
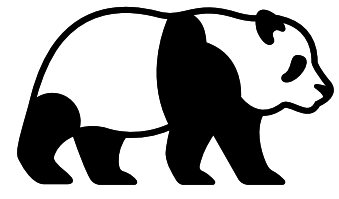


# Natural Language Processing with Deep Learning

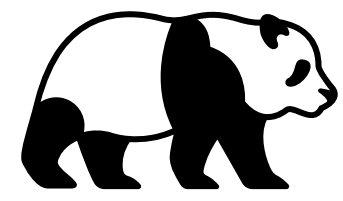
 IFT6289, Winter 2022

Lecture 7: Transformer and BERT  
Bang Liu



# Recap: Mini-Lectures

- The mini-lectures will take place at **Feb 18th, Feb 22th and Feb 25th**, with each class presenting 6~7 papers. After all groups decide their paper selection, we will release the **presentation order** on this Thursday.
- The mini-lecture presentation accounts for 3 reading assignments, which means it is **worth 3 points** of your final grade. Our evaluation will focus on the **contents** and **clarity** of your slides, and the quality of your **presentation delivery**.
- Each presentation should be no more than **17 minutes (~15min presentation + ~2min Q&A)**. **Please don't exceed this time limit**, or it would occupy other groups' time and also **may affect your presentation score**.
- For each group, no matter when your presentation takes place, please **submit your slides before 11:59am, Feb 18th (Fri.)** in **#mini-lectures**. Please indicate your group number in your file name (e.g., Group1\_What Does BERT Look At.pdf). **Late submission of the slides results in a -1 point penalty**.



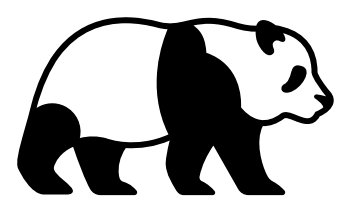
# Lecture outline

1. Transformer
2. BERT and a Few Variants

# Transformer



# **Recap Attention Mechanism**

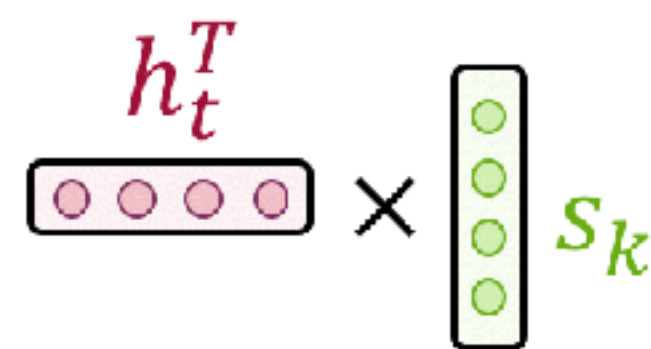


# Recap Attentions

The most popular ways to compute attention scores are:

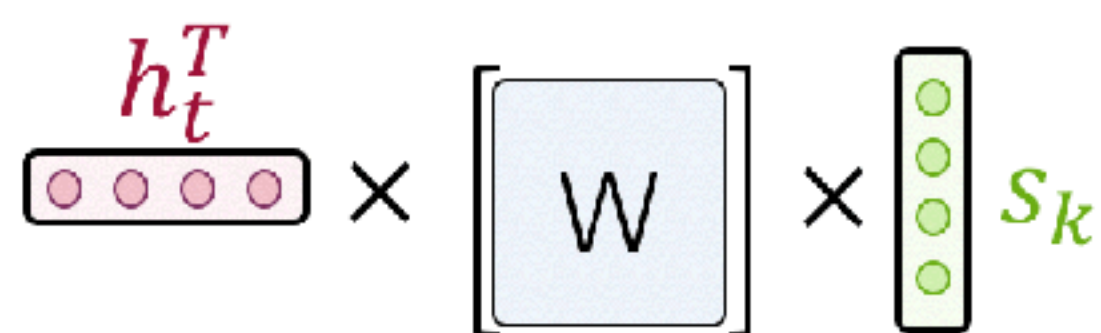
- dot-product - the simplest method;
- bilinear function (aka "Luong attention") - used in the paper [Effective Approaches to Attention-based Neural Machine Translation](#);
- multi-layer perceptron (aka "Bahdanau attention") - the method proposed in the [original paper](#).

Dot-product



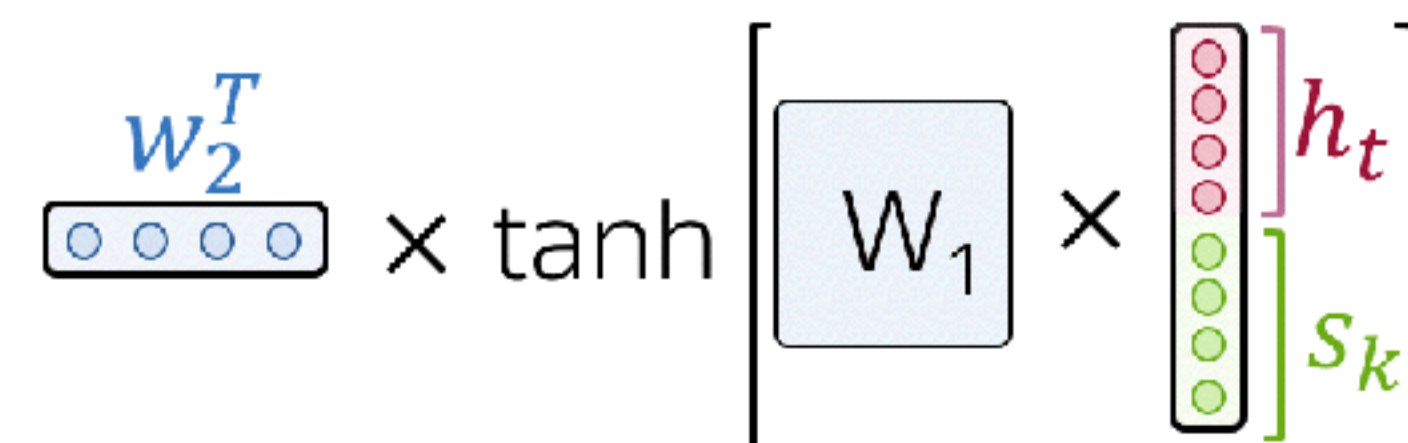
$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear

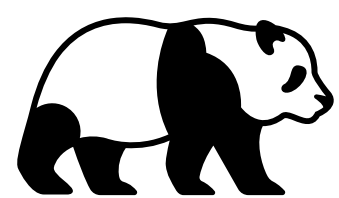


$$\text{score}(h_t, s_k) = h_t^T W s_k$$

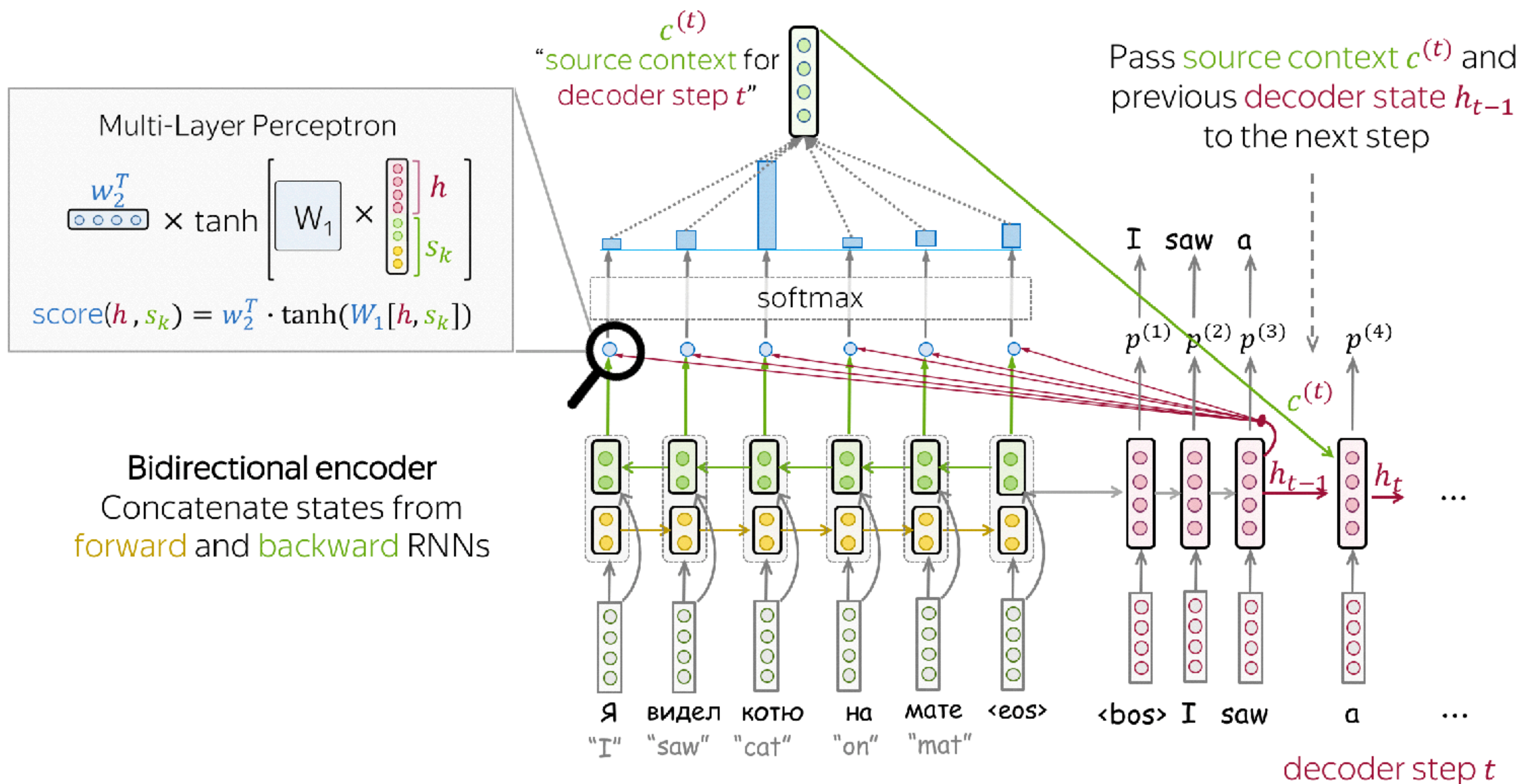
Multi-Layer Perceptron

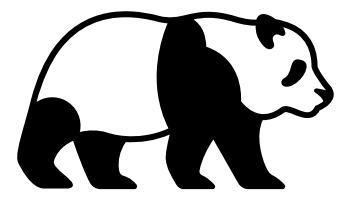


$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

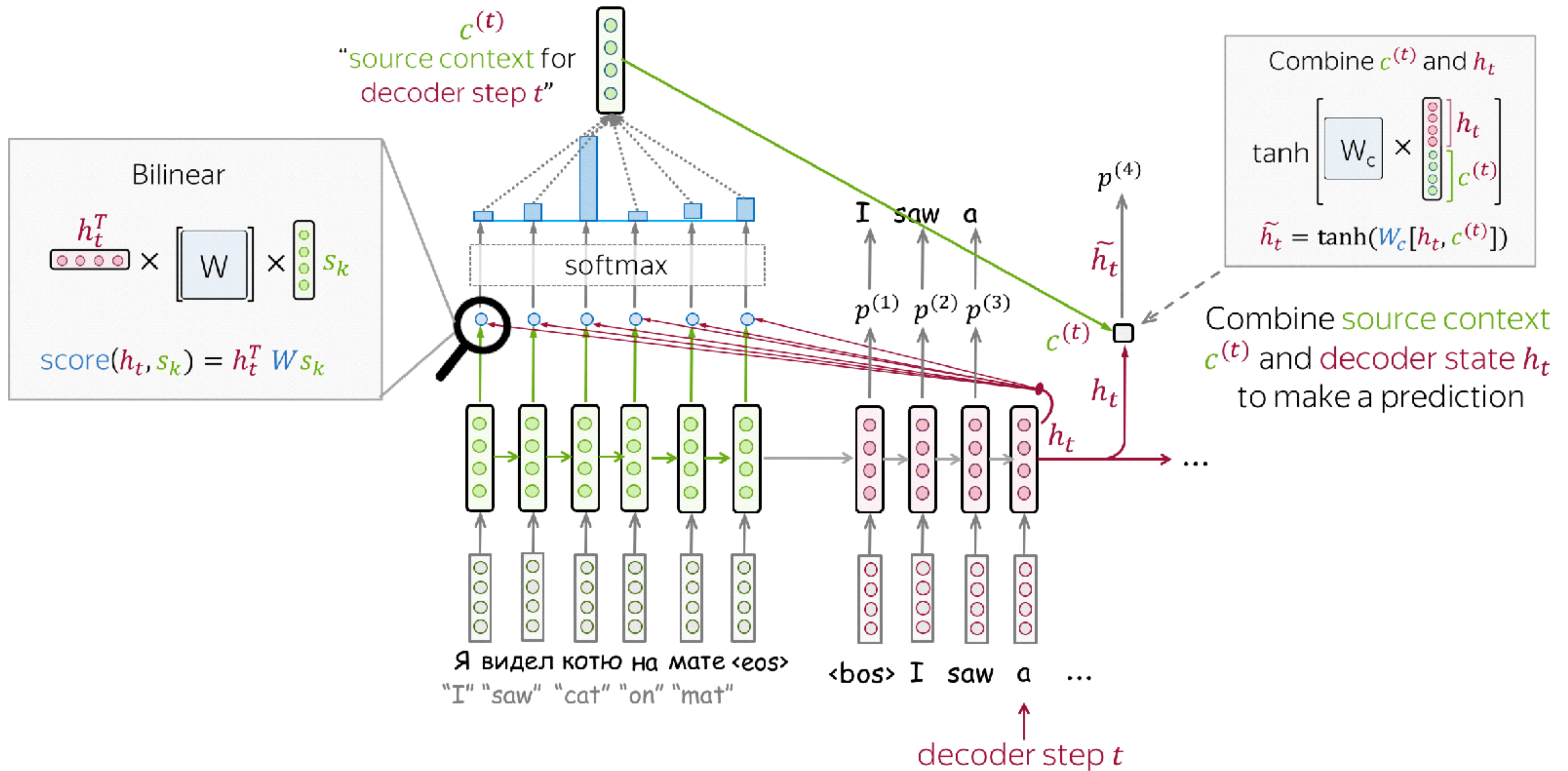


# Bahdanau Model





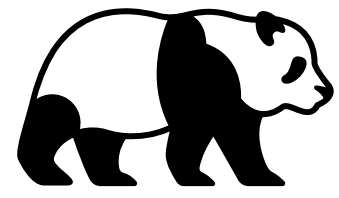
# Luong Model







**Transformer:**  
**Attention is All You Need**

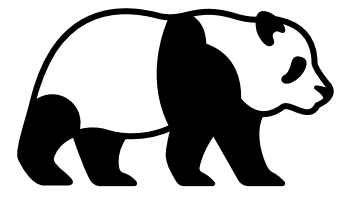


# What is Transformer

- A model introduced in the paper “**Attention is All You Need**” in 2017.
- Based **solely on attention** mechanisms (i.e., no recurrence or convolutions).
- Higher translation quality, faster to train.

	Seq2seq without attention	Seq2seq with attention	Transformer
processing within <b>encoder</b>	RNN/CNN	RNN/CNN	attention
processing within <b>decoder</b>	RNN/CNN	RNN/CNN	attention
<b>decoder-encoder</b> interaction	static fixed-sized vector	attention	attention





# What We Just Saw

## Encoder

Who is doing:

- all source tokens

What they are doing:

- look at each other
  - update representations
- repeat  
N times

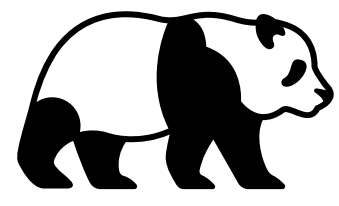
## Decoder

Who is doing:

- target token at the current step

What they are doing:

- looks at previous target tokens
  - looks at source representations
  - update representation
- repeat  
N times



# Why Such Design

- RNN won't understand what "bank" means until they read the whole sentence.
- Transformer's encoder tokens interact with each other all at once.

I arrived at the **bank** after crossing the ... ..street? ...river?

What does **bank** mean in this sentence?



I've no idea: let's wait until I read the end

RNNs

$O(N)$  steps to process a sentence with length  $N$

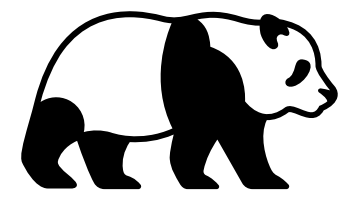


I don't need to wait - I see all words at once!

Transformer

Constant number of steps to process any sentence

# How to Implement



# Self-Attention: the "Look at Each Other" Part

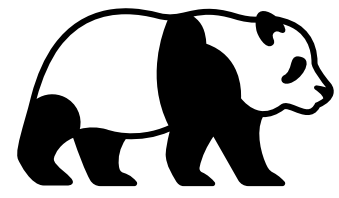
- Self-attention is one of the key components of the model.
- The difference between attention and self-attention is that self-attention operates between representations of the same nature: e.g., all encoder states in some layer.

