

EfficientML.ai Lecture 12

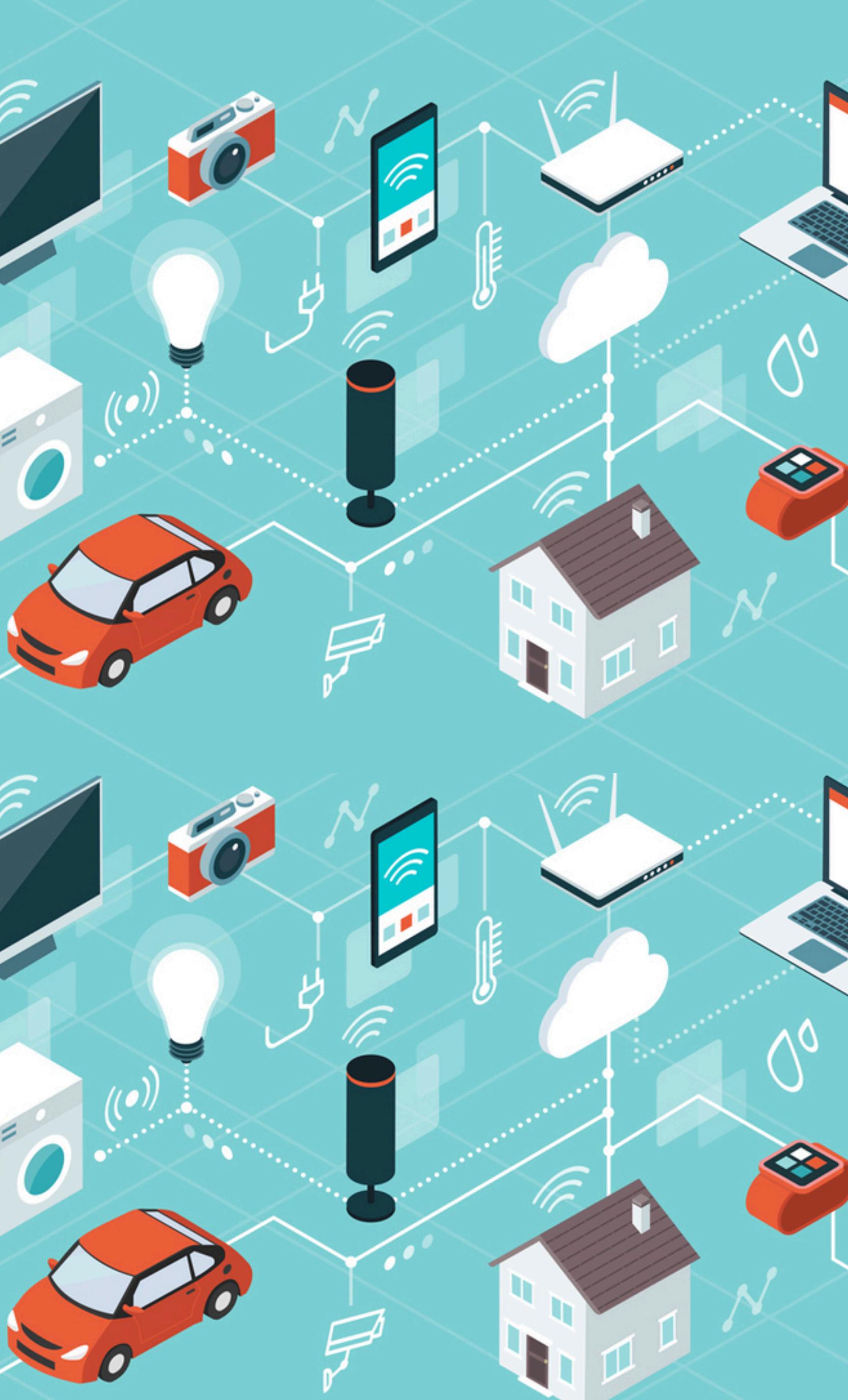
Transformer and LLM



Song Han

Associate Professor, MIT
Distinguished Scientist, NVIDIA

 @SongHan/MIT



Large Language Model

ChatGPT



GPT-4

JL

Please make a syllabus for a course titled "TinyML and Efficient Deep Learning Computing".



Course Syllabus: "TinyML and Efficient Deep Learning Computing"

Course Overview:

This course provides an in-depth exploration of the principles and practices of designing, developing, and deploying efficient deep learning models suitable for resource-constrained environments, with a special emphasis on TinyML (Tiny Machine Learning) applications for edge devices.

Course Objectives:

By the end of this course, students will be able to:

1. Understand the challenges and limitations of deploying deep learning models on edge and embedded devices.
2. Implement model quantization, pruning, and knowledge distillation techniques to optimize model size and computational efficiency.
3. Design, train, and deploy TinyML models tailored for specific edge computing applications.
4. Evaluate trade-offs between model accuracy, size, power consumption, and latency.

Prerequisites:

- Fundamental understanding of deep learning and neural networks.
- Basic programming knowledge in Python.
- Familiarity with TensorFlow or PyTorch.

Week-by-Week Breakdown:

Week 1: Introduction to TinyML and Edge Computing

- The landscape of edge computing
- Introduction to TinyML
- Challenges and opportunities in edge AI

Week 2: Efficient Neural Network Architectures

- MobileNets, EfficientNet, and other compact architectures
- Design principles for resource-efficient networks

Week 3: Model Compression Techniques I

- Quantization: Introduction, types, and benefits
- Post-training quantization vs. quantization-aware training

Week 4: Model Compression Techniques II

- Model pruning: Weight pruning, channel pruning, structured pruning
- Knowledge distillation: Basics and techniques

Week 5: Hardware Considerations for TinyML

- Overview of edge AI hardware
- Hardware accelerators and NPUs
- Understanding latency, throughput, and power trade-offs

Week 6: TinyML Frameworks and Tools

- Introduction to TensorFlow Lite (TFLite)

Image credit: <https://www.gorgias.com/blog/chatbot-vs-live-chat>

Large Language Model

Google Translate

The screenshot shows the Google Translate interface comparing English and Chinese (Simplified) text. The English text is a detailed course description about deploying neural networks to mobile devices. The Chinese translation is a summary of the course's goals and topics, including efficient inference, model compression, and quantum machine learning. Both versions include a 'Show more' link at the bottom.

ENGLISH - DETECTED ENGLISH SPANISH FRENCH CHINESE (SIMPLIFIED) ENGLISH SPANISH

Have you found it difficult to deploy neural networks on mobile devices and IoT devices? Have you ever found it too slow to train neural networks? This course is a deep dive into efficient machine learning techniques that enable powerful deep learning applications on resource-constrained devices. Topics cover efficient inference techniques, including model compression, pruning, quantization, neural architecture search, distillation; and efficient training techniques, including gradient compression and on-device transfer learning; followed by application-specific model optimization techniques for videos, point cloud and NLP; and efficient quantum machine learning. Students will get hands-on experience implementing deep learning applications on microcontrollers, mobile phones and quantum machines with an open-ended design project related to mobile AI.

您是否发现很难在移动设备和物联网设备上部署神经网络？你有没有发现训练神经网络太慢了？本课程深入探讨有效的机器学习技术，这些技术可在资源受限的设备上实现强大的深度学习应用。主题涵盖高效推理技术，包括模型压缩、剪枝、量化、神经架构搜索、蒸馏；和高效的训练技术，包括梯度压缩和设备迁移学习；其次是针对视频、点云和 NLP 的特定应用模型优化技术；和高效的量子机器学习。学生将通过与移动 AI 相关的开放式设计项目获得在微控制器、手机和量子机器上实施深度学习应用程序的实践经验。

Nín shìfǒu fāxiàn hěn nán zài yídòng shèbèi hé wù liánwǎng shèbèi shàng bùshǔ shénjīng wǎngluò? Nǐ yǒu méiyǒu fāxiàn xùnliàn shénjīng wǎngluò tài mànle? Běn kèchéng shēnrù tantǎo yóuxiào de jīqì xuéxí jìshù, zhèxiē jìshù kě zài zīyuán shòu xiàn de

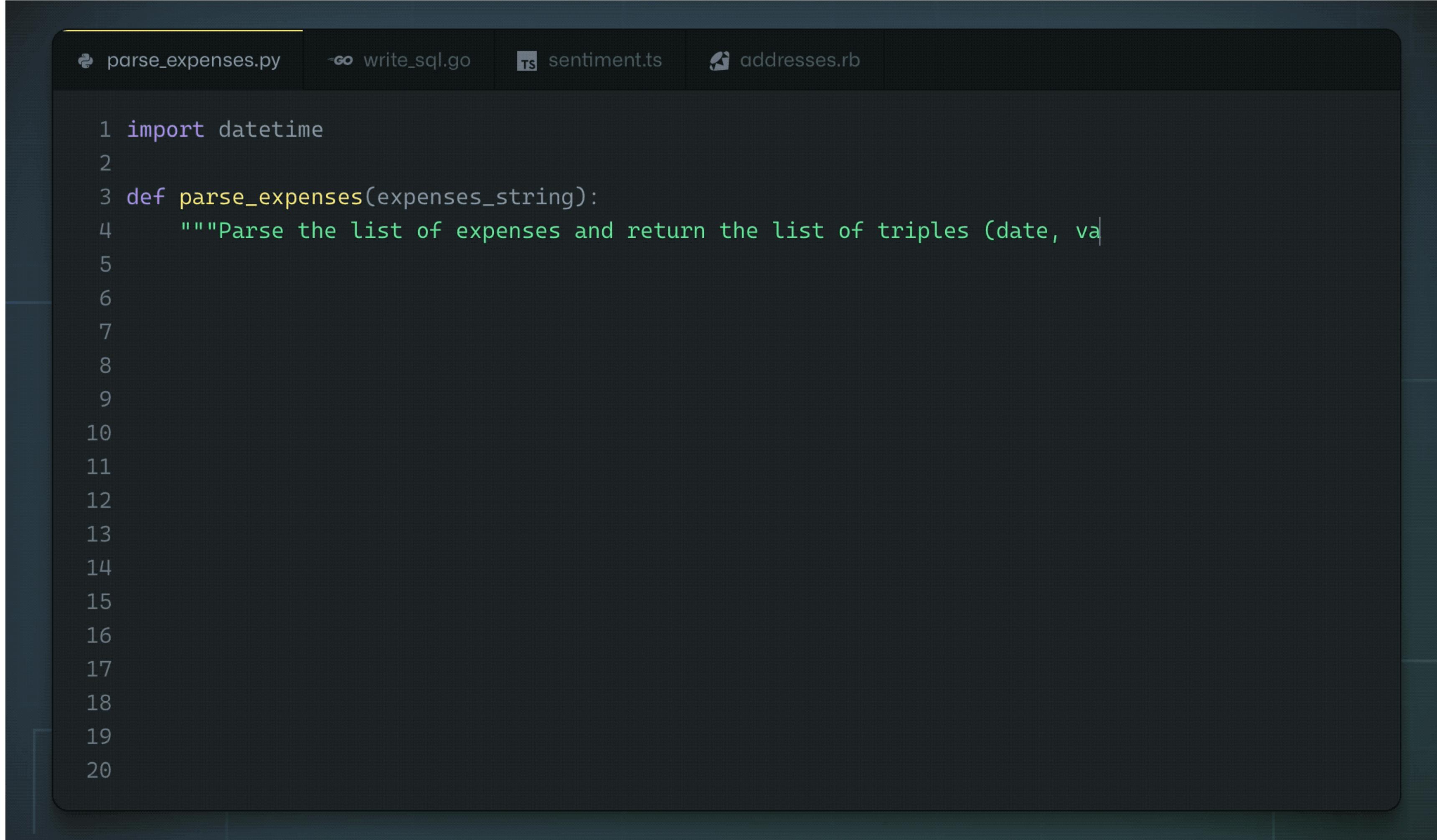
Show more

862 / 5,000

<https://translate.google.com/>

Large Language Model

GitHub Copilot



The screenshot shows a dark-themed code editor interface for GitHub Copilot. At the top, there are four tabs: 'parse_expenses.py' (selected), 'write_sql.go', 'sentiment.ts', and 'addresses.rb'. The main code editor area displays the following Python code:

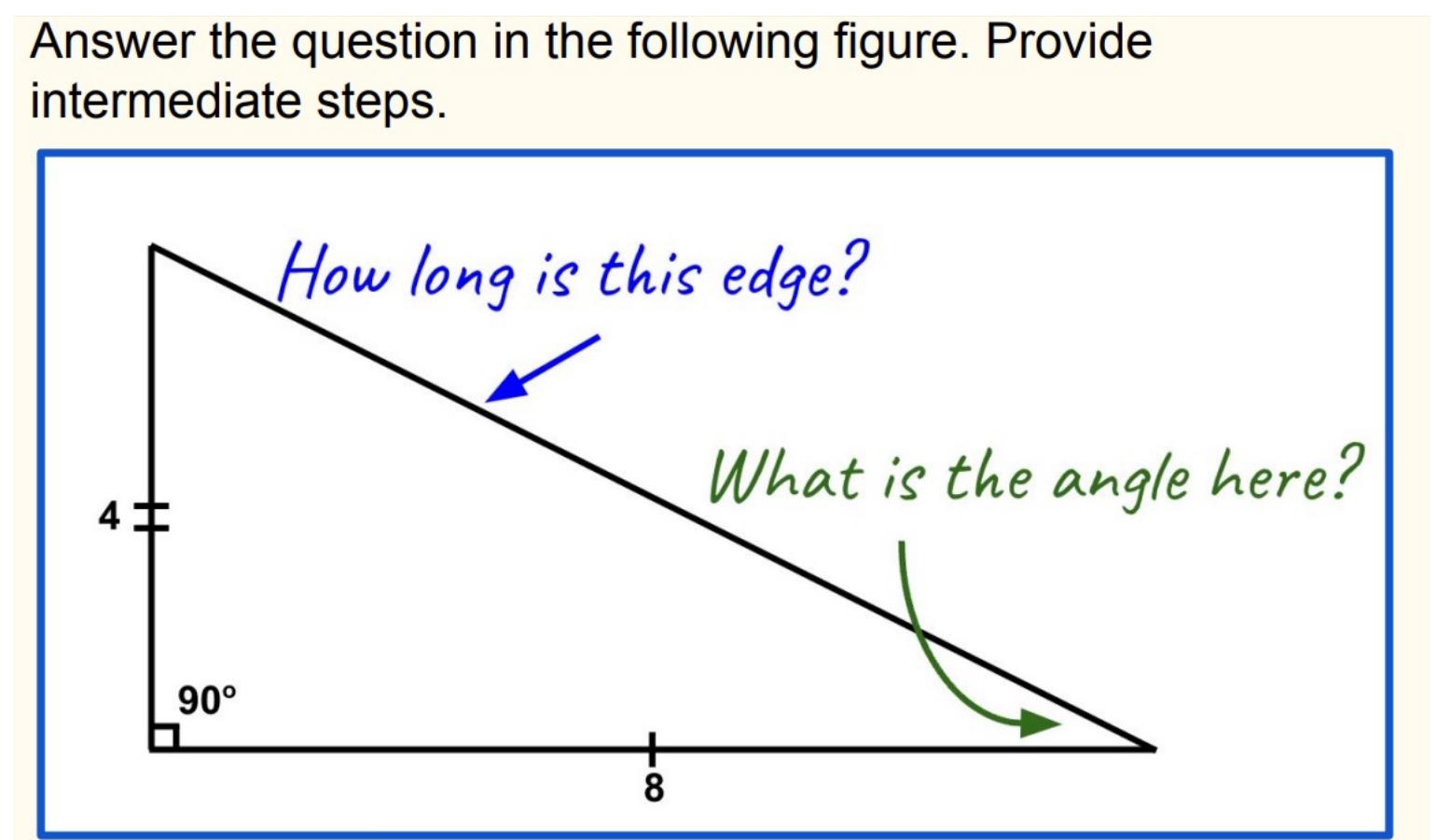
```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, va
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

The code editor has a light gray grid background and a dark sidebar on the left.

Image credit: <https://techcrunch.com/2021/06/29/github-previews-new-ai-tool-that-makes-coding-suggestions/>

Multi-modal LLM

Personal assistant w/ visual input



GPT-4V:

To find the length of the blue arrowed edge, we can use the Pythagorean theorem since we have a right triangle. The theorem states that $a^2 + b^2 = c^2$, where a and b are the two legs of the right triangle and c is the hypotenuse. Plugging in the values we have, we get:

$$4^2 + 8^2 = c^2 \quad 16 + 64 = c^2 \quad 80 = c^2 \quad \sqrt{80} = c \quad 8.94 \approx c$$

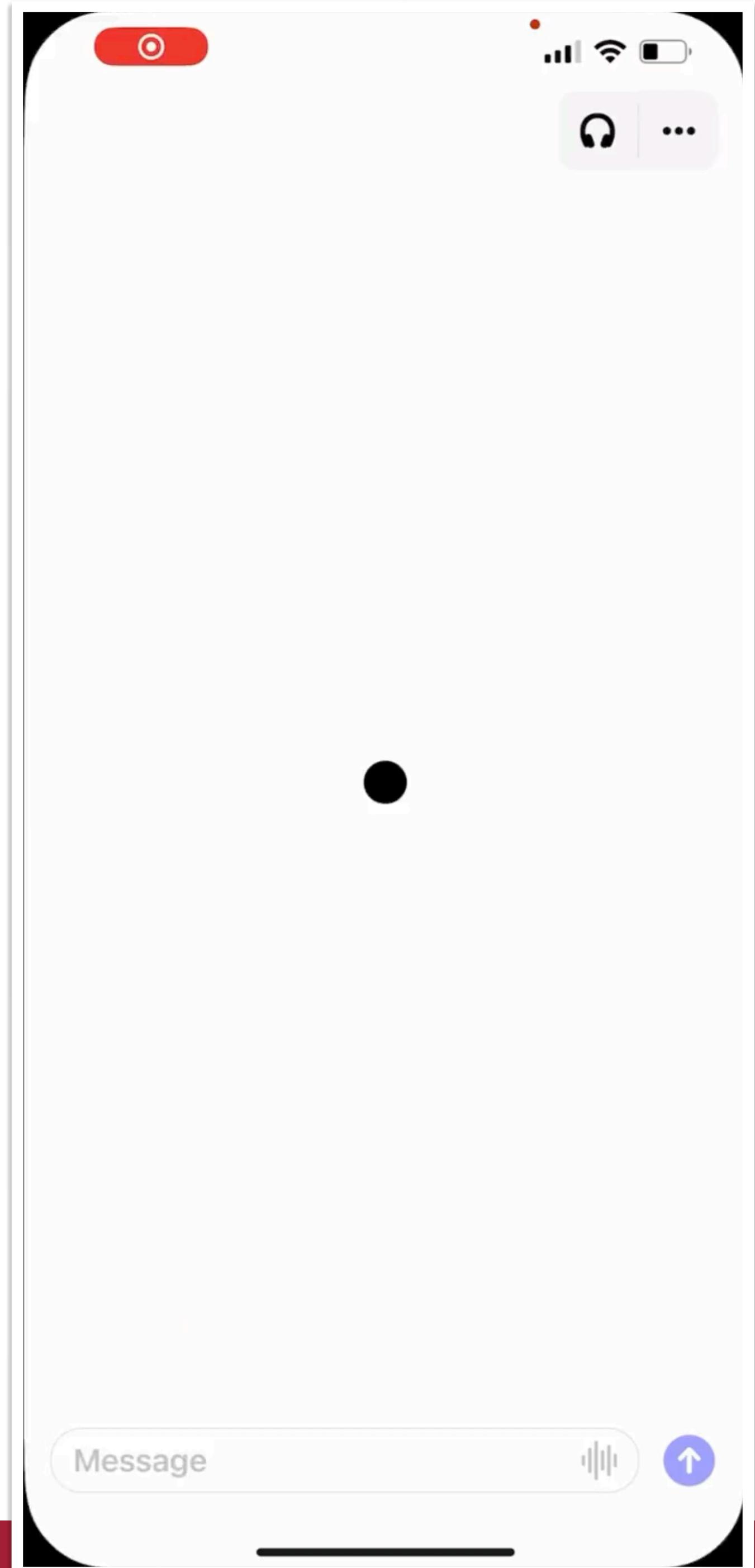
Therefore, the length of the blue arrowed edge is approximately 8.94.

To find the angle at the green arrow, we can use the tangent function since we have the opposite and adjacent sides of the angle. The tangent function states that $\tan\theta = \text{opposite/adjacent}$. Plugging in the values we have, we get:

$$\tan\theta = 4/8 \quad \tan\theta = 1/2 \quad \theta = \tan^{-1}(1/2) \quad \theta \approx 26.57^\circ$$

Therefore, the angle at the green arrow is approximately 26.57° .

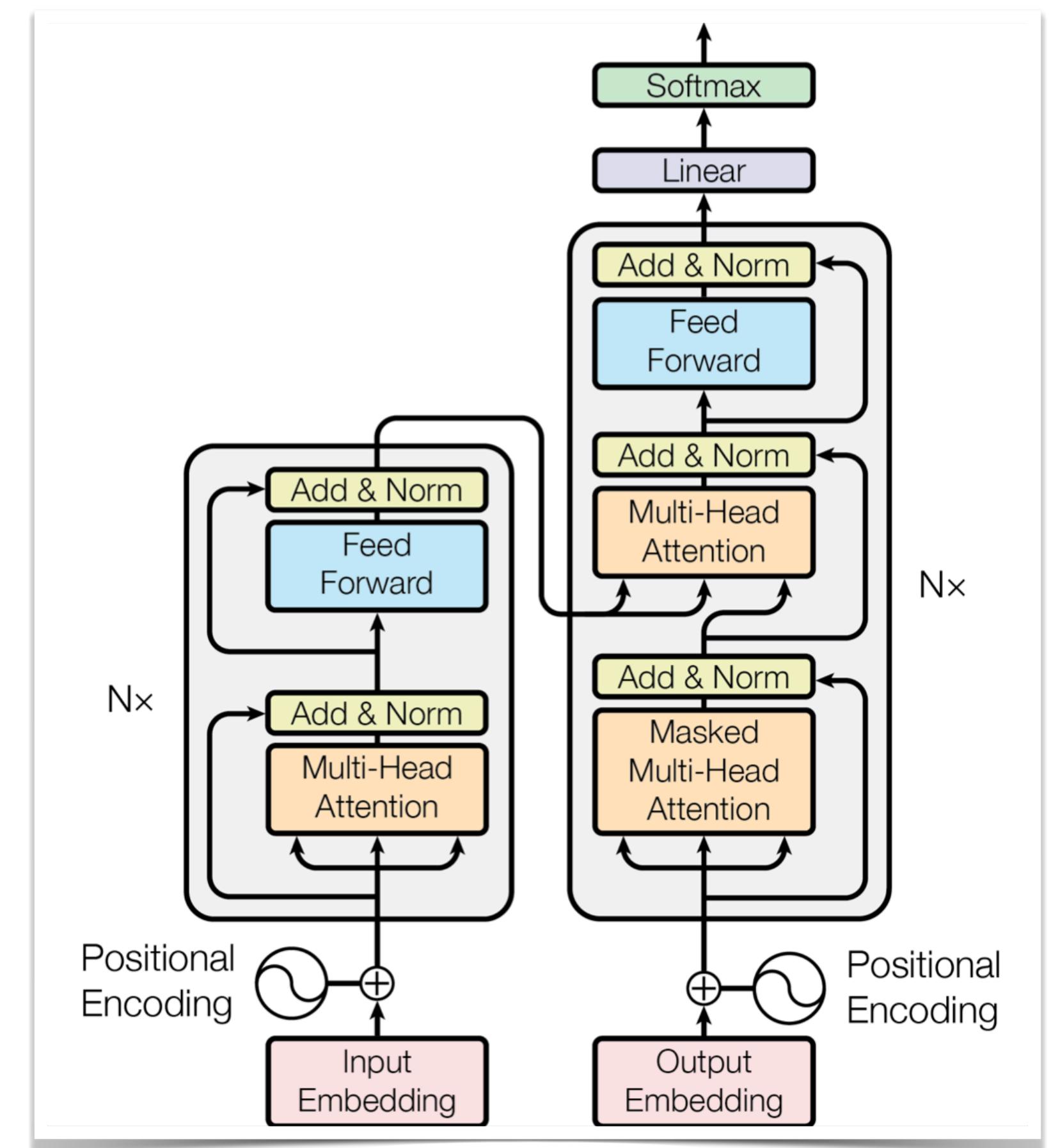
<https://openai.com/blog/chatgpt-can-now-see-hear-and-speak>



Lecture Plan

Today, we will cover:

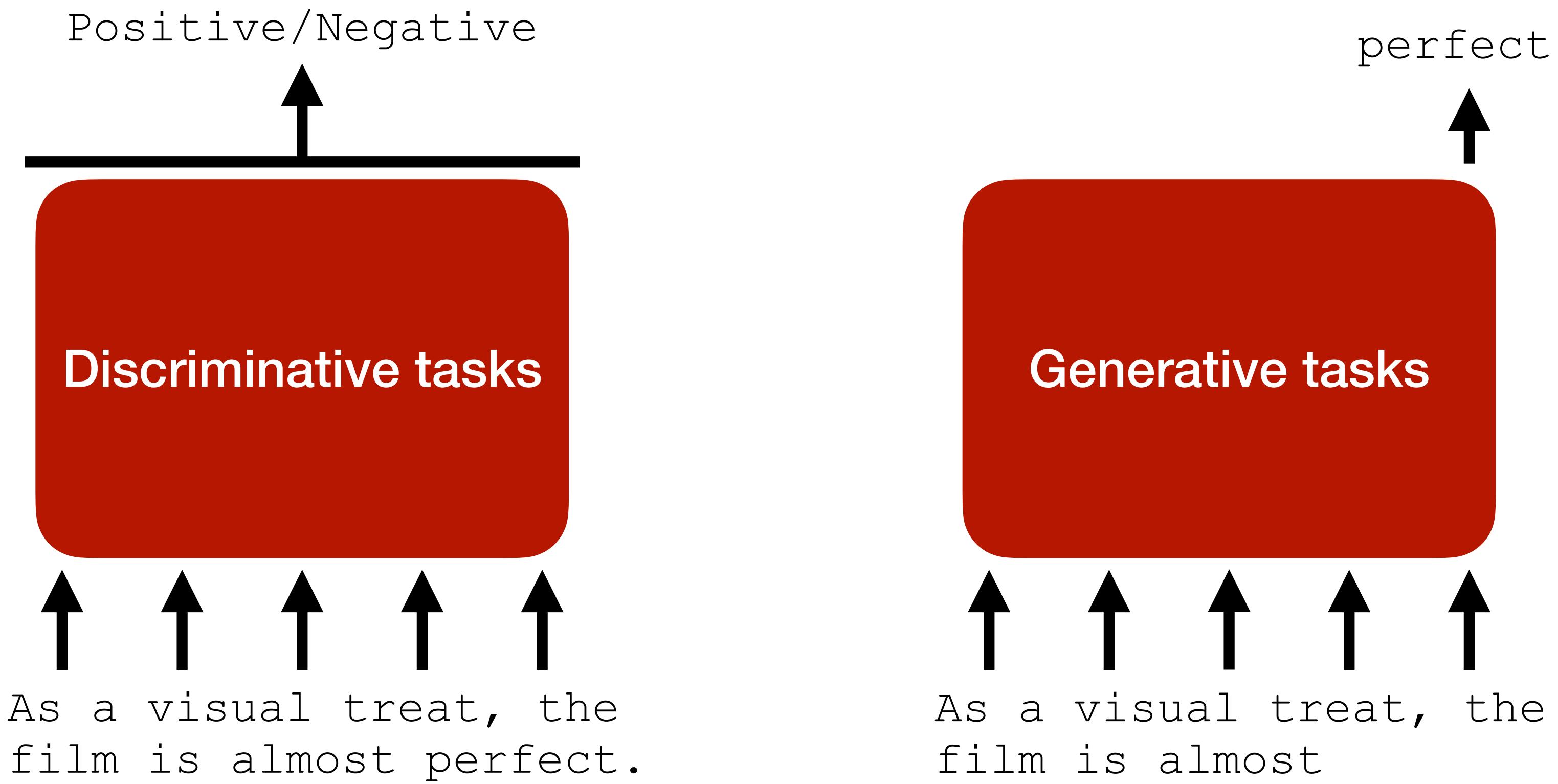
1. Transformer basics
2. Transformer design variants
3. Large language models (LLMs)
4. Advanced topics, multi-modal LLM



I. Transformer Basics

Revisit: NLP Tasks

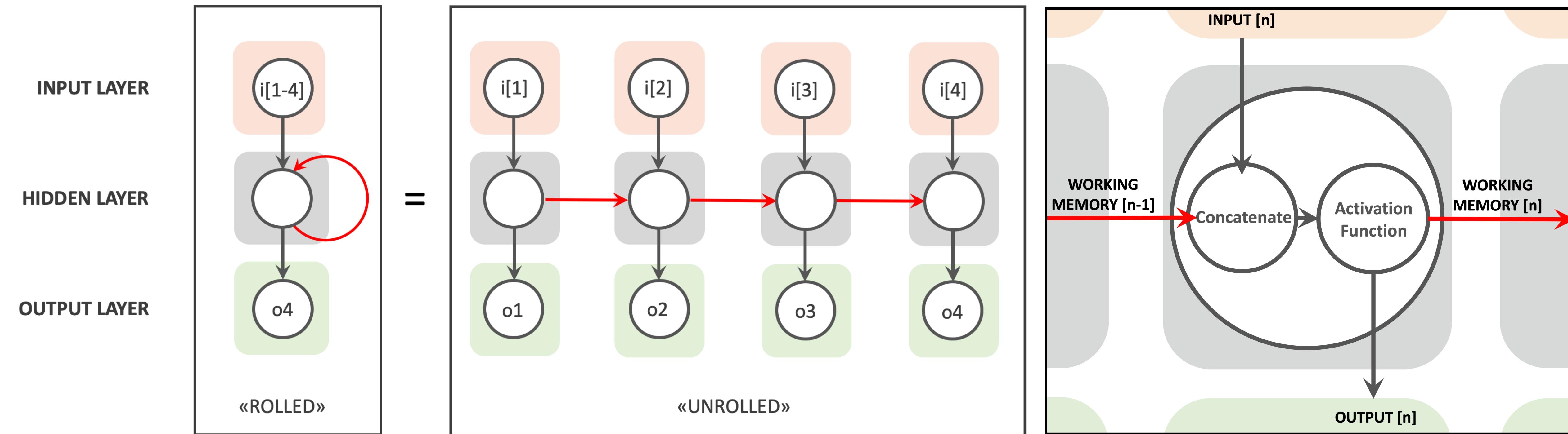
- **Discriminative tasks** (e.g., sentiment analysis, text classification, textual Entailment)
- **Generative tasks** (e.g., language modeling, machine translation, summarization)



Pre-Transformer Era

Recurrent Neural Networks (RNNs)

- The "working memory" struggles to retain **long-term** dependancies (can be solved by LSTM).
- There is **dependency** across tokens, limiting the training efficiency.

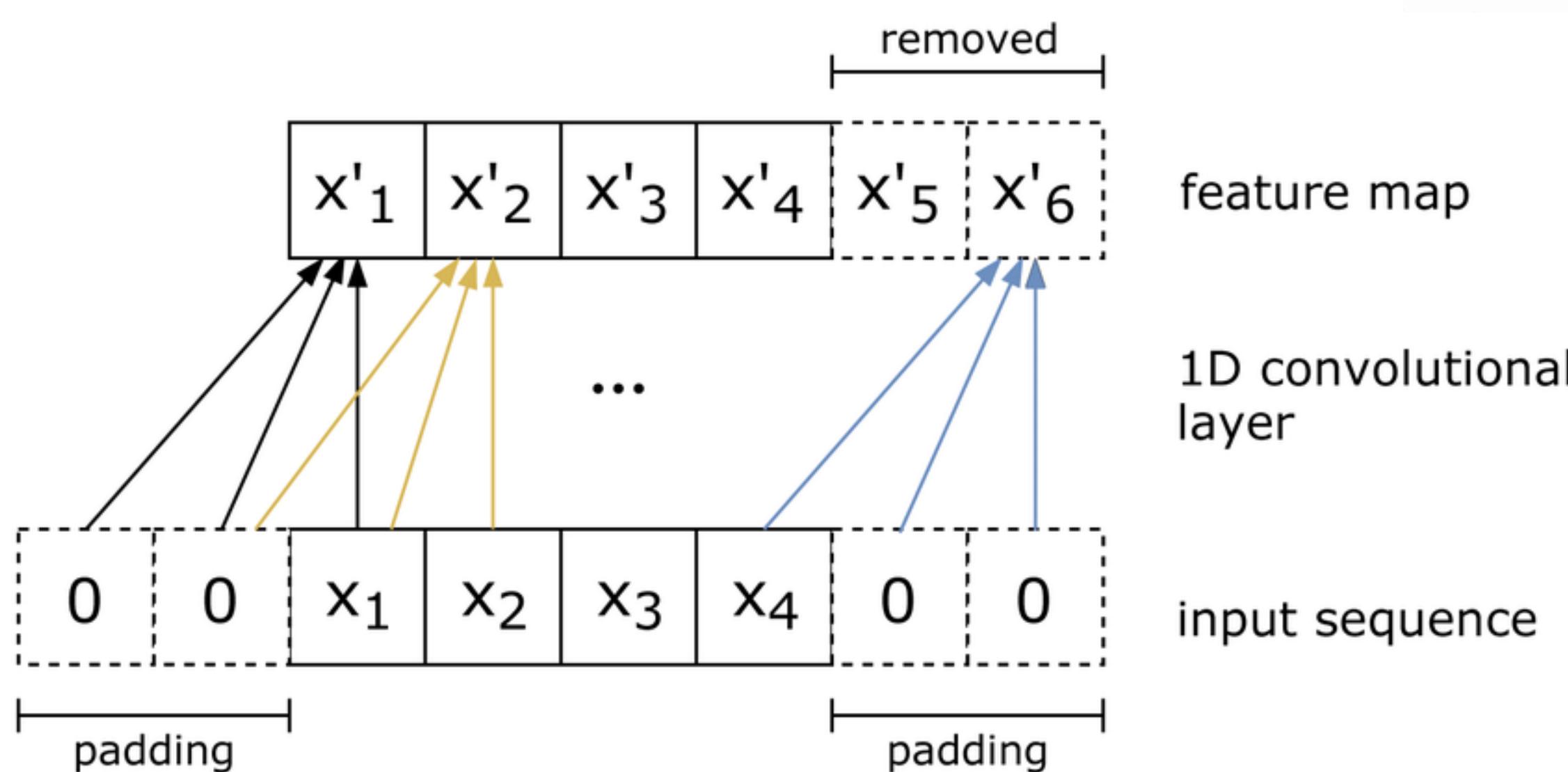


Content credit: <https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks>

Pre-Transformer Era

Convolutional Neural Networks (CNNs)

- No dependency between tokens, leading to **better scalability**.
- **Limited context information**, resulting in worse modeling capability.



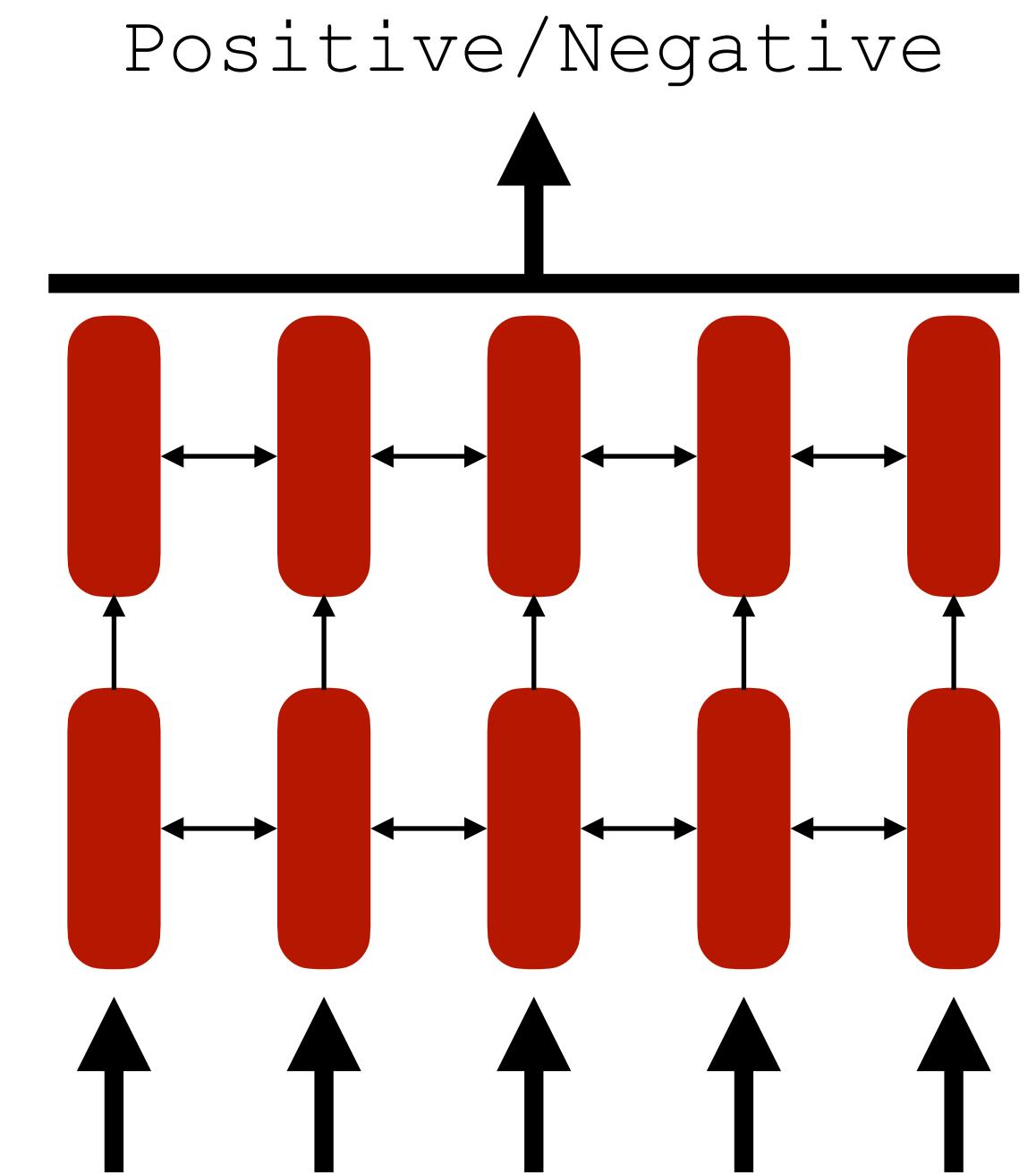
this	→	0.2	0.4	-0.3	
movie	→	0.1	0.2	0.6	
has	→	-0.1	0.4	-0.1	
amazing	→	0.7	-0.5	0.4	
diverse	→	0.1	-0.2	0.1	
characters	→	0.6	-0.3	0.8	

Image credit: https://cezannec.github.io/CNN_Text_Classification/

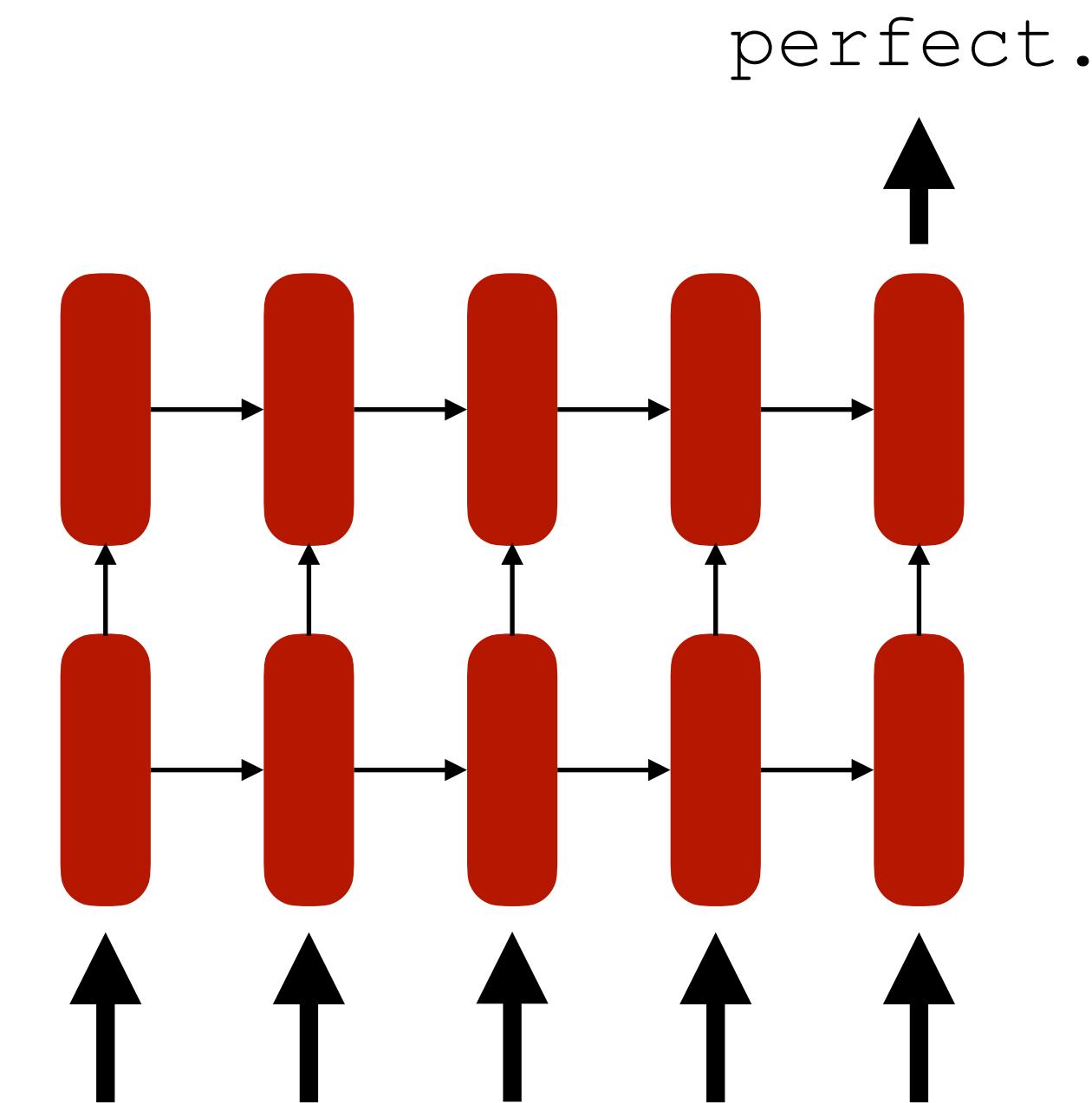
Pre-Transformer Era

NLP with RNN/LSTM

- **Bi-directional** RNNs for discriminative tasks (encoding)
- **Uni-directional** RNNs for generative tasks (decoding)



As a visual treat, the film is almost perfect.

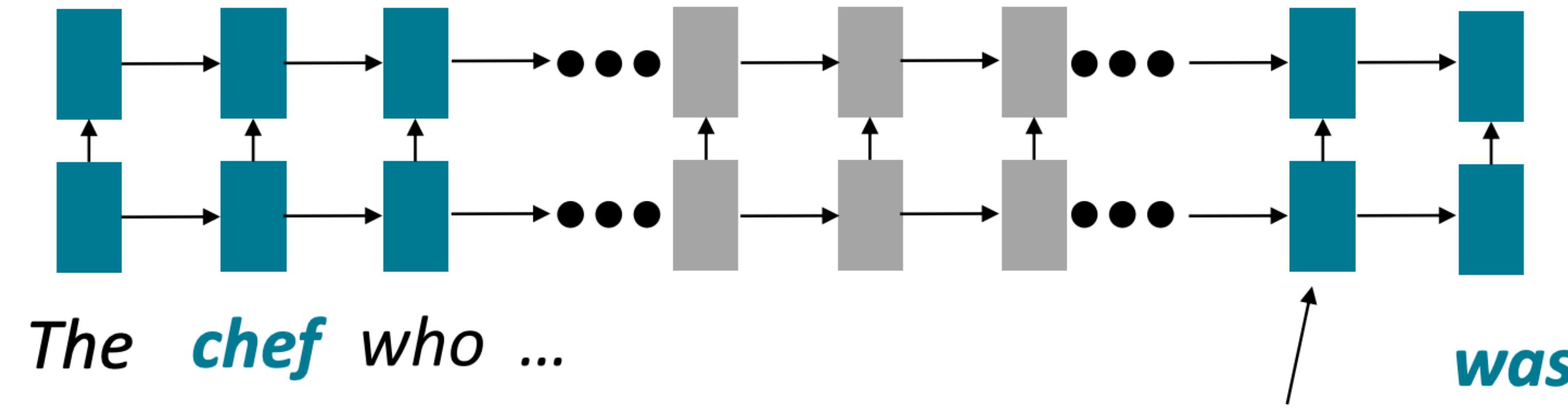


As a visual treat, the film is almost perfect.

Pre-Transformer Era

Problem with RNNs/LSTMs

- Hard to model long-term relationships
 - It takes $O(\text{seq_len})$ steps to model the interaction between two tokens
 - Images have locality; but languages may not have locality:

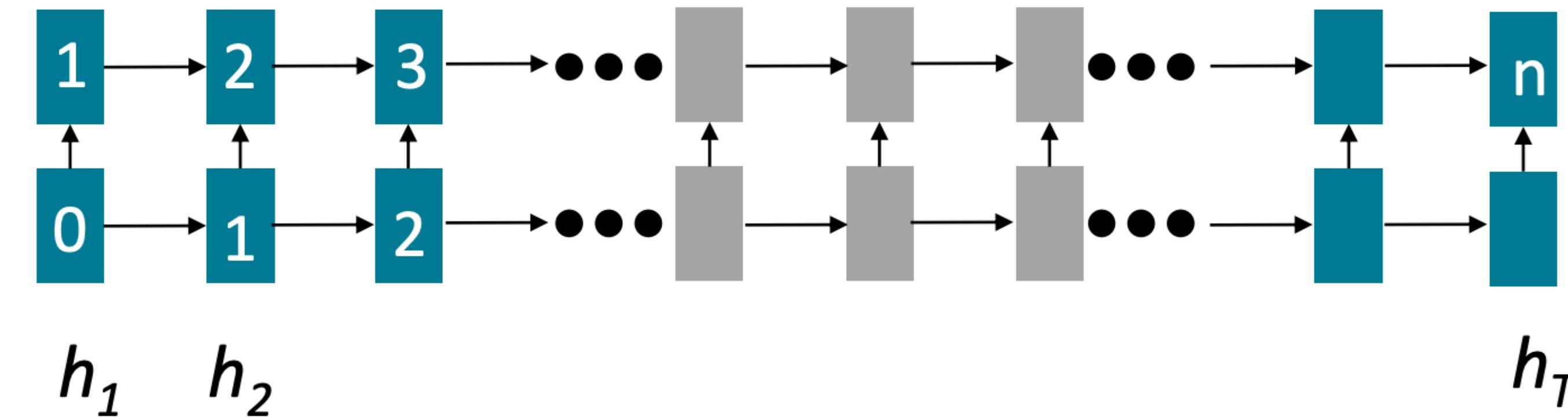


Info of *chef* has gone through
 $O(\text{sequence length})$ many layers!

Pre-Transformer Era

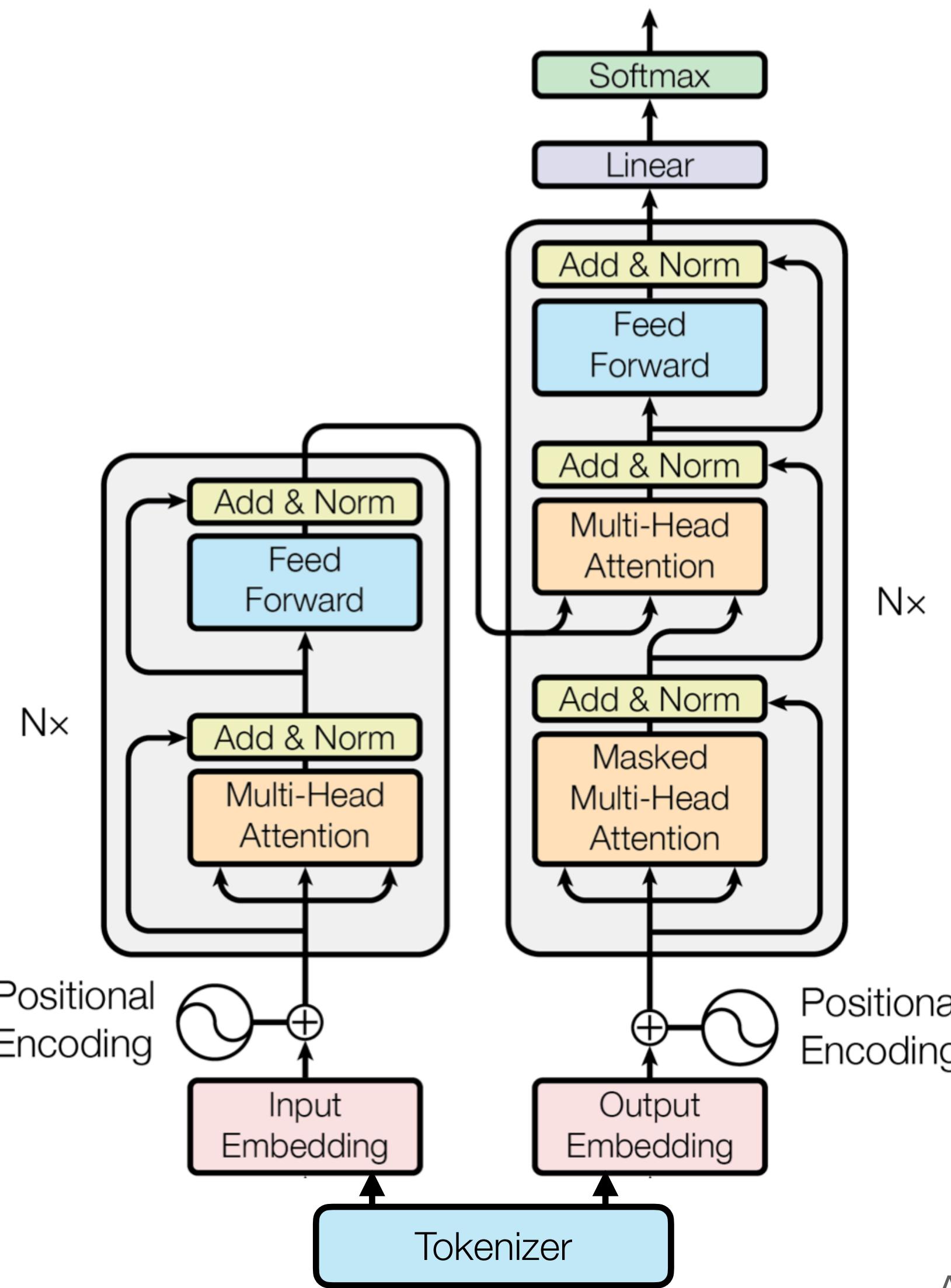
Problem with RNNs/LSTMs

- Hard to model long-term relationships
- Limited training parallelism
 - The states are strictly dependent on earlier states
 - It takes n steps to get state n



Numbers indicate min # of steps before a state can be computed

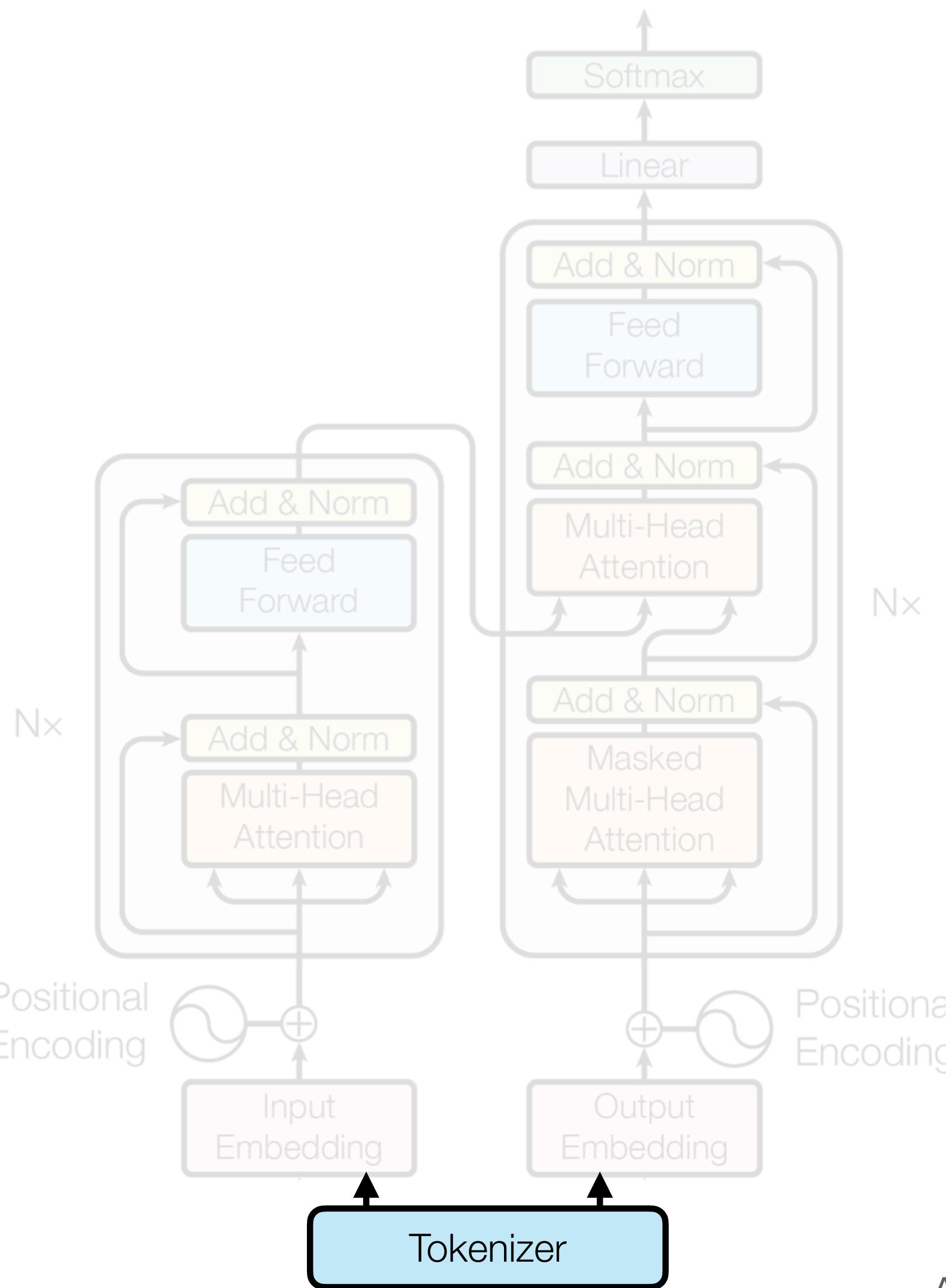
Transformer



- Tokenize words (word -> tokens)
- Map tokens into embeddings
- Embeddings go through Transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
 - LayerNorm
 - Residual connection
- Positional encoding
- Final prediction with Linear head

Attention Is All You Need [Vaswani et al., 2017]

Transformer



- **Tokenize words (word -> tokens)**
- Map tokens into embeddings
- Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
- LayerNorm
- Residual connection
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Tokenization

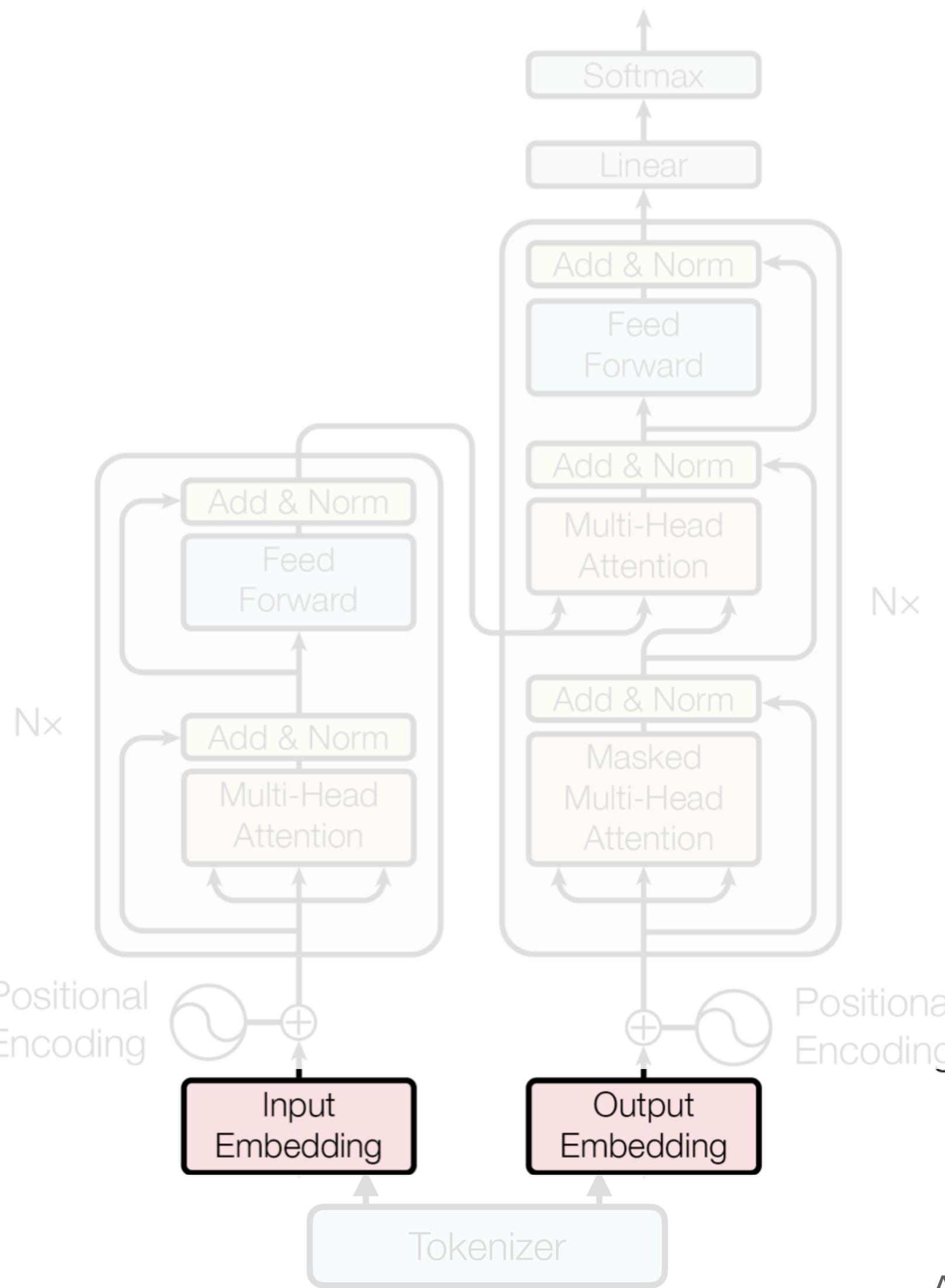
- A tokenizer maps a word to one/multiple tokens:
- **110 words => 162 tokens**

Large generative models (e.g., large language models, diffusion models) have shown remarkable performance, but they require a massive amount of computational resources. To make them more accessible, it is crucial to improve their efficiency. This course will introduce efficient AI computing techniques that enable powerful deep learning applications on resource-constrained devices. Topics include model compression, pruning, quantization, neural architecture search, distributed training, data/model parallelism, gradient compression, and on-device fine-tuning. It also introduces application-specific acceleration techniques for large language models, diffusion models, video recognition, and point cloud. This course will also cover topics about quantum machine learning. Students will get hands-on experience deploying large language models (e.g., LLaMA 2) on a laptop.

TEXT

TOKEN IDS

Transformer



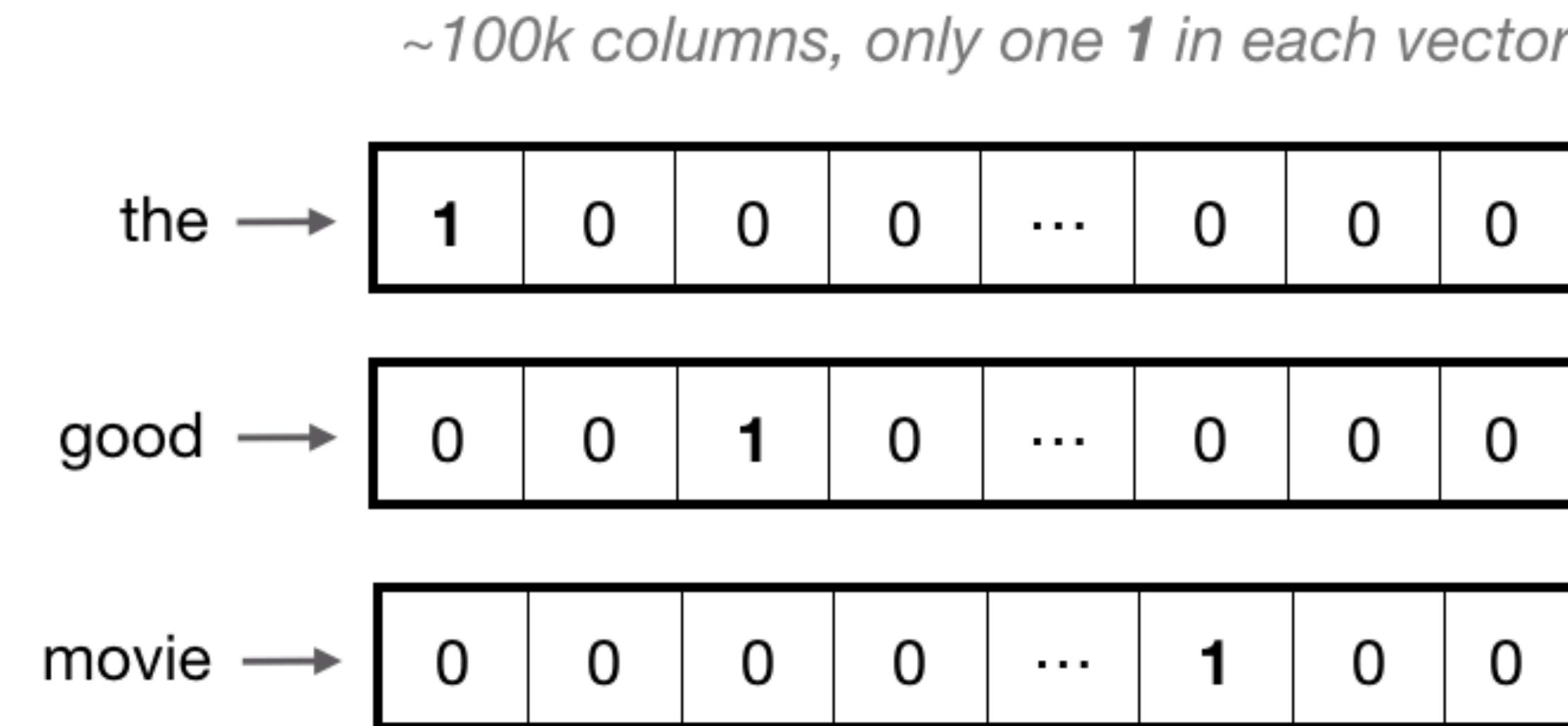
- Tokenize words (word -> tokens)
- **Map tokens into embeddings**
- Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
- LayerNorm
- Residual connection
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Word Representation

One-Hot Encoding

- **Key idea:** Representing each word as a vector that has as many values in it as there are words in the vocabulary. Each column in a vector represents one possible word in a vocabulary.



- For **large vocabularies**, these vectors can get **very long**, and they contain all 0's except for one value. This is considered a very sparse representation.

Content credit: https://cezannec.github.io/CNN_Text_Classification/

Word Representation

Word Embedding

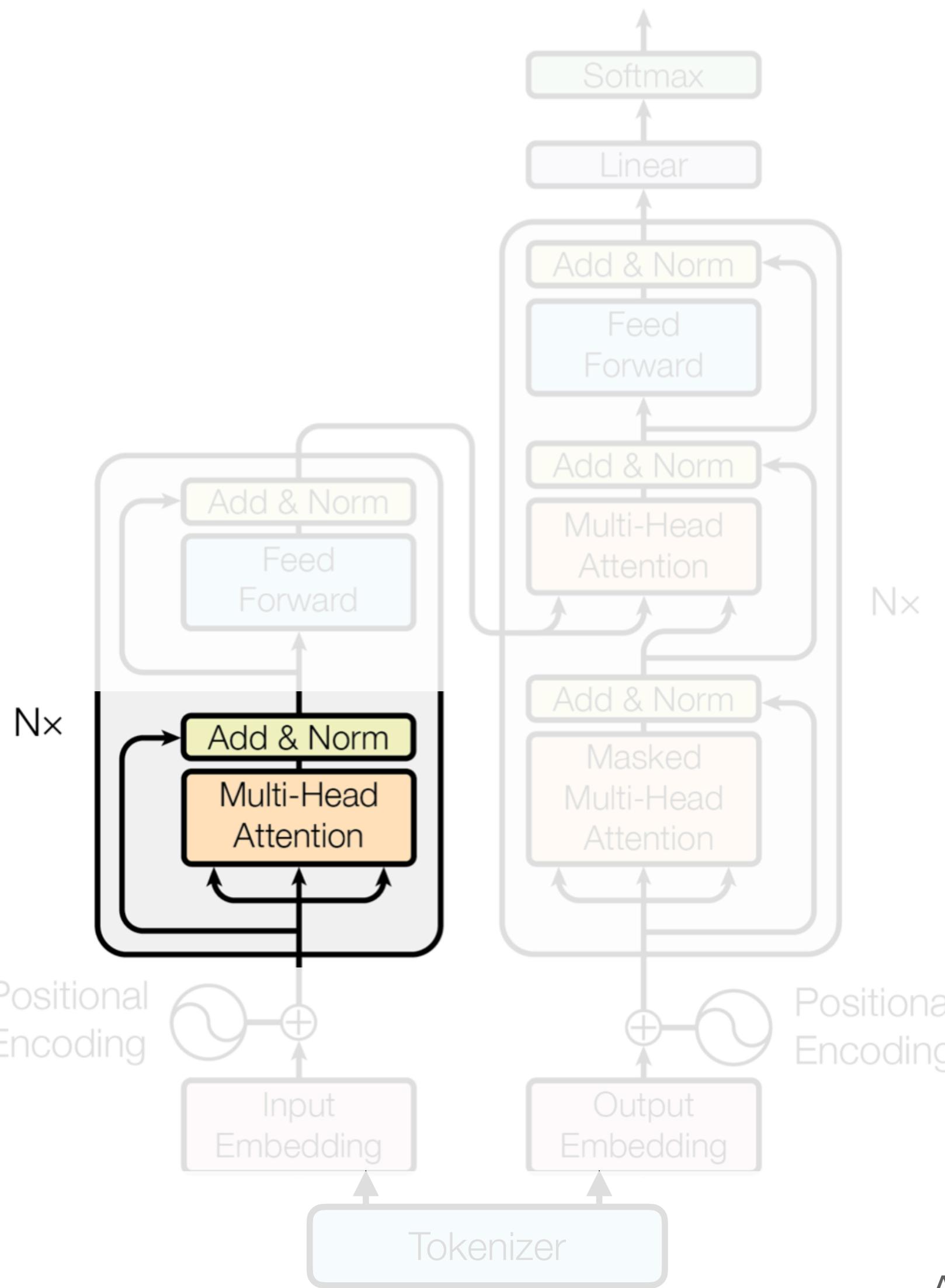
- **Key idea:** Map the word index to a **continuous** word embedding through a **look-up table**.

the	→	0.2	0.4	-0.1
good	→	0.7	-0.5	0.3
movie	→	0.1	0.2	0.6

- The word embedding can be trained **end-to-end** with the model for the downstream tasks.
 - Popular pre-trained word embeddings: Word2Vec, GloVe.

Content credit: https://cezannec.github.io/CNN_Text_Classification/

Transformer



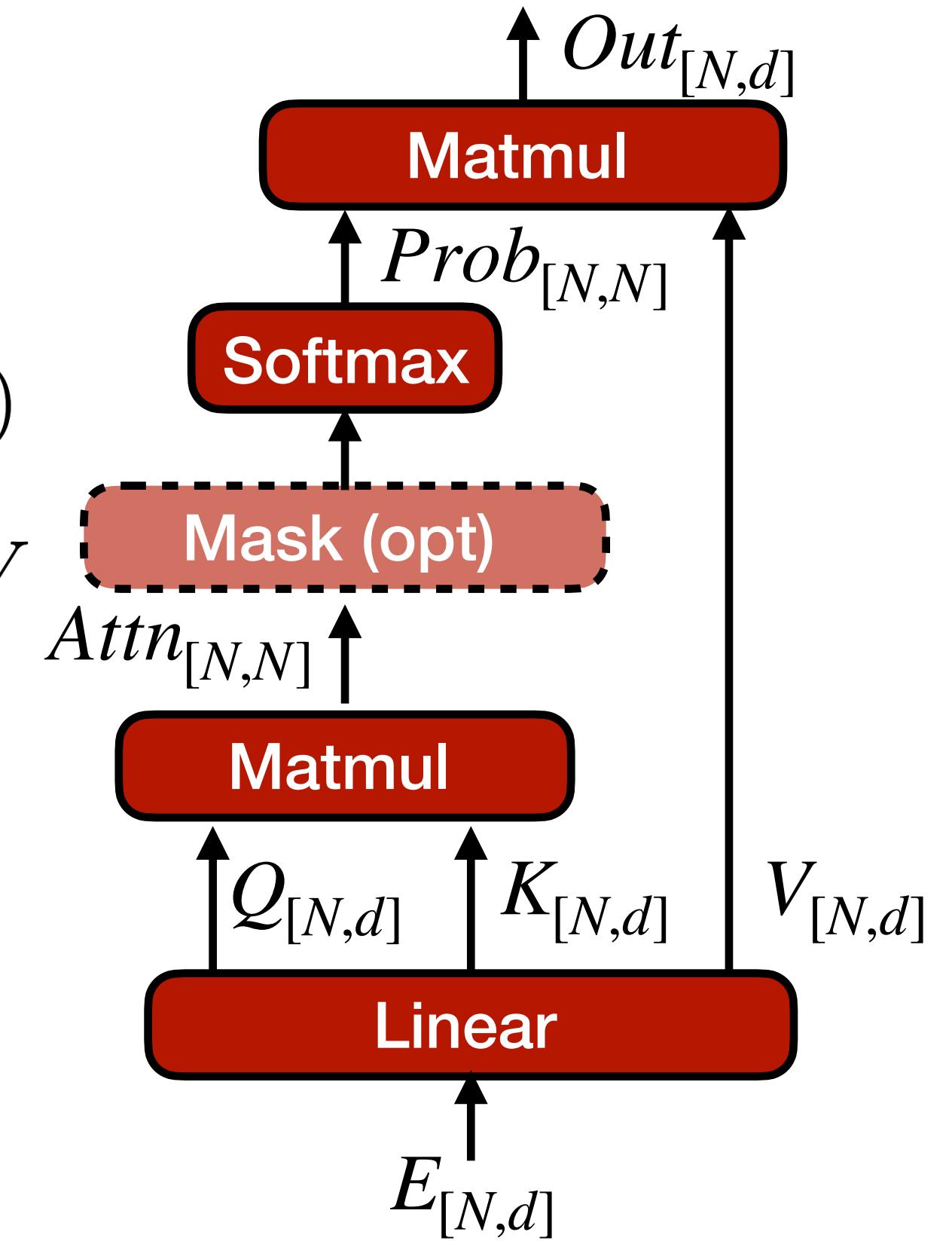
- Tokenize words (word -> tokens)
- Map tokens into embeddings
- Embeddings go through transformer blocks
 - **Multi-Head Attention (MHA)**
 - Feed-Forward Network (FFN)
 - LayerNorm
 - Residual connection
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Self-Attention

- Project the embedding E into query, key, and value (Q, K, V)
