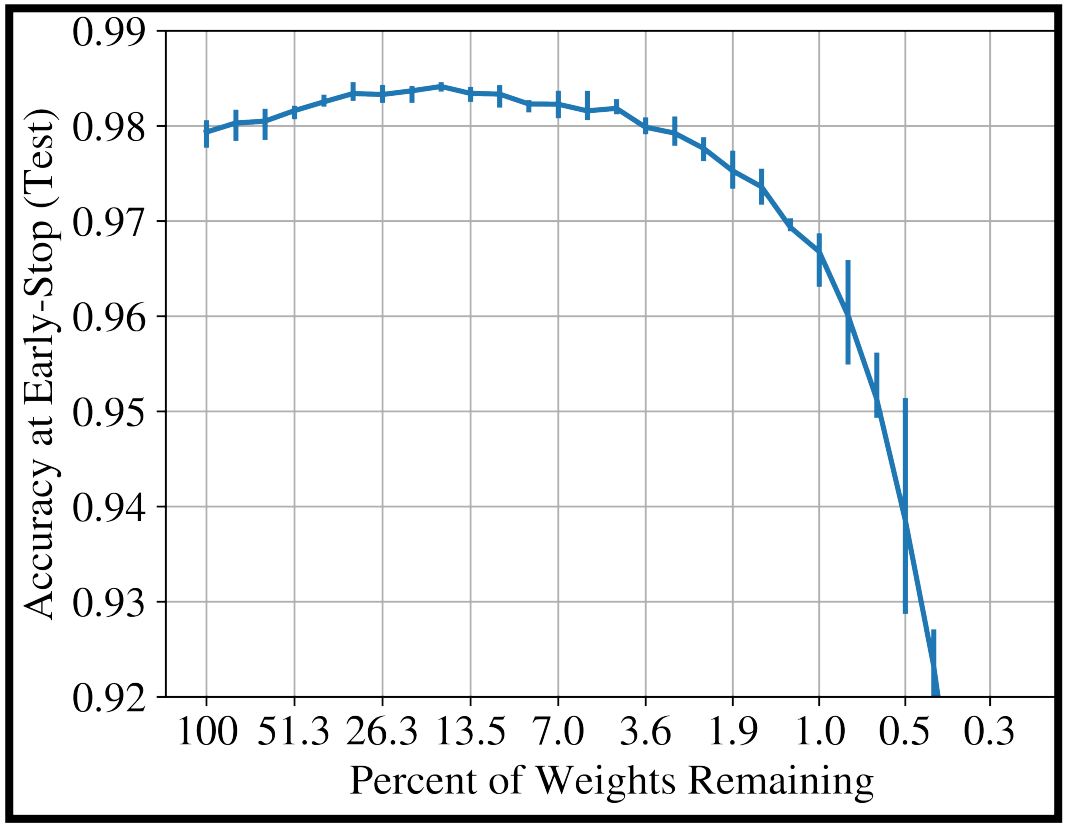
# Results




LeNet 300-100-10 for MNIST fully-connected 300K parameters

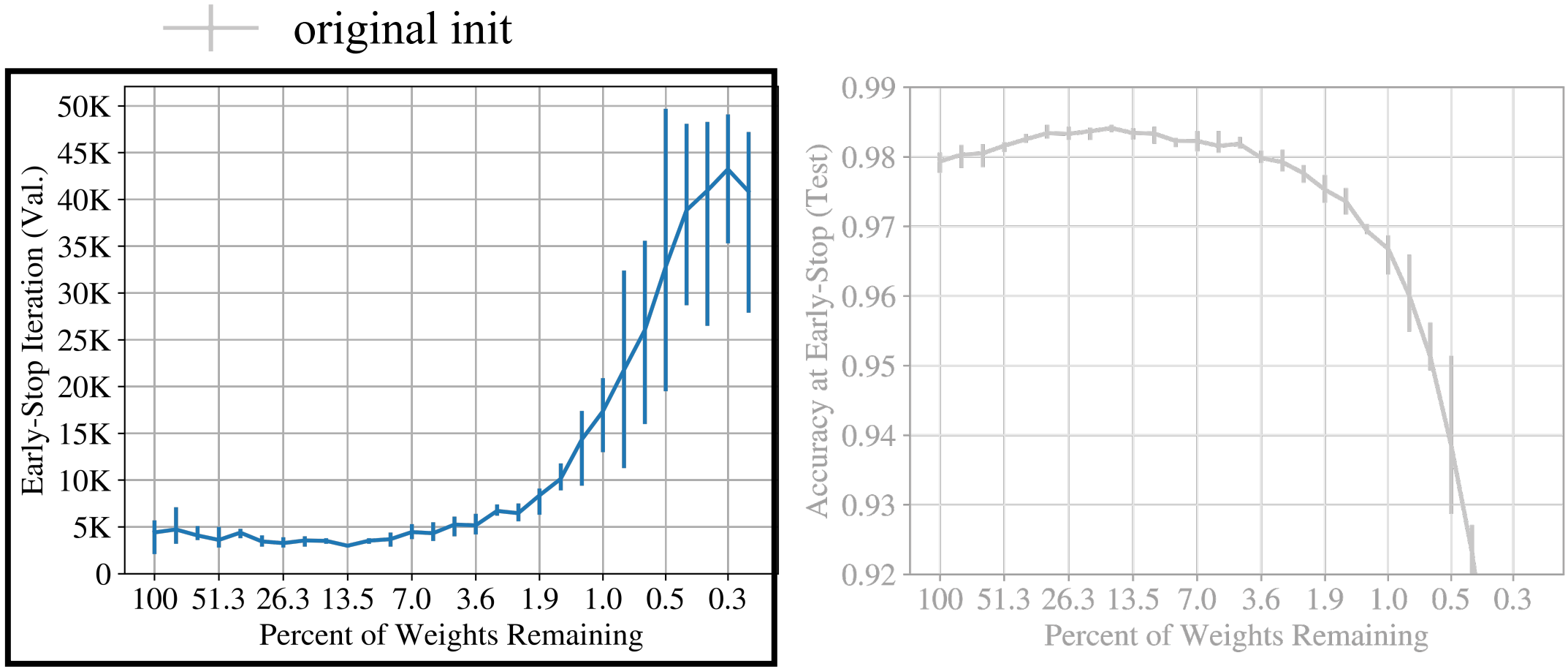
# Results



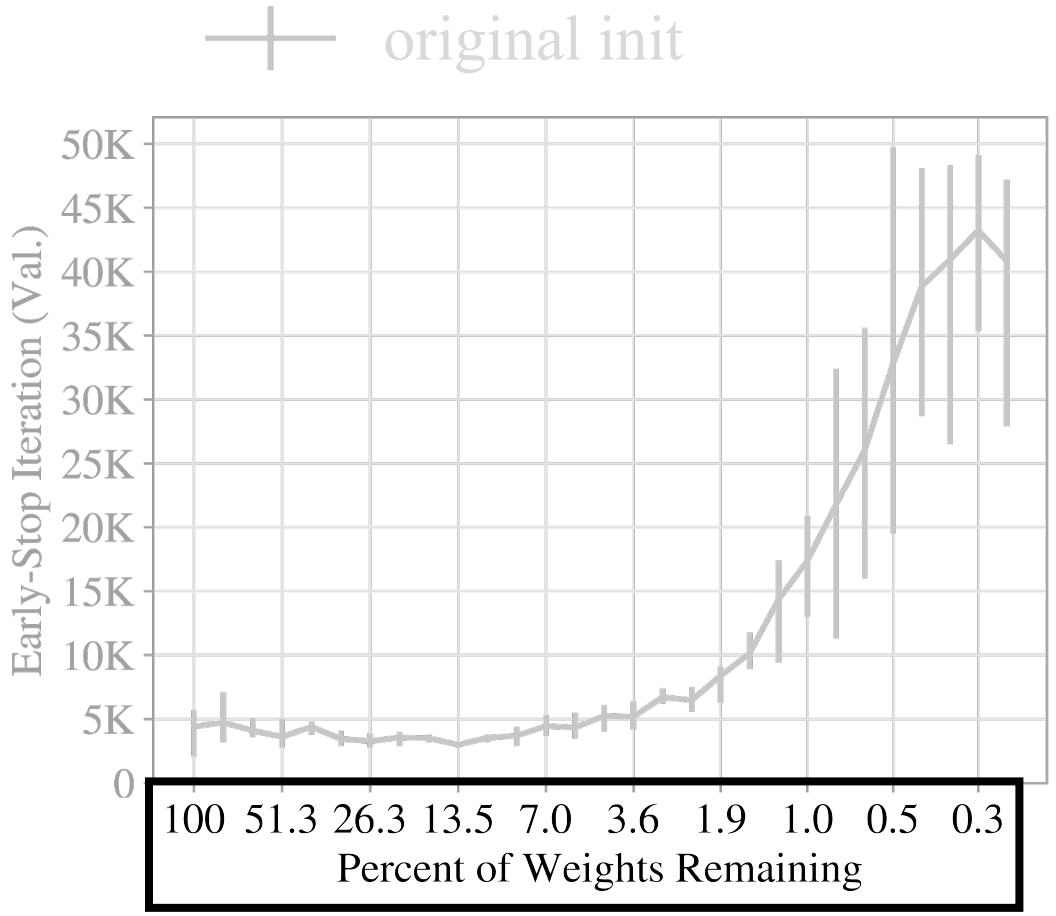