Decoder-encoder attention is looking

- **from:** one current decoder state
- **at:** all encoder states

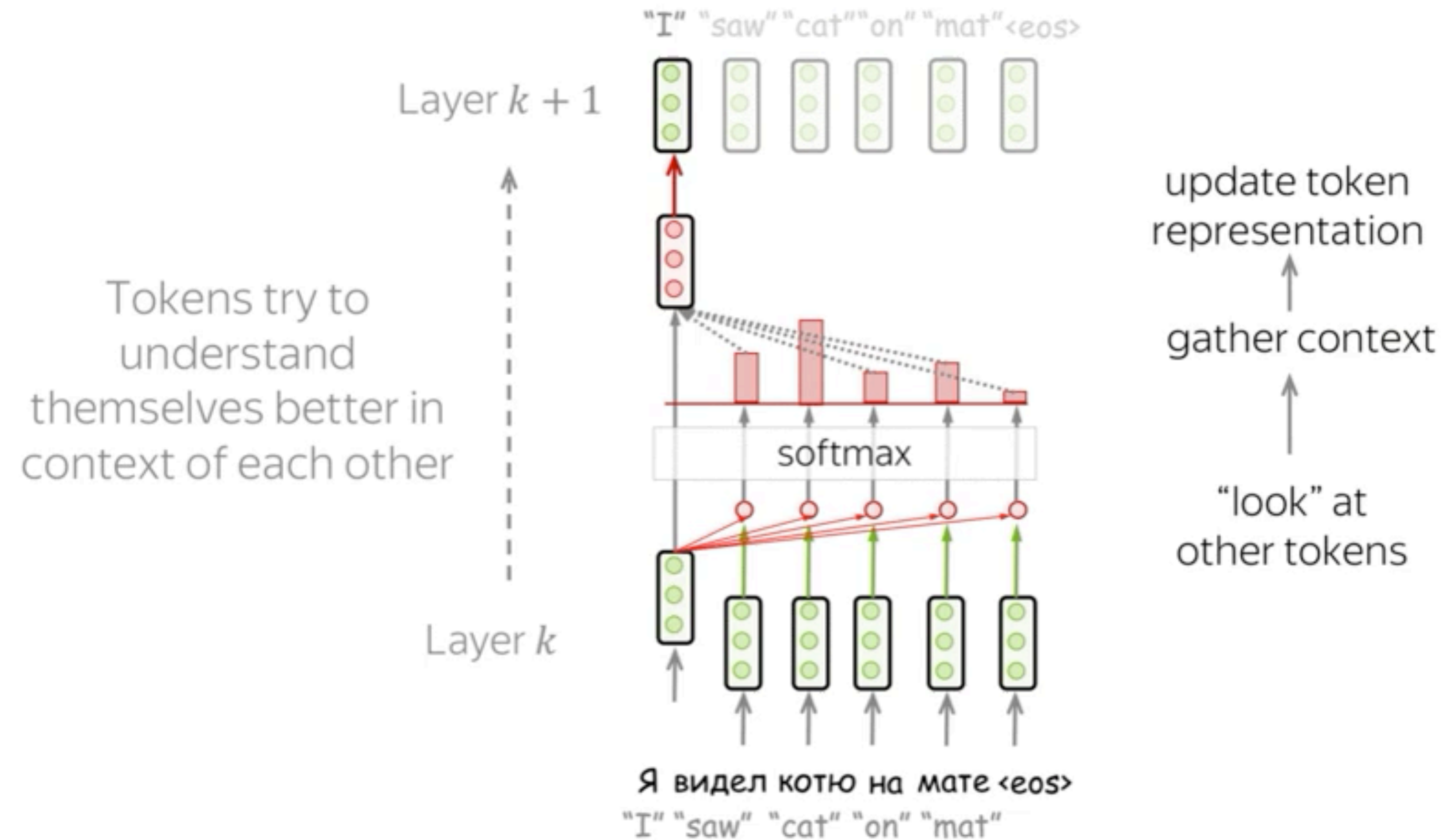
Self-attention is looking

- **from:** each state from a set of states
- **at:** all other states in the same set

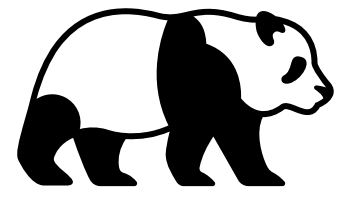


# Self-Attention: the "Look at Each Other" Part

- **Self-attention** is the part of the model where **tokens interact with each other**.
- Each token "looks" at other tokens in the sentence with an attention mechanism, gathers context, and updates the previous representation of "self".
- Note that in practice, this happens **in parallel**.







# Query, Key, and Value in Self-Attention

- Each input token in **self-attention** receives three representations corresponding to the roles it can play:
  - **query** - asking for information;
  - **key** - saying that it has some information;
  - **value** - giving the information.
- The **query** is used when a token looks at others - it's **seeking the information** to understand itself better.
- The **key** is **responding to a query's request**: it is used to **compute attention weights**.
- The **value** is used to **compute attention output**: it gives information to the tokens which "say" they need it (i.e. assigned large weights to this token).

Each vector receives three representations (“roles”)

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

**Query:** vector from which the attention is looking

“Hey there, do you have this information?”

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

**Key:** vector at which the query looks to compute weights

“Hi, I have this information – give me a large weight!”

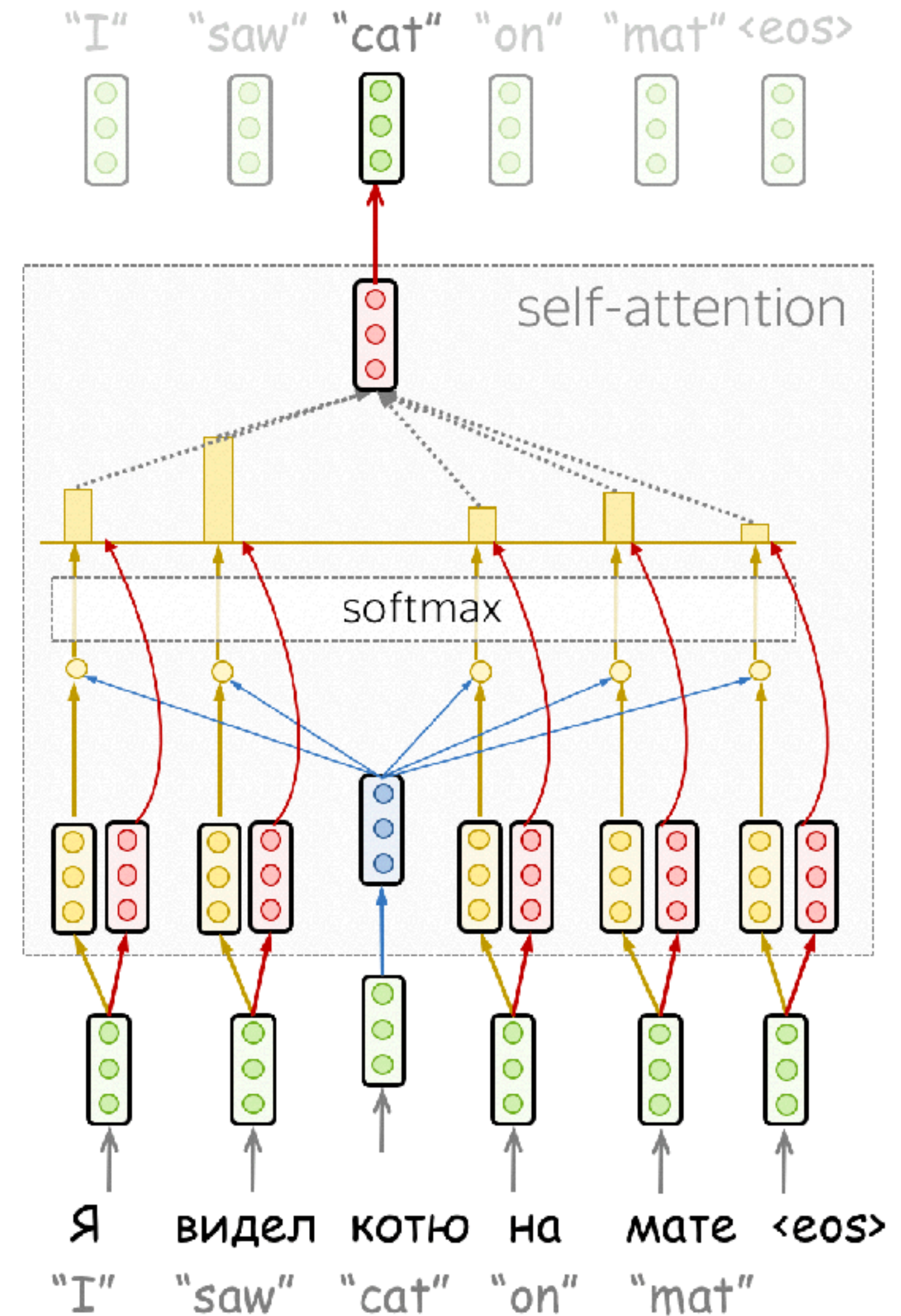
$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

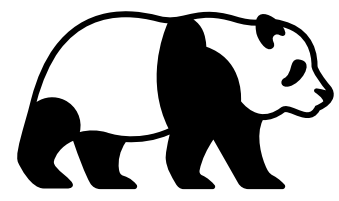
**Value:** their weighted sum is attention output

“Here’s the information I have!”

$$Attention(q, k, v) = \overbrace{\text{softmax}\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from to vector dimensionality of K, V

