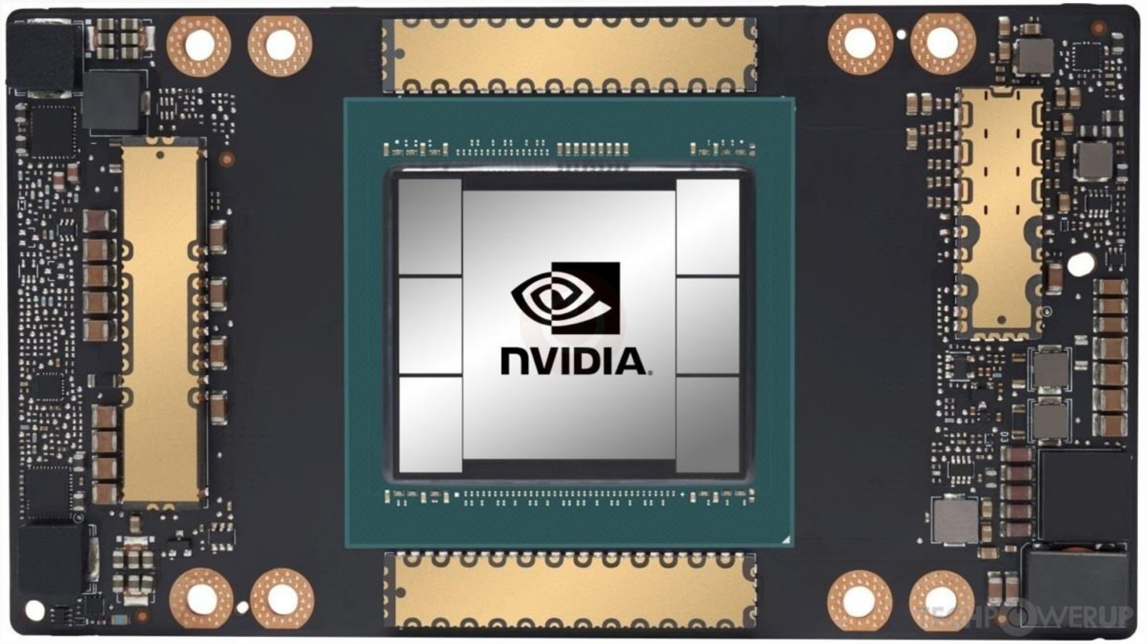
System

NVIDIA A100 SXM4 – 40 GB

- Bandwidth: 1.56 TB/s
- FP32 Peak Performance 19.49 TFLOPs
  (measured ca. 15 TFLOPs)
- Arithmetic Intensity: 12.5 FLOP/Byte
  (measured ca. 9.5 FLOP/Byte)



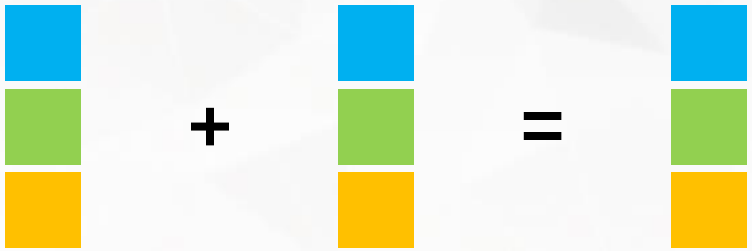
- 108 Streaming Multiprocessors
- 6912 Cuda Cores
- 432 Tensor Cores

- TP32 Peak Performance 155.92 TFLOPs
- Arithmetic Intensity: 99.9

https://www.techpowerup.com/gpu-specs/a100-sxm4-40-gb.c3506

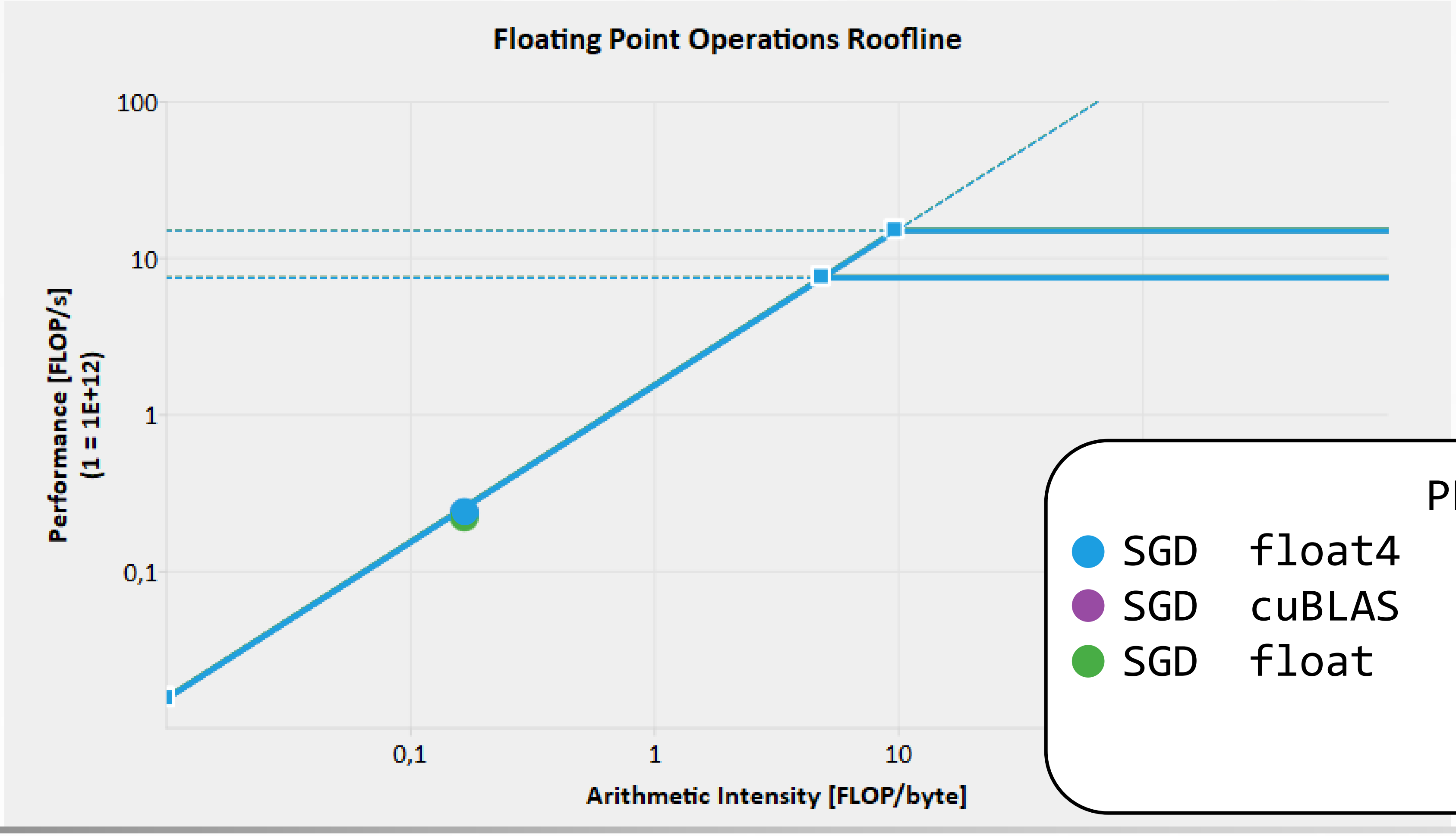
Element-Wise Operations

e.g. Optimizers or Activations



Stochastic Gradient Descent (SGD) – Parameter Update

```
weights[i] = weights[i] - learning_rate * gradient[i];
```

