Attention / Transformer / GPT

Neural-LM [2003]

1. Bengio et al., “A neural probabilistic language model“, NIPS 2000
2. Bengio et al., “A neural probabilistic language model”, JMLR 2003.

RNN-LM [2010]

T. Mikolov, M. Karafi´at, L. Burget, J. Cernock´y, and S. Khudanpur. “Recurrent neural network based language model”, In INTERSPEECH, pages 1045–1048, 2010.

Seq2seq learning (Encoder-Decoder) [2014]

Sutskever, I., Vinyals, O., & Le, Q. V., “Sequence to sequence learning with neural networks”, Advances in Neural Information Processing Systems (pp. 3104–3112), 2014

Attention [2015]

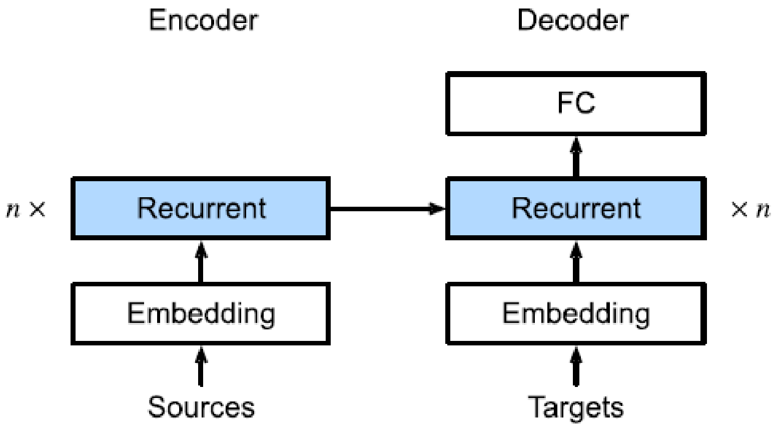
Bahdanau, D., Cho, K., & Bengio, Y., “Neural machine translation by jointly learning to align and translate”, Proc. ICLR 2015

Transformer [2017]

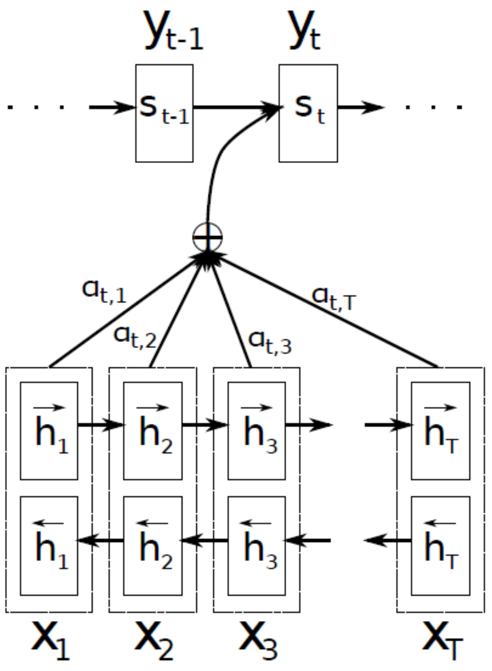
A. Vaswani et al., “Attention is all you need”, In NIPS, pages 6000–6010, 2017.

GPT [2018]

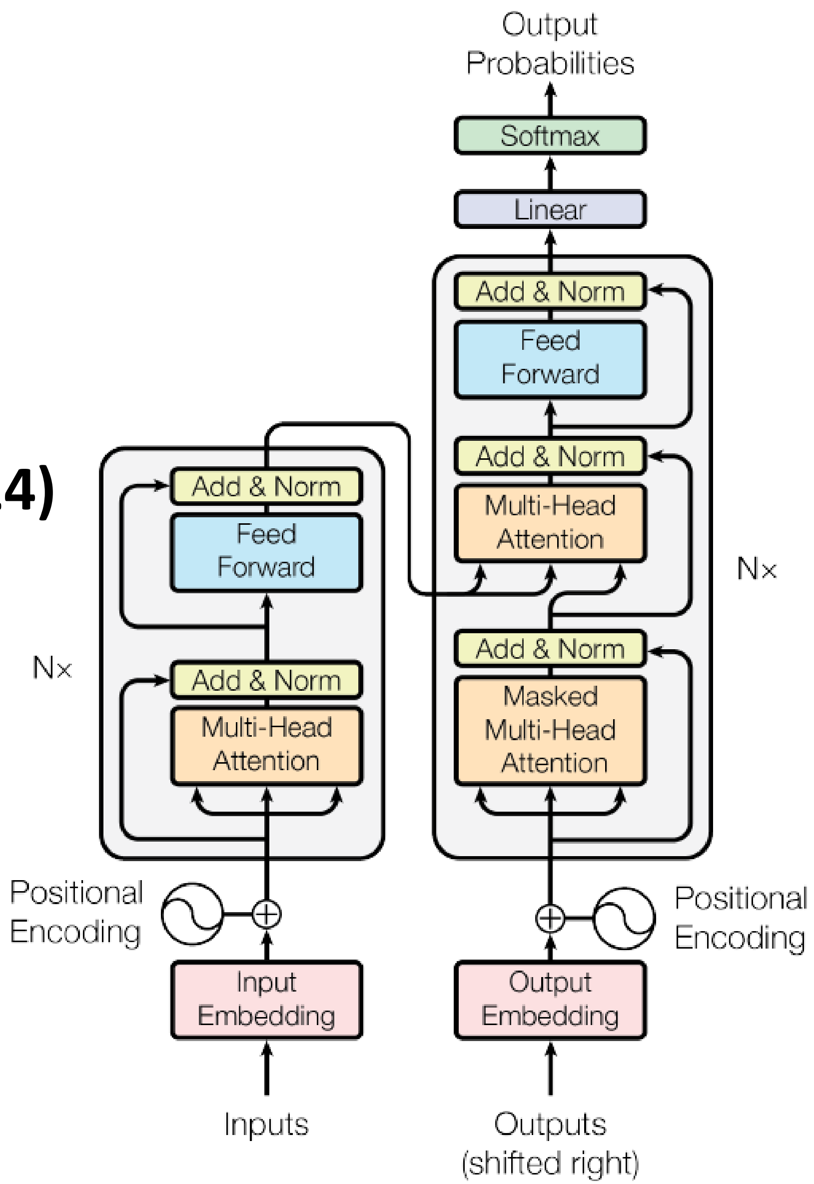
A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, “Improving Language Understanding by Generative Pre-Training”, 2018 (arXiv)



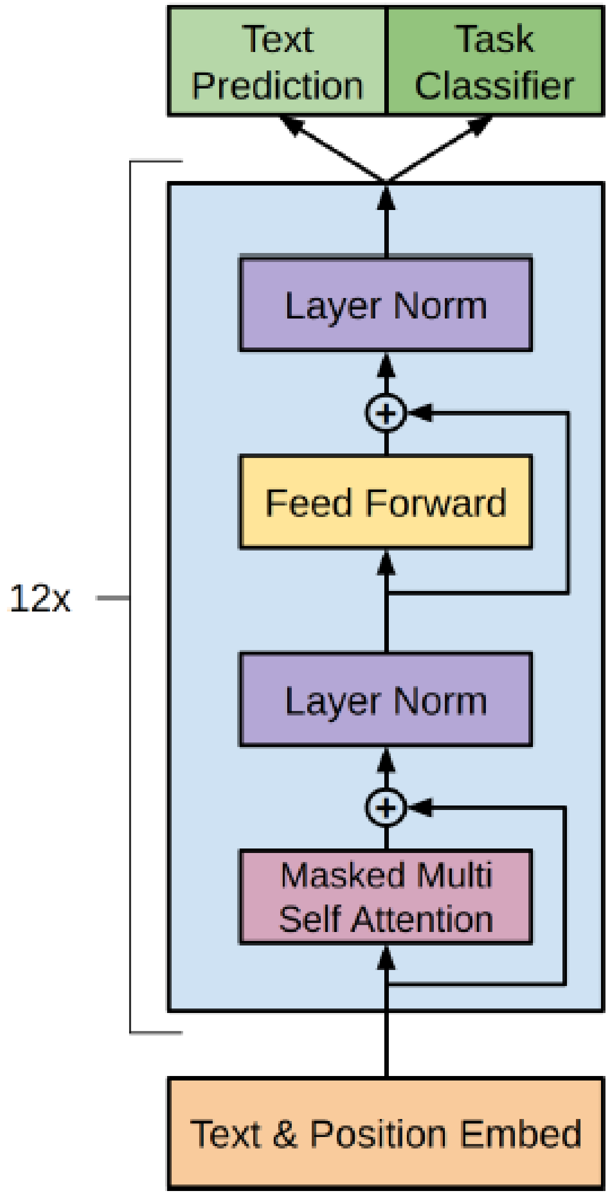
RNN Encoder-Decoder Model (2014)



Bahdanau Attention (2015)



Transformer Model (2017)



Decoder-only GPT (2018)

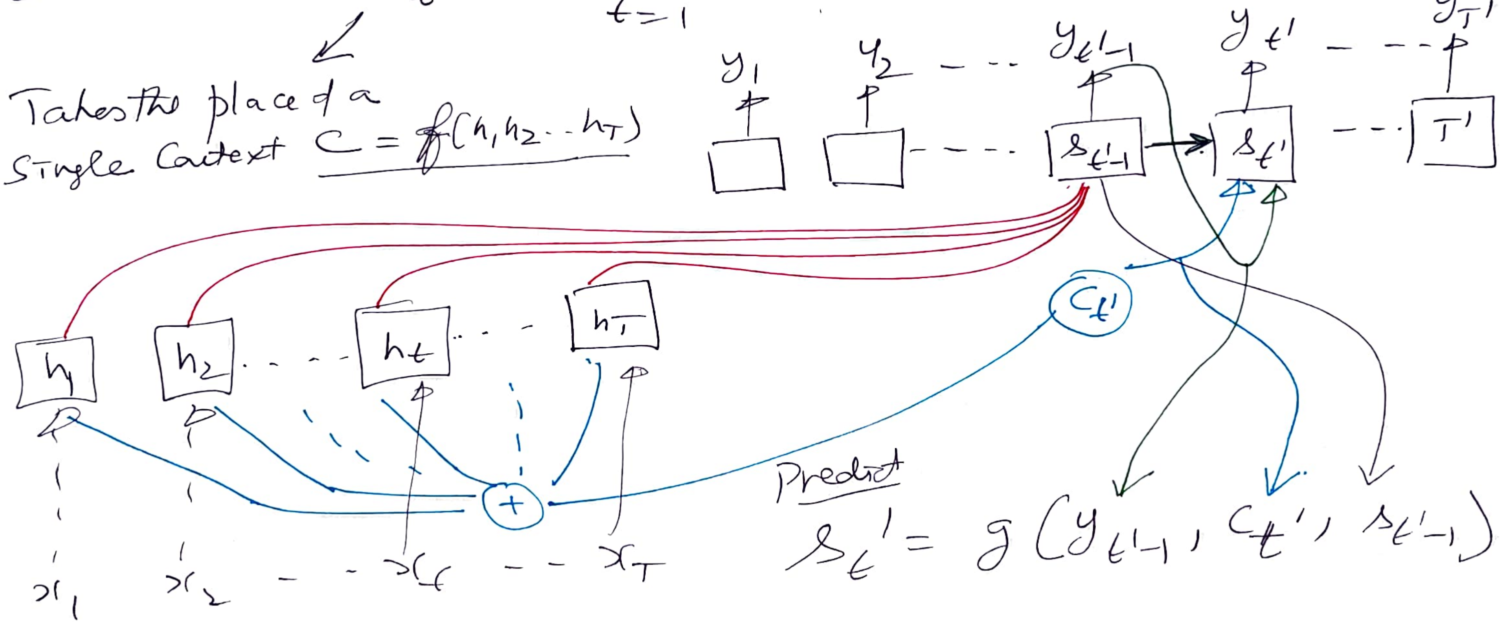
Bahdanau A·M $\Rightarrow$ Most influential idea in past decade in DL.

$\Rightarrow$ Giving rise to TRANSFORMERS [Vaswani, 2017]

BAM Model

$$c_{t'} = \sum_{t=1}^{T} \alpha(\underbrace{s_{t'-1}}_{\text{QUERY}}, \underbrace{h_t}_{\text{KEY}})\, \underbrace{h_t}_{\text{VALUE}}$$

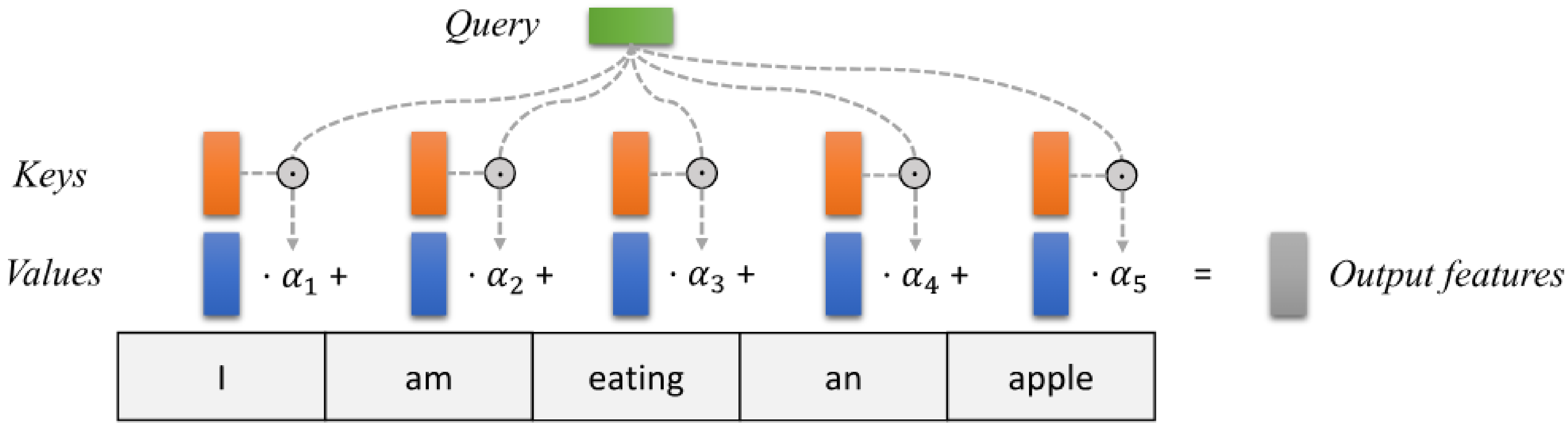
Takes the place of a single context $C = f(h_1, h_2 \ldots h_T)$