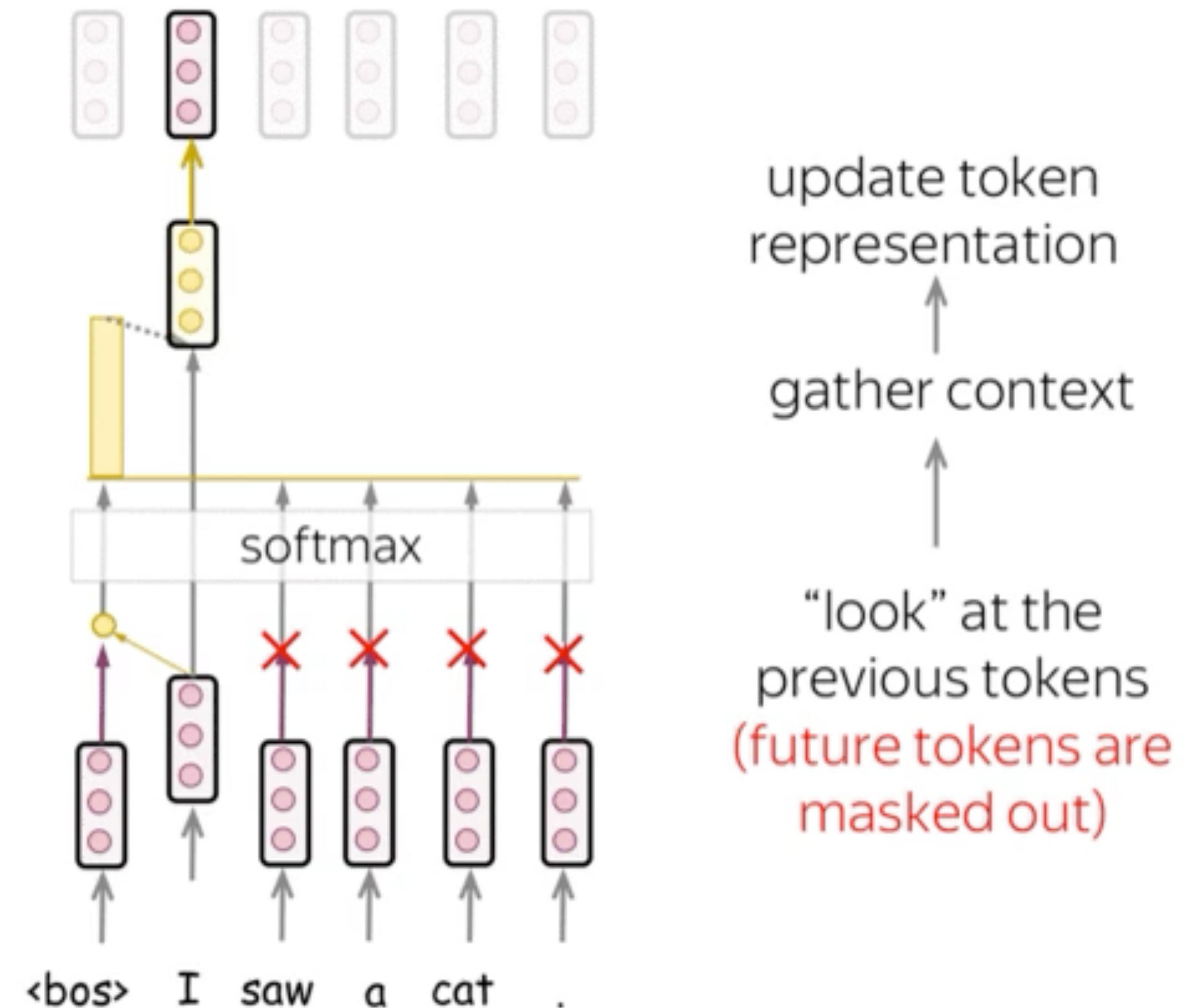


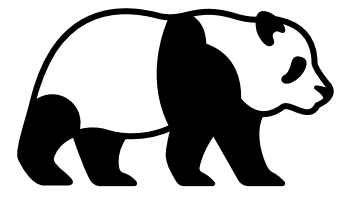


# Masked Self-Attention

## "Don't Look Ahead" for the Decoder

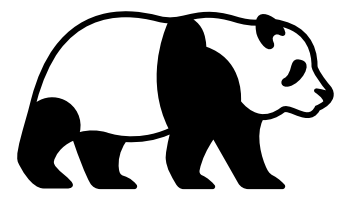
- In the **decoder**, there's also a **self-attention** mechanism: it is the one performing the "**look at the previous tokens**" function.
- In the decoder, self-attention is a bit **different from the one in the encoder**. While the encoder receives all tokens at once and the tokens can look at all tokens in the input sentence, in the decoder, we generate one token at a time: **during generation, we don't know which tokens we'll generate in future**.
- To forbid the decoder to look ahead, the model uses **masked self-attention**: future tokens are masked out. Look at the illustration.





# But How Can The Decoder Look Ahead?

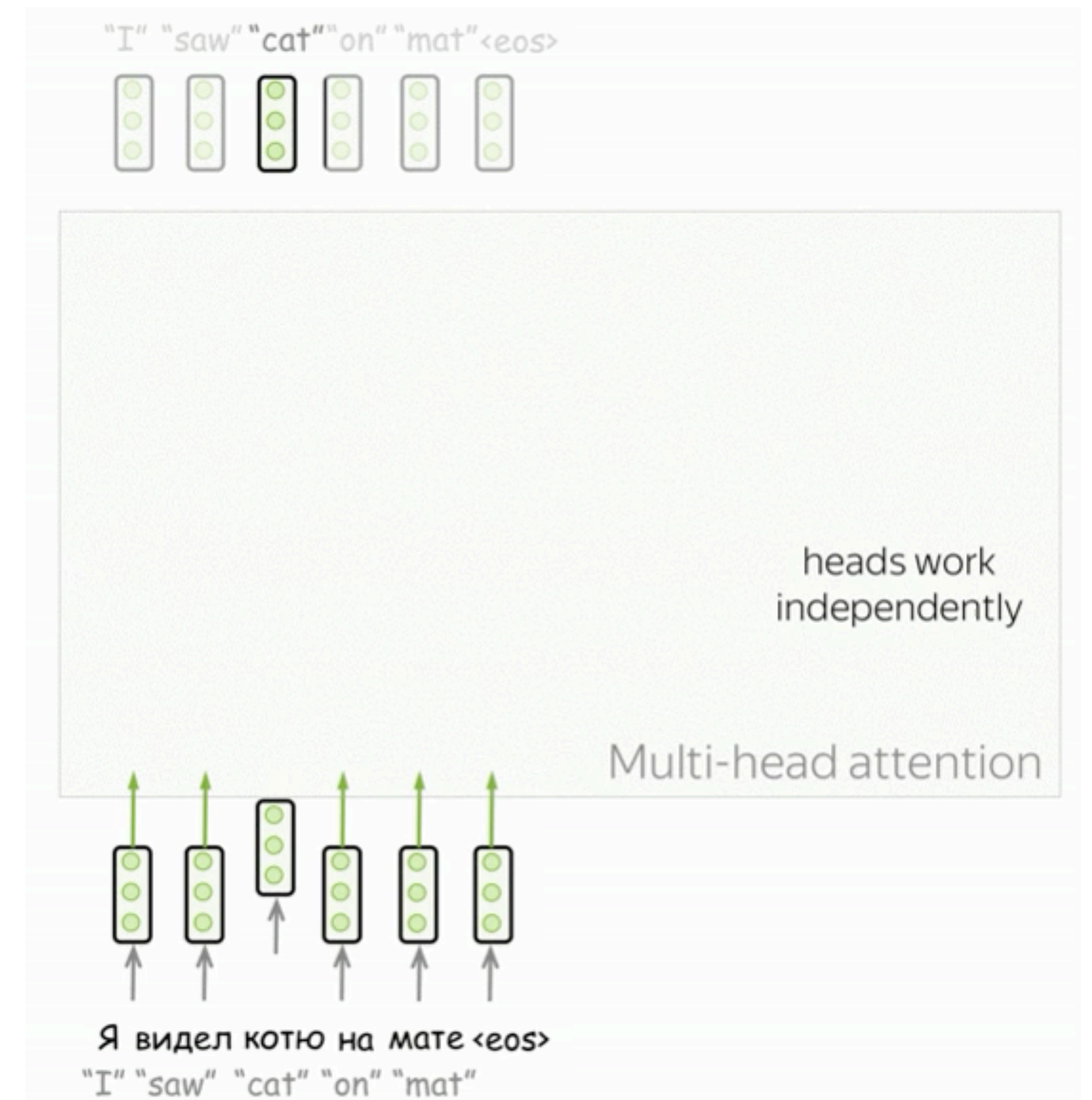
- During generation, it can't - we don't know what comes next.
- But in training, we use reference translations (which we know). Therefore, in training, we feed the whole target sentence to the decoder - without masks, the tokens would "see future", and this is not what we want.
- This is done for computational efficiency: **the Transformer does not have a recurrence, so all tokens can be processed at once**. This is one of the reasons it has become so popular for machine translation - **it's much faster to train** than the once dominant recurrent models. For recurrent models, one training step requires  $O(\text{len}(\text{source}) + \text{len}(\text{target}))$  steps, but for Transformer, it's  $O(1)$ , i.e. constant.

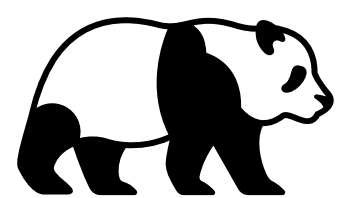


# Multi-Head Attention

## Independently Focus on Different Things

- Usually, understanding the role of a word in a sentence requires understanding how it is related to different parts of the sentence.
- This is important not only in processing source sentence but also in generating target. For example, in some languages, subjects define verb inflection (e.g., gender agreement), verbs define the case of their objects, and many more. What I'm trying to say is: **each word is part of many relations.**
- Therefore, we have to **let the model focus on different things**: this is the motivation behind Multi-Head Attention. **Instead of having one attention mechanism, multi-head attention has several "heads" which work independently.**

