UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

ST7003 Procesamiento Natural del Lenguaje

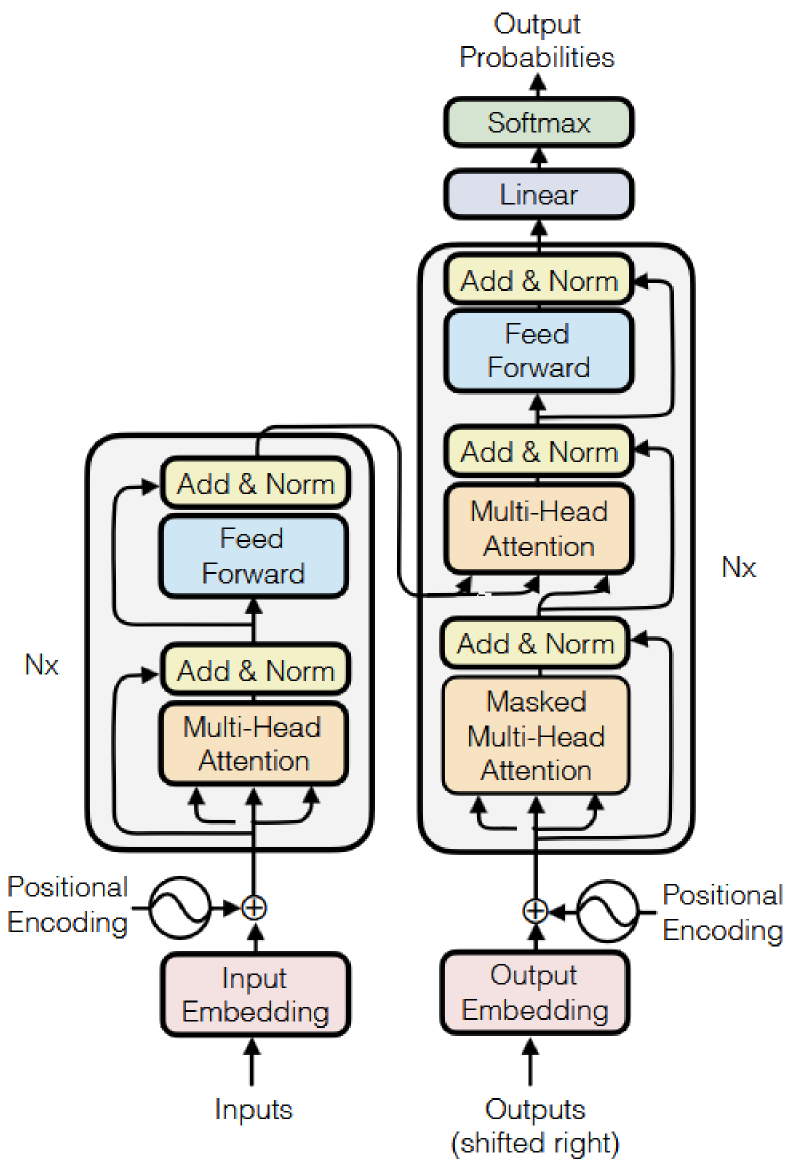
Lecture10 – Supervised Fine-tuning

Contenido

1. Masked multi head attention
2. LLMs training
3. Model architecture
4. Fine-tuning
5. Hands-on

UNIVERSIDAD EAFIT

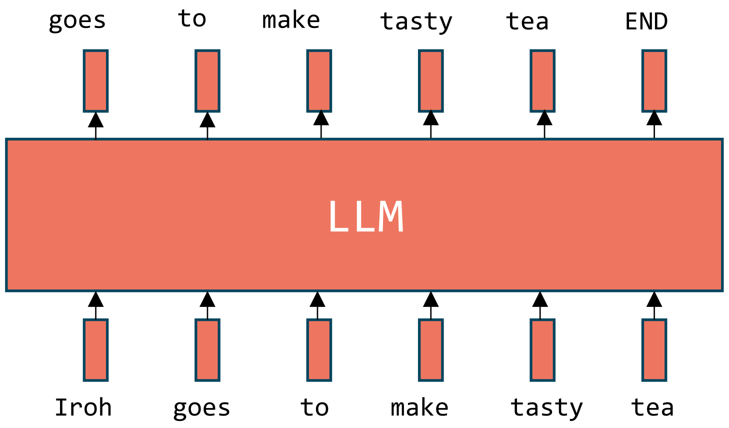
Escuela de Ciencias Aplicadas e Ingeniería



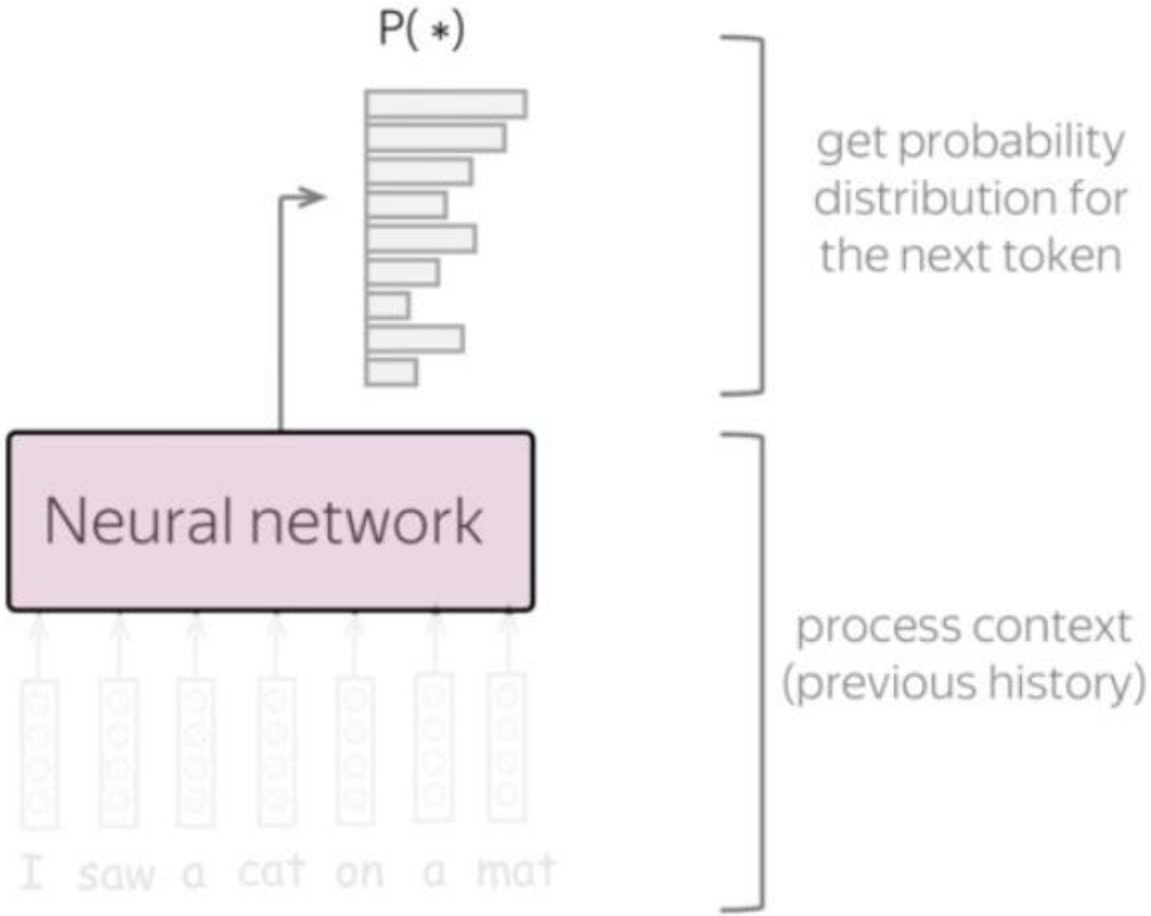
Next token prediction

Universidad EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

- Model $p\theta(w_t \mid w_{1:t-1})$, the probability distribution over words given their preceding context.
- There is a lot of data for this! (In English!)
- Pre-training using language modeling: Train a neural network to perform language modeling on a large amount of text.
- Save the network's parameters.



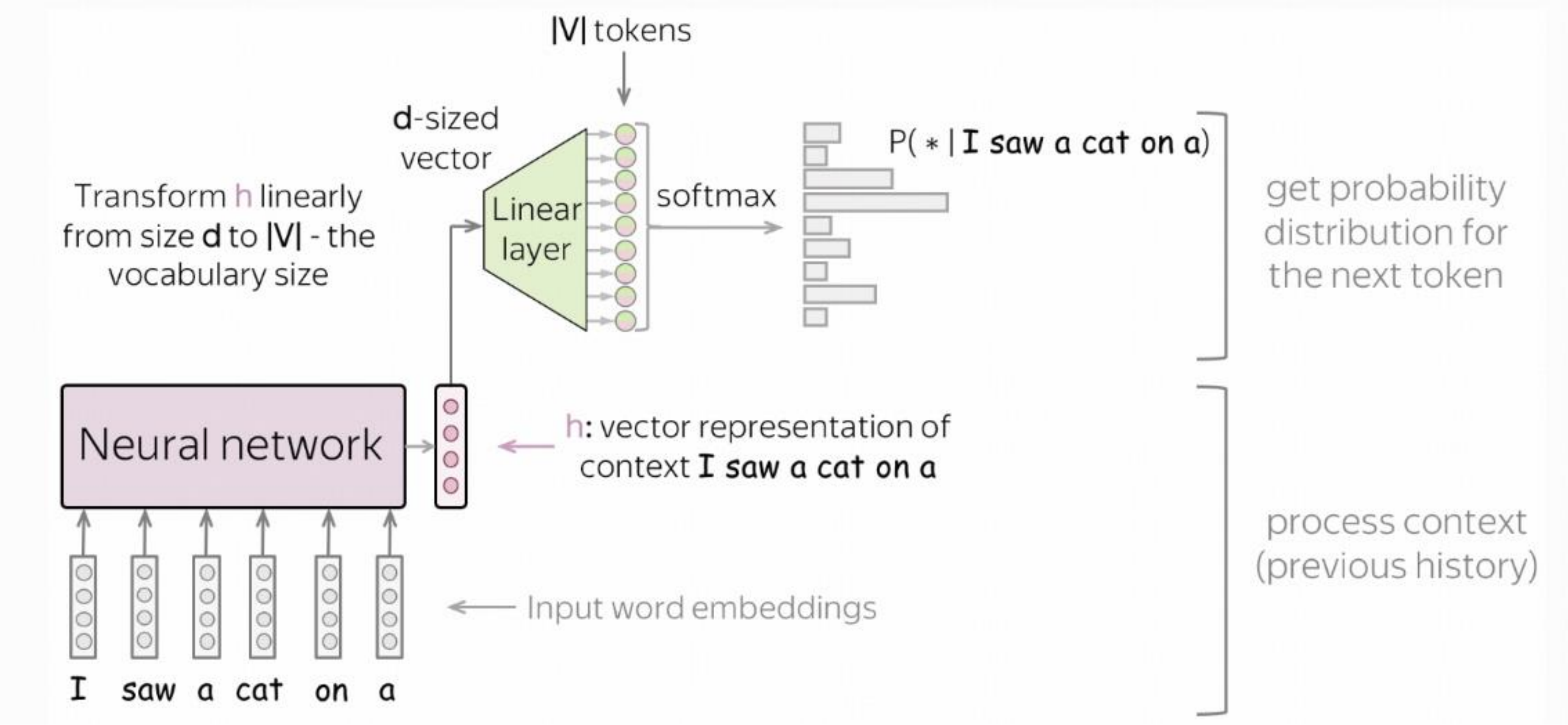
Next token prediction



This is classification! We can think of neural language models as neural classifiers. They classify prefix of a text into |V| classes, where the classes are vocabulary tokens.

