UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

CS 498: Machine Learning System
Spring 2025

Minjia Zhang

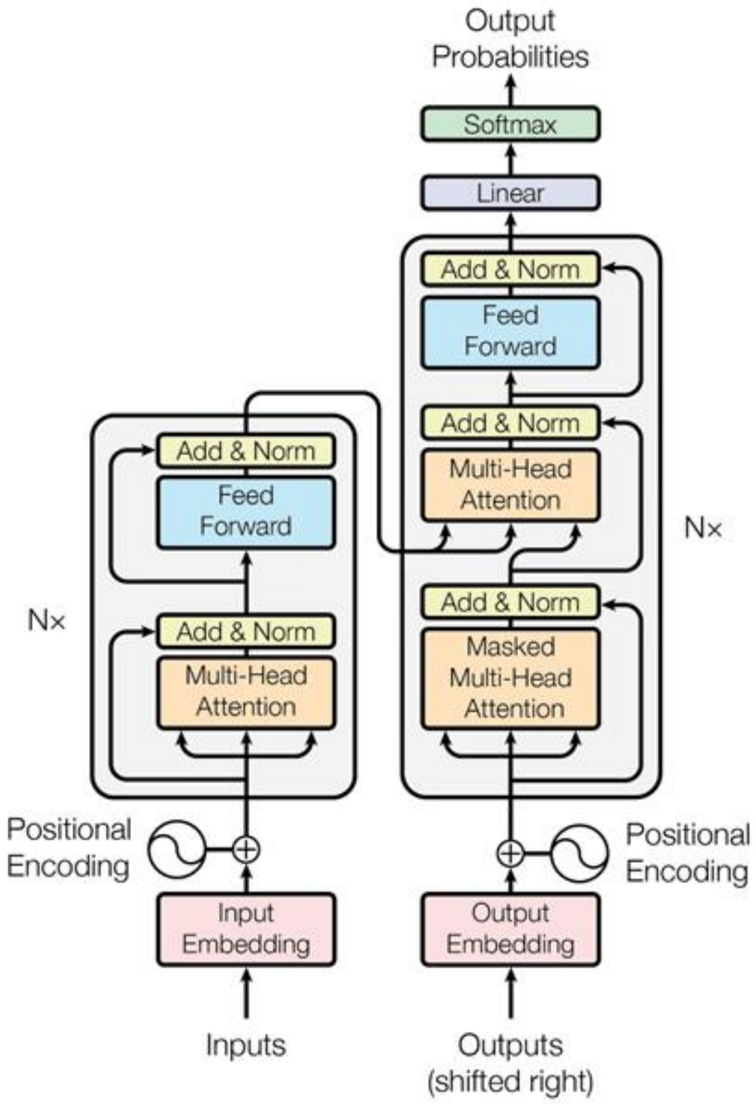
The Grainger College of Engineering

Today

- **Transformers Deep Dive**
- **Arithmetic Intensity**

Transformers

- Tokenization
- Input embeddings
- Position encodings
- Query, Key, Value
- Attention
- Self-attention
- Multi-head attention
- Feed forward
- Add & norm
- Residual
- Masked attention
- Causal attention
- Linear
- Softmax
- Encoders
- Decoder
- Encoder-decoder



Machine Translation

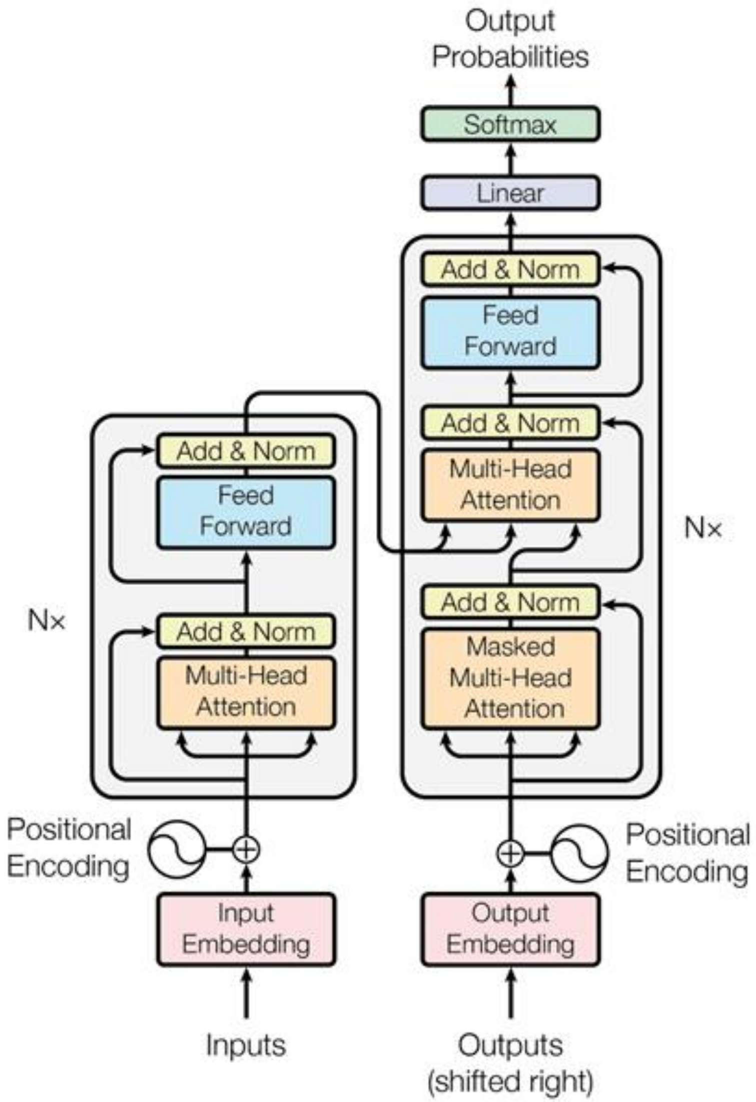
Targets

Ich habe einen Apfel gegessen

⬆

Inputs

I ate an apple



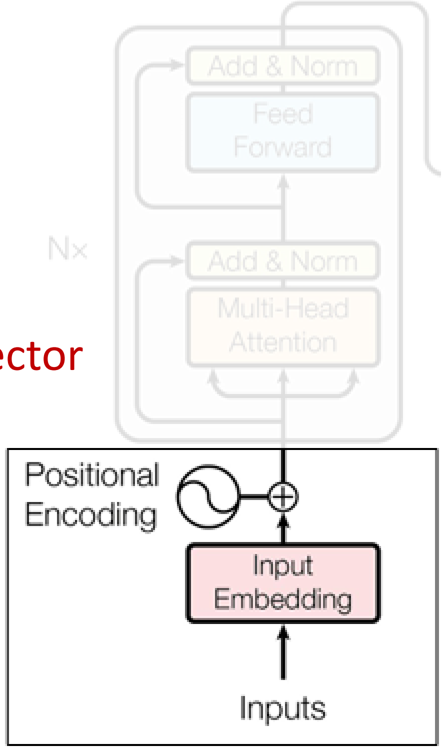
Tokenization

I | ate | an | apple | <eos>

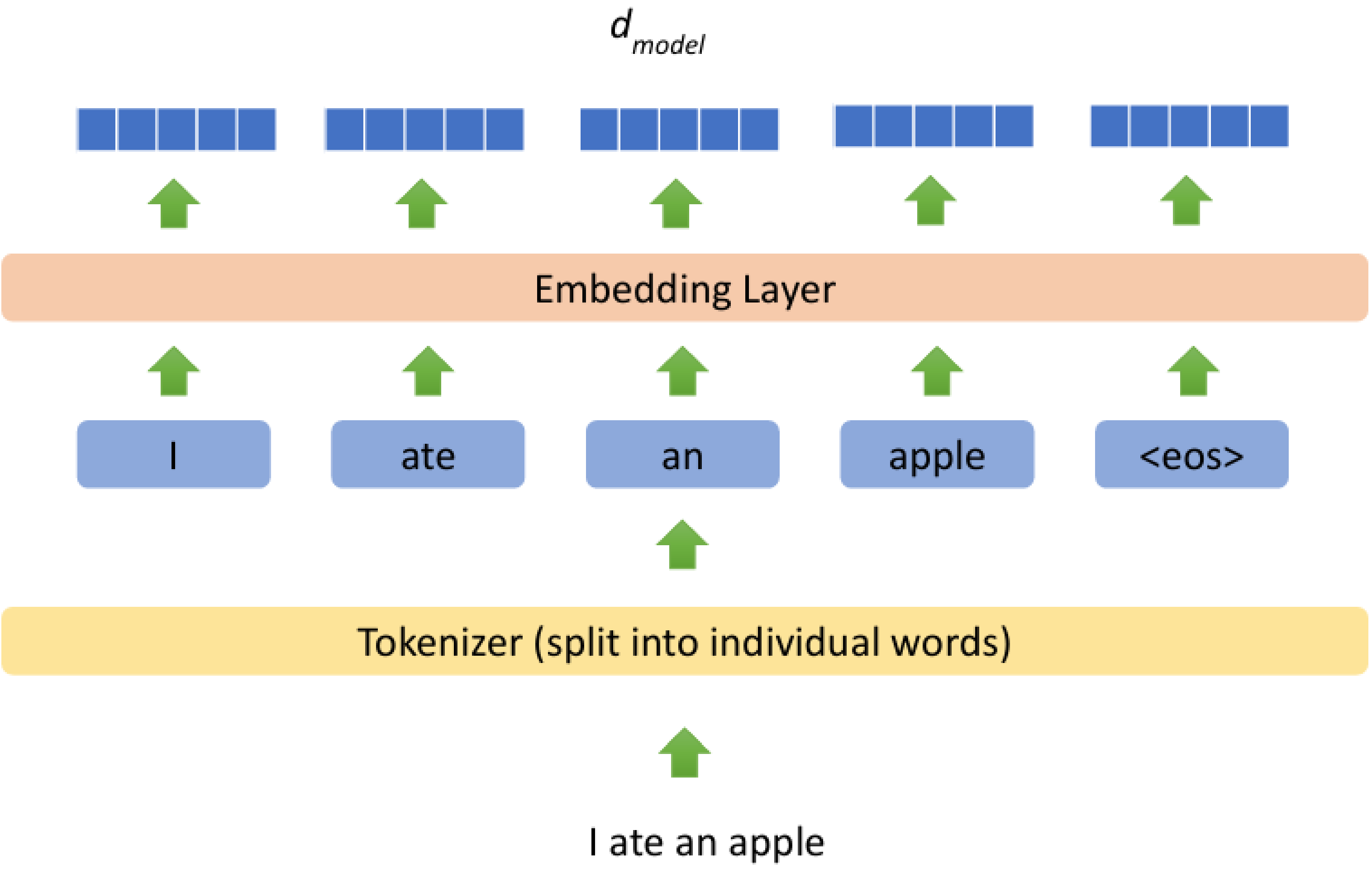
One hot vector

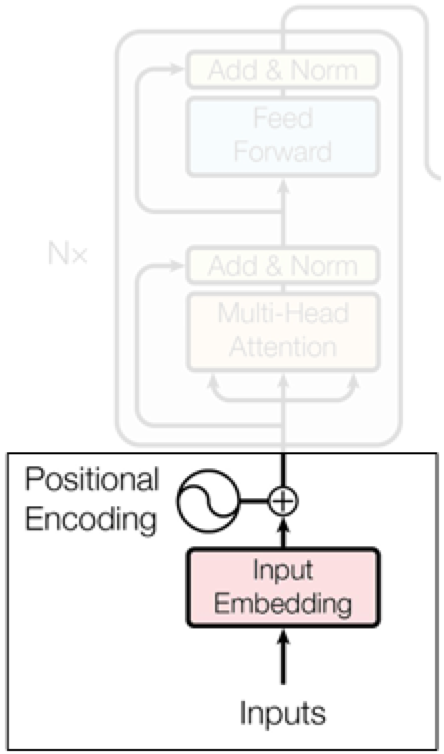
Tokenizer (split into individual words)