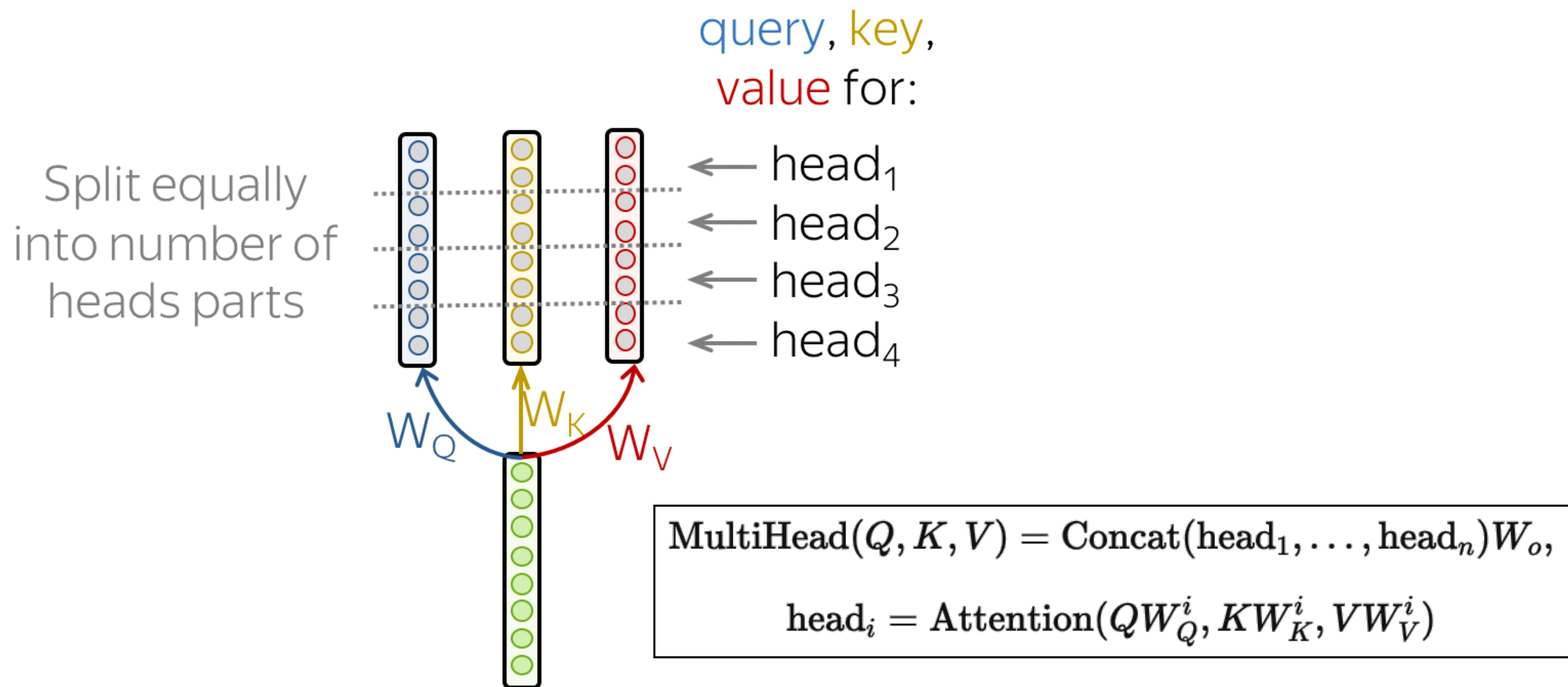


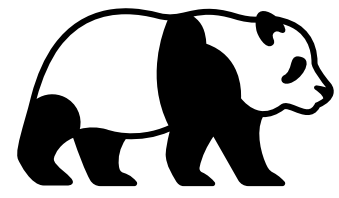


# Multi-Head Attention

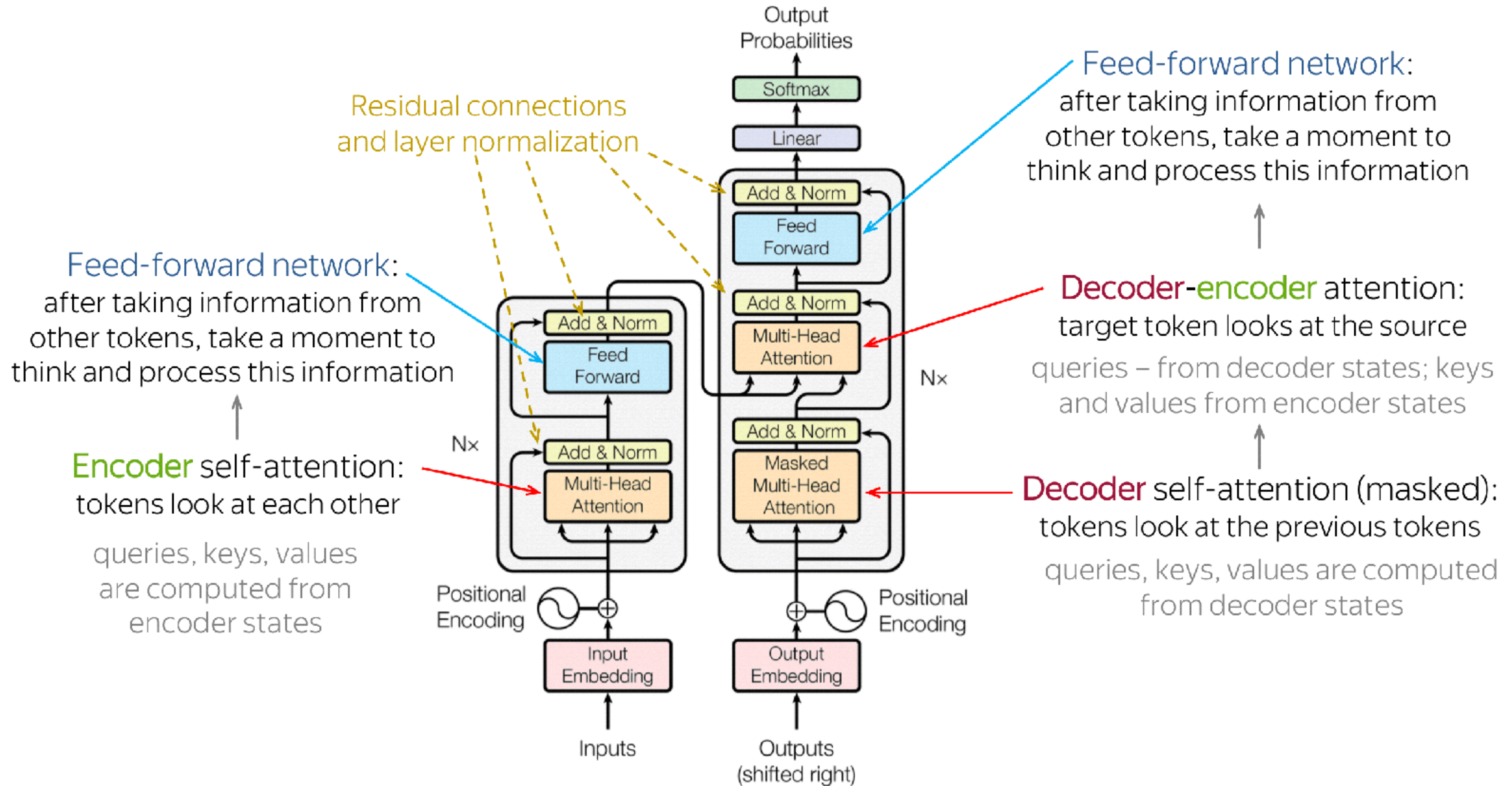
## Independently Focus on Different Things

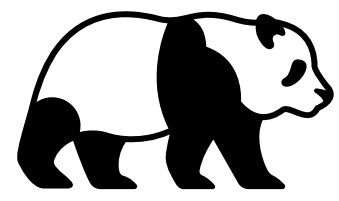
- Formally, this is implemented as several attention mechanisms whose results are combined:





# Transformer: Model Architecture



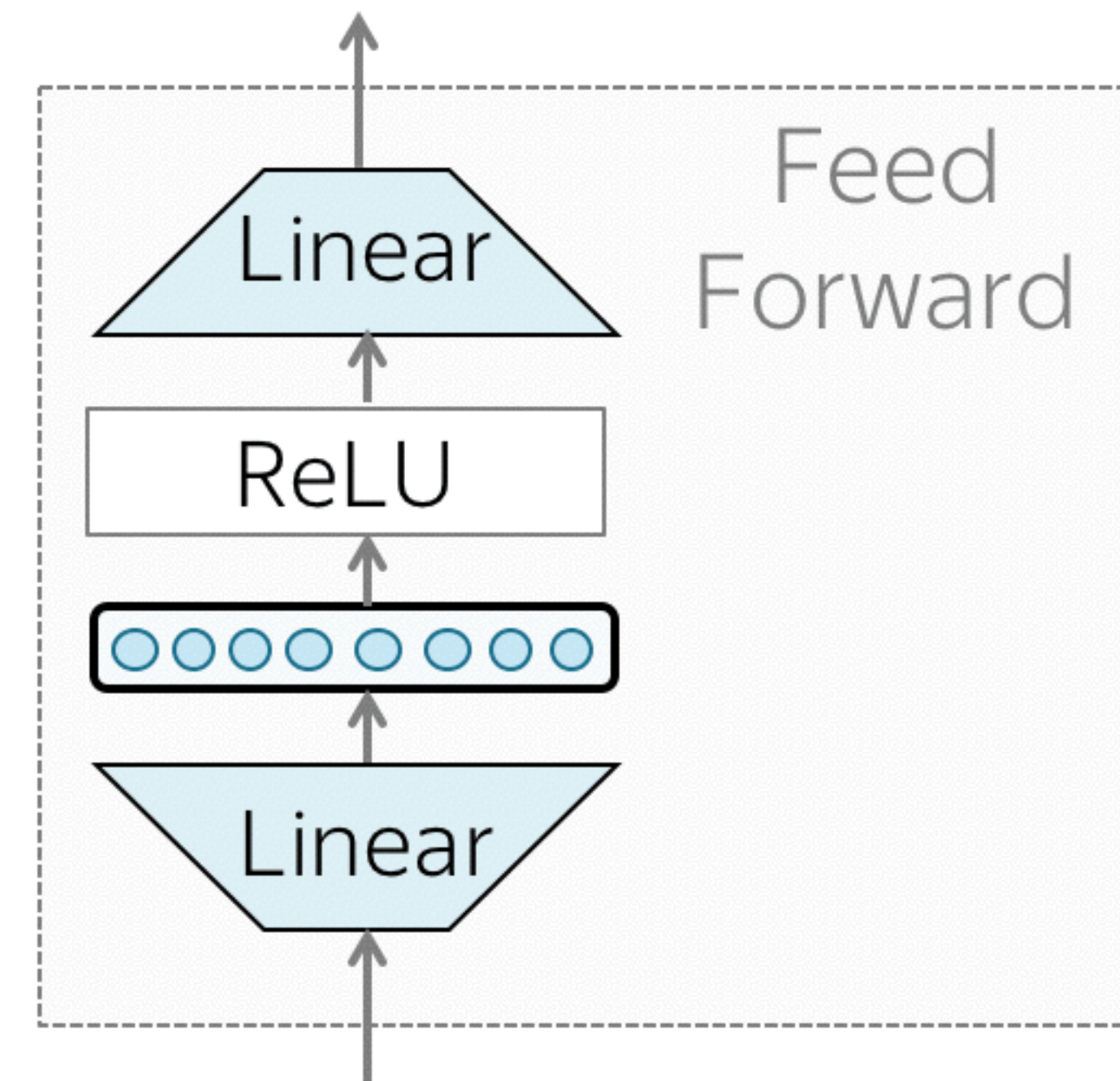


# Transformer: Feed-forward Blocks

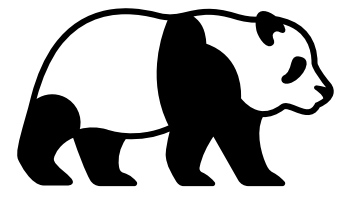
- In addition to attention, each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

- After looking at other tokens via an attention mechanism, a model uses an FFN block to process this new information (attention - "look at other tokens and gather information", FFN - "take a moment to think and process this information").

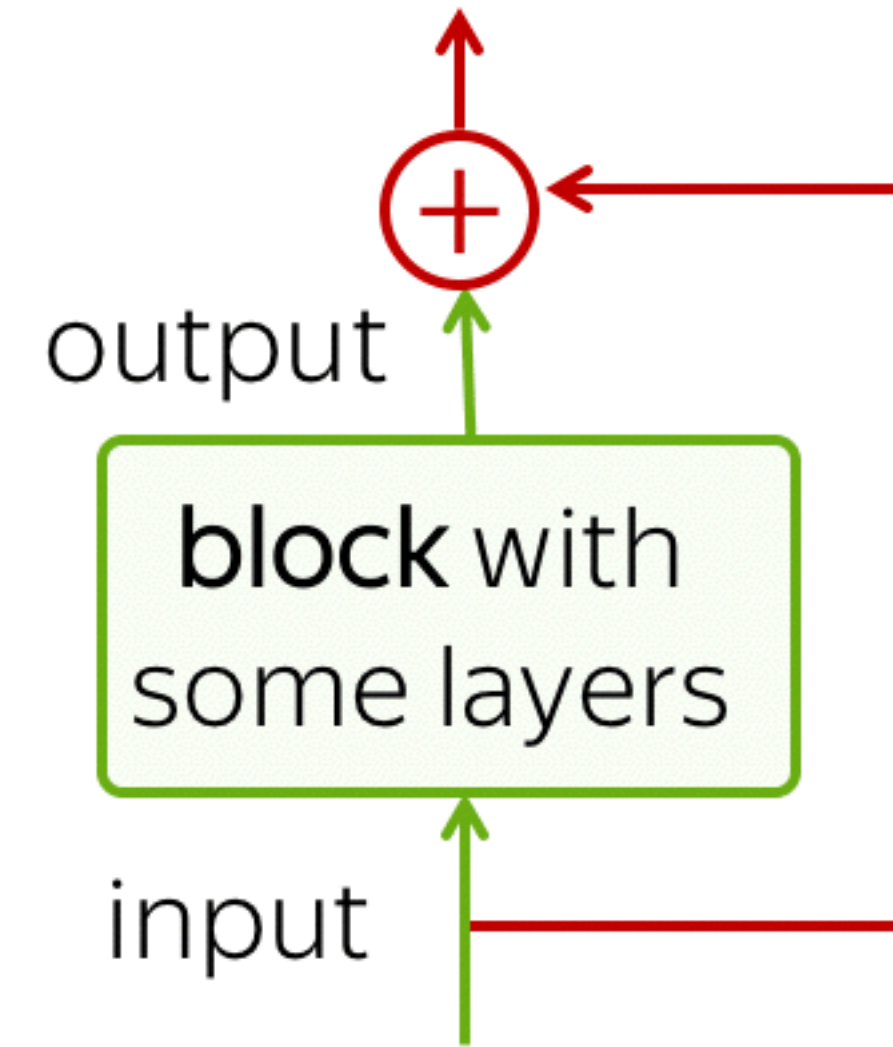




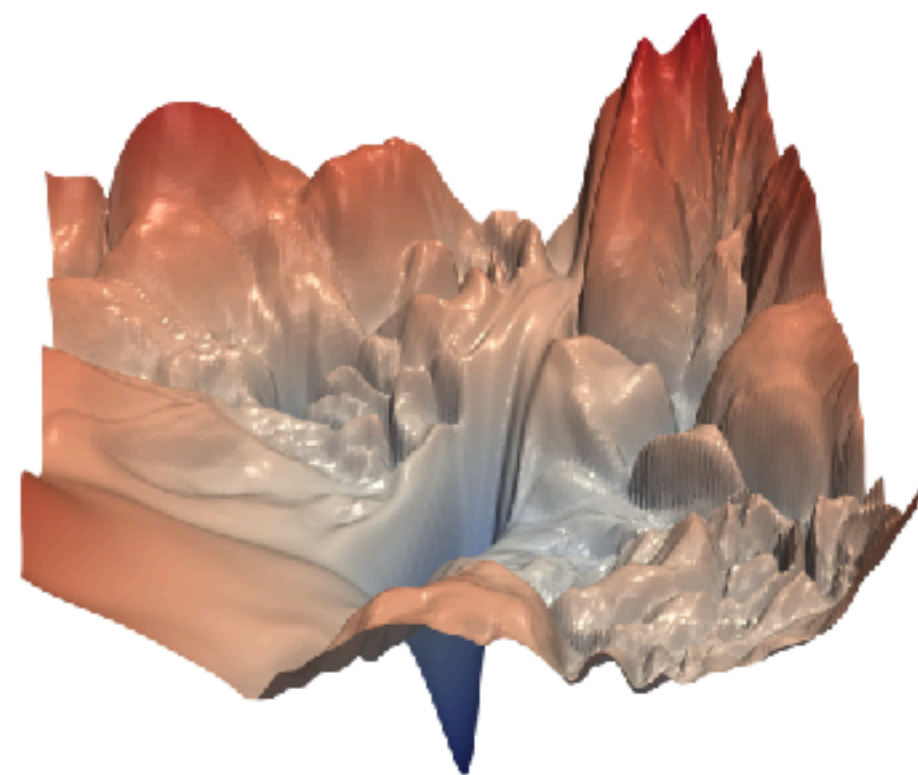


# Transformer: Residual Connections

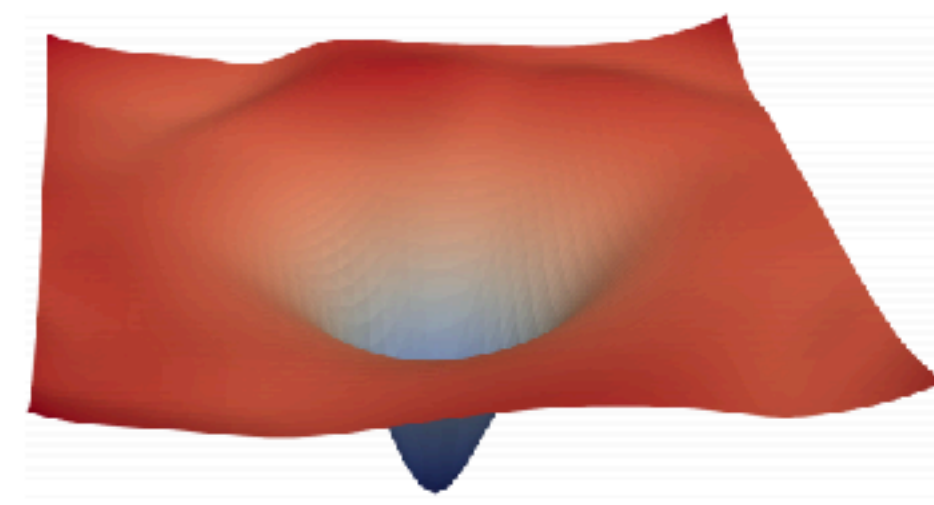
- Residual connections are very simple (**add a block's input to its output**), but at the same time are very useful: they ease the gradient flow through a network and allow stacking a lot of layers.
- In the Transformer, residual connections are used after each attention and FFN block. On the illustration above, residuals are shown as arrows coming around a block to the yellow "Add & Norm" layer. In the "Add & Norm" part, the "Add" part stands for the residual connection.