Predict

$$s_{t'} = g(y_{t'-1}, c_{t'}, s_{t'-1})$$

Attention

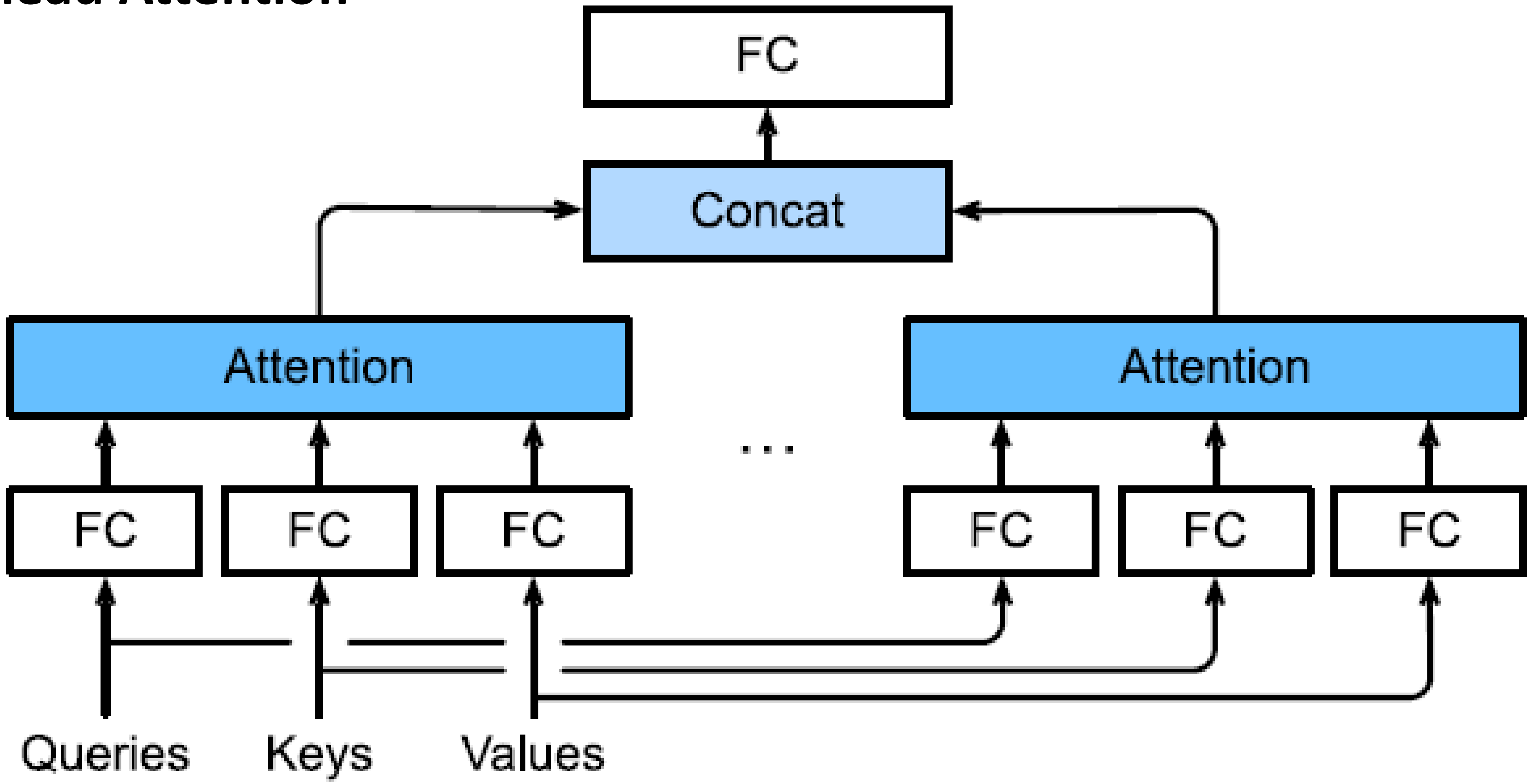


Query
Keys
Values
$\cdot\, \alpha_1$ +
$\cdot\, \alpha_2$ +
$\cdot\, \alpha_3$ +
$\cdot\, \alpha_4$ +
$\cdot\, \alpha_5$
=
Output features
I
am
eating
an
apple

MULTI-HEAD ATTENTION

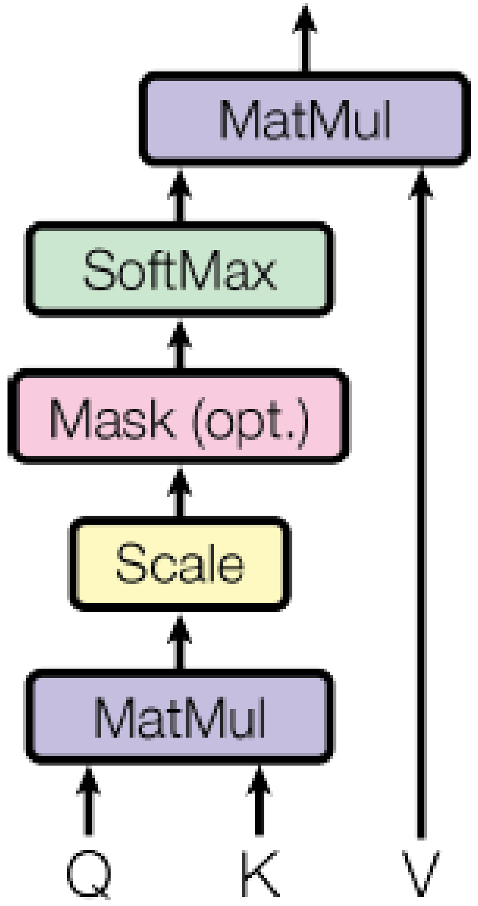
- Capture short range & long-range dependencies within a sequence
- Given a same set Q, K, V
  - Q — s'_{t-1}
  - K — h_t
  - V — h_t
- $Q, K, V \Rightarrow$ Transformed with "h" independently learned linear projection [FC layers]
- These "h" projected Q, K, Vs are fed into Attention Pooling in Parallel.
- Each of "h" attn pooling o/p : "head"

Multi-head Attention

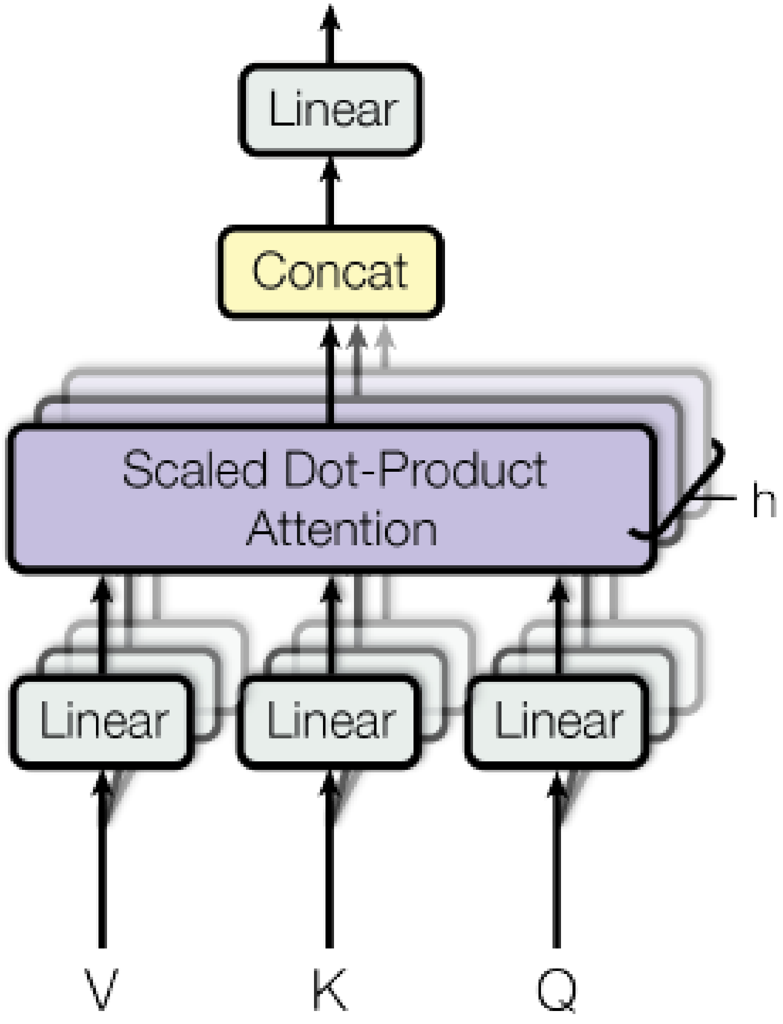


Multi-head attention, where multiple heads are concatenated then linearly transformed.