LeNet 300-100-10 for MNIST  fully-connected  300K parameters

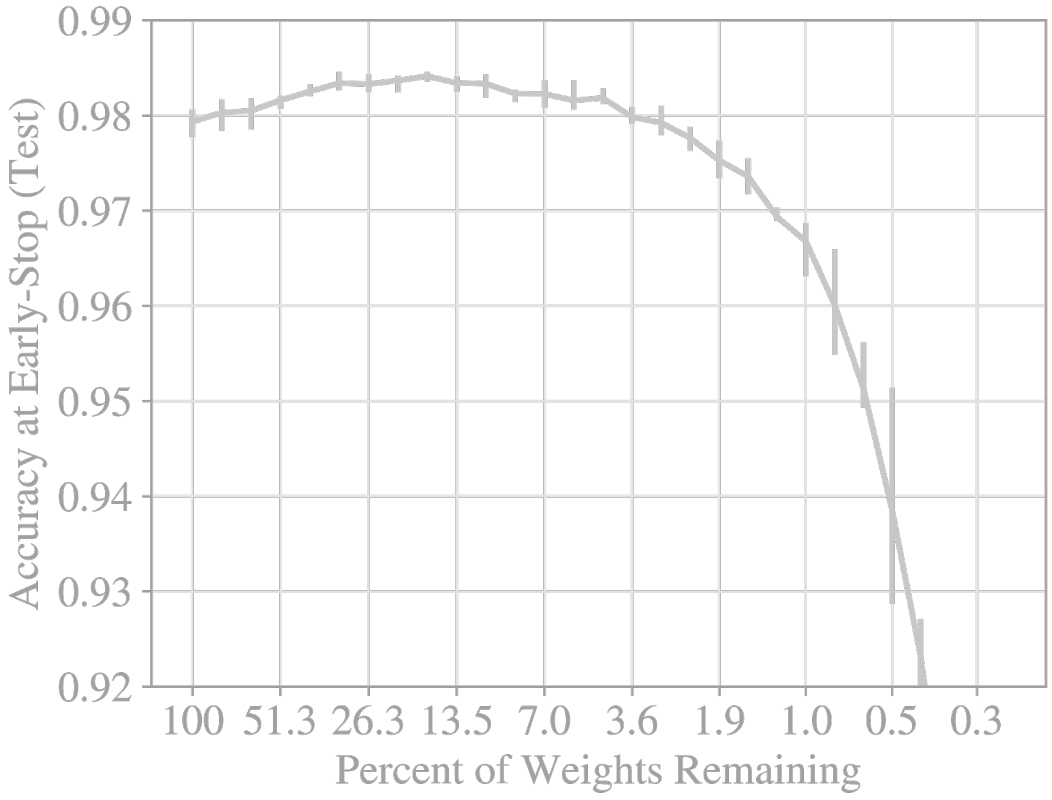
# Results




LeNet 300-100-10 for MNIST fully-connected 300K parameters

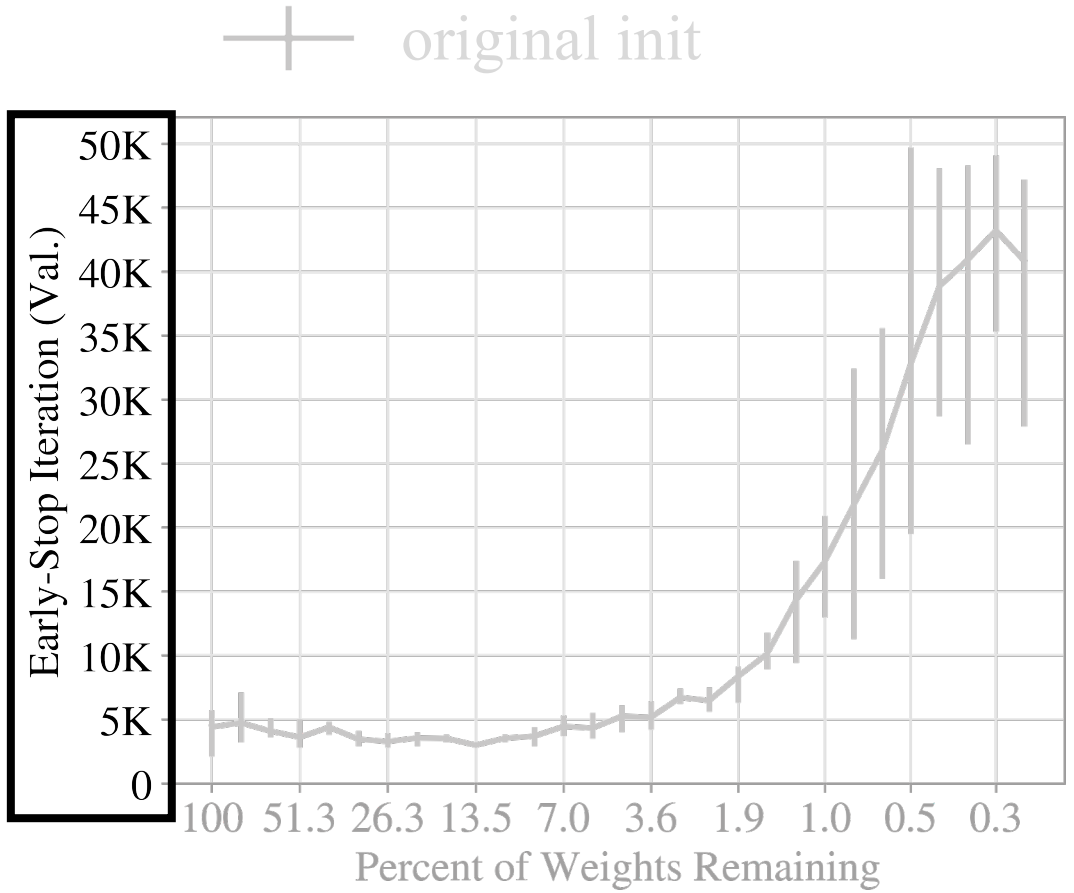
# Results

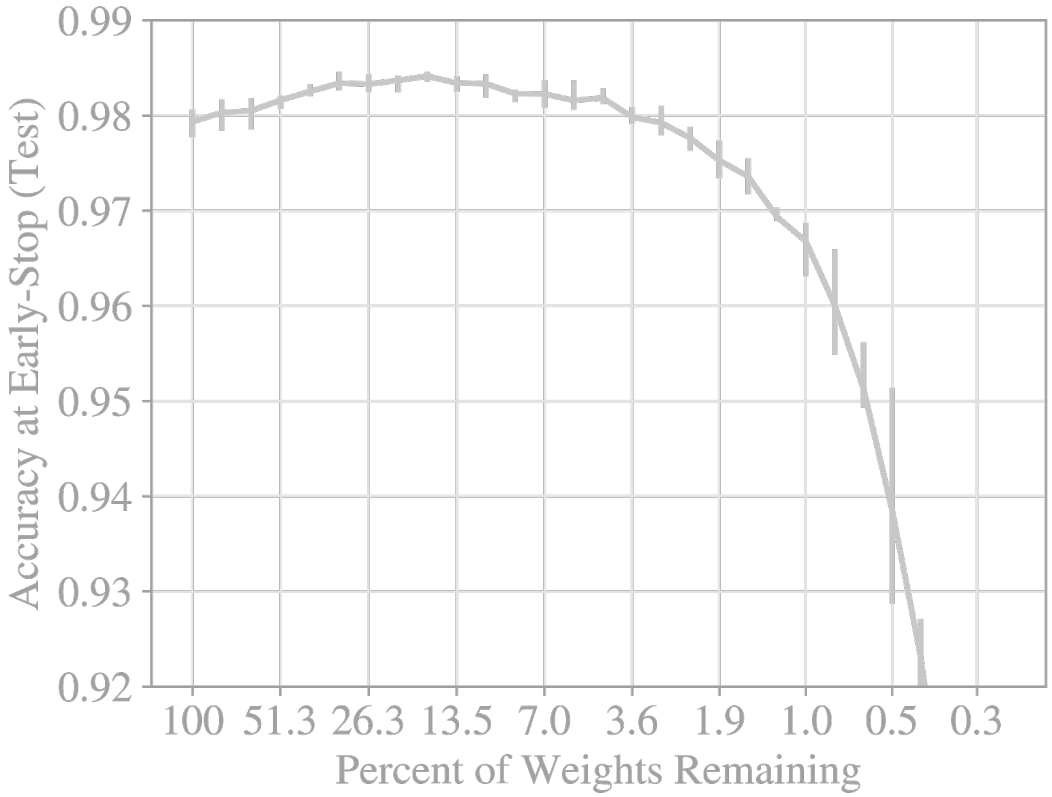



LeNet 300-100-10 for MNIST fully-connected 300K parameters

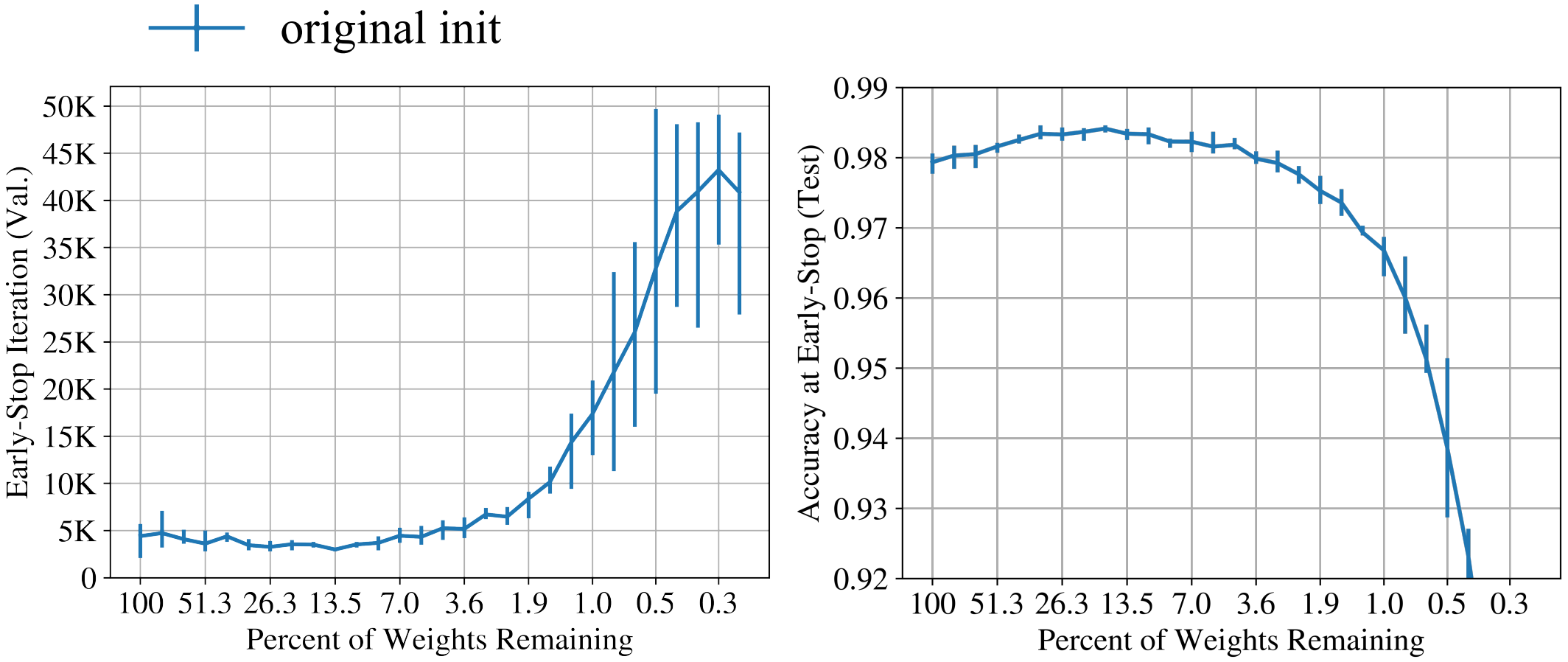
# Results



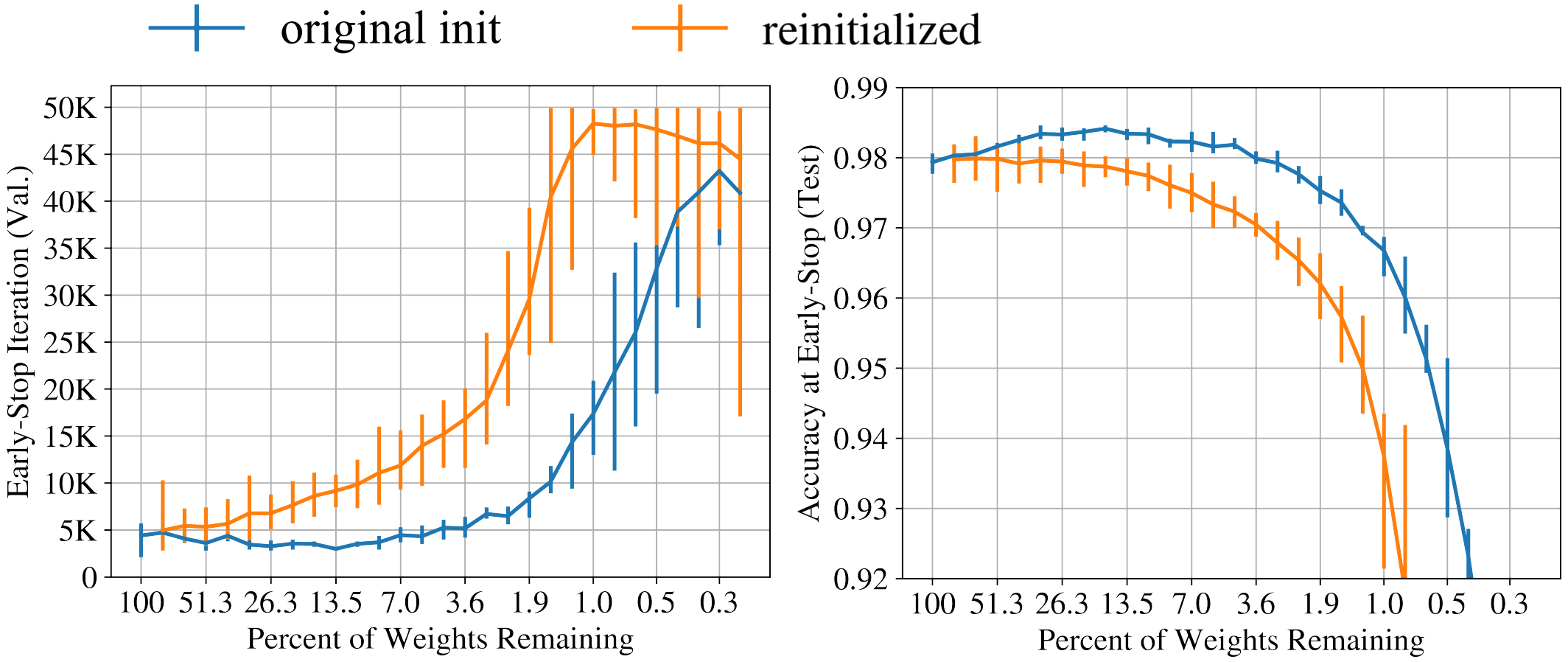

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