Residual connection:  
add a block's input to  
its output



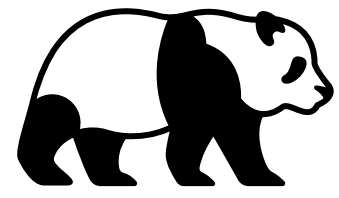
(a) without skip connections



(b) with skip connections

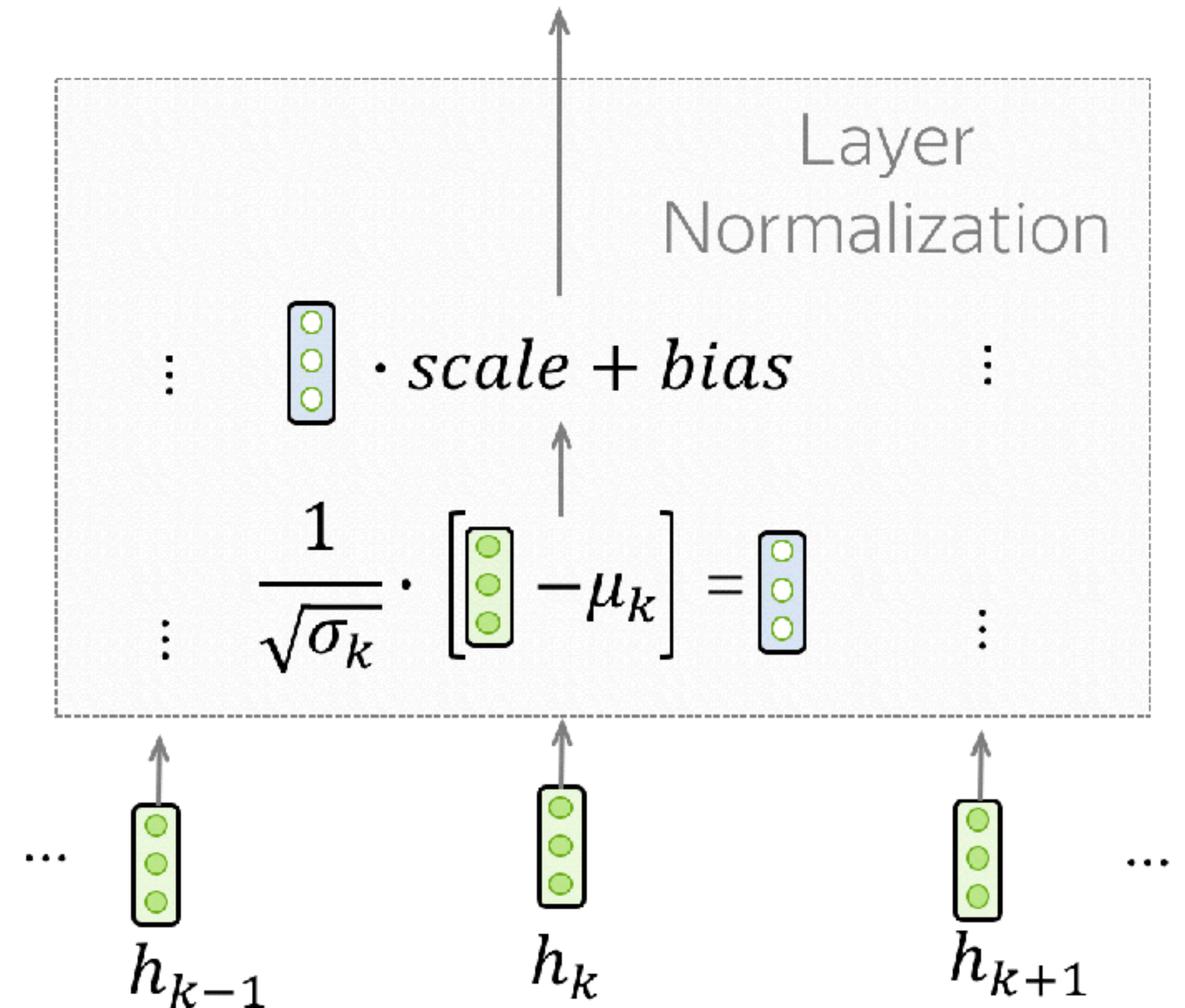
- Residual connections are thought to **make the loss landscape considerably smoother** (thus easier training!)

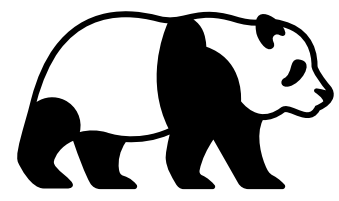
[Loss landscape visualization, [Li et al., 2018](#), on a ResNet]



# Transformer: Layer Normalization

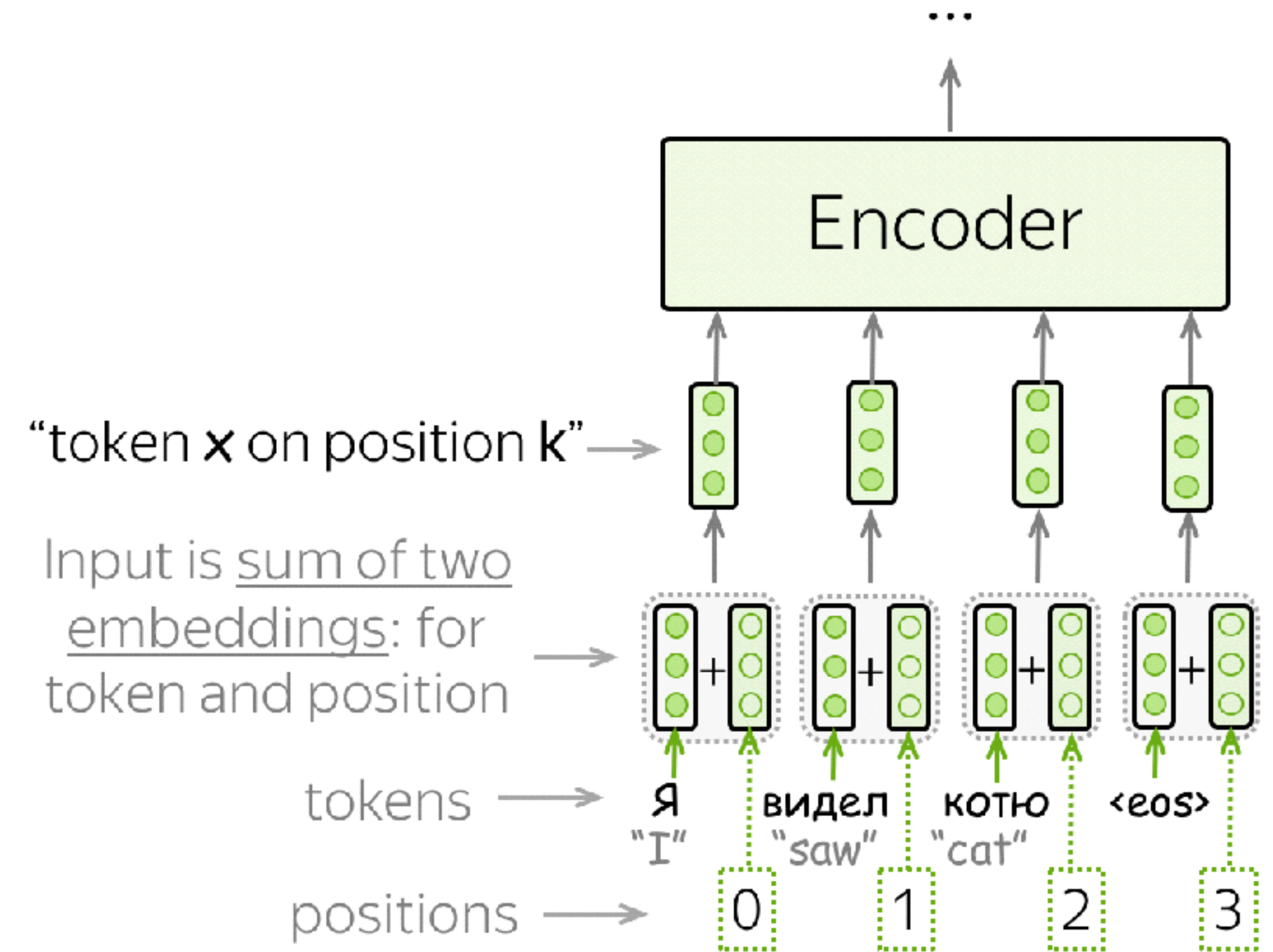
- The "**Norm**" part in the "**Add & Norm**" layer denotes **Layer Normalization**. It independently normalizes vector representation of each example in batch - this is done to control "flow" to the next layer. Layer normalization improves convergence stability and sometimes even quality.
- In the Transformer, you have to normalize vector representation of each token. Additionally, here LayerNorm has trainable parameters, *scale* and *bias*, which are used after normalization to rescale layer's outputs (or the next layer's inputs). Note that  $\mu_k$  and  $\sigma_k$  are evaluated for each example, but *scale* and *bias* are the same - these are layer parameters.

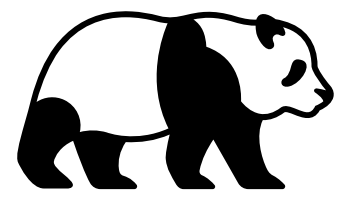




# Transformer: Positional Encoding

- Note that since Transformer does not contain recurrence or convolution, **it does not know the order of input tokens**.
- Therefore, we have to let the model know the positions of the tokens explicitly. For this, **we have two sets of embeddings: for tokens** (as we always do) and **for positions** (the new ones needed for this model). Then input representation of a token is the sum of two embeddings: token and positional.