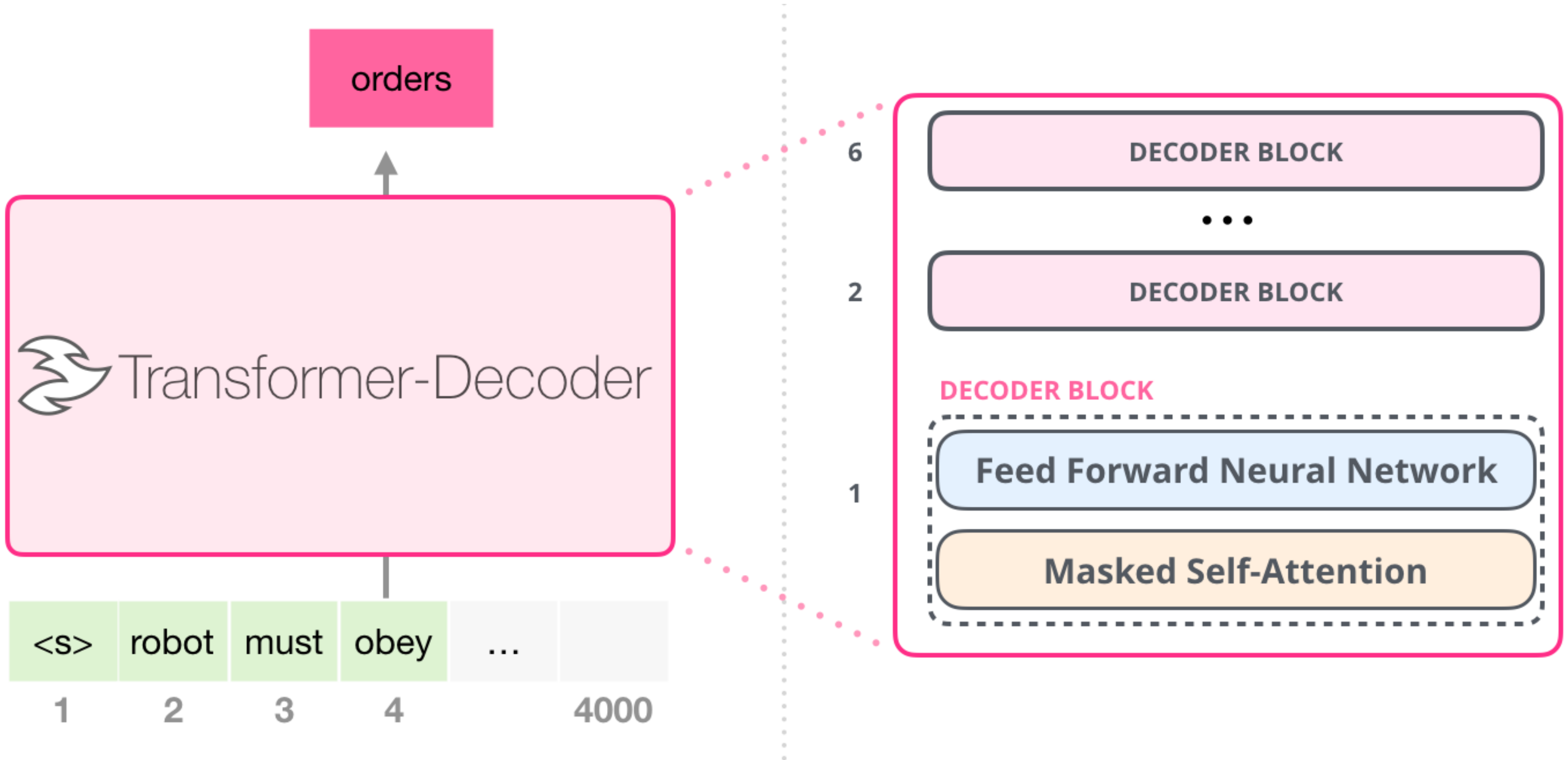
https://lena-voita.github.io/nlp_course/language_modeling.html

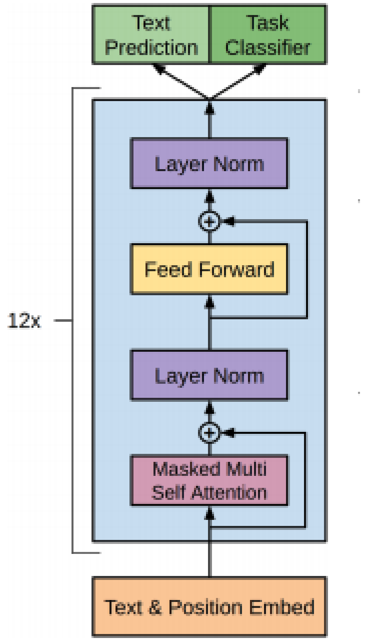
Next token prediction



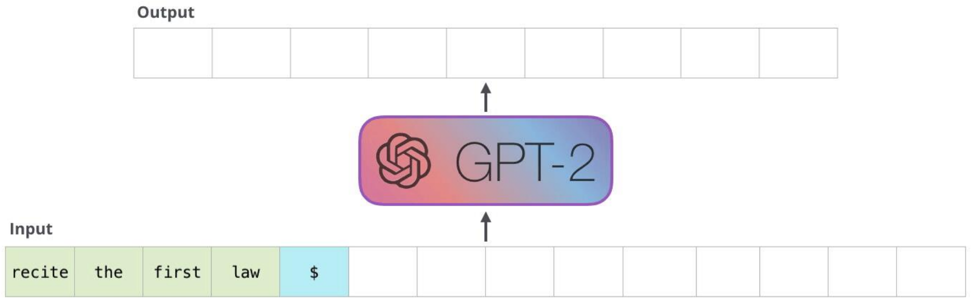
- Feed word embedding for previous (context) words into a network;
- get vector representation of context from the network;
- from this vector representation, predict a probability distribution for the next token