- The Query-Key-Value design is analogous to a retrieval system,
Let's take YouTube search as an example:
 - Query: text prompt in the search bar
 - Key: the titles/descriptions of videos
 - Value: the corresponding videos
- Multiply Q and K to get the inner product (normalize by \sqrt{d})
- Followed by Softmax to get the attention weights of shape $N \times N$
 - The attention computation has $O(N^2)$ complexity!
- Multiply attention weights with V to get the output

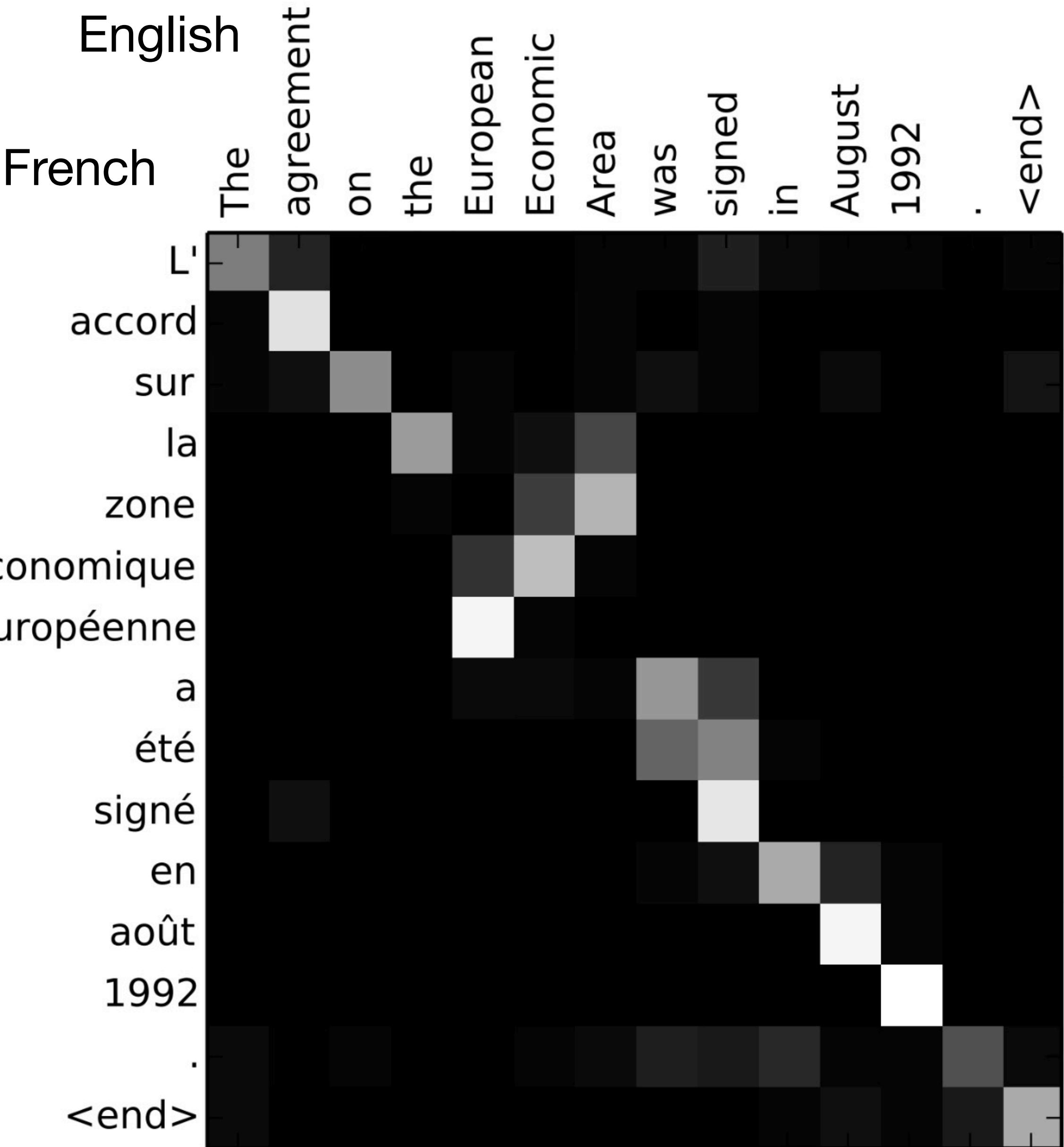
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Content credit: <https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms>

Self-Attention

- Example of attention matrix from machine translation
- The attention mechanism finds the correspondence between words



Example from Greg Durrett

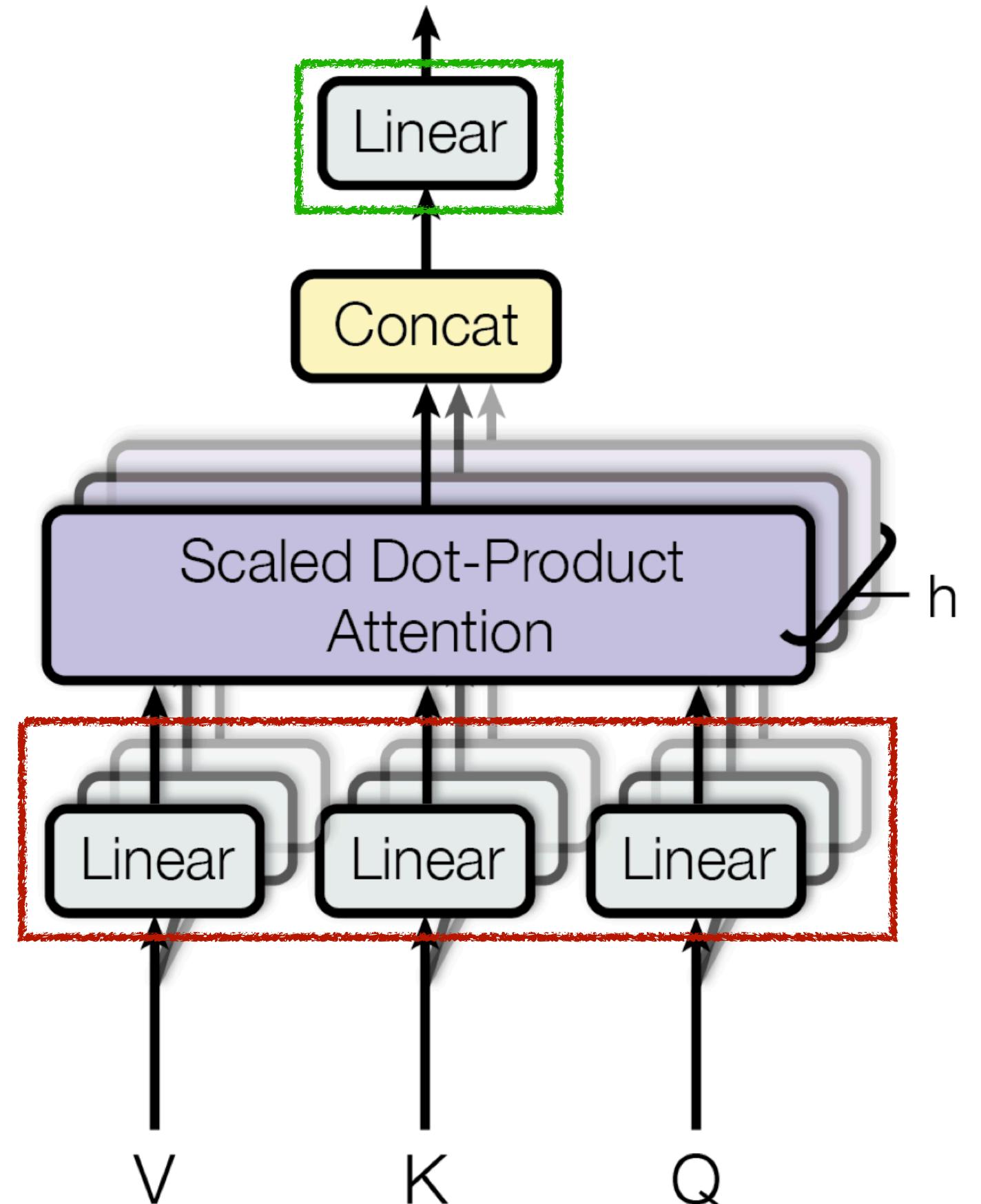
Multi-Head Attention (MHA)

Each head captures different semantics

- We need different attention maps to capture different semantic relationships
- Multi-head attention models different attention functionalities by having $H > 1$ heads (parallel branches of QKV attention)
- The final output is concatenated and goes through a linear projection to merge the features

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

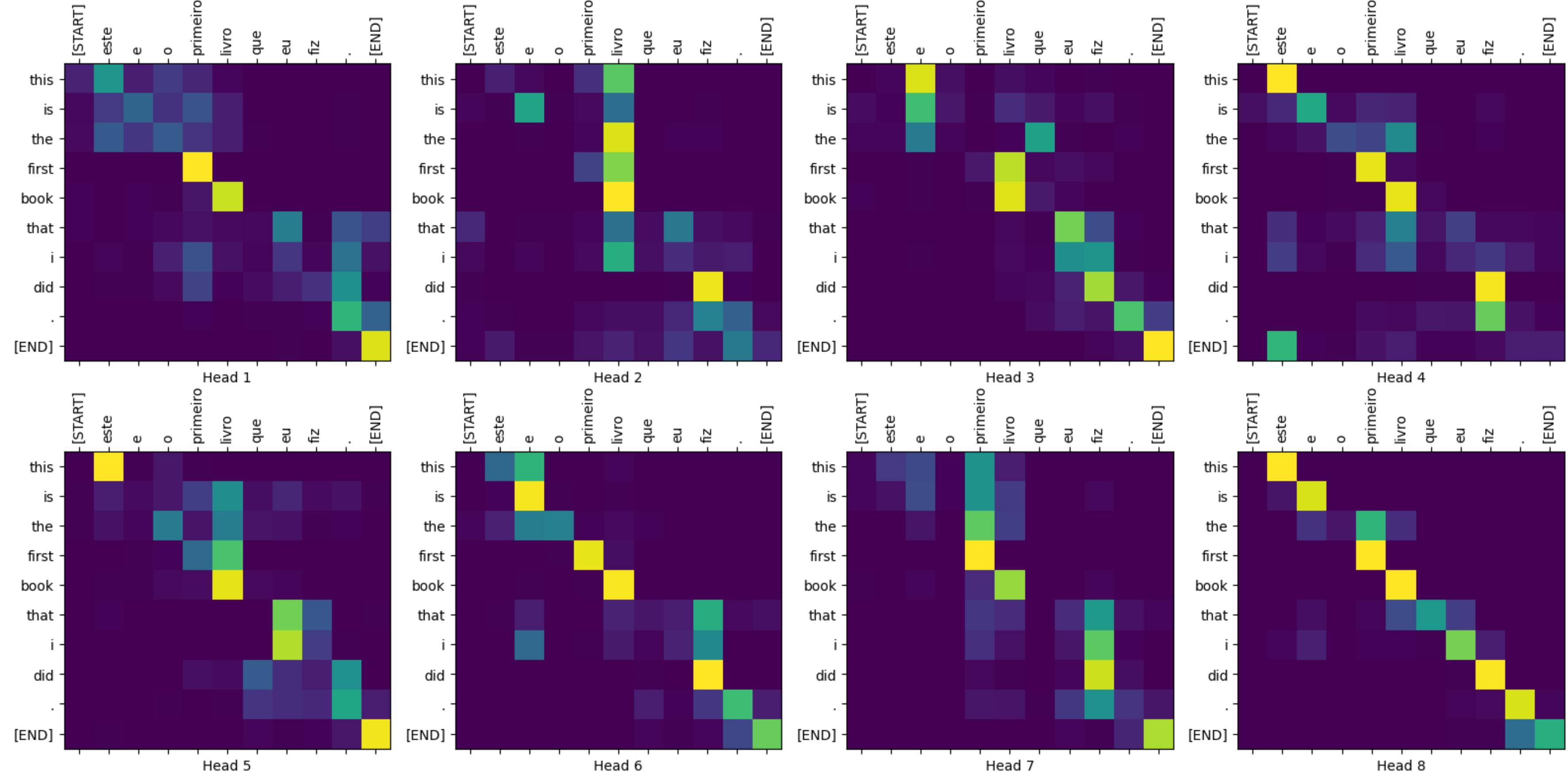


Content credit: https://cezannec.github.io/CNN_Text_Classification/

Multi-Head Attention (MHA)

Each head captures different semantics

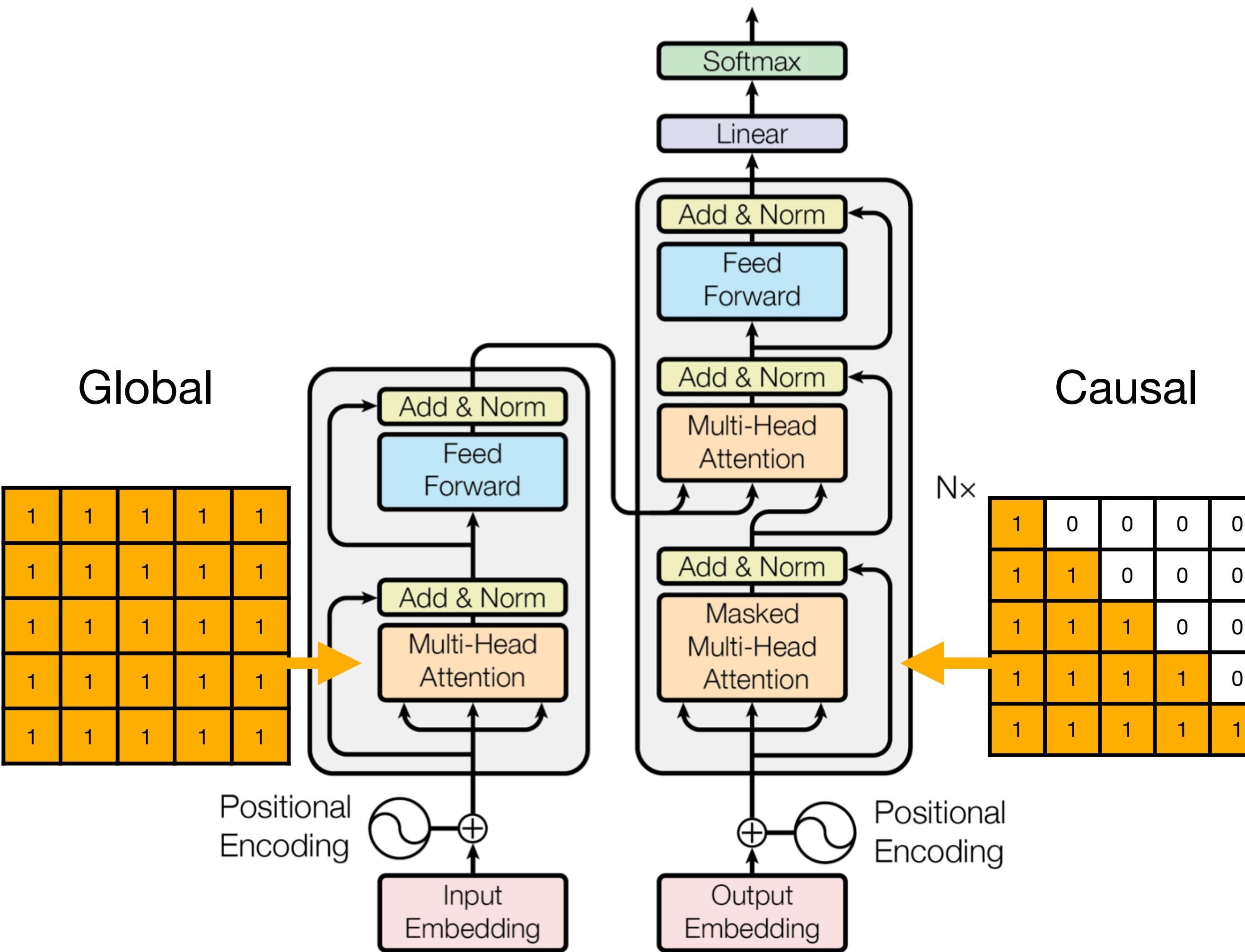
- Visualization from different heads



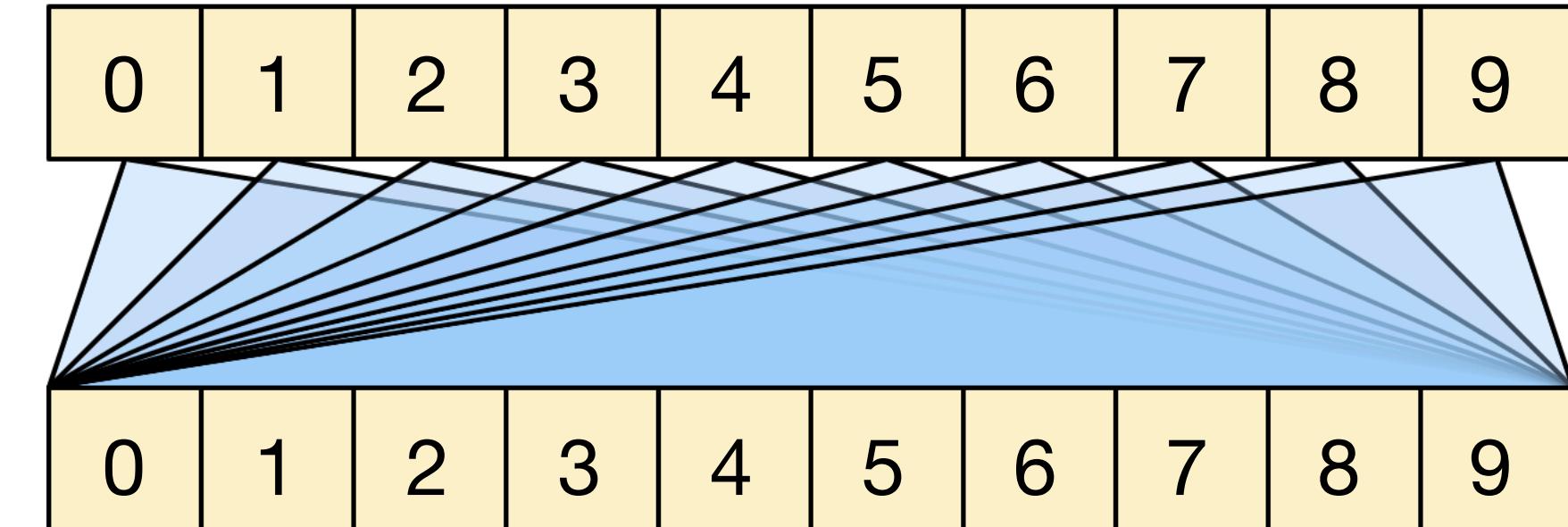
<https://www.tensorflow.org/text/tutorials/transformer>

Attention Masking

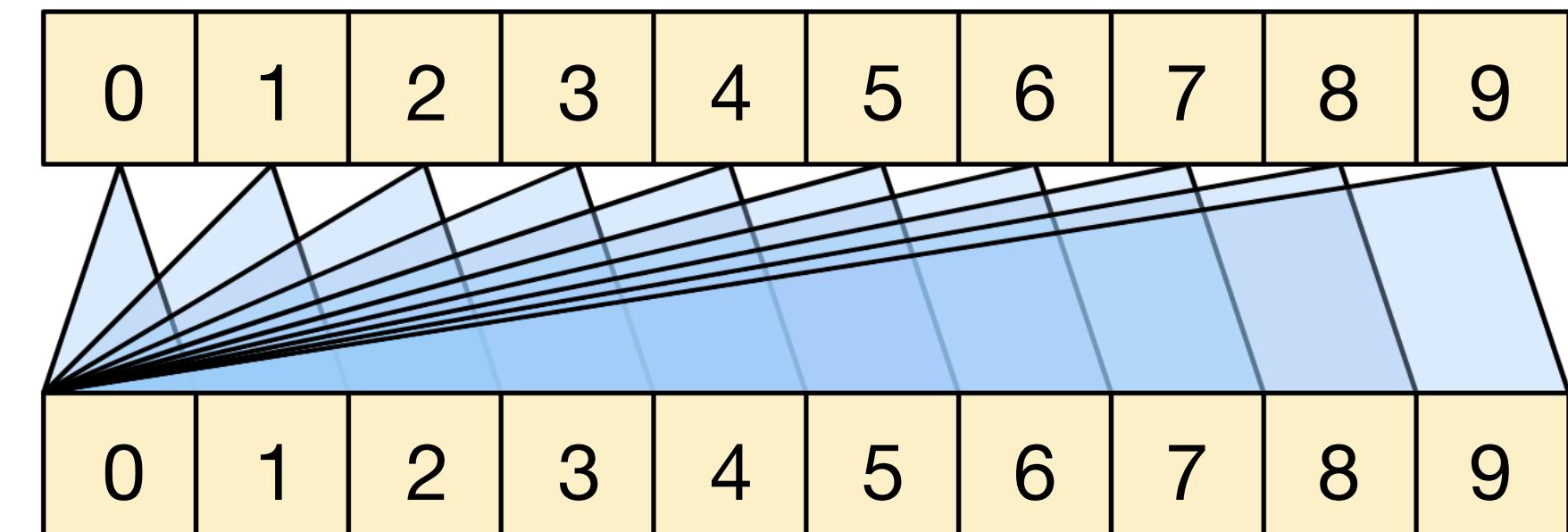
Causal/masked language modeling



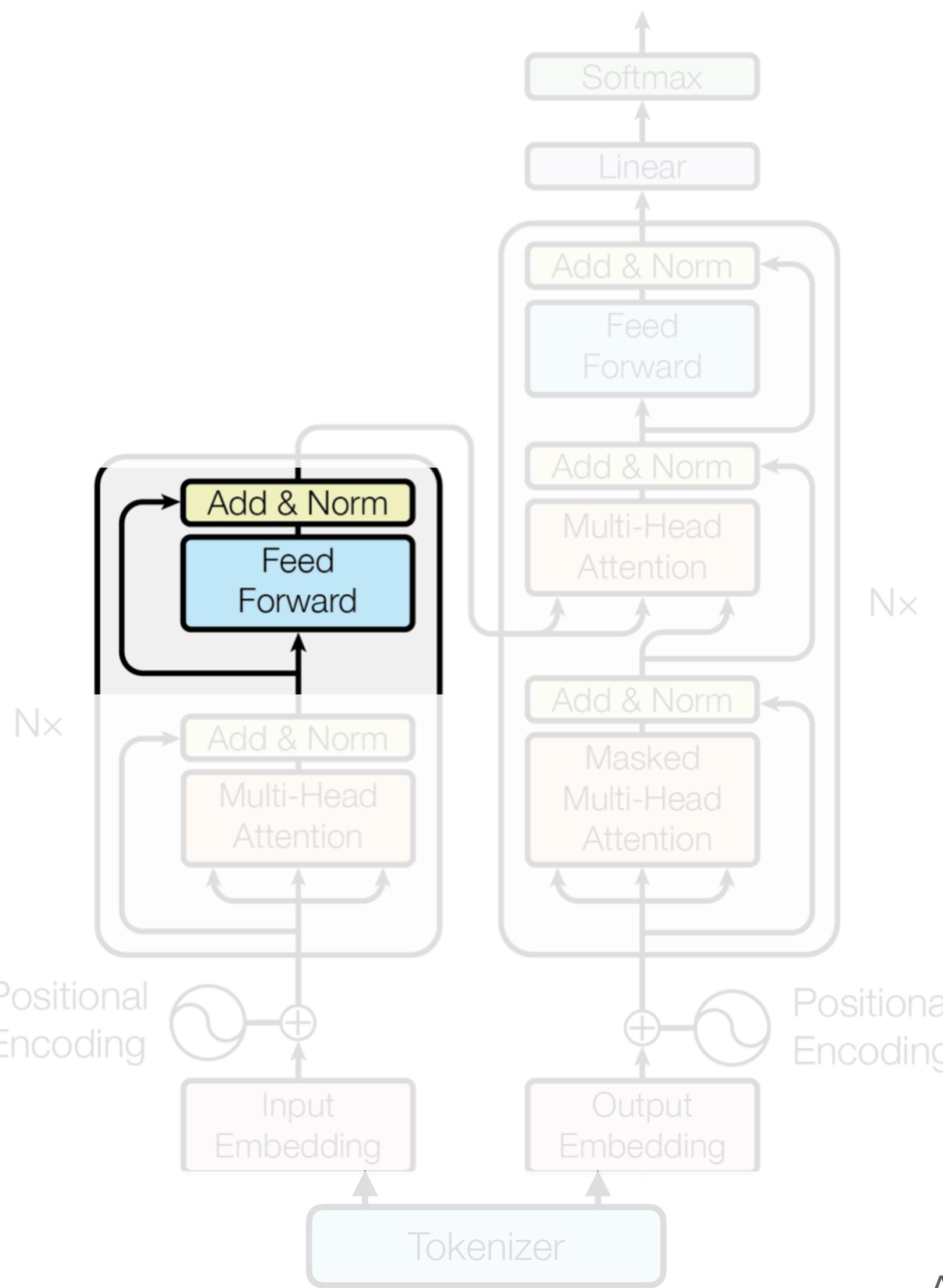
- Global self attention layer



- Causal self attention layer



Transformer



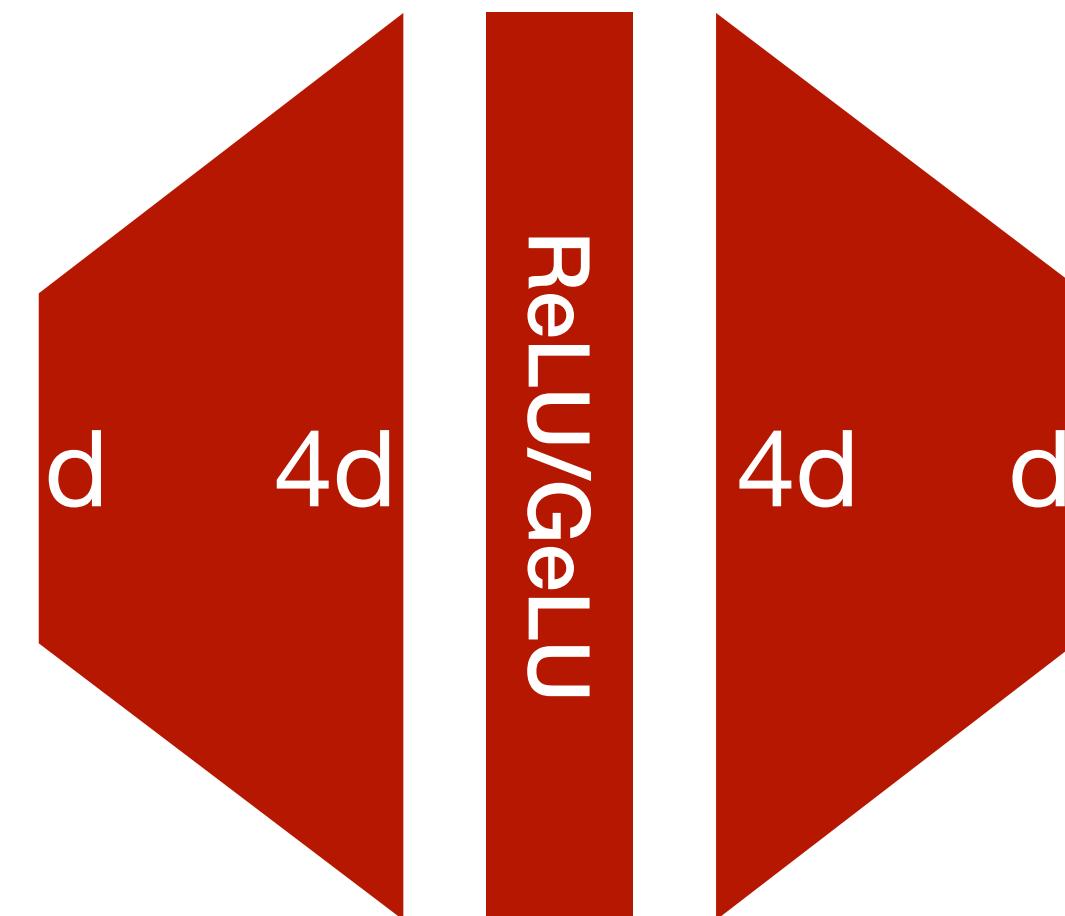
- Tokenize words (word -> tokens)
- Map tokens into embeddings
- Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - **Feed-Forward Network (FFN)**
- LayerNorm
- Residual connection
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Feed-Forward Network (FFN)

- Self-attention models relationships **between** tokens, but there is no **element-wise** non-linearity
- Add a feed-forward network (FFN) to help with feature modeling
- The vanilla implementation is a two-layer MLP with a larger hidden state size (**inverted bottleneck**) and ReLU/GeLU **activation**

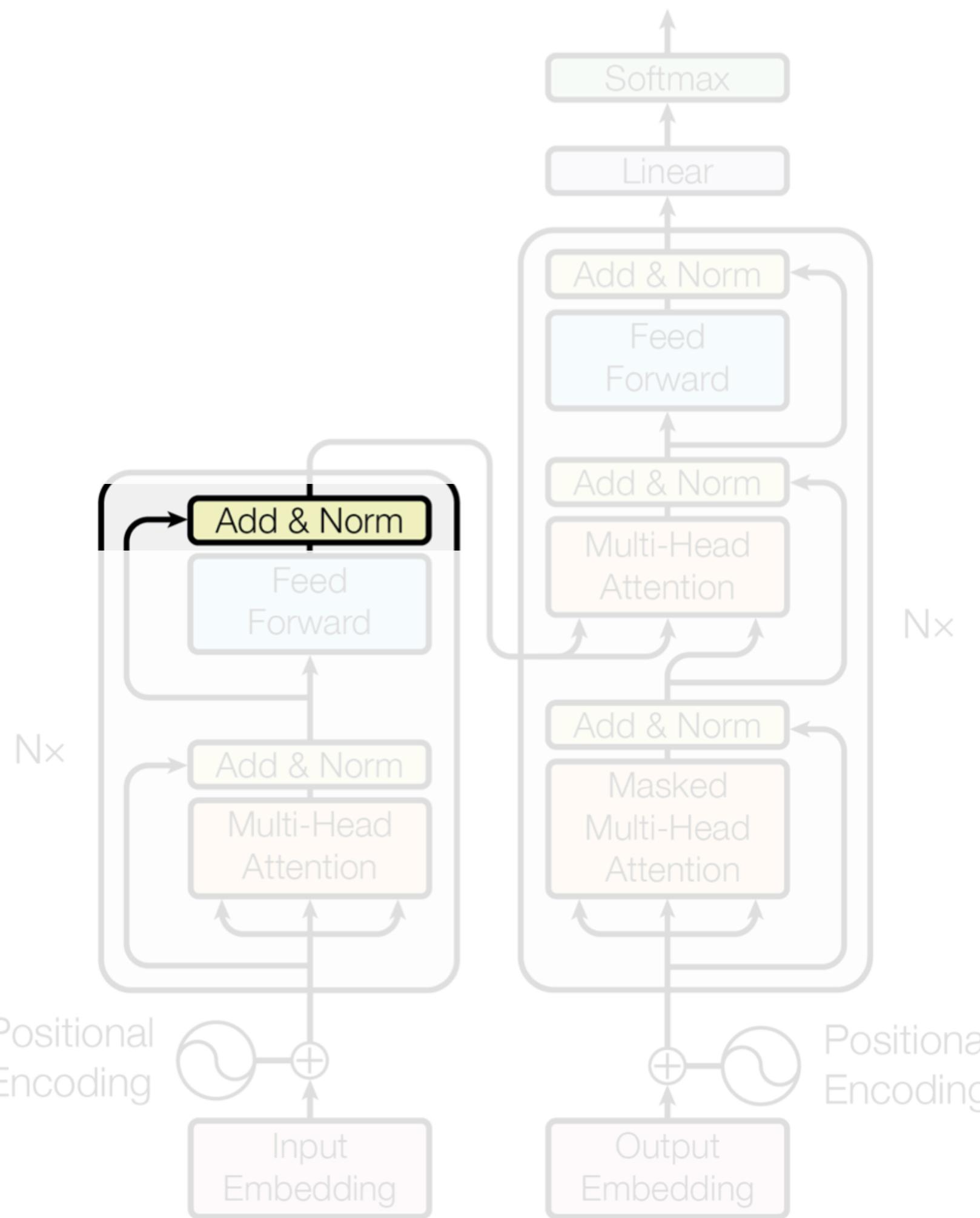
$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



- Later we are going to visit several design variants

Model	#L	#H	d_{model}	LR	Batch
125M	12	12	768	$6.0e-4$	0.5M
350M	24	16	1024	$3.0e-4$	0.5M
1.3B	24	32	2048	$2.0e-4$	1M
2.7B	32	32	2560	$1.6e-4$	1M
6.7B	32	32	4096	$1.2e-4$	2M
13B	40	40	5120	$1.0e-4$	4M
30B	48	56	7168	$1.0e-4$	4M
66B	64	72	9216	$0.8e-4$	2M
175B	96	96	12288	$1.2e-4$	2M

Transformer

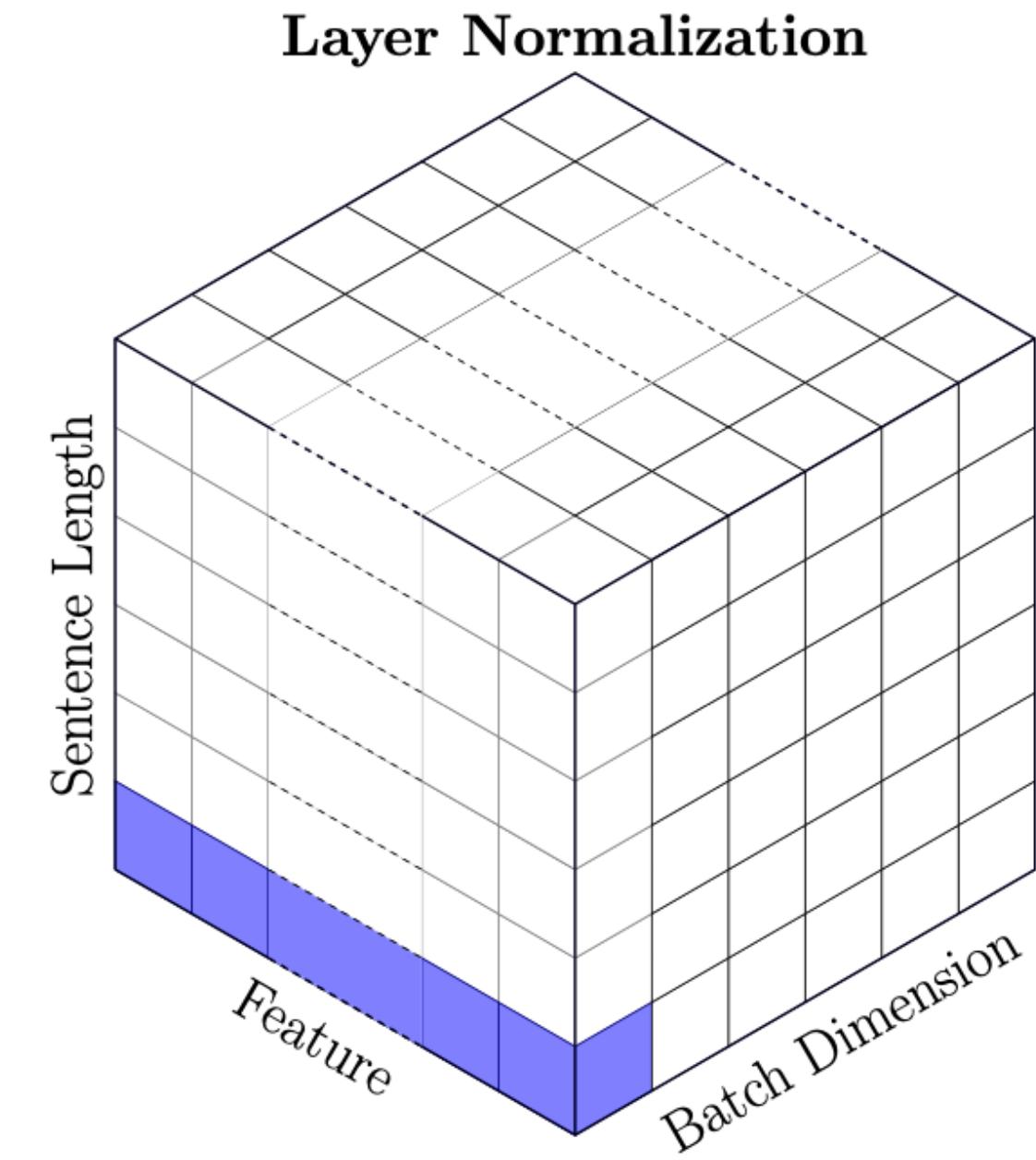
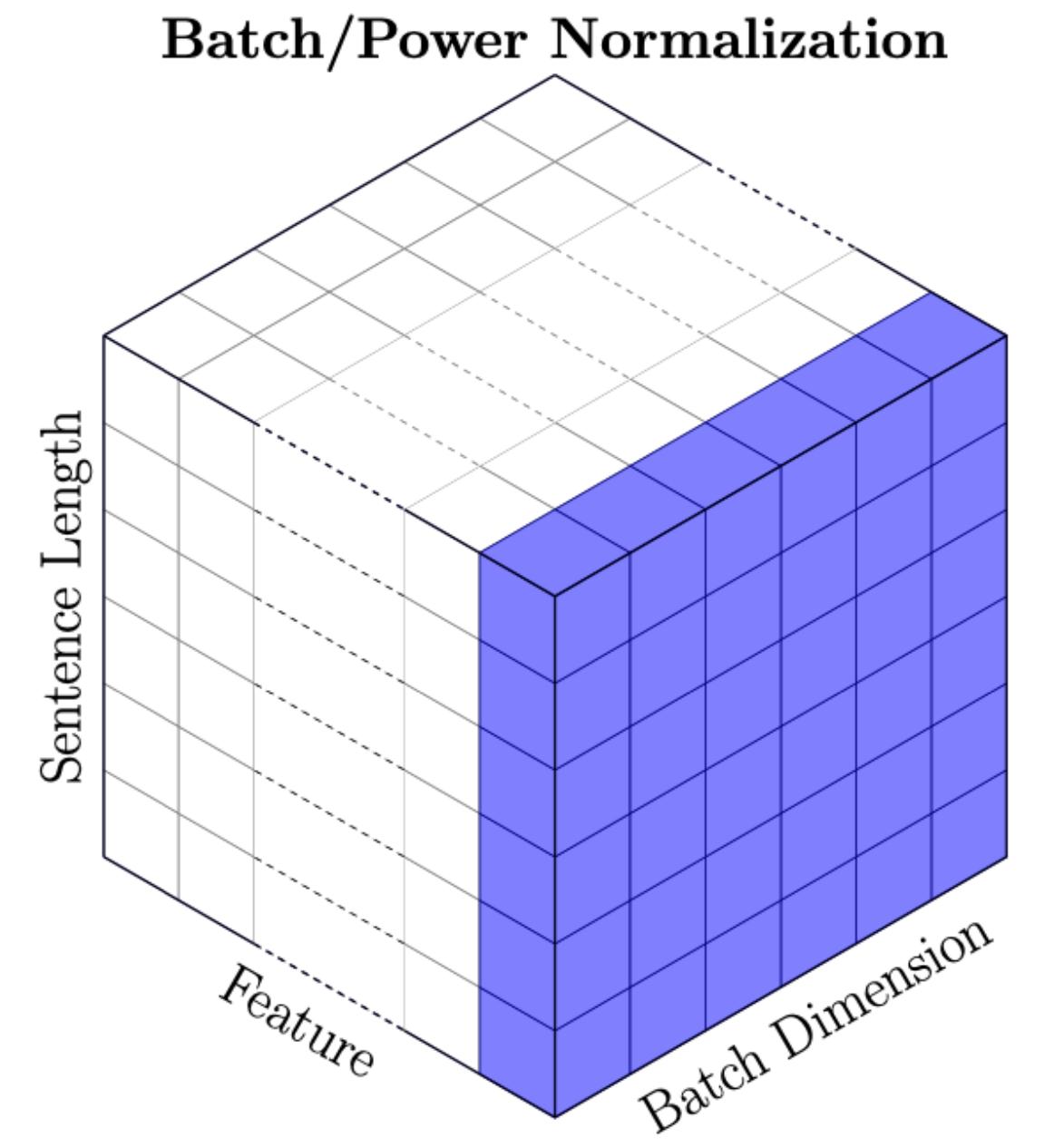


- Tokenize words (word -> tokens)
- Map tokens into embeddings
- Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
- **LayerNorm**
- **Residual connection**
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Layer Normalization (LN)

- Unlike BatchNorm (BN) in CNNs, Transformers use **LayerNorm (LN)**
- LN performs normalization for each token's embedding, then applies a learnable affine transformation

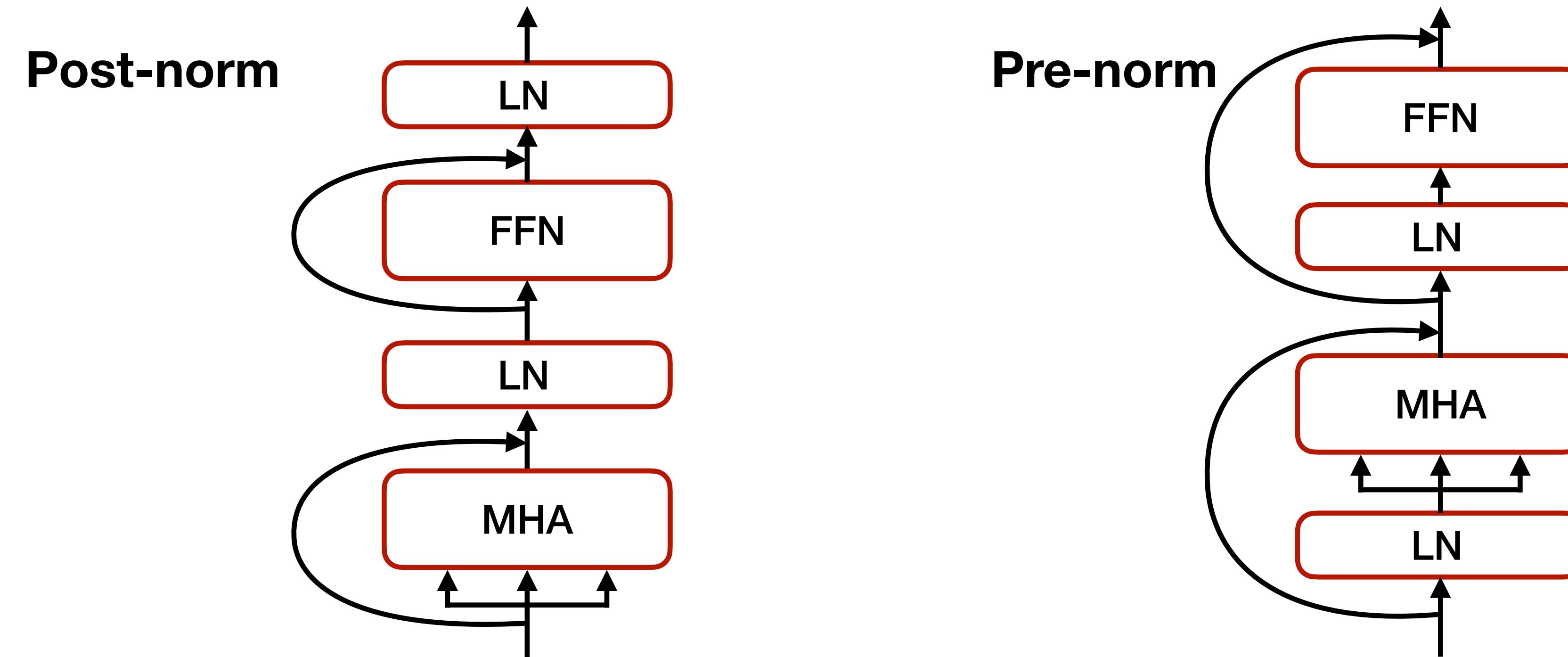


$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

Image credit: <http://proceedings.mlr.press/v119/shen20e/shen20e.pdf>

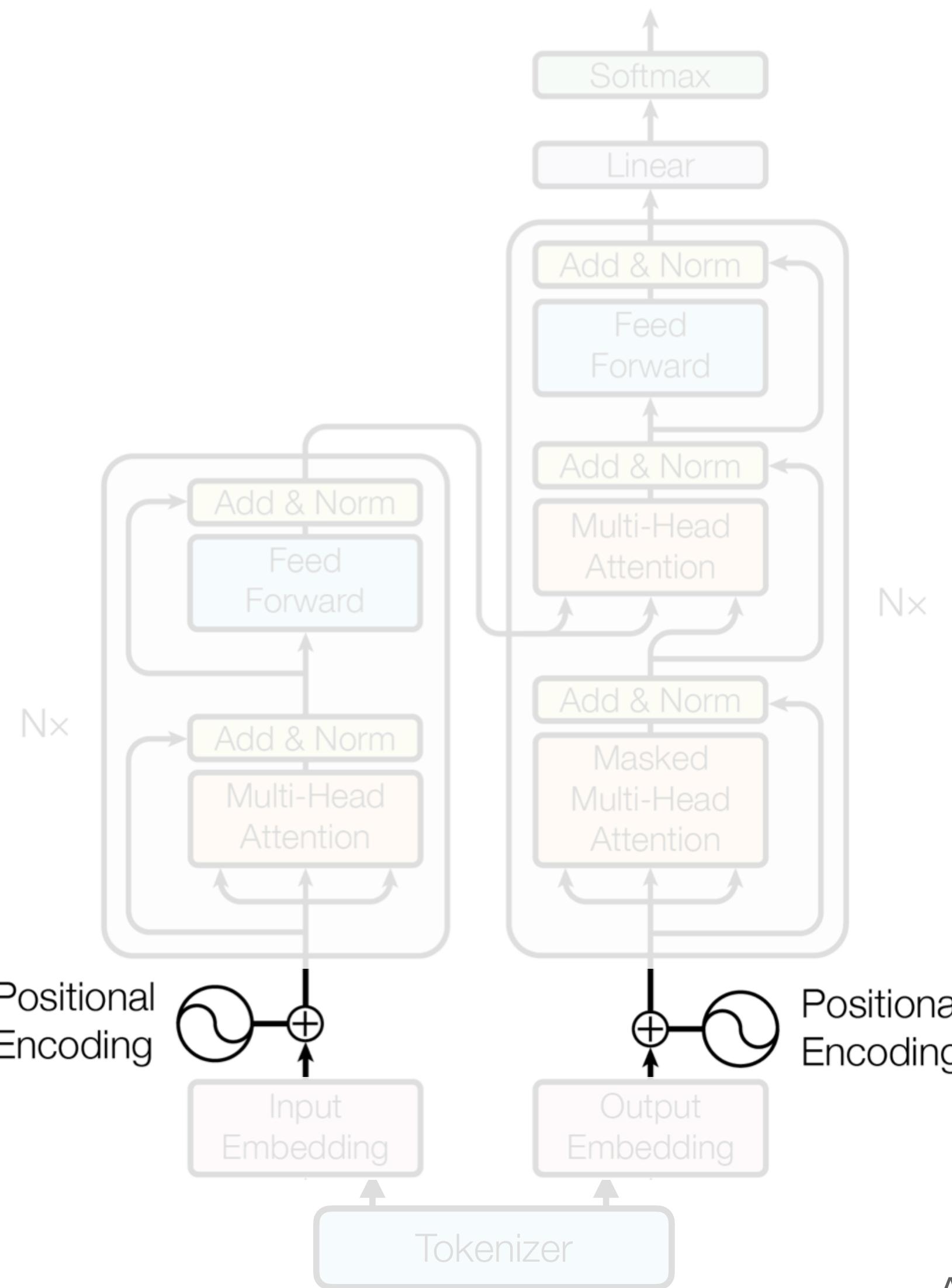
Transformer Block

- Now we put everything together to build a Transformer block
- We add LayerNorm and Residual Connections for training stability
- **Note:** the pre-norm design is more popular now due to better training stability



<https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html>

Transformer



- Tokenize words (word -> tokens)
 - Map tokens into embeddings
 - Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
 - LayerNorm
 - Residual connection
 - **Positional encoding**
 - Final prediction with linear head
-
- **Problem:** attention and FFN do not differentiate the **order** of the input tokens (unlike convolutions)
 - Set encoding (bad) instead of sequence encoding (good)
 - **Solution: Positional encoding!**

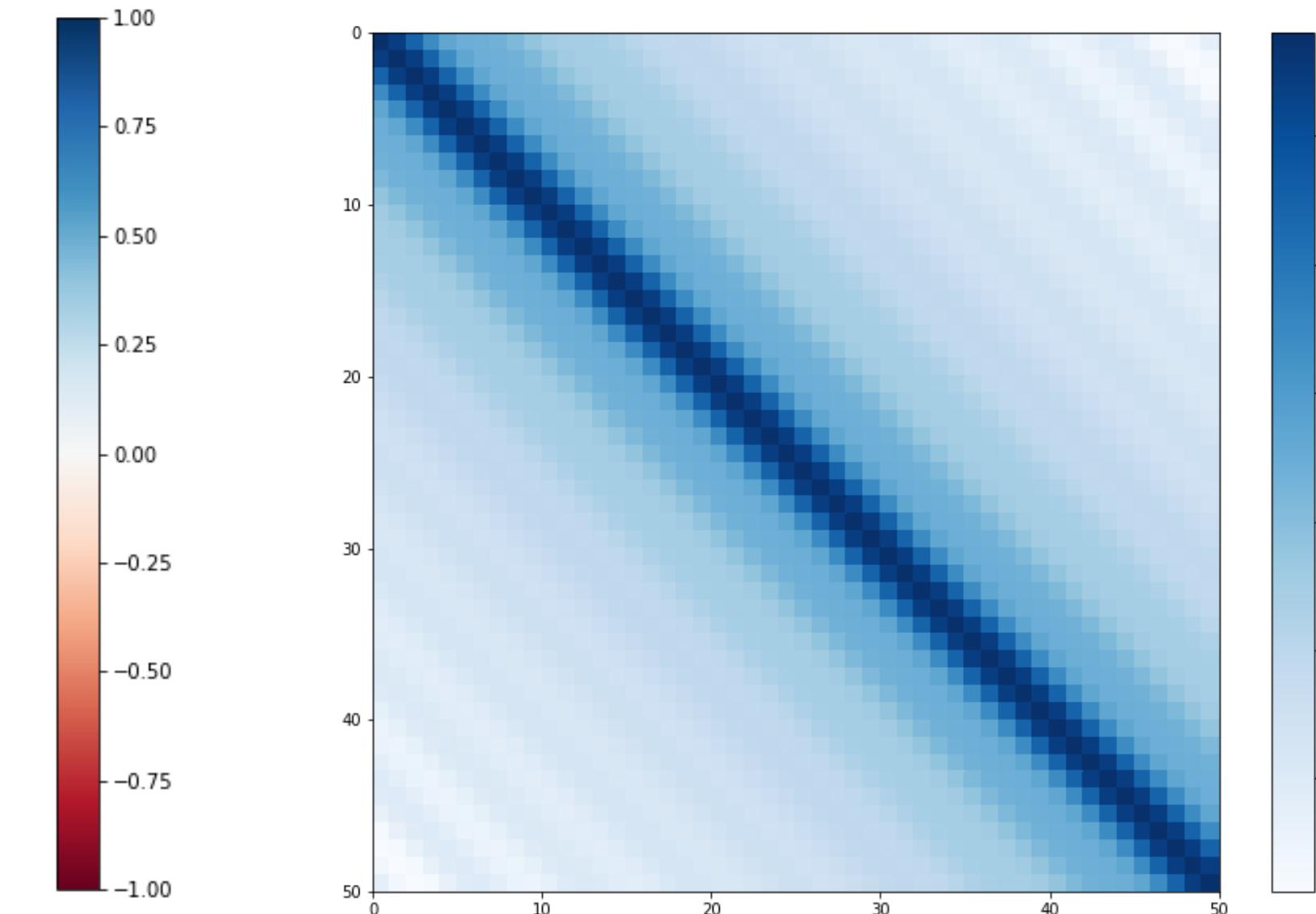
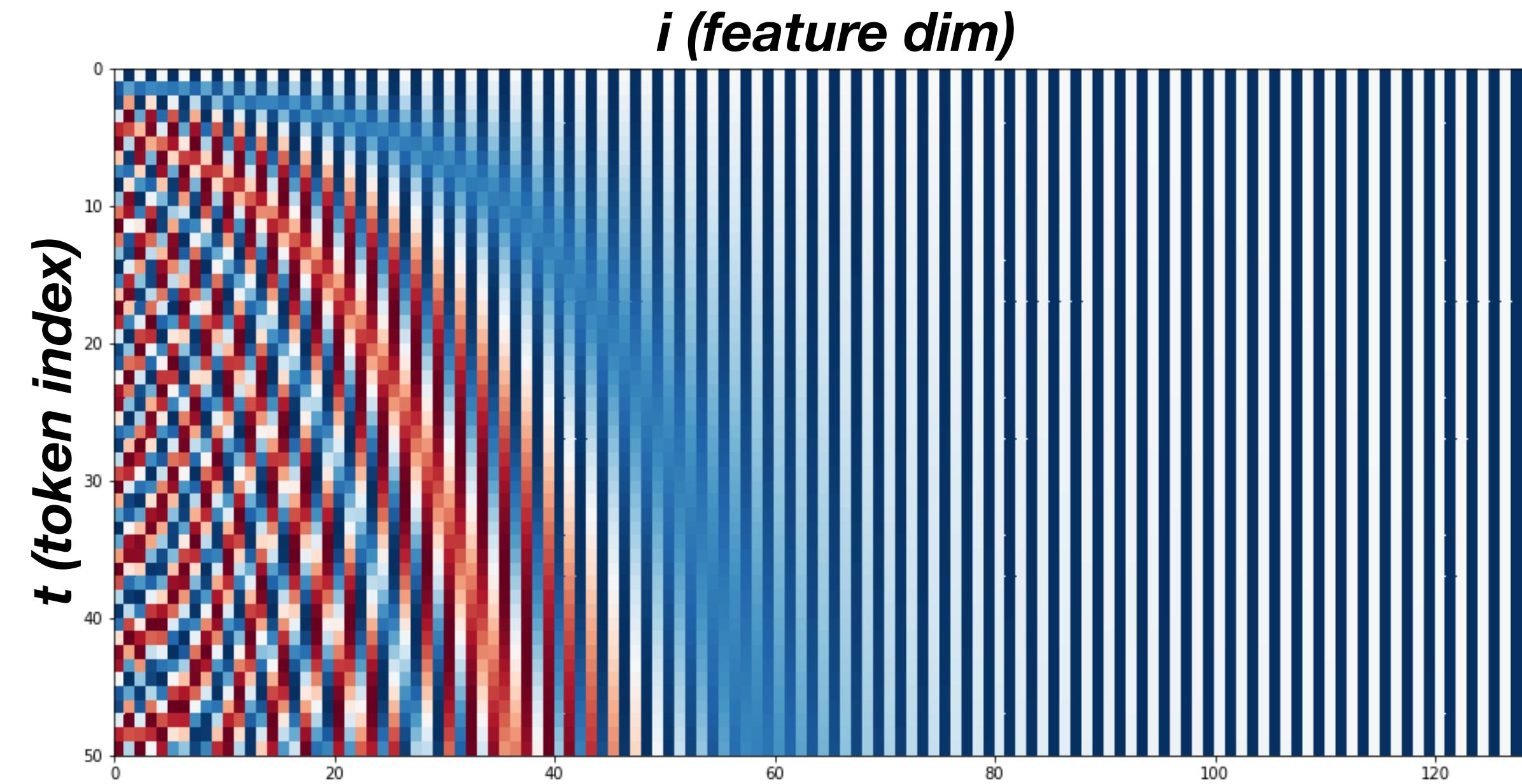
Attention Is All You Need [Vaswani et al., 2017]

Position Encoding (PE)

- Positional encoding provides positional information.
 - **Unique** encoding for each word's position in a sentence.
 - Each element is a function of t (i.e., index, absolute PE)
 - Added to the raw word embedding to fuse in positional information

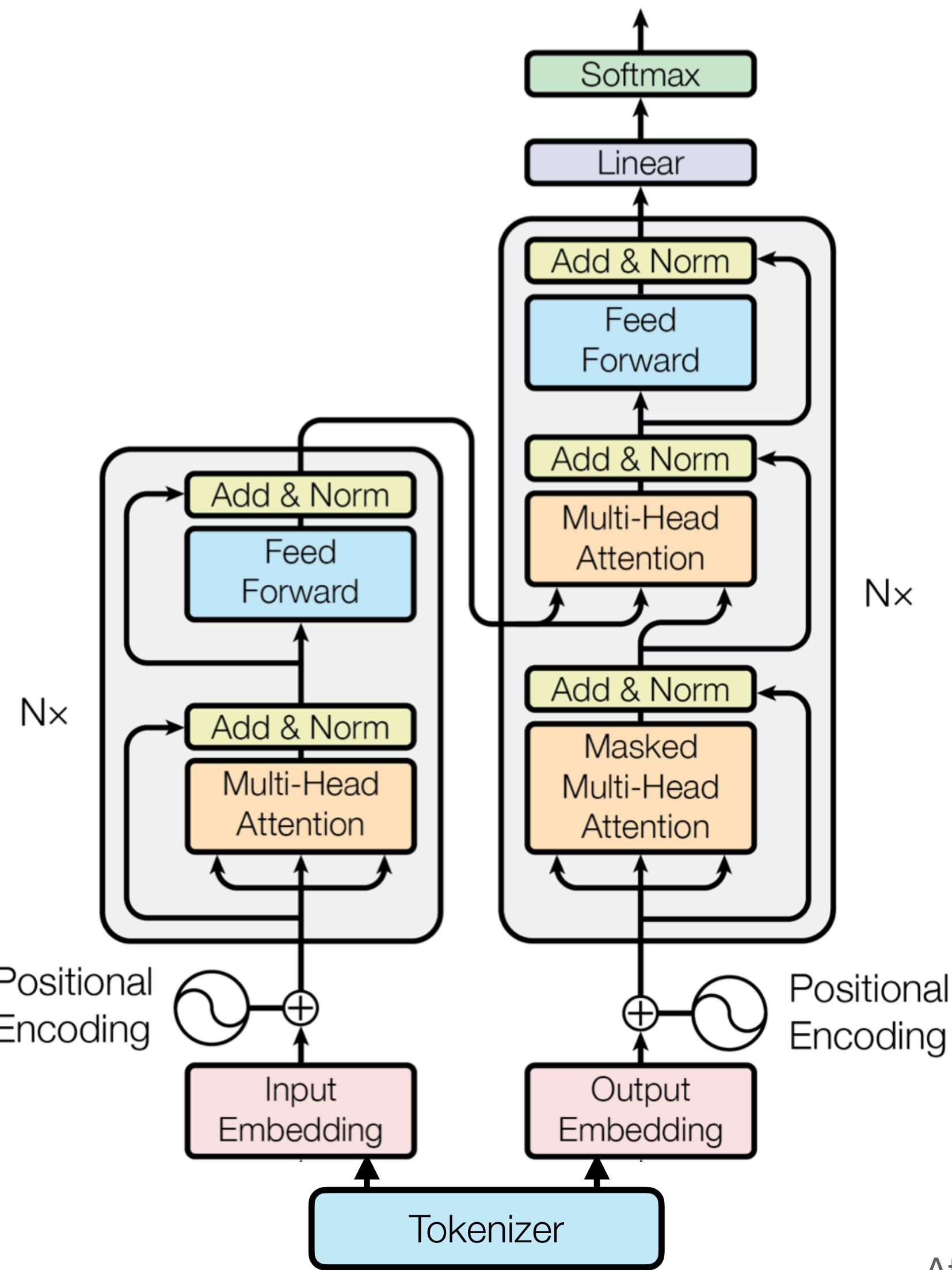
$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$



Content credit: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Putting it together



- Tokenize words (word -> tokens)
- Map tokens into embeddings
- Embeddings go through transformer blocks
 - Multi-Head Attention (MHA)
 - Feed-Forward Network (FFN)
 - LayerNorm
 - Residual connection
- Positional encoding
- Final prediction with linear head

Attention Is All You Need [Vaswani et al., 2017]

Results

Neural Machine Translation (NMT)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

- Transformer surpasses previously published models.
- At a fraction of the training cost.
- (BLEU measures the similarity of the machine-translated text to a set of reference translations).

Attention Is All You Need [Vaswani et al., 2017]

II. Transformer Design Variants

Transformer Design Variants

Improved designs after the initial transformer paper

- Most designs from the initial Transformer paper have been widely used by the community
- Nonetheless, people proposed various alternative designs. For example:
 - Encoder-decoder (T5), encoder-only (BERT), decoder-only (GPT)
 - Absolute positional encoding -> Relative positional encoding
 - KV cache optimizations:
 - Multi-Head Attention (MHA) -> Multi-Query Attention (MQA) -> Grouped-Query Attention (GQA)
 - FFN -> GLU (gated linear unit)

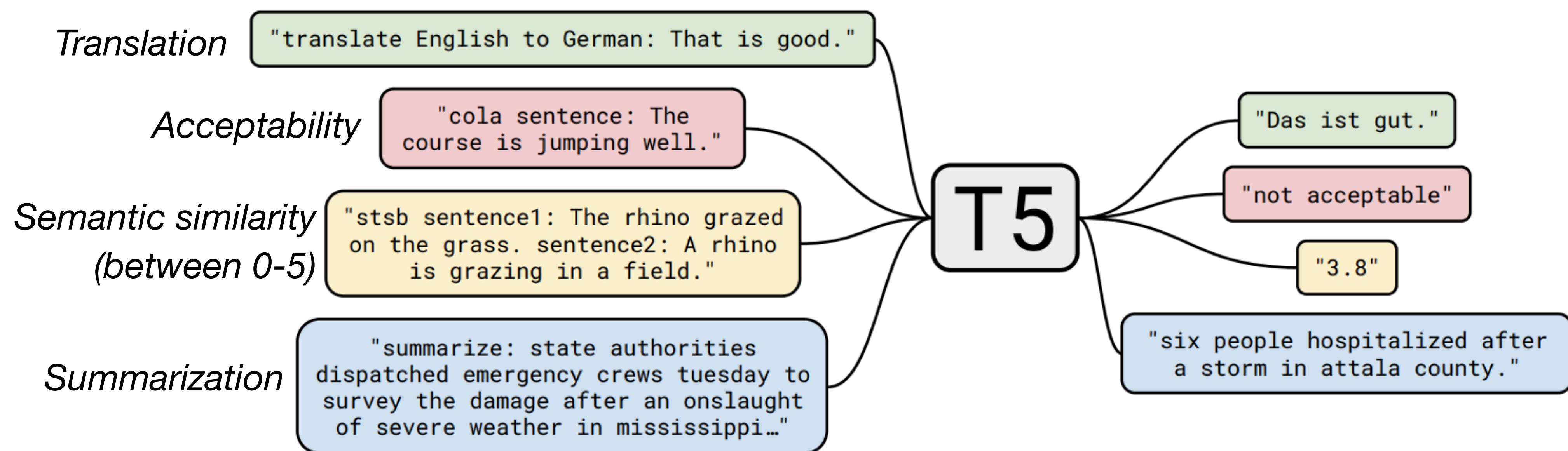
Transformer Design Variants

Improved designs after the initial transformer paper

- Most designs from the initial Transformer paper have been widely used by the community
- Nonetheless, people proposed various alternative designs. For example:
 - Encoder-decoder (T5), encoder-only (BERT), decoder-only (GPT)
 - Absolute positional encoding -> Relative positional encoding
 - KV cache optimizations:
 - Multi-Head Attention (MHA) -> Multi-Query Attention (MQA) -> Grouped-Query Attention (GQA)
 - FFN -> GLU (gated linear unit)

Encoder-Decoder

- The original Transformer is an Encoder-Decoder architecture
- T5 offers a unified text-to-text model for transfer learning on various NLP tasks



Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [Raffel et al., 2019]

Encoder-Decoder

- The original Transformer is an Encoder-Decoder architecture
- T5 offers a unified text-to-text model for transfer learning on various NLP tasks
- The **prompt** is fed to the encoder, the decoder generates the **answer**.

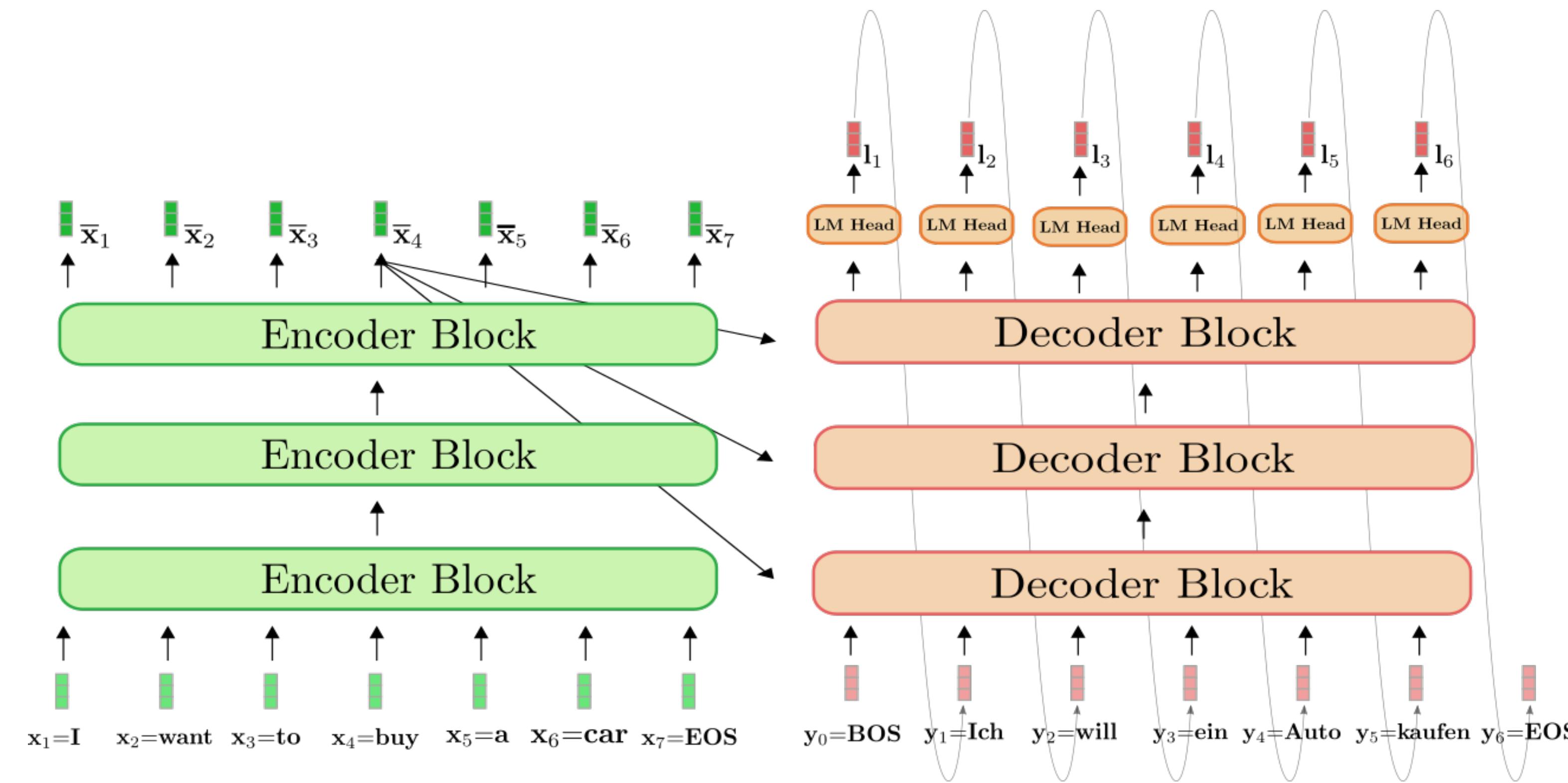


Image credit: <https://huggingface.co/blog/encoder-decoder>

Encoder-only (BERT)

Bidirectional Encoder Representations from Transformers (BERT)

- BERT is an **encoder-only** language model
- Two pre-training objectives
 - Task #1: **Masked Language Model (MLM)**
 - Mask some percentage (15%) of the input tokens at random; train model to predict masked tokens
 - Task #2: **Next Sentence Prediction (NSP)**
 - Whether sentence B is the next sentence of sentence A (less used)
- The pre-trained model is fine-tuned to downstream tasks

Attention mask:

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

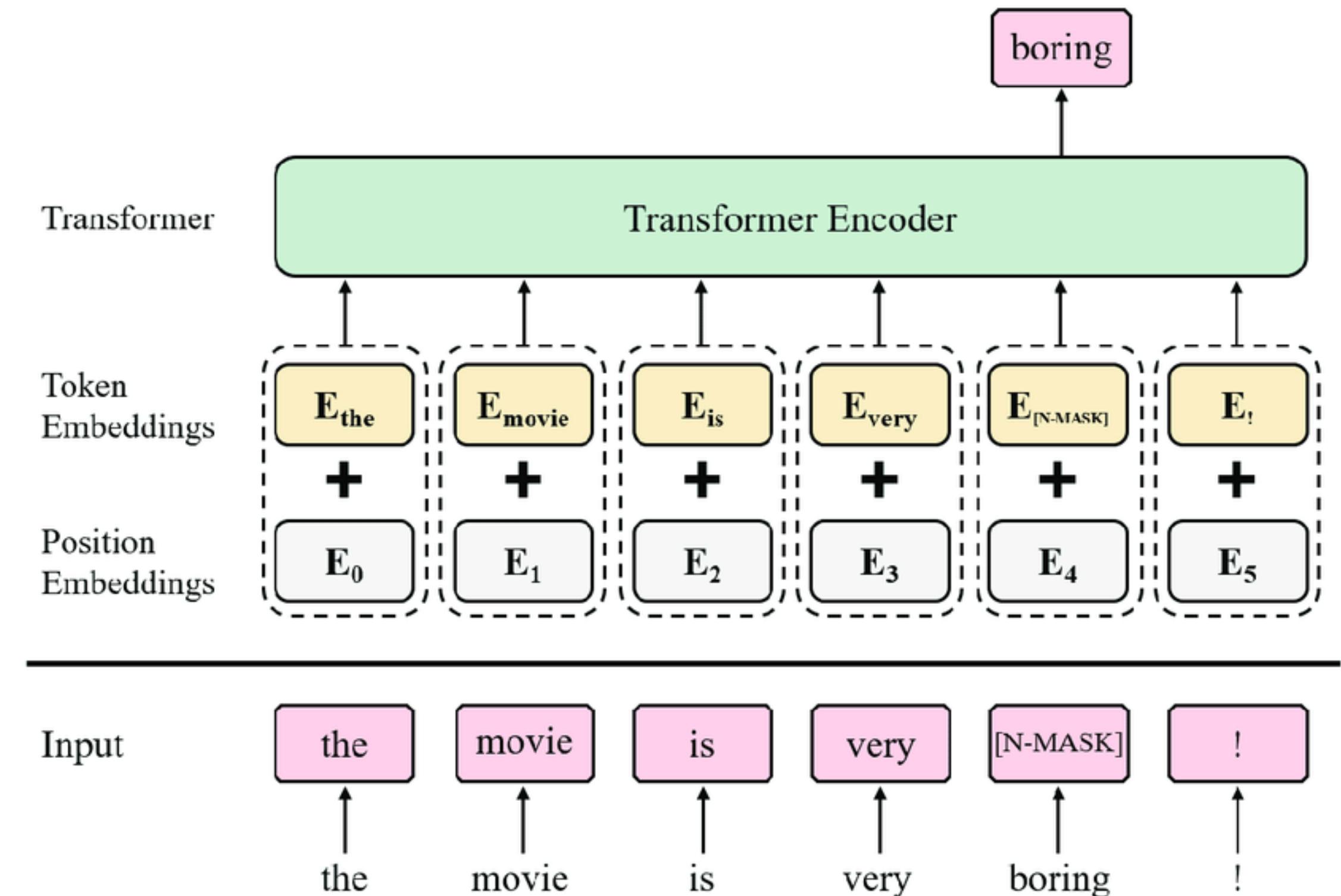


Image credit: <https://www.mdpi.com/2073-8994/11/11/1393/htm>

BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding [Devlin et al., 2018]

Decoder-only (GPT)

Generative Pre-trained Transformer (GPT)

- GPT is a **decoder-only** language model
- Pre-training objective:
 - Next word prediction

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$$

- For smaller models (GPT-2), the pre-trained model is fine-tuned to downstream tasks
- Larger model can run in zero-shot/few-shot

Attention mask:

1	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1

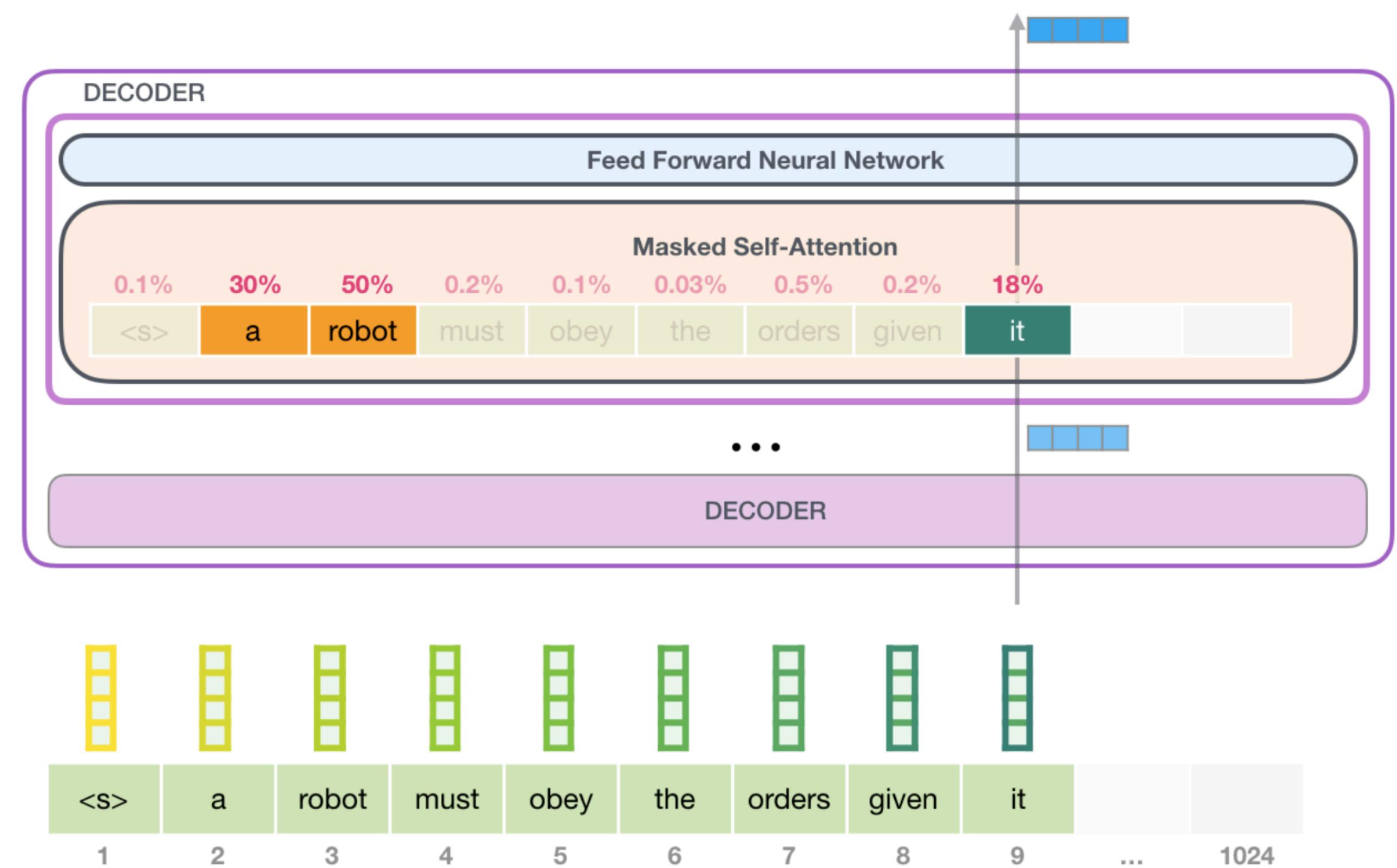


Image credit: <https://jalammar.github.io/illustrated-gpt2/>

Transformer Design Variants

Improved designs after the initial transformer paper

- Most designs from the initial Transformer paper have been widely used by the community
- Nonetheless, people proposed various alternative designs. For example:
 - Encoder-decoder (T5), encoder-only (BERT), decoder-only (GPT)
 - Absolute positional encoding -> Relative positional encoding
 - KV cache optimizations:
 - Multi-Head Attention (MHA) -> Multi-Query Attention (MQA) -> Grouped-Query Attention (GQA)
 - FFN -> GLU (gated linear unit)

Absolute / Relative Positional Encoding

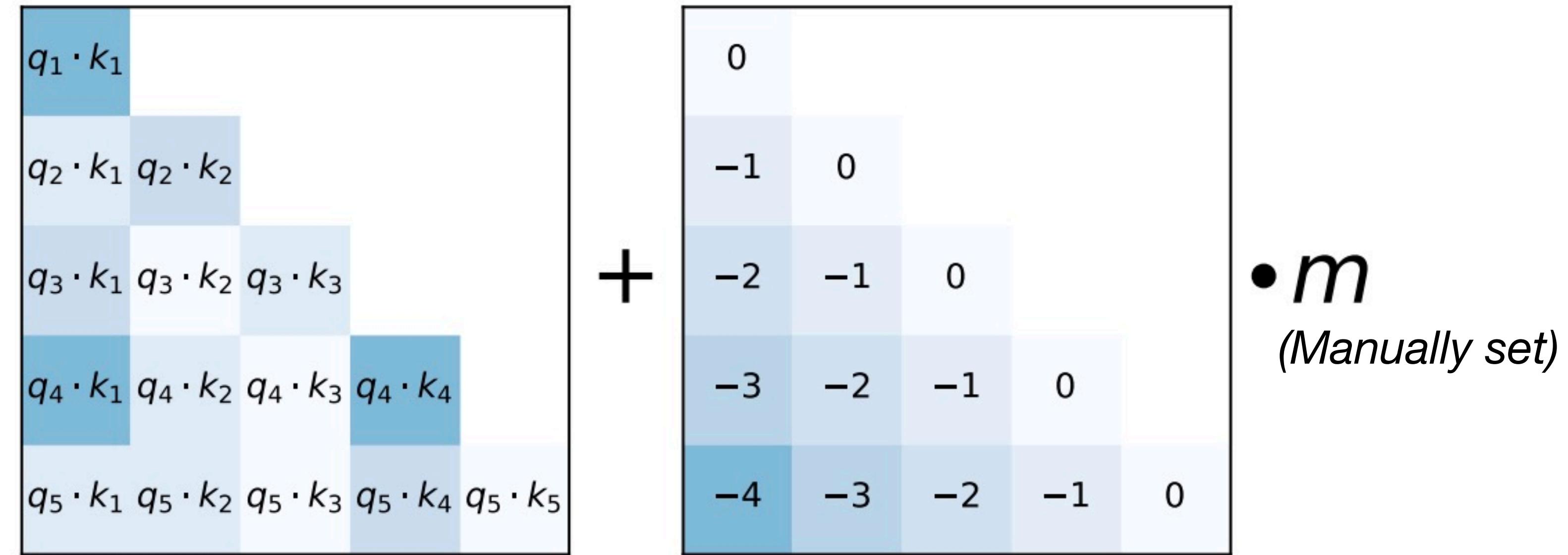
Comparing absolute/relative positional encoding

- **Absolute positional encoding** fuses the positional information into the input embeddings (thus Q/K/V). The information is propagated through the entire Transformer.
- **Relative positional encoding** provides relative distance information by impacting the attention scores (either adding a bias or modifying queries and keys), not V.
 - **Advantage:** generalize to sequence length not seen during training, i.e., train short, test long (does not always apply)

Relative Positional Encoding

Attention with Linear Biases (ALiBi)

- The idea is to change the positional encoding to be only related to **relative distance**, but not the **absolute index**, with the hope that it can be “train short, test long”.
- ALiBi proposed to **add an offset to the attention matrix** (defined as the relative distance) instead of adding it to the input token embeddings.



Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation [Press et al., 2021]

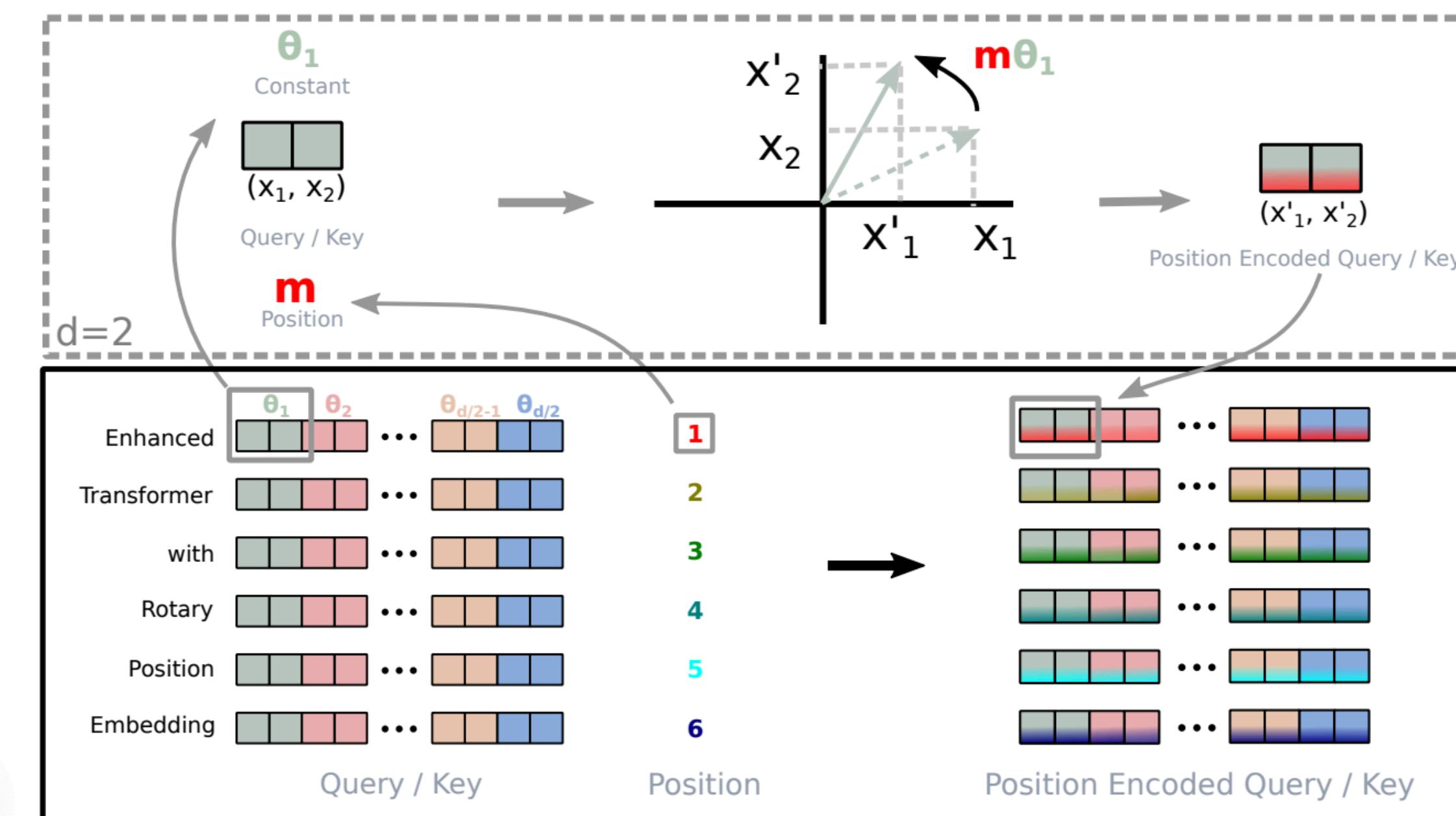
Relative Positional Encoding

Rotary Positional Embedding (RoPE)

- A popular implementation for relative positional embedding (used in LLaMA)
- Rotate the embeddings in 2D space:
 - Split an embedding of dimension d into $d/2$ pairs, each pair considered as a 2D coordinate
 - Apply rotation according to the position m

We need a big enough number to distinguish more tokens

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$$



$$\begin{aligned} \text{RoPE}(x, m) &= xe^{mi\varepsilon} \\ \langle \text{RoPE}(q_j, m), \text{RoPE}(k_j, n) \rangle &= \langle q_j e^{mi\varepsilon}, k_j e^{ni\varepsilon} \rangle \\ &= q_j k_j e^{mi\varepsilon} \overline{e^{ni\varepsilon}} \\ &= q_j k_j e^{(m-n)i\varepsilon} \\ &= \text{RoPE}(q_j k_j, m - n) \end{aligned}$$

- The phase angle of the inner product of two complex vectors is the phase difference between the two complex vectors (thus $m-n$)

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al., 2021]

Relative Positional Encoding

Rotary Positional Embedding (RoPE)

- A popular implementation for relative positional embedding (used in LLaMA)
- Rotate the embeddings in 2D space:
 - Split an embedding of dimension d into $d/2$ pairs, each pair considered as a 2D coordinate
 - Apply rotation according to the position m
- The general form:

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}_{\Theta,m}^d \mathbf{W}_{\{q,k\}} \mathbf{x}_m$$

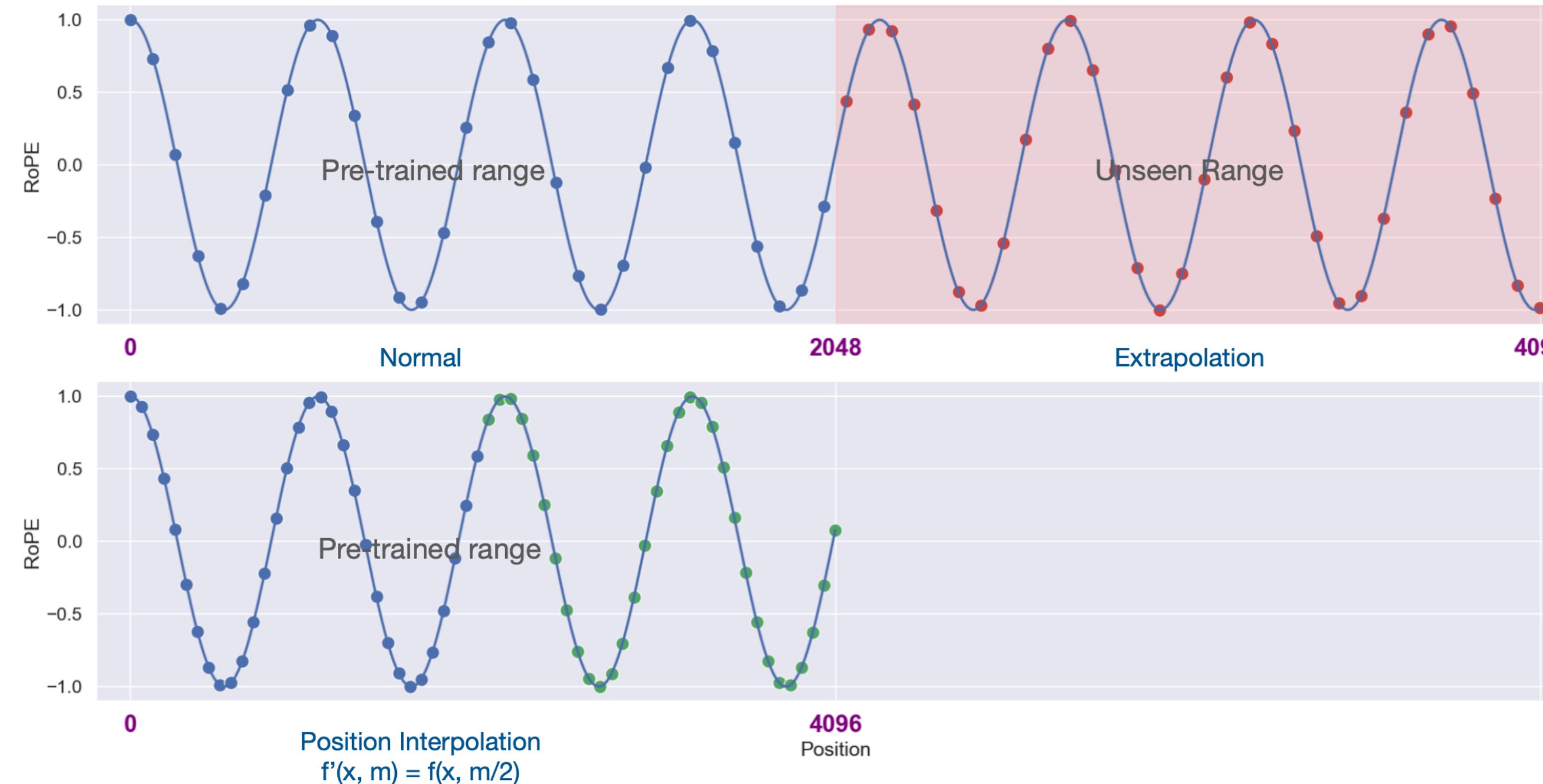
$$\mathbf{R}_{\Theta,m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots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

RoFormer: Enhanced Transformer with Rotary Position Embedding [Su et al., 2021]

Relative Positional Embedding

Advantage of RoPE: extending the context window

- LLM is usually trained with a context length constraint (e.g., 2k for LLaMA, 4k for Llama-2, 8k for GPT-4) and fails with a larger context length
- We can extend the context length support by **interpolating** RoPE PE (i.e., use a smaller θ_i)
- Extend the context length of LLaMA from 2k to 32k



$m \in [0, 2048 * 2], \quad \theta'_i = \theta_i$ X

$m \in [0, 2048 * 2], \quad \theta'_i = \theta_i/2$ ✓

Extending Context Window of Large Language Models via Position Interpolation [Chen et al., 2023]

Transformer Design Variants

Improved designs after the initial transformer paper

- Most designs from the initial Transformer paper have been widely used by the community
- Nonetheless, people proposed various alternative designs. For example:
 - Encoder-decoder (T5), encoder-only (BERT), decoder-only (GPT)
 - Absolute positional encoding -> Relative positional encoding
 - KV cache optimizations:
 - Multi-Head Attention (MHA) -> Multi-Query Attention (MQA) -> Grouped-Query Attention (GQA)
 - FFN -> GLU (gated linear unit)

KV Cache Optimizations

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **Keys** and **Values** of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
 - Only need the **current** query token

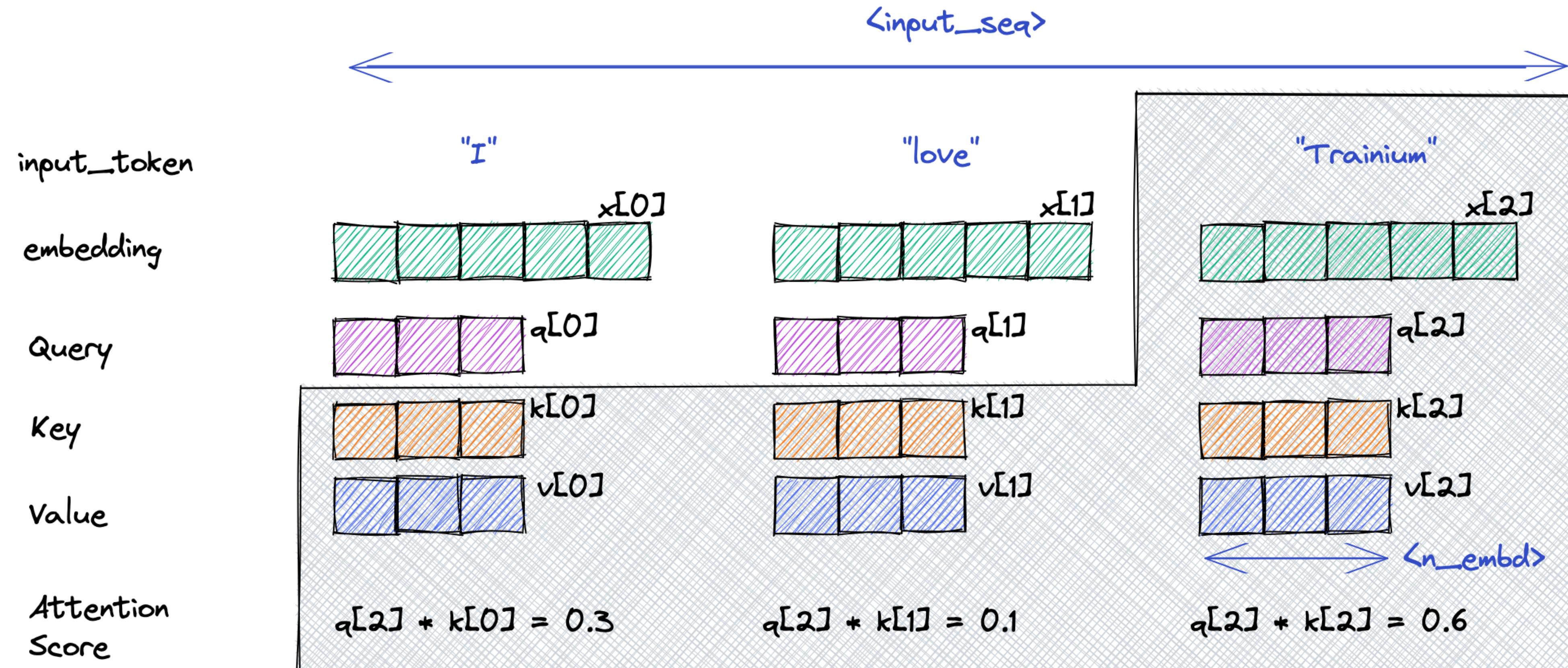
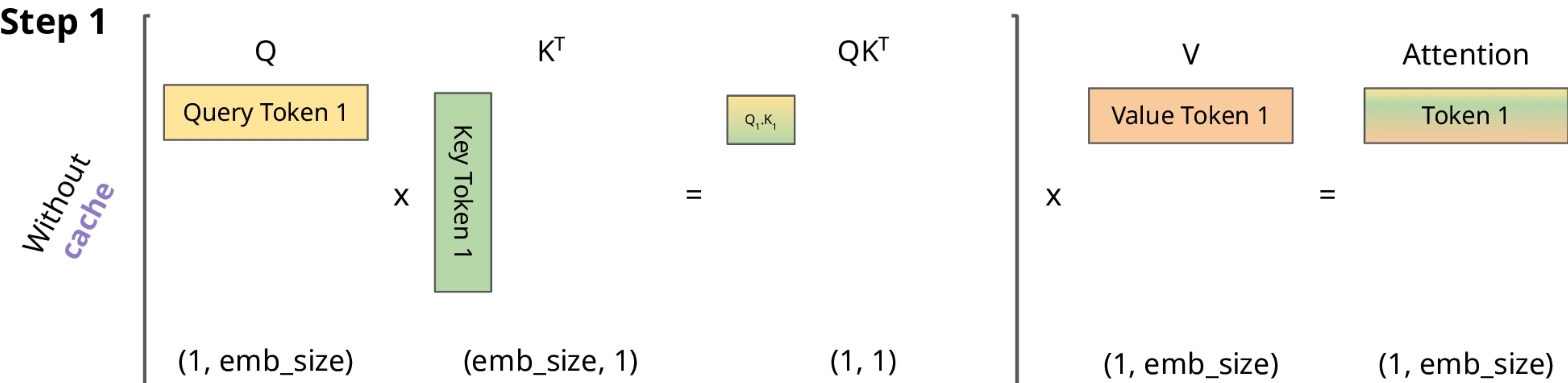


Image credit: <https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/appnotes/transformers-neuronx/generative-llm-inference-with-neuron.html>

KV Cache Optimizations

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **Keys** and **Values** of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
 - Only need the **current** query token



KV Cache Optimizations

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **Keys** and **Values** of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