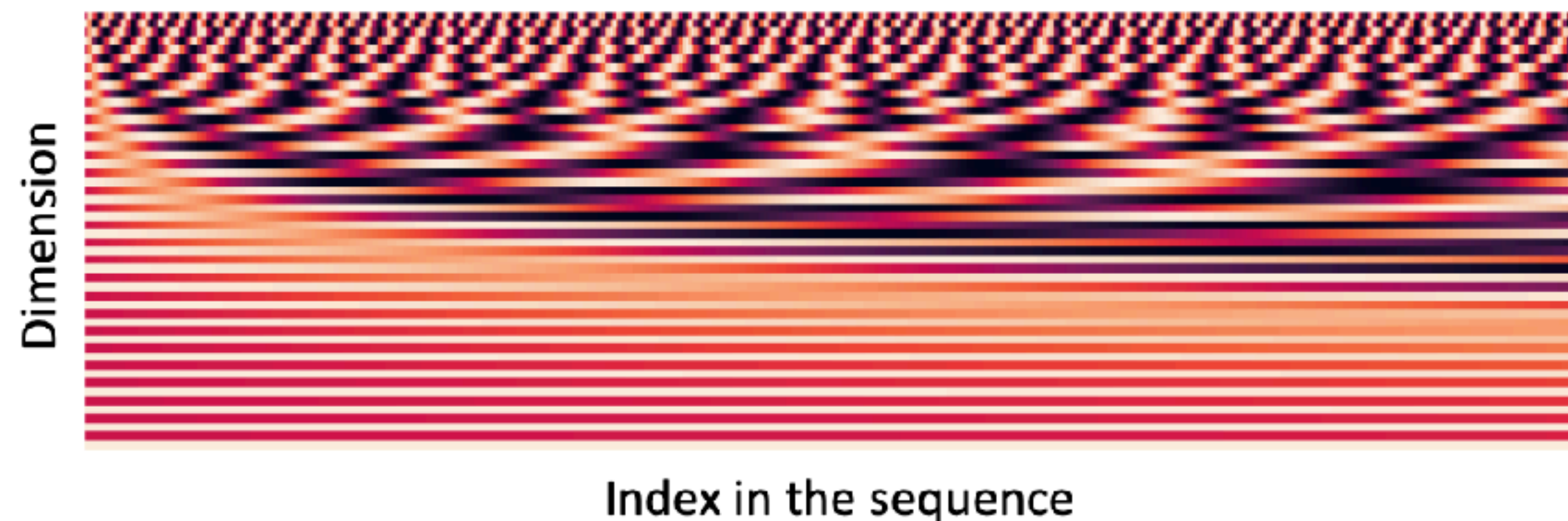
# Transformer: Positional Encoding

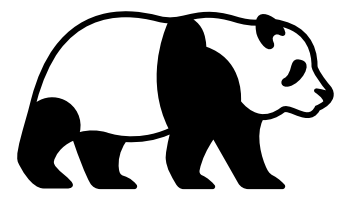
- The positional embeddings can be learned, but the authors found that having fixed ones does not hurt the quality. The fixed positional encodings used in the Transformer are:

$$\text{PE}_{pos,2i} = \sin(pos/10000^{2i/d_{model}}),$$

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

- where  $pos$  is position and  $i$  is the vector dimension. Each dimension of the positional encoding corresponds to a sinusoid, and the wavelengths form a geometric progression from  $2\pi$  to  $10000 \cdot 2\pi$ .





# Transformer: Positional Encoding

- **Fixed positional encoding**

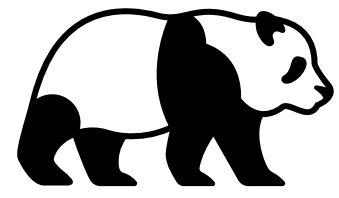
- Pros: Periodicity indicates that maybe “absolute position” isn’t as important
- Pros: Maybe can extrapolate to longer sequences as periods restart!
- Cons: Not learnable; also the extrapolation doesn’t really work!

- **Learned absolute position representations**

- Pros: Flexibility: each position gets to be learned to fit the data
- Cons: Definitely can’t extrapolate to indices outside  $1, \dots, T$ .
- Most systems use this.

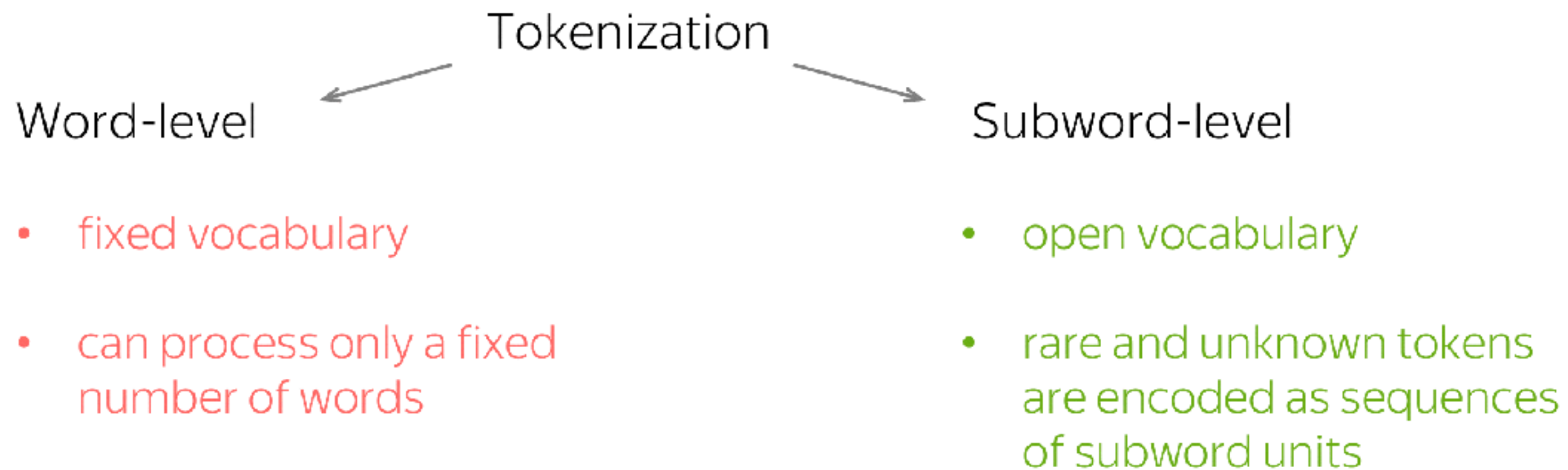
- **Sometimes people try more flexible representations of position:**

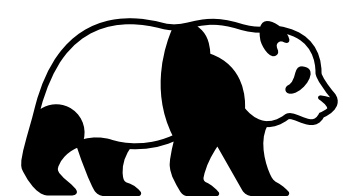
- Relative linear position attention [[Shaw et al., 2018](#)]
- Dependency syntax-based position [[Wang et al., 2019](#)]



# Subword Segmentation: Byte Pair Encoding

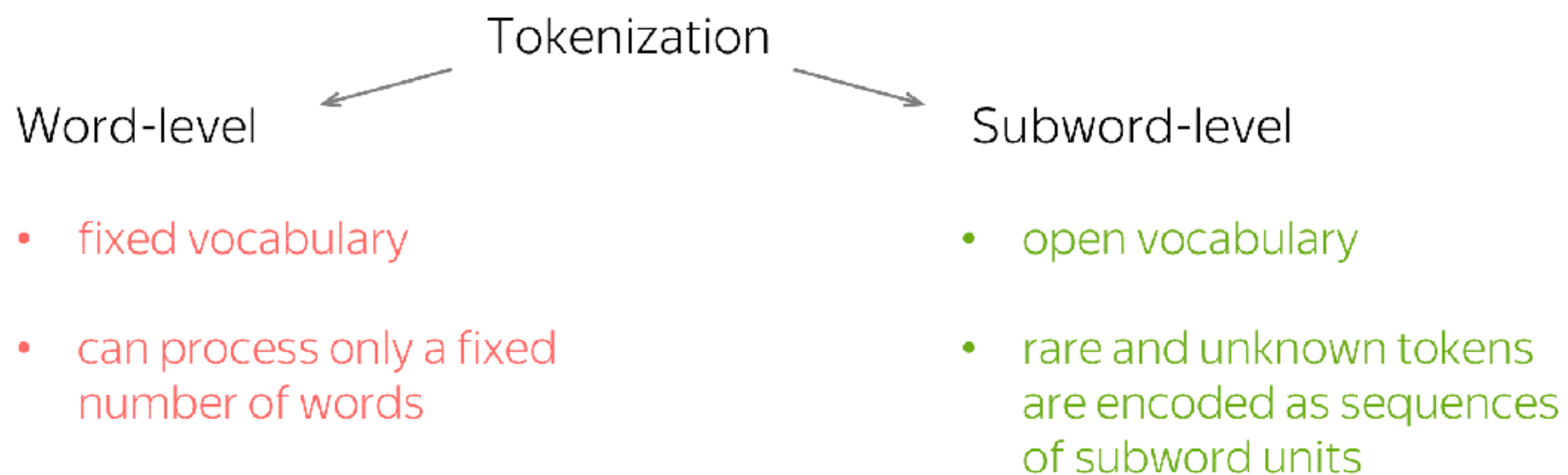
- A model has a predefