Deep Learning

A course about theory & practice

UNIVERSITÀ DI PAVIA

Large Models
(LLMs, VLMs, and All That)

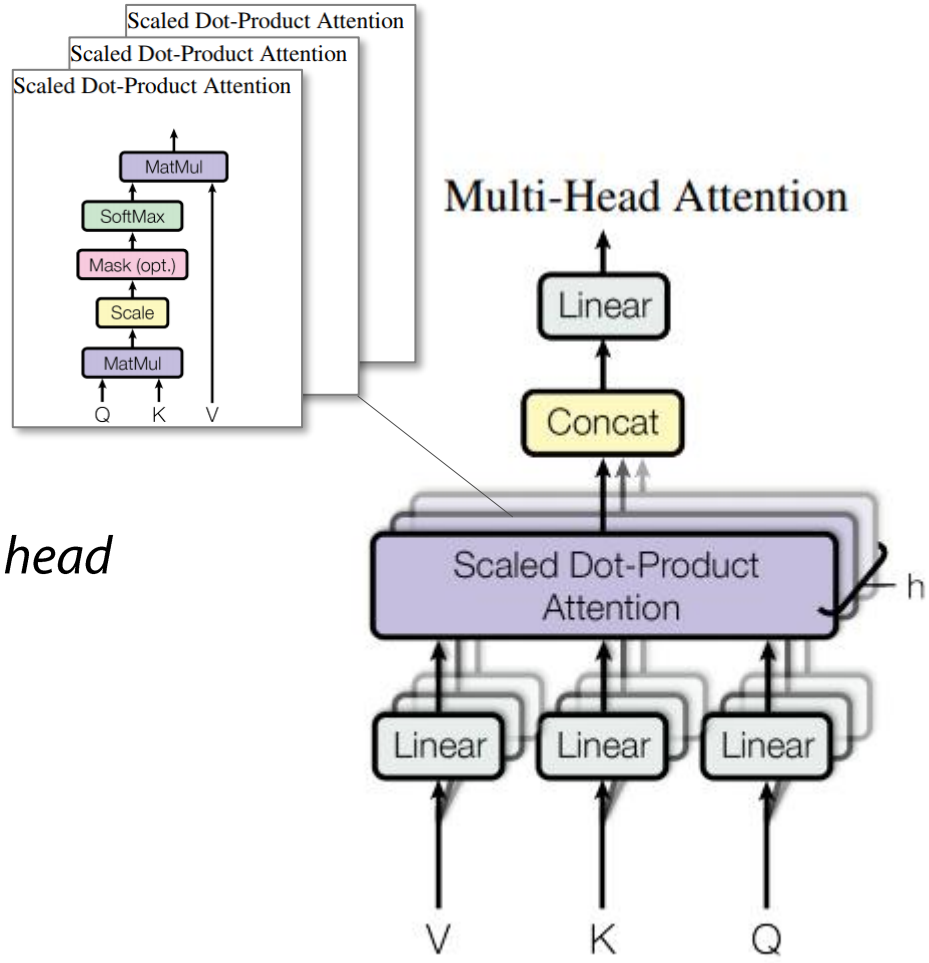
Marco Piastra

Transformers:
Main Architectural Patterns

Multiple Attention Heads

Multi-head attention consists of four parts:

1. Linear layers *(i.e., fully connected, no activation function)*
2. Scaled dot-product attention
3. Output concatenation
4. Final linear layer



Each input combination of Q (query), K (key), V (values)
is passed to each a separate linear layer hence to an attention head

The output of multiple attention heads is the concatenated
and fed to a final linear layer

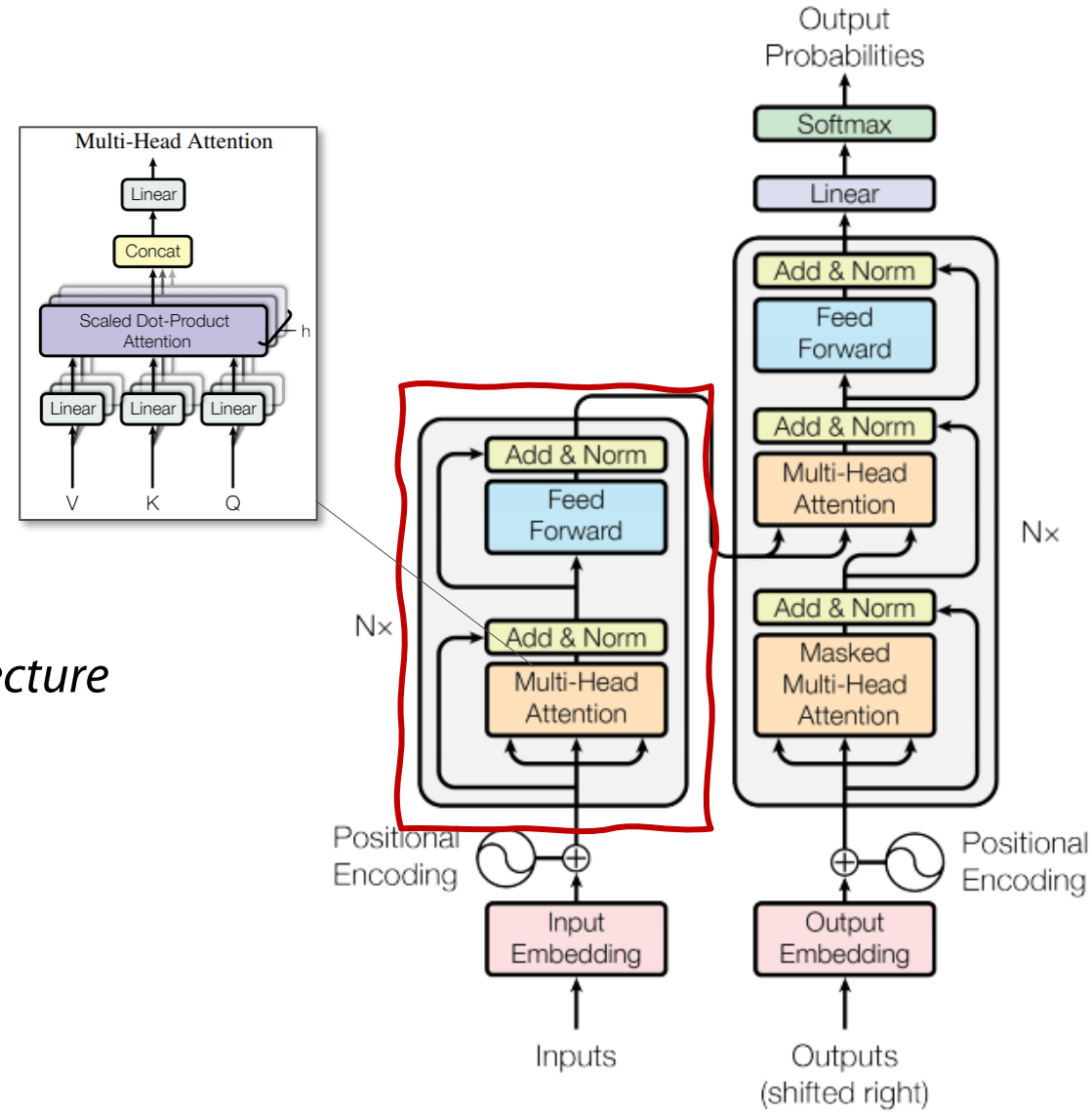
[image from https://arxiv.org/pdf/1706.03762.pdf]

Encoder Layer

Each encoder layer includes:

1. Multi-head attention
2. Addition *(ResNet style)*
3. Normalization *(per each input)*
4. Feed-forward network
   (one hidden layer with ReLU plus one linear layer)
5. Addition
6. Normalization

There could be many encoder layers in the overall architecture



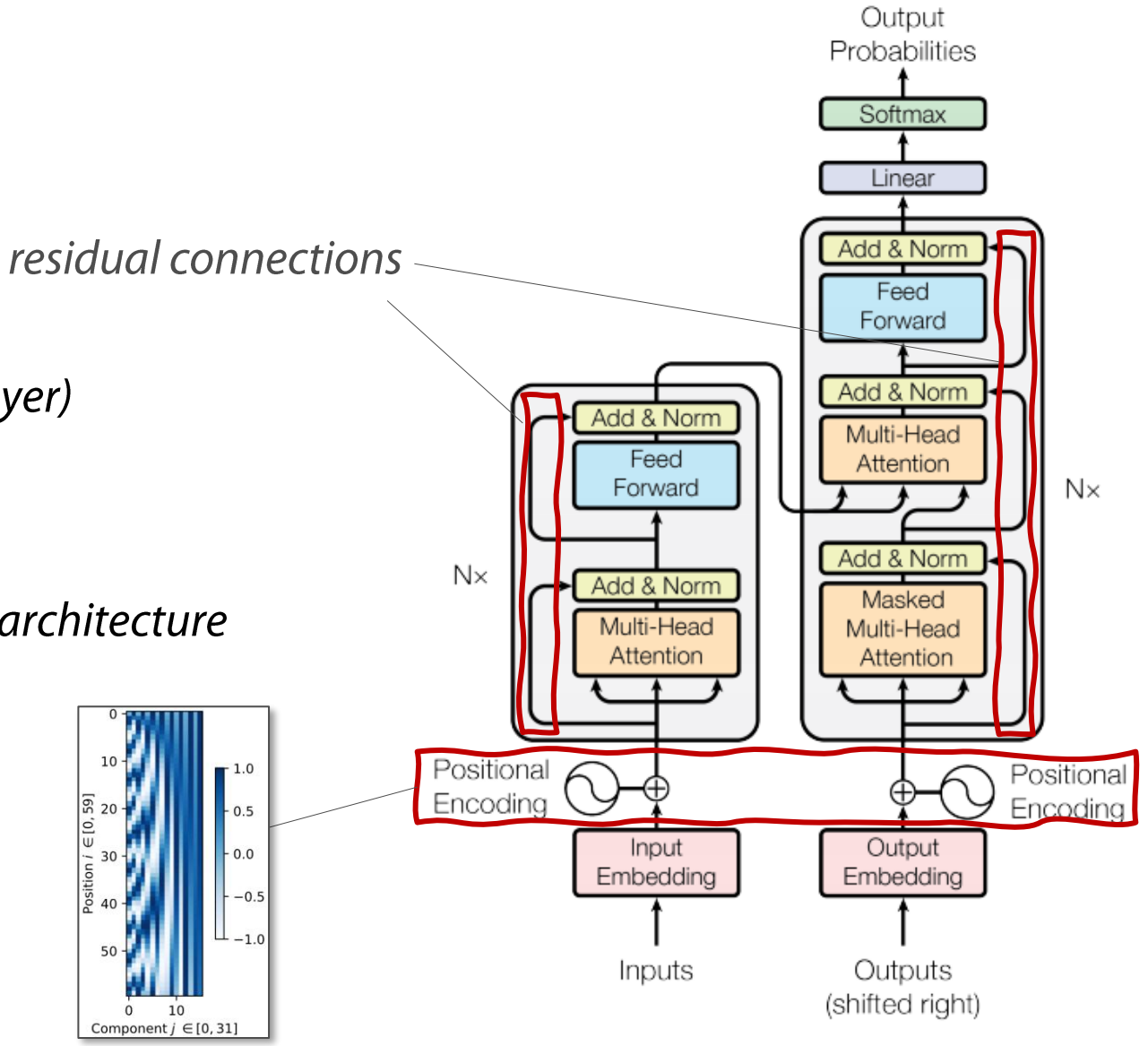
[image from https://arxiv.org/pdf/1706.03762.pdf]

Encoder Layer

Each encoder layer includes:

1. Multi-head attention
2. Addition *(ResNet style)*
3. Normalization *(per each input)*
4. Feed-forward network
   (one hidden layer with ReLU plus one linear layer)
5. Addition
6. Normalization

There could be many encoder layers in the overall architecture



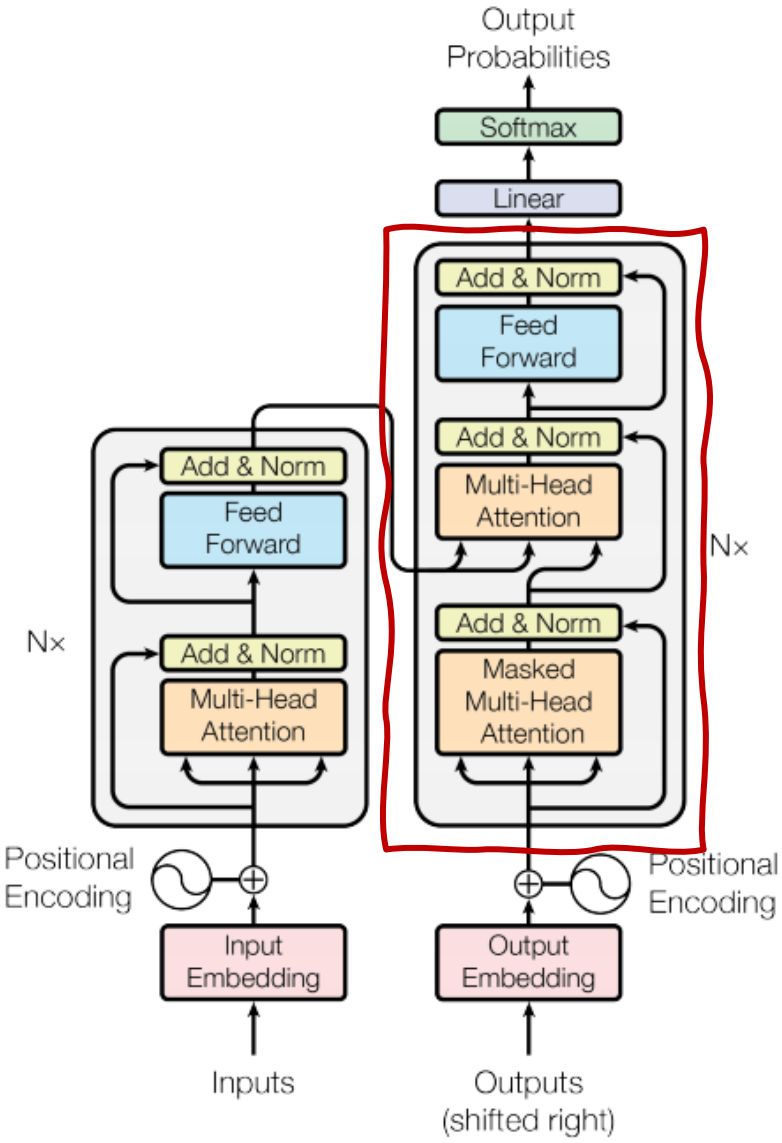
[image from https://arxiv.org/pdf/1706.03762.pdf]

Decoder Layer

Each decode layer includes:

1. Multi-head attention
2. Addition
3. Normalization
4. Multi-head attention
   values and *keys* come from the encoder output
   while *queries* come from the previous decoder layer
5. Addition
6. Normalization
7. Feed-forward network
   (one hidden layer with ReLU plus one linear layer)
8. Addition
9. Normalization

There could be many decoder layers in the overall architecture



[image from https://arxiv.org/pdf/1706.03762.pdf]

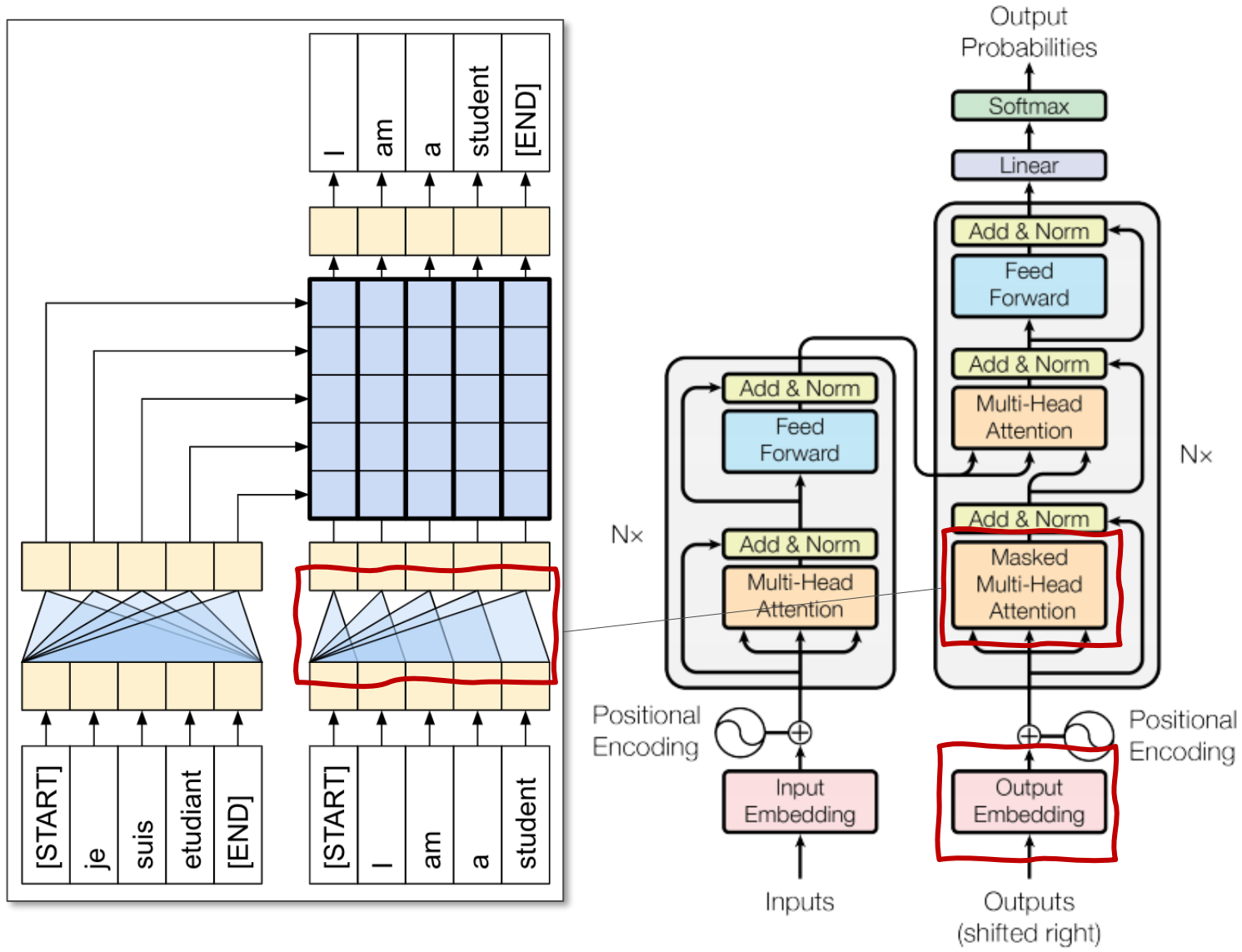
Decoder Layer

Why masked multi-head attention in the decoder layer?

The production of the output is incremental: one word at time

The output embedding 'input' can only see what has been generated thus far

Masks are not trained, they are 'superimposed' as the generation process advances

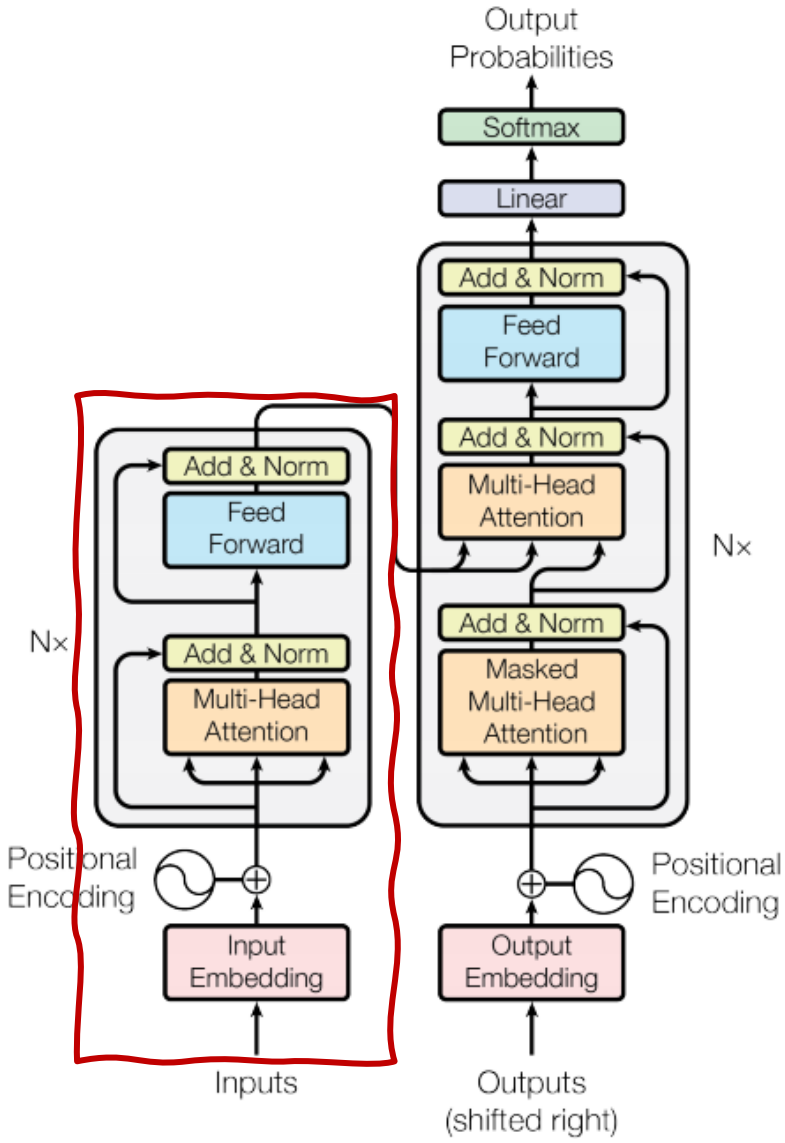


[image from https://arxiv.org/pdf/1706.03762.pdf]

Encoder

The encoder block includes:

1. Input embedding *(word2vec style)*
2. Positional encoding
3. Addition
4. N encoder layers

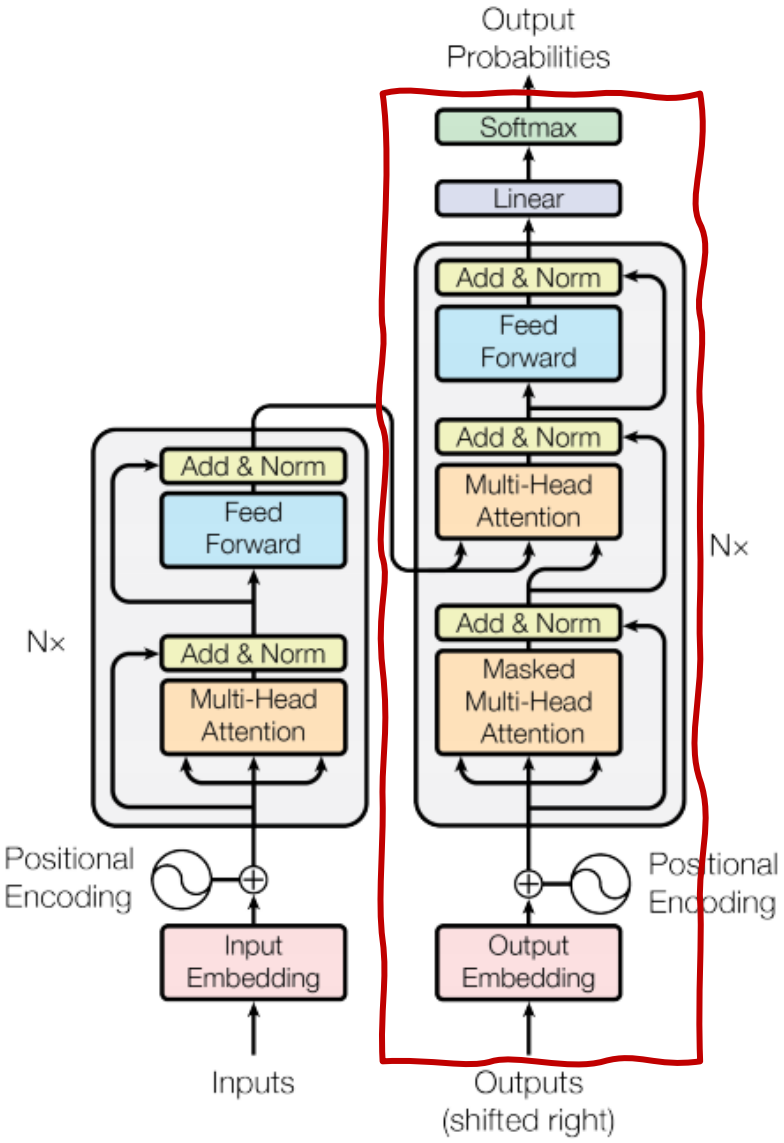


[image from https://arxiv.org/pdf/1706.03762.pdf]

Decoder

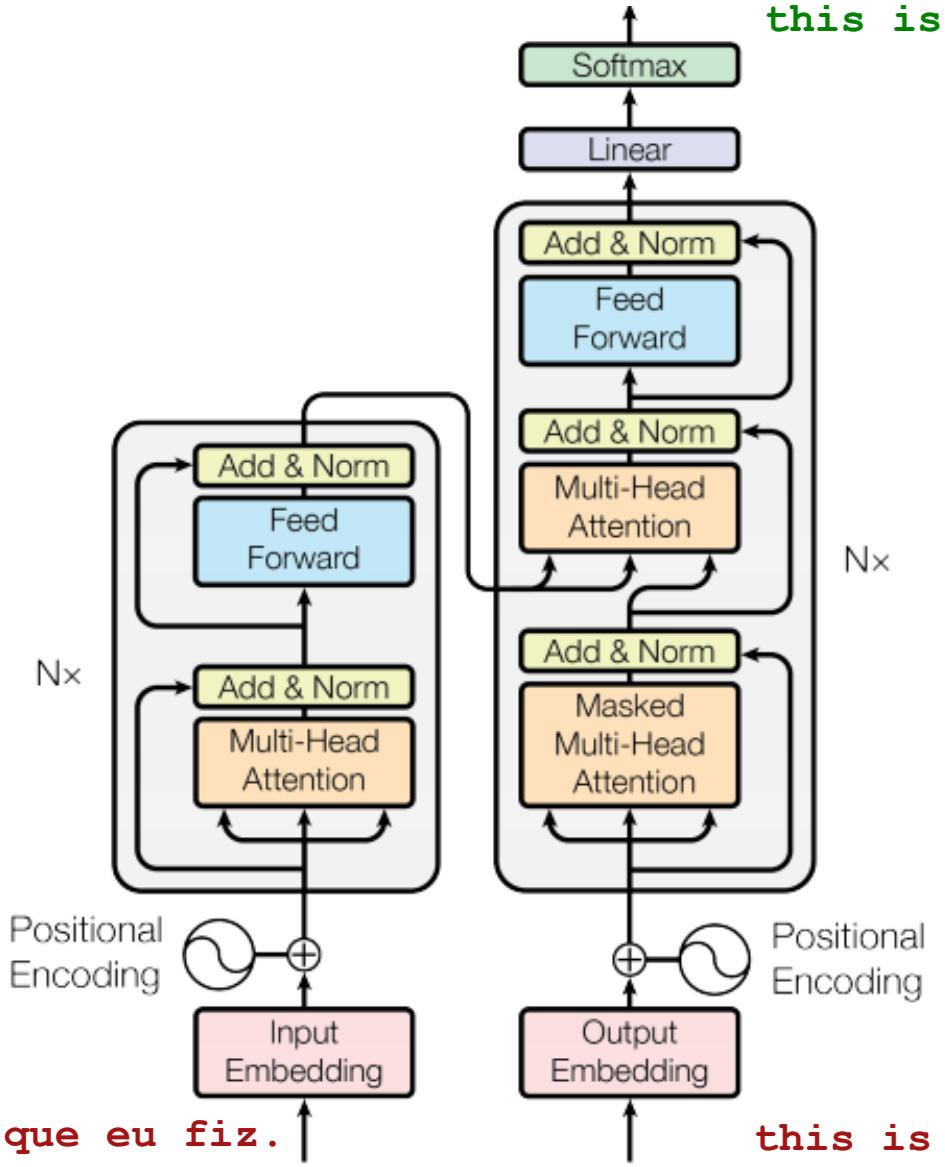
The decoder block includes:

1. Output embedding *(word2vec style)*
   It encodes the output produced so far
2. Positional encoding
3. Addition
4. N decoder layers
   Each connected to a corresponding encoder layer
5. Linear layer
6. Softmax layer
   It predicts the next token in the sequence



[image from https://arxiv.org/pdf/1706.03762.pdf]

Translator (Encoder-Decoder)



este é o primeiro livro que eu fiz.

this is the first book i've ever done.

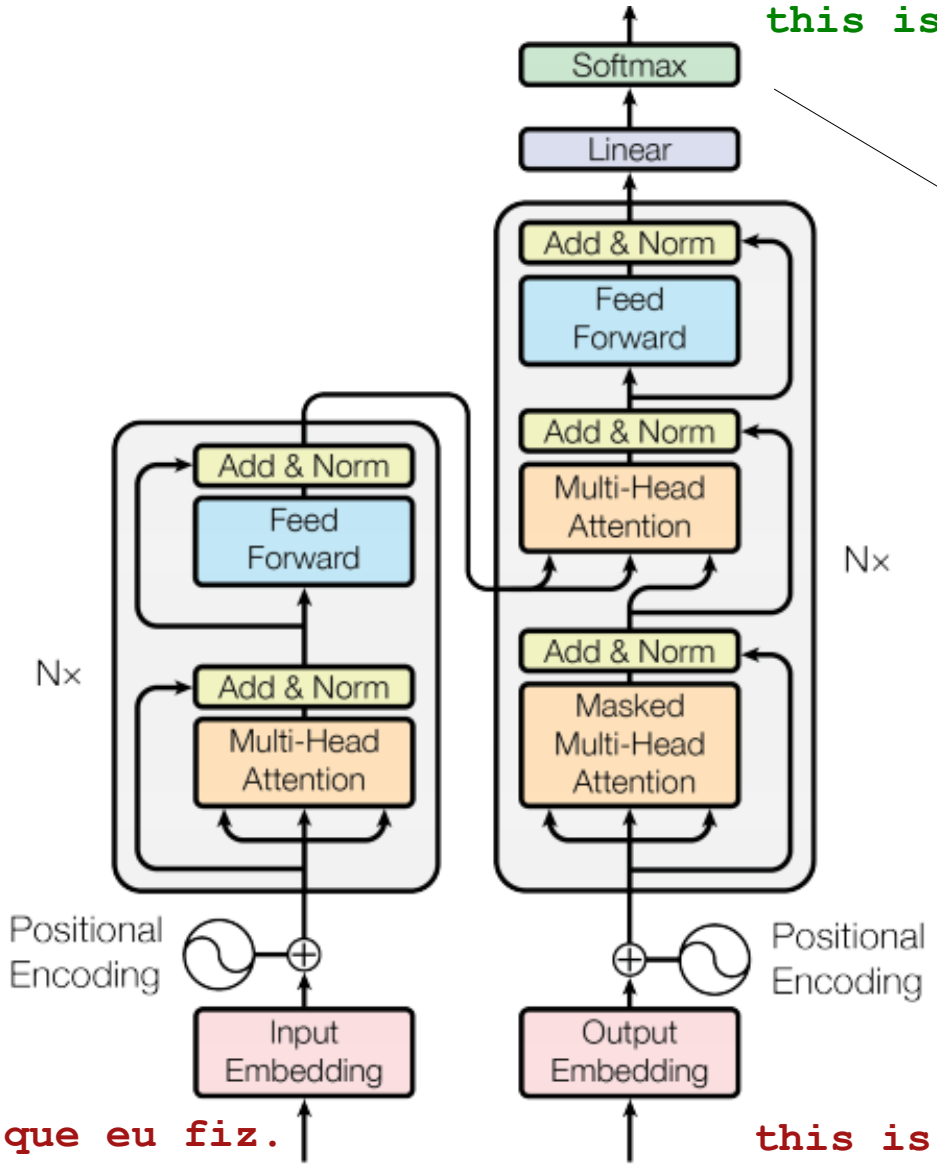
Feedback

this is the first book i've ever done

[image from https://arxiv.org/pdf/1706.03762.pdf]

Translator (Encoder-Decoder)

During Training



`this is the first book i've ever done.`

This is the predicted output

Loss is computed on the softmax

This one input (in one data item)

This the true output (teacher forcing)

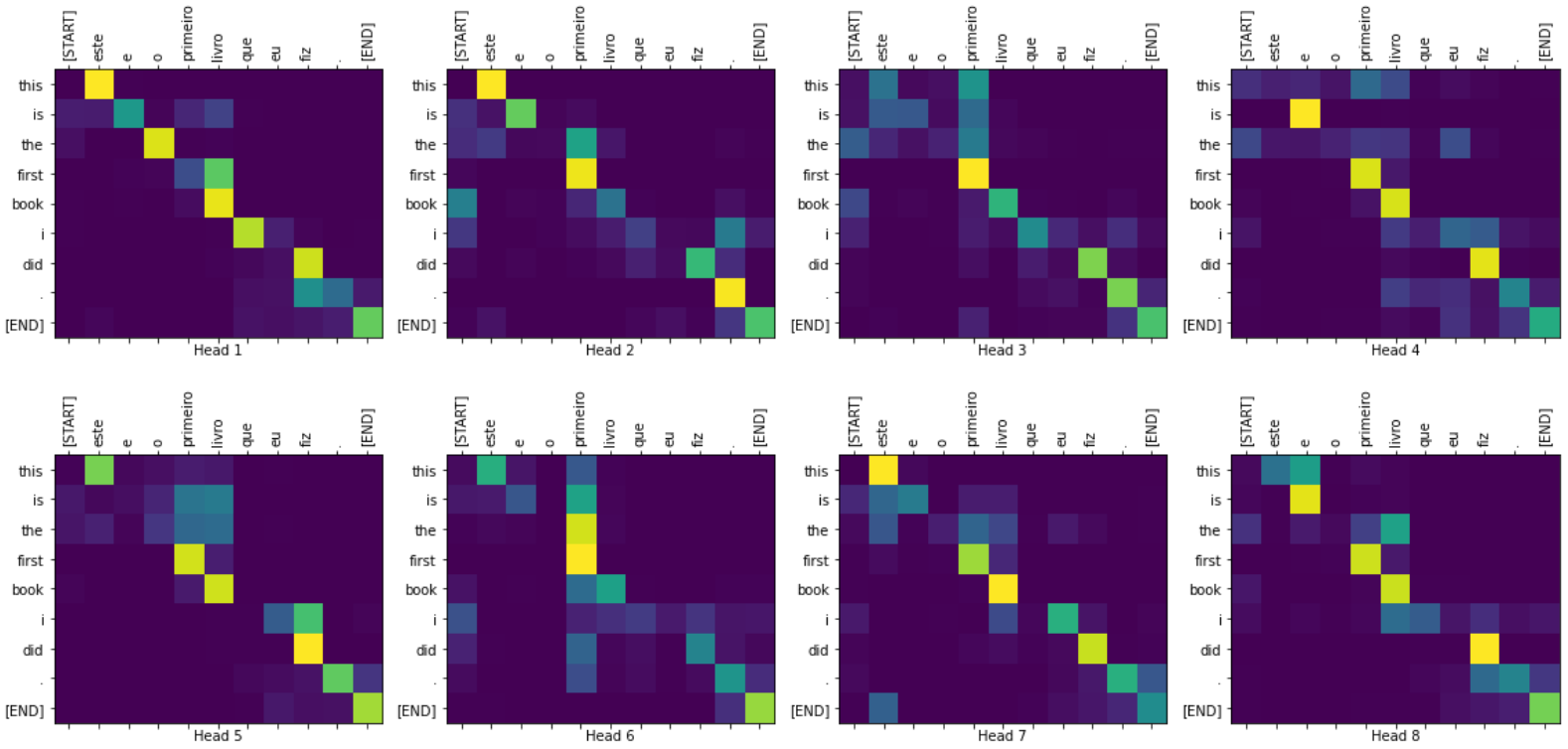
`este é o primeiro livro que eu fiz.`

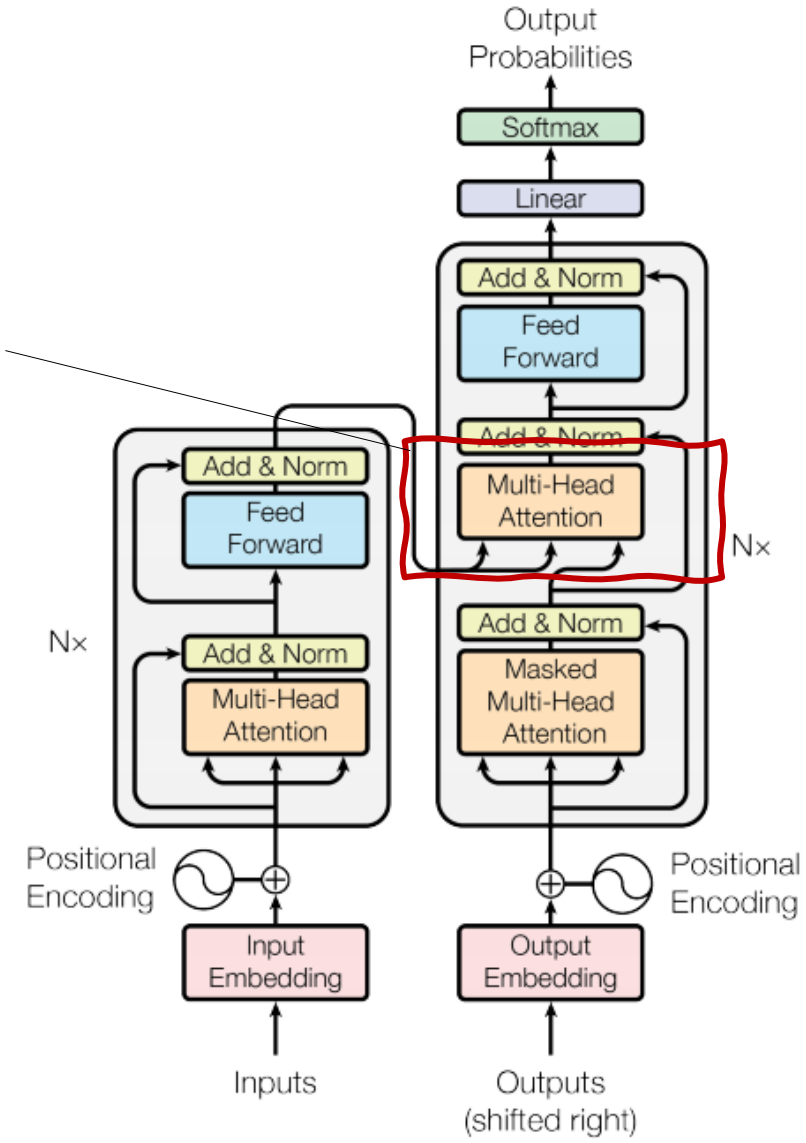
`this is the first book i've ever done`

[image from https://arxiv.org/pdf/1706.03762.pdf]

Attention Maps

Multiple heads, mixing layer ("Mixer"), topmost decoder block





[image from https://arxiv.org/pdf/1706.03762.pdf]

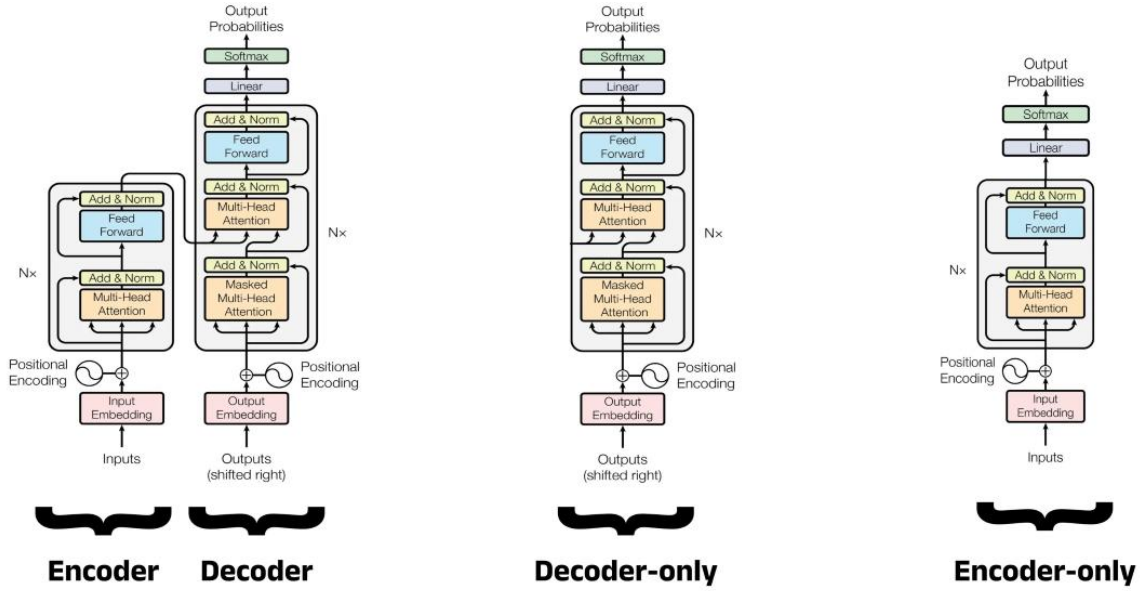
Attention in Blocks

1. Encoder (all layers)
   input tokens over input tokens (self-attention)

2. Decoder (lower layers)
   output tokens over output tokens (self-attention)

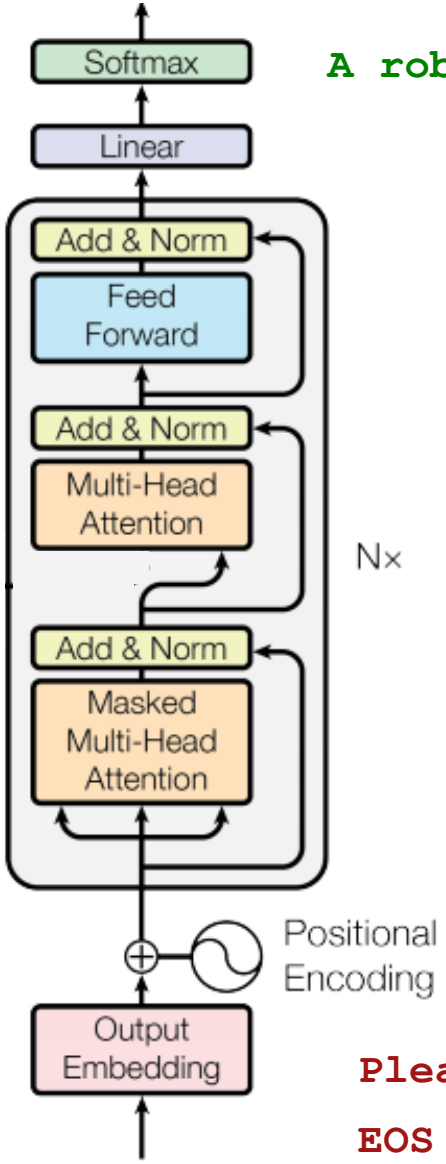
3. Decoder (top layer, “Mixer”)
   input tokens over output tokens (cross-attention)

Architectural Variants



Characteristic	Encoder-Only (e.g., BERT)	Decoder-Only (e.g., GPT, Gemma)	Encoder-Decoder (e.g., T2T, T5)
Attention Type	**Bidirectional:** Every token sees every other token.	**Causal:** Tokens only see themselves and previous tokens.	**Hybrid:** Bidirectional in Encoder; Causal in Decoder.
Primary Goal	**Understanding:** Extracting deep meaning from a full sequence.	**Generation:** Predicting the next token in a sequence.	**Transformation:** Mapping one sequence to another.
Training Task	Masked Language Modeling (Fill-in-the-blanks).	Causal Language Modeling (Predict-the-future).	Span corruption or Translation.
Use Cases	Classification, Sentiment Analysis, NER.	Chatbots, Creative Writing, Code Generation.	Translation, Summarization, Paraphrasing.

Chat (Decoder-only)



A robot may not injure a human being

When in chat mode, the transformer network only performs text completions

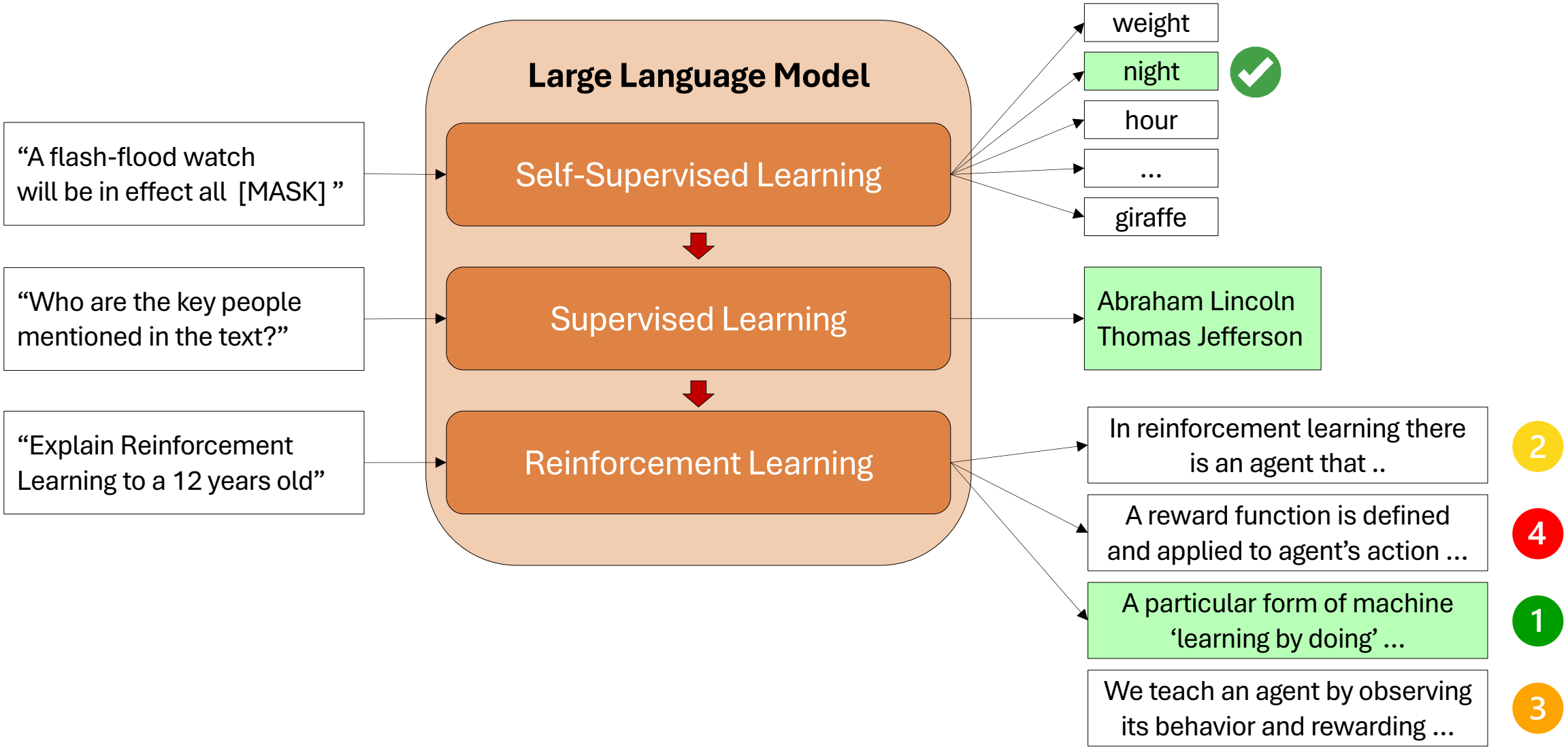
Please recite the first law of robotics

EOS A robot may not injure a human

[image from https://arxiv.org/pdf/1706.03762.pdf]

Training LLMs

LLMs learning: three main steps



LLMs learning: loss function

Token prediction: cross-entropy

Any LLM architecture can be seen as a parametric probability distribution

It yields the probability of a token given a context (whatever that may be)

$$p(token \mid context)$$

The probability distribution is over the set of *all tokens*

The basic loss function is *cross-entropy*:

$$L(token, context) := -\log p(token \mid context)$$

ground truth for predicted token

The loss is computed only over those tokens to be predicted (***Masked Cross-Entropy Loss***)

LLMs learning: loss function

Self-supervised step

Cross-entropy loss is measured only for the missing token, given the masked context

Supervised step

Cross-entropy loss is measured for completion tokens, given the previous part of the sequence as context

Multiple tokens (sequence)

Cross-entropy loss is accumulated along the sequence

$$L(token\_sequence, context) := -\frac{1}{n}\sum_{i=1}^{n} \log p(token \mid context + token\_sequence_{<i})$$

Reinforcement Learning

Stay tuned, we will deal with this topic later on…

LLMs learning: loss function

Self-supervised step

Cross-entropy loss is measured only for the missing token, given the masked context

Supervised step

Cross-entropy loss is measured for completion tokens, given the previous part of the sequence as context

Multiple tokens (sequence)

Cross-entropy loss is accumulated along the sequence

$$L(token\_sequence, context) := -\frac{1}{n}\sum_{i=1}^{n} \log p(token \mid context + token\_sequence_{<i})$$

Reinforcement Learning

Stay tuned, we will deal with this topic later on…

LLMs learning: most used text corpora for training

Corpus Name	Description	Size / Language Coverage	Notable Usage / Models
Common Crawl	Massive web crawl dataset, billions of web pages	Petabytes; multilingual	GPT, LLaMA, Falcon, BLOOM
Common Corpus	Largest public domain dataset, diverse and multilingual	2 trillion tokens; multilingual	Open LLMs, research
Wikipedia	Cleaned encyclopedic articles, all topics	Millions of articles; multilingual	GPT, T5, RoBERTA
BooksCorpus	11,000+ unpublished books, narrative text	740M–985M words; English	RoBERTA, XLNet
OpenWebText	Web pages curated to mimic OpenAI's WebText (Reddit-based)	Billions of tokens; English	GPT-2, GPT-Neo
C4 (Colossal Clean Crawled Corpus)	Cleaned English web crawl, deduplicated	750GB; English	T5, MPT-7B
The Pile	22 diverse academic/professional datasets, broad coverage	800GB; mostly English	GPT-Neo, LLaMA, OPT
ROOTS	Multilingual, deduplicated web/code/crowdsourced text	1.6TB; 59 languages	BLOOM
Google News	News articles from 70,000+ sources	Large; multilingual	Topic modeling, classification
Starcoder Data	Programming code from GitHub, Jupyter, etc.	783GB; 86 languages (code)	Starcoder, CodeGen

Attention Variants & Alternatives: Beyond Plain Transformers

Rotary Position Embedding (RoPE)

Ensure that the positional encoding does not vanish as processing proceeds

NoPE + RoPE

When RoPE is used, no *positional encoding* is added to the input (NoPE)

The token position is encoded anew at each attention layer

1. Each token index *m* is explicitly propagated through the layers
2. Each token encoding in Q and K is divided in *pairs*

$$\begin{aligned}\boldsymbol{x} &= [x_0, \ldots, x_d] \\ &= [(x_0, x_1), (x_2, x_3), \ldots (x_{d-1}, x_d)]\end{aligned}$$

3. Each pair is rotated by a specific angle, depending on index m and pair index i

$$\begin{pmatrix} x'_{2i} \\ x'_{2i+1} \end{pmatrix} = \begin{pmatrix} \cos(m\theta_i) & -\sin(m\theta_i) \\ \sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix} \begin{pmatrix} x_{2i} \\ x_{2i+1} \end{pmatrix}$$

where θ_i is a constant angle that gets smaller (slower rotation) as the pair index i increases

When computing the dot product $Q_m \cdot K_n$ between token encodings, we have

$$\cos(m\theta - n\theta) = \cos((m-n)\theta)$$

Only the relative distance matters

KV Caching

Save computation time by storing intermediate token-wise results

Note that:

In a transformer block, all operations on Q are performed row-wise (i.e., *token-wise*)

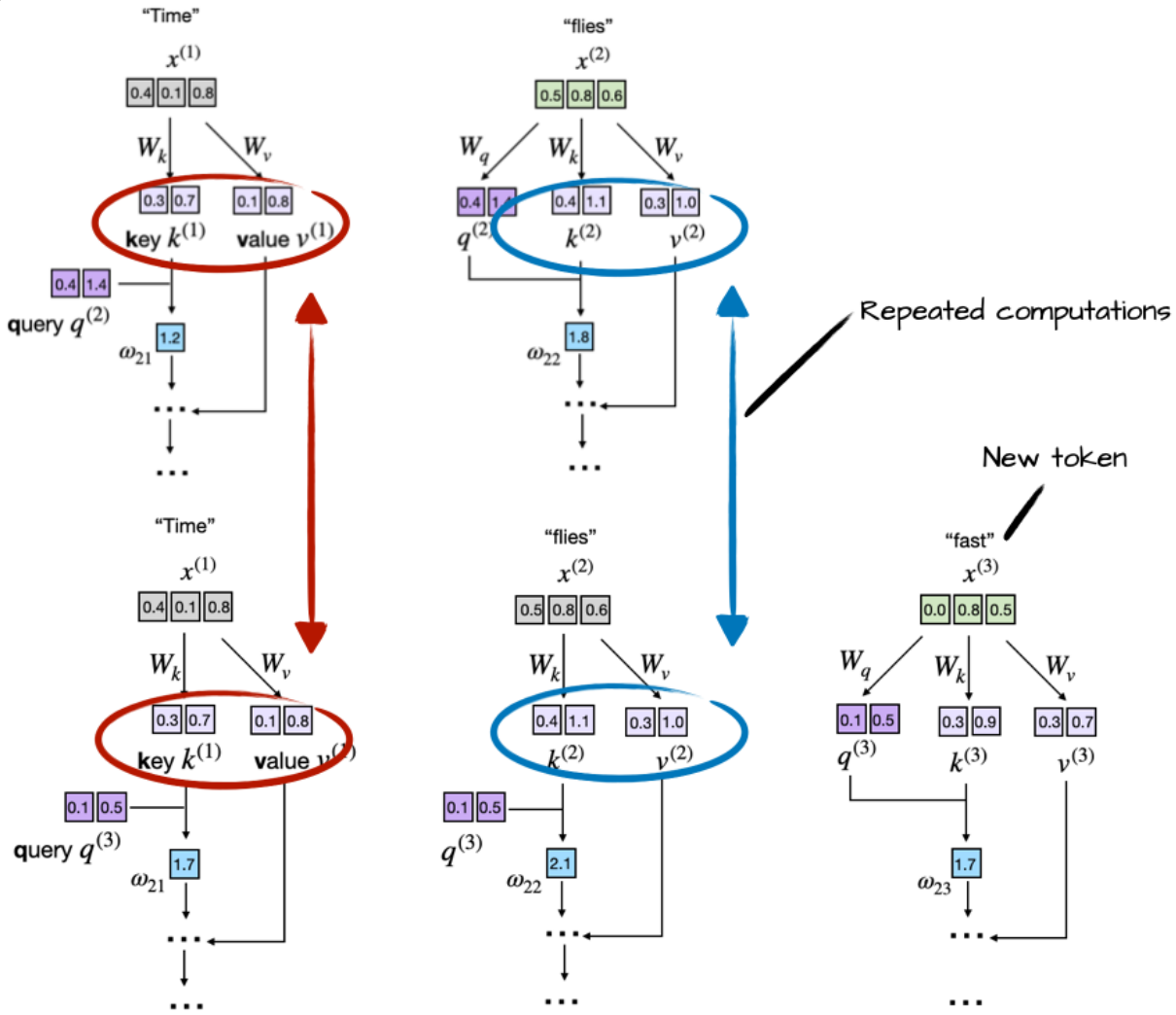
"Time flies **fast**"
("**fast**" is the t-th token in the sequence)

In standard, non-cached architecture
all matrices Q, K, V
are recomputed from scratch

Yet, all $t-1$ rows in V (the output)
will remain unaltered

Solution:

At inference time, cache the $t-1$ rows in K, V and compute only the new lines for token t



[image from https://github.com/rasbt/LLMs-from-scratch/tree/main/ch04/03_kv-cache]

Grouped-Query Attention (GPA)

Lower the number of parameters and save memory

Multi-Head Attention (MHA) entails having separate parameters

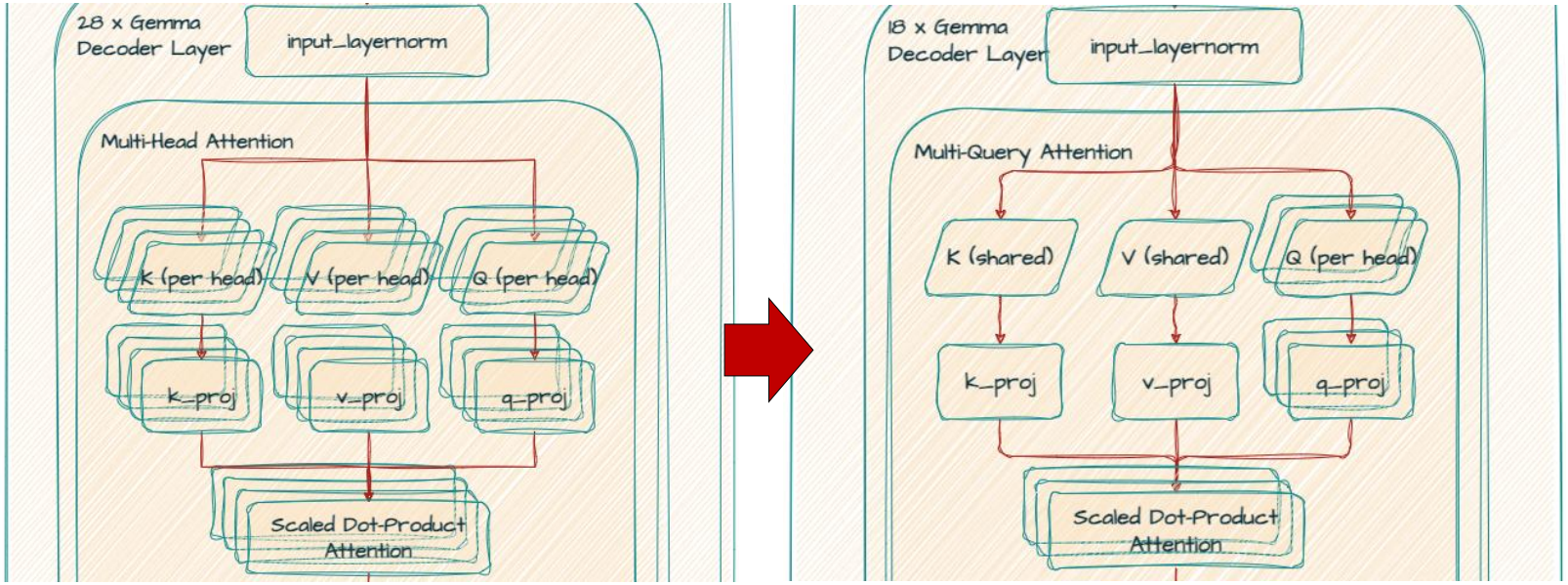
$W_{Q1}, \ldots, W_{Qk}$ $W_{K1}, \ldots, W_{Kk}$ $W_{V1}, \ldots, W_{Vk}$

to compute $Q_1, \ldots, Q_k$ $K_1, \ldots, K_k$ $V_1, \ldots, V_k$

With Grouped-Query Attention (GQA) the K and V parameters are shared across the multi-head transformer

Given the quadratic parameter complexity of parameters K and V with respect to context window size GQA introduces substantial savings (and allows larger sizes)

Empirical evidence shows that for most LLM tasks, this compression is remarkably lossless



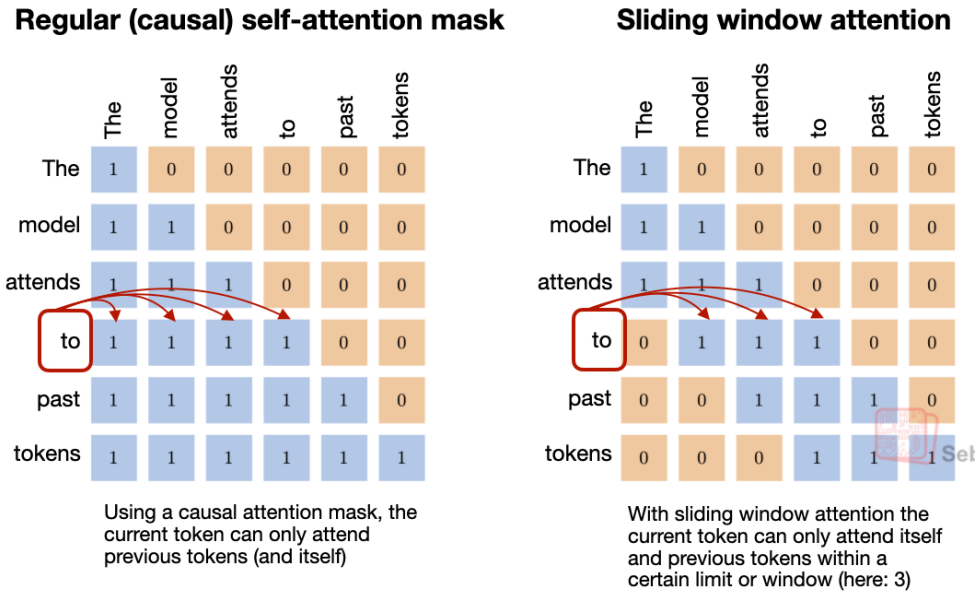
[image from https://developers.googleblog.com/gemma-explained-overview-gemma-model-family-architectures/]

Sliding Window Attention (SWA)

Allowing larger context windows

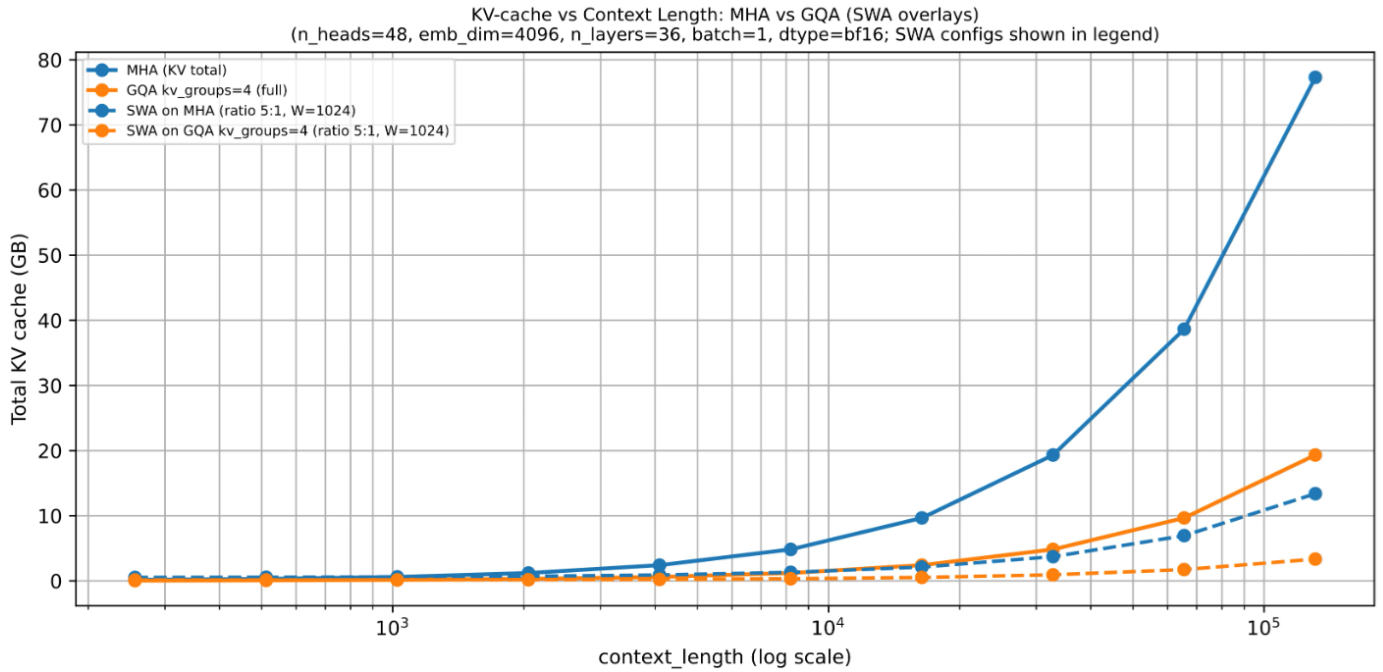
Introducing a limit on how many previous tokens each position can attend to

Each token only attends to a fixed window of recent tokens around its position



Typically, SWA is used in combination with GPA and KV caching

In this cases, the reduction in the number of parameters can be substantial

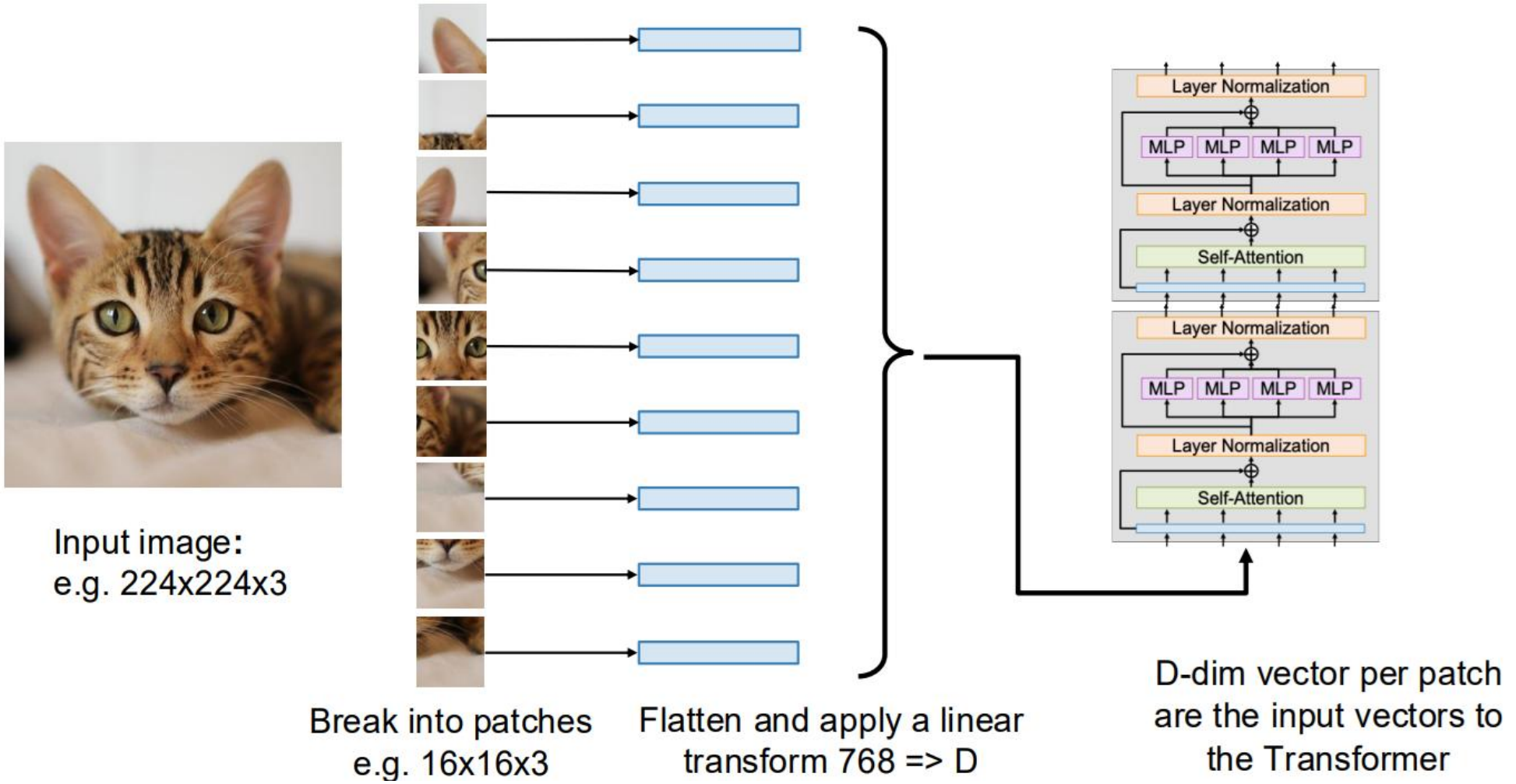


[images from https://sebastianraschka.com/llm-architecture-gallery/swa/]

Multimodality: Vision Transformers

Vision Transformers Basics

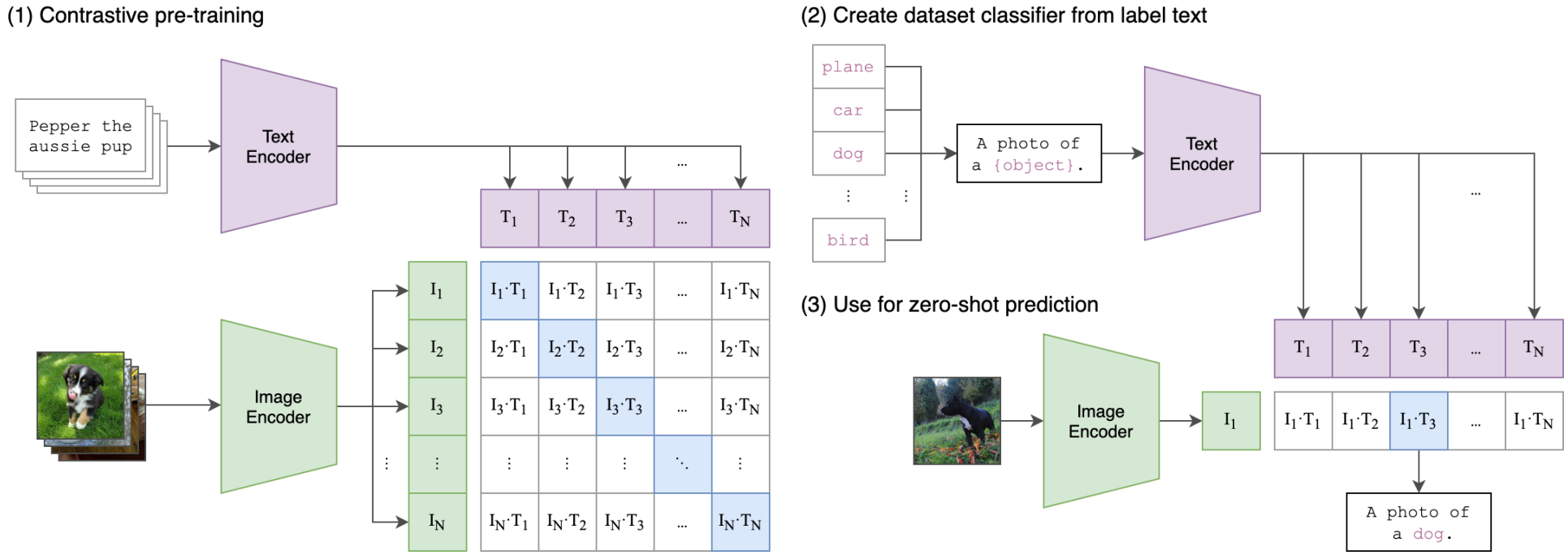
Image patches instead of language tokens



[image from https://cs231n.stanford.edu/slides/2026/lecture_8.pdf]

Contrastive Language-Image Pre-Training (CLIP)

Connecting text and images



CLIP pre-trains an image encoder and a text encoder
to predict which images were paired with which texts in the dataset

All dataset's classes are translated into captions such as "a photo of a dog"

CLIP predicts the class of the caption by estimating best pairs with a given image

[image from https://github.com/openai/CLIP]

Multi-Modal Vision Language Models

Multimodal Vision Language Models

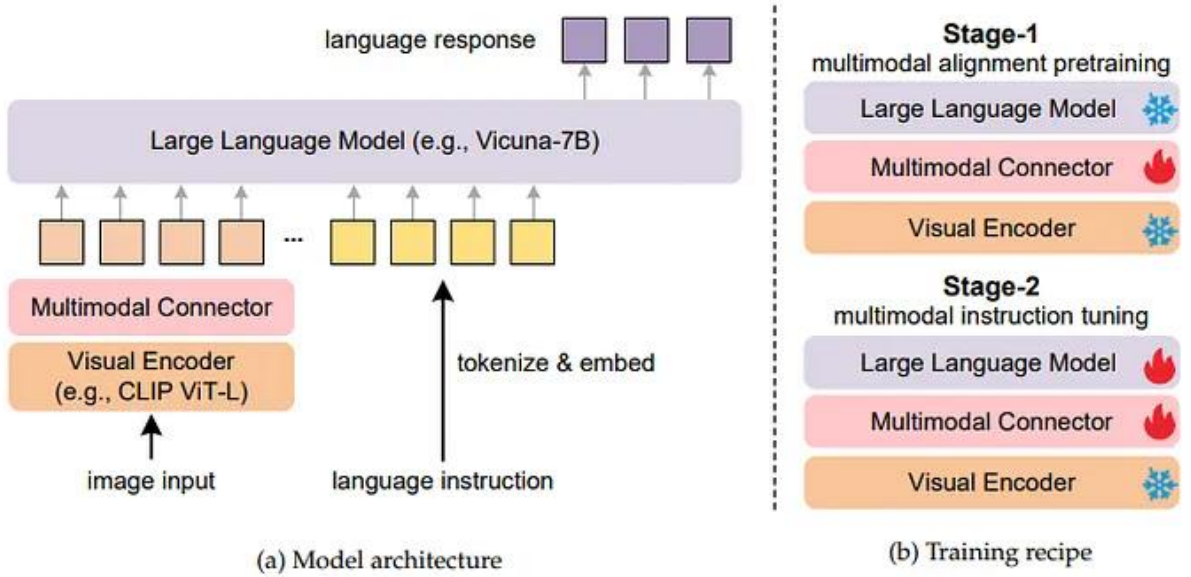


Figure 1 | LLaVA-1.5's model architecture and its two-stage training recipe.

1. Input images are processed by the vision encoder
2. Visual features are mapped to the LLM input space, creating *embeddings* for *visual token*
3. Visual tokens are concatenated (and interleaved) with the input sequence of *text embeddings*
4. The sequence of visual and textual tokens is fed into LLM, to predict the output text tokens

Multi-Modal Vision Language Models

Multimodal Vision Language Models ***Training***

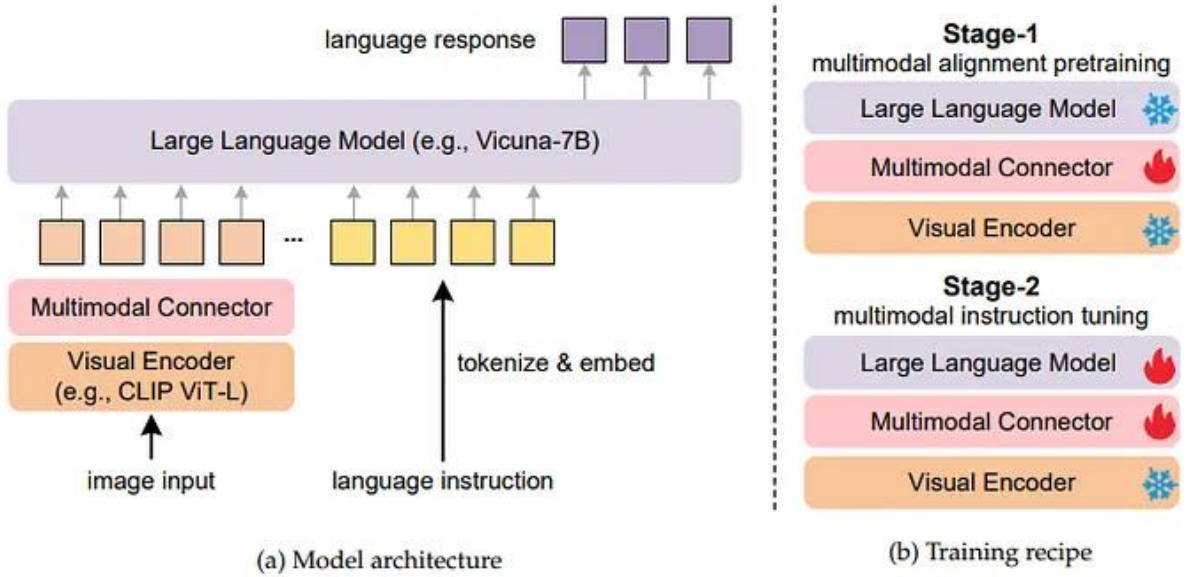


Figure 1 | LLaVA-1.5's model architecture and its two-stage training recipe.

1. **Pre-training**: keep Visual Encoder and LLM frozen, train the Connector
2. **Multimodal Instruction Fine Tuning**: unfreeze the LLM and make it learn to follow linguistic instructions
3. **RLHF**: like with LLM, align the model with human preferences

Multi-Modal Vision Language Models

Multimodal Vision Language Models ***Inference***

Large Language and Vision Assistant (LLaVA)

Visual input example, Different Format Prompts:	
Normal prompt	What is the color of the shirt that the man is wearing?
Response	The man is wearing a yellow shirt.
Ambiguous prompt	Q: What is the color of the shirt that the man is wearing? A:
Response	The man is wearing a yellow shirt.
Formatting prompt	What is the color of the shirt that the man is wearing? **Answer the question using a single word or phrase.**
Response	Yellow.

[image from https://arxiv.org/pdf/2310.03744]

Links

https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Language Models are Unsupervised Multitask Learners

https://arxiv.org/pdf/1904.02679

Visualizing Attention in Transformer-Based Language Representation Models

https://arxiv.org/pdf/2203.02155

Training language models to follow instructions with human feedback

https://medium.com/@row3no6/why-chatgpt-uses-decoder-only-eaf0223143e6

Why ChatGPT Uses Decoder-Only

https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Decoder-Only Transformers: The Workhorse of Generative LLMs