I ate an apple



Input Embeddings



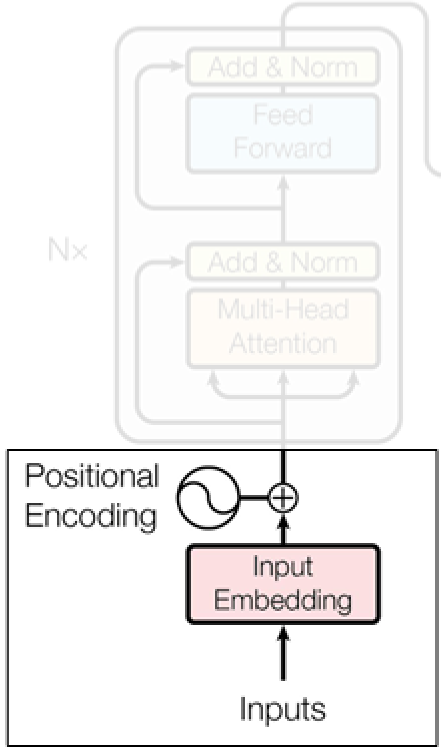


Generate Input Embeddings

Position Matters

I | ate | an | apple | <eos>

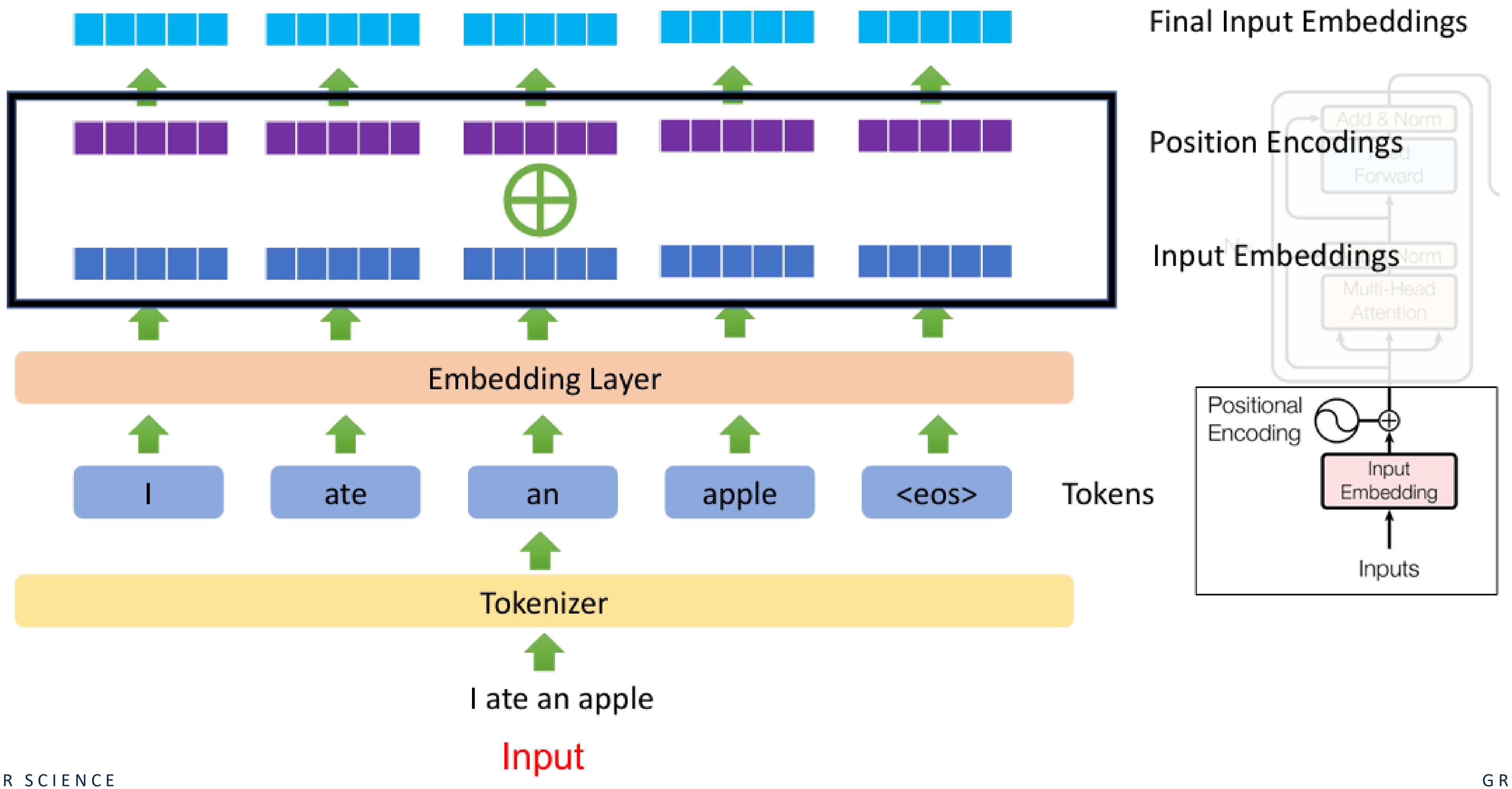
apple | ate | an | I | <eos>



- The position of words in a sentence carries information!
- **Idea:** Add some information to the representation at the beginning that indicates where it is in the sentence

Position Encodings

Absolute positional encoding (original Transformer): The token embeddings and the absolute position embedding are added together element-wise.

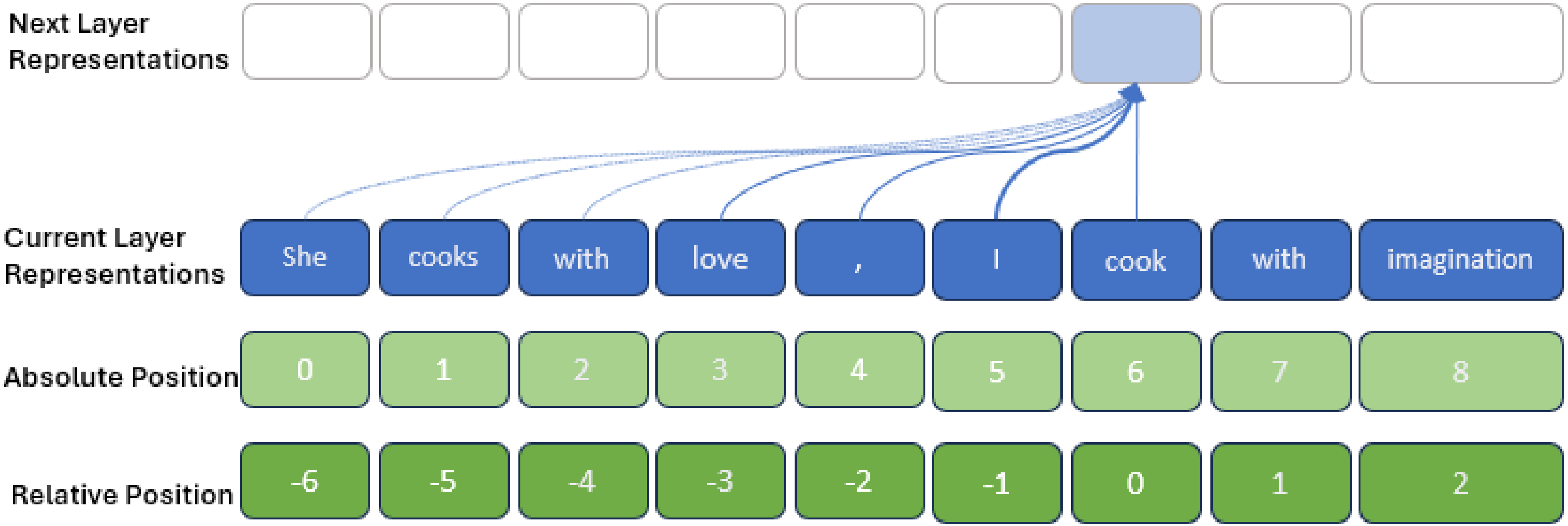


Position Encodings

- The original position embedding in Transformer is not a great idea, because absolute position is less important than relative position

Relative Position Encodings

- Translation invariance to position information
- Lead to performance improvements



Rotary Position Embedding

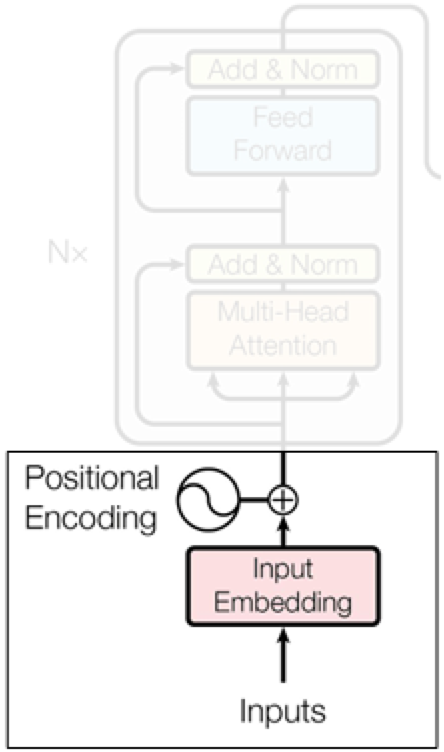
- Most recent LLMs adopt RoPE

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

Table 2: Comparing RoFormer and BERT by fine tuning on downstream GLEU tasks.

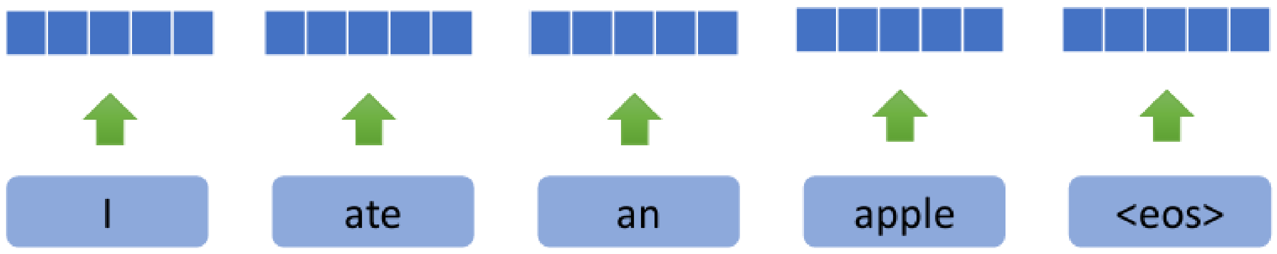
Model	MRPC	SST-2	QNLI	STS-B	QQP	MNLI(m/mm)
BERTDevlin et al. [2019]	88.9	93.5	90.5	85.8	71.2	84.6/83.4
RoFormer	**89.5**	90.7	88.0	**87.0**	**86.4**	80.2/79.8

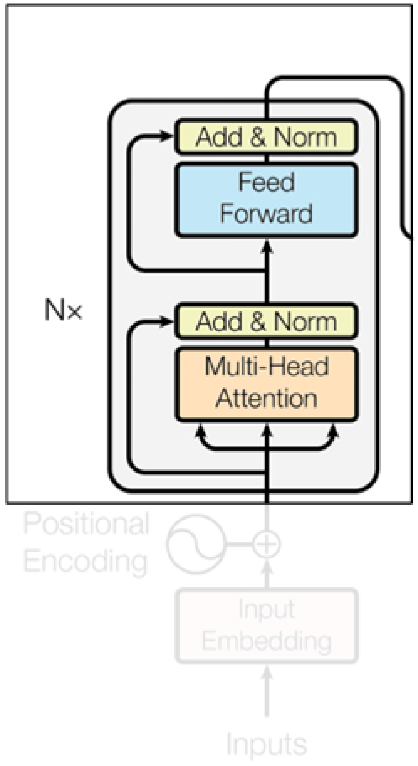


[REF: Rotary Position Embeddings]()