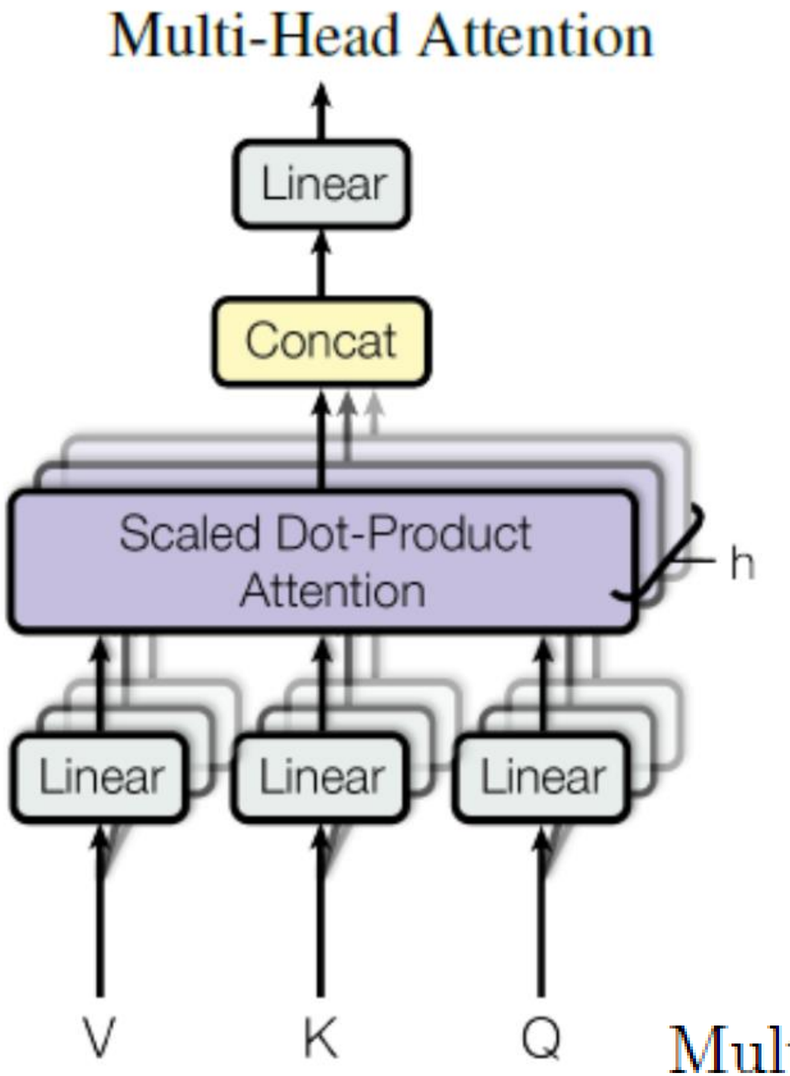
Scaled Dot-Product Attention



Multi-Head Attention

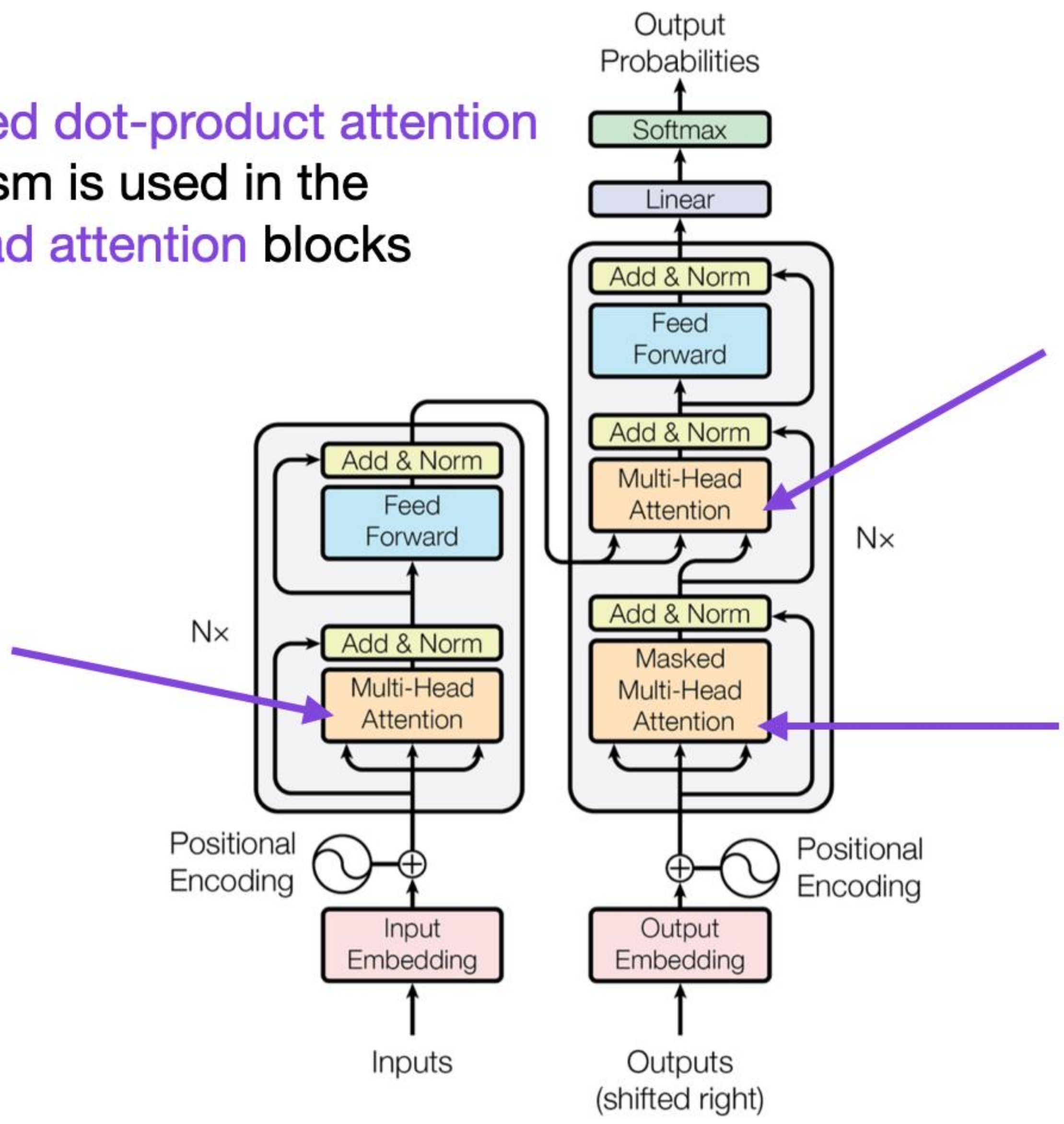


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



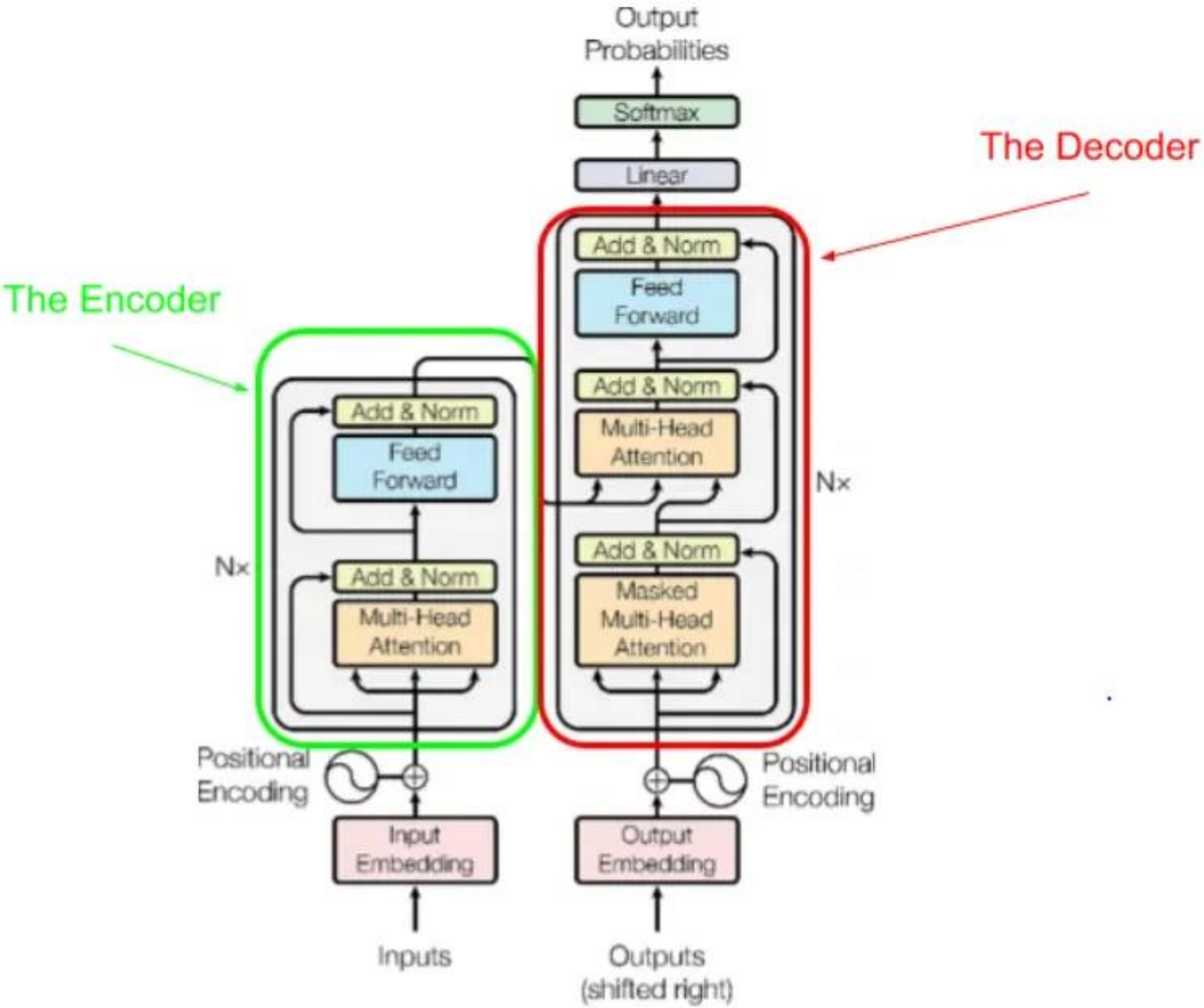
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

The scaled dot-product attention mechanism is used in the multi-head attention blocks



Source: “Attention Is All You Need” (https://arxiv.org/abs/1706.03762)

Transformer



Attention in Transformer

The Transformer uses multi-head attention in three different ways:

- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

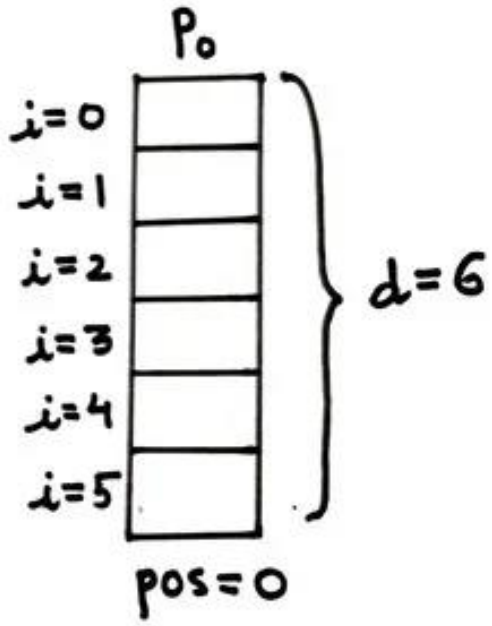
Positional Encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

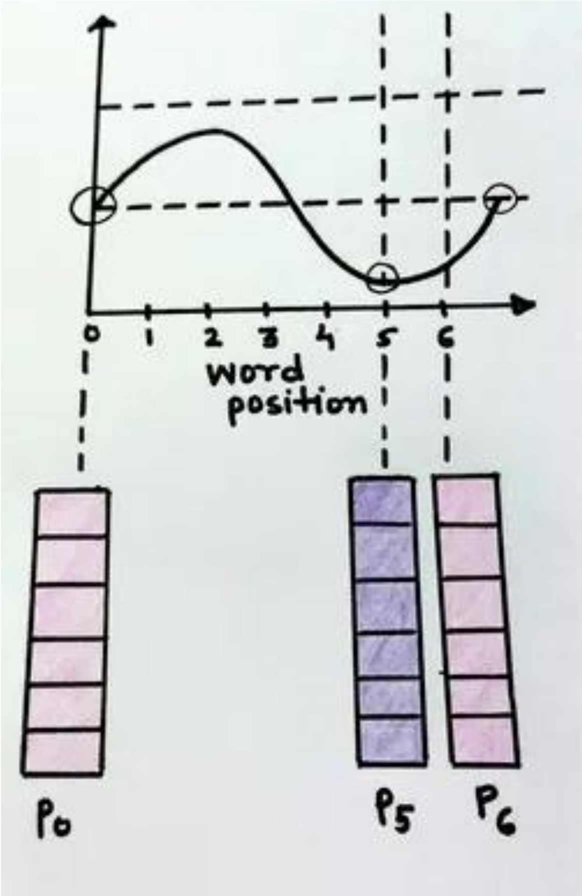
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

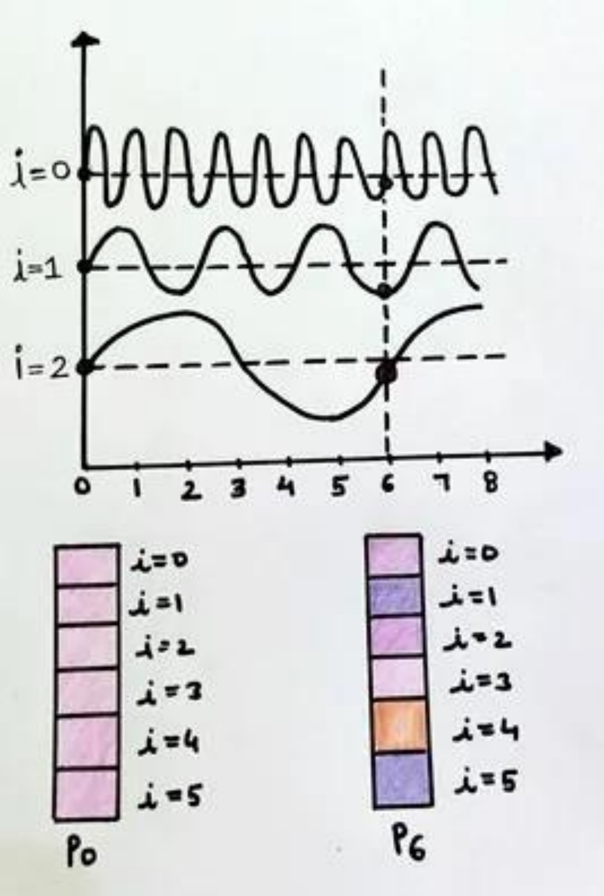
vary

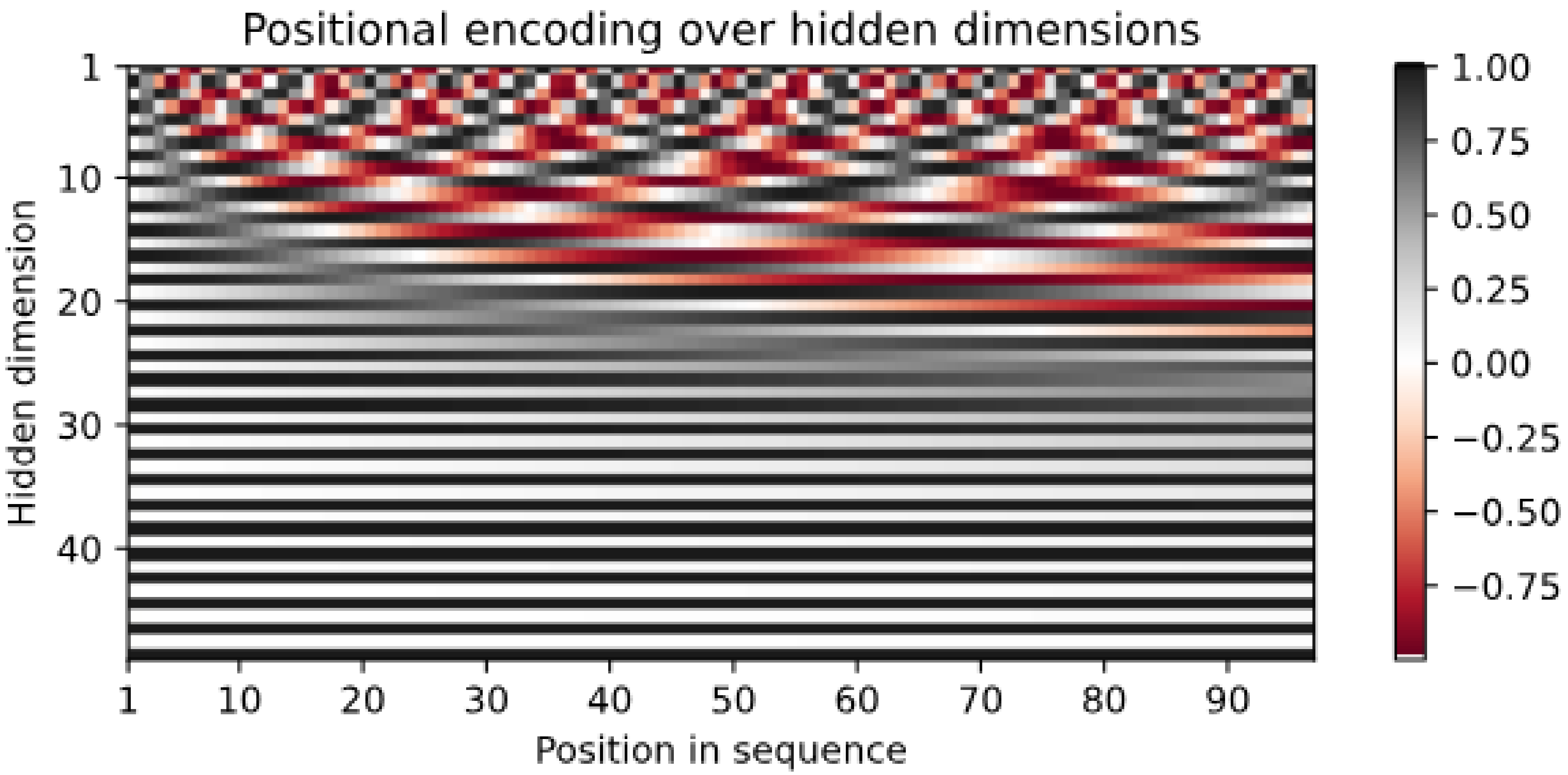
$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\underbrace{2i/d}_{\text{constant}}}}\right)$$