 - Only need the **current** query token

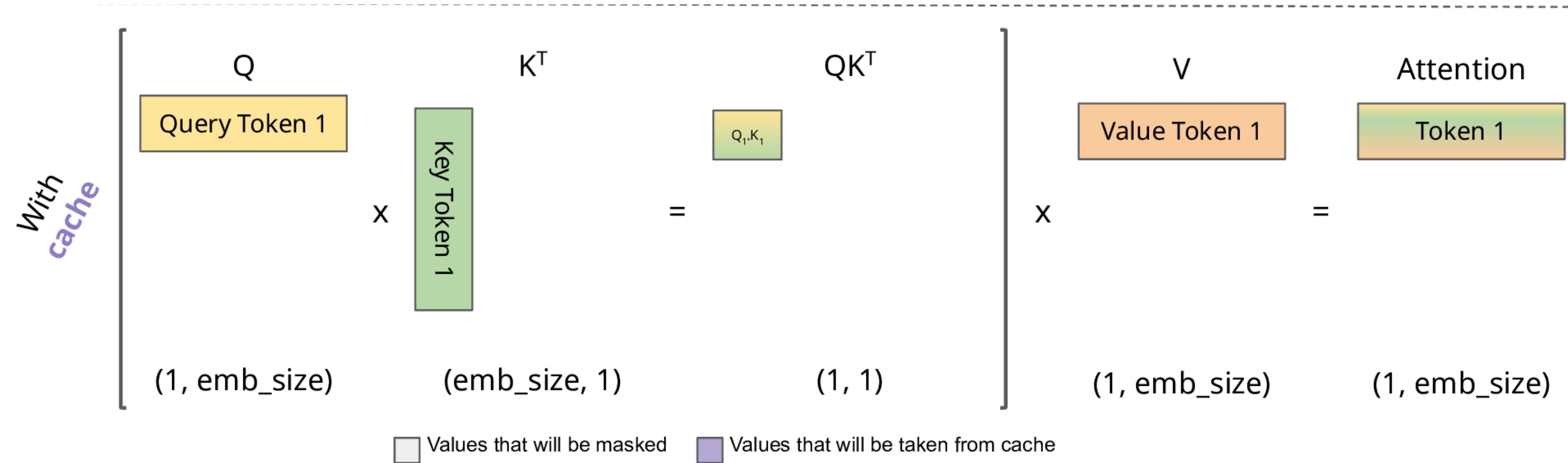


Image credit: <https://medium.com/@joaolages/kv-caching-explained-276520203249>

KV Cache Optimizations

The KV cache could be large with long context

- We can calculate the memory required to store the KV cache:
- Llama-2-7B, KV cache size =

$$\underbrace{BS}_{minibatch} * \underbrace{32}_{layers} * \underbrace{32}_{heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \overbrace{2\text{bytes}}^{\text{fp16}} = 512\text{KB} \times BS \times N$$

- Llama-2-13B, KV cache size =

$$\underbrace{BS}_{minibatch} * \underbrace{40}_{layers} * \underbrace{40}_{heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \overbrace{2\text{bytes}}^{\text{fp16}} = 800\text{KB} \times BS \times N$$

- Llama-2-70B (if using MHA), KV cache size =

$$\underbrace{BS}_{minibatch} * \underbrace{80}_{layers} * \underbrace{64}_{kv-heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \overbrace{2\text{bytes}}^{FP16} = 2.5\text{MB} \times BS \times N$$

KV Cache Optimizations

The KV cache could be large with long context

- We can calculate the memory required to store the KV cache
- Consider Llama-2-70B (if using MHA), KV cache requires

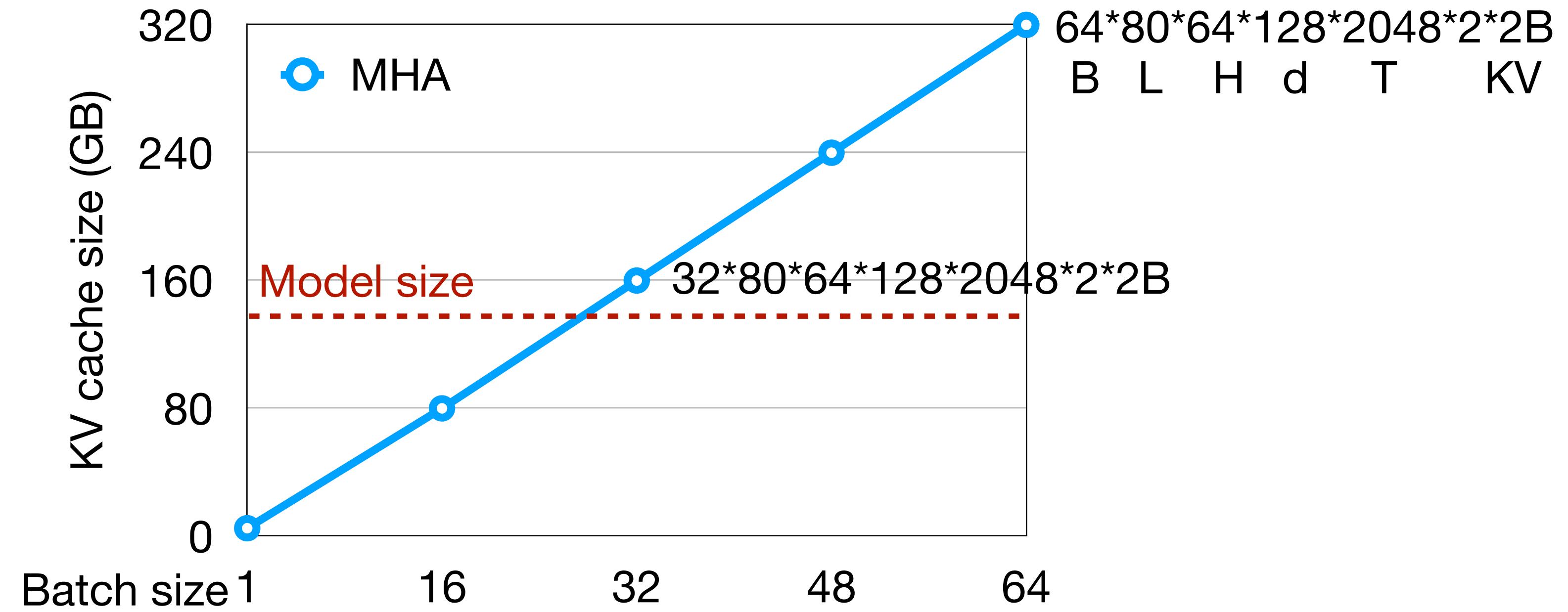
$$\underbrace{BS}_{minibatch} * \underbrace{80}_{layers} * \underbrace{64}_{kv-heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \overbrace{2\text{bytes}}^{\text{FP16}} = 2.5\text{MB} \times BS \times N$$

- bs=1, n_seq=512: 1.25GB
- bs=1, n_seq=4096 : 10GB (~ a paper)
- bs=16, n_seq=4096: 160GB (requires two A100 GPUs!)

KV Cache Optimizations

The KV cache could be large with long context

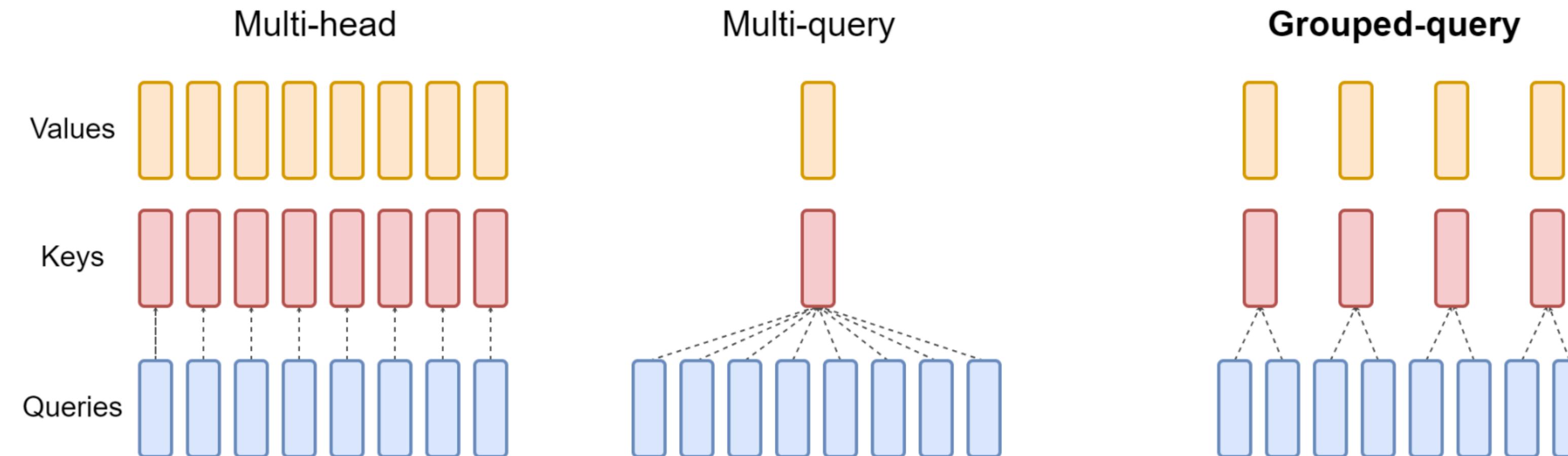
- Now, we calculate the KV cache size under $N = 2048$ different batch sizes
 - The KV cache size goes quickly larger than the model weights



Multi-Query Attention

Reduce the KV cache memory usage with MQA/GQA

- Reducing the KV cache size by reducing #kv-heads
 - Multi-head attention (MHA): N heads for query, N heads for key/value
 - Multi-query attention (MQA): N heads for query, 1 heads for key/value
 - Grouped-query attention (GQA): N heads for query, G heads for key/value (typically $G = N/8$)



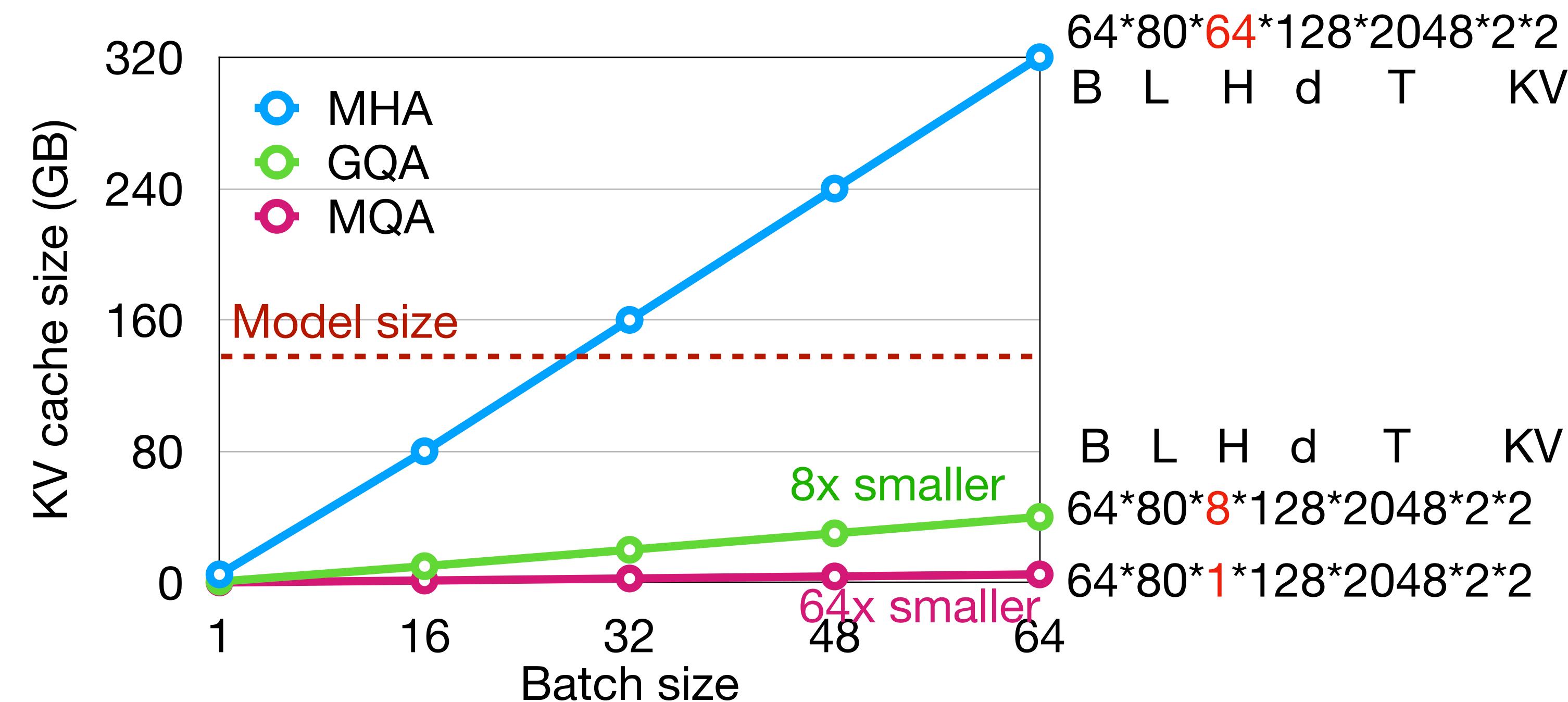
Fast Transformer Decoding: One Write-Head is All You Need [Shazeer et al., 2019]

GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints [Ainslie et al., 2023]

Multi-Query Attention

Reduce the KV cache memory usage with MQA/GQA

- Reducing the KV cache size by reducing #kv-heads
 - Multi-head attention (MHA): N heads for query, N heads for key/value
 - Multi-query attention (MQA): N heads for query, 1 heads for key/value
 - Grouped-query attention (GQA): N heads for query, G heads for key/value (typically $G = N/8$)



Multi-Query Attention

Reduce the KV cache memory usage with MQA/GQA

- Reducing the KV cache size by reducing #kv-heads
 - Multi-head attention (MHA): N heads for query, N heads for key/value
 - Multi-query attention (MQA): N heads for query, 1 heads for key/value
 - Grouped-query attention (GQA): N heads for query, G heads for key/value (typically $G = N/8$)
- GQA matches the accuracy of MHA under a large model size

	BoolQ	PIQA	SIQA	Hella-Swag	ARC-e	ARC-c	NQ	TQA	MMLU	GSM8K	Human-Eval
MHA	71.0	79.3	48.2	75.1	71.2	43.0	12.4	44.7	28.0	4.9	7.9
MQA	70.6	79.0	47.9	74.5	71.6	41.9	14.5	42.8	26.5	4.8	7.3
GQA	69.4	78.8	48.6	75.4	72.1	42.5	14.0	46.2	26.9	5.3	7.9

Llama-2, 30B, 150B tokens

Transformer Design Variants

Improved designs after the initial transformer paper

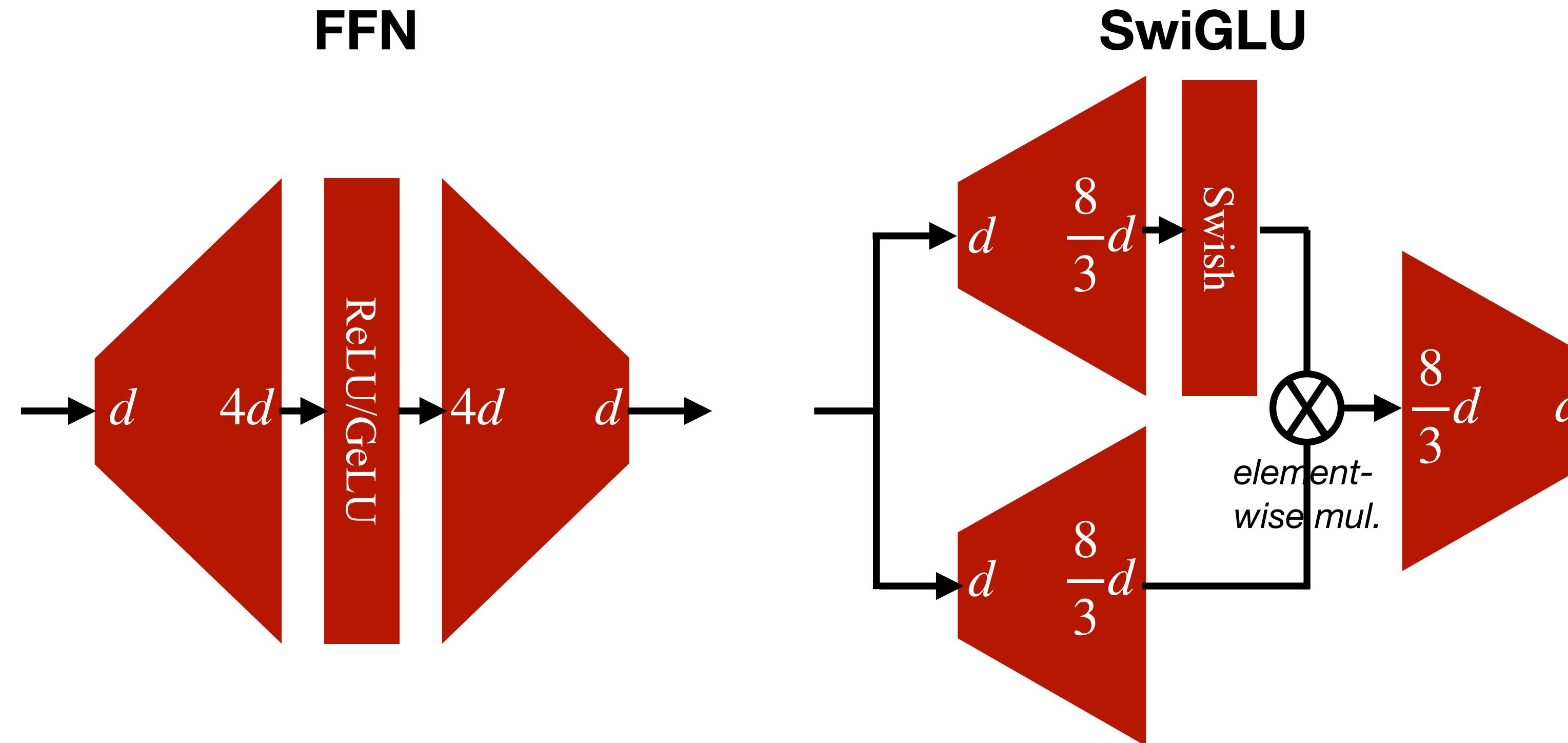
- Most designs from the initial Transformer paper have been widely used by the community
- Nonetheless, people proposed various alternative designs. For example:
 - Encoder-decoder (T5), encoder-only (BERT), decoder-only (GPT)
 - Absolute positional encoding -> Relative positional encoding
 - KV cache optimizations:
 - Multi-Head Attention (MHA) -> Multi-Query Attention (MQA) -> Grouped-Query Attention (GQA)
 - FFN -> GLU (gated linear unit)

Gated Linear Units (GLU)

Improving over FFN

- Replacing the vanilla FFN with GLU improves Transformers

$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2$$



PPL results

Training Steps	65,536	524,288
FFN _{ReLU} (baseline)	1.997 (0.005)	1.677
FFN _{GELU}	1.983 (0.005)	1.679
FFN _{Swish}	1.994 (0.003)	1.683
FFN _{GLU}	1.982 (0.006)	1.663
FFN _{Bilinear}	1.960 (0.005)	1.648
FFN _{GEGLU}	1.942 (0.004)	1.633
FFN _{SwiGLU}	1.944 (0.010)	1.636
FFN _{ReGLU}	1.953 (0.003)	1.645

GLU Variants Improve Transformer [Shazeer, 2020]

Gated Linear Units (GLU)

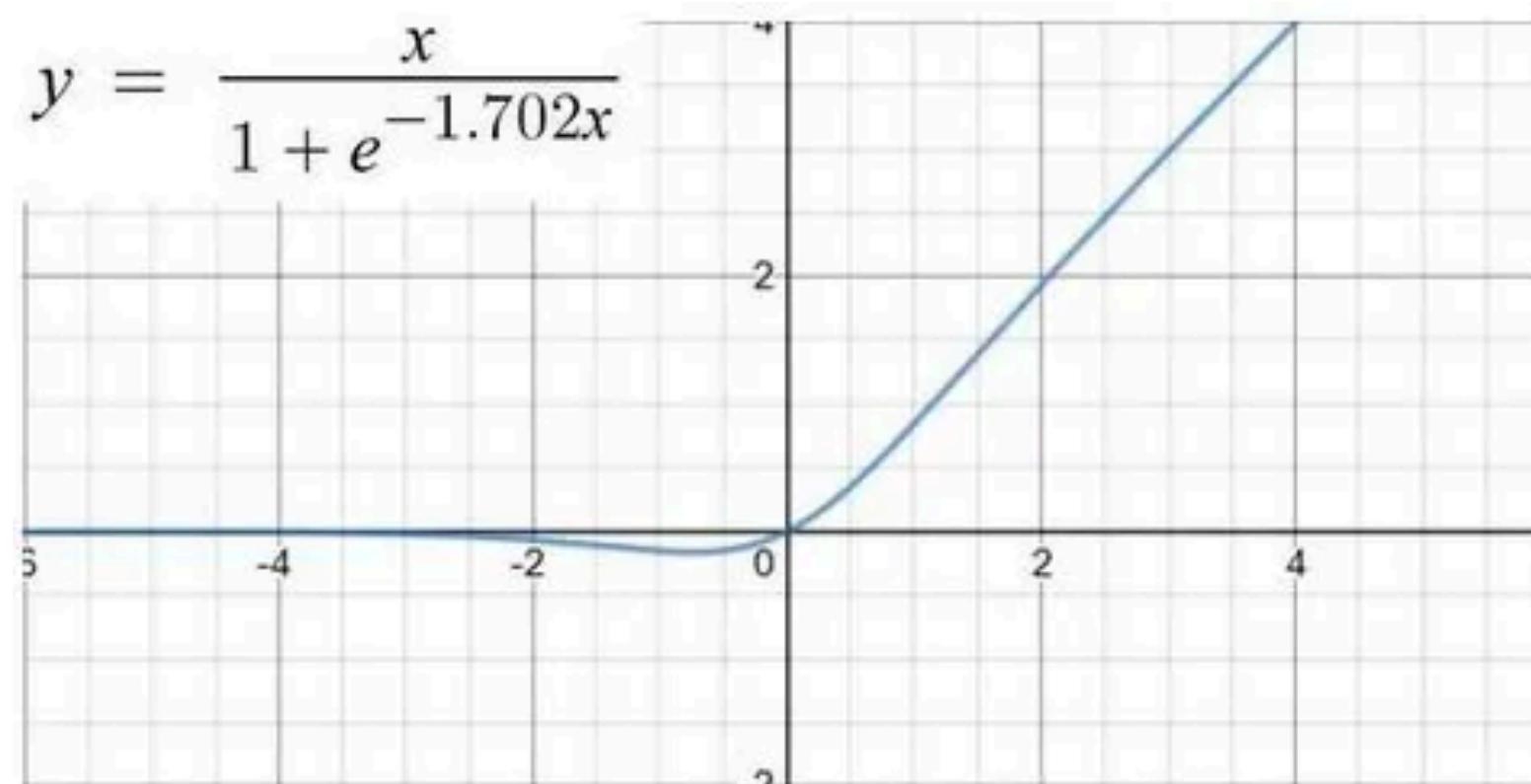
Improving over FFN

- Replacing the vanilla FFN with GLU improves Transformers

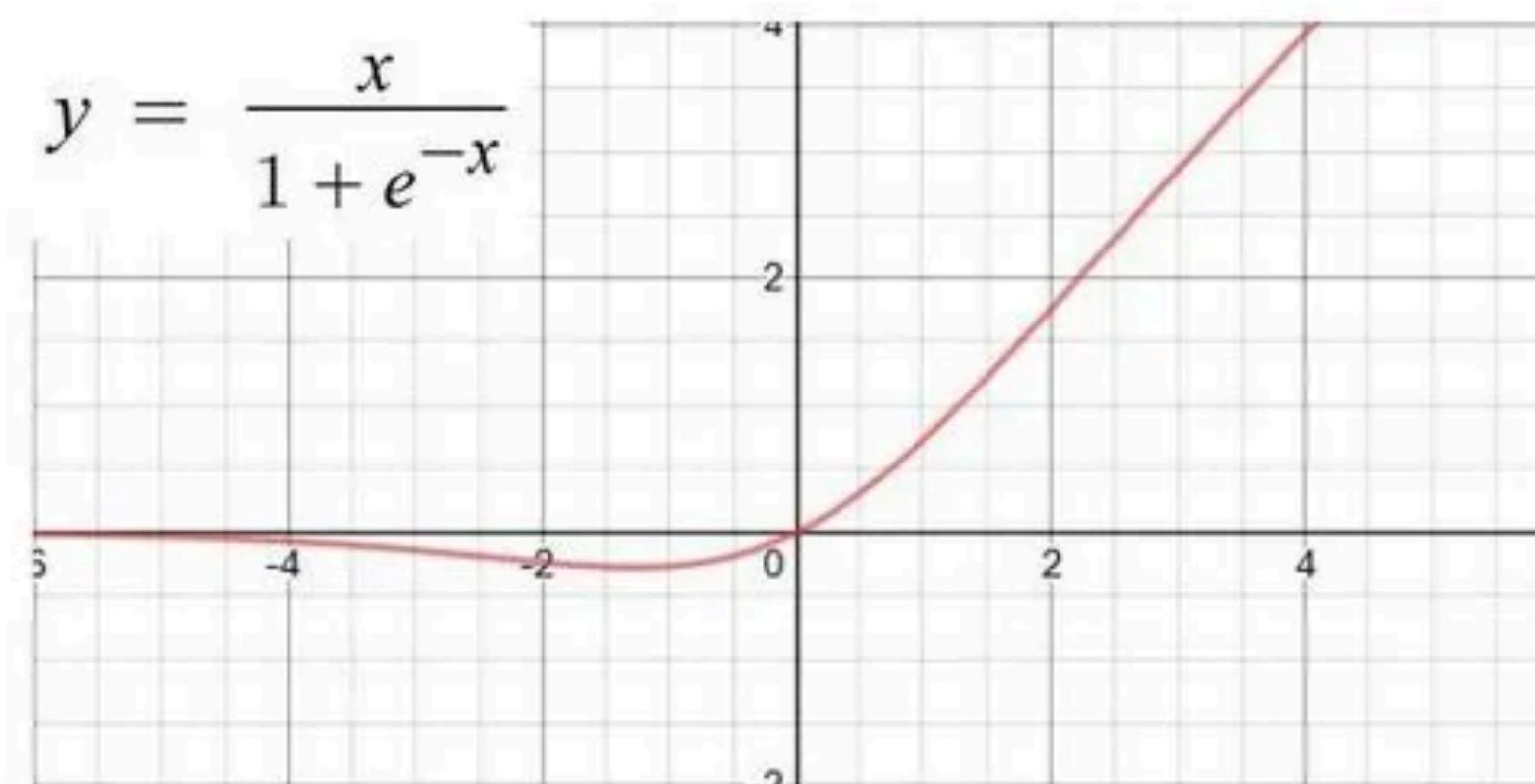
$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2$$

- Equation of activation functions

GeLU



Swish



PPL results

Training Steps	65,536	524,288
FFN _{ReLU} (baseline)	1.997 (0.005)	1.677
FFN _{GELU}	1.983 (0.005)	1.679
FFN _{Swish}	1.994 (0.003)	1.683
FFN _{GLU}	1.982 (0.006)	1.663
FFN _{Bilinear}	1.960 (0.005)	1.648
FFN _{GEGLU}	1.942 (0.004)	1.633
FFN _{SwiGLU}	1.944 (0.010)	1.636
FFN _{ReGLU}	1.953 (0.003)	1.645

Image credit: <https://towardsdatascience.com/on-the-disparity-between-swish-and-gelu-1ddde902d64b>

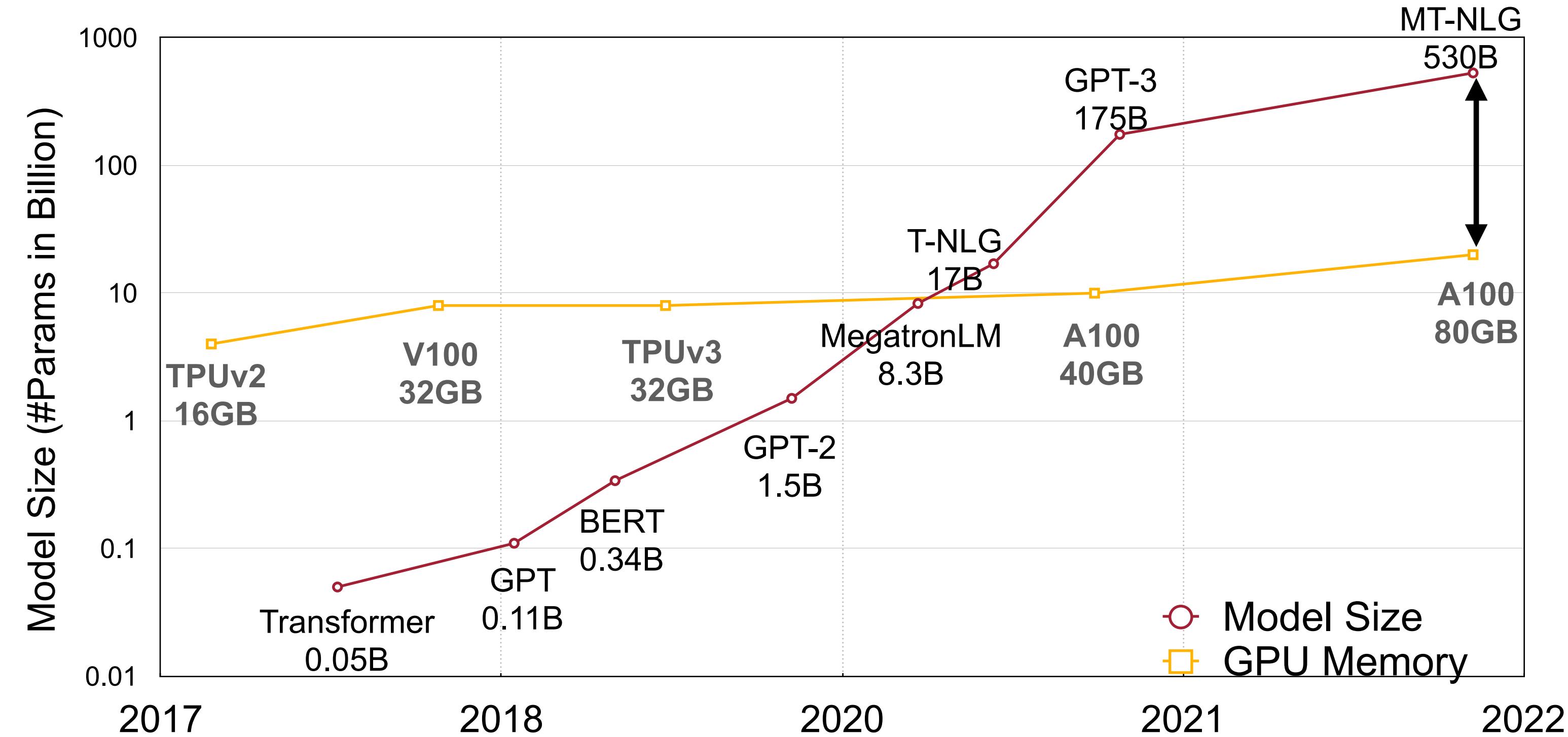
Searching for Activation Functions [Ramachandran et al., 2020]
Gaussian Error Linear Units (GELUs) [Hendrycks and Gimpel, 2016]

III. Large language models (LLMs)

Large Language Models (LLMs)

The growing trend of scaling up

- LLMs are scaled-up transformers models trained on large language corpus (natural language, code, etc.)

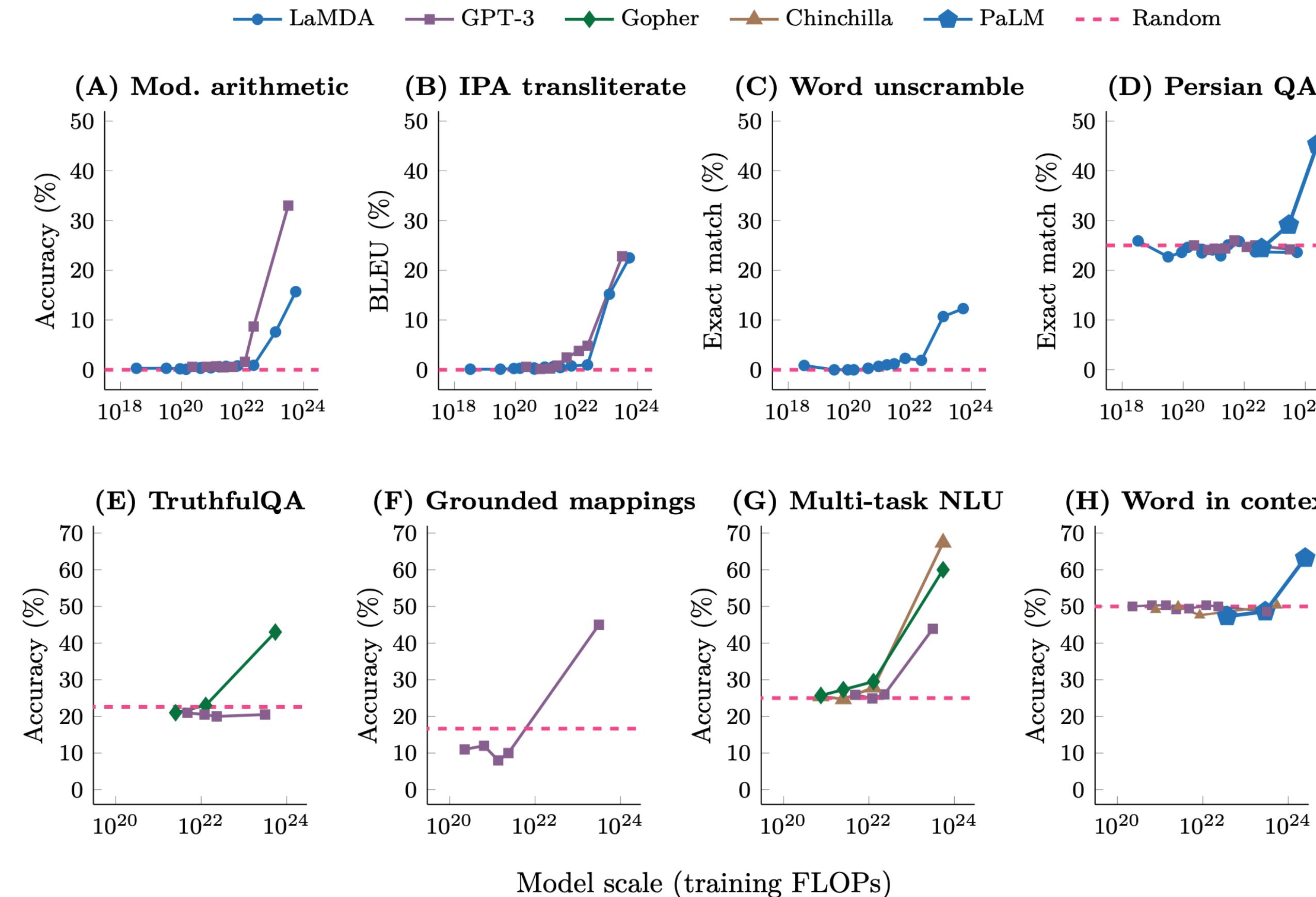


SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

Scaling Up Unlocks More Capabilities

The “emergent” abilities

- LLMs exhibit some “emergent” abilities that are only available with a large enough model size



(a) Modified arithmetic

In the following lines, the symbol \rightarrow represents a simple mathematical operation.

$$100 + 200 \rightarrow 301$$

$$1 + 1 \rightarrow 3$$

$$2 + 2 \rightarrow 5$$

(c) Word unscrambling

Input: The word **hte** is a scrambled version of the English word

Output: the

Input: The word **sohpto** is a scrambled version of the English word

Output: photos

GPT-3

Scaling up Transformers to be few-shot learners (in-context learning)

- Traditional NLP pipeline:
 - Pretraining
 - **Fine-tuning** on downstream tasks



Language Models are Few-Shot Learners [Brown et al., 2020]

GPT-3

Scaling up Transformers to be few-shot learners (in-context learning)

- Scaled-up LLM (175B) can generalize to new tasks w/o fine-tuning by either
 - **Zero-shot:** answer questions given task descriptions
 - **Few-shot:** learn with demonstrations

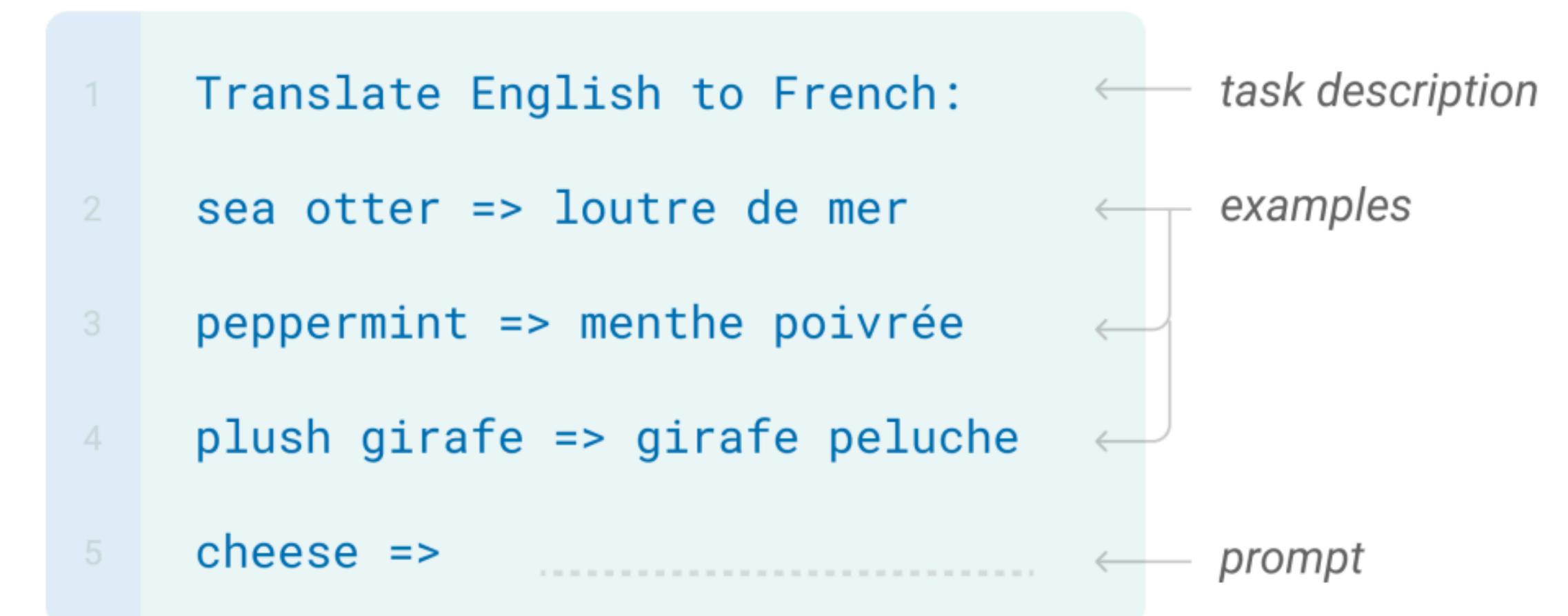
Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

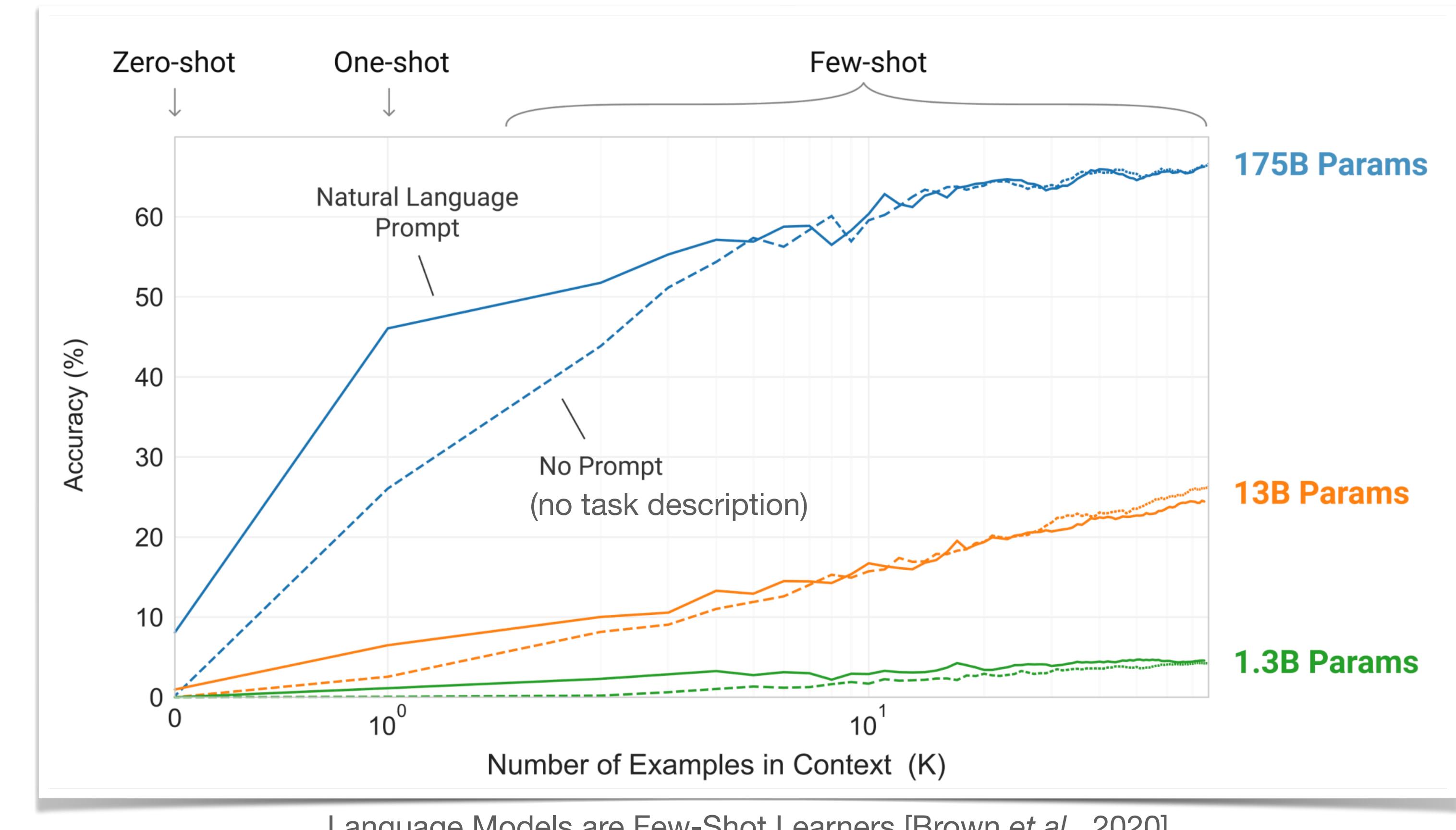


Language Models are Few-Shot Learners [Brown et al., 2020]

GPT-3

Scaling up Transformers to be few-shot learners (in-context learning)

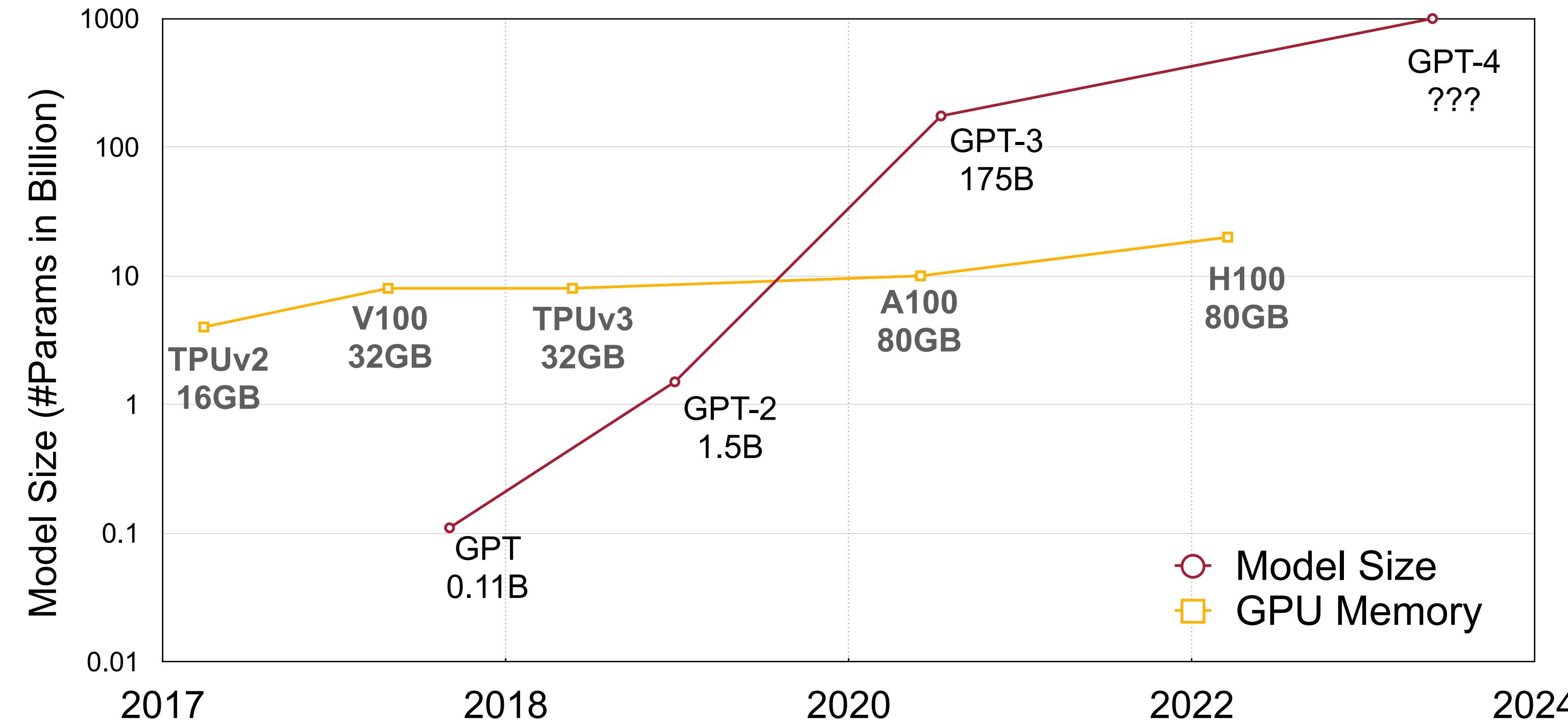
- Scaled-up LLM (175B) can generalize to new tasks w/o fine-tuning by either
 - **Larger** models can more **effectively** utilize few-shot demonstrations
 - With more demonstrations, the no-prompt accuracy catches up with the task-prompted accuracy



GPT Family

The history of scaling for GPT models

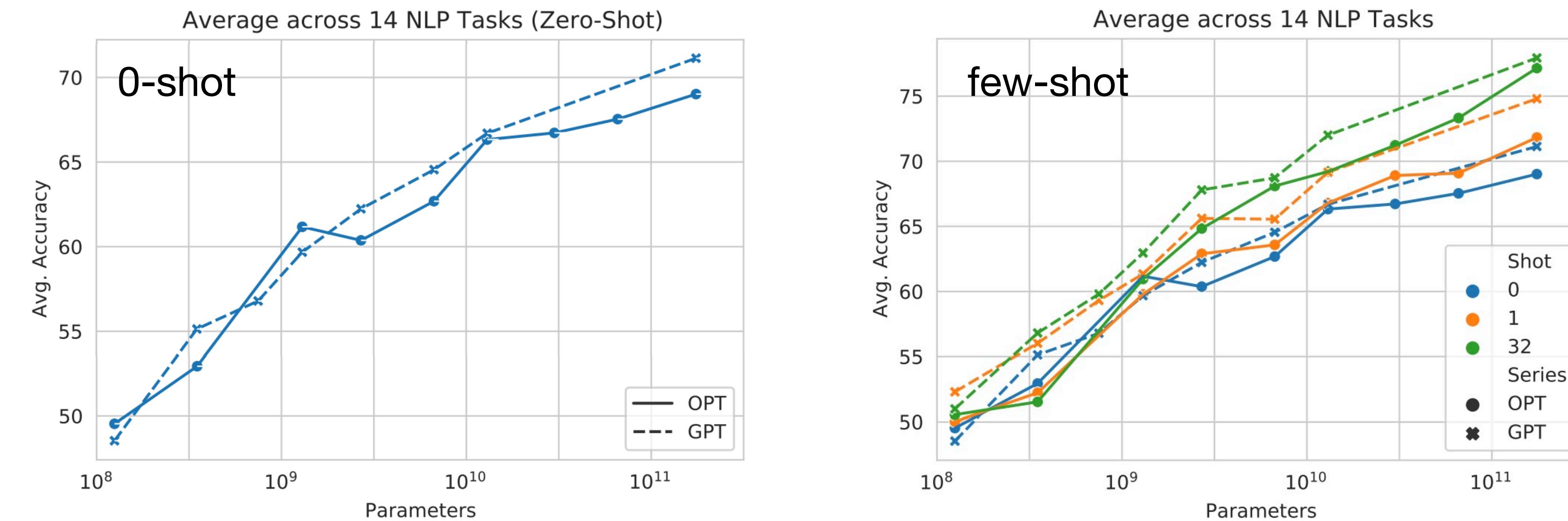
- GPT-1 (2018): 117 million
- GPT-2 (2019): 1.5B
- GPT-3 (2020): 175B
- GPT-4 (2023): ???



Language Models are Few-Shot Learners [Brown et al., 2020]

Open Pre-trained Transformer Language Models

- Open-source pre-trained LLM from Meta
- Size: 125M/350M/1.3B/2.7B/6.7B/13B/30B/66B/175B
- Design choices: Decoder-only, Pre-norm (post-norm for 350M), ReLU activation in FFN
- 175B model: hidden dimension 12288, #heads 96, vocab size 50k, context length 2048
- Close performance compared to GPT models



OPT: Open Pre-trained Transformer Language Models [Zhang et al., 2022]

LLaMA

Significantly better opensource LLMs

- Open-source pre-trained LLM from Meta
- Size: 7B/13B/33B/65B
- Design choices: Decoder-only, Pre-norm, **SwiGLU (swish, gated linear units)**, rotary positional embedding (**RoPE**)
- 7B model: hidden dimension 4096, #heads 32, #layers 32, vocab size 32k, context length 2048
- 65B model: hidden dimension 8192, #heads 64, #layers 80, vocab size 32k, context length 2048
- Much better performance compared to previous opensource models

$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2$$

		BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
Gopher	280B	79.3	81.8	50.6	79.2	70.1	-	-	-
Chinchilla	70B	83.7	81.8	51.3	80.8	74.9	-	-	-
PaLM	62B	84.8	80.5	-	79.7	77.0	75.2	52.5	50.4
PaLM-cont	62B	83.9	81.4	-	80.6	77.0	-	-	-
PaLM	540B	88.0	82.3	-	83.4	81.1	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	80.0	57.8	58.6
	65B	85.3	82.8	52.3	84.2	77.0	78.9	56.0	60.2

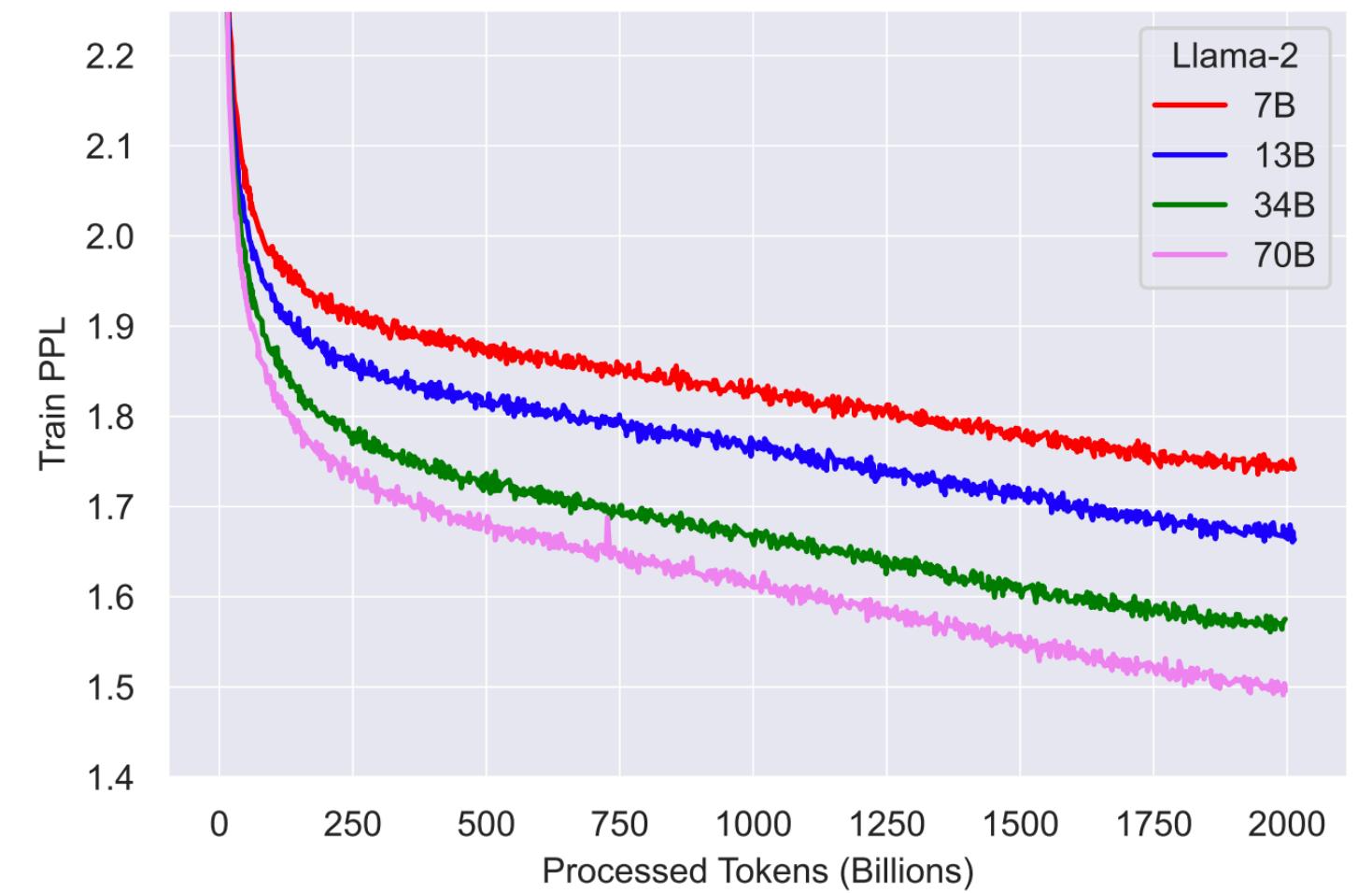
LLaMA: Open and Efficient Foundation Language Models [Touvron et al., 2023]

Llama 2

Better performance from a larger training corpus

- Larger context length (2k -> 4k)
- **More training tokens** (1T/1.4T -> 2T, no sign of saturation yet)
- **GQA** for larger models (70B, 64 heads, 8 kv heads)
- Also include Llama-2-chat, an **instruction-tuned** version

Benchmark (shots)	GPT-3.5	GPT-4	PaLM	PaLM-2-L	LLAMA 2
MMLU (5-shot)	70.0	86.4	69.3	78.3	68.9
TriviaQA (1-shot)	–	–	81.4	86.1	85.0
Natural Questions (1-shot)	–	–	29.3	37.5	33.0
GSM8K (8-shot)	57.1	92.0	56.5	80.7	56.8
HumanEval (0-shot)	48.1	67.0	26.2	–	29.9
BIG-Bench Hard (3-shot)	–	–	52.3	65.7	51.2



Llama 3

Better performance from a larger training corpus

- **More training tokens: 1.8T → 15.6T multilingual tokens.**
- Increased scale: 50x more FLOPs than Llama 2. The flagship model has 405B parameters.
- Larger context length: Base model 8K. Instruction-tuned model 128K (long-context).
- Simplified post-training: Supervised fine-tuning (SFT), rejection sampling (RS), and direct preference optimization (DPO).

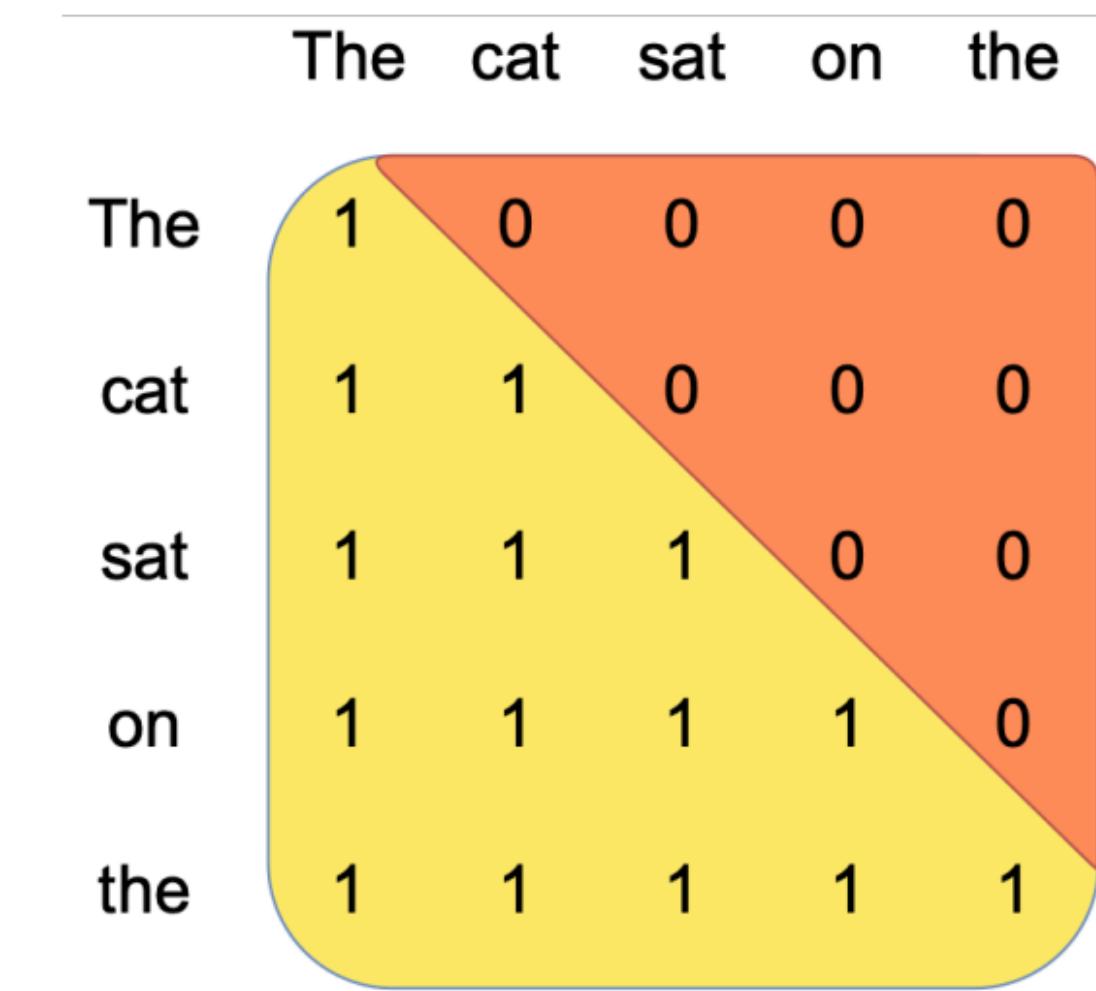
Category	Benchmark	Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 _(c125)	GPT-4o	Claude 3.5 Sonnet
General	MMLU (5-shot)	69.4	72.3	61.1	83.6	76.9	70.7	87.3	82.6	85.1	89.1	89.9
	MMLU (0-shot, CoT)	73.0	72.3 [△]	60.5	86.0	79.9	69.8	88.6	78.7 [△]	85.4	88.7	88.3
	MMLU-Pro (5-shot, CoT)	48.3	—	36.9	66.4	56.3	49.2	73.3	62.7	64.8	74.0	77.0
	IFEval	80.4	73.6	57.6	87.5	72.7	69.9	88.6	85.1	84.3	85.6	88.0
Code	HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0	89.0	73.2	86.6	90.2	92.0
	MBPP EvalPlus (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0	88.6	72.8	83.6	87.8	90.5
Math	GSM8K (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6	96.8	92.3 [△]	94.2	96.1	96.4 [△]
	MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1	73.8	41.1	64.5	76.6	71.1
Reasoning	ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7	96.9	94.6	96.4	96.7	96.7
	GPQA (0-shot, CoT)	32.8	—	28.8	46.7	33.3	30.8	51.1	—	41.4	53.6	59.4
Tool use	BFCL	76.1	—	60.4	84.8	—	85.9	88.5	86.5	88.3	80.5	90.2
	Nexus	38.5	30.0	24.7	56.7	48.5	37.2	58.7	—	50.3	56.1	45.7
Long context	ZeroSCROLLS/QuALITY	81.0	—	—	90.5	—	—	95.2	—	95.2	90.5	90.5
	InfiniteBench/En.MC	65.1	—	—	78.2	—	—	83.4	—	72.1	82.5	—
	NIH/Multi-needle	98.8	—	—	97.5	—	—	98.1	—	100.0	100.0	90.8
Multilingual	MGSM (0-shot, CoT)	68.9	53.2	29.9	86.9	71.1	51.4	91.6	—	85.9	90.5	91.6

LLaMA: Open and Efficient Foundation Language Models [Touvron et al., 2023]

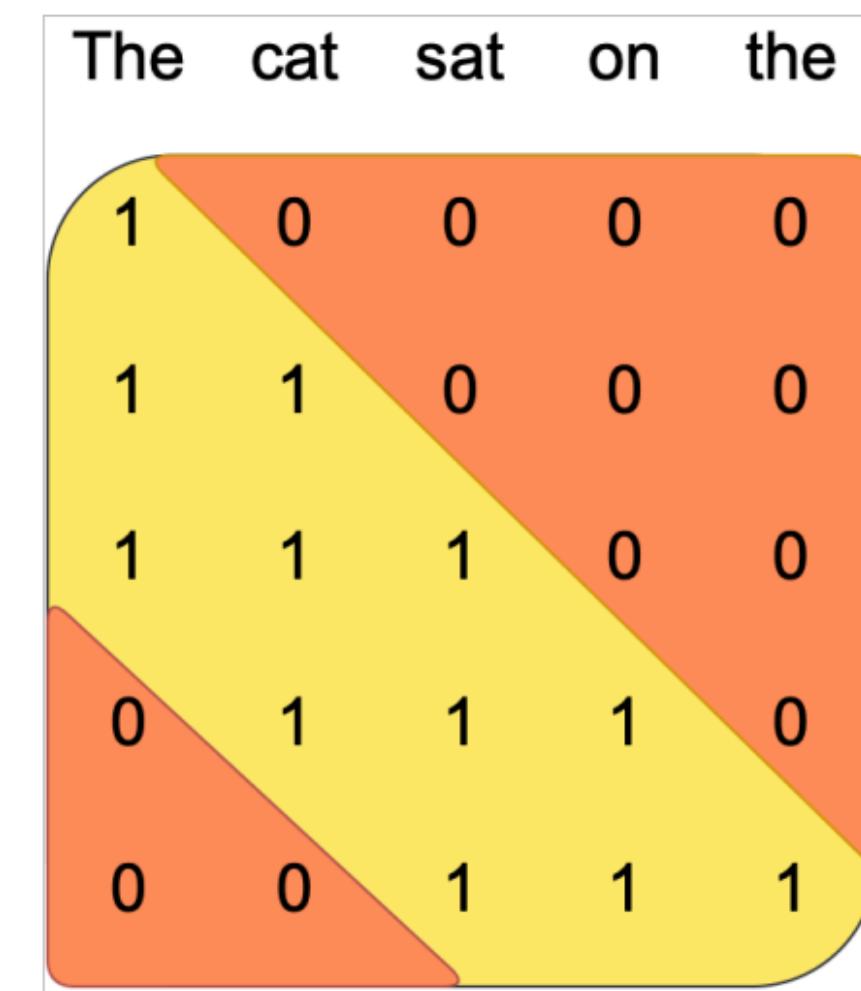
Mistral-7B

Small model with superior performance

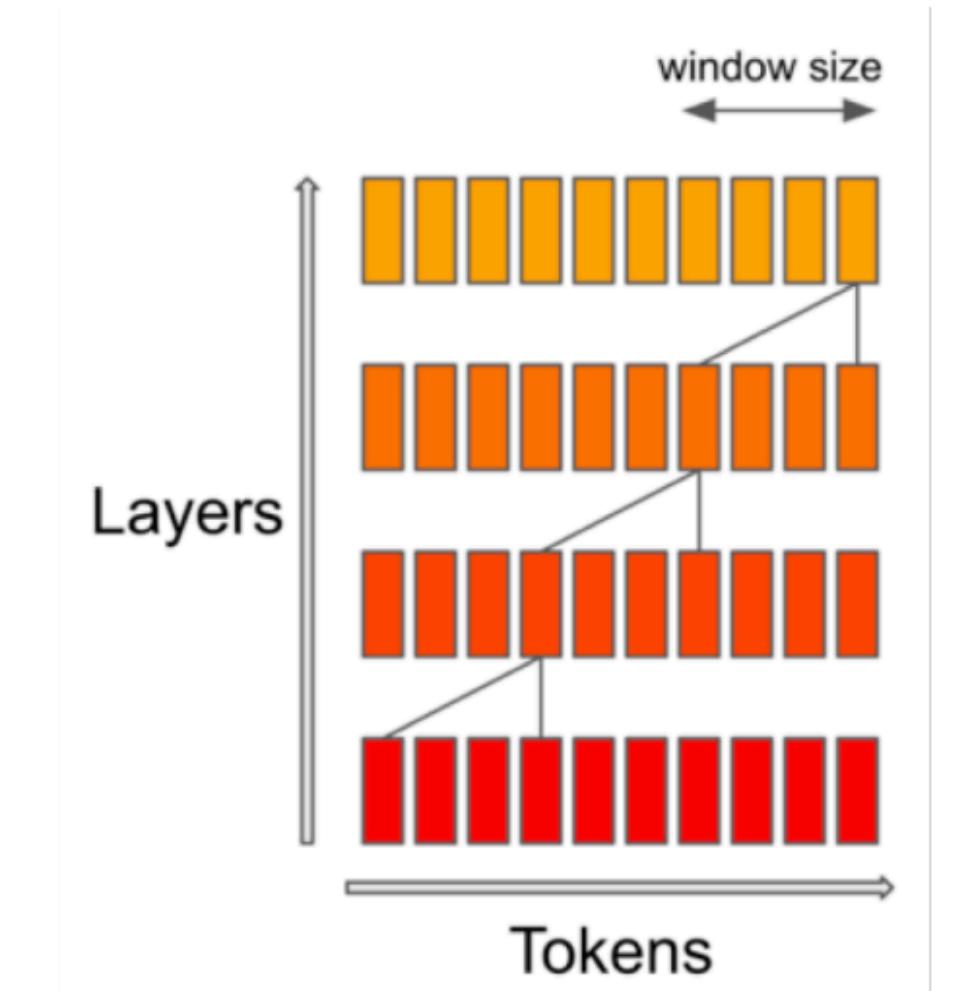
- 7B model that outperforms Llama-2-13B, Llama-34B
- hidden dimension 4096, #heads 32, #kv heads 8, #layers 32, vocab size 32k, context length 8k
- Grouped-query attention (GQA)
- **Sliding window attention (SWA) for longer effective context length (deprecated since v2)**



Vanilla Attention



Sliding Window Attention



Effective Context Length

How to scale up?

The Chinchilla Law

- We need to scale up **both** the model size and data size for training to have the best **training** computation vs. accuracy trade-off
- **Note:** the trade-off is different if we consider the **inference** computation trade-off
 - You want to train a smaller model longer to save inference costs (e.g., LLaMA)!

Parameters	FLOPs	FLOPs (in <i>Gopher</i> unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
<u>10 Billion</u>	<u>1.23e+22</u>	<u>1/46</u>	<u>205.1 Billion</u>
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

Llama-2 model goes beyond the #tokens by a lot (7B, 2T tokens)

IV. Advanced Topics

Multi-modal LLMs

Towards LLMs that can see

- GPT-4V(ision) can take visual inputs

Prompt:

How much did I pay for tax?
Where is the information?



GPT-4V:

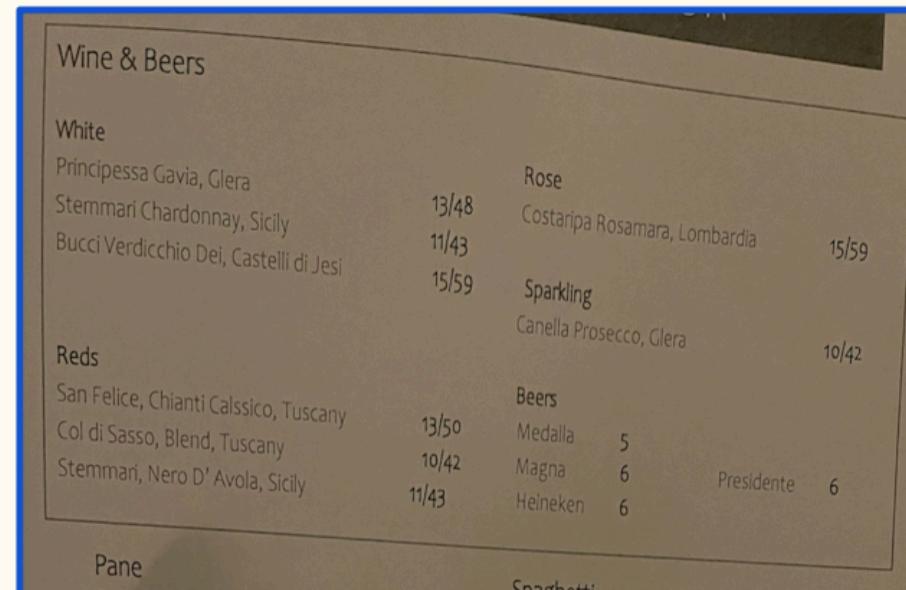
In the first receipt, you paid \$3.72 for tax. The information is located at the bottom of the receipt. In the second receipt, you paid \$42.23 for tax. The information is located near the bottom of the receipt, under the "Sales Tax" line. In the third receipt, you paid \$4.58 for tax. The information is located at the bottom of the receipt, under the "TAX" line. In total, you paid \$50.53 in tax across all three receipts.

Prompt:

How much should I pay for the beer on the table according to the price on the menu?



Here is a photo of the menu.



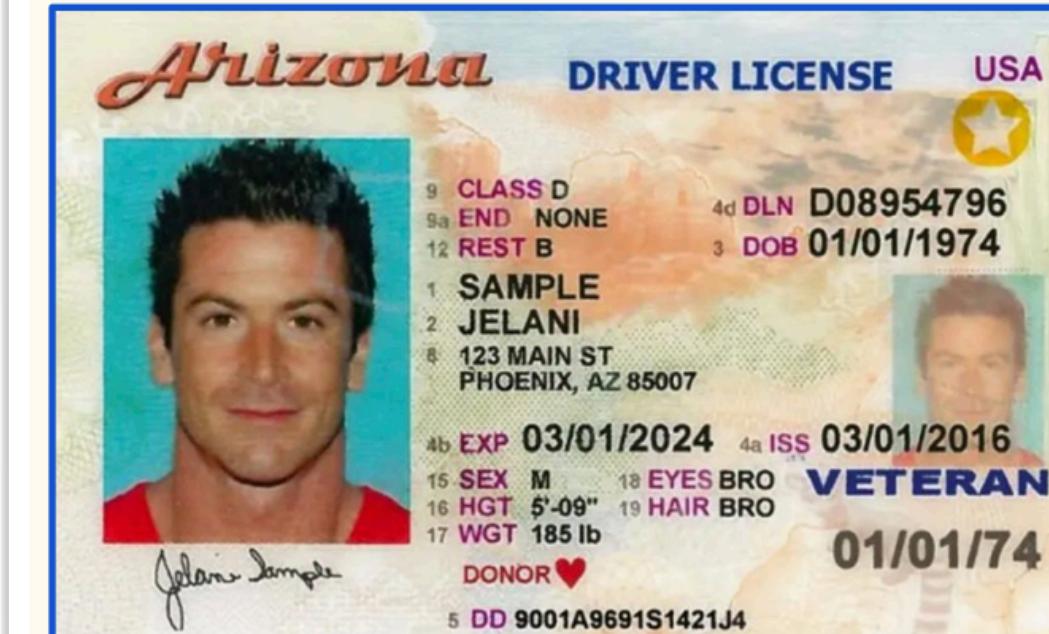
GPT-4V:

According to the menu, the price for a Magna beer is \$6. Since there are two Magna beers on the table, the total cost for the beers would be \$12.

Prompt:

Please read the text in this image and return the information in the following JSON format (note xxx is placeholder, if the information is not available in the image, put "N/A" instead).

```
{"class": xxx, "DLN": xxx, "DOB": xxx, "Name": xxx, "Address": xxx, "EXP": xxx, "ISS": xxx, "SEX": xxx, "HGT": xxx, "WGT": xxx, "EYES": xxx, "HAIR": xxx, "DONOR": xxx}
```



GPT-4V:

```
{"class": "D", "DLN": "D08954796", "DOB": "01/01/1974", "Name": "Jelani Sample", "Address": "123 Main St, Phoenix, AZ 85007", "EXP": "03/01/2024", "ISS": "03/01/2016", "SEX": "M", "HGT": "5-09", "WGT": "185 lbs", "EYES": "BRO", "HAIR": "N/A", "DONOR": "Y"}
```

Multi-modal LLMs

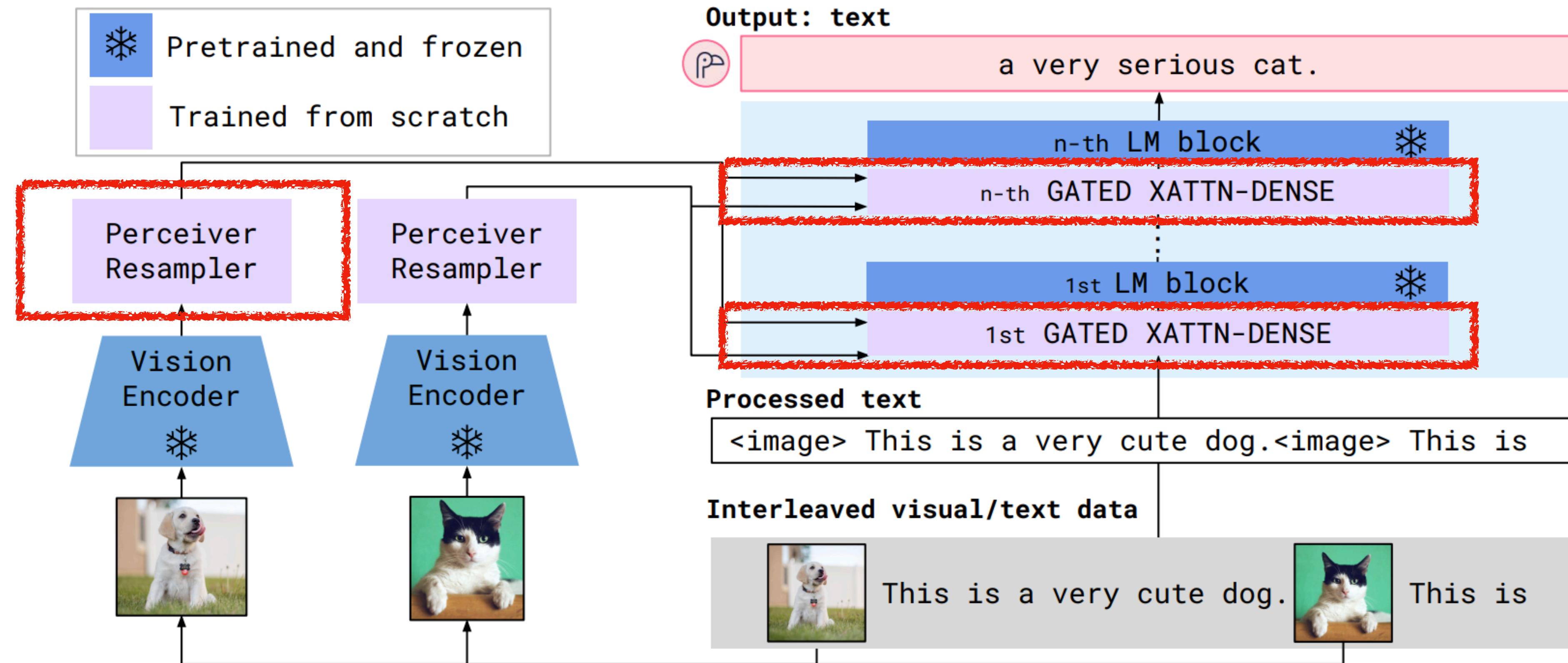
Two ways for visual language models

- Cross-attention to inject visual info into LLM (Flamingo style)
- Visual tokens as input (PaLM-E style)

Flamingo

Cross-attention-based VLMs

- Pioneer work on enabling visual input support for LLMs
- LLM is frozen, added cross-attention layers to the intermediate layers of LLMs



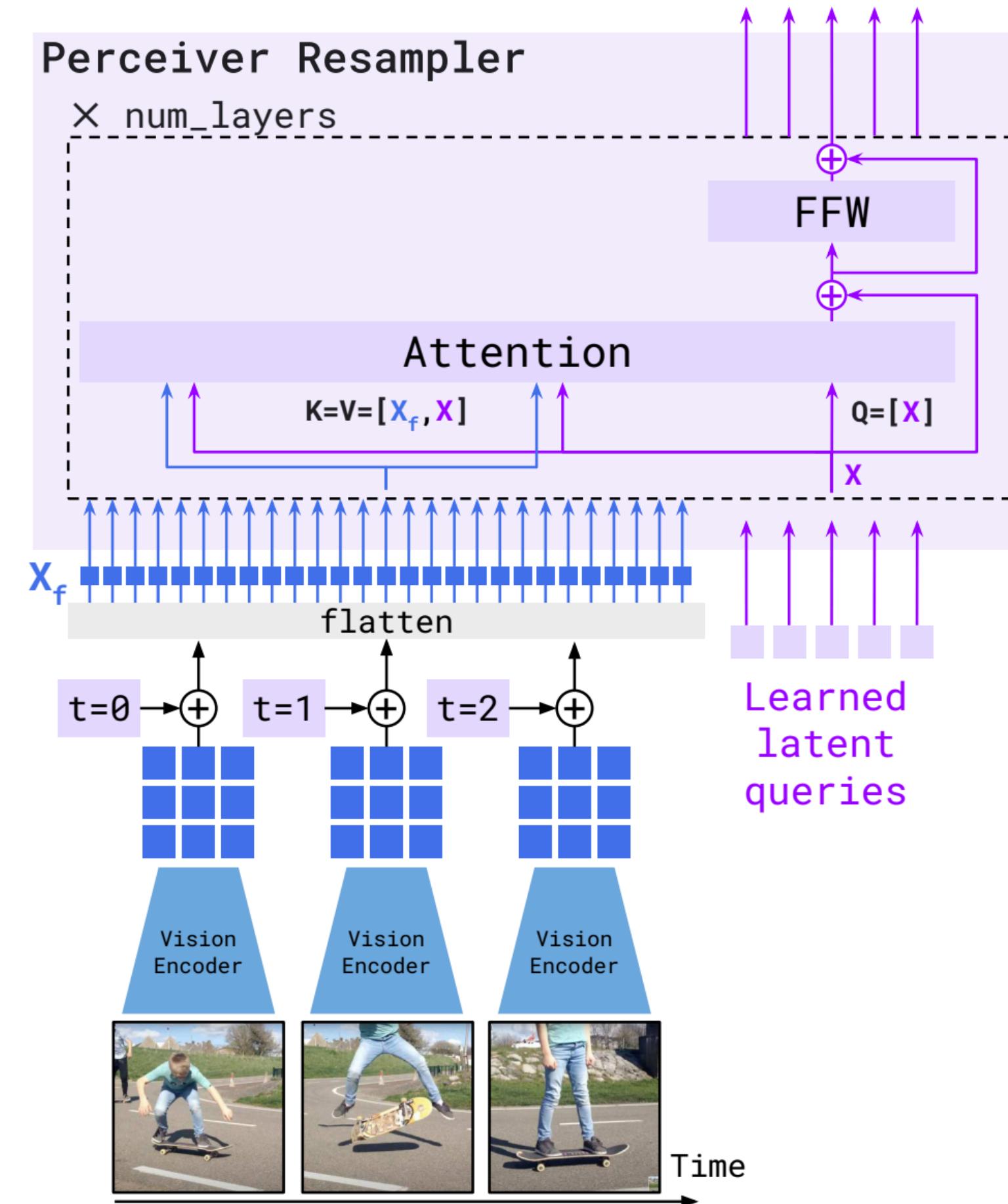
Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]

Flamingo

Cross-attention-based VLMs

- **Perceiver Resampler:** from varying-size large feature maps to few fixed-size visual tokens

- visual tokens: [27, dim]
- learned queries (Q): [5, dim]
- K&V: [32, dim]
- Attention map: [5, 32]
- output: [5, dim]



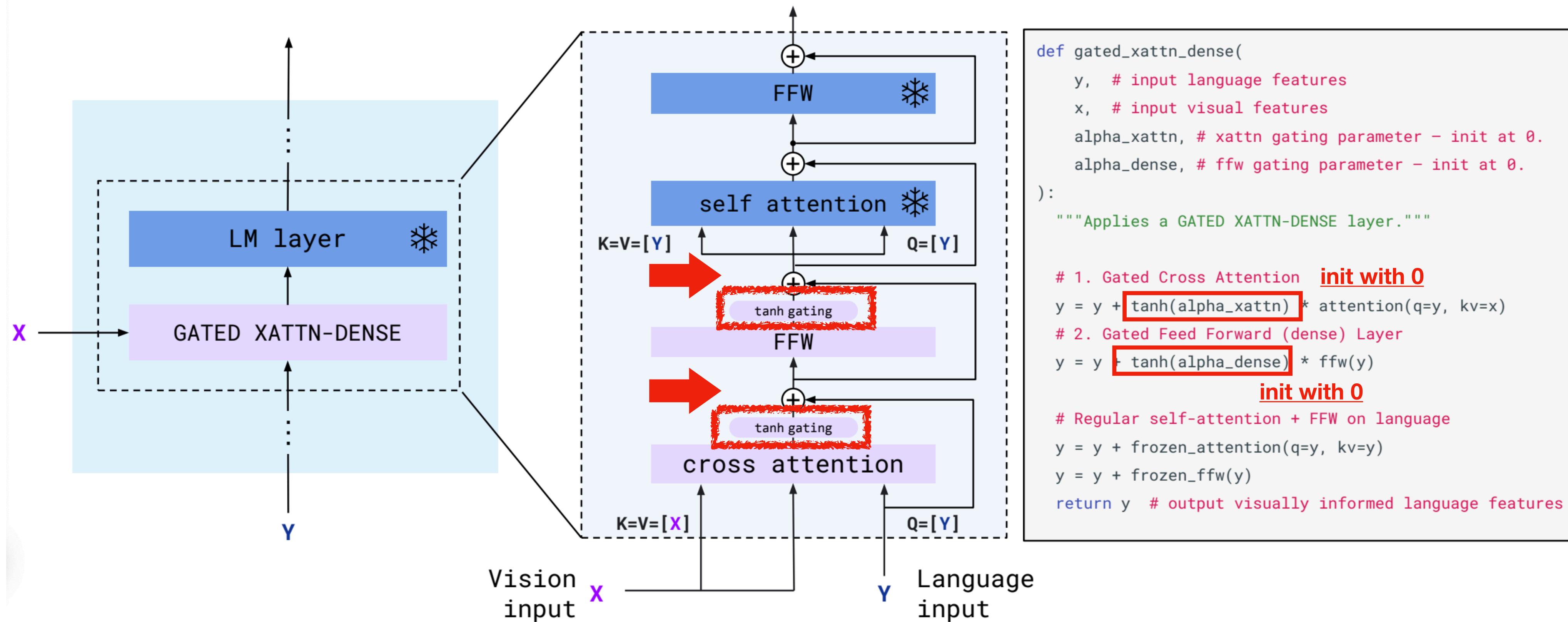
```
def perceiver_resampler(  
    x_f, # The [T, S, d] visual features (T=time, S=space)  
    time_embeddings, # The [T, 1, d] time pos embeddings.  
    x, # R learned latents of shape [R, d]  
    num_layers, # Number of layers  
):  
    """The Perceiver Resampler model."""  
  
    # Add the time position embeddings and flatten.  
    x_f = x_f + time_embeddings  
    x_f = flatten(x_f) # [T, S, d] -> [T * S, d]  
    # Apply the Perceiver Resampler layers.  
    for i in range(num_layers):  
        # Attention.  
        x = x + attention_i(q=x, kv=concat([x_f, x]))  
        # Feed forward.  
        x = x + ffw_i(x)  
    return x
```

Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]

Flamingo

Cross-attention-based VLMs

- **Gated cross-attention layer**
 - Use a Tanh gate to control the amount of visual information

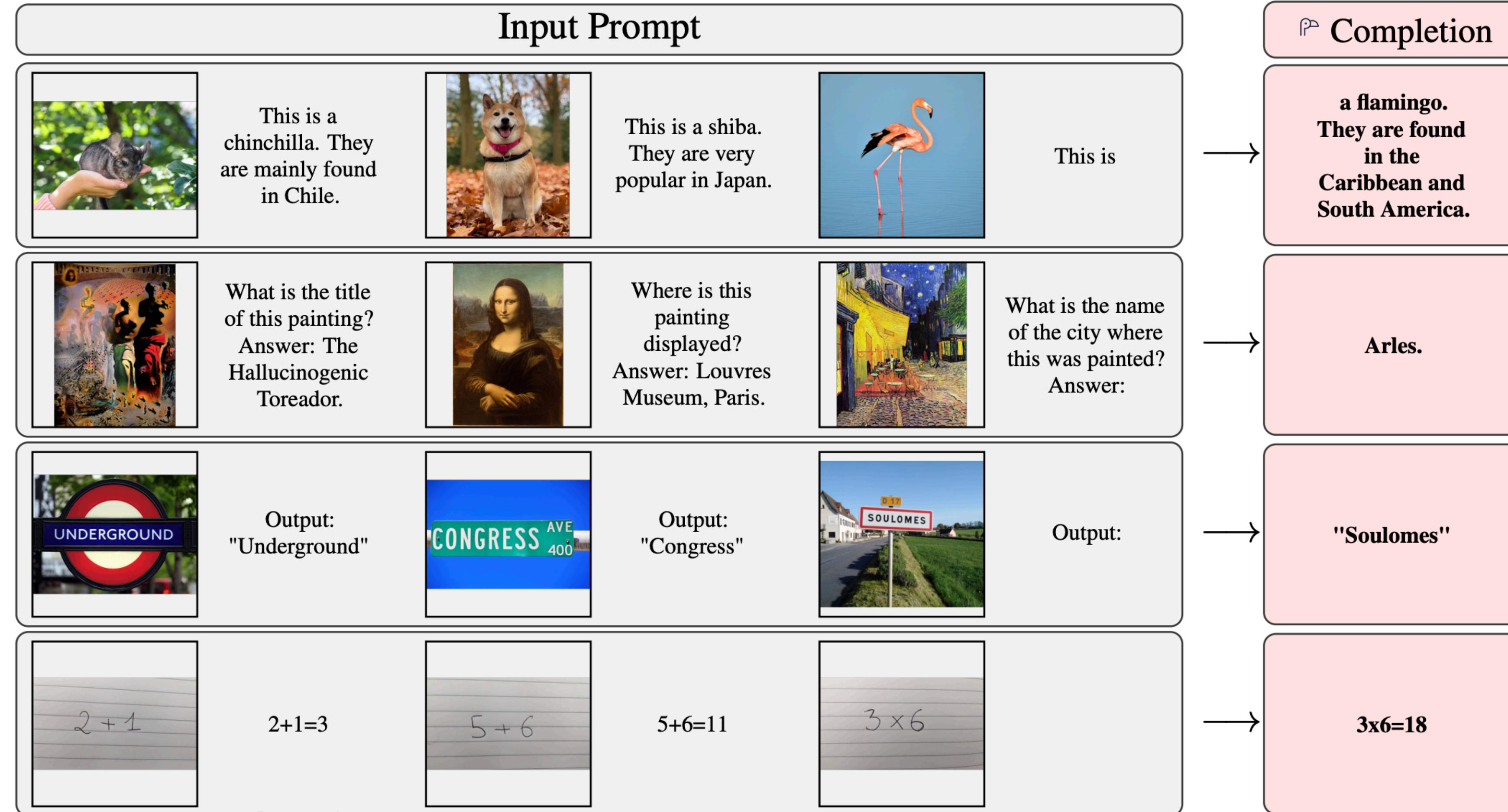


Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]

Flamingo

Cross-attention-based VLMs

- In-context learning capability



Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]

Flamingo

Cross-attention-based VLMs

- Visual dialogue

The figure illustrates a visual dialogue system named Flamingo, showing four rounds of interaction between a user and a model.

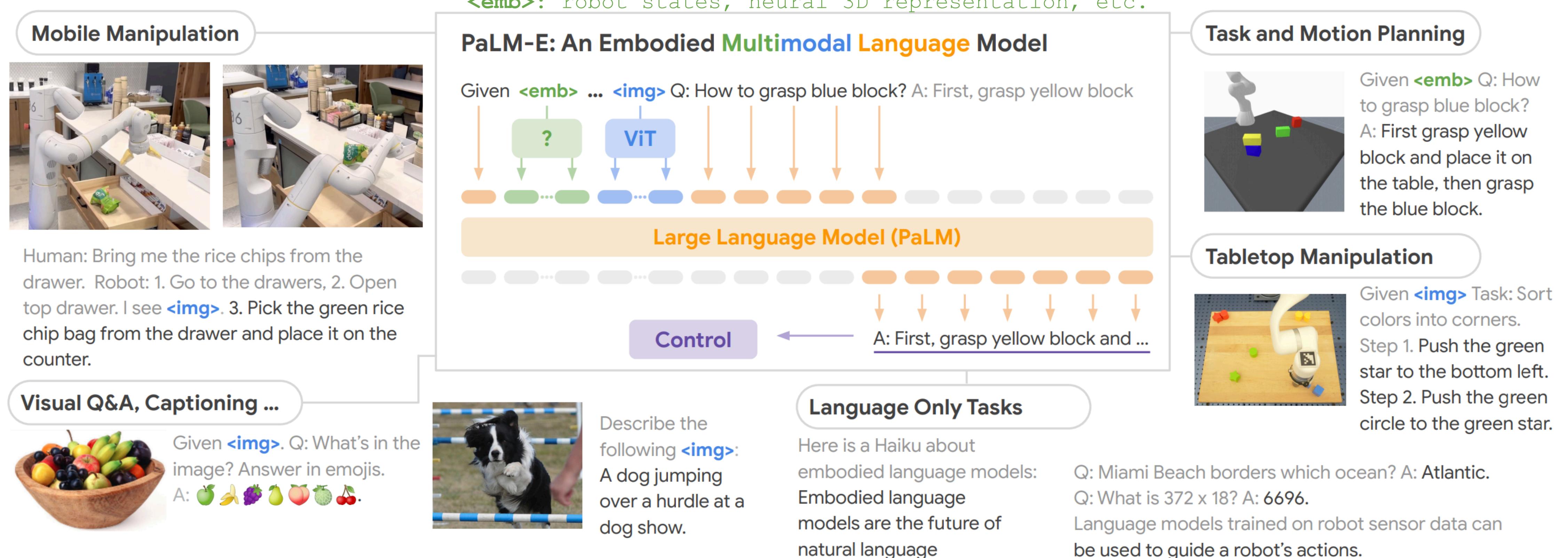
- Round 1:** User shows a photo of two teddy bears on the moon. Model asks: "What are they doing?", "They are having a conversation.", "What object are they using?", "It looks like a computer.", "Is this surprising?", "Yes, it is surprising.", "Why is this picture surprising to you?", and "I think it is surprising because teddy bears are not usually found on the moon."
- Round 2:** User shows three images of flamingos. Model asks: "What is the common thing about these three images?", "They are all flamingos.", "What is the difference between these three images?", and "The first one is a cartoon, the second one is a real flamingo, and the third one is a 3D model of a flamingo."
- Round 3:** User shows a photo of an apple with an "iPod" sticker. Model asks: "What does the sticker say?", "The sticker says 'iPod'.", "Where is the photo taken?", "It looks like it's taken in a backyard.", "Do you think it is printed or handwritten?", "It looks like it's handwritten.", "What color is the sticker?", and "It's white."
- Round 4:** User shows a photo of a cityscape (Chicago). Model asks: "This is a cityscape. It looks like Chicago.", "What makes you think this is Chicago?", and "I think it's Chicago because of the Shedd Aquarium in the background."

Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]

PaLM-E

Taking visual inputs (or other modalities) as tokens

- Support flexible modalities (e.g., visual, action)

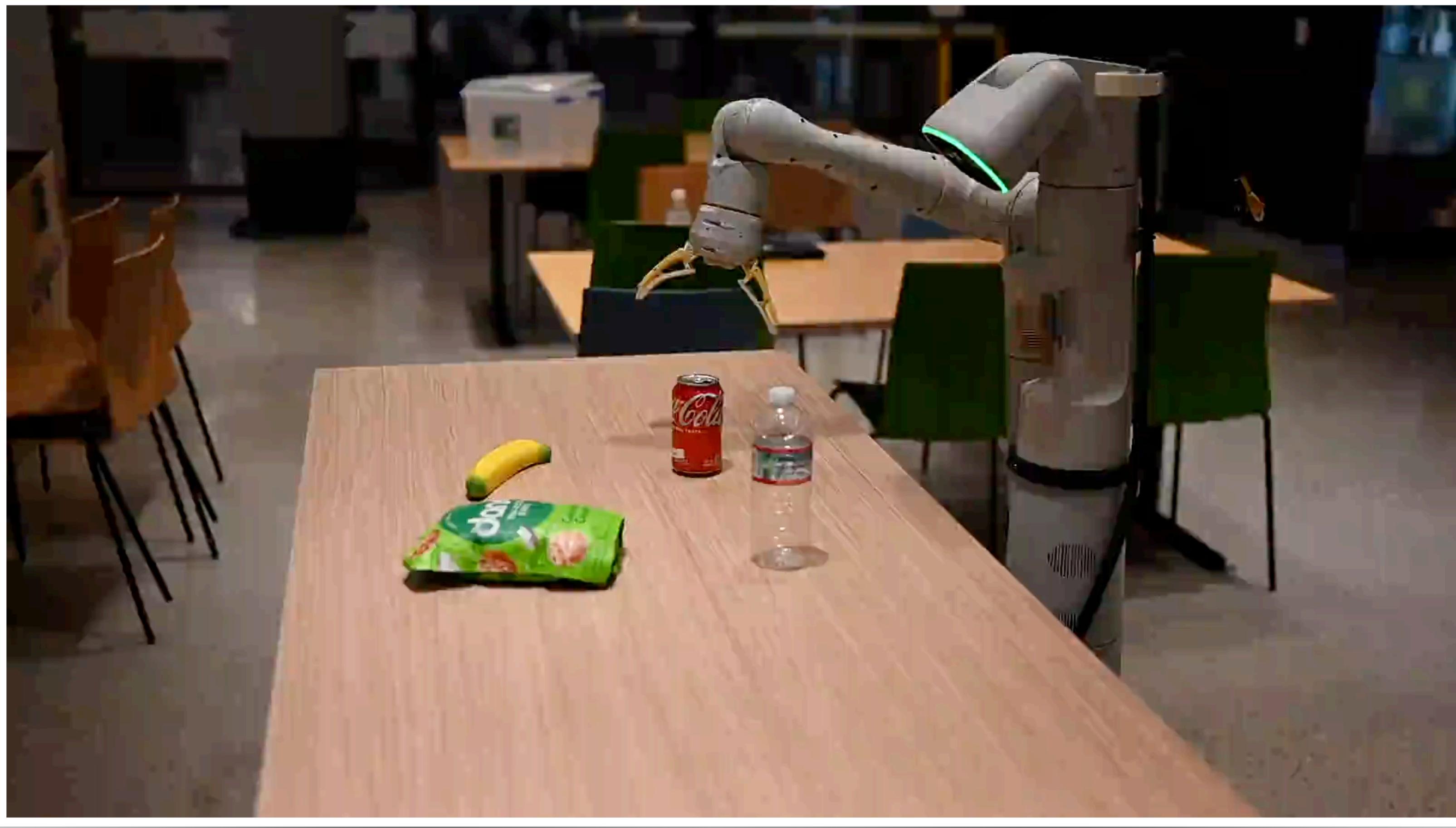


PaLM-E: An Embodied Multimodal Language Model [Driess et al., 2022]

PaLM-E -> RT-2

Taking visual inputs (or other modalities) as tokens

- Can directly output control signals



RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control [Brohan et al., 2023]

Multi-modal LLMs

Applications

- *Corner case handling* in autonomous driving



image credit: <https://datagen.tech/blog/how-synthetic-data-addresses-edge-cases-in-production/>

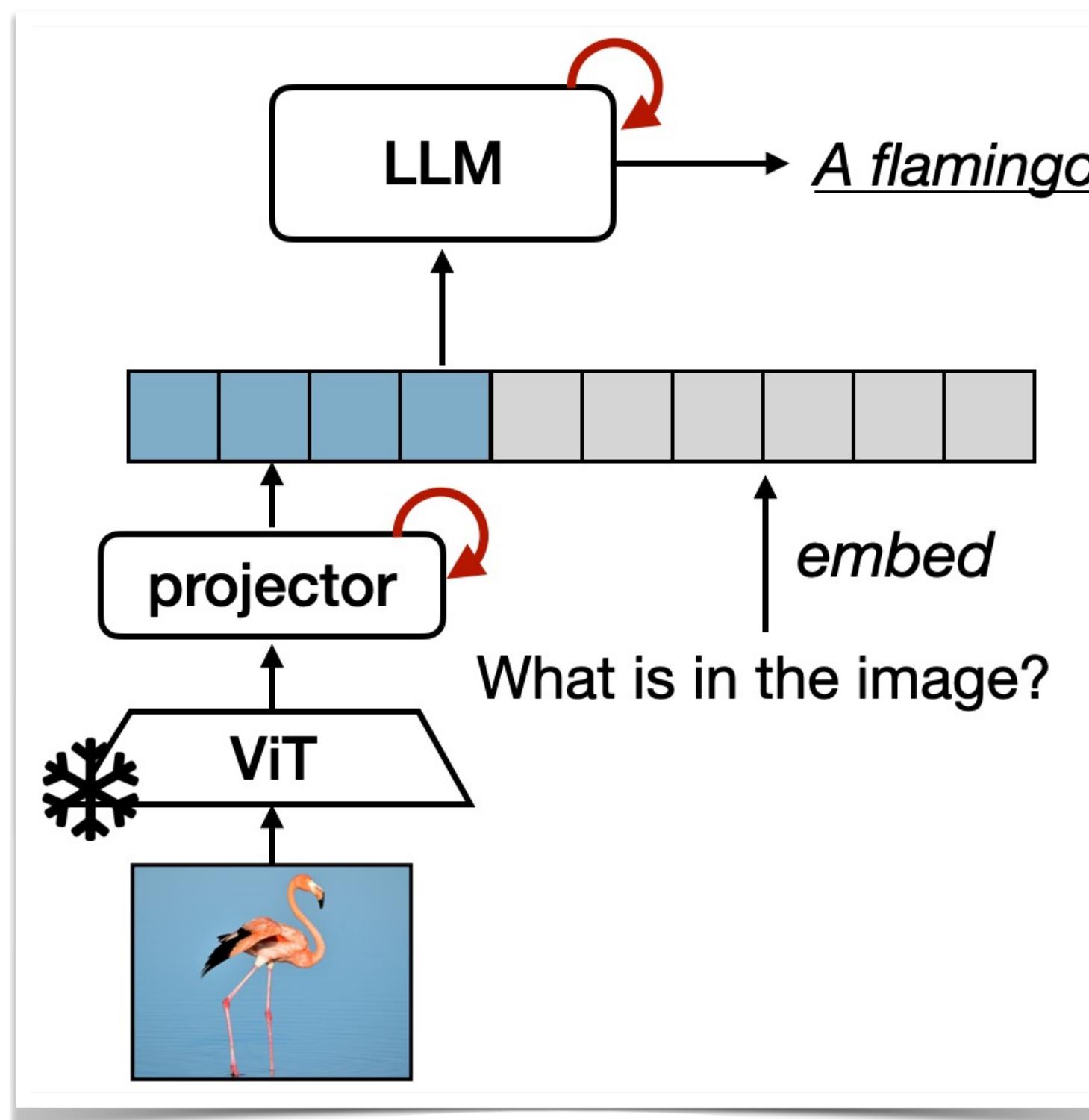
Q: What is unusual about this image?

A: The unusual aspect of this image is that a chair is flying through the air on a highway, seemingly coming out of the back of a truck.

Q: What should you do if you encounter this?

A: If you encounter this situation, you should immediately stop your vehicle and move to a safe distance from the truck and the flying chair. It is essential to avoid any potential hazards and contact the authorities to report the incident and ensure the safety of everyone involved.

Vision Language Model



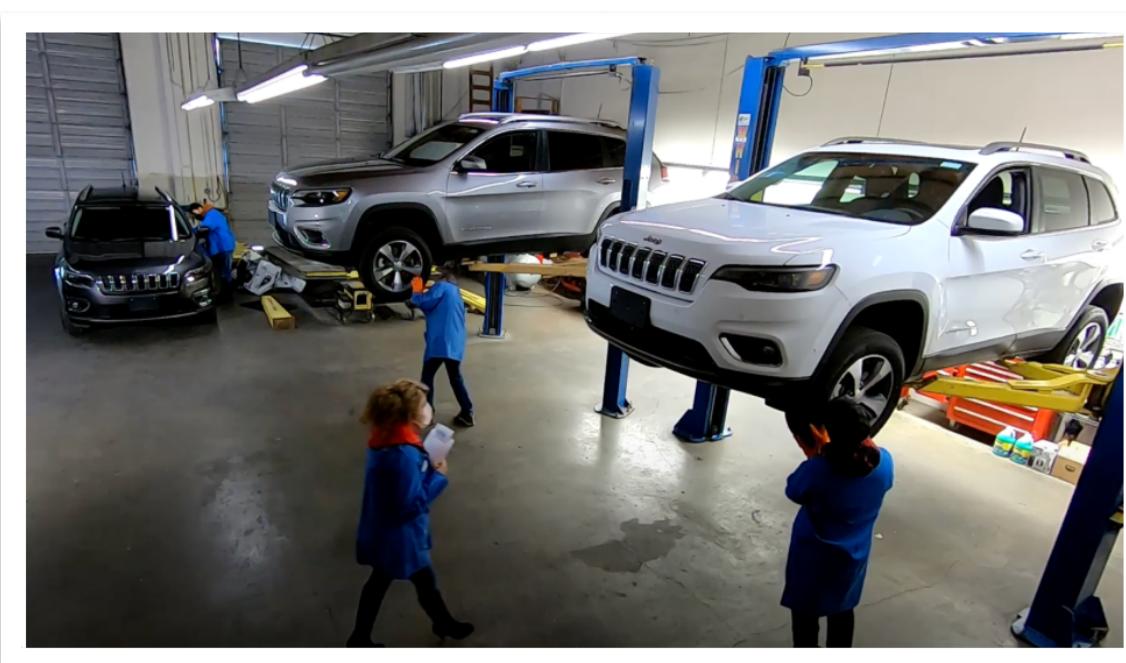
Q: What is unusual about this image?

A: The unusual aspect of this image is that a chair is flying through the air on a highway, seemingly coming out of the back of a truck.



Q: Can the vehicle proceed through the traffic now?

A: Based on the image, the vehicle cannot proceed through the traffic yet. There are multiple people and bicycles in the crosswalk, and the traffic light is red. The vehicle must wait for the traffic light to turn green before proceeding.



Q: How many cars are jacked up?

A: There are two cars jacked up in the image.

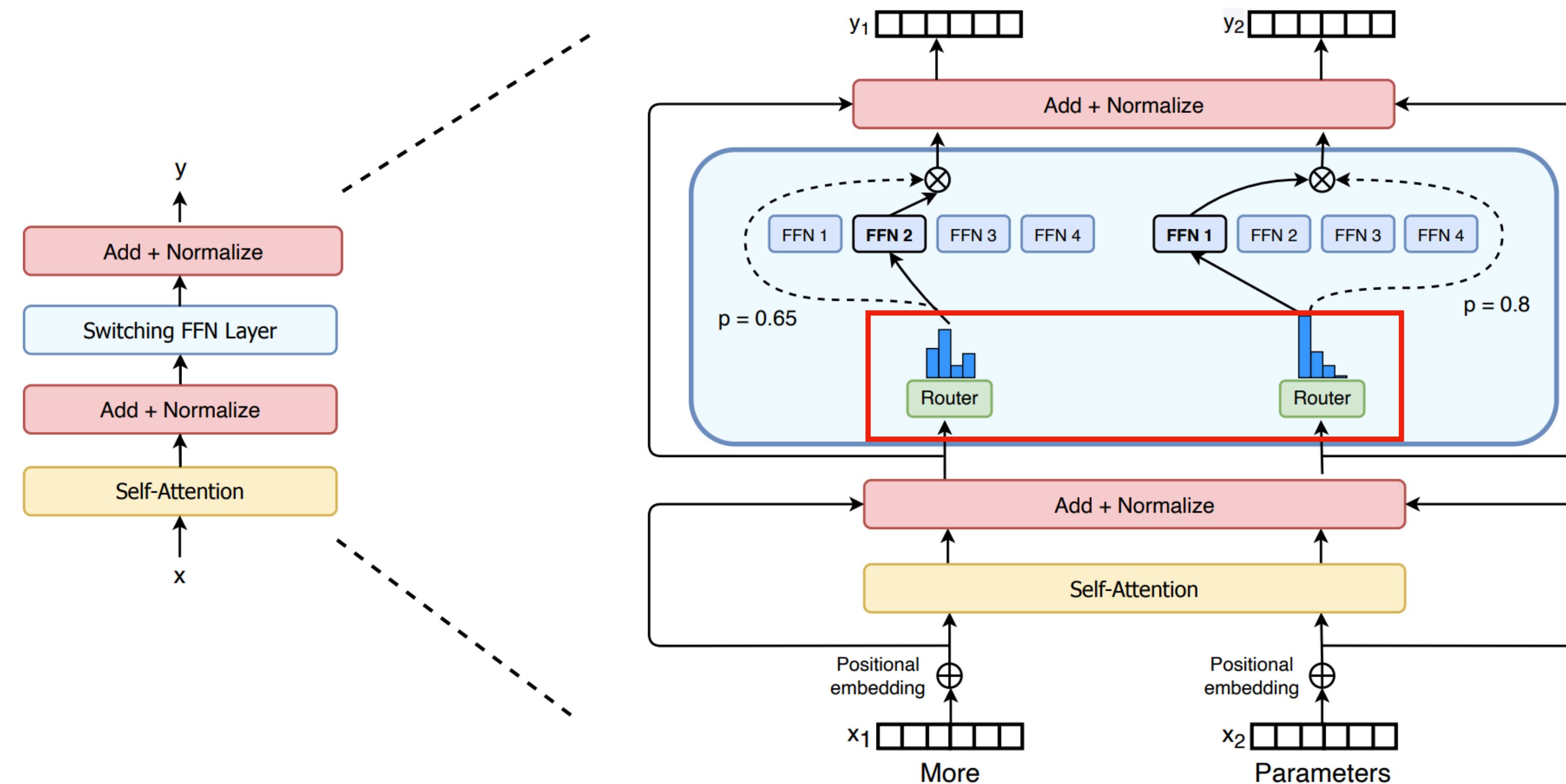
Q: How many persons are there whose head is under the jacked up car?

A: There are two persons whose head is under the jacked up car.

Mixture-of-Experts (MoE)

Sparsely activated for each token

- MoE allows us to sparsely use part of the parameters for each token during inference
- It can increase the **total** amount of parameters without increasing inference costs **per token**
- A router will distribute the workload among different experts

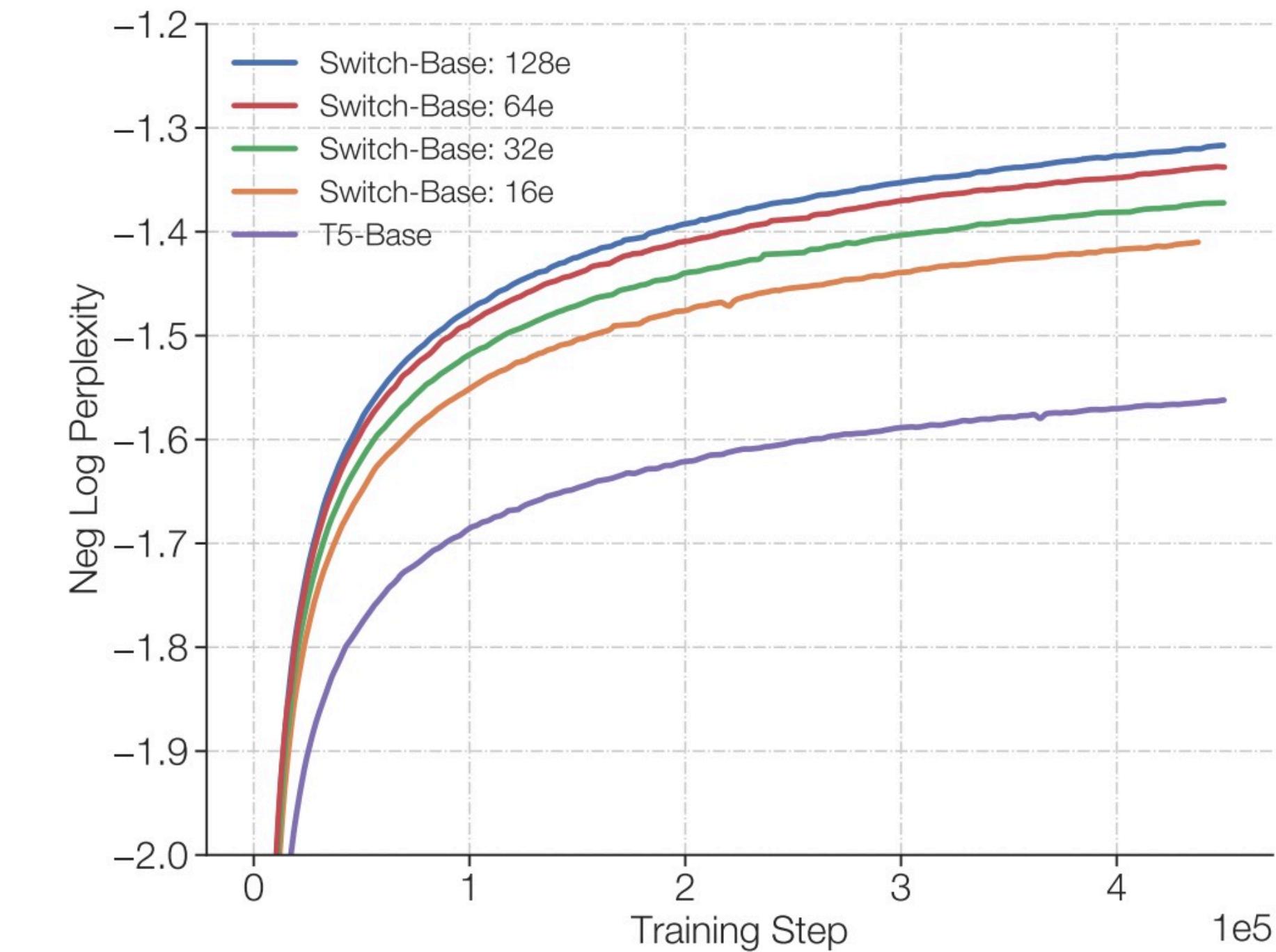
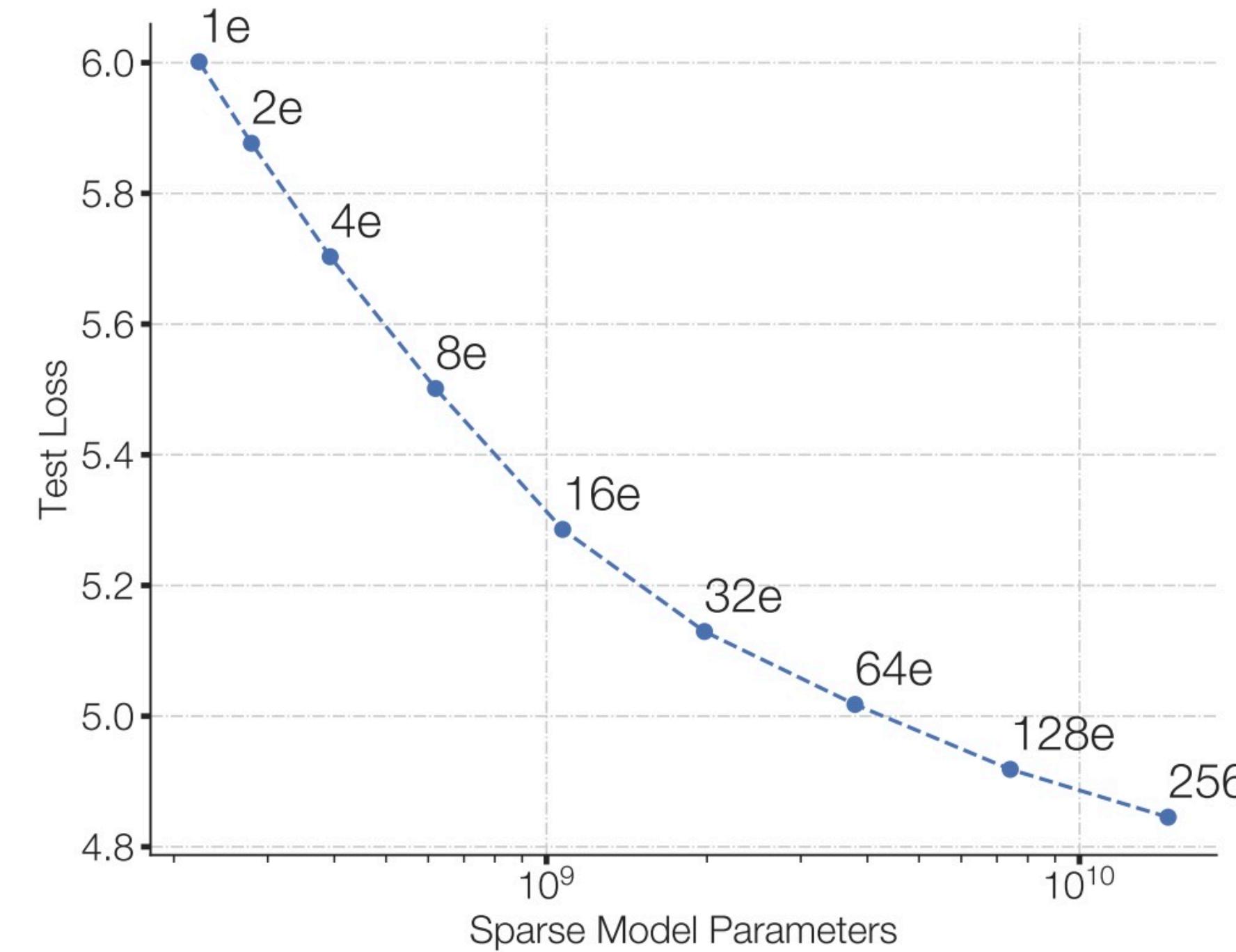


Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., 2022]

Mixture-of-Experts (MoE)

Sparsely activated for each token

- MoE allows us to use only part of the parameters for each token during inference
- It can increase the **total** amount of parameters without increasing inference costs **per token**
- A router will distribute the workload among different experts
- More experts -> larger total model size -> lower loss/better perplexity

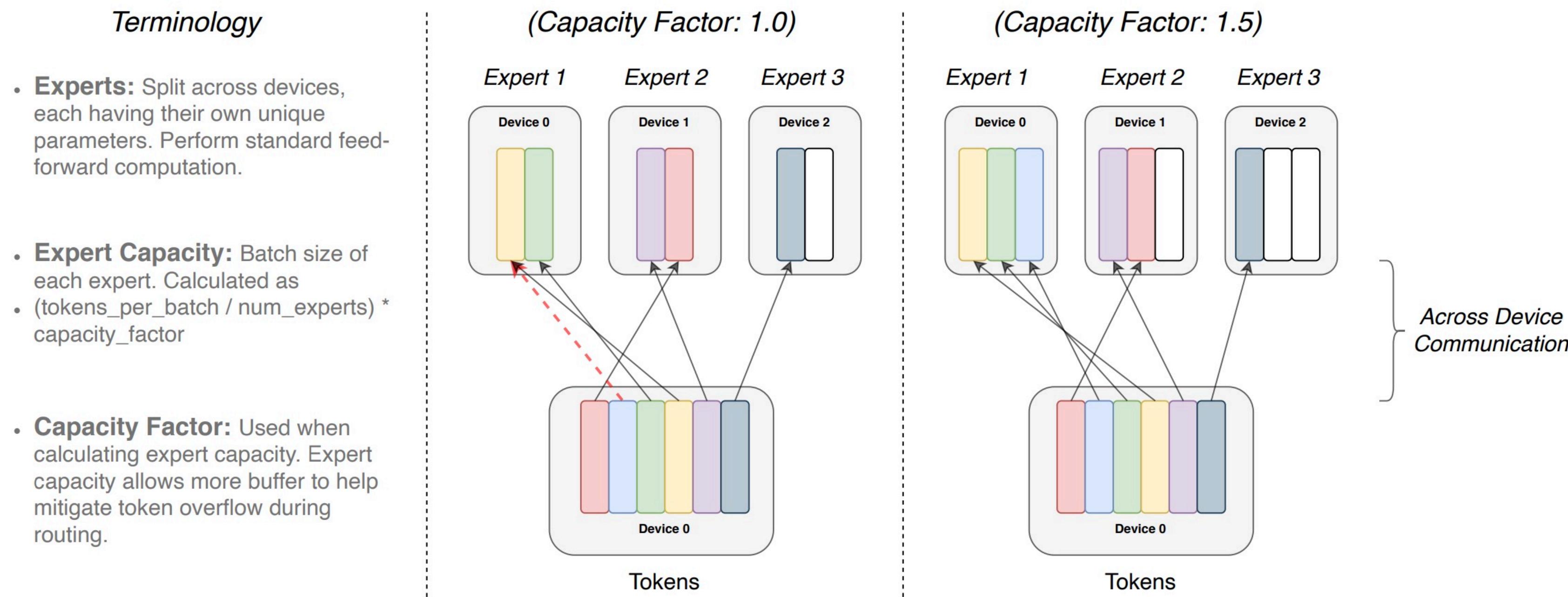


Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., 2022]

Mixture-of-Experts (MoE)

Sparsely activated for each token

- A key component of MoE is the routing function
- Capacity factor $C = \left(\frac{\text{tokens per batch}}{\text{number of experts}} \right) \times \text{capacity factor}$.
- $C = 1$, every expert can process at most $6/3*1=2$ tokens; one token is skipped
- $C = 1.5$, every expert can process at most $6/3*1.5=3$ tokens; slack capacity for expert 2&3

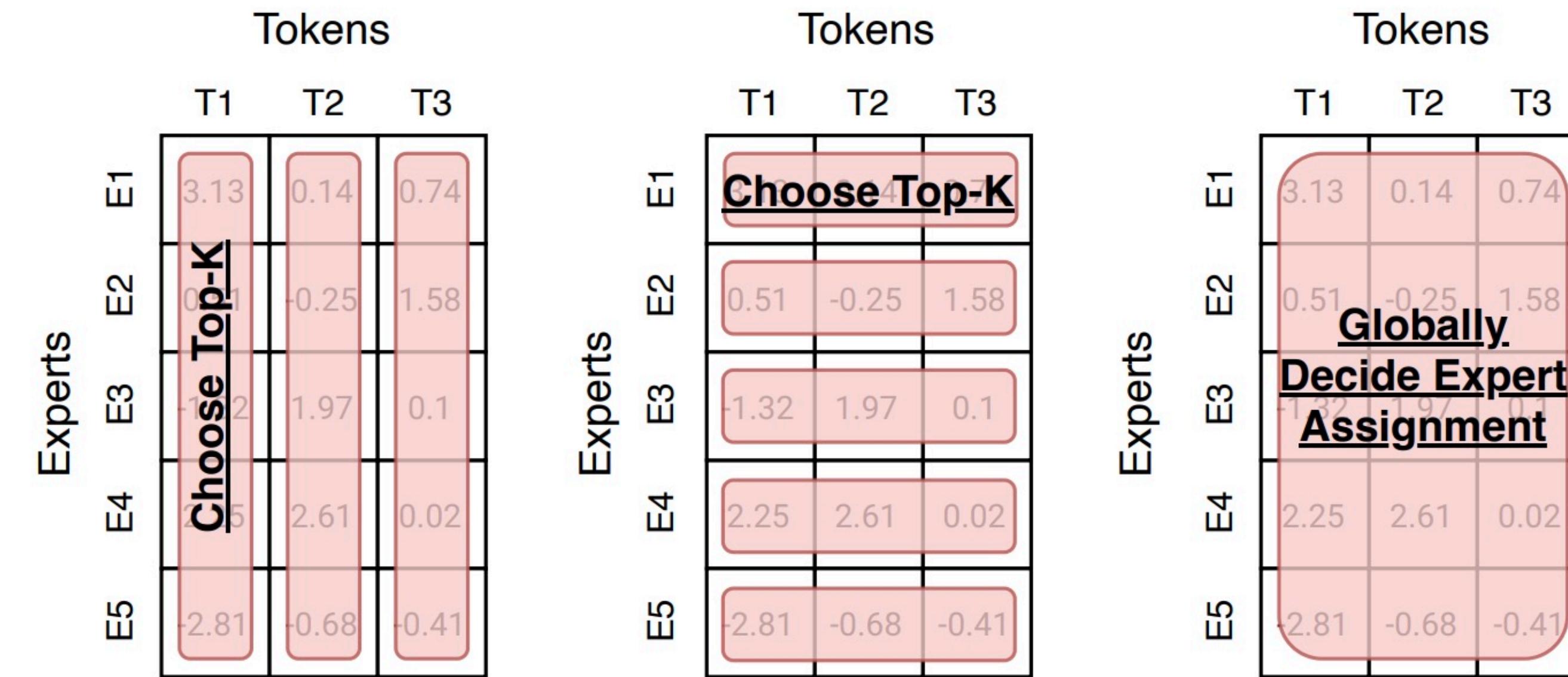


Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., 2022]

Mixture-of-Experts (MoE)

Sparsely activated for each token

- Different routing mechanisms



A Review of Sparse Expert Models in Deep Learning [Fedus et al., 2022]

Summary of Today's Lecture

In this lecture, we introduce:

1. Basics of transformers
2. Transformer design variants
3. Large language models (LLMs)
4. Advanced topics, multi-modal LLM