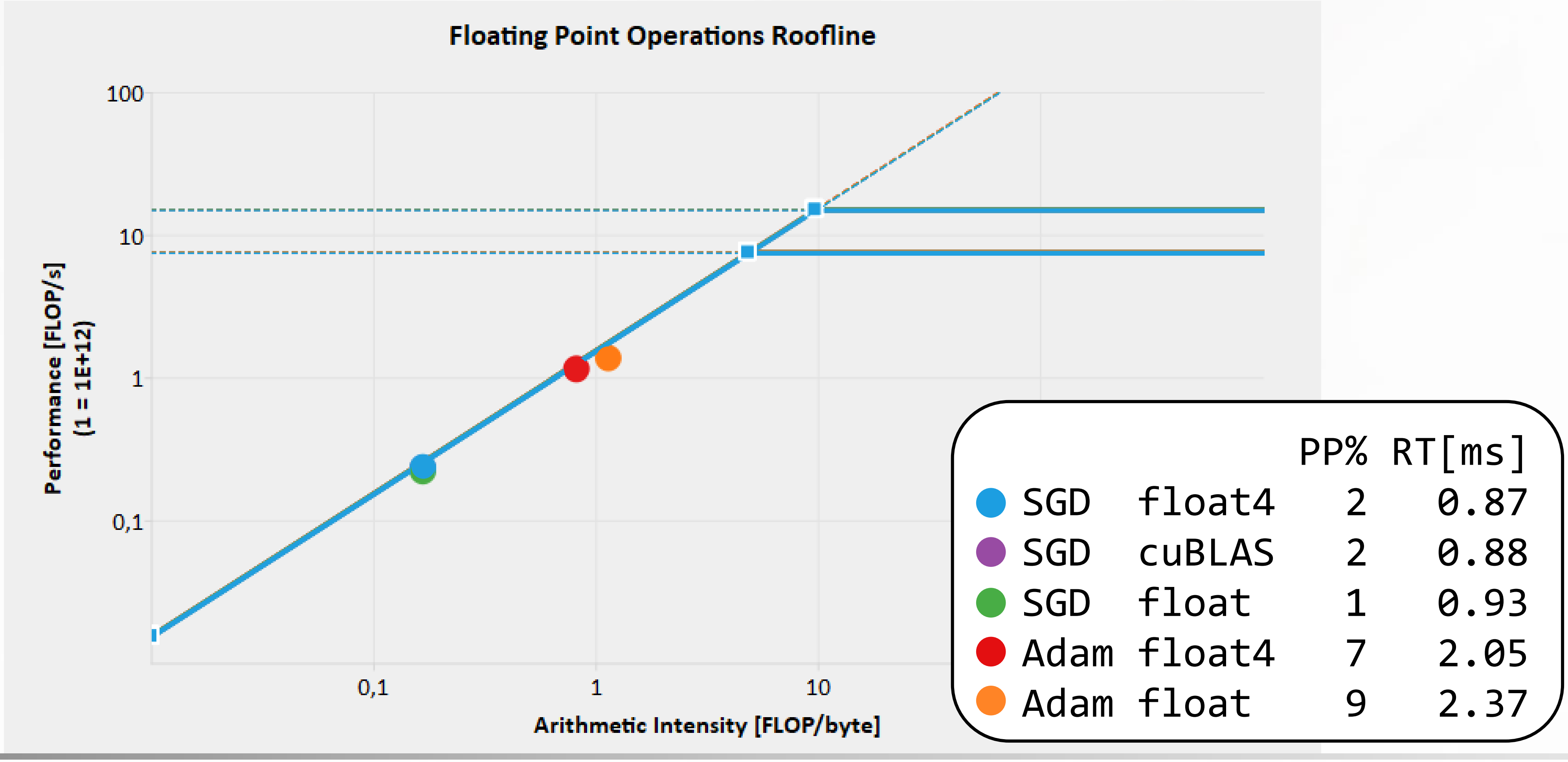


			PP%	RT[ms]
●	SGD	float4	2	0.87
●	SGD	cuBLAS	2	0.88
●	SGD	float	1	0.93

PP% = Peak Performance Percentage, RT[ms] = Runtime per Iteration in milliseconds

Adam – Parameter Update

```
v[i] = mu * v[i] + (1 - mu) * gradient[i];
r[i] = rho * r[i] + (1 - rho) * gradient[i] * gradient[i];
weights[i] = weights[i] - learning_rate * v[i] / (sqrt(r[i]) + epsilon);
```

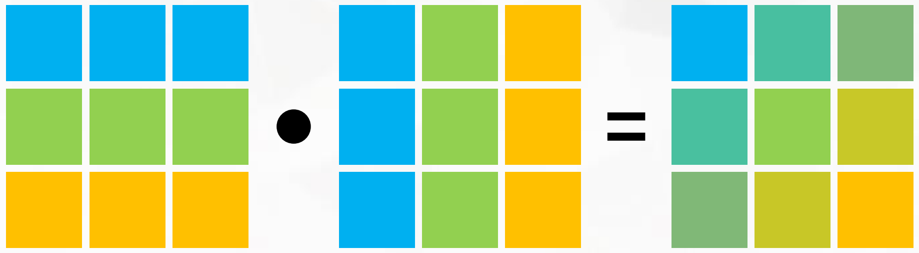


			PP%	RT[ms]
🔵	SGD	float4	2	0.87
🟣	SGD	cuBLAS	2	0.88
🟢	SGD	float	1	0.93
🔴	Adam	float4	7	2.05
🟠	Adam	float	9	2.37

PP% = Peak Performance Percentage, RT[ms] = Runtime per Iteration in milliseconds

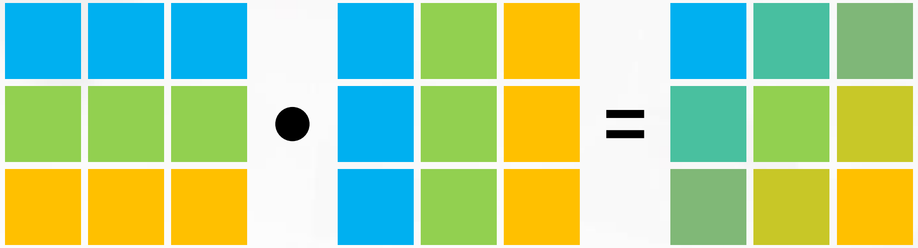
Matrix-Matrix Multiplications

e.g. Dense or Embedding Layers



Matrix – Matrix Multiplication

$C = A \cdot B \qquad A, B, C \in \mathrm{Mat}(N \times N)$



```
float val = 0;
for(int i = 0; i < col_a; ++i)
    val += A[r * col_a + i]
         * B[i * col_b + c];
C[r * col_c + c] = val;
```

Theoretic Arithmetic Intensity:

- Loads [4Byte]: $2N^2$
- Stores [4Byte]: N^2
- Operations [FLOP]: $2N^3$

➔ $\frac{1}{6}N$ FLOP/Byte

➔ Compute Bound

Arithmetic Intensity:

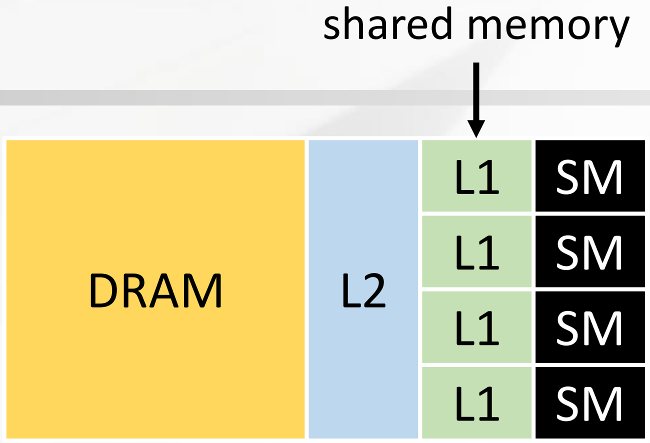
- Loads [4Byte]: $2N^3$
- Stores [4Byte]: N^2
- Operations [FLOP]: $2N^3$

➔ $N/(4N+2)$ FLOP/Byte

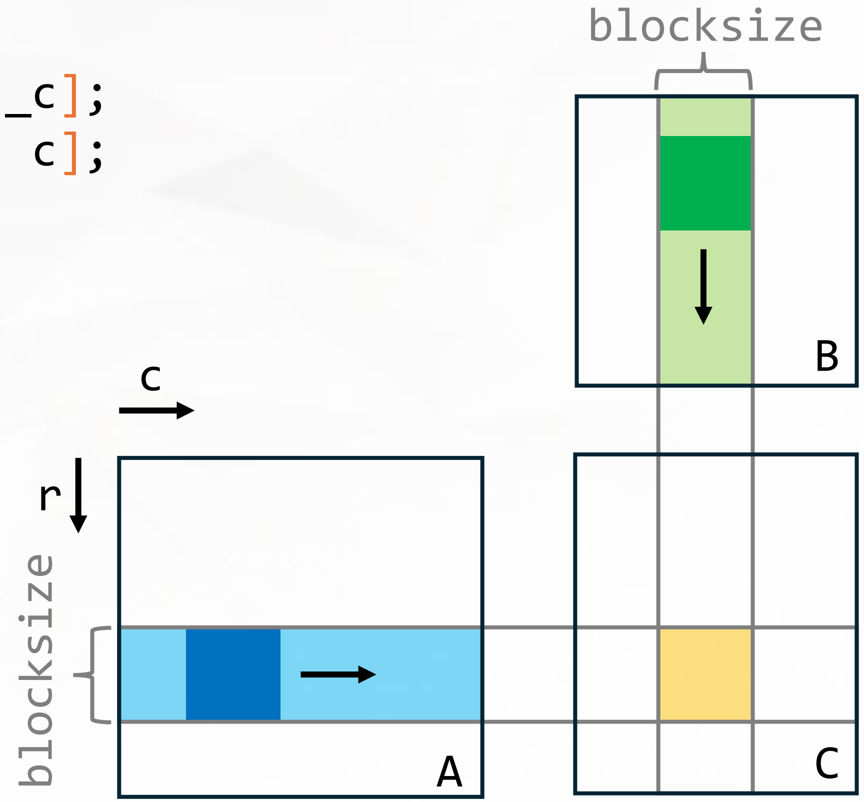
➔ Worst Case: Memory Bound

(assuming no compiler optimization)

Matrix – Matrix Multiplication



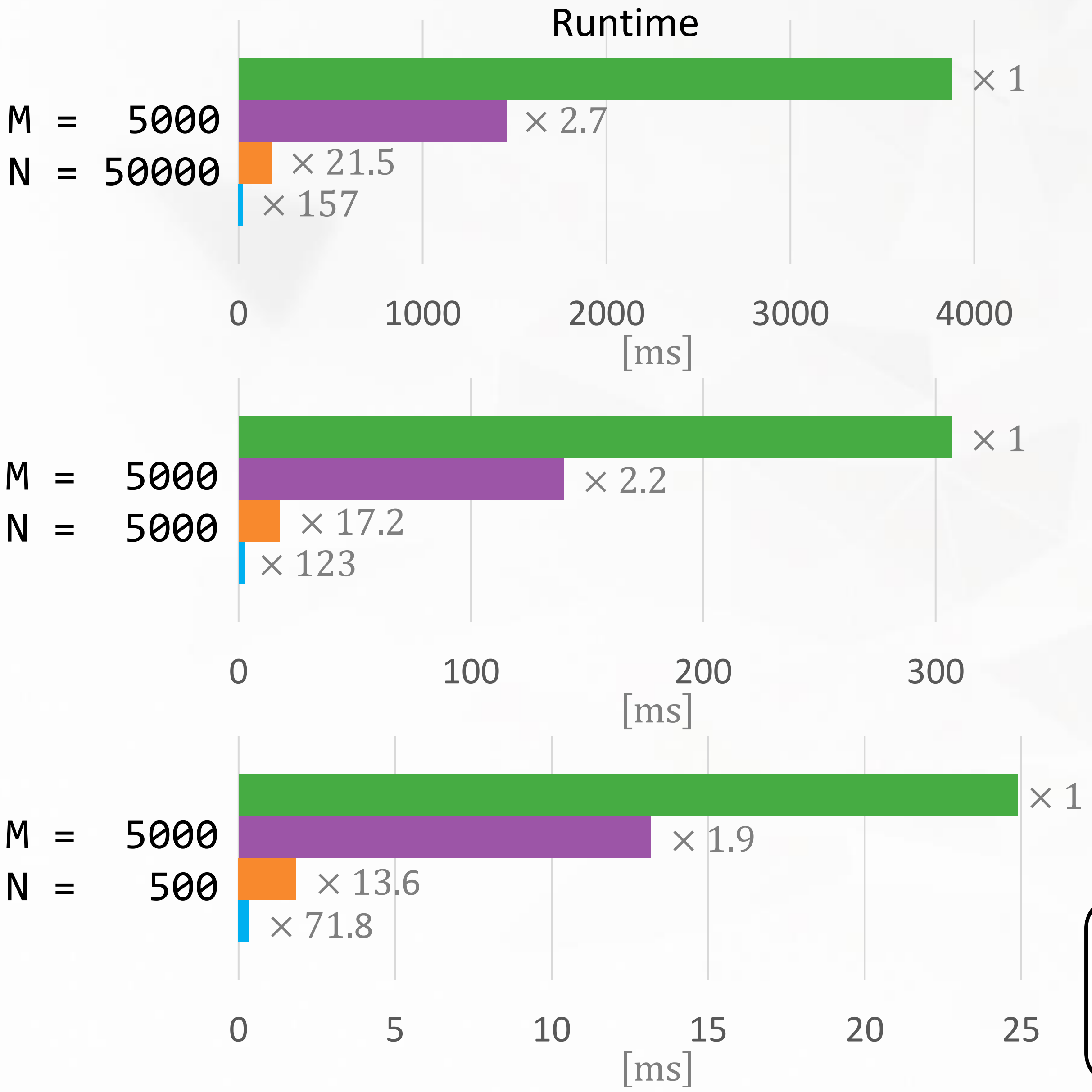
```
float val = 0;
for(int shift = 0; shift < col_a; shift += blocksize){
    int _c = c_block + shift;
    int _r = r_block + shift;
    shared_A[r_block * blocksize + c_block] = A[ r * col_a + _c];
    shared_B[r_block * blocksize + c_block] = B[_r * col_b +  c];
    __syncthreads();

    for(int i = 0; i < blocksize; ++i)
        val += shared_A[r_block * blocksize + i       ]
             * shared_B[i       * blocksize + c_block];
    __syncthreads();
}
C[r * col_c + c] = val;
```

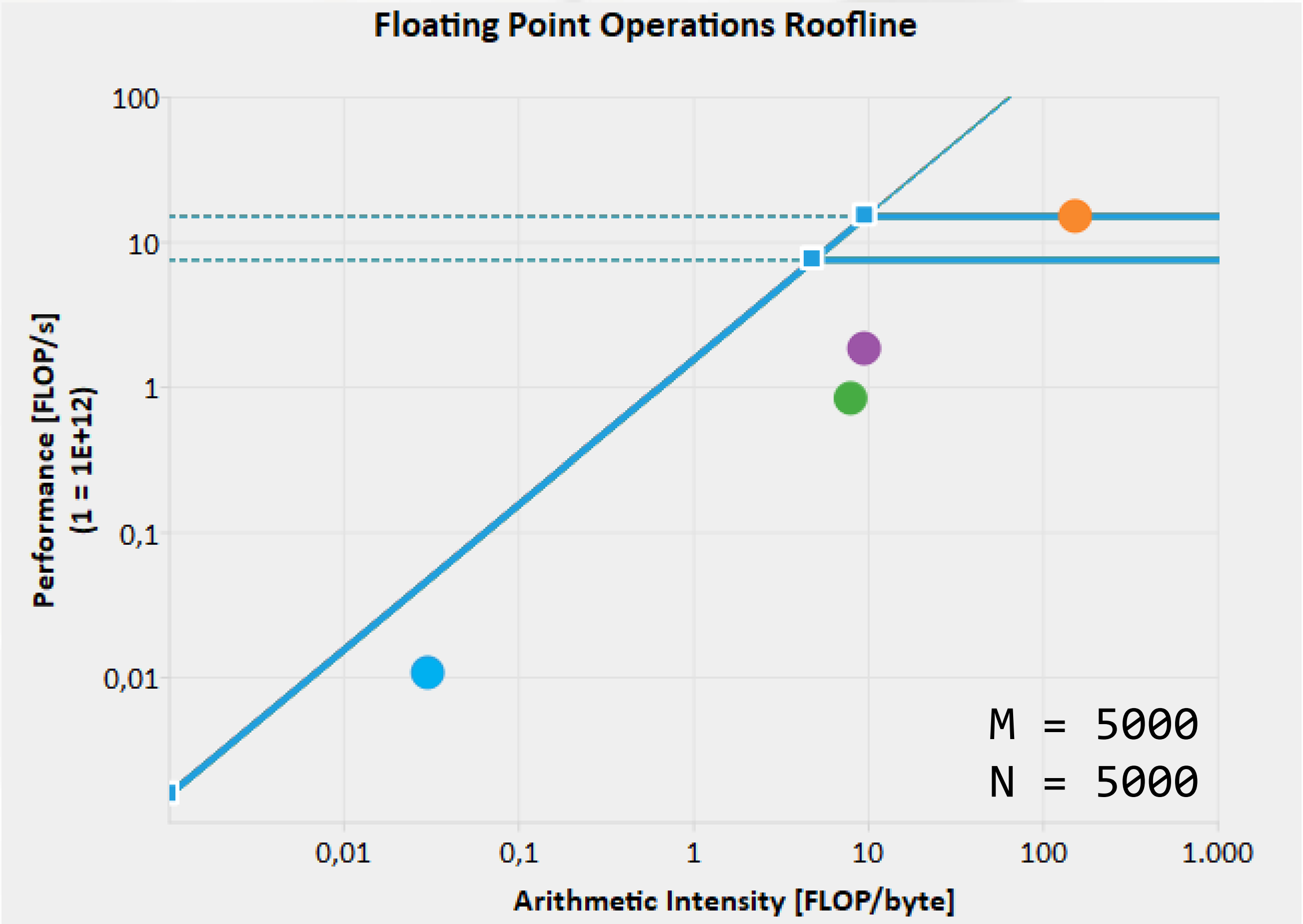


https://siboehm.com/articles/22/CUDA-MMM

Matrix – Matrix Multiplication

$A \in \mathrm{Mat}(M \times N), B \in \mathrm{Mat}(N \times M)$



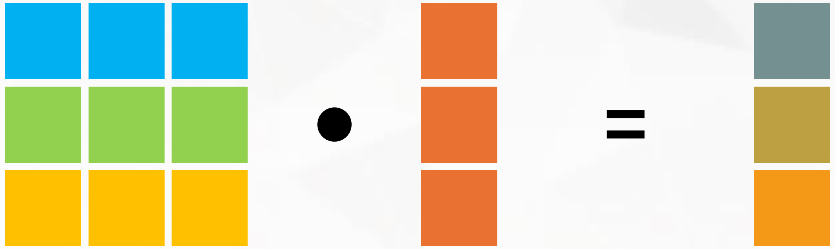


	PP%		PP%
● simple	5	● cuBLAS (Cuda Cores)	97
● shared	12	● cuBLAS (Tensor Cores)*	0

*PP% = Peak Performance Percentage, *measured Roofline Model only considers Cuda cores*

Matrix-Vector Operations

e.g. average along rows



Softmax – Converting Raw Outputs into Probabilities

$$f(x_j) = \frac{\exp(x_j - x_{\max})}{\sum_k \exp(x_k - x_{\max})}$$

$$\forall x \in \text{Samples}$$

$M \coloneqq$ #Samples
$N \coloneqq \dim(x)$

$$x_{\max} \xrightarrow[\#\text{Threads: } M]{}$$ Matrix-Vector Operation *„reduction“*

$$y_j = \exp(x_j - x_{\max}) \xrightarrow[\#\text{Threads: } M \cdot N]{}$$ element-wise operation

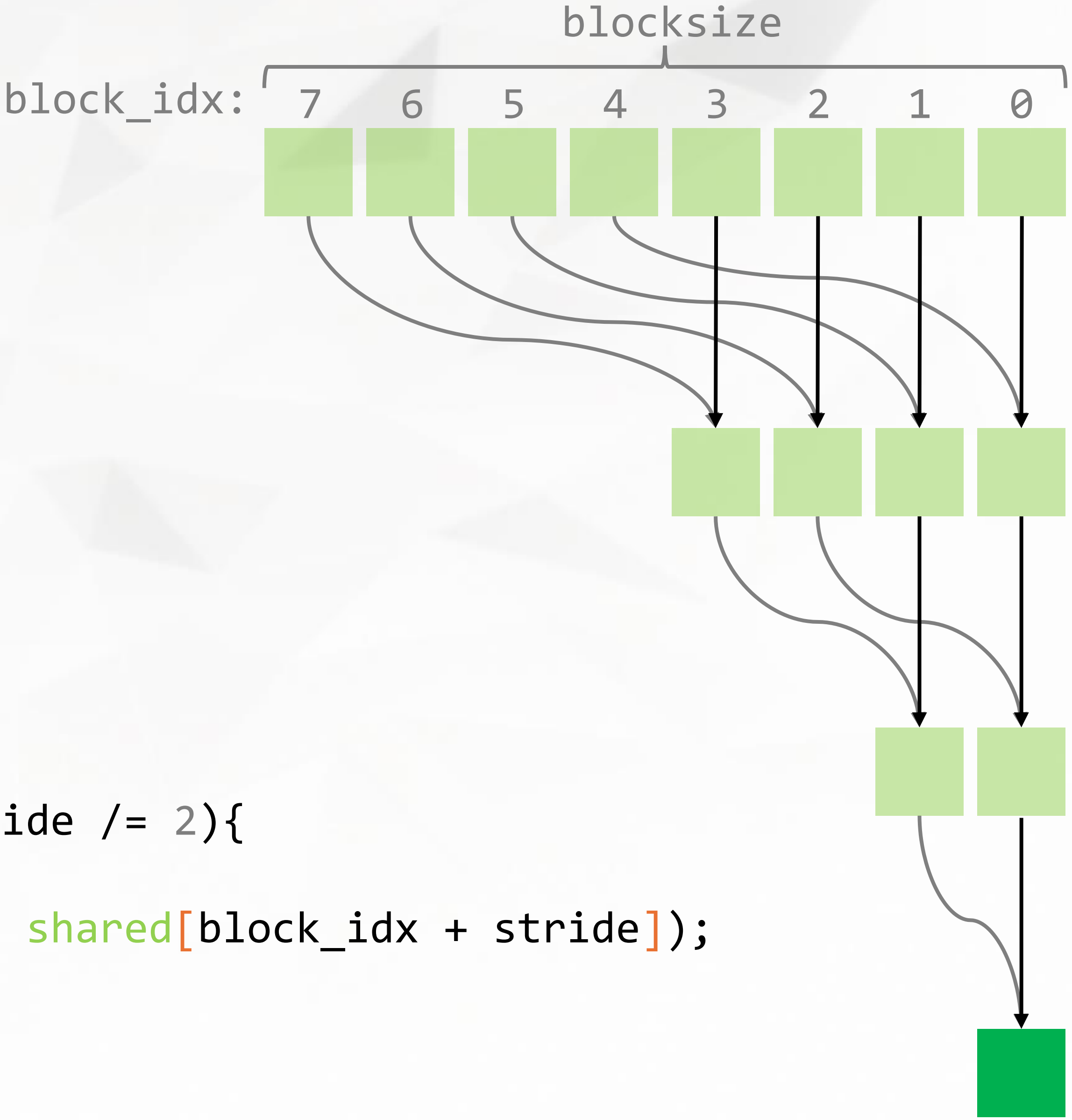
$$z = \sum_k y_k \xrightarrow[\#\text{Threads: } M]{}$$ Matrix-Vector Multiplication

$$f(x_j) = \frac{\mathrm{y_j}}{z} \xrightarrow[\#\text{Threads: } M \cdot N]{}$$ element-wise operation

Softmax – Max Reduction

```
float _max = -FLT_MAX;
for(int i = 0; i < col_A; ++i)
    _max = fmax(A[r * col_A + i], _max);
max[r] = _max;
```

```
int block_idx = r_block * blocksize + c_block;

float _max = -FLT_MAX;
for(int i = c_block; i < col_A; i += blocksize)
    _max = fmax(A[r * col_A + i], _max);
shared[block_idx] = _max;

__syncthreads();

for(int stride = blocksize / 2; stride > 0; stride /= 2){
    if(c_block >= stride) continue;
    shared[block_idx] = fmax(shared[block_idx], shared[block_idx + stride]);
    __syncthreads();
}
if (c_block == 0) max[r] = shared[r_block];
```

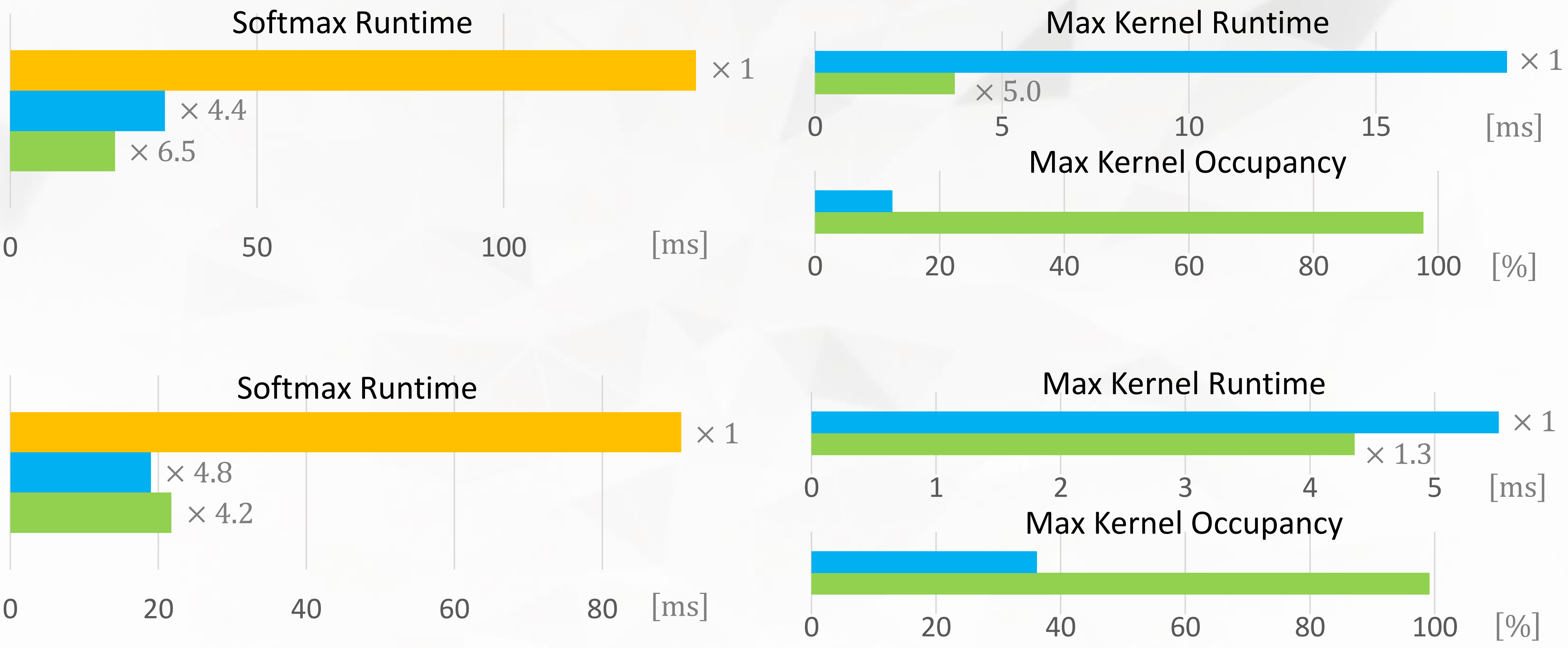


Softmax – Measurement

$A \in \mathrm{Mat}(M \times N)$

```
M =  10000
N = 100000
```

```
M = 100000
N =  10000
```



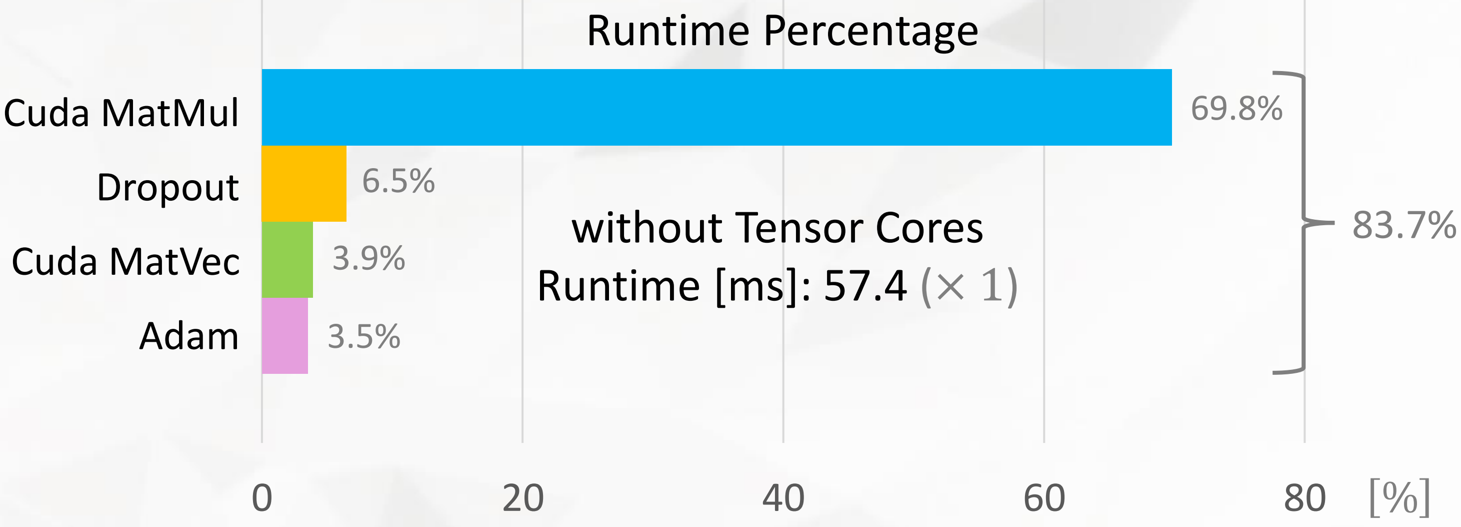
● `single kernel softmax`
● `split kernel softmax – simple max`
● `split kernel softmax – reduction max`

Transformer

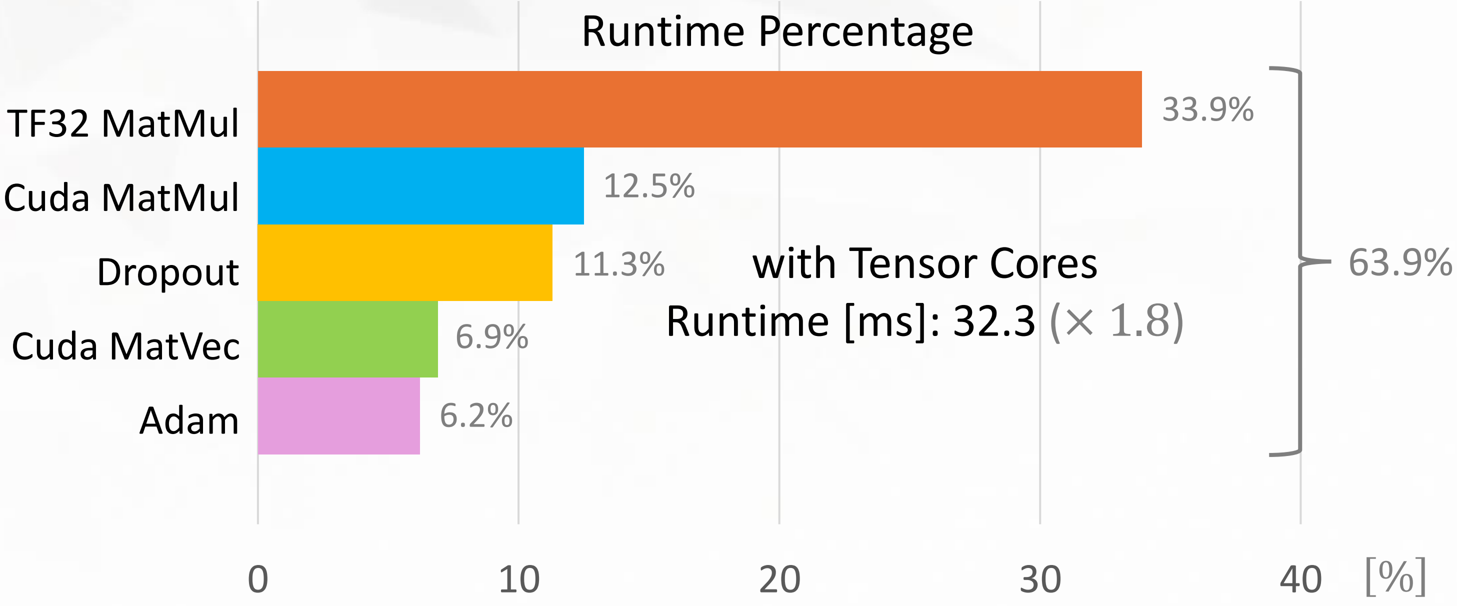
Transformer Benchmark

Default Configuration:

- Batch Size: 100
- Sequence Length: 13
- Encoder Layers: 6
- Decoder Layers: 6
- Number of Attention Heads: 8
- Number of Embeddings: 6880
- Embedding Dimension Length: 512
- Hidden Dimension Length: 2048
- Iterations: 200
- Warmup Steps: 50
- Tensor Cores activated

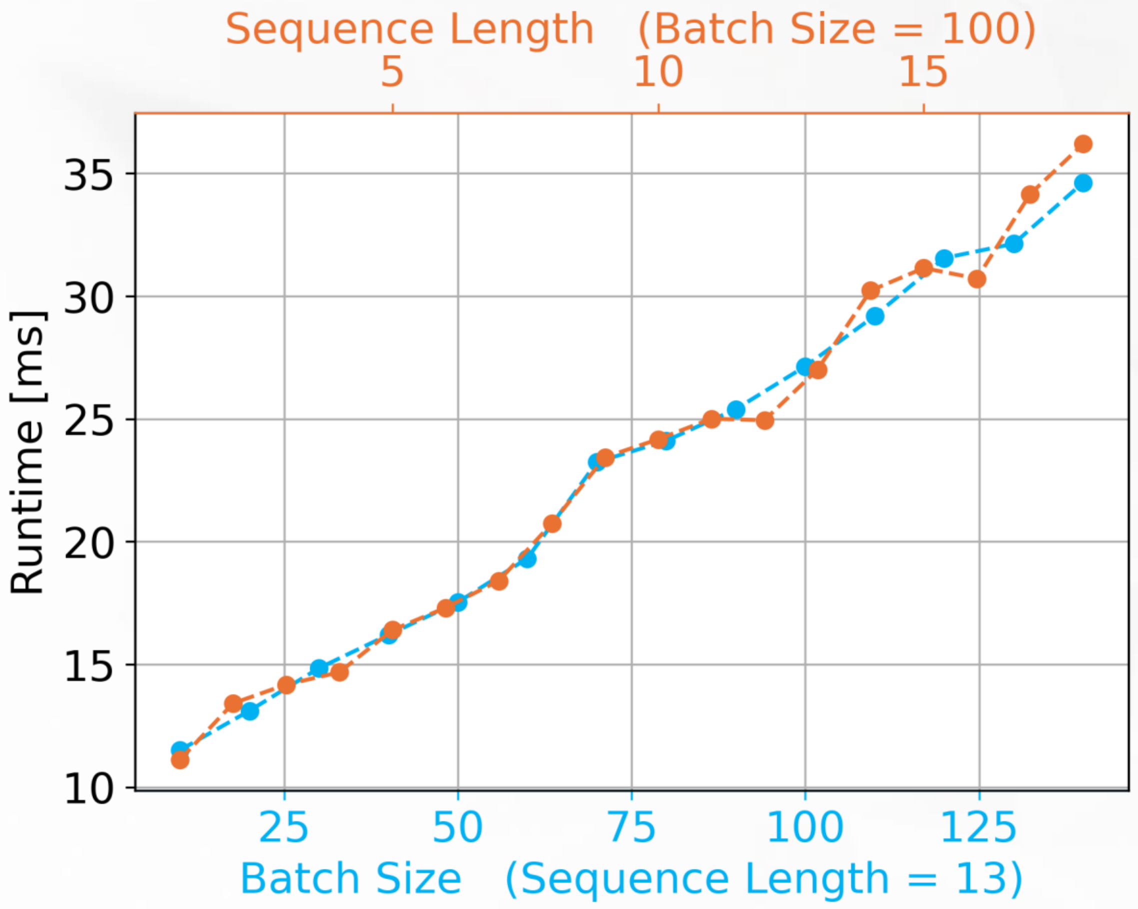


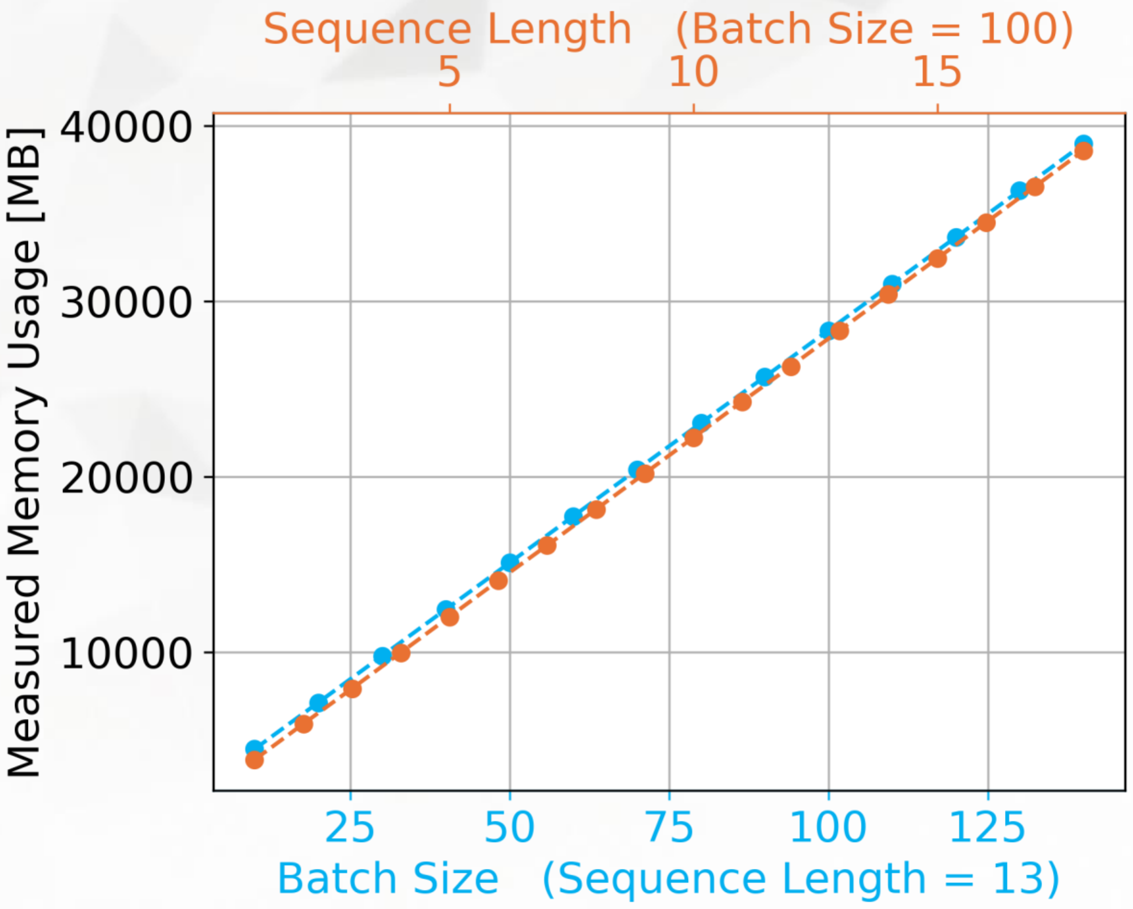
without Tensor Cores
Runtime [ms]: 57.4 (× 1)



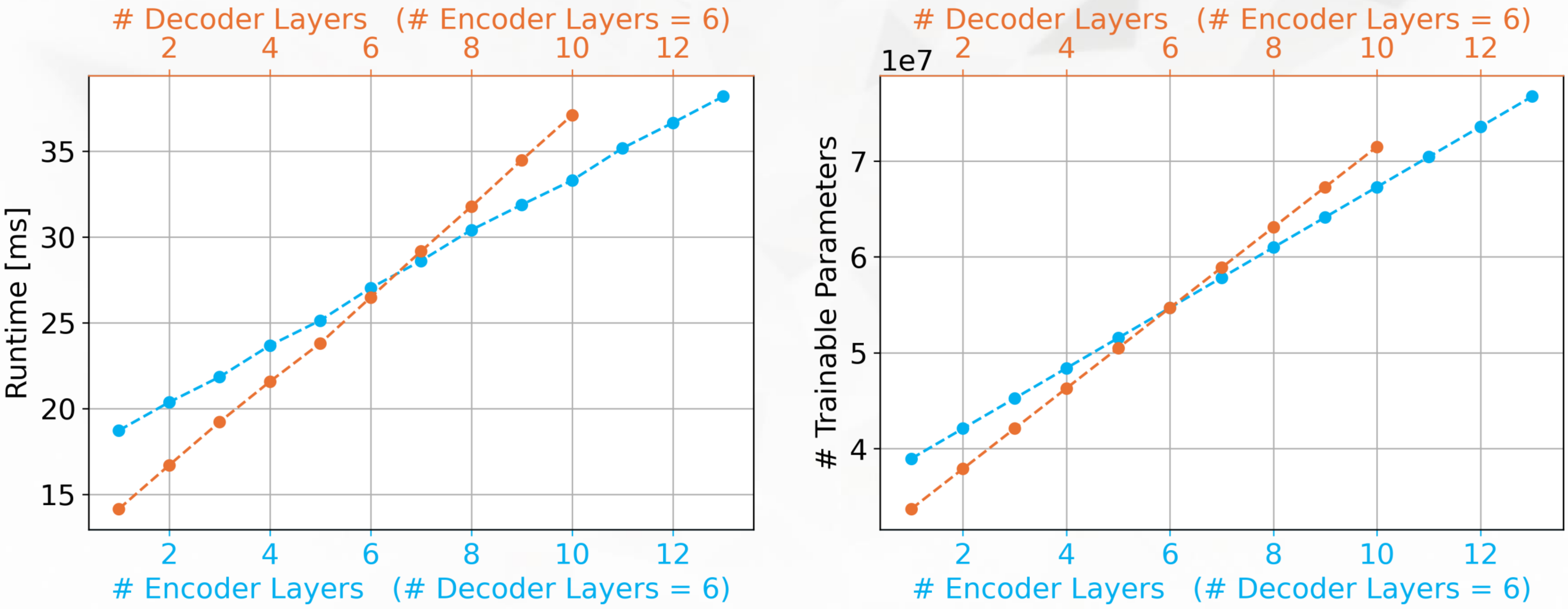
with Tensor Cores
Runtime [ms]: 32.3 (× 1.8)

Transformer Benchmark

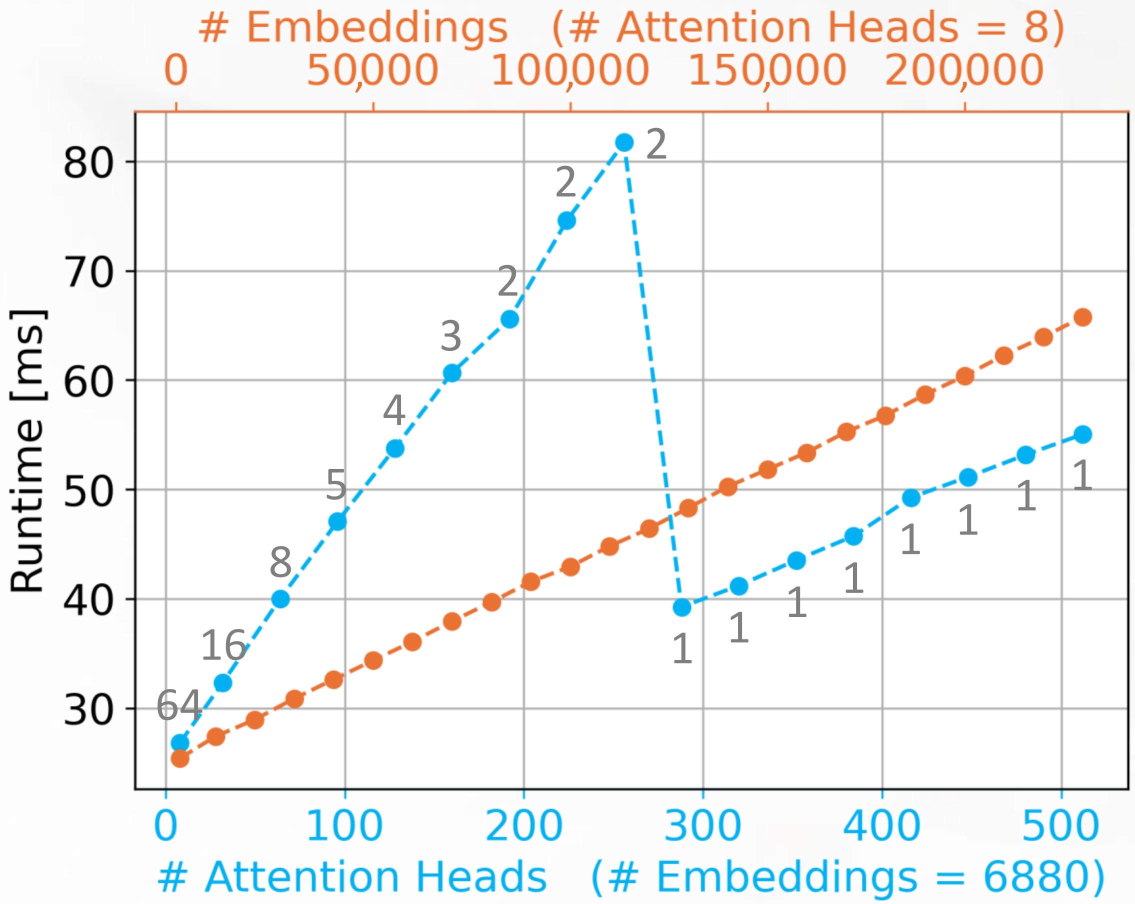


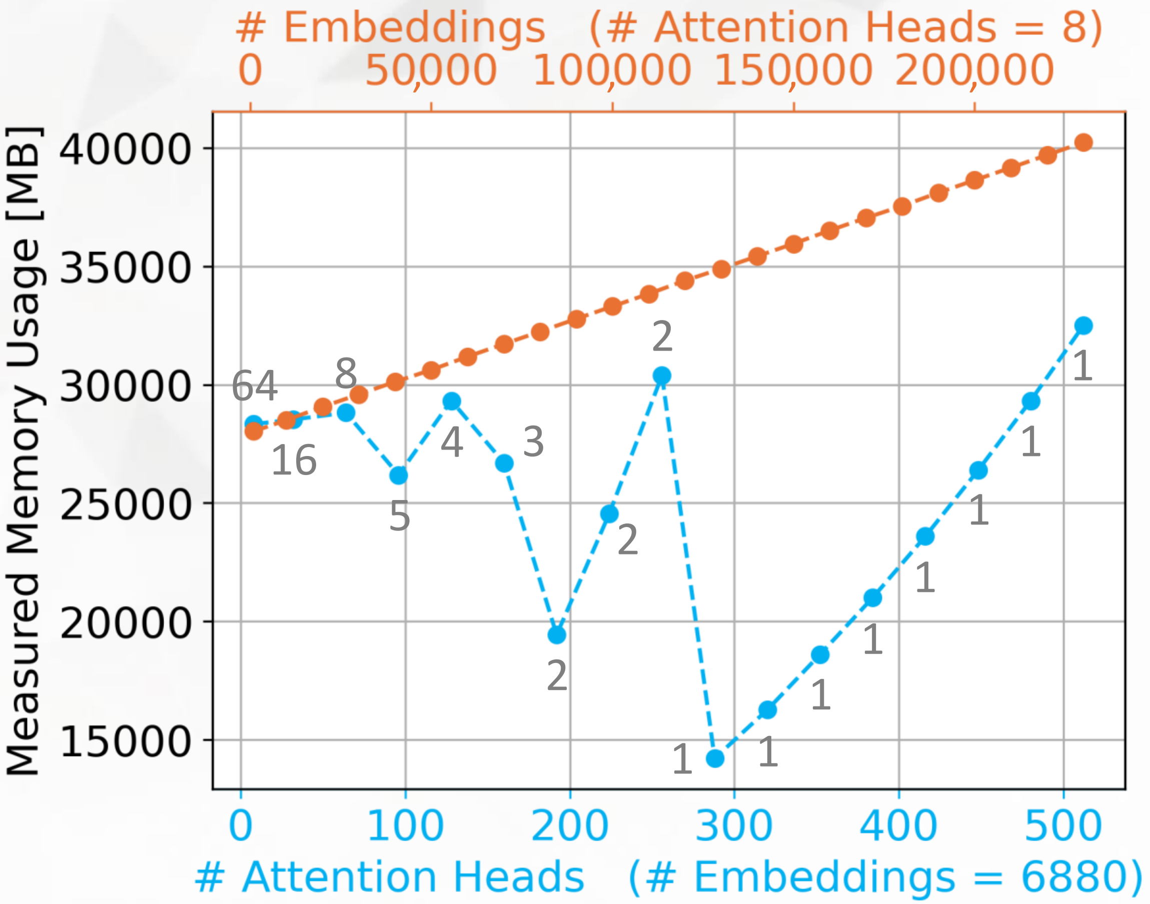


Transformer Benchmark



Transformer Benchmark





embedding
length per head