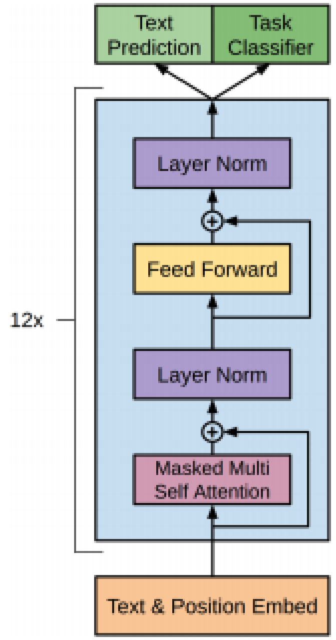
Transformer decoder

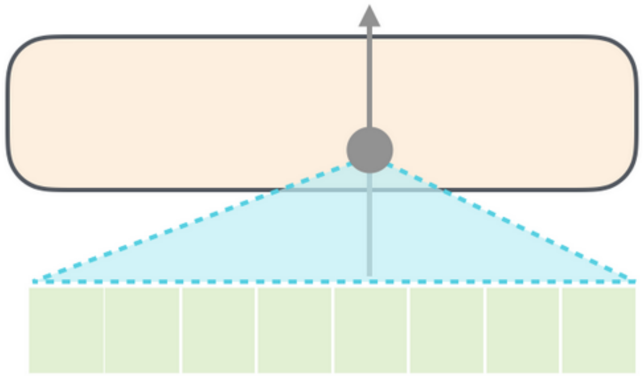


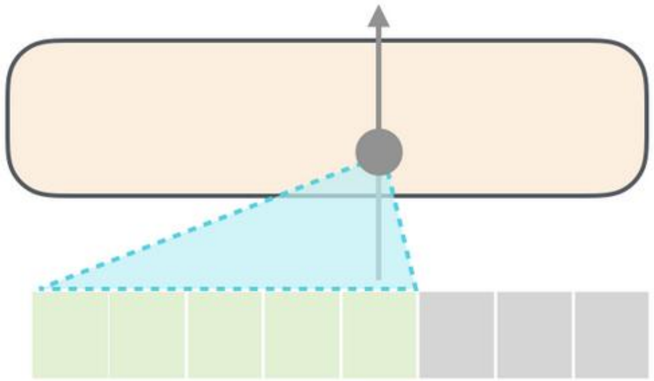


Masked Multi-Head Attention

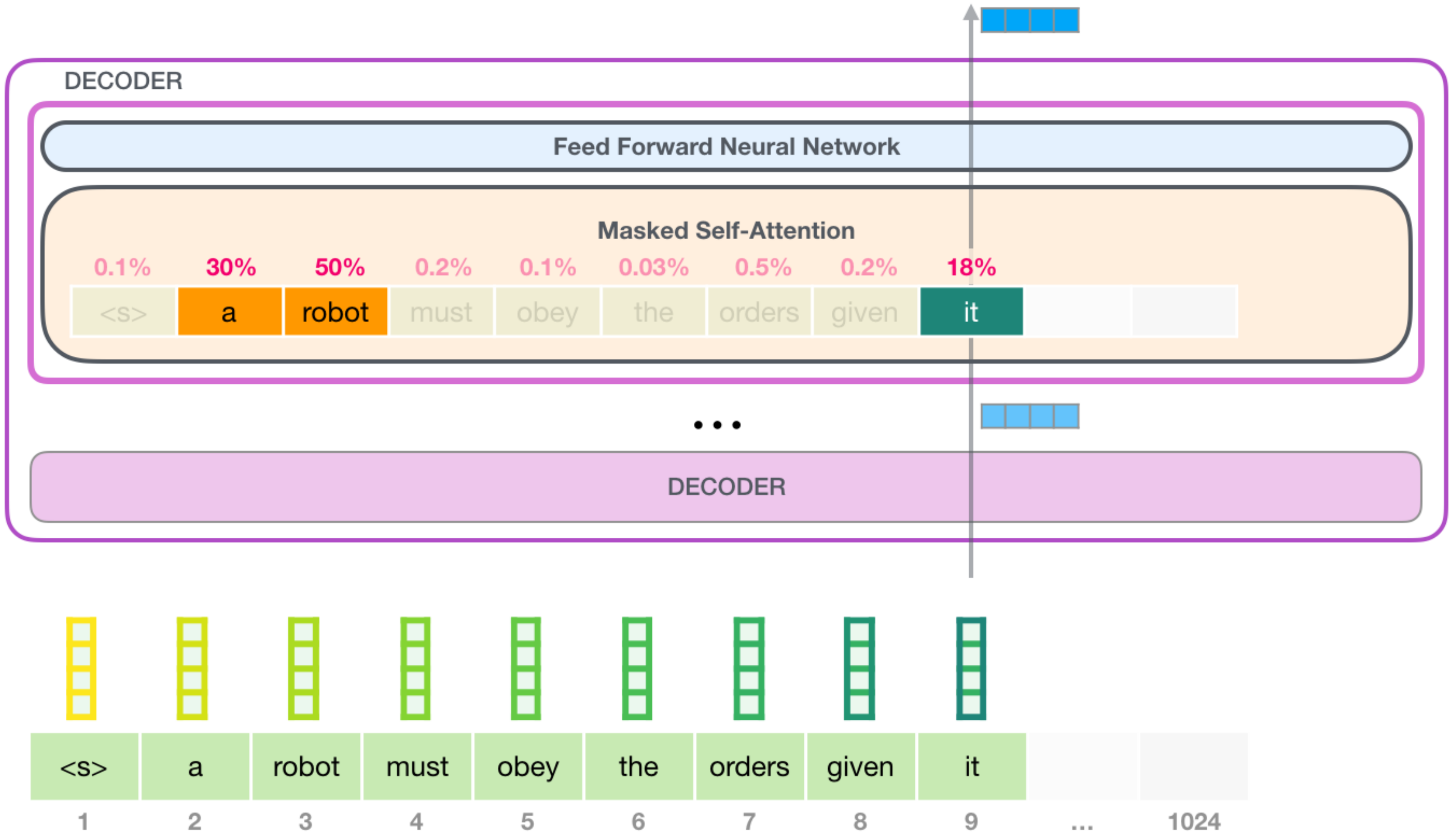








Masked Multi-Head Attention



Masked Multi-Head Attention

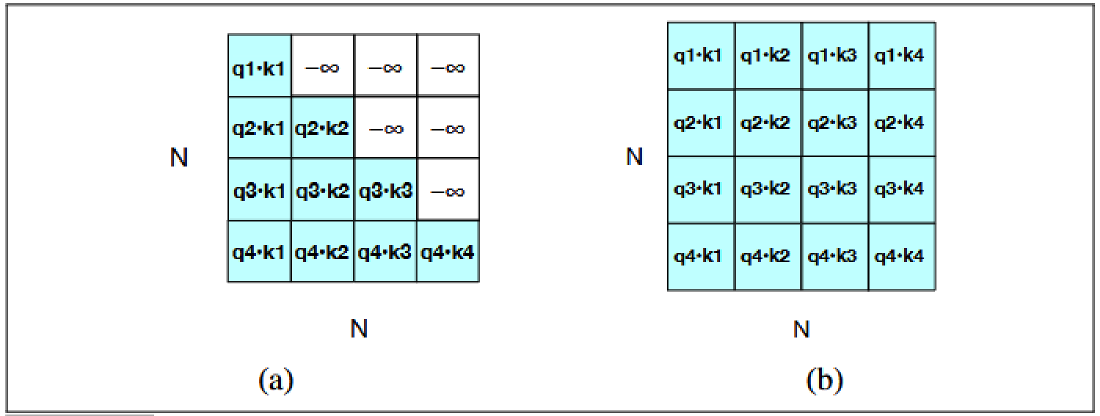
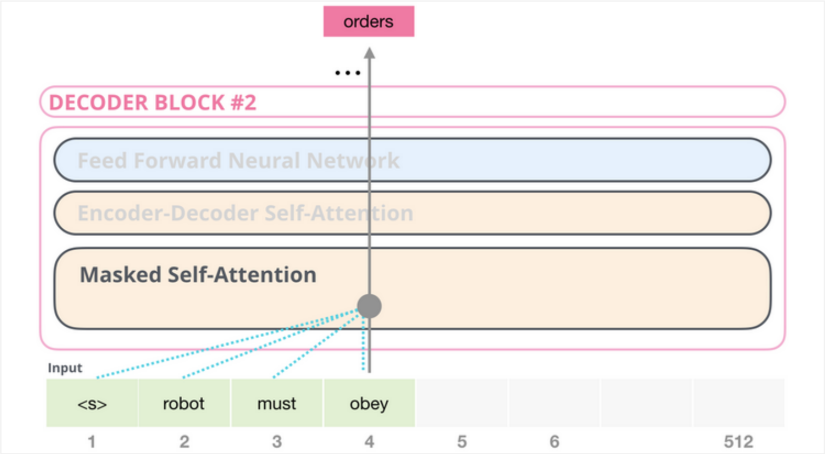
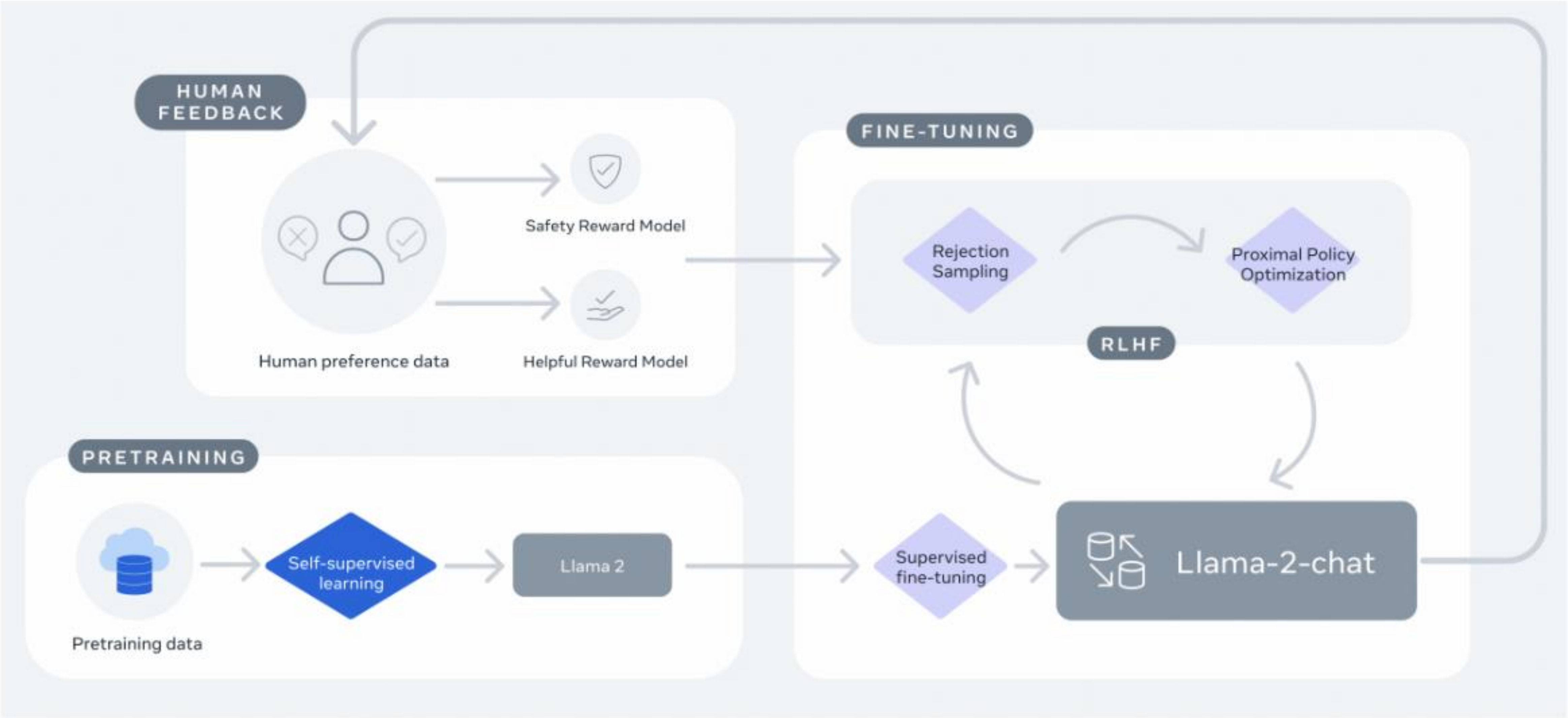


Figure 11.2 The $N \times N$ $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).



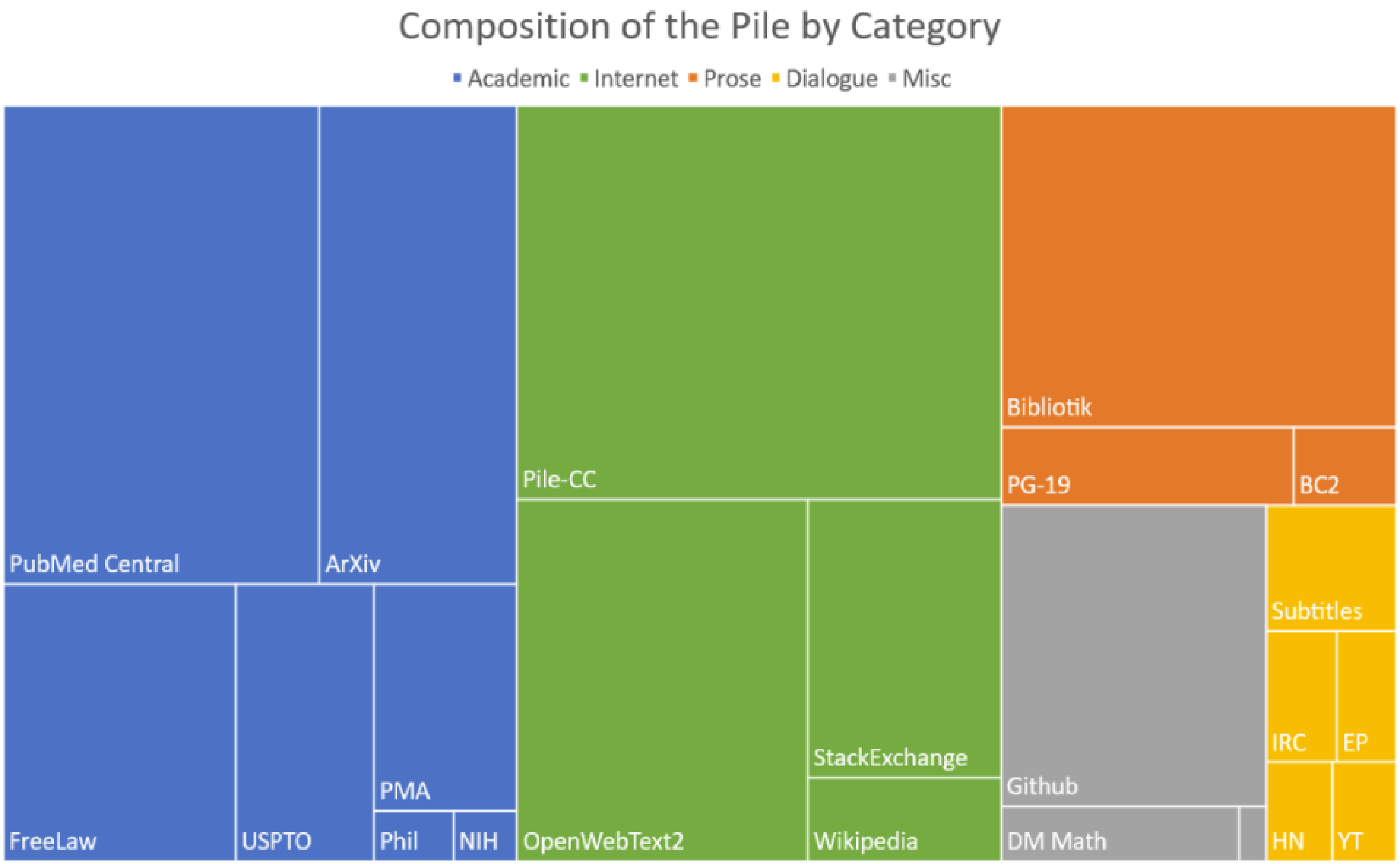
Overview of LLMs Training



Pre-training data

Pre-training data

Universidad EAFIT — Escuela de Ciencias Aplicadas e Ingeniería



Model	Training Data
BERT	BookCorpus, English Wikipedia
GPT-1	BookCorpus
GPT-3	CommonCrawl, WebText, English Wikipedia, and 2 book databases (“Books 1” and “Books 2”)
GPT-3.5+	Undisclosed

Pre-training data

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Input Sequence	Expected Output (Next Token)
"The"	"Eiffel"
"The Eiffel"	"Tower"
"The Eiffel Tower is"	"located"
"The Eiffel Tower is located in"	"the"
"The Eiffel Tower is located in the city of"	"Paris"

python

```python
def add_numbers(a, b):
    return a + b
```

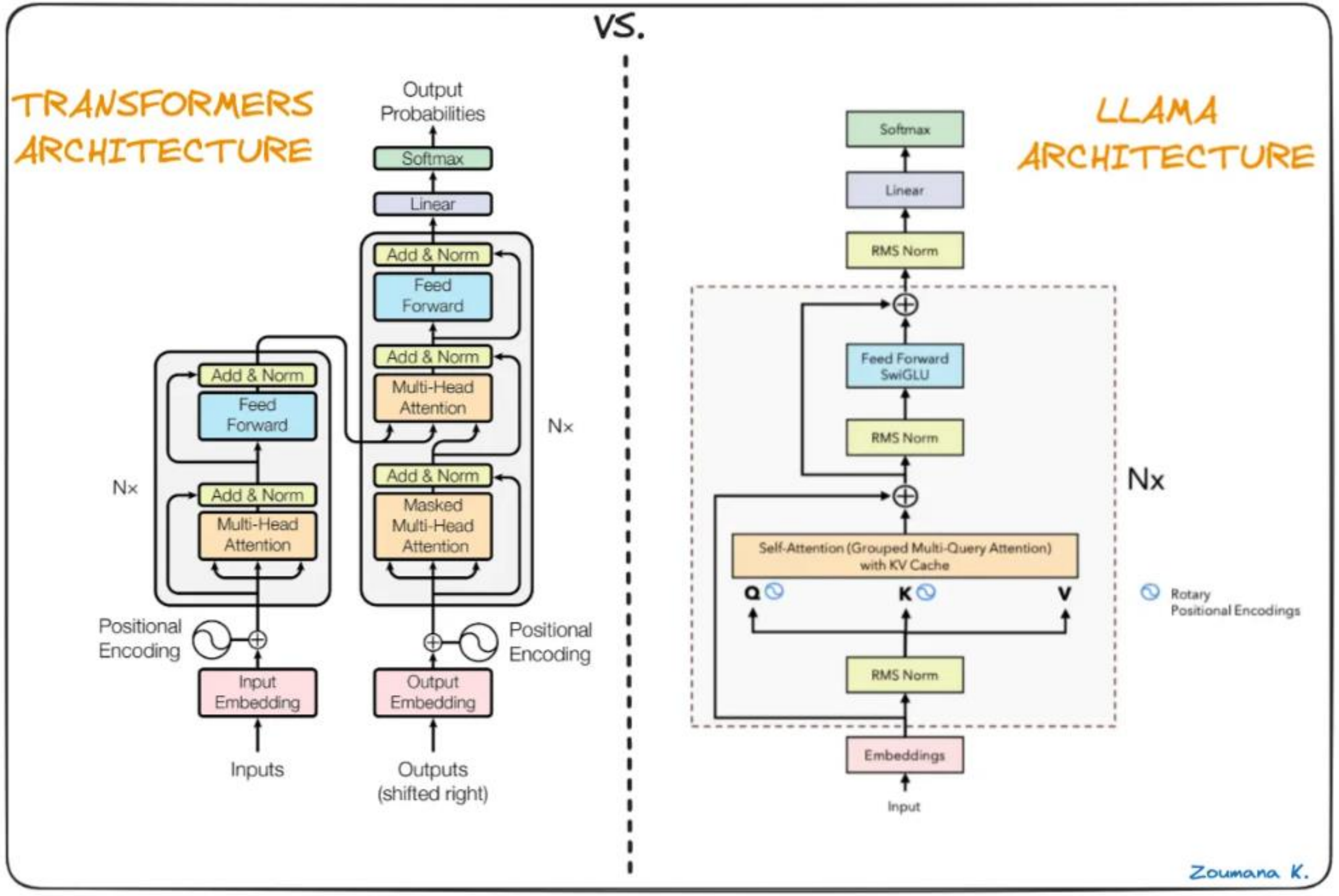
Input Sequence	Expected Output (Next Token)
"def"	"add_numbers"
"def add_numbers("	"a"
"def add_numbers(a,"	"b"
"return a"	"+"
"return a +"	"b"

Pre-training data – Llama 1

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

- 1.4 trillion tokens

Architecture



https://medium.com/@pranjalkhadka/llama-explained-a70e71e706e9

Grouped Query Attention

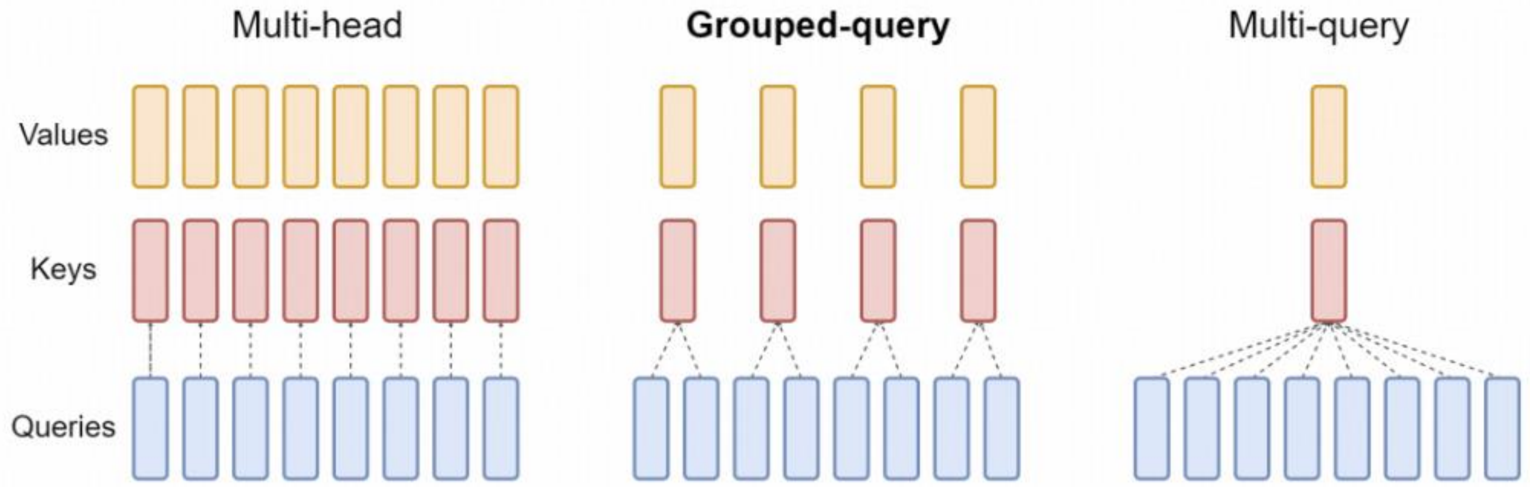


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

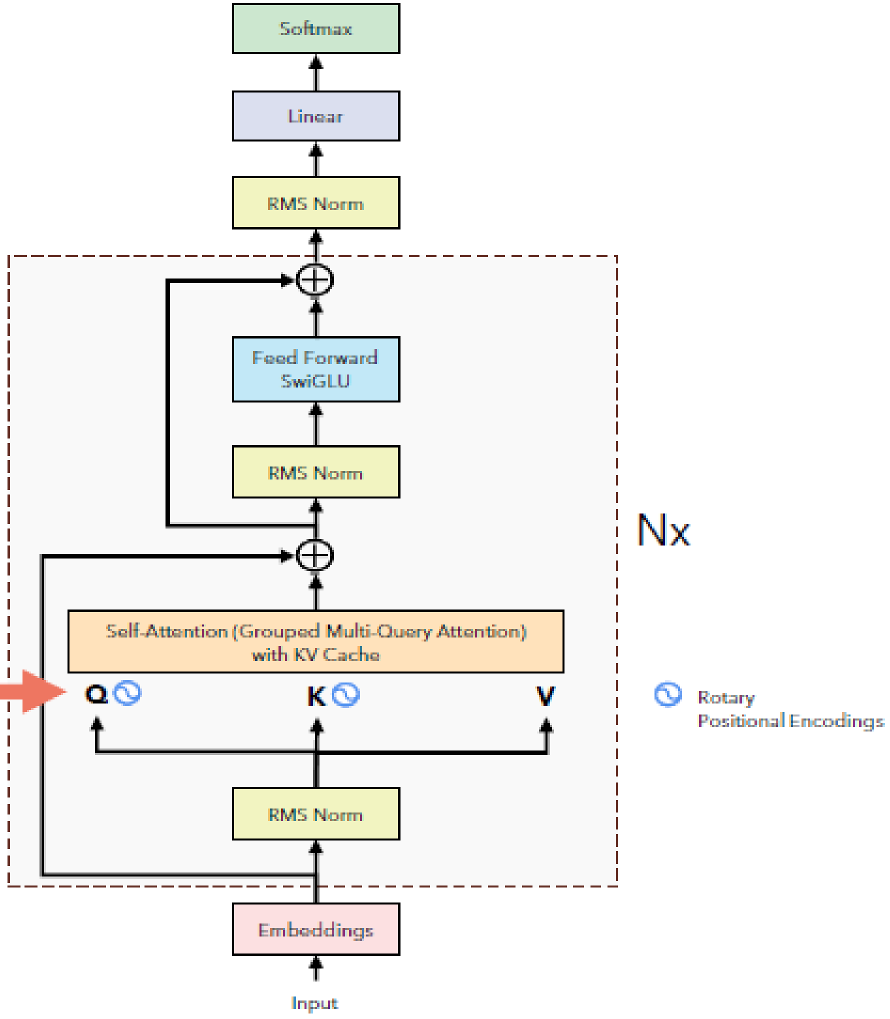
Model	**T_{infer}**	**Average**	**CNN**	**arXiv**	**PubMed**	**MediaSum**	**MultiNews**	**WMT**	**TriviaQA**
	s		**R_1**	**R_1**	**R_1**	**R_1**	**R_1**	**BLEU**	**F1**
MHA-Large	0.37	46.0	42.9	44.6	46.2	35.5	46.6	27.7	78.2
MHA-XXL	1.51	47.2	43.8	45.6	47.5	36.4	46.9	28.4	81.9
MQA-XXL	0.24	46.6	43.0	45.0	46.9	36.1	46.5	28.5	81.3
GQA-8-XXL	0.28	47.1	43.5	45.4	47.7	36.3	47.2	28.4	81.6

https://arxiv.org/pdf/2305.13245

Positional encodings

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- **Rotary Positional Encodings** apply only to **queries (q)** and **keys (k)**, not to values (v).
- In **rotary encodings**, they are applied **after** multiplying q and k by matrix **W**, whereas in a **vanilla transformer**, they are applied **before**.



LLaMA

Positional encodings

Escuela de
Ciencias Aplicadas
e Ingeniería

Self-Attention with Relative Position Representations

Peter Shaw
Google
petershaw@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

Ashish Vaswani
Google Brain
avaswani@google.com

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

Positional encodings

- The **dot product** used in the attention mechanism is a type of inner product, which can be considered a generalization of the dot product
- Can we find an inner product between the two vectors **q** (query) and **k** (key) used in the attention mechanism that depends only on the two vectors and the relative distance of the token they represent?

Under the case of $d = 2$, we consider two-word embedding vectors $\boldsymbol{x}_q$, $\boldsymbol{x}_k$ corresponds to query and key and their position m and n, respectively. According to eq. (1), their position-encoded counterparts are:

$$\begin{aligned} \boldsymbol{q}_m &= f_q(\boldsymbol{x}_q, m), \\ \boldsymbol{k}_n &= f_k(\boldsymbol{x}_k, n), \end{aligned} \tag{20}$$

where the subscripts of $\boldsymbol{q}_m$ and $\boldsymbol{k}_n$ indicate the encoded positions information. Assume that there exists a function g that defines the inner product between vectors produced by $f_{\{q,k\}}$:

$$\boldsymbol{q}_m^{\intercal}\boldsymbol{k}_n = \langle f_q(\boldsymbol{x}_m, m), f_k(\boldsymbol{x}_n, n)\rangle = g(\boldsymbol{x}_m, \boldsymbol{x}_n, n - m), \tag{21}$$

Positional encodings

- We can define a function **g** as follows, which depends only on the two embedding vectors **q** and **k** and their relative distance:

$$f_q(x_m, m) = (W_q x_m)\, e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n)\, e^{in\theta}$$

$$g(x_m, x_n, m-n) = \mathrm{Re}\left[(W_q x_m)\,(W_k x_n)^*\, e^{i(m-n)\theta}\right]$$

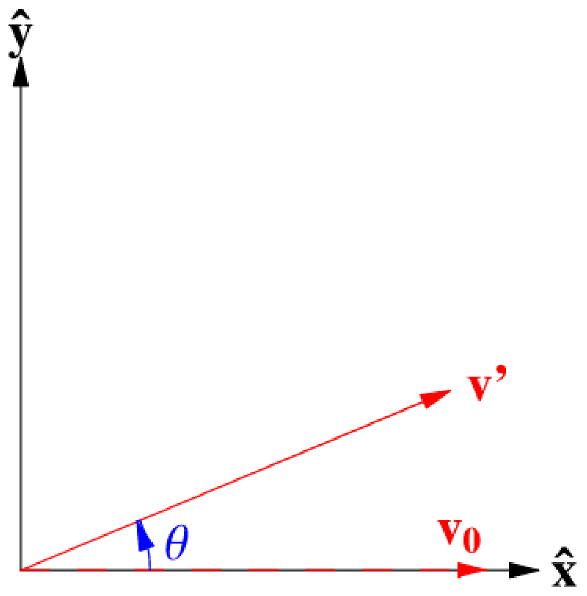
- Using euler:

$$f_{(q,k)}(x_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{q,k}^{(11)} & W_{q,k}^{(12)} \\ W_{q,k}^{(21)} & W_{q,k}^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

- Rotation matrix in a 2D space, hence the name **rotary positional encodings**.

Positional encodings

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



In $\mathbb{R}^2$, consider the matrix that rotates a given vector $\mathbf{v}_0$ by a counterclockwise angle θ in a fixed coordinate system. Then

$$\mathsf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{1}$$

so

$$\mathbf{v}' = \mathsf{R}_\theta \, \mathbf{v}_0. \tag{2}$$

Positional encodings

- Since the matrix is **sparse**, it is not convenient to use it for computing positional encodings.

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$. A graphic illustration of RoPE is shown in Figure (1). Applying our RoPE to self-attention in Equation (2), we obtain:

$$\boldsymbol{q}^\intercal_m \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n \tag{16}$$

where $\boldsymbol{R}^d_{\Theta,n-m} = (\boldsymbol{R}^d_{\Theta,m})^\intercal \boldsymbol{R}^d_{\Theta,n}$. Note that $\boldsymbol{R}^d_\Theta$ is an orthogonal matrix, which ensures stability during the process of encoding position information. In addition, due to the sparsity of R^d_Θ, applying matrix multiplication directly as in Equation (16) is not computationally efficient; we provide another realization in theoretical explanation.

Positional encodings

- Given a token with the embedding vector **x** and the token’s position **m** within the sentence, this is how we compute positional encodings for the token.

3.4.2 Computational efficient realization of rotary matrix multiplication

Taking the advantage of the sparsity of $\boldsymbol{R}^d_{\Theta,m}$ in Equation (15), a more computational efficient realization of a multiplication of R^d_Θ and $\boldsymbol{x} \in \mathbb{R}^d$ is:

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{34}$$

Positional encodings

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- **Upper bound** for the inner product decreases as token distance increases.
- **Rotary Positional Encodings** reduce interaction strength between distant tokens.

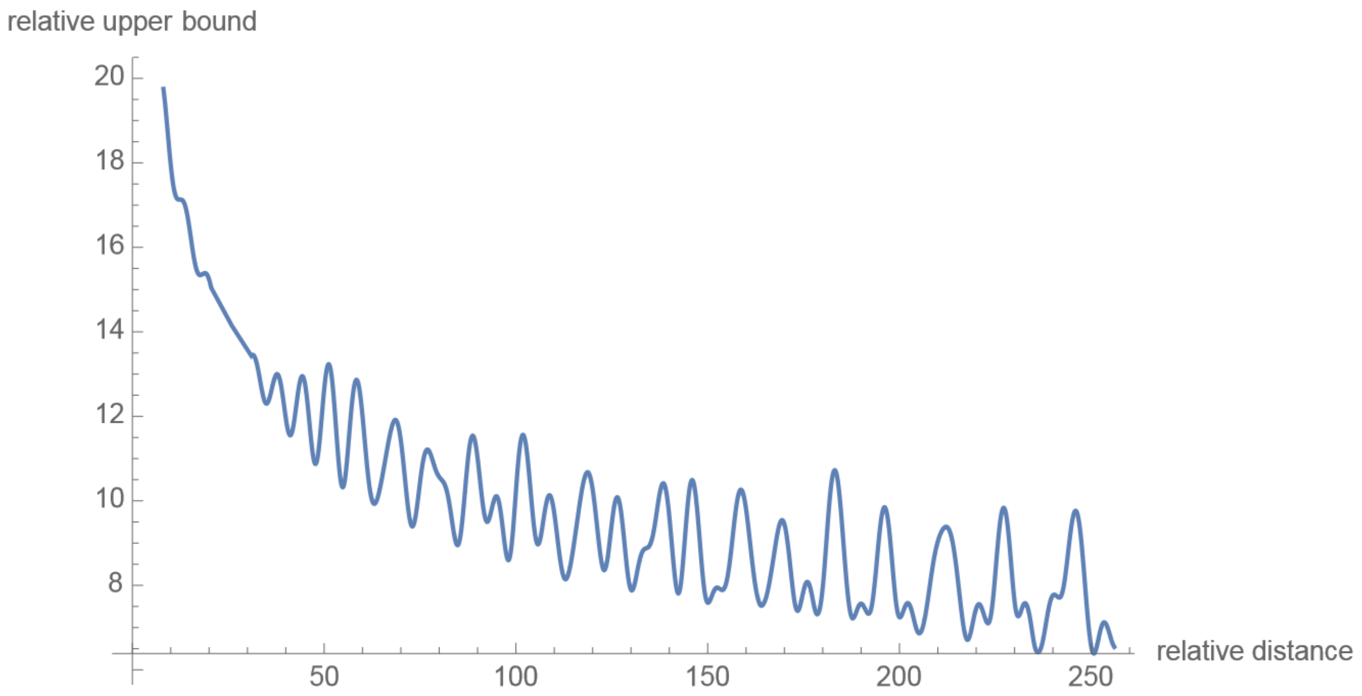
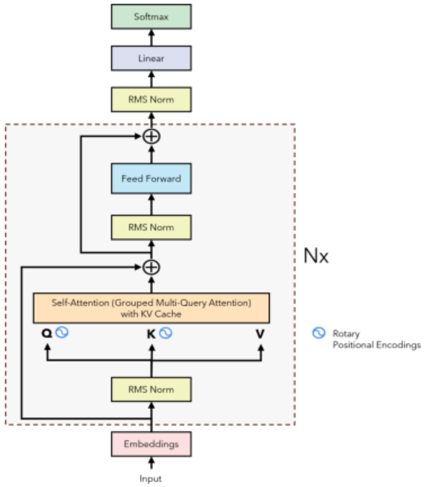


Figure 2: Long-term decay of RoPE.

KV Cache

Inference Optimization: We only need the **last token** at each step.
Challenge: The model still requires **all previous tokens** as context.
Solution: Use a **KV (Key-Value) Cache** to avoid redundant computations.

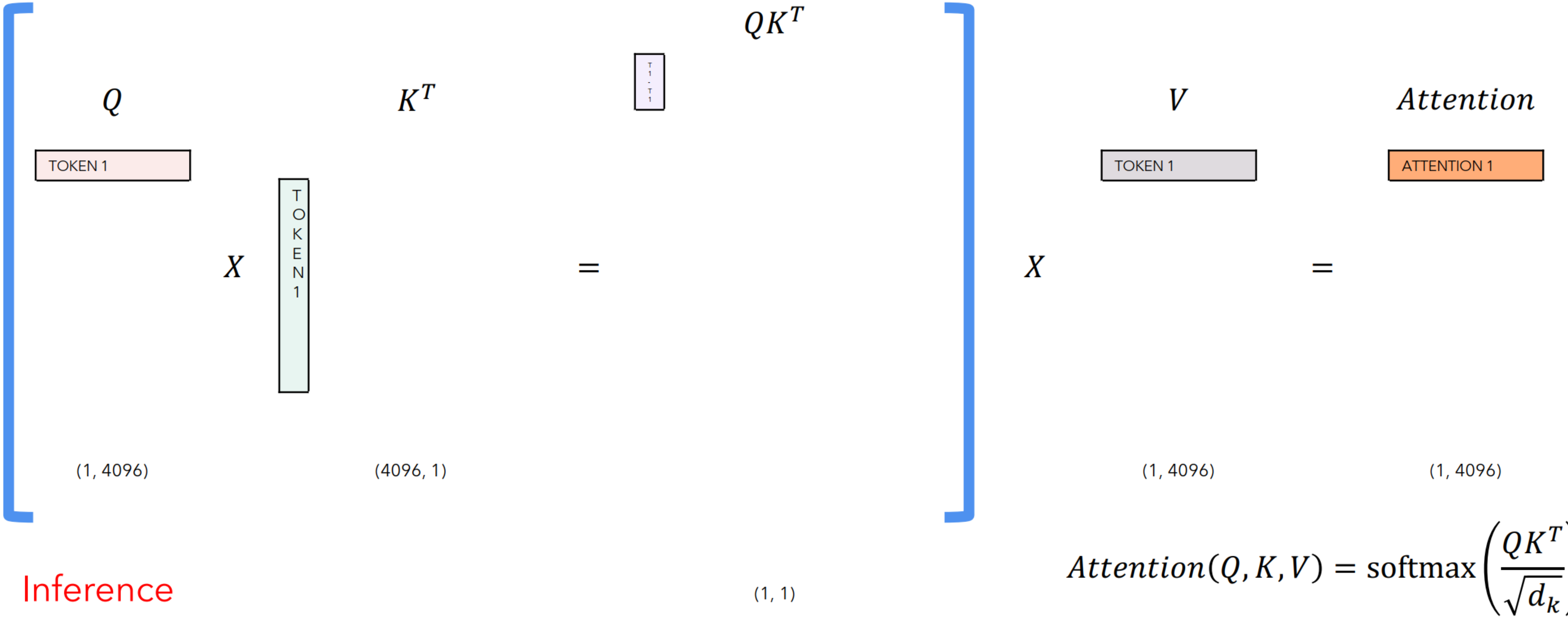
Output Love that can quickly seize the gentle heart [EOS]



Input [SOS] Love that can quickly seize the gentle heart

KV Cache - Motivation

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

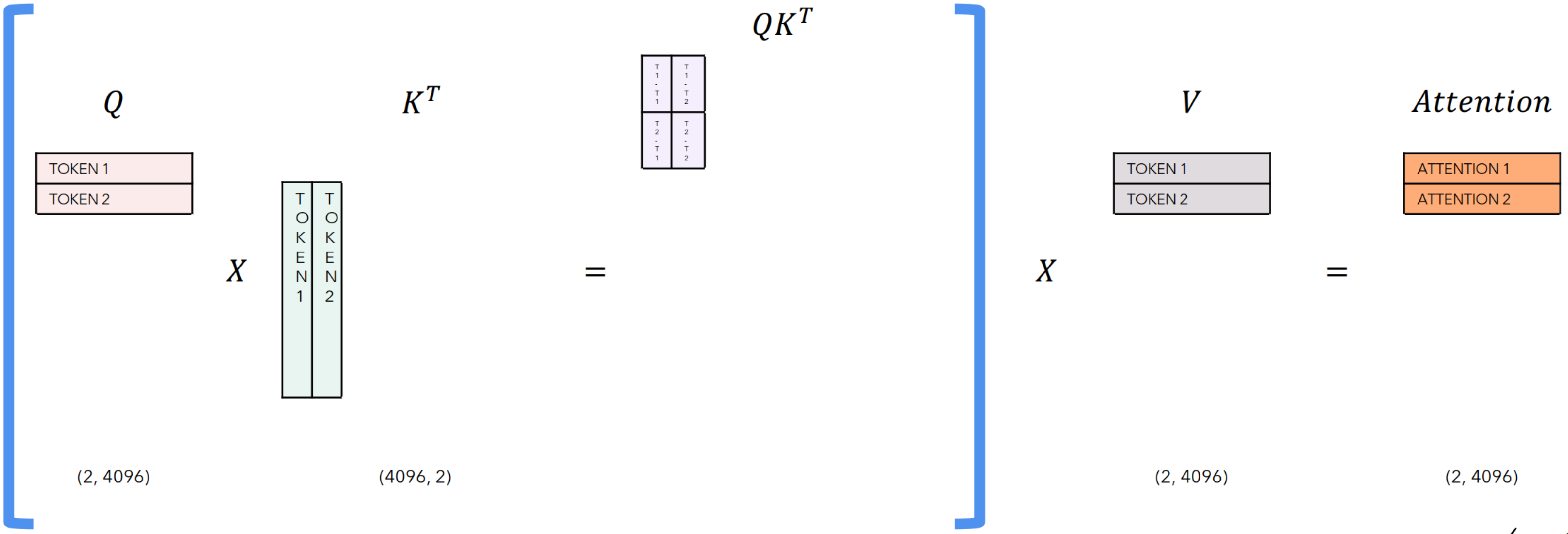


Inference
T = 1

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

KV Cache - Motivation

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



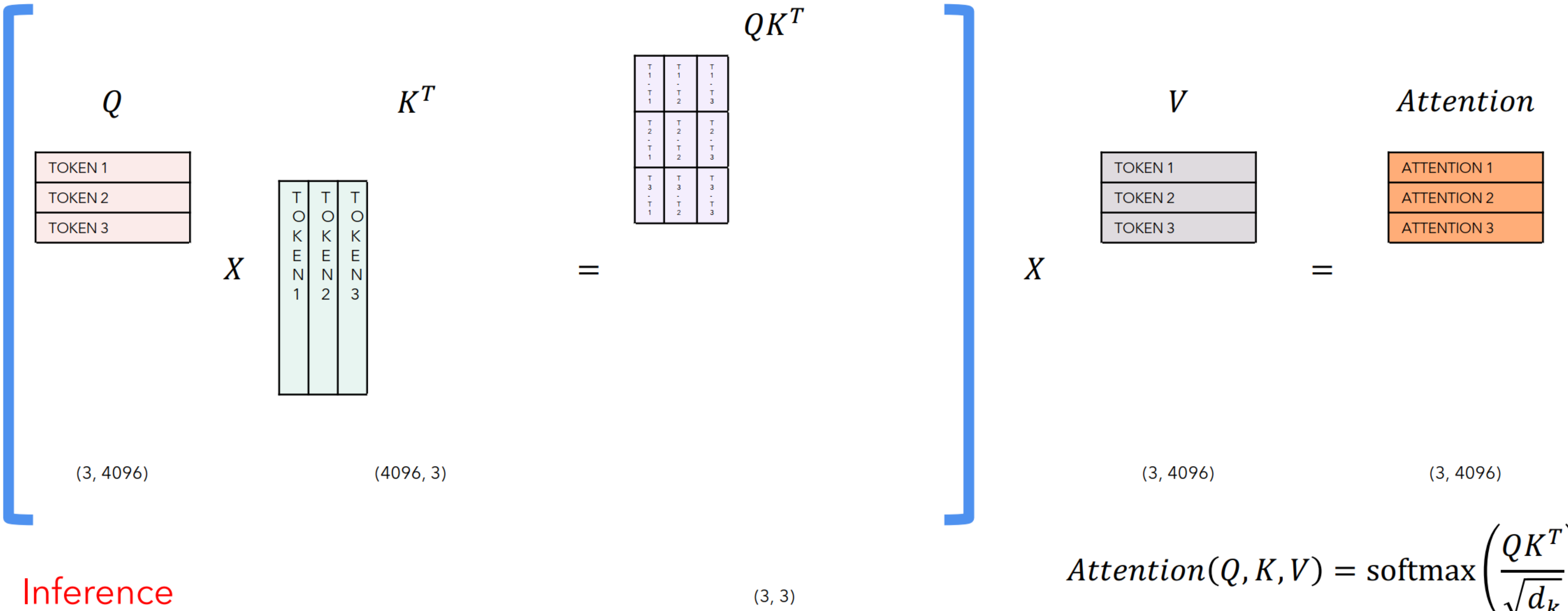
(2, 2)

Inference
T = 2

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

KV Cache - Motivation

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

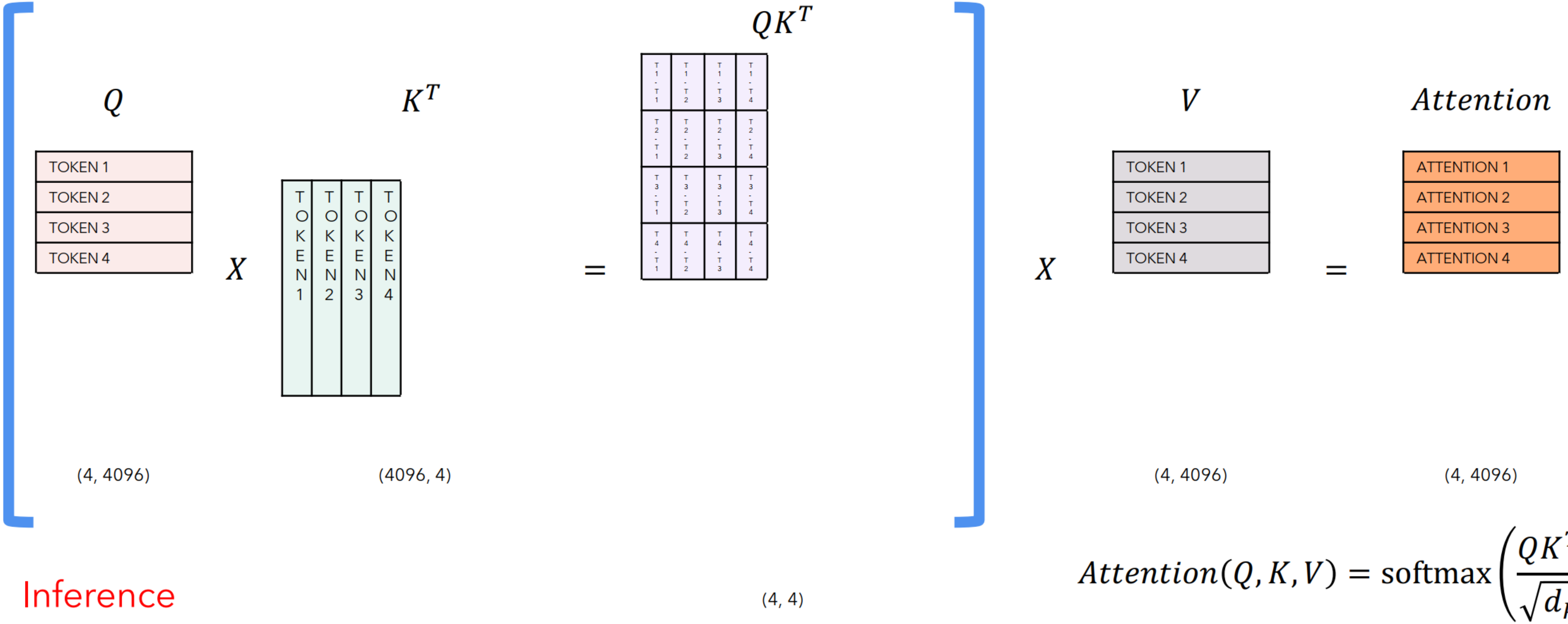


Inference
T = 3

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

KV Cache - Motivation

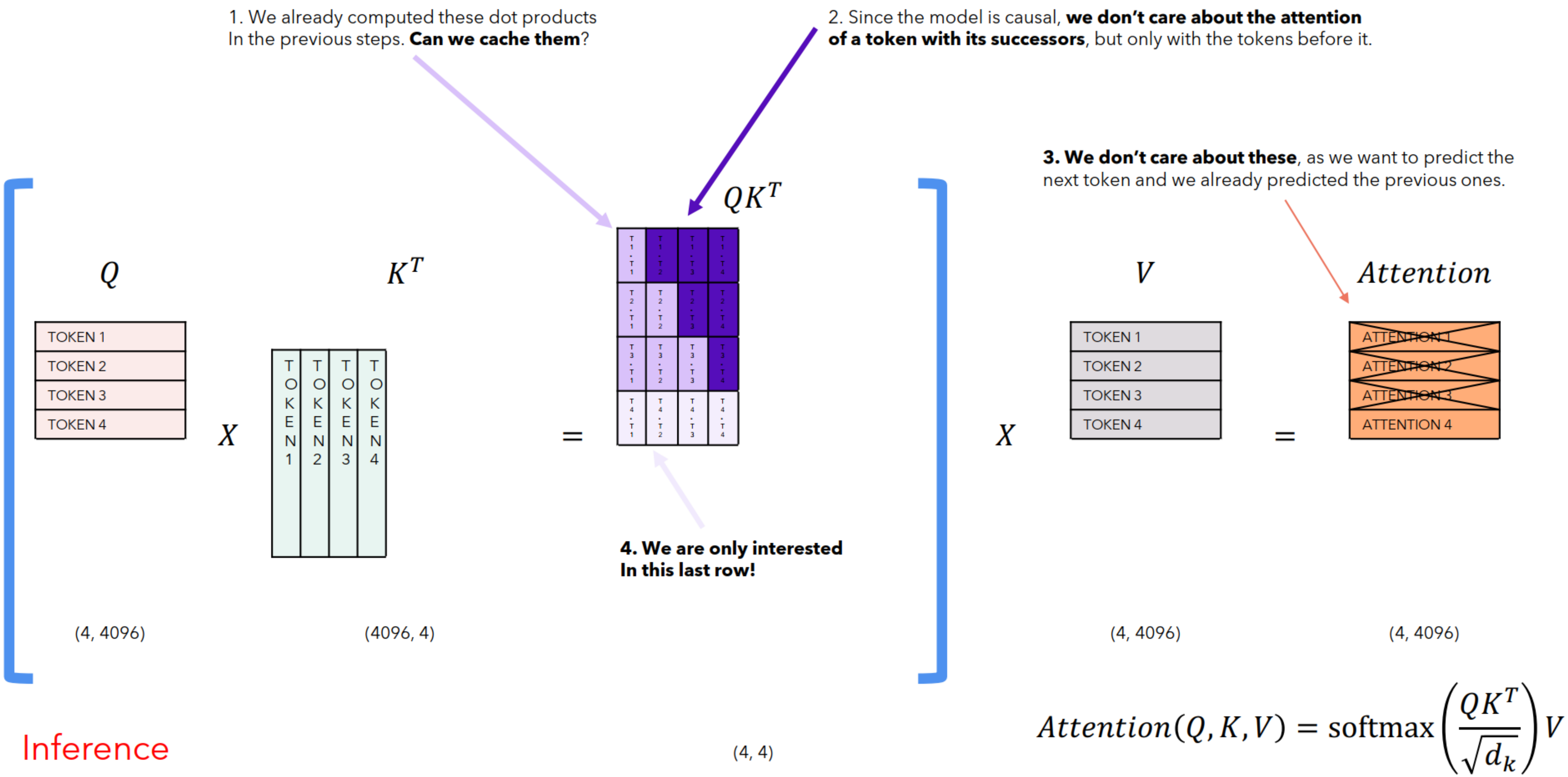
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



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

Inference
T = 4

KV Cache - Motivation



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

Inference
T = 4

Self Attention with KV Cache

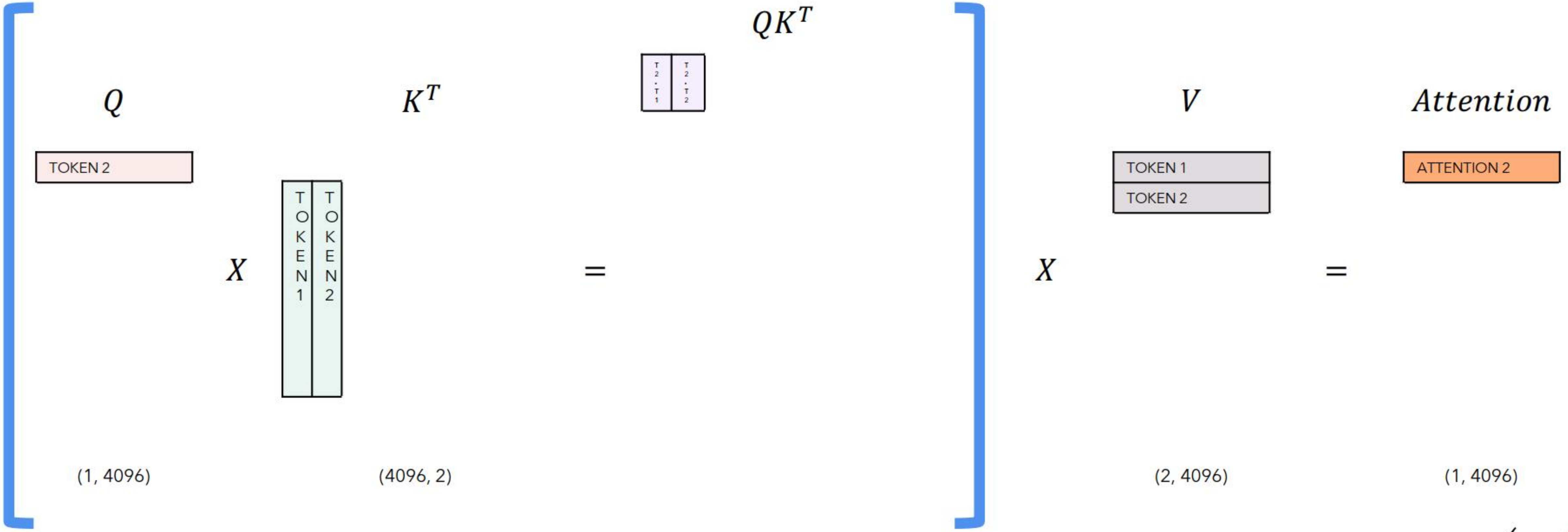
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

$$\left[\underset{(1,\,4096)}{\overset{Q}{\boxed{\text{TOKEN 1}}}} \; X \; \underset{(4096,\,1)}{\overset{K^T}{\boxed{\text{TOKEN 1}}}} \; = \; \underset{(1,\,1)}{\overset{QK^T}{\boxed{\text{T1.T1}}}} \right] \; X \; \underset{(1,\,4096)}{\overset{V}{\boxed{\text{TOKEN 1}}}} \; = \; \underset{(1,\,4096)}{\overset{Attention}{\boxed{\text{ATTENTION 1}}}}$$

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

Inference
T = 1

Self Attention with KV Cache



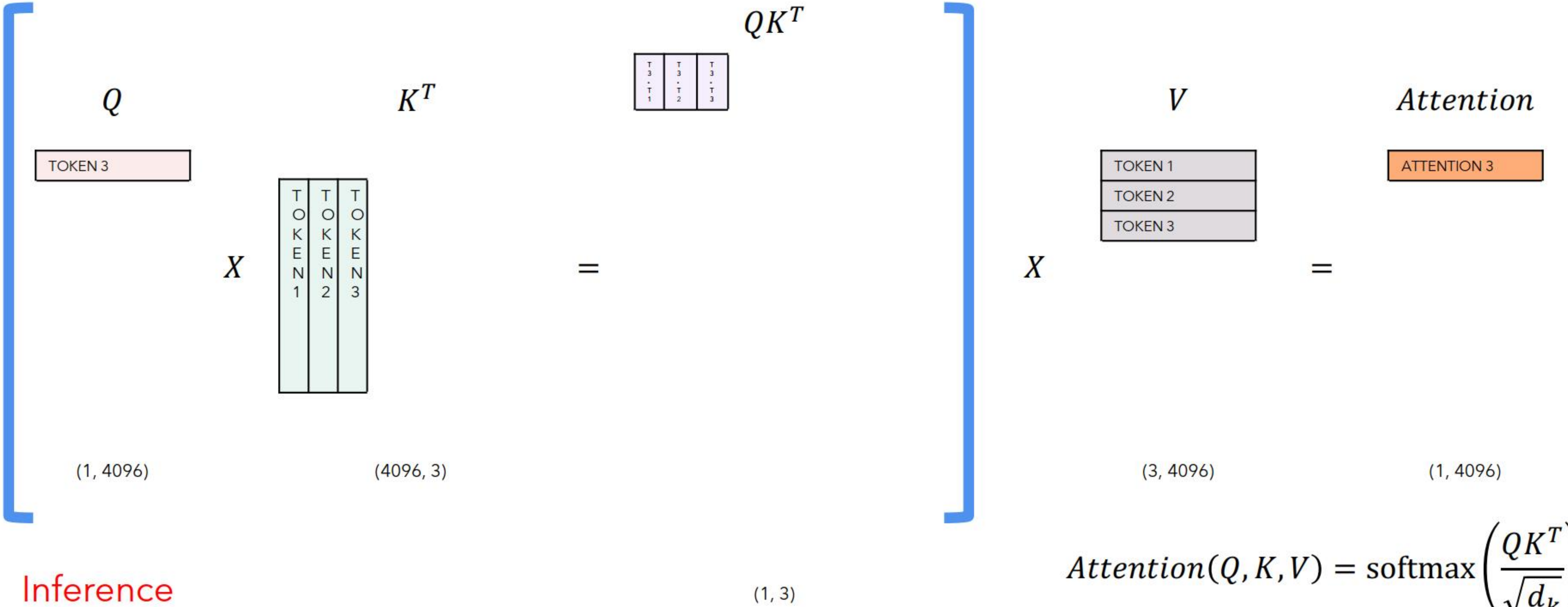
(1, 2)

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

Inference
T = 2

Self Attention with KV Cache

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

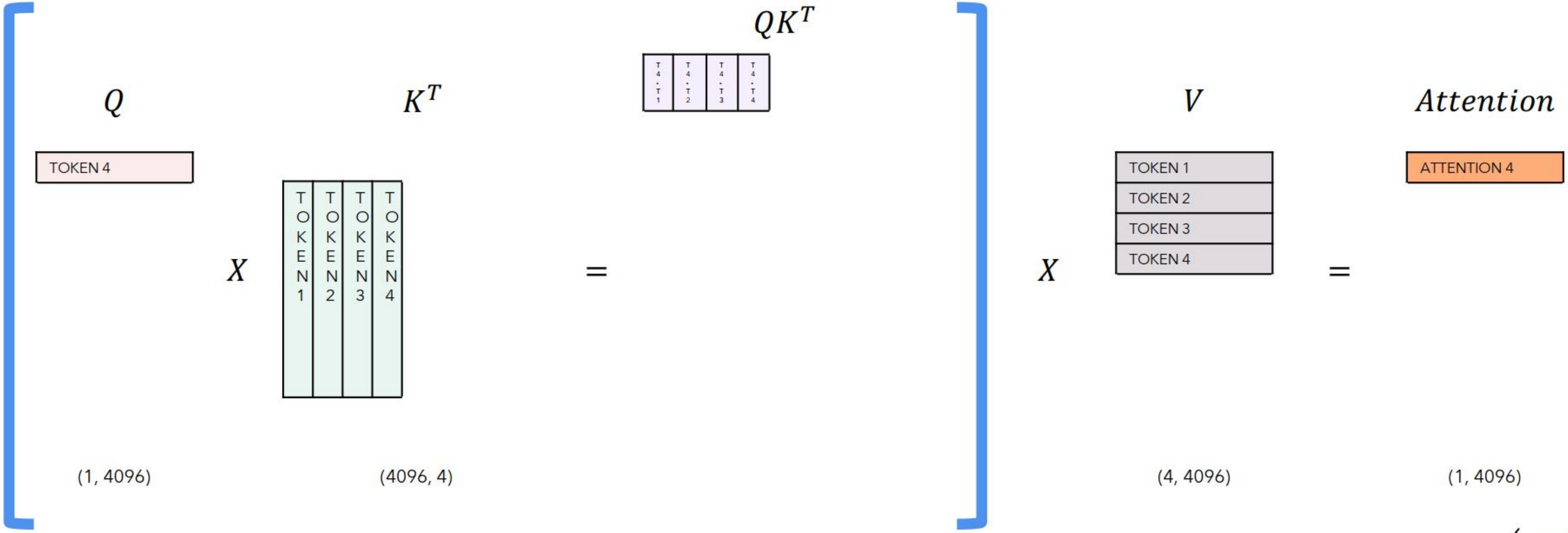


$$Attention(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Inference
T = 3

Self Attention with KV Cache

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



(1, 4)

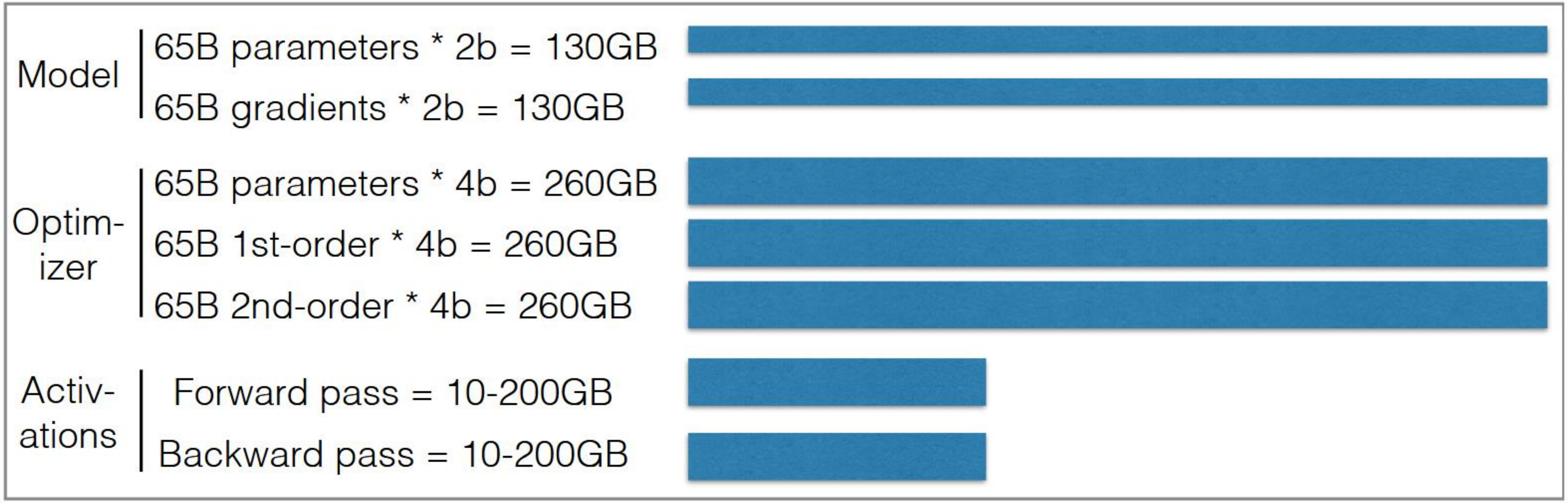
$$Attention(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Inference
T = 4

Fine-tuning

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

- **Continue training** the language model (LM) on the output.
- **Issue:** Optimization method can be **memory-intensive** depending on the optimizer.
- **Example:** Training a **65B parameter model** using **16-bit mixed precision (FP16)** (Rajbhandari et al., 2019).



1000-1400GB of GPU memory!

NLP tasks

Supervised Fine-Tuning (SFT) Data Examples

Summarization

📜 **Input:**

The Eiffel Tower, built in 1889, is one of the most famous landmarks in the world. Located in Paris, it stands at 330 meters tall and attracts millions of visitors each year.

✍️ **Output (Summary):**

The Eiffel Tower is a famous 330-meter-tall landmark in Paris, built in 1889.

Question Answering (QA)

📜 **Context:**

The Great Wall of China is a historic fortification built to protect Chinese states from invasions. Construction started as early as the 7th century BC and continued for centuries.

❓ **Question:** *When did the construction of the Great Wall begin?*

✅ **Answer:** *The 7th century BC.*

Instruction Following

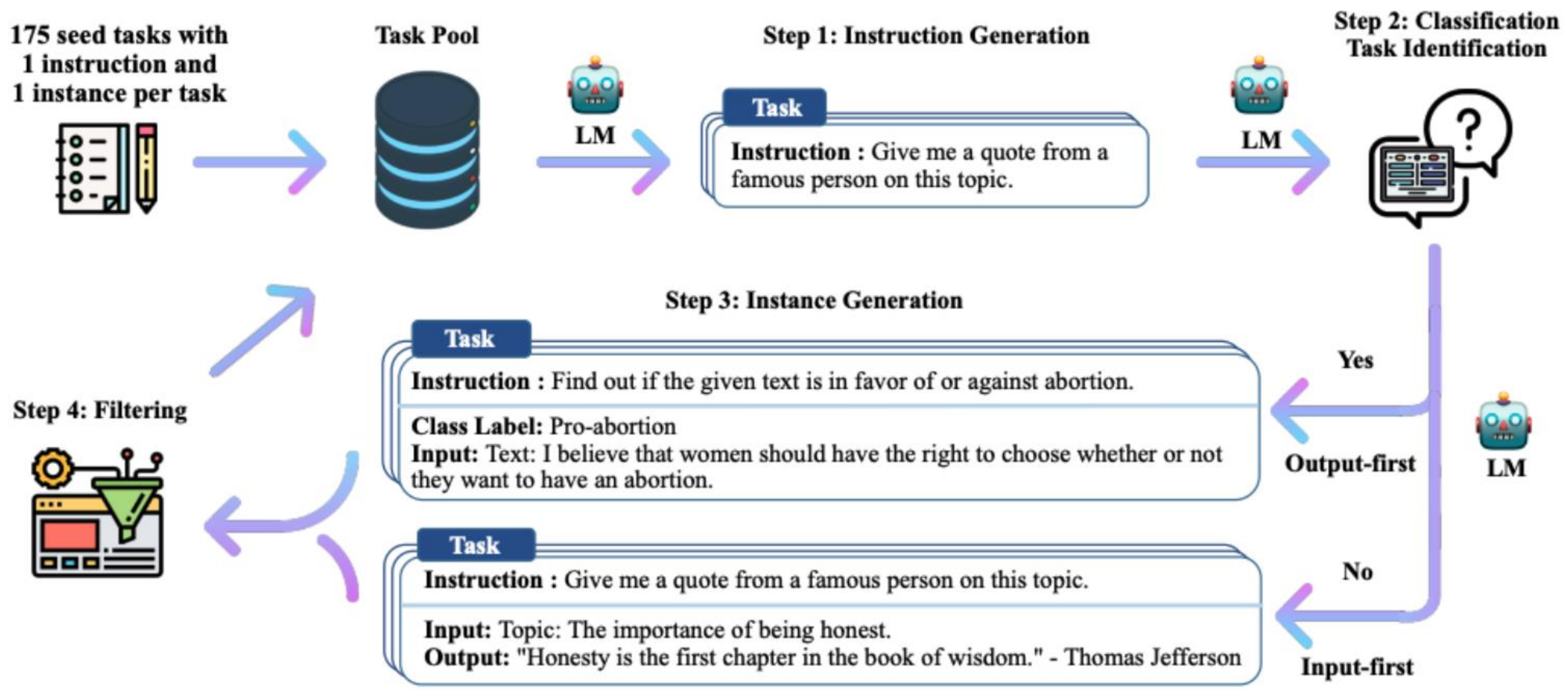
📜 **Instruction:**

Translate the following sentence into French: "The weather is beautiful today."

✅ **Response:** *Le temps est magnifique aujourd'hui.*

Self-Instruct

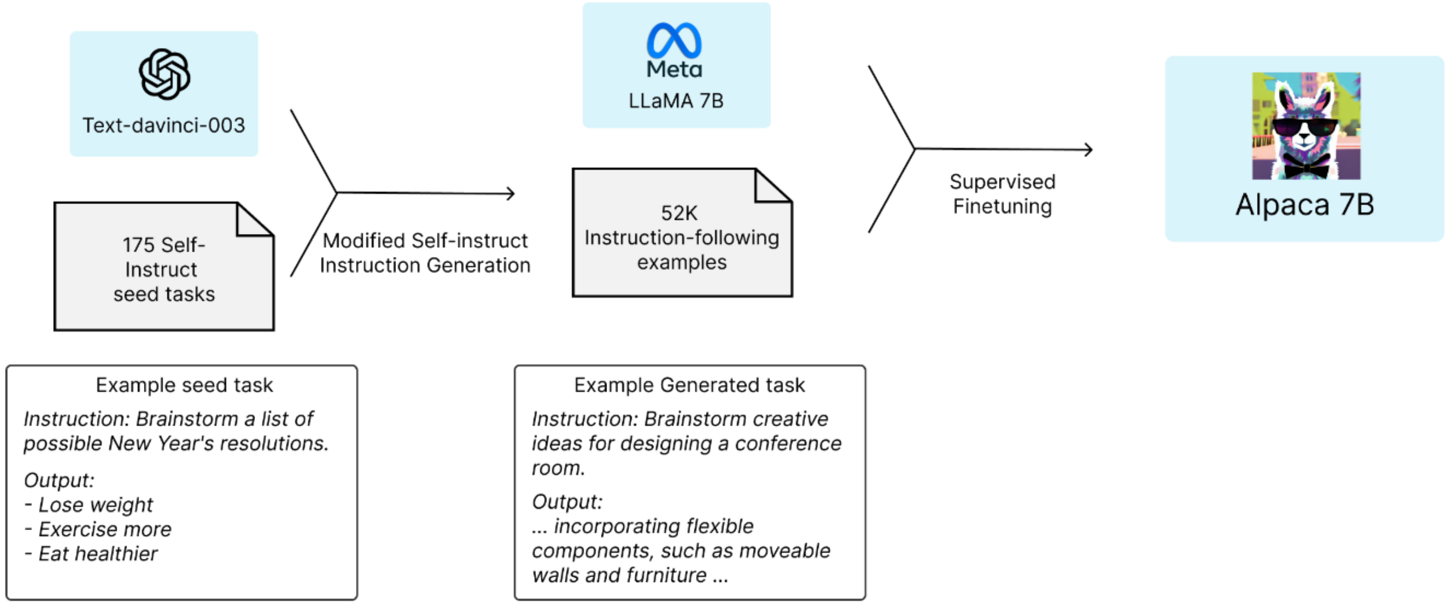
- It is possible to automatically generate instruction tuning datasets, e.g. self-instruct (Wang et al. 2022)



https://arxiv.org/abs/2212.10560

Alpaca

- Generating high-quality instruction tuning dataset with Self-Instruct using OpenAI text-devinci-003



Step 1: Start with LLaMA 7B

- Used **Meta's LLaMA 7B** as the **base model**.

Step 2: Generate Synthetic Data Using GPT-3.5

- Started with **175 human-written instruction examples**.
- Prompted **text-davinci-003** (OpenAI API) to **generate 52,000 diverse instructions & responses**.
- Cost of API calls: **Only ~$500 USD** 💰 (compared to millions for human annotation).

Step 3: Fine-Tune LLaMA on This Data

- The **synthetic instruction dataset** was used to fine-tune **LLaMA 7B**, turning it into **Alpaca**.

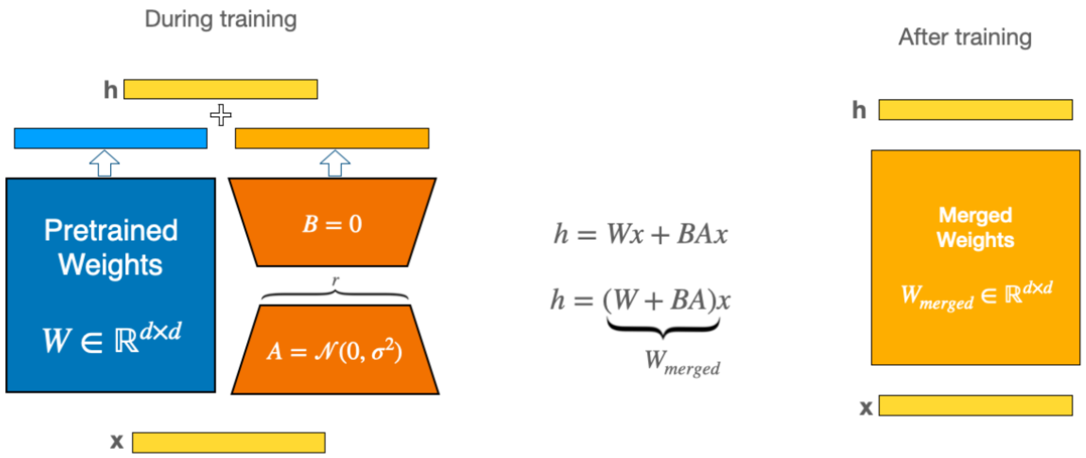
Step 4: Model Evaluation

- Researchers tested **Alpaca vs. text-davinci-003** on various tasks.
- **Alpaca exhibited similar instruction-following performance** but at a fraction of the cost.

Parameter Efficient Fine-tuning

Don’t tune all of the parameters, but just some!

- LoRA



LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

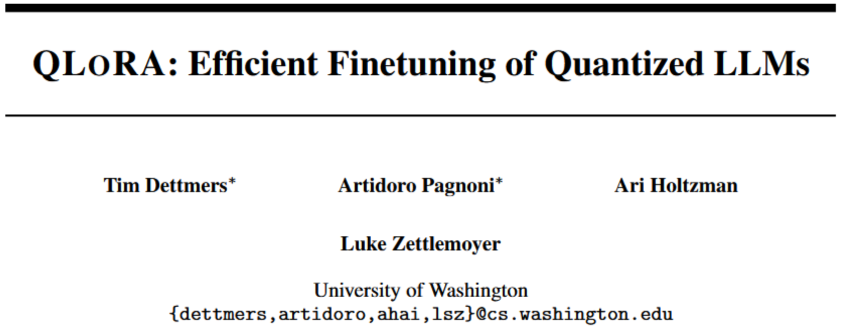
Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

- Freeze pre-trained weights, train low-rankapproximation of difference from pre-trained weights
- Advantage: after training, just add in to pre-trained weights – no new components!

Parameter Efficient Fine-tuning

Don’t tune all of the parameters, but just some!

- Q-LoRA

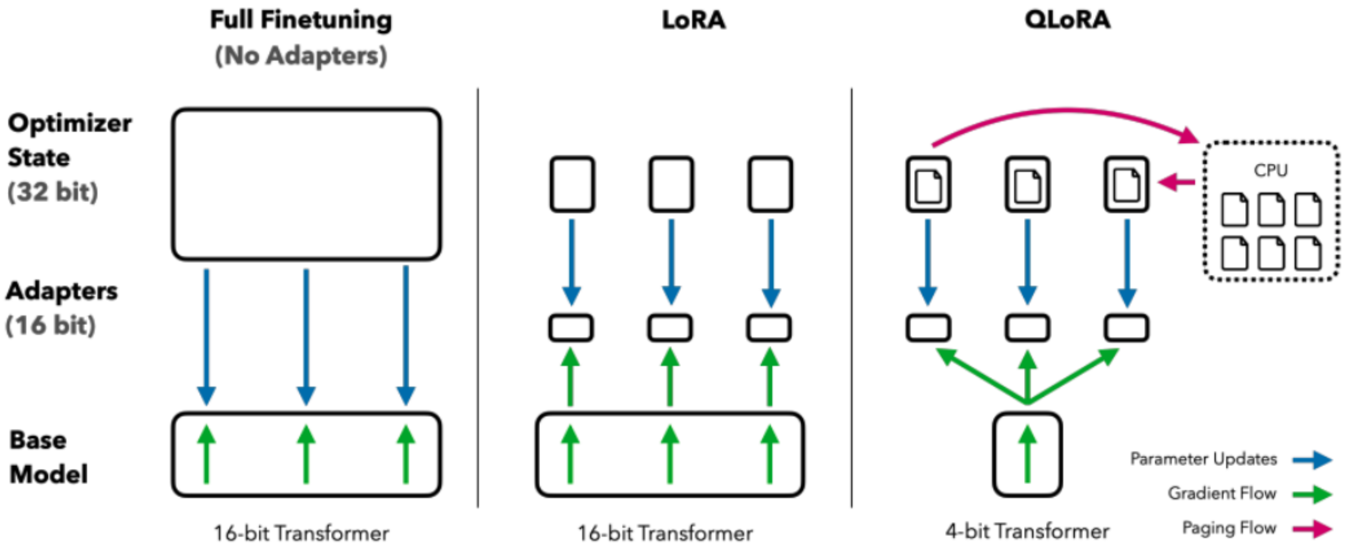


QLoRA: Efficient Finetuning of Quantized LLMs

Tim Dettmers* Artidoro Pagnoni* Ari Holtzman

Luke Zettlemoyer

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu



Further compress memory requirements for training by 4-bit quantization of the model

Quantization

Deep learning models, such as **LLaMA 2,** have **billions of parameters**, each stored as **floating-point numbers** (typically **FP32** or **FP16**). This leads to **huge memory requirements.**

•**Example:**

- LLaMA 2 **7B parameters**
- Each parameter stored as **FP32 (4 bytes per value)**
- Total memory needed:

$$\frac{7 \times 10^9 \times 32}{8 \times 10^9} = 28GB$$

Binary	Number
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

With 3 bits we can represent 2^3 distinct numbers

$$110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6$$

By reducing parameter precision (e.g., **FP32 → INT8**), we can **cut memory usage** dramatically, enabling deployment on **smaller GPUs** and **faster inference.**

Quantization

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Decimal numbers are just numbers that also include negative powers of the base. For example:

$$\mathbf{85.612 = 8 \times 10^{1} + 5 \times 10^{0} + 6 \times 10^{-1} + 1 \times 10^{-2} + 2 \times 10^{-3}}$$

The IEEE-754 standard defines the representation format for floating point numbers in 32 bit.

Sign | **Exponent (8 bit)** | **Fraction/Mantissa (23 bits)**

Sign = 0 (positive)

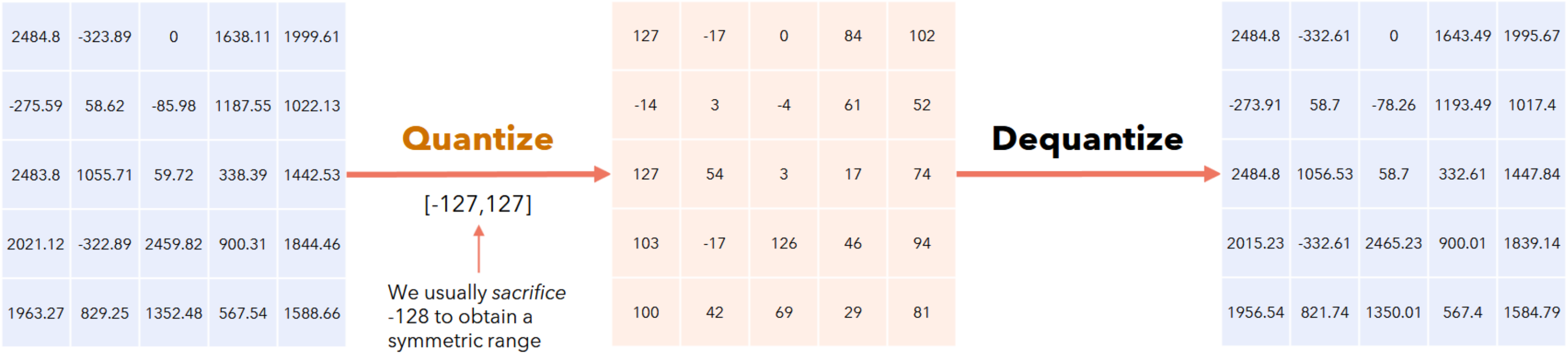
Exponent (E) = 124

Fraction = $1 + 1 \times 2^{-2} = 1.25$

$$\textbf{Value} = (-1)^{sign} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) = (+1) \times 2^{-3} \times 1.25 = +0.15625$$

Modern GPUs also support a 16-bit floating point number, with less precision.

Quantization



Step 1: Define Quantization Scale and Zero-Point

To convert FP32/FP16 values to INT8/INT4, we define:

Scale (S): A small floating-point number that helps reconstruct the original value.

Zero-point (Z): An integer shift to handle negative values.

$$q = \text{round}\left(\frac{x}{S} + Z\right)$$

- q = quantized integer value (e.g., INT8, INT4)
- x = original floating-point value
- S = scale factor
- Z = zero-point

FP16 Value (x)	Scale (S)	Zero-Point (Z)	Quantized INT8 (q)
0.45	0.1	128	132
-0.30	0.1	128	125
1.20	0.1	128	140

Quantization

Escuela de Ciencias Aplicadas e Ingeniería

2484.8	-323.89	0	1638.11	1999.61
-275.59	58.62	-85.98	1187.55	1022.13
2483.8	1055.71	59.72	338.39	1442.53
2021.12	-322.89	2459.82	900.31	1844.46
1963.27	829.25	1352.48	567.54	1588.66

Quantize → [-127,127]

We usually *sacrifice* -128 to obtain a symmetric range

127	-17	0	84	102
-14	3	-4	61	52
127	54	3	17	74
103	-17	126	46	94
100	42	69	29	81

Dequantize →

2484.8	-332.61	0	1643.49	1995.67
-273.91	58.7	-78.26	1193.49	1017.4
2484.8	1056.53	58.7	332.61	1447.84
2015.23	-332.61	2465.23	900.01	1839.14
1956.54	821.74	1350.01	567.4	1584.79

Step 2: Dequantization process

Since computation in hardware (e.g., matrix multiplications) happens in lower precision, **dequantization** is needed to recover approximate floating-point values.

$$x' = (q - Z) \times S$$

- x' = dequantized floating-point value (approximate x)
- q = quantized value (e.g., INT8)
- S = scale factor
- Z = zero-point

Quantized INT8 (q)	Scale (S)	Zero-Point (Z)	Dequantized FP16 (x')
132	0.1	128	0.4
125	0.1	128	-0.3
140	0.1	128	1.2