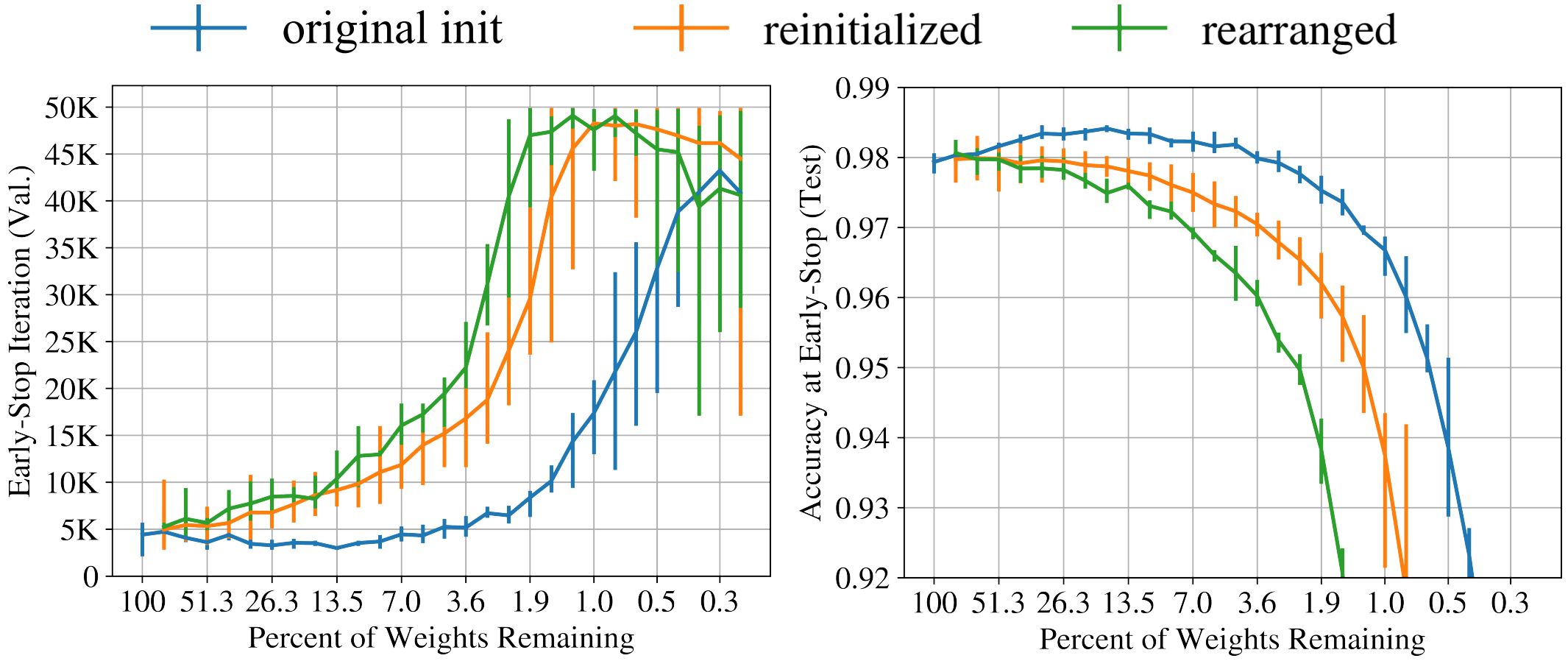

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



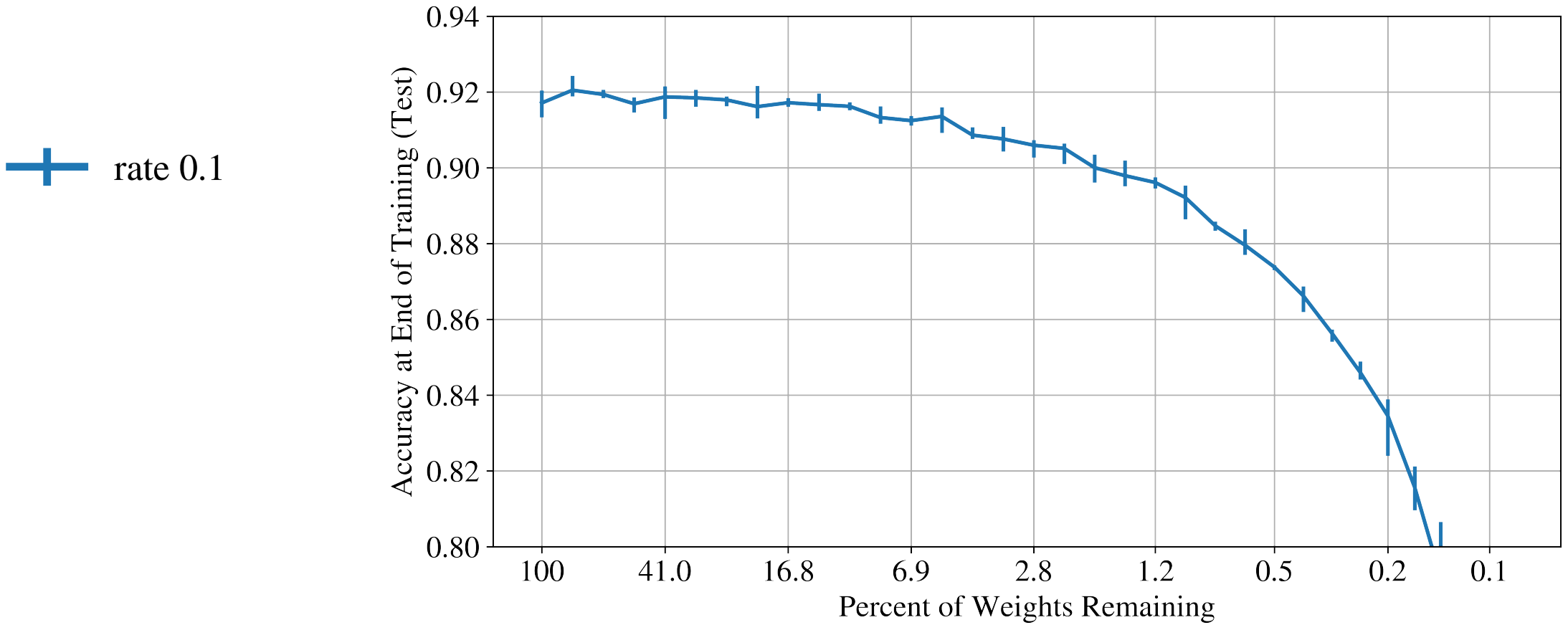

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



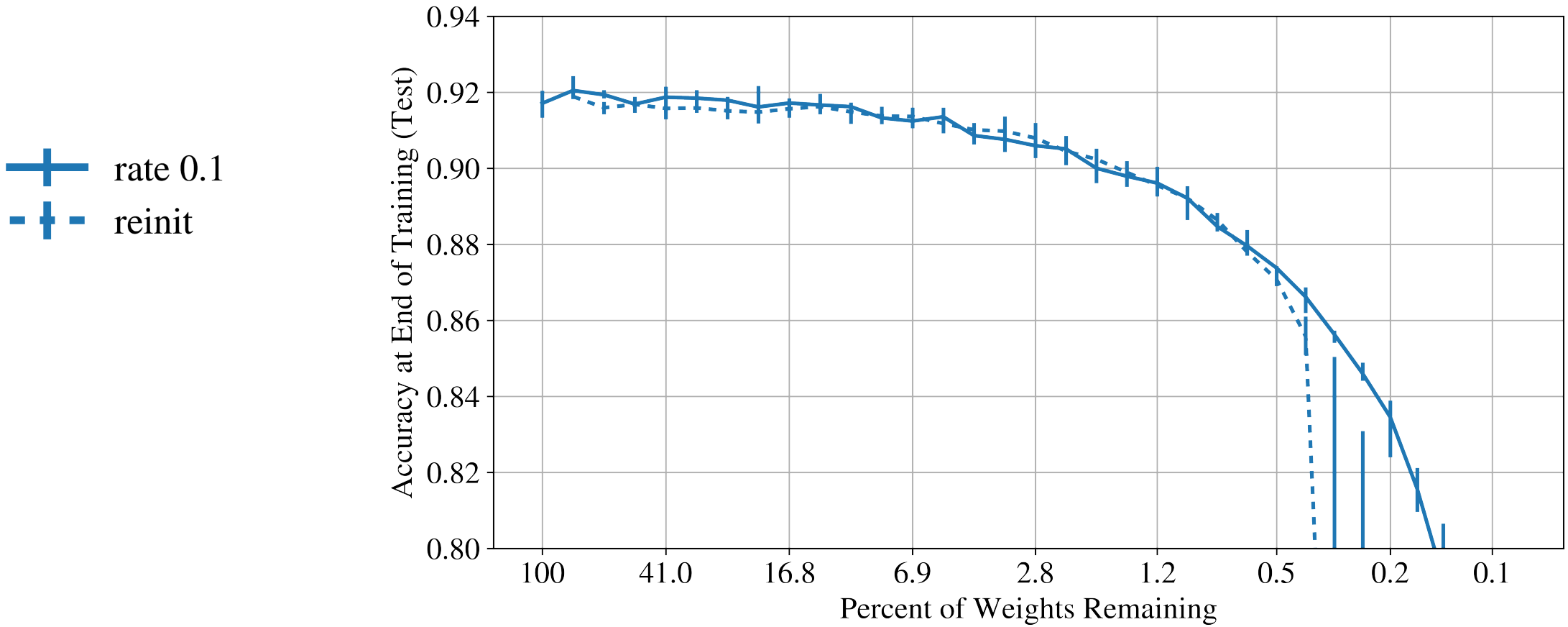

LeNet 300-100-10 for MNIST fully-connected 300K parameters

# Results



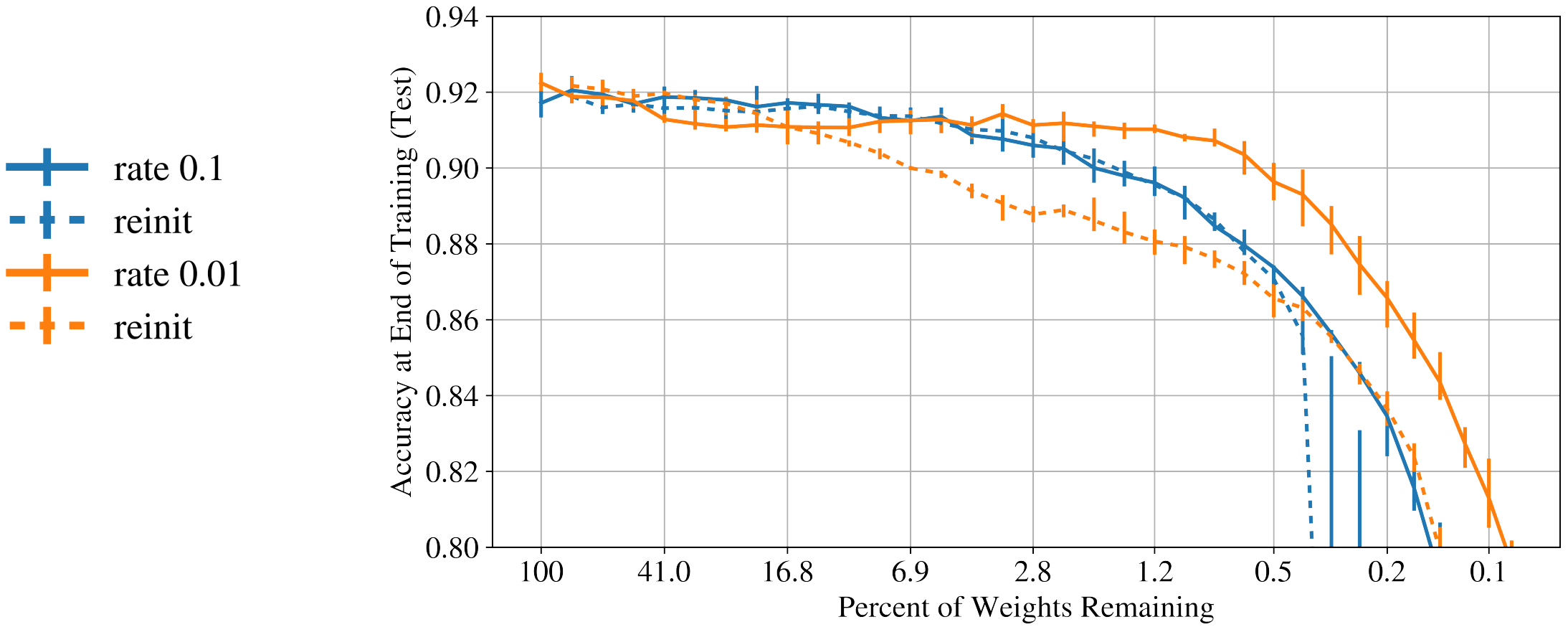

VGG-19 for CIFAR10 convolutional 20M parameters

# Results



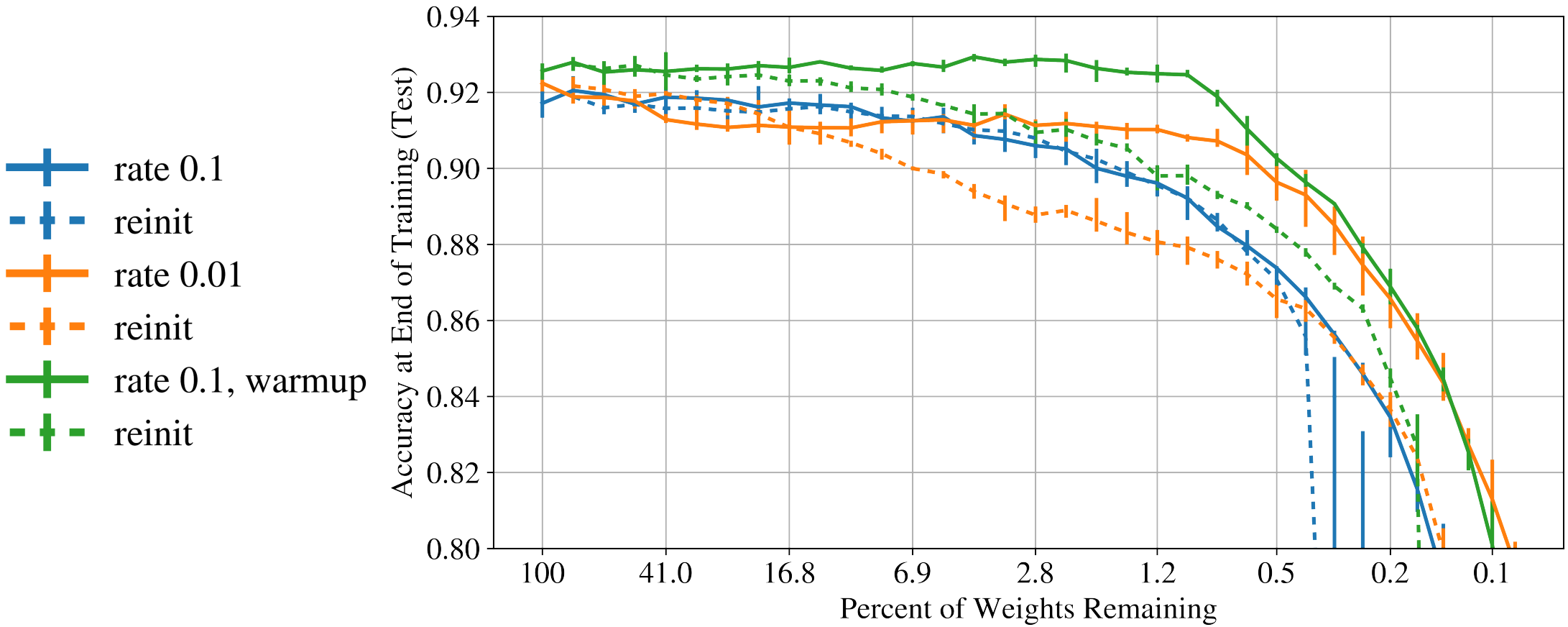