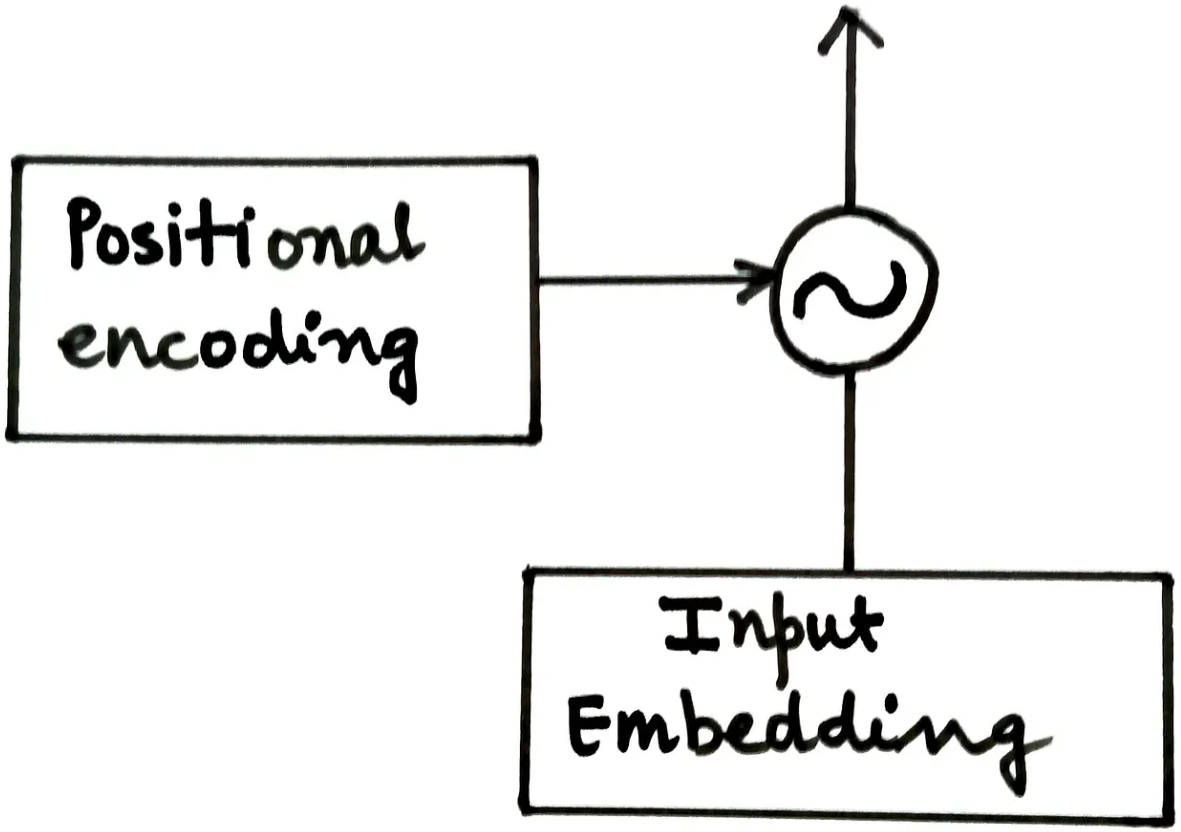
$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

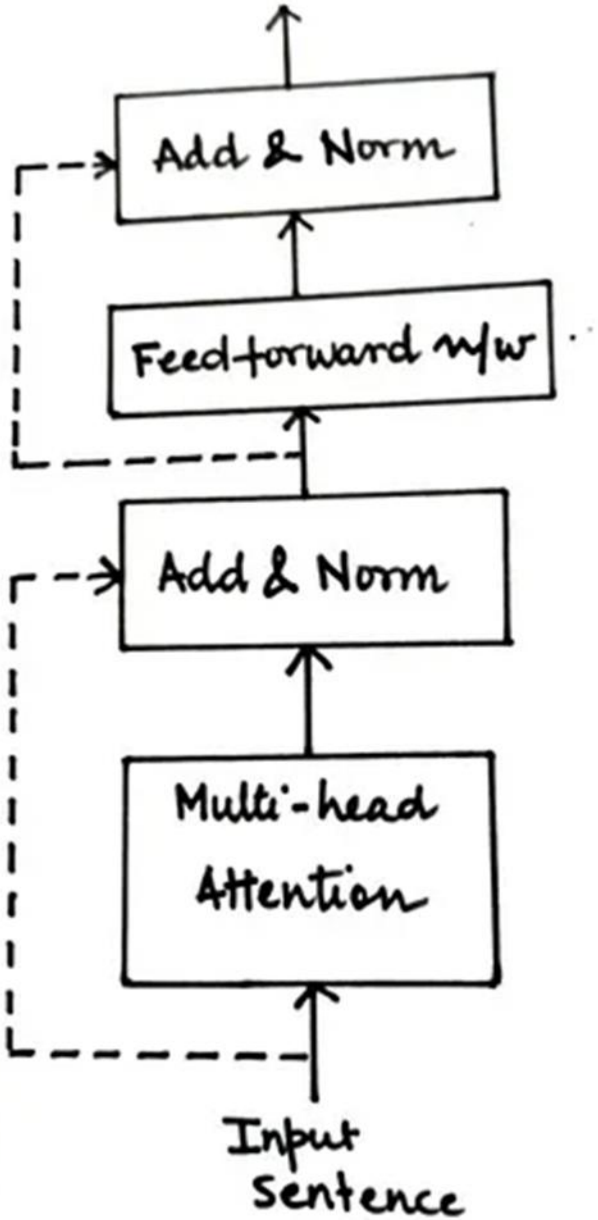




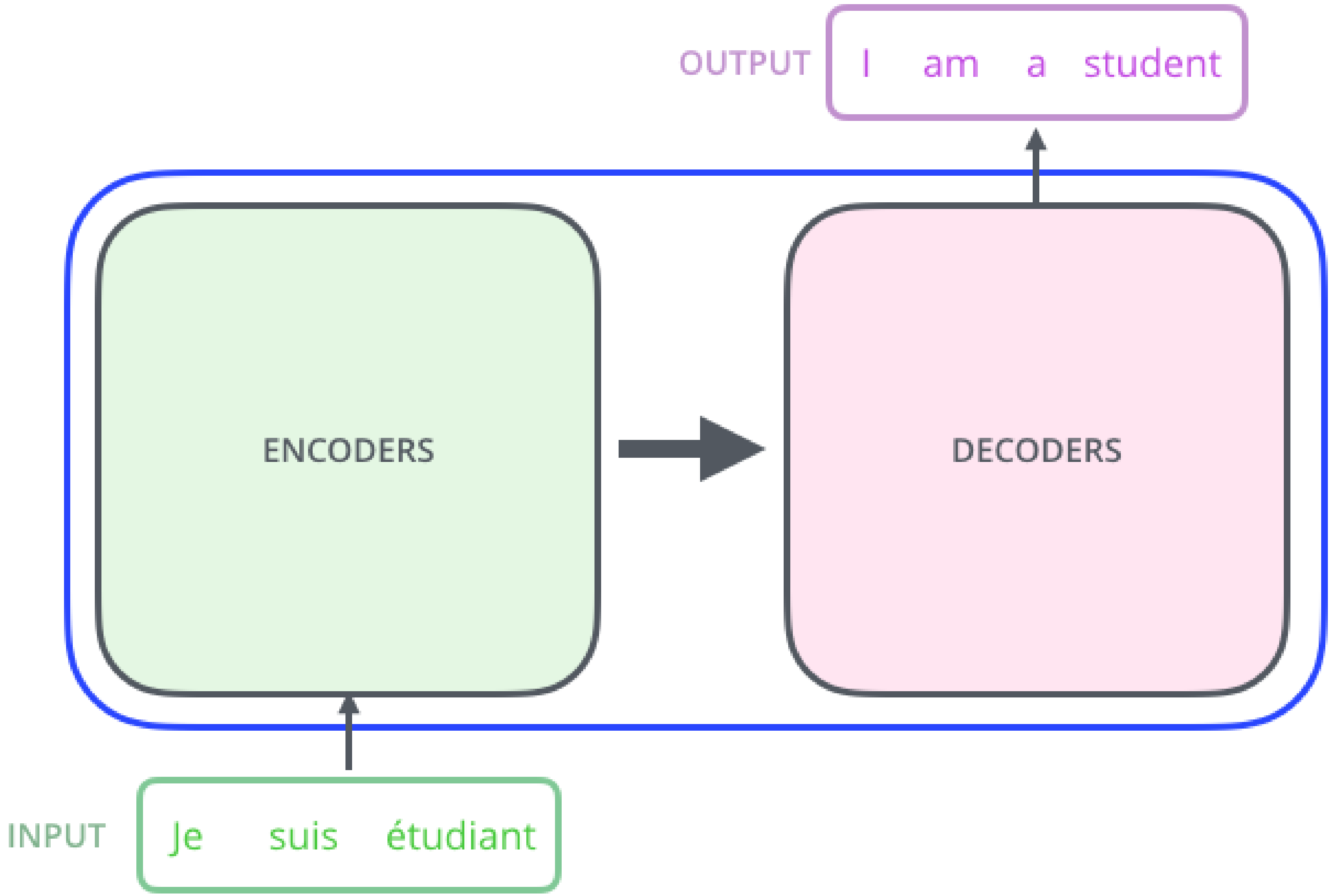


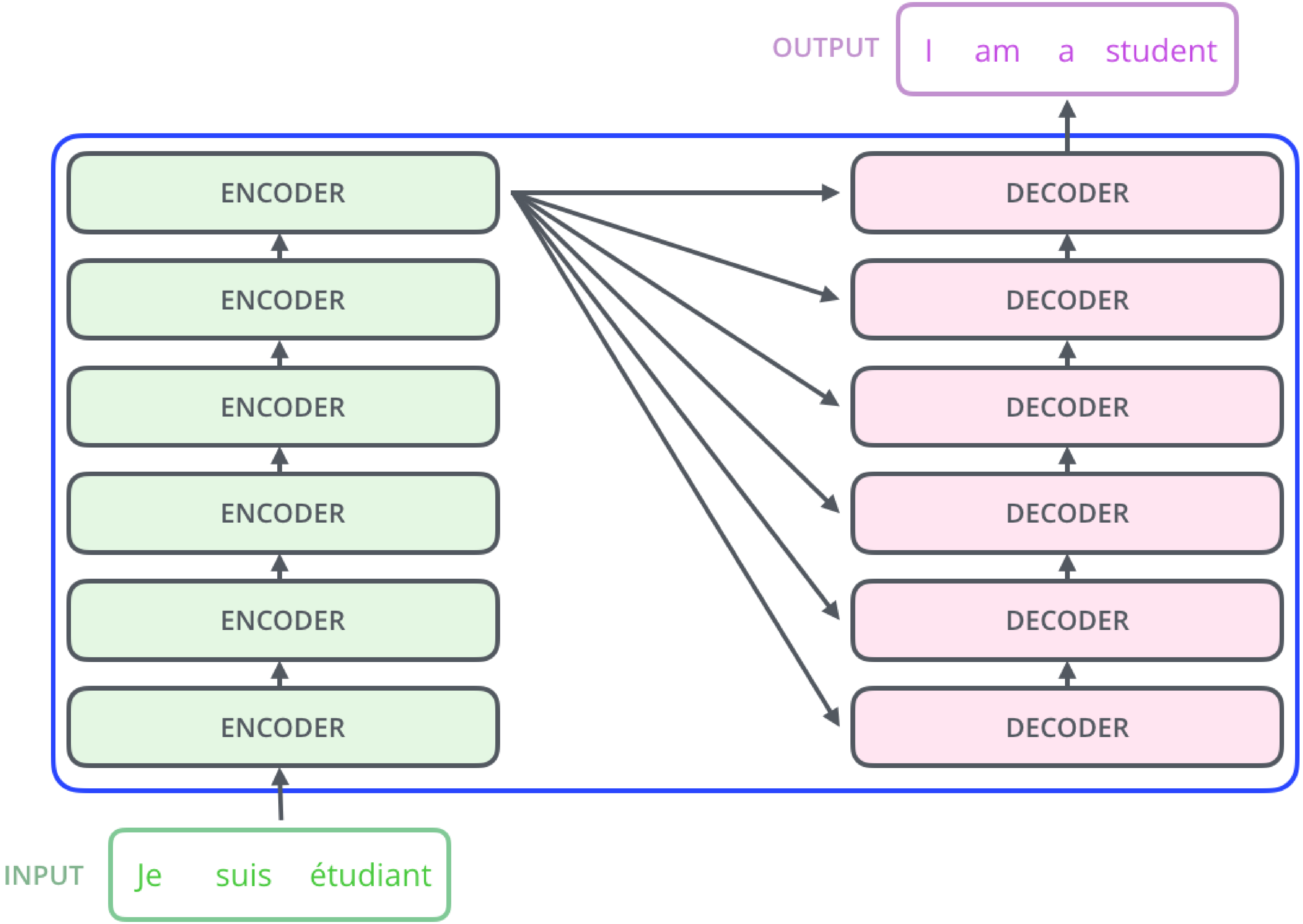
Positional encoding over hidden dimensions
Hidden dimension
1
10
20
30
40
Position in sequence
1
10
20
30
40
50
60
70
80
90
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