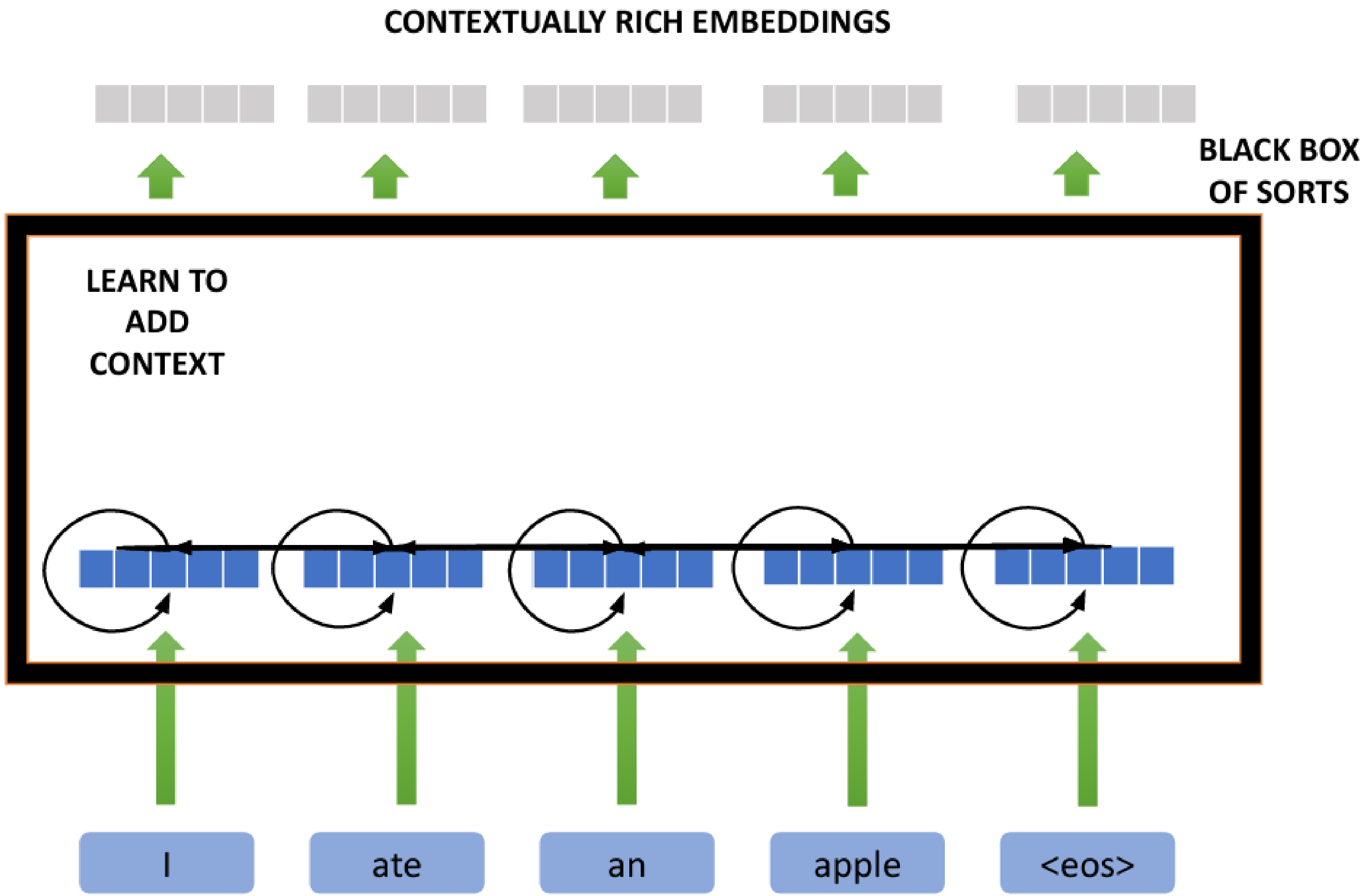
Encoder

WHERE IS THE CONTEXT ?

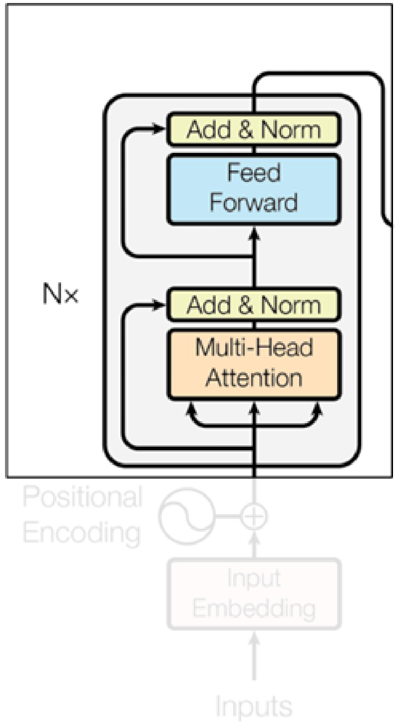




Encoder

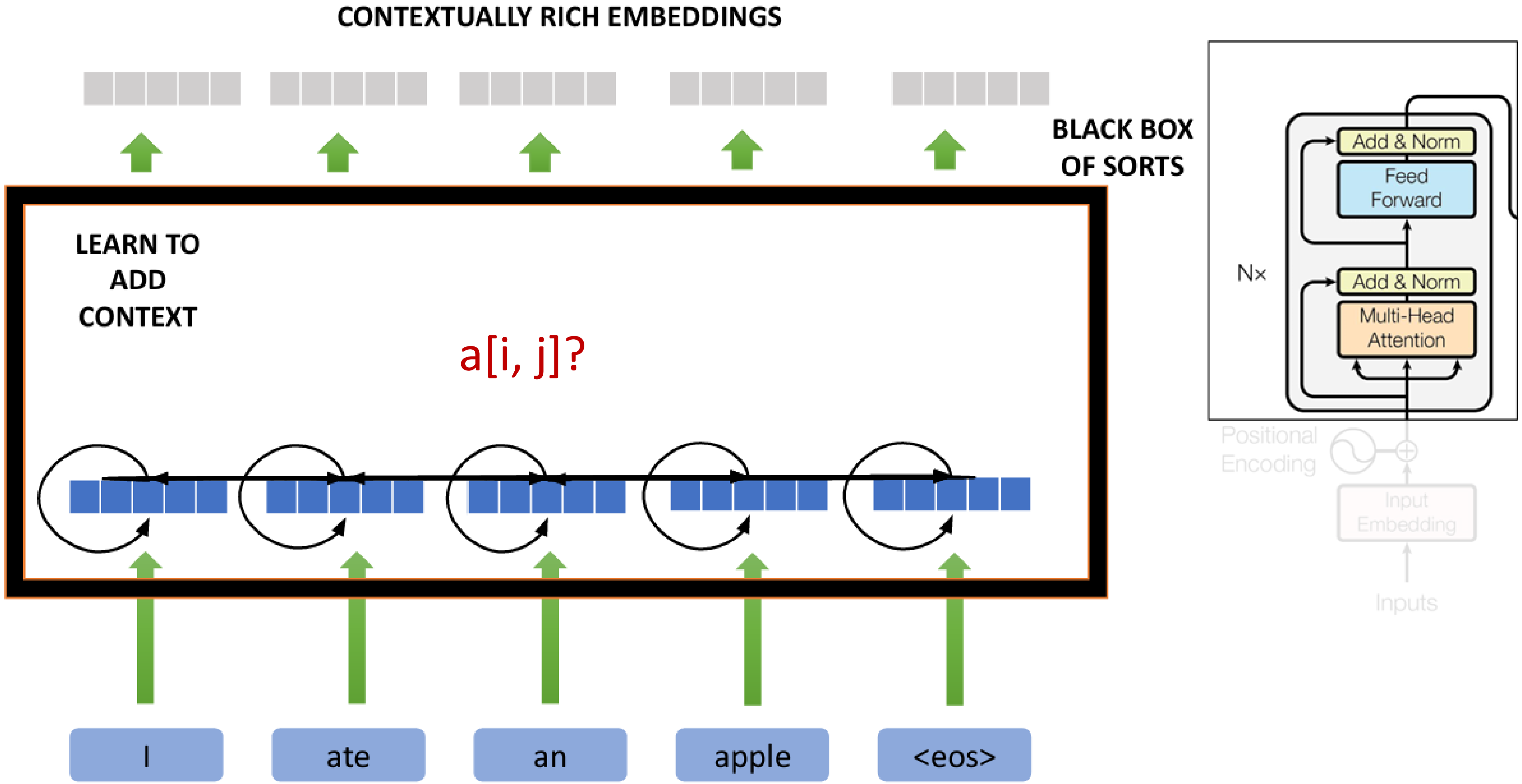


CONTEXTUALLY RICH EMBEDDINGS
BLACK BOX OF SORTS
LEARN TO ADD CONTEXT
I
ate
an
apple
<eos>



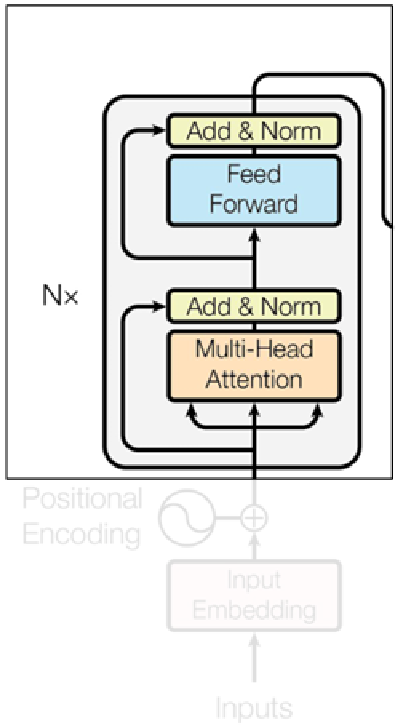
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

Encoder


CONTEXTUALLY RICH EMBEDDINGS
BLACK BOX OF SORTS
LEARN TO ADD CONTEXT
a[i, j]?
I
ate
an
apple
<eos>
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs

Multi-Head Self-Attention

- Before “multi-head” attention, what is “single head” attention?



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- Query
- Key
- Value

Multi-Head Self-Attention: Query, Key, & Value

- Database {key, value store}

{Query: "Order details of order_104"}

OR

{Query: "Order details of order_106"}

```
{"order_100": {"items":"a1", "delivery_date":"a2", ...}},
{"order_101": {"items":"b1", "delivery_date":"b2", ...}},
{"order_102": {"items":"c1", "delivery_date":"c2", ...}},
{"order_103": {"items":"d1", "delivery_date":"d2", ...}},
{"order_104": {"items":"e1", "delivery_date":"e2", ...}},
{"order_105": {"items":"f1", "delivery_date":"f2", ...}},
{"order_106": {"items":"g1", "delivery_date":"g2", ...}},
{"order_107": {"items":"h1", "delivery_date":"h2", ...}},
{"order_108": {"items":"i1", "delivery_date":"i2", ...}},
{"order_109": {"items":"j1", "delivery_date":"j2", ...}},
{"order_110": {"items":"k1", "delivery_date":"k2", ...}}
```

Query

1. Search for info

Key

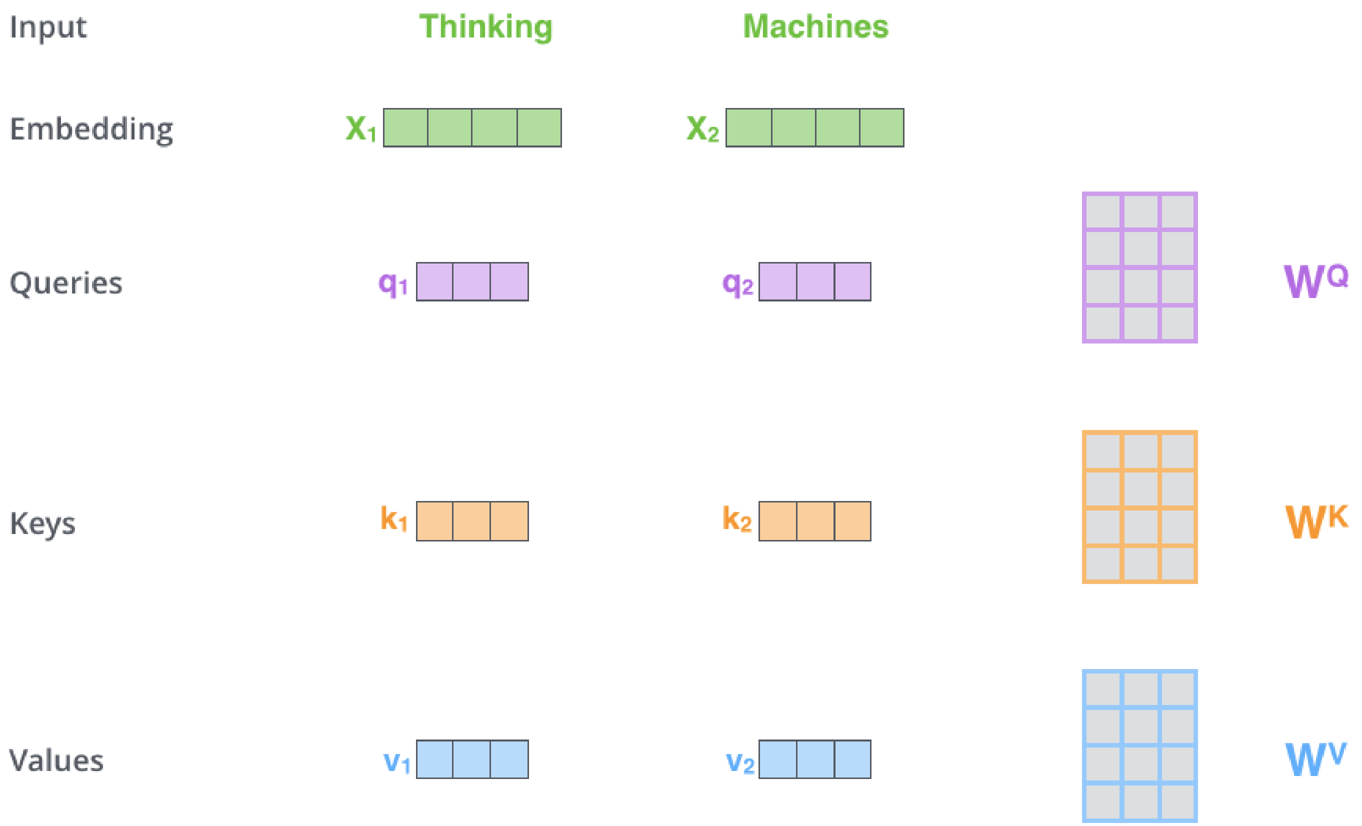
1. Interacts directly with Queries
2. Distinguishes one object from another
3. Identify which object is the most relevant and by how much