VGG-19 for CIFAR10 convolutional 20M parameters

# Results




VGG-19 for CIFAR10 convolutional 20M parameters

# Results



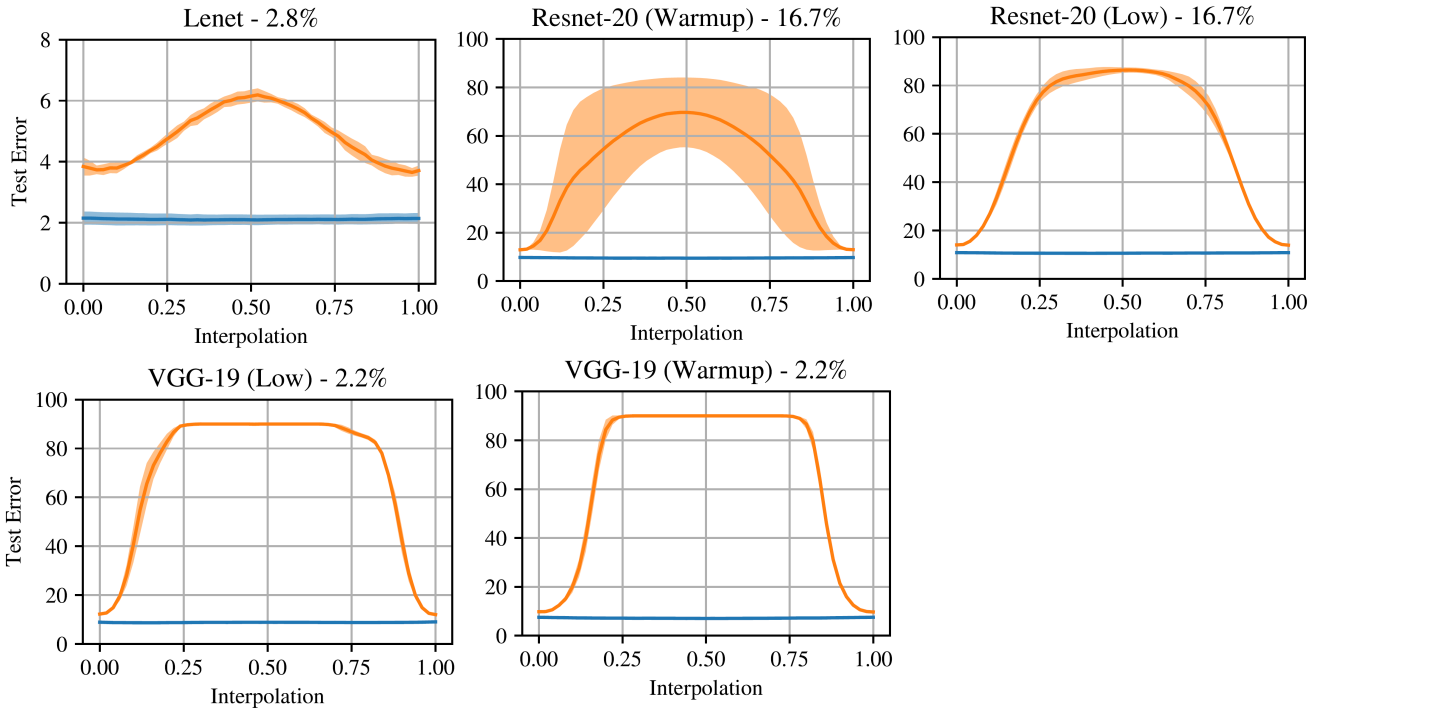

VGG-19 for CIFAR10 convolutional 20M parameters

# The Lottery Ticket Hypothesis

- The networks we typically train contain much smaller sparse, trainable subnetworks.
- We can find these subnetworks via pruning.
- The subnetworks have won the initialization lottery such that they can train in isolation.

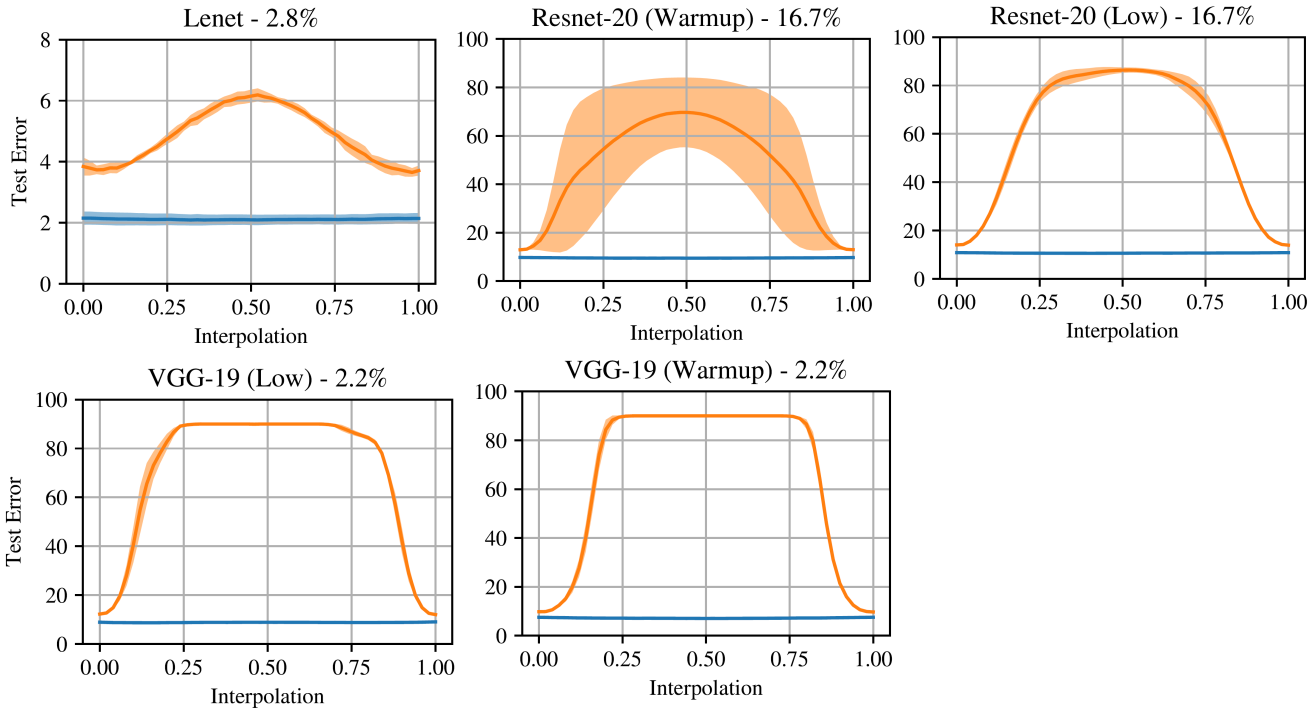
# Why? A First Start: Stability

## IMP Works

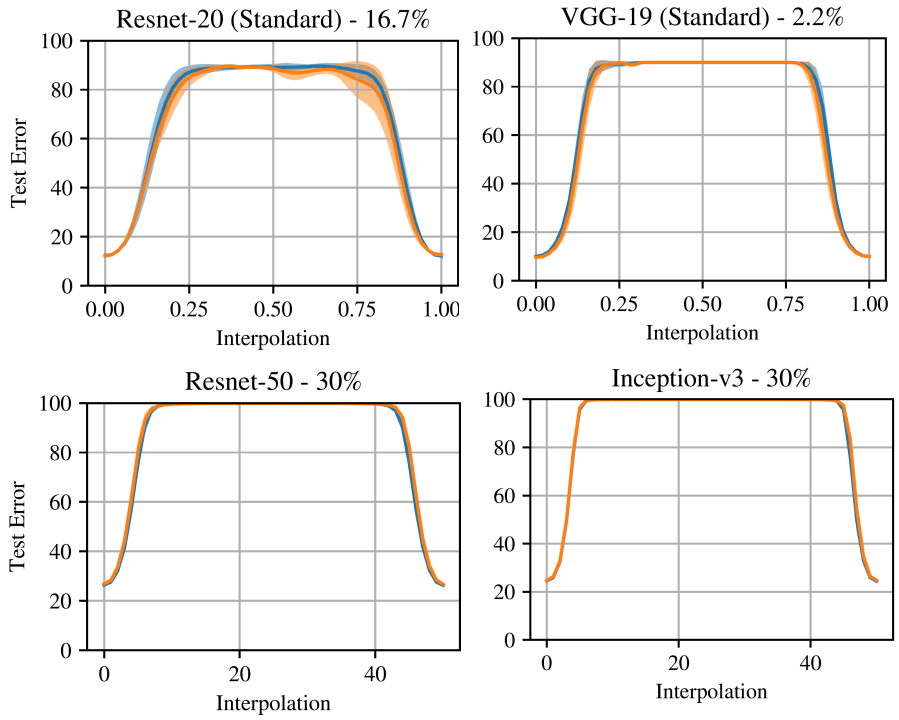


# Why? A First Start: Stability

## IMP Works




## IMP Doesn't Work




IMP Subnetwork

Random Subnetwork

# The Bottom Line

In networks we typically train, sparse, trainable subnetworks emerge very early in training.

COMPARING FINE-TUNING AND REWINDING IN NEURAL NETWORK PRUNING

**Anonymous authors**
Paper under double-blind review

We find that this procedure…makes it possible to prune network further…for a given target accuracy.

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

Systems Building Challenges:

- Performance
- Semantics (Real-valued, Differentiable, Probabilistic, Programs)
- Correctness

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

Systems Building Challenges:

- Performance
- **Semantics (Real-valued, Differentiable, Probabilistic Programs)**
- Correctness

# Semantics: Real-valued Computation

```
float x1 = ...;
float x2 = ... ;
if (x1 == x2) {
   ...
}
```

```
float x = ... ;
if (0 <= x)  {
   return x;
} else {
   return 0;
}
```

# Semantics : Real-valued Computation

```
real  x1 = ...;
real  x2 = ... ;
if (x1 == x2) {
   ...
}
```

```
real  x = ... ;
if (0 <= x)  {
   return x;
} else {
   return 0;
}
```

# Semantics : Real-valued Computation

```
real  x1 = ...;
real  x2 = ... ;
if (x1 == x2) {
   ...
}
```

Not computable!

```
real  x = ... ;
if (0 <= x)  {
   return x;
} else {
   return 0;
}
```

# Semantics : Real-valued Computation

```
real  x1 = ...;
real  x2 = ... ;
if (x1 == x2) {
   ...
}
```

Not computable!

```
real  x = ... ;
if (0 <= x)  {
   return x;
} else {
   return 0;
}
```

Not differentiable!

# Semantics : Real-valued Computation

```
real  x1 = ...;
real  x2 = ... ;
if (x1 == x2) {
   ...
}
```

Not computable!

```
real  x = ... ;
if (0 <= x)  {
   return x;
} else {
   return 0;
}
```

Not differentiable!

Sherman, Sciarappa, Chlipala, and Carbin. Computable decision making on the reals. LICS 18
Sherman, Michel, and Carbin. Sound solid modelling with arbitrary-precision real arithmetic. ICFP 2019 (Distinguished Paper Award)

# Engineering Approximate Computations

Case Study: Performance Modeling with Deep Learning

- Embracing approximation yields improved capability

Systems Building Challenges:

- Performance
- Semantics (Real-valued, Differentiable, Probabilistic, Programs)
- **Correctness**

# Correctness (Generalization Bounds)

How do I know this system works for unseen inputs?

$$\text{with probability } 1-\delta,\ \mathcal{L}(u) \leq B(\mathcal{L}(s), \delta)$$

$\delta$ – confidence parameter
$\mathcal{L}(\cdot)$ – loss or error function
$u$ – a sample of unseen data
$B(\cdot,\cdot)$ – bound
$s$ – a sample of seen data (a test set)

# Takeaway

- Landscape of software systems is changing. Deep learning and statistical machine learning will permeate software.
- New systems are approximate – results known only up to a given precision or given probability.
- Opportunity to leverage specification for resilience and performance, but challenges to delivering a sound programming methodology