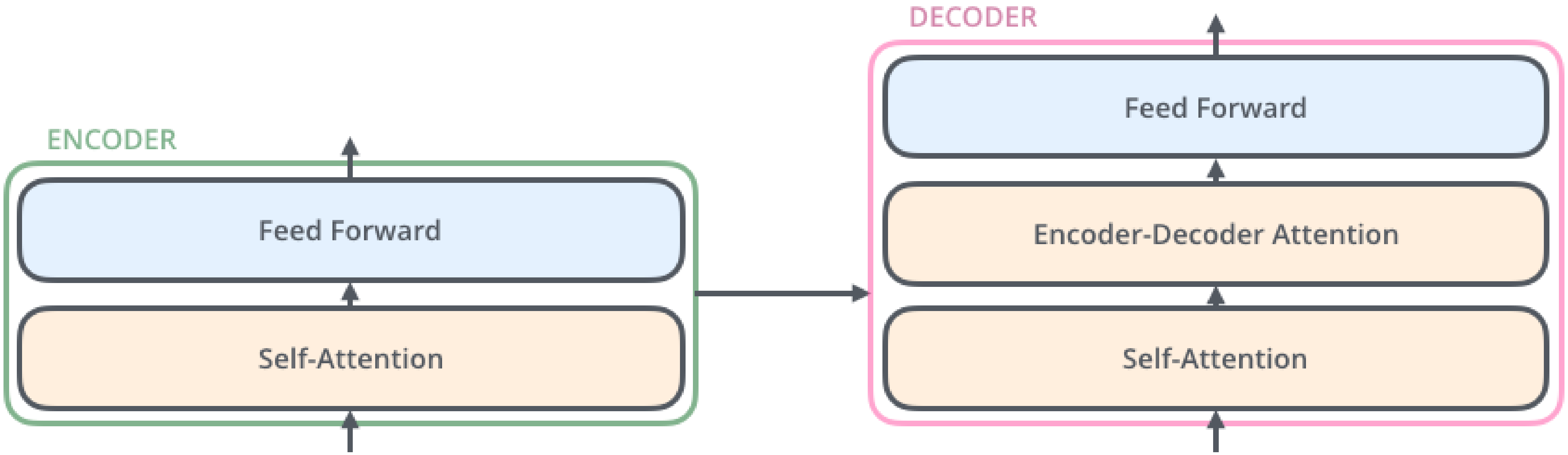


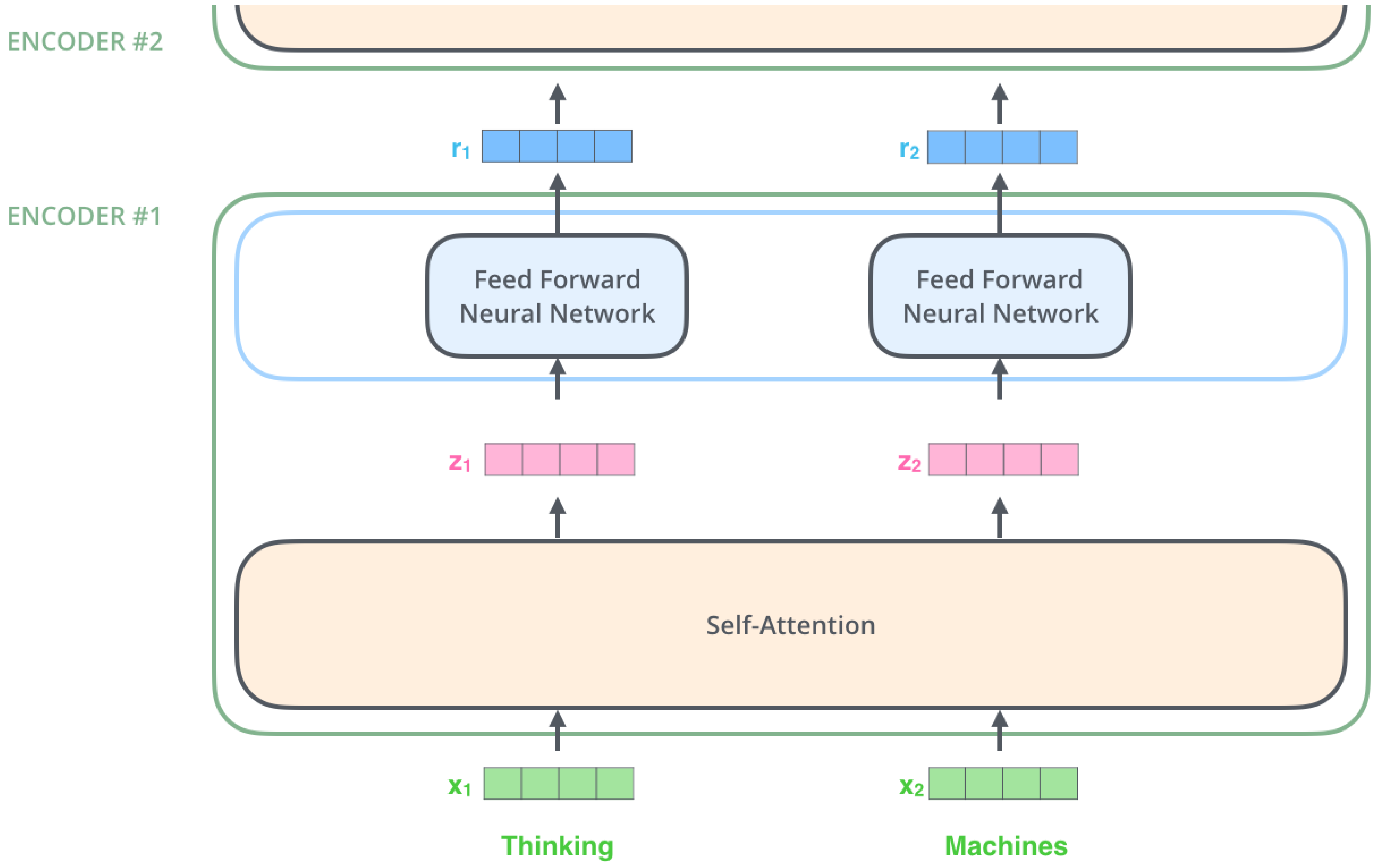


$$\mathrm{LayerNorm}(x + \mathrm{Multihead}(x, x, x))$$


OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant


OUTPUT
I am a student
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER
DECODER
DECODER
DECODER
INPUT
Je suis étudiant


ENCODER
Feed Forward
Self-Attention
DECODER
Feed Forward
Encoder-Decoder Attention
Self-Attention



ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

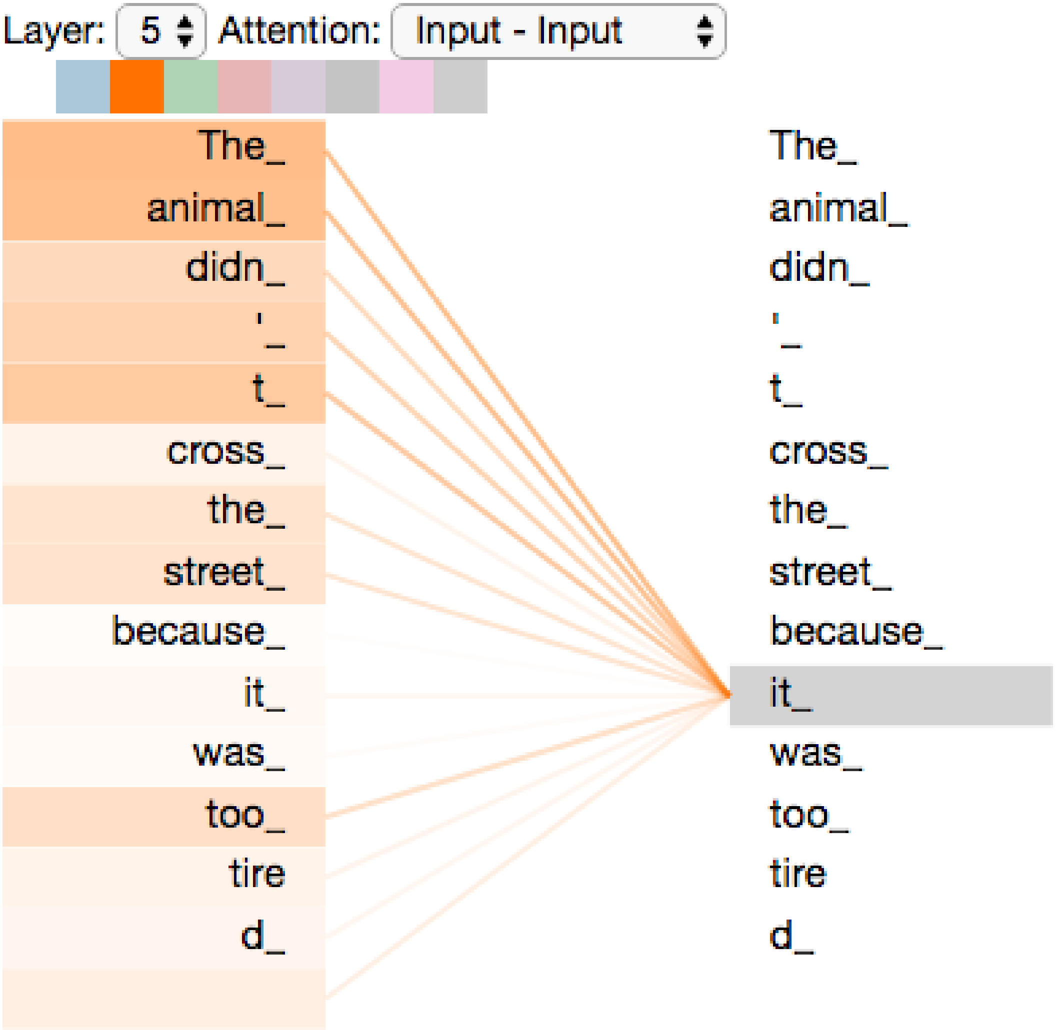
SELF-ATTENTION (SA)

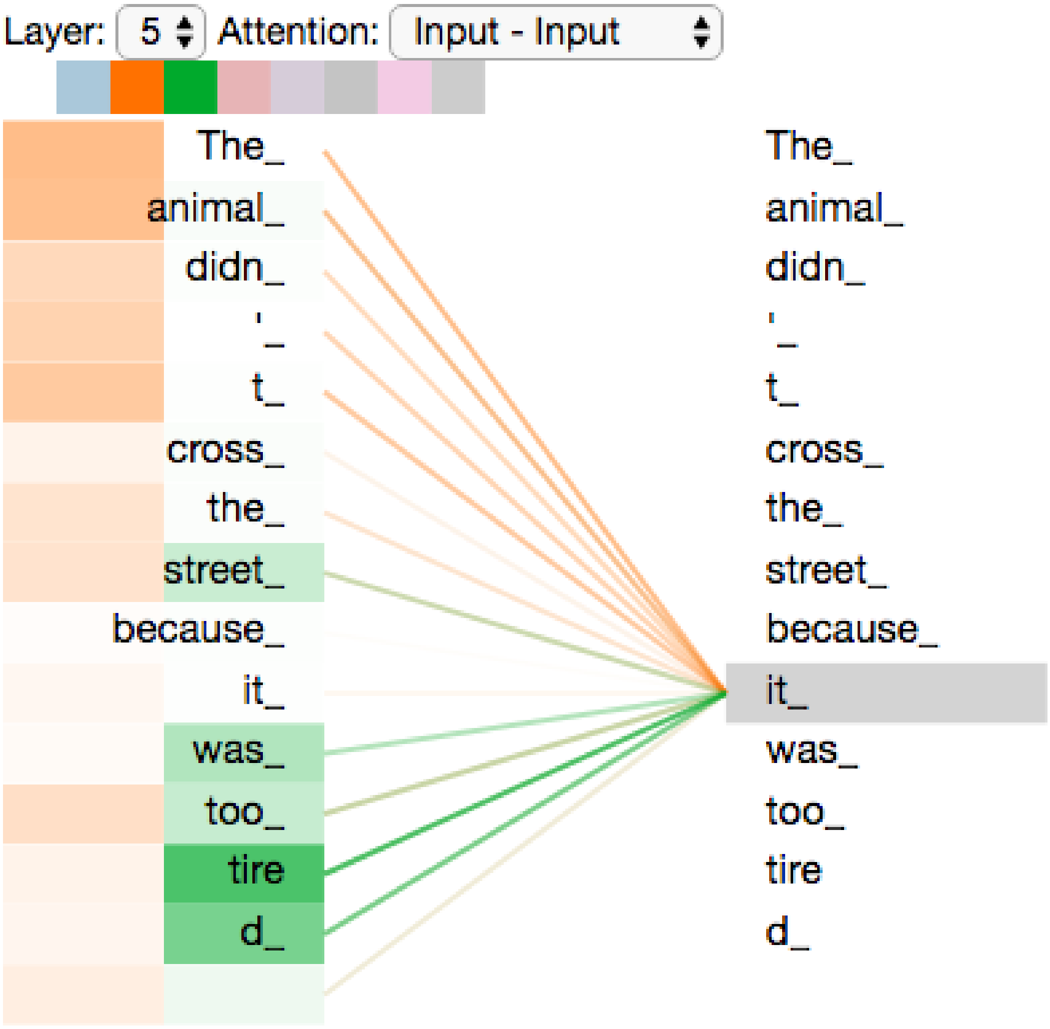
- Every token (in an i/p seq) attends to every other token.

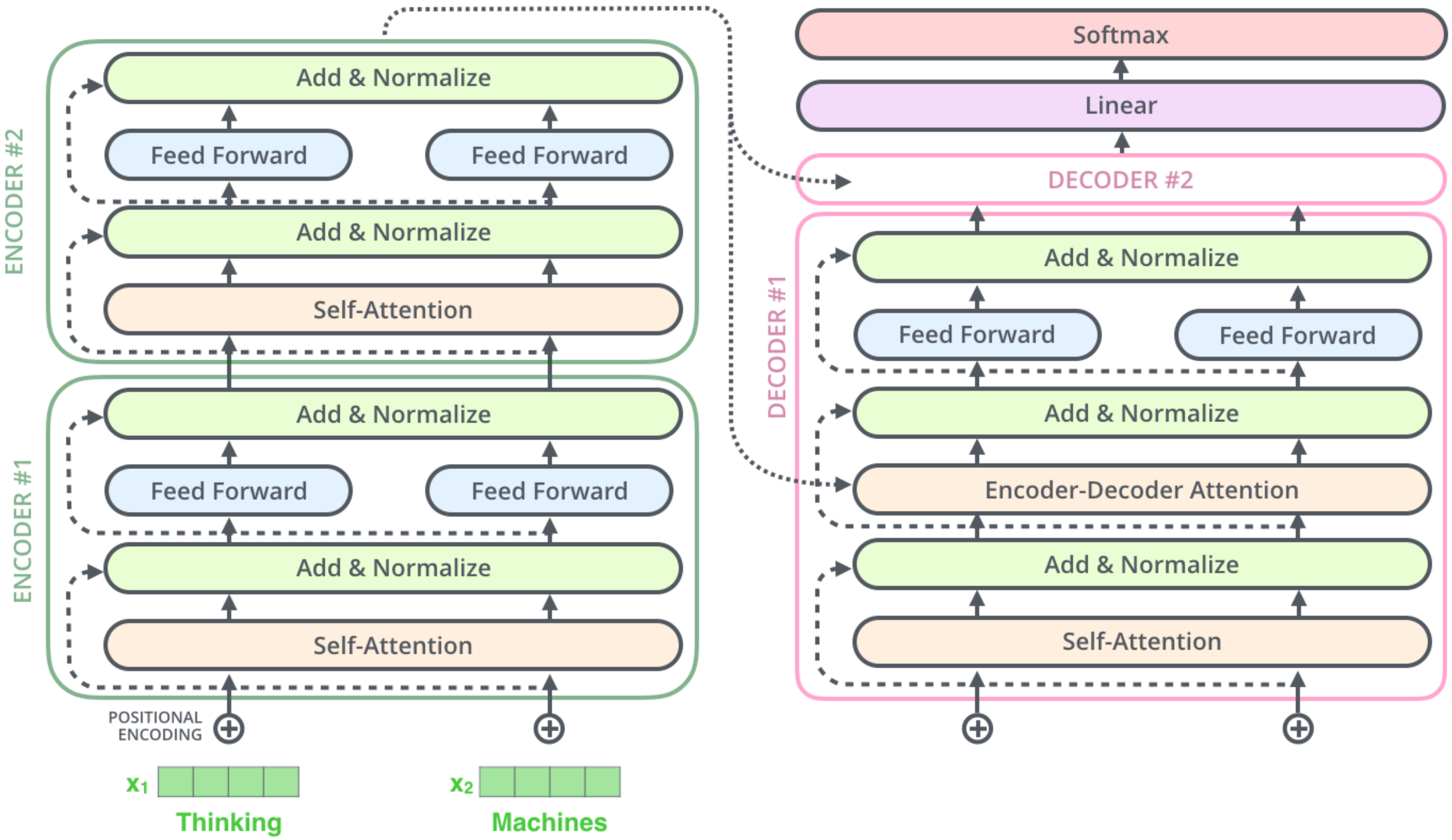
[Unlike where Decoder steps attend to Encoder steps] as in B.A.M.

Self-Attention

”The animal didn't cross the street because it was too tired”

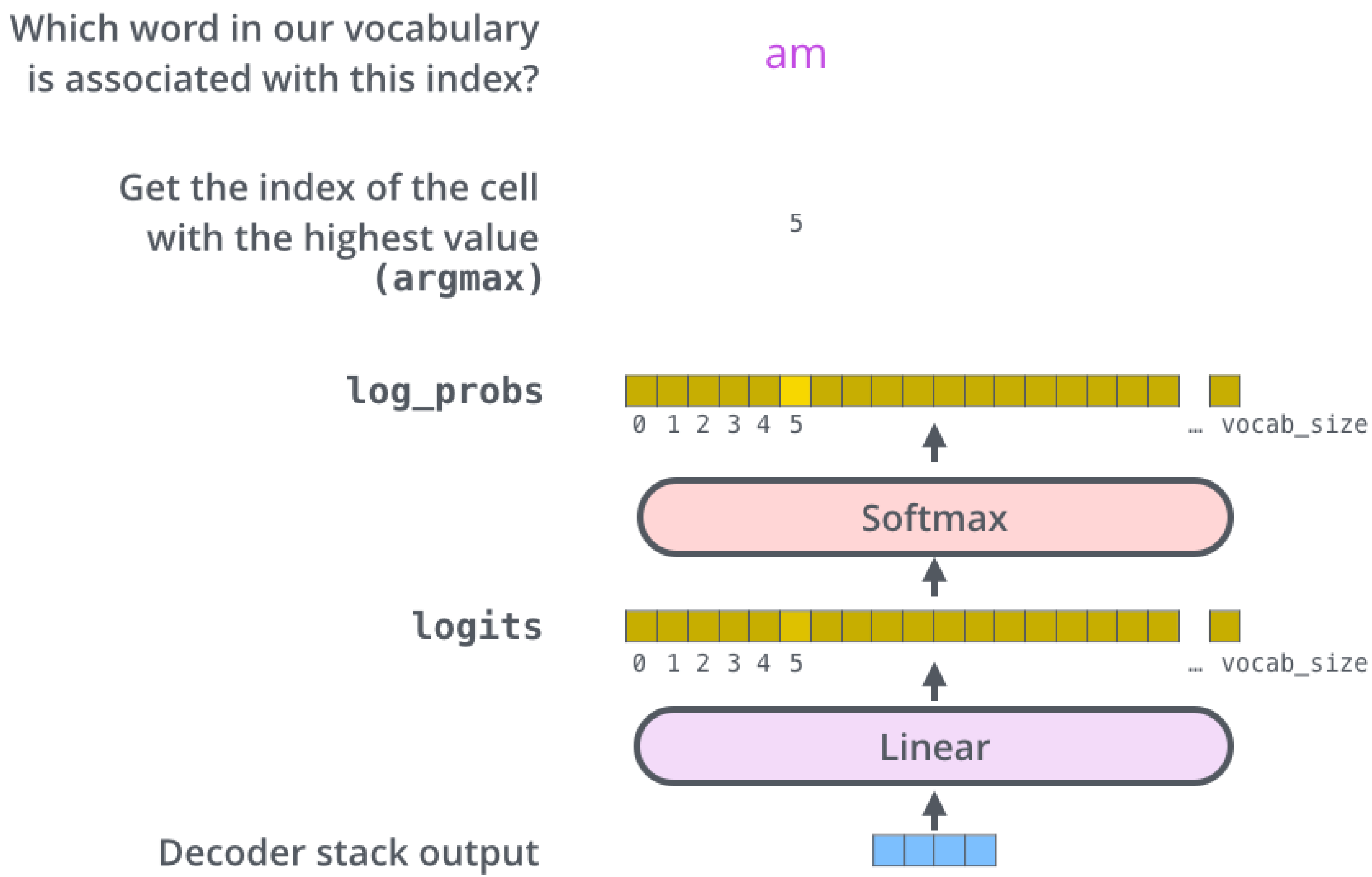






$$\mathrm{LayerNorm}(x + \mathrm{Multihead}(x, x, x))$$

The Final Linear and Softmax Layer



Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

Karthik Narasimhan
OpenAI
karthikn@openai.com

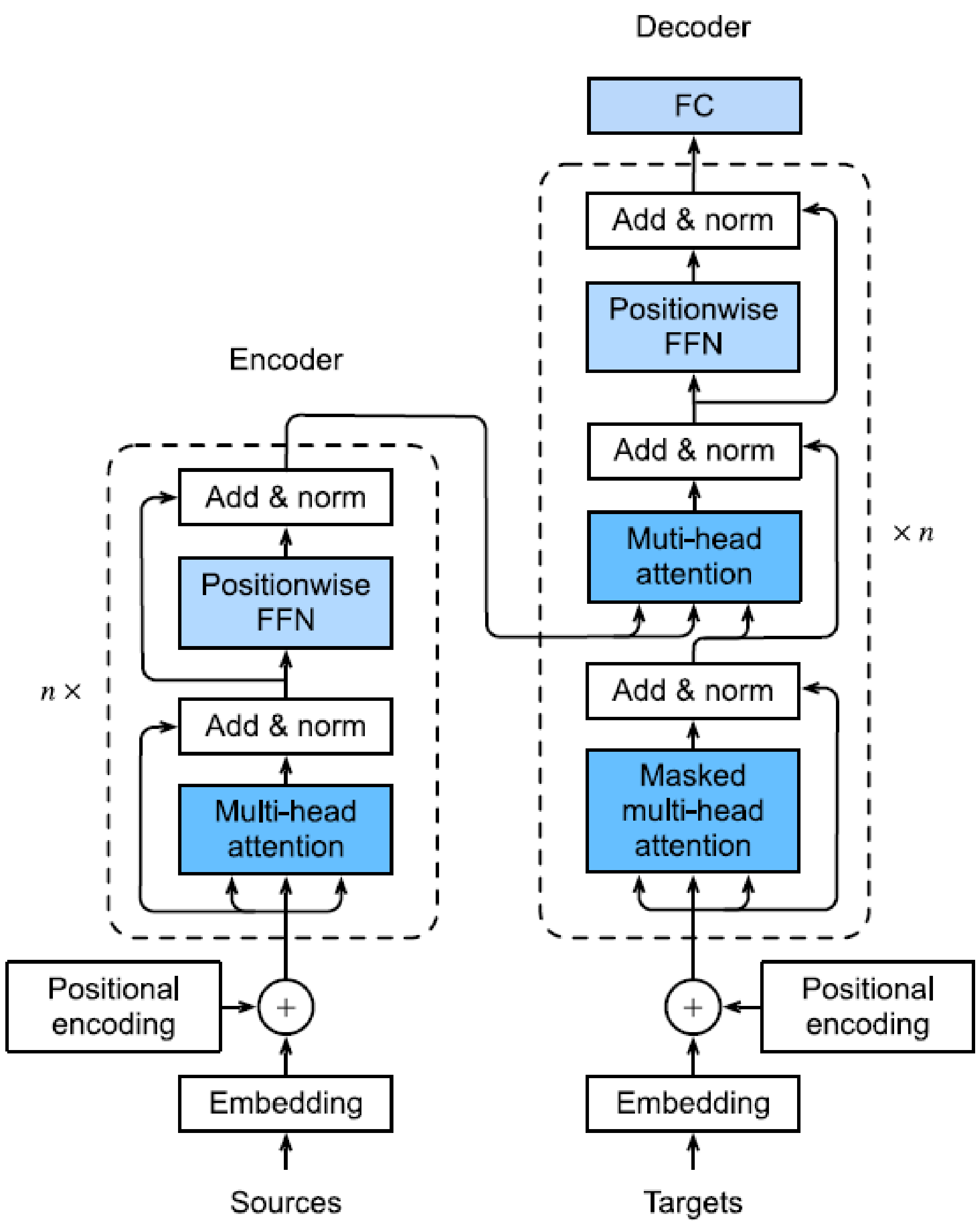
Tim Salimans
OpenAI
tim@openai.com

Ilya Sutskever
OpenAI
ilyasu@openai.com

GPT: Decoder-only Transformer

Decoder only Transformers remove the entire encoder and the decoder sublayer with the encoder–decoder cross-attention from the original encoder–decoder architecture

Nowadays, decoder-only Transformers have been the de facto architecture in large scale language modeling, which leverages the world’s abundant unlabeled text corpora via self-supervised learning.



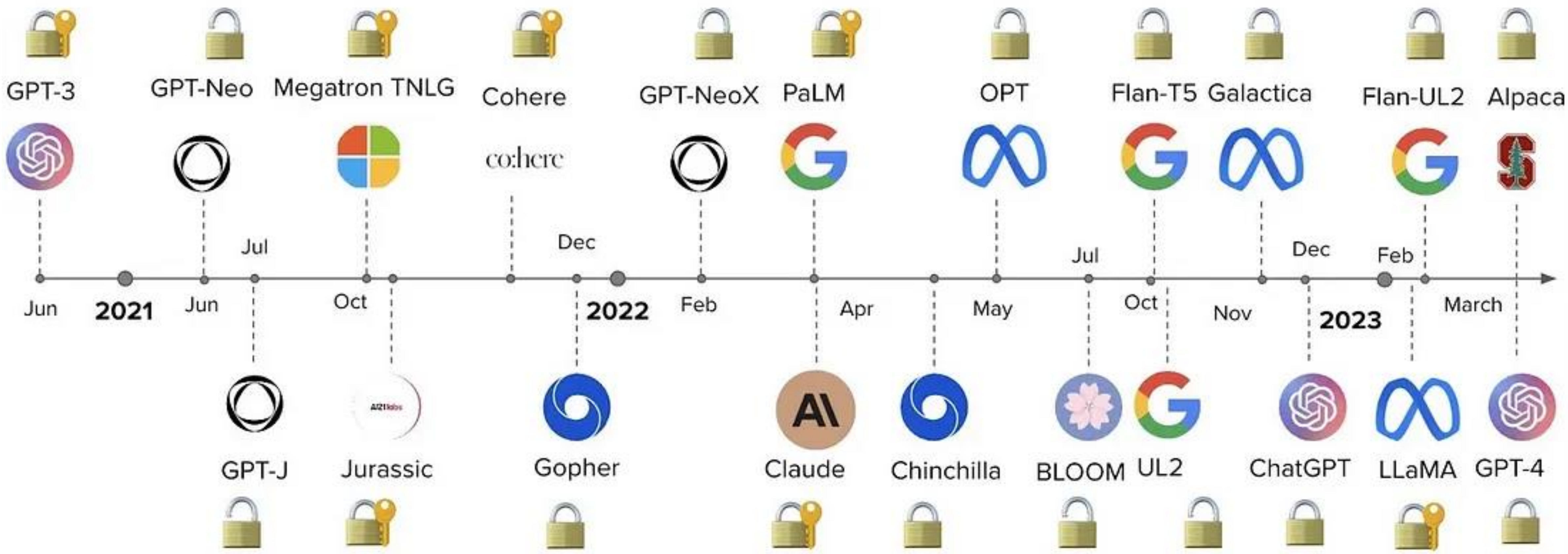
Generating Wikipedia by Summarizing Long Sequences

Peter J. Liu*, **Mohammad Saleh***,
Etienne Pot[†], **Ben Goodrich**, **Ryan Sepassi**, **Łukasz Kaiser**, **Noam Shazeer**
Google Brain
Mountain View, CA
{peterjliu,msaleh,epot,bgoodrich,rsepassi,lukaszkaiser,noam}@google.com

OpenAI's "GPT-n" series

Model	Architecture	Parameter count	Training data	Release date	Training cost
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	117 million	BookCorpus:[27] 4.5 GB of text, from 7000 unpublished books of various genres.	June 11, 2018[8]	30 days on 8 P600 GPUs, or 1 petaFLOP/s-day.[8]
GPT-2	GPT-1, but with modified normalization	1.5 billion	WebText: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	February 14, 2019 (initial/limited version) and November 5, 2019 (full version)[28]	"tens of petaflop/s-day",[29] or 1.5e21 FLOP.[30]
GPT-3	GPT-2, but with modification to allow larger scaling	175 billion[31]	499 billion tokens consisting of CommonCrawl (570 GB), WebText, English Wikipedia, and two books corpora (Books1 and Books2).	May 28, 2020[29]	3640 petaflop/s-day (Table D.1 [29]), or 3.1e23 FLOP.[30]
GPT-3.5	Undisclosed	175 billion[31]	Undisclosed	March 15, 2022	Undisclosed
GPT-4	Also trained with both text prediction and RLHF; accepts both text and images as input. Further details are not public.[26]	Undisclosed. Estimated 1.7 trillion[32]	Undisclosed	March 14, 2023	Undisclosed. Estimated 2.1e25 FLOP.[30]

Model	Organization	Date	Size (# params)
ELMo	AI2	Feb 2018	94,000,000
GPT	OpenAI	Jun 2018	110,000,000
BERT	Google	Oct 2018	340,000,000
XLM	Facebook	Jan 2019	655,000,000
GPT-2	OpenAI	Mar 2019	1,500,000,000
RoBERTa	Facebook	Jul 2019	355,000,000
Megatron-LM	NVIDIA	Sep 2019	8,300,000,000
T5	Google	Oct 2019	11,000,000,000
Turing-NLG	Microsoft	Feb 2020	17,000,000,000
GPT-3	OpenAI	May 2020	175,000,000,000
Megatron-Turing NLG	Microsoft, NVIDIA	Oct 2021	530,000,000,000
Gopher	DeepMind	Dec 2021	280,000,000,000


GPT-3
GPT-Neo
Megatron TNLG
Cohere
co:here
GPT-NeoX
PaLM
OPT
Flan-T5
Galactica
Flan-UL2
Alpaca
Jun
2021
Jun
Jul
Oct
Dec
2022
Feb
Apr
May
Jul
Oct
Nov
Dec
2023
Feb
March
GPT-J
Jurassic
Gopher
Claude
Chinchilla
BLOOM
UL2
ChatGPT
LLaMA
GPT-4

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

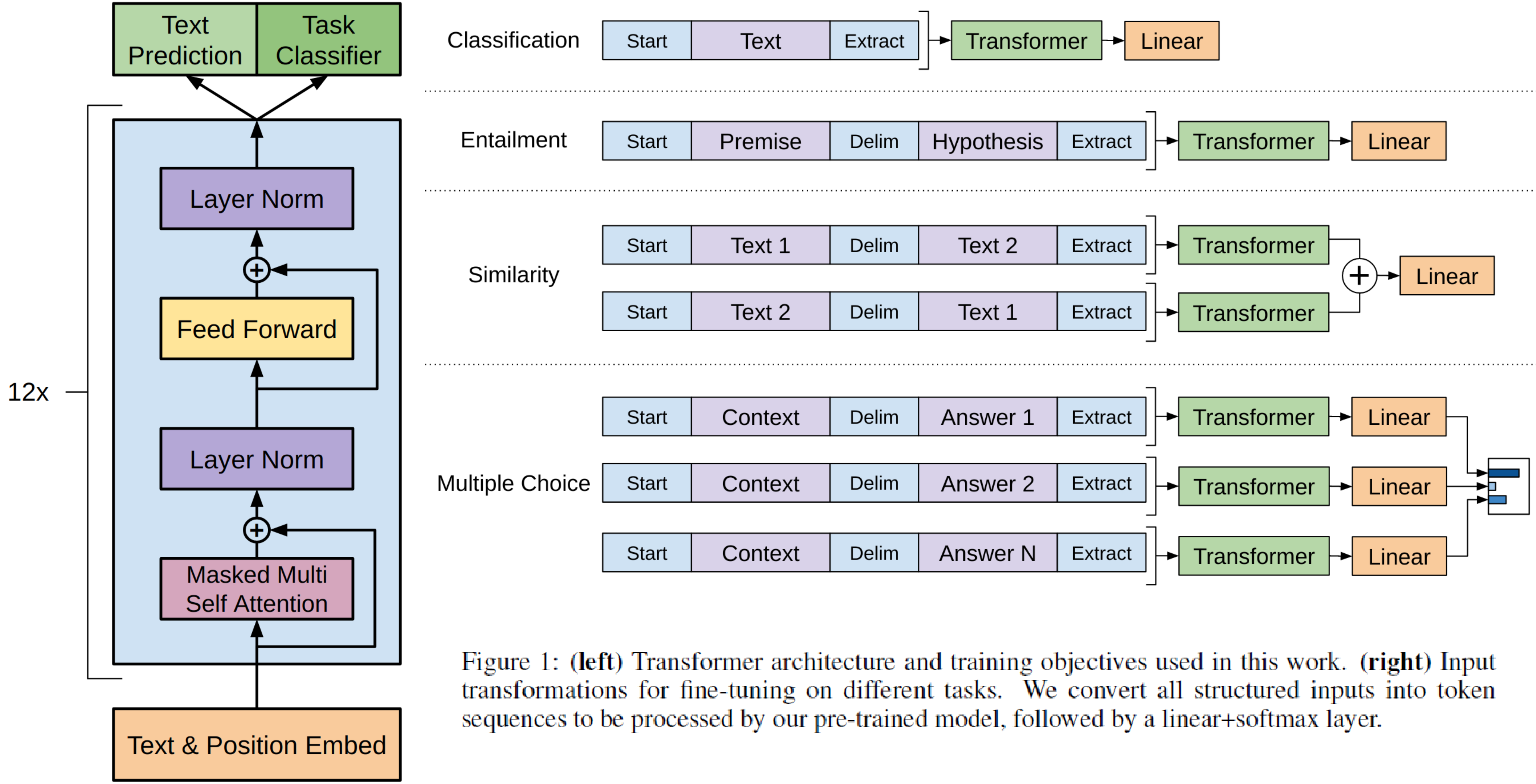
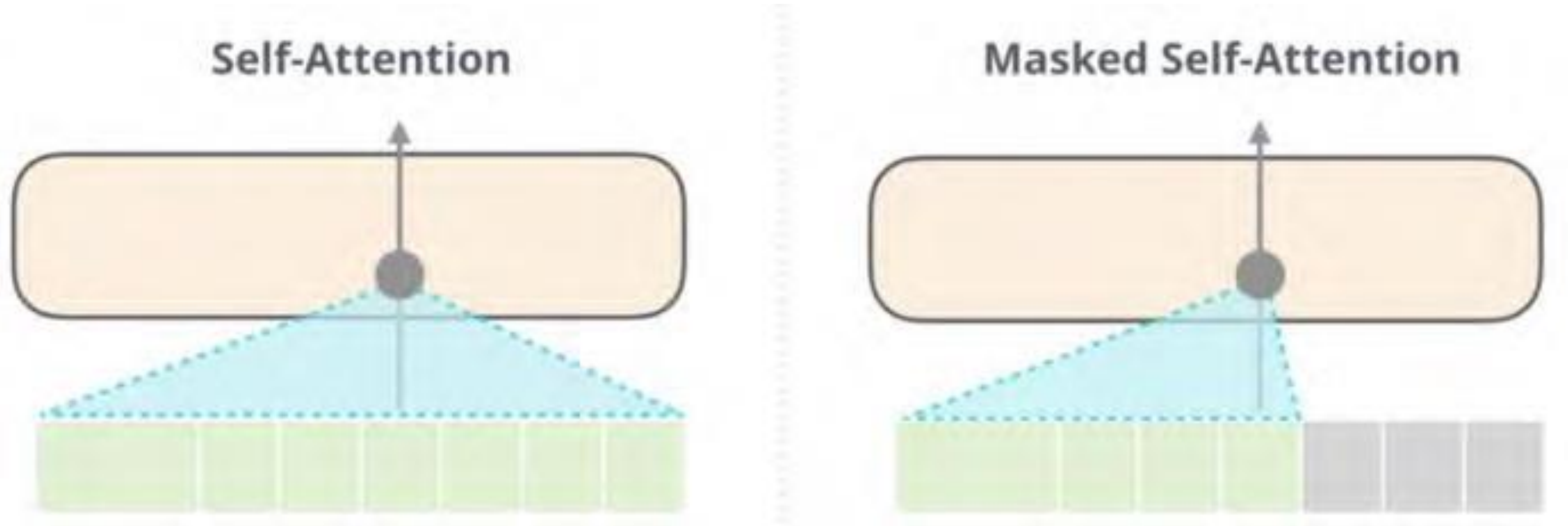
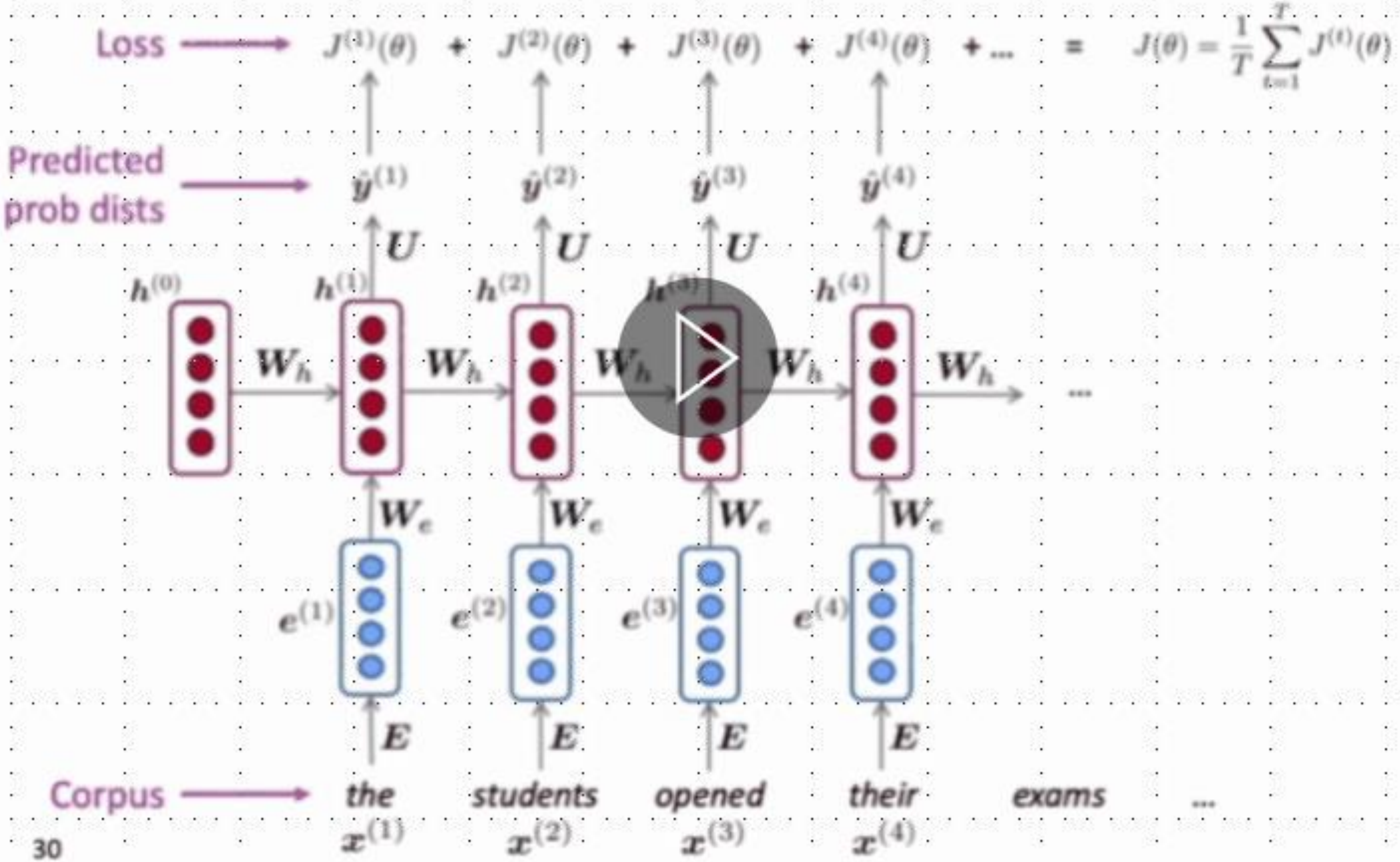


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

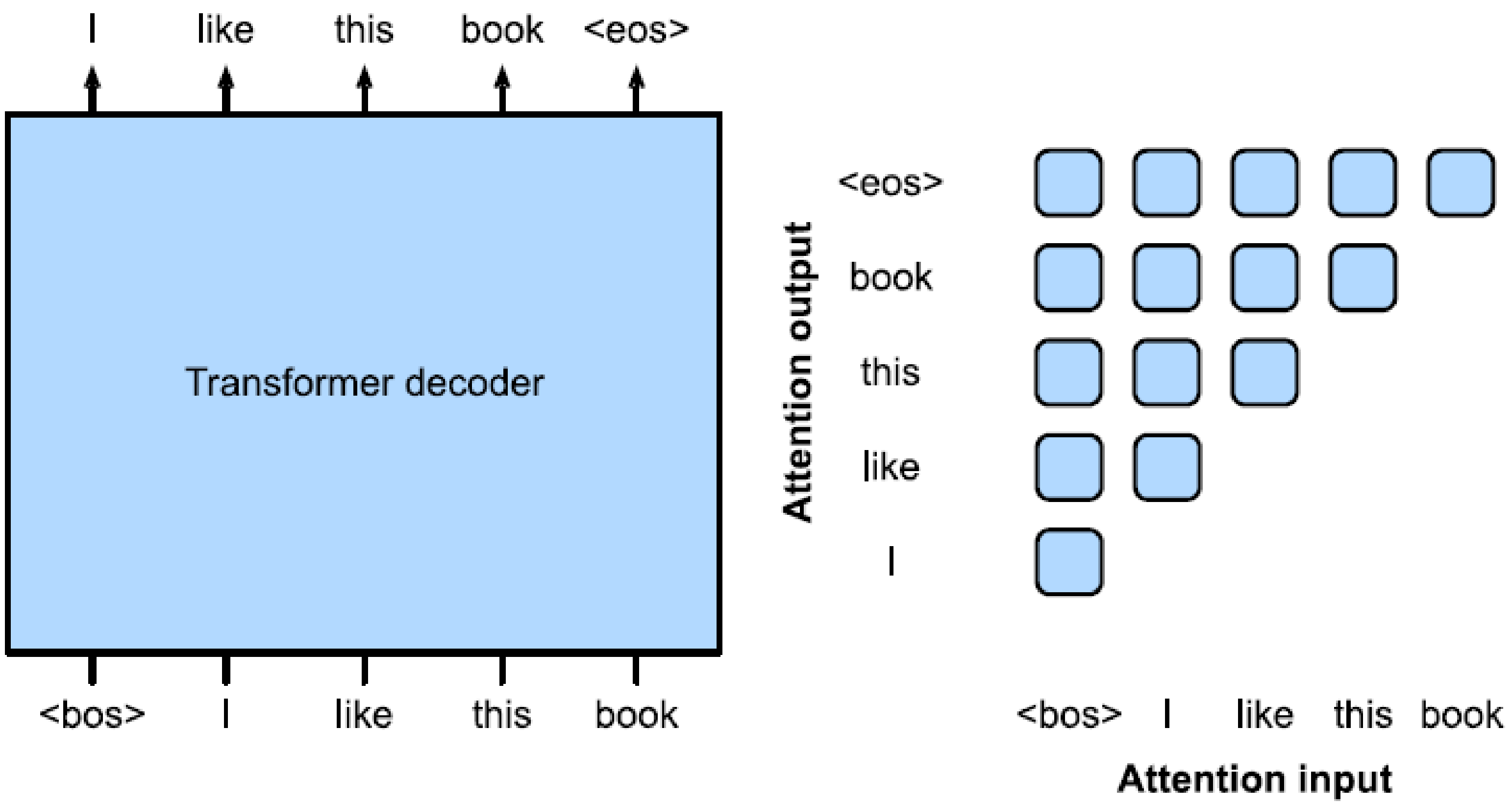
Dataset	Task	SOTA	Ours
SNLI	Textual entailment	89.3	89.9
MNLI matched	Textual entailment	80.6	82.1
MNLI mismatched	Textual entailment	80.1	81.4
SciTail	Textual entailment	83.3	88.3
QNLI	Textual entailment	82.3	88.1
RTE	Textual entailment	61.7	56.0
STS-B	Semantic similarity	81.0	82.0
QQP	Semantic similarity	66.1	70.3
MRPC	Semantic similarity	86.0	82.3
RACE	Reading comprehension	53.3	59.0
ROCStories	Commonsense reasoning	77.6	86.5
COPA	Commonsense reasoning	71.2	78.6
SST-2	Sentiment analysis	93.2	91.3
CoLA	Linguistic acceptability	35.0	45.4
GLUE	Multi task benchmark	68.9	72.8


Self-Attention
Masked Self-Attention

Training a RNN Language Model



GPT pretraining



Left: Pretraining GPT with language modeling. The target sequence is the input sequence shifted by one token. Both “<bos>” and “<eos>” are special tokens marking the beginning and end of sequences, respectively. Right: Attention pattern in the Transformer decoder. Each token along the vertical axis attends to only its past tokens along the horizontal axis (causal).

Unsupervised pre-training

Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$, we use a standard language modeling objective to maximize the following likelihood:

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

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent [51].

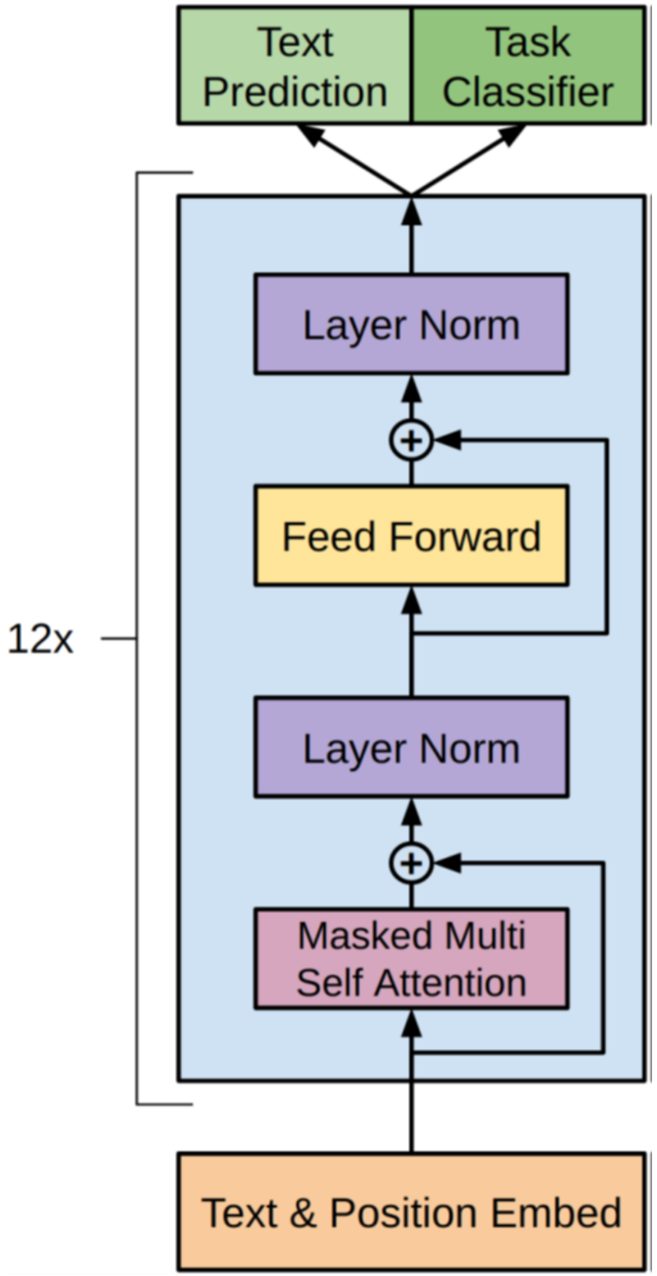
In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

In our experiments, we use a multi-layer *Transformer decoder* [34] for the language model, which is a variant of the transformer [62]. This model applies a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens:

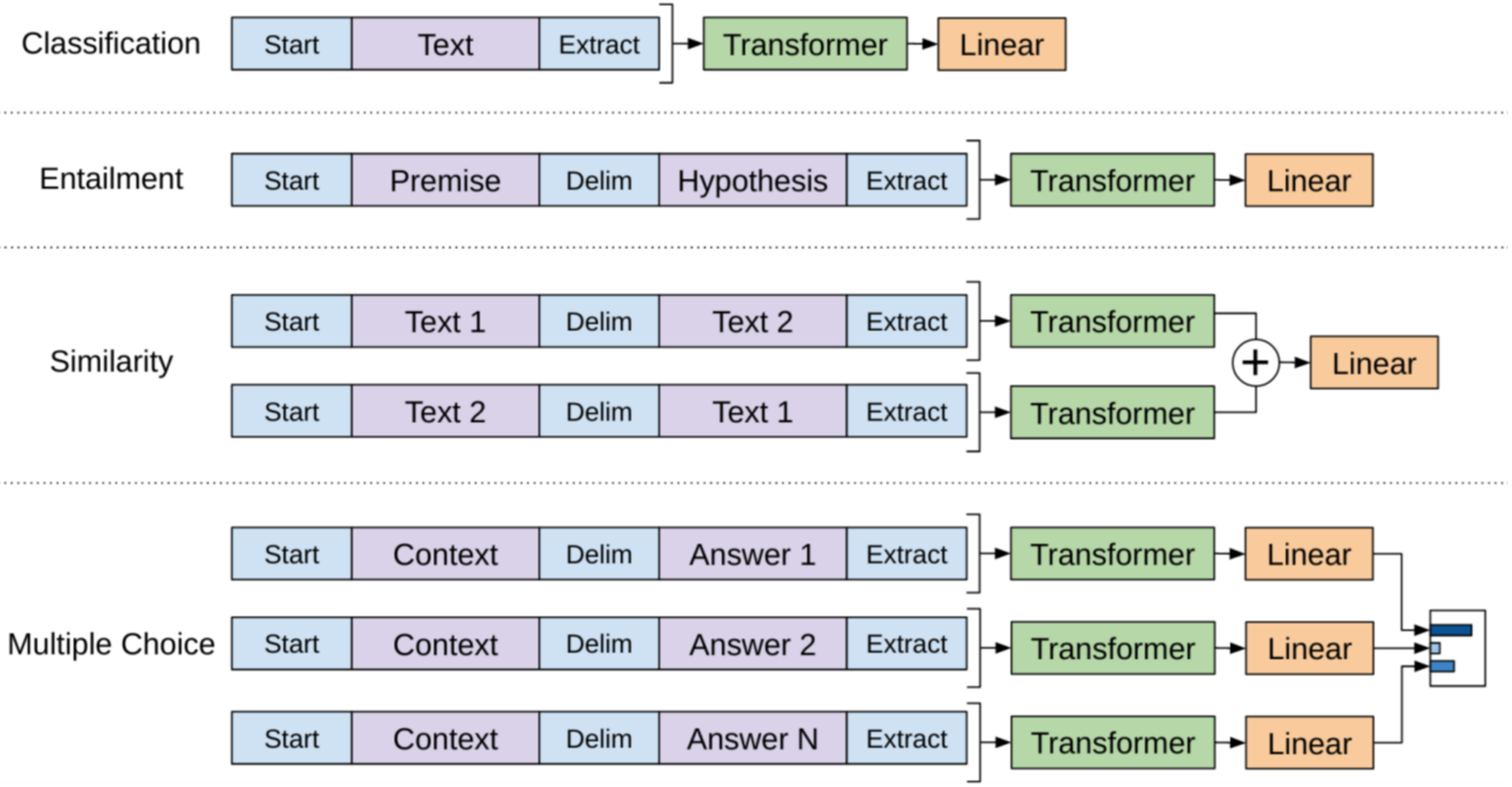
$$
\begin{aligned}
h_0 &= UW_e + W_p \\
h_l &= \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n] \\
P(u) &= \texttt{softmax}(h_n W_e^T)
\end{aligned}
$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

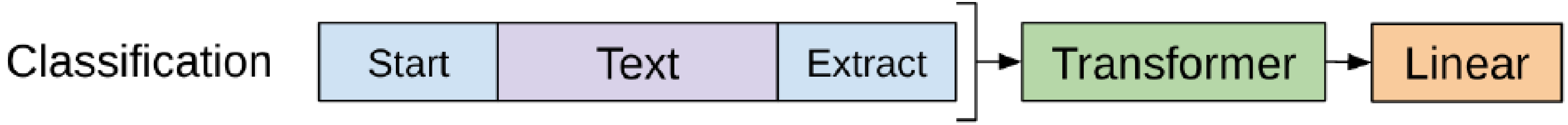
$$h_0 = UW_e + W_p$$
$$h_l = \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n]$$
$$P(u) = \texttt{softmax}(h_n W_e^T)$$



Supervised fine-tuning



Supervised fine-tuning



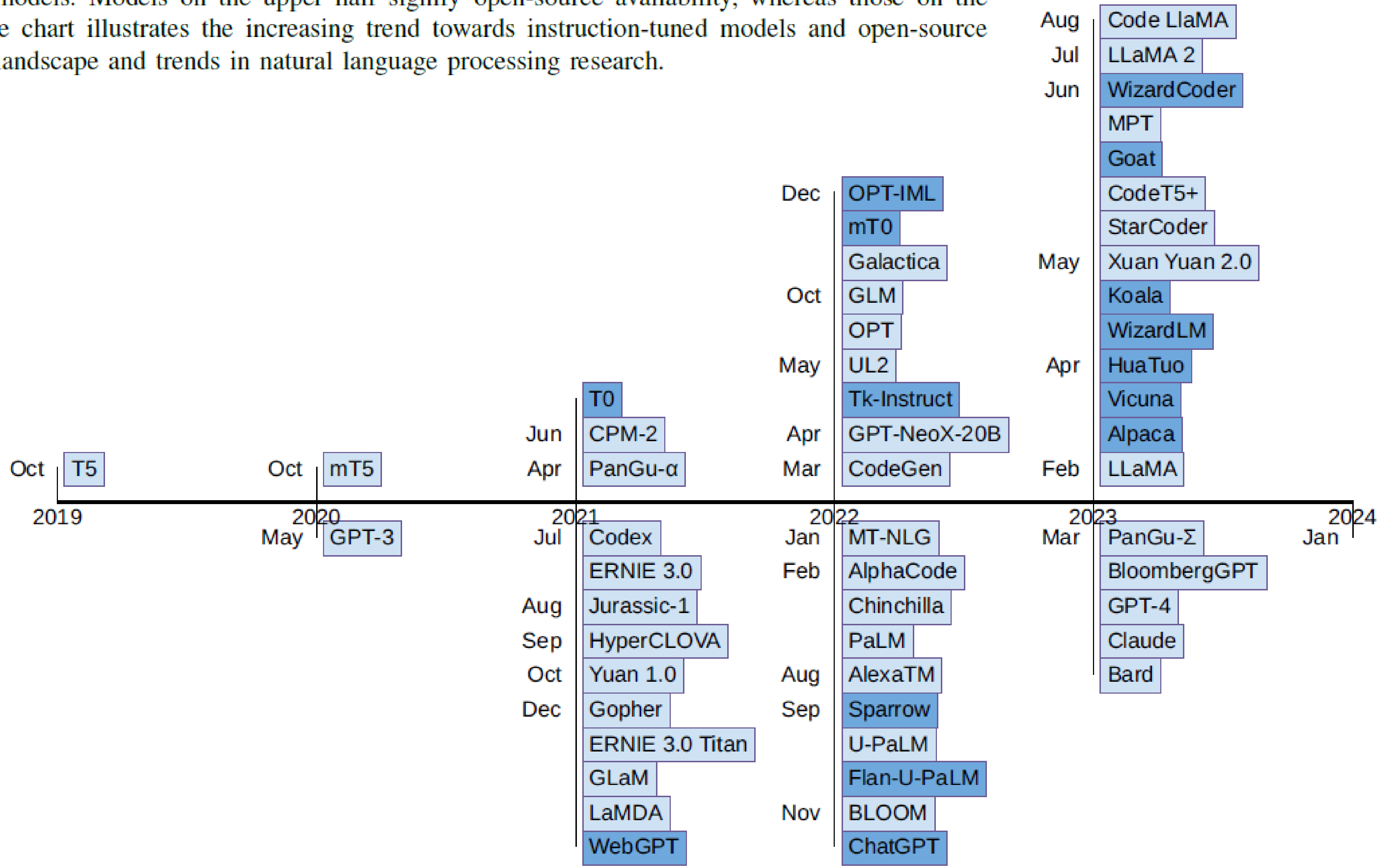
After training the model with the objective in Eq. 1, we adapt the parameters to the supervised target task. We assume a labeled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens, $x^1, \ldots, x^m$, along with a label y. The inputs are passed through our pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y:

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y). \tag{3}$$

This gives us the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m). \tag{4}$$

Fig. 2: Chronological display of LLM releases: light blue rectangles represent 'pre-trained' models, while dark rectangles correspond to 'instruction-tuned' models. Models on the upper half signify open-source availability, whereas those on the bottom half are closed-source. The chart illustrates the increasing trend towards instruction-tuned models and open-source models, highlighting the evolving landscape and trends in natural language processing research.



Thank you !!