Value

1. Actual details of the object
2. More fine grained

Self-Attention

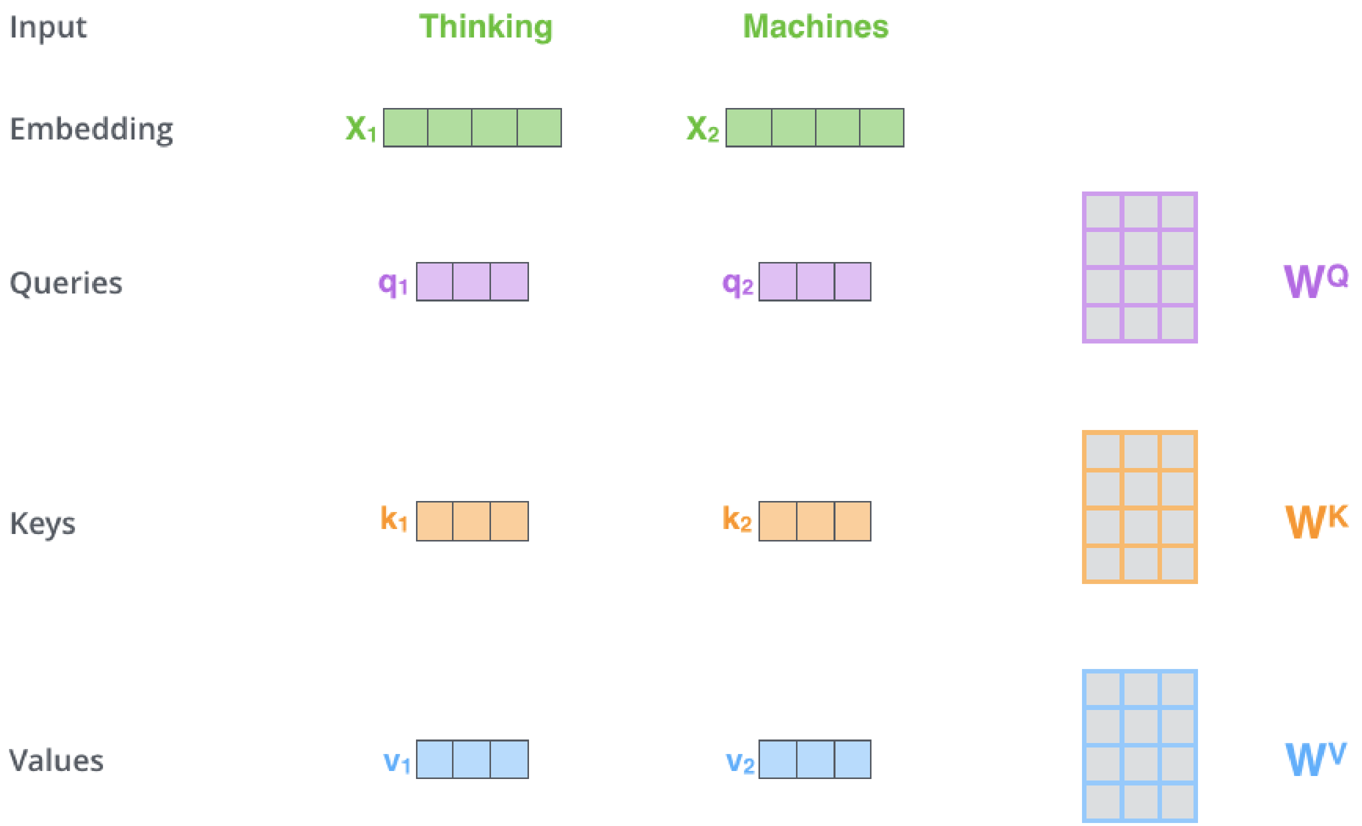
- Step 1: compute “key, value, query” embedding for each input token

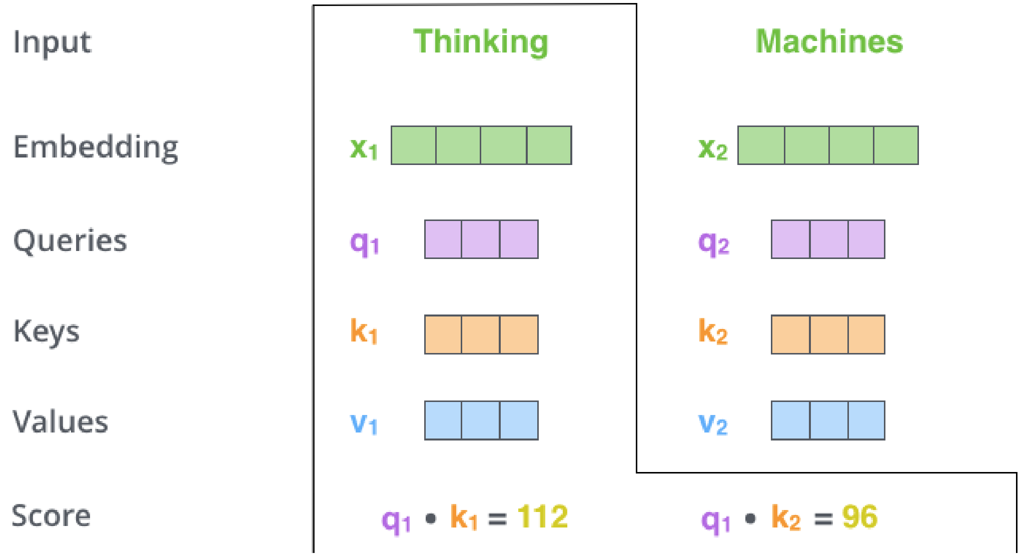


[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Self-Attention

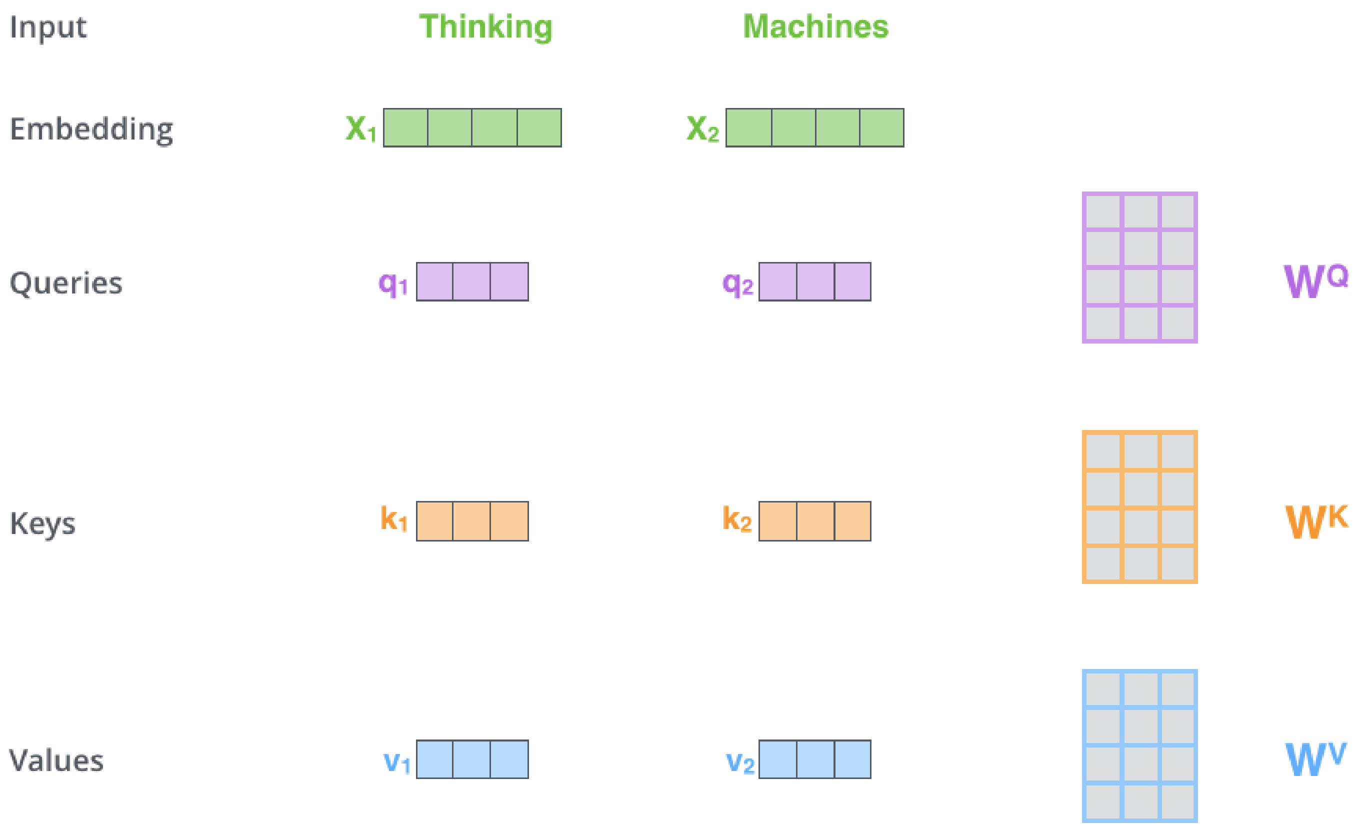
- **Step 1**: compute "key, value, query" embedding for each input token
- **Step 2:** compute scores between pairs of tokens

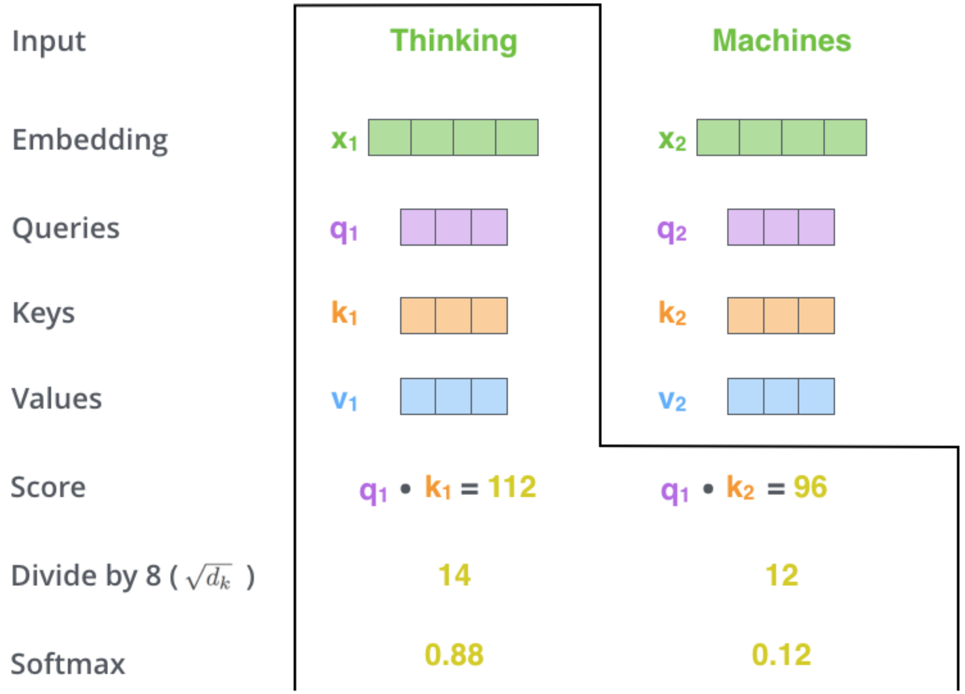




Self-Attention

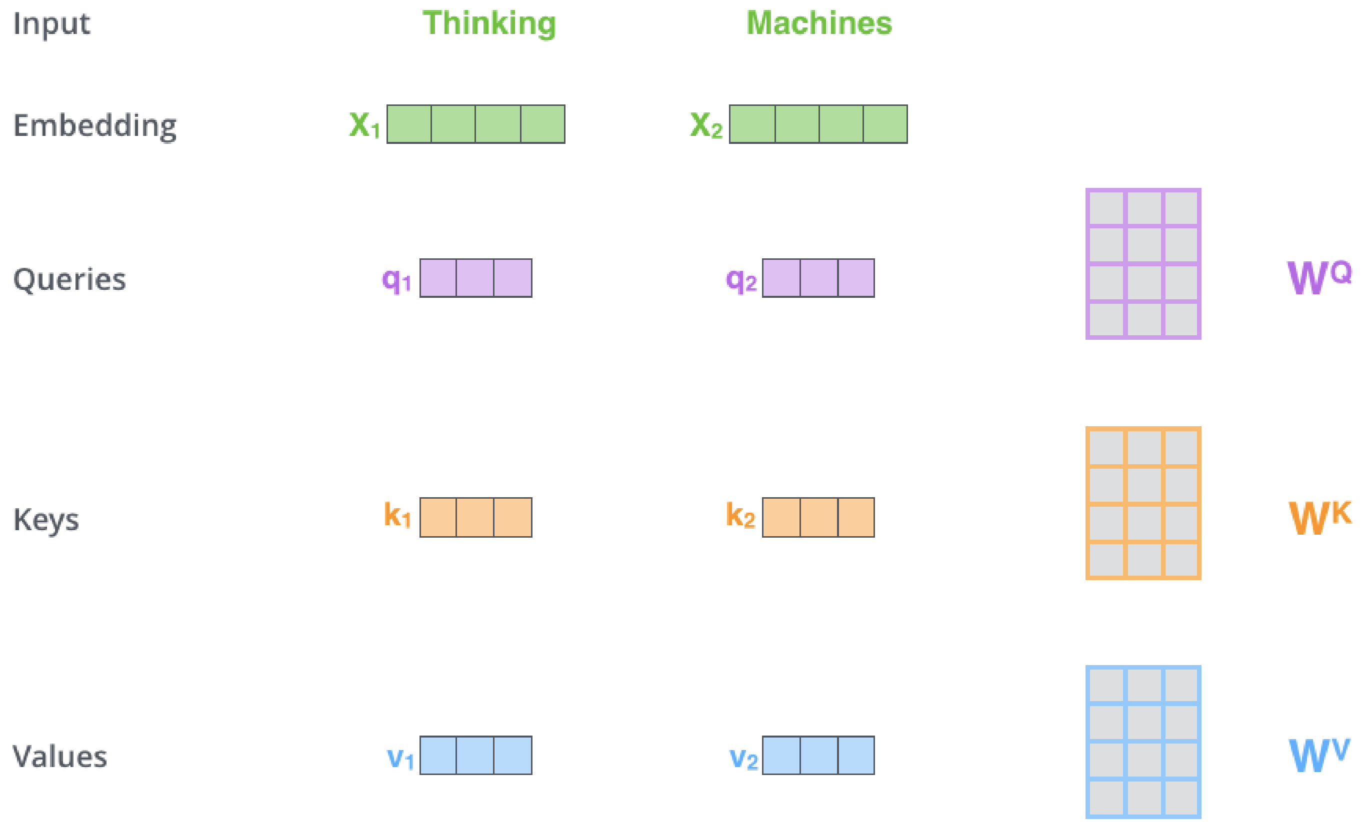
- Step 1: compute "key, value, query" embedding for each input token
- Step 2: compute scores between pairs of tokens
- Step 3: compute normalized attention scores

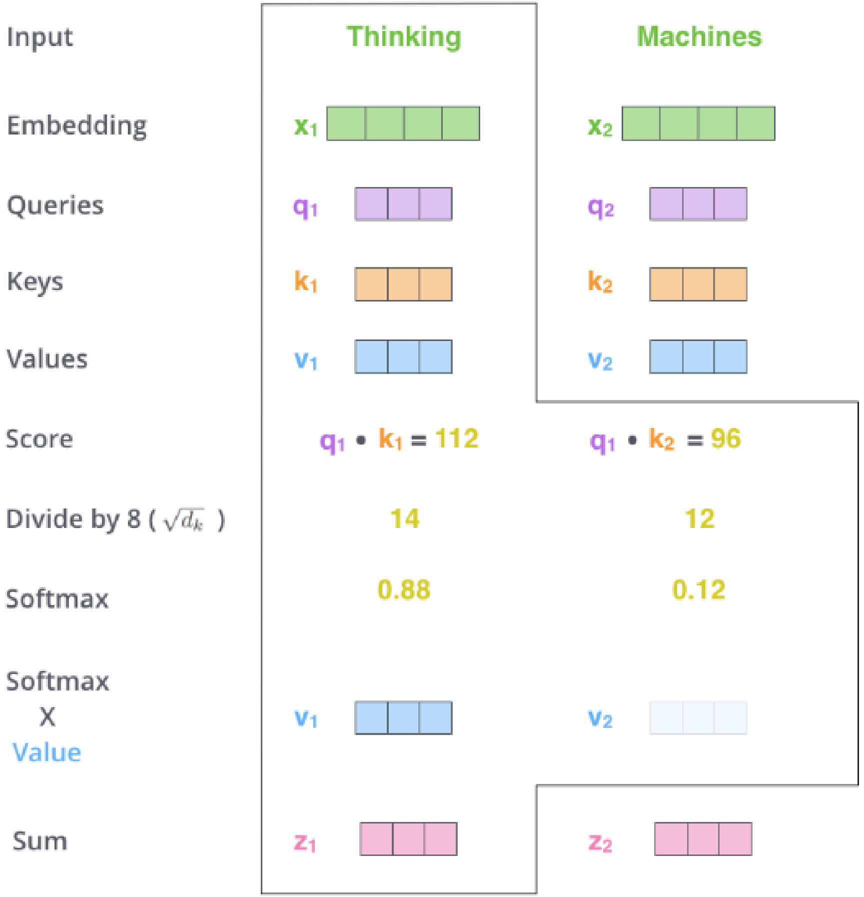




Self-Attention

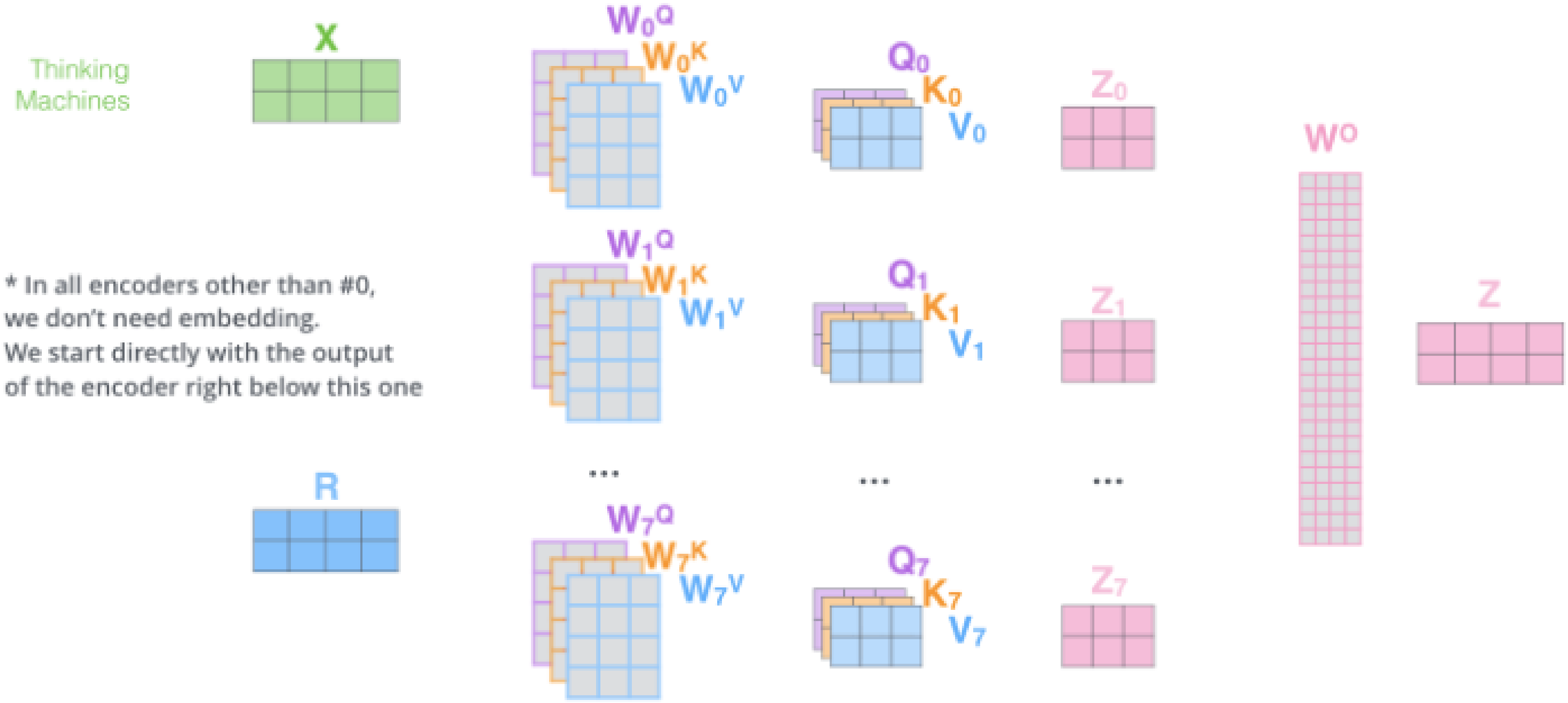
- Step 1: compute "key, value, query" embedding for each input token
- Step 2: compute scores between pairs of tokens
- Step 3: compute normalized attention scores
- Step 4: get new representation by weighted sum of values



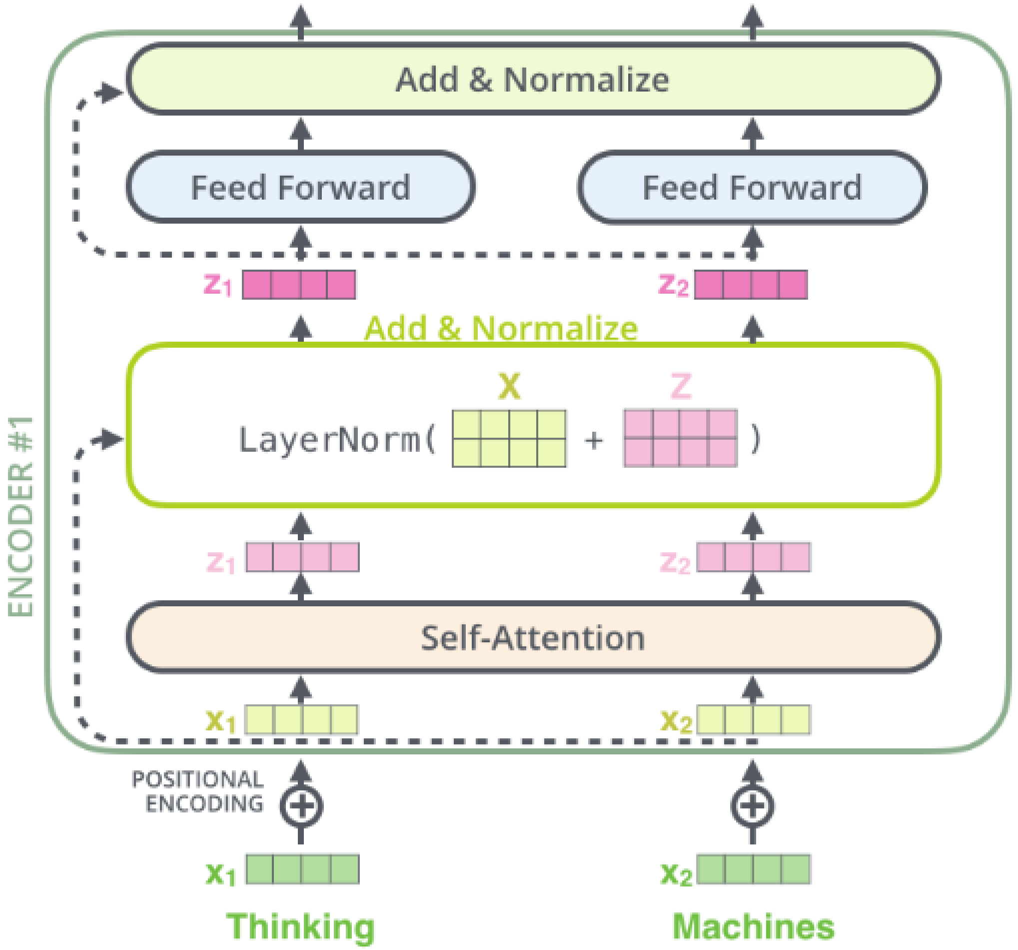


Multi-head Attention

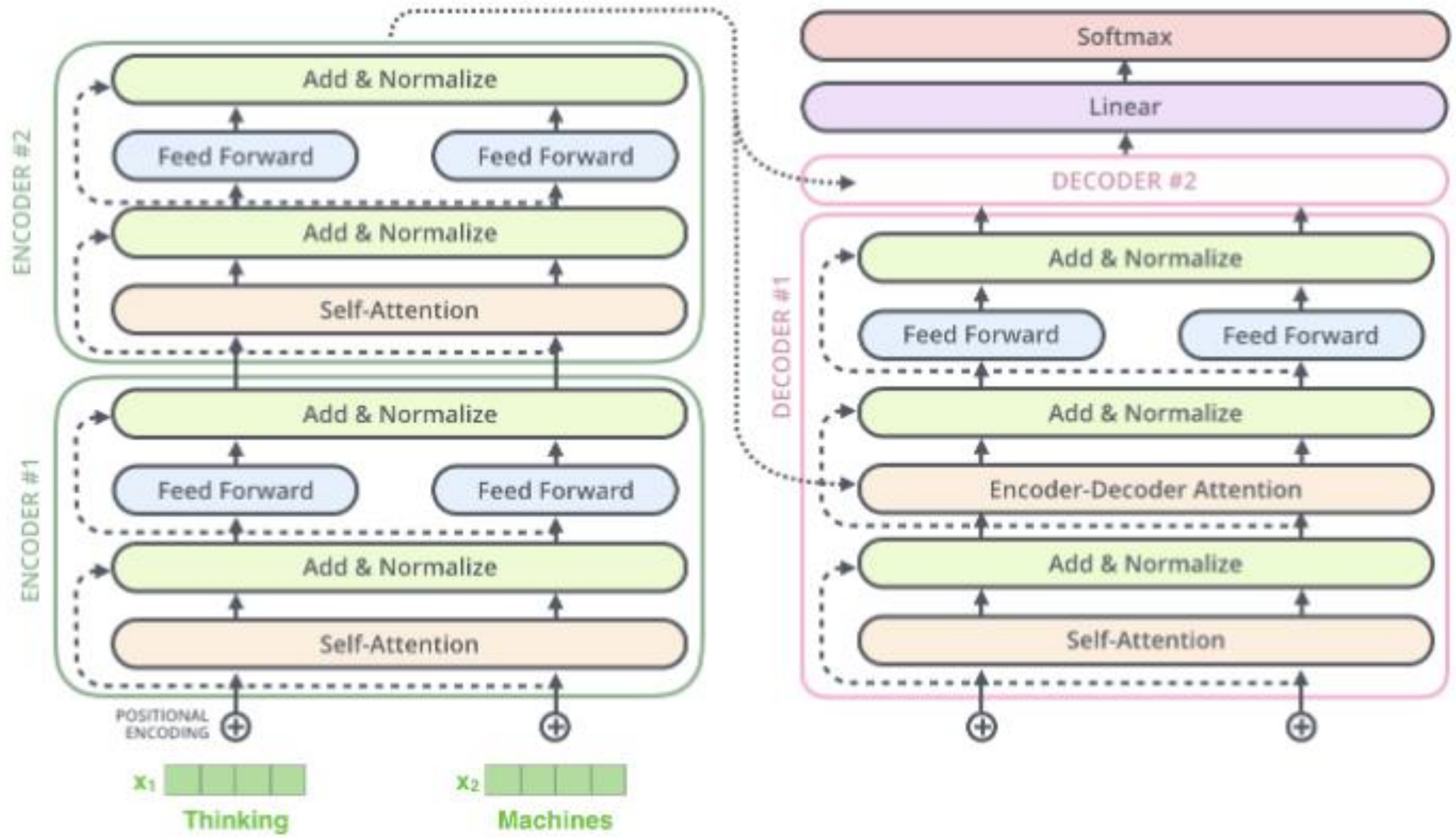
- Do many attention head calculation in parallel, and combine
- Each head has its own set of parameters
- Different heads can learn different “interactions” between inputs



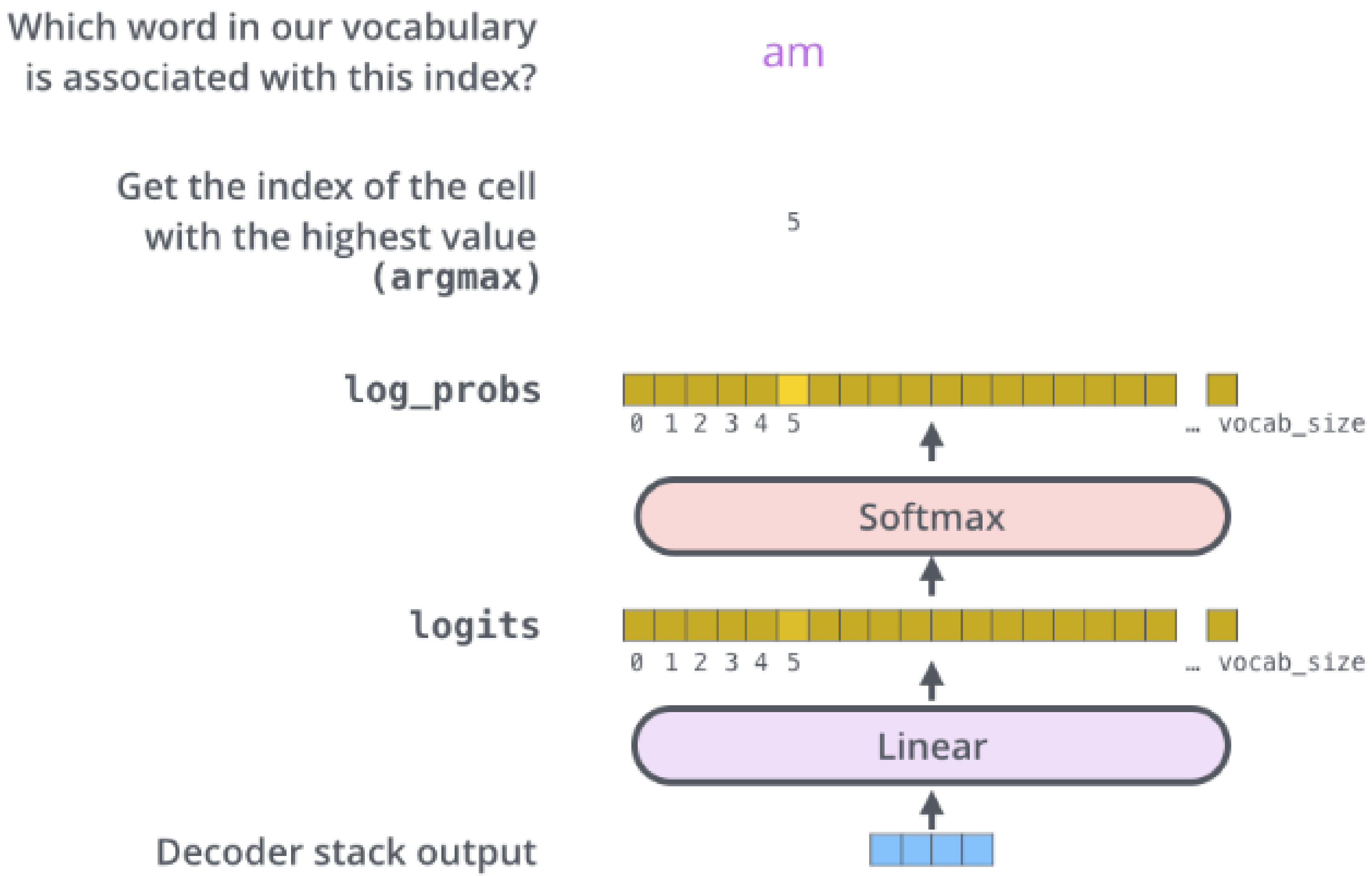
The Residuals and LayerNorm



The Decoder Side



The Final Linear and Softmax Layer



Arithmetic Intensity

AI = #ops / #bytes

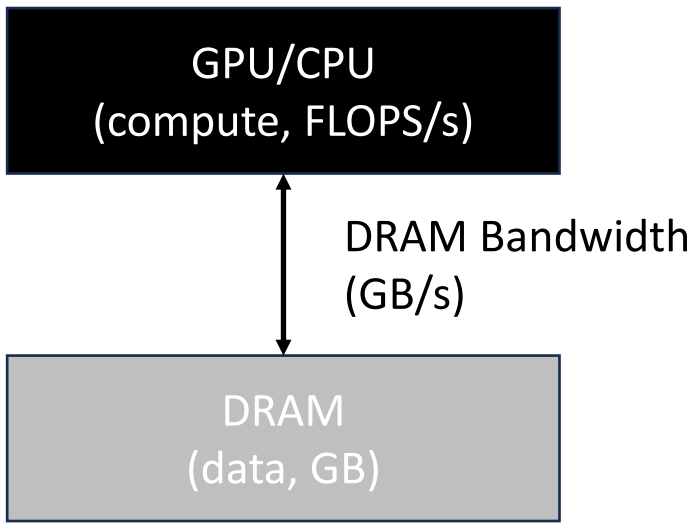
Arithmetic Intensity

AI= #ops / #bytes

Used to evaluate the efficiency of computational algorithms

System performance is bound by

1) The peak compute TFLOPS
2) The memory bandwidth

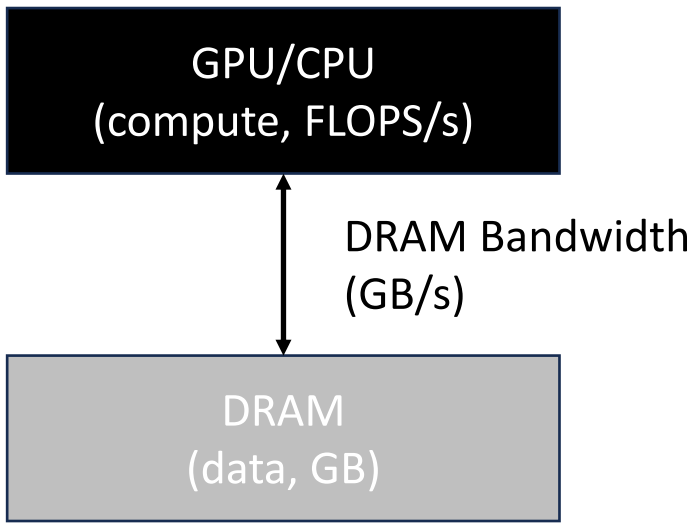


Arithmetic Intensity

AI= #ops / #bytes

System performance is bound by

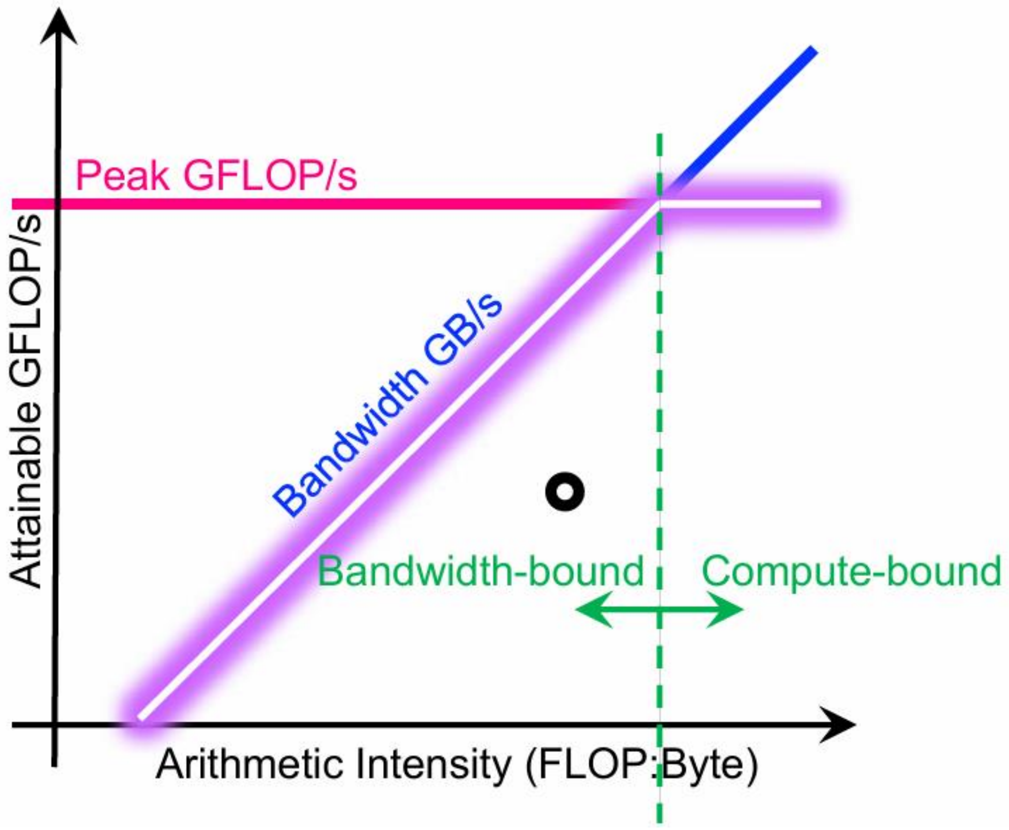
TFLOPS/s = min(Peak TFLOPS/s, Peak bw GB/s * AI)



Roofline Model

The Roofline model provides a relatively simple way for performance estimates based on the computation of workload and hardware characteristics

- High AI: Compute-bound
- Low AI: Memory bandwidth bound



Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, 2008

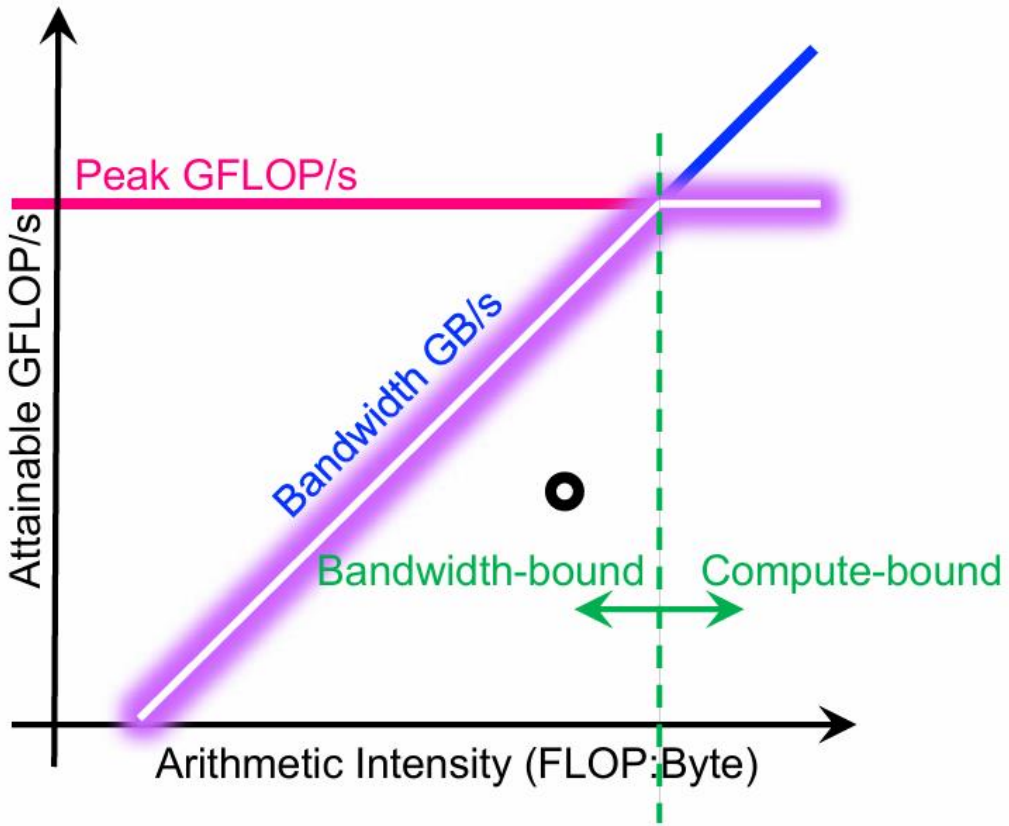
Why Roofline Model?

Helps identify the bottlenecks

Program performance depends on how well it fits the hardware architecture

Create optimizations to exhaust both compute and bandwidth at the same time (many times it is impossible)

The mode also tells you when to stop



Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, 2008

Theoretical Peak of AI

AI_A100 = compute / memory_bandwidth
= 312 TFLOPS / 800 GB/s
= 390 ops/ byte

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS
(SXM4 AND PCIE FORM FACTORS)

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935GB/s	2,039GB/s
Max Thermal Design Power (TDP)	300W	400W***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB
Form Factor	PCIe dual-slot air cooled or single-slot liquid cooled	SXM
Interconnect	NVIDIA® NVLink® Bridge for 2 GPUs: 600GB/s ** PCIe Gen4: 64GB/s	NVLink: 600GB/s PCIe Gen4: 64GB/s

Arithmetic Intensity

```
void add(int n, float* A, float* B, float* C){
  for (int i=0; i<n; i++)
    C[i] = A[i] + B[i];
}
```

Arithmetic Intensity

```
void add(int n, float*  A, float*  B, float*  C){
  for  (int  i=0; i<n; i++)
    C[i]  = A[i]  + B[i];
}
```

Two loads, one store per math op (arithmetic intensity = 1/3)

1. Read A[i]
2. Read B[i]
3. Add A[i] + B[i]
4. Store C[i]

Arithmetic Intensity

```
void add(int n, float*  A, float*  B, float*  C){
   for  (int  i=0; i<n; i++)
      C[i]  = A[i]  + B[i];
}

void mul(int  n, float*  A, float*  B, float*  C) {
   for  (int  i=0; i<n; i++)
      C[i]  = A[i]  *  B[i];
}

float*  A, *B,  *C,  *D,  *E,  *tmp1, *tmp2;
//  assume arrays are  allocated here
//  compute  E  = D + ((A   + B) *  C)
add(n, A, B, tmp1);
mul(n,  tmp1,  C, tmp2);
add(n, tmp2,  D, E);
```

Two loads, one store per math op
(arithmetic intensity = 1/3)

Arithmetic Intensity

```
void add(int n, float* A, float* B, float* C){
   for (int i=0; i<n; i++)
      C[i] = A[i] + B[i];
}

void mul(int n, float* A, float* B, float* C) {
   for (int i=0; i<n; i++)
      C[i] = A[i] * B[i];
}

float* A, *B, *C, *D, *E, *tmp1, *tmp2;
// assume arrays are allocated here
// compute E = D + ((A + B) * C)
add(n, A, B, tmp1);
mul(n, tmp1, C, tmp2);
add(n, tmp2, D, E);
```

Two loads, one store per math op (arithmetic intensity = 1/3)

Two loads, one store per math op (arithmetic intensity = 1/3)

Overall arithmetic intensity = 1/3

Which Program Performs Better?

```
float*  A, *B,  *C,  *D,  *E,  *tmp1, *tmp2;
//   assume arrays are  allocated here
//   compute  E  = D + ((A  + B) *  C)
add(n,  A, B, tmp1);
mul(n,  tmp1,  C, tmp2);
add(n,  tmp2,  D, E);



void fused(int  n, float*  A, float*  B, float*  C, float*  D,
   float*  E) {
   for  (int  i=0; i<n;  i++)
      E[i]  = D[i]  + (A[i] + B[i])  *  C[i];
}
//   compute  E  = D + (A + B) *  C
fused(n,  A,  B,C,   D, E);
```

Which Program Performs Better?

```
float*  A, *B,  *C,  *D,  *E,  *tmp1, *tmp2;
//  assume arrays are  allocated here
//  compute  E = D + ((A  + B) *  C)
add(n, A, B, tmp1);
mul(n,  tmp1,  C, tmp2);
add(n, tmp2,  D, E);



void fused(int  n, float*  A, float*  B, float*  C, float*  D,
   float*  E) {
   for  (int  i=0; i<n; i++)
      E[i]  = D[i]  + (A[i] + B[i])  *  C[i];
}
//  compute  E = D + (A + B) *  C
fused(n,  A,  B,C,   D, E);
```

Overall arithmetic intensity = 1/3

Four loads, one store per 3 math ops
Arithmetic intensity = 3/5

Understanding Transformer Calculations

Input b: batch size; s: sequence length

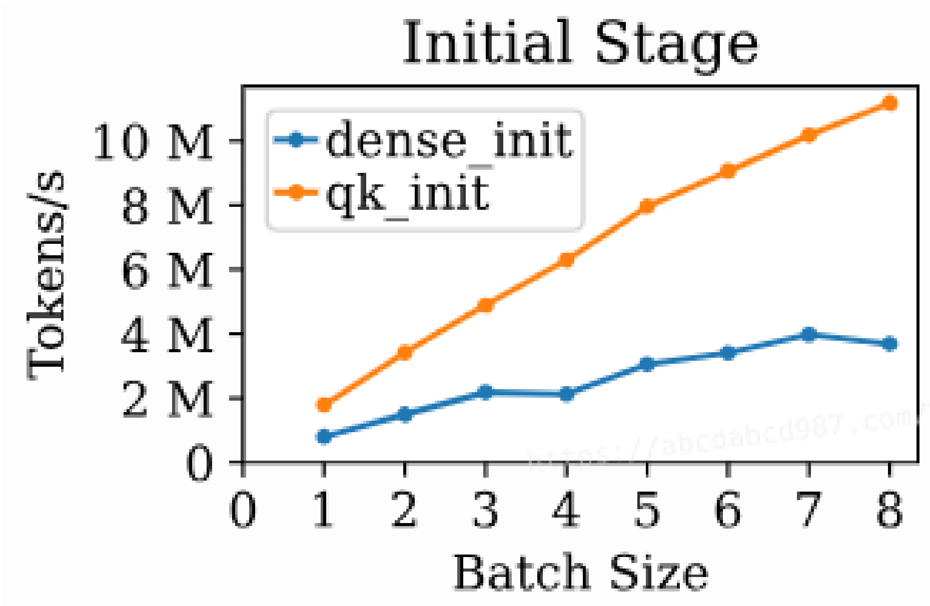
Model n: the number of attention heads; d: dimension of single attention head

h: hidden dimension (h = n x d)

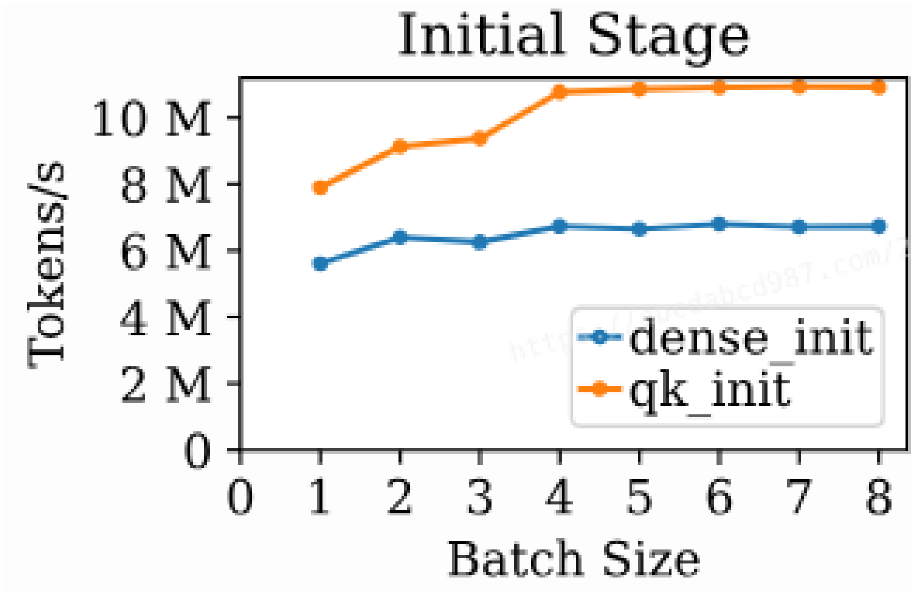
Symbol	Definition	Shape	FLOP	I/O	FLOP:I/O
Input					
X	Input for self attention	(b, s, h)			
Self-Attention					
$Q^{\ddagger}, K^{\ddagger}, V^{\ddagger}$	XW_Q, XW_K, XW_V	(b, s, h)	$O(bsh^2)$	$O(2bsh + h^2)$	$O(1/(1/h + 1/bs))$
$Q^{\dagger}, K^{\dagger}, V^{\dagger}$	Reshape $Q^{\ddagger}, K^{\ddagger}, V^{\ddagger}$	(b, s, n, d)			
Q, K, V	Transpose $Q^{\dagger}, K^{\dagger}, V^{\dagger}$	(b, n, s, d)			
K^T	Transpose K	(b, n, d, s)			
P	Softmax $\left(QK^T/\sqrt{d}\right)$	(b, n, s, s)	$O(bs^2nd)$	$O(2bsnd + bs^2n)$	$O(1/(1/d + 1/s))$
$A^{\ddagger}$	PV	(b, n, s, d)	$O(bs^2nd)$	$O(2bsnd + bs^2n)$	$O(1/(1/d + 1/s))$
$A^{\dagger}$	Transpose $A^{\ddagger}$	(b, s, n, d)			
A	Reshape $A^{\dagger}$	(b, s, h)			
Y	AW_O	(b, s, h)	$O(bsh^2)$	$O(2bsh + h^2)$	$O(1/(1/h + 1/bs))$
Feed-Forward Network					
Z	$\mathrm{ReLU}(YW_1)W_2$	(b, s, h)	$O(16bsh^2)$	$O(2bsh + 8h^2)$	$O(1/(1/h + 1/bs))$

Transformer Performance: Varying Batch Sizes

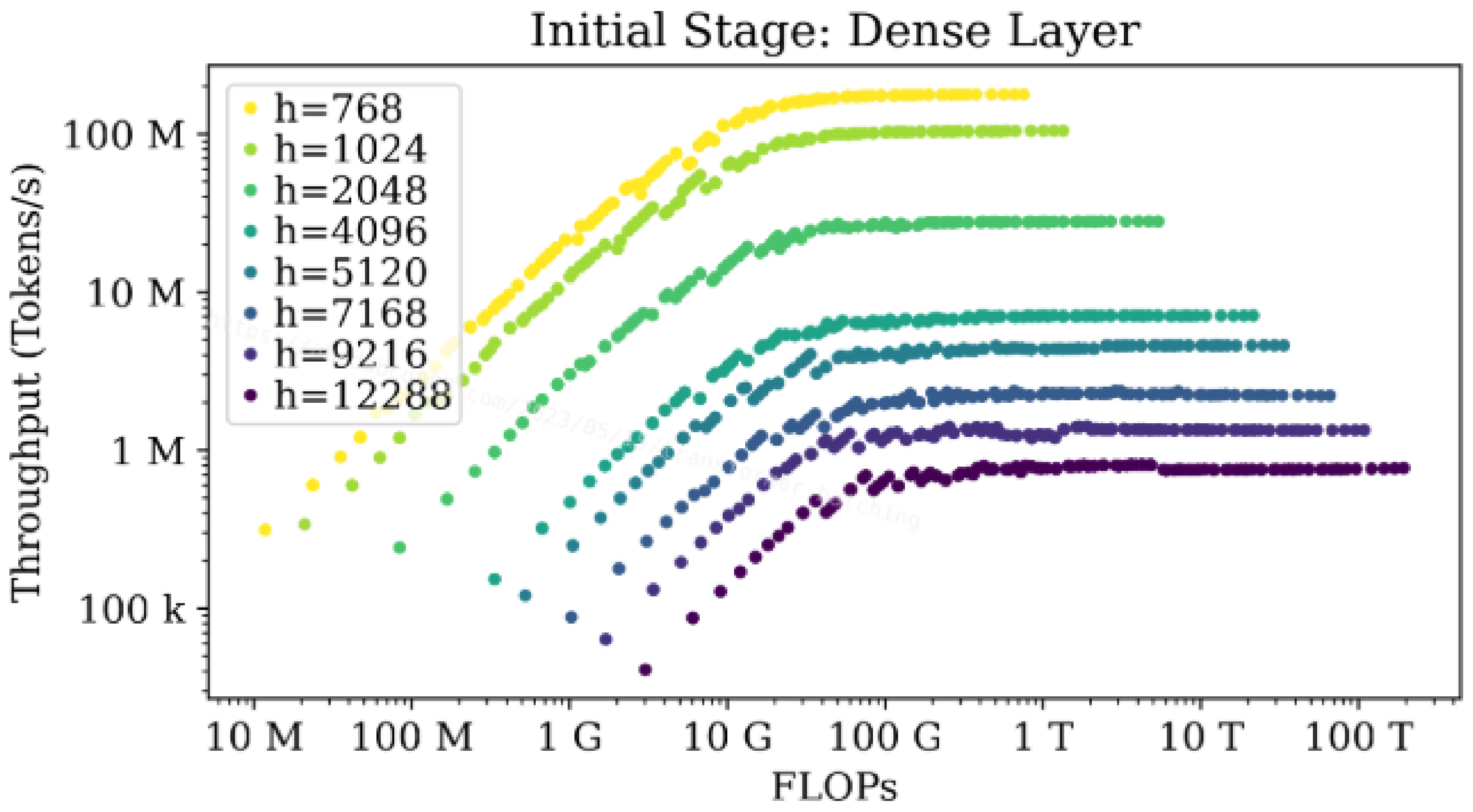
h=4096 s=50



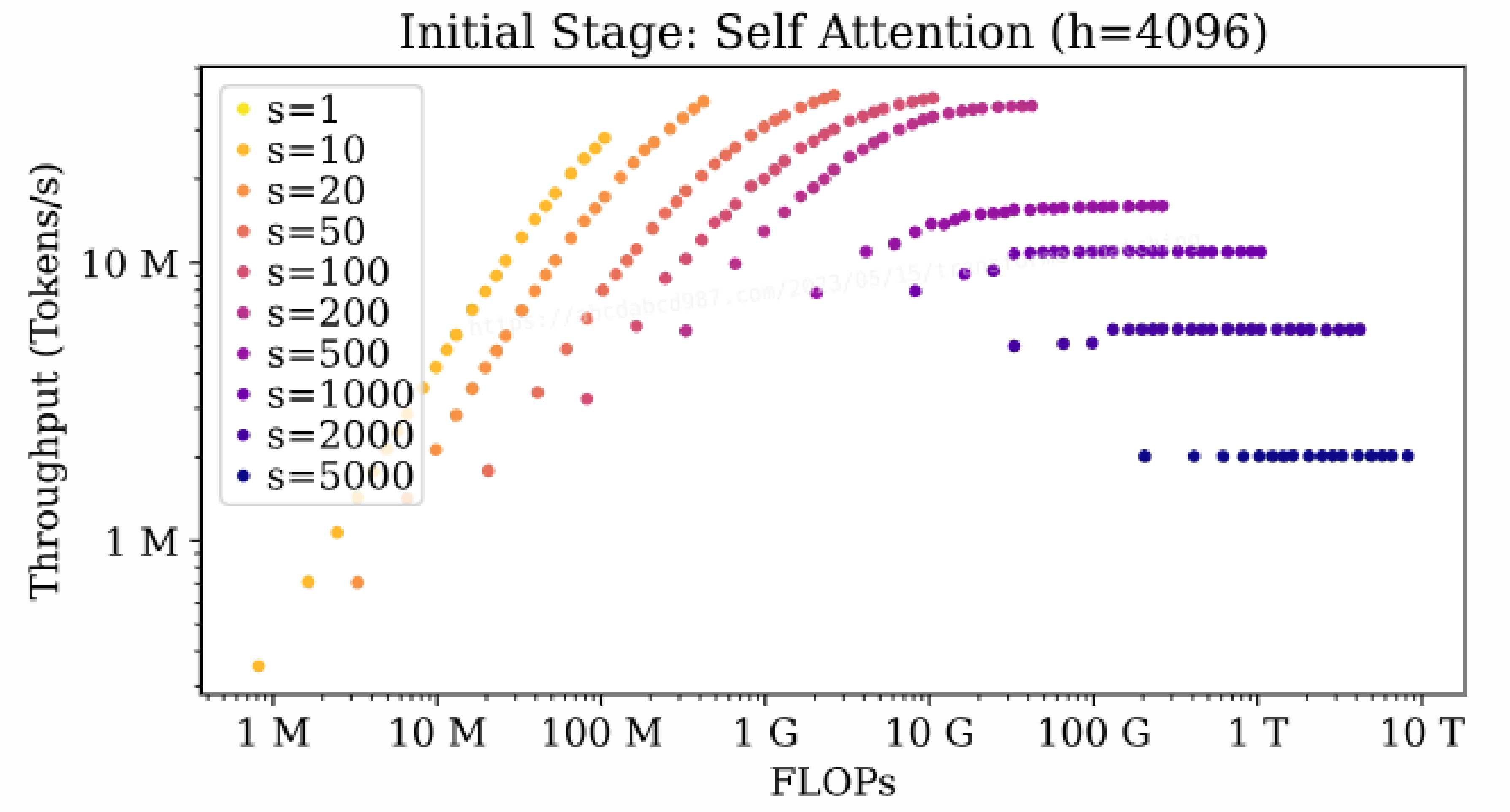
h=4096 s=1000



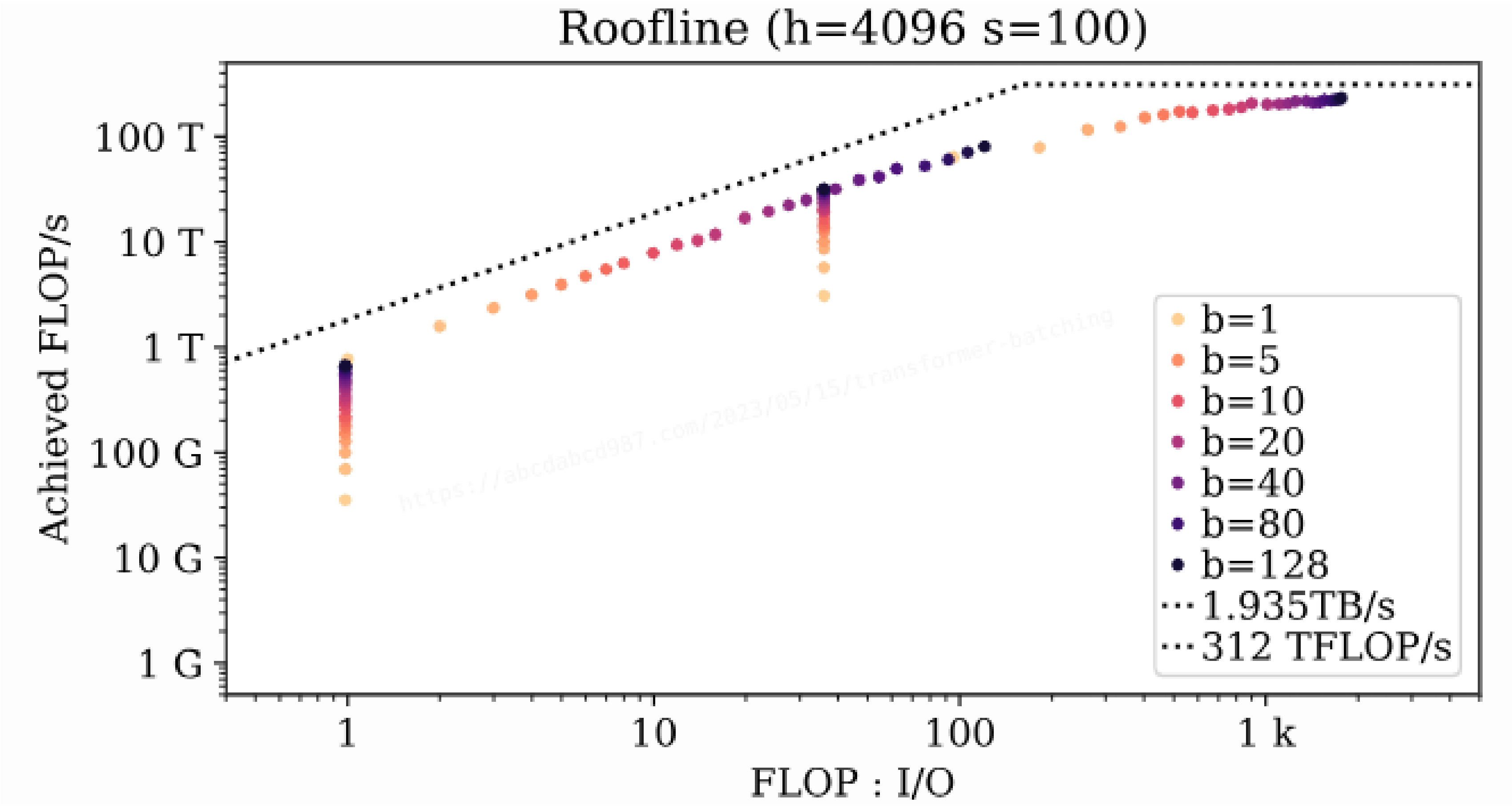
Transformer Performance: Batching of Dense Layer



Transformer Performance: Batching of Self-Attention



Transformer Performance: Roofline



- After class:
  - Walk through the AI calculation of Transformers
  - How many floating-point operations in total for a 7B decoder-only model?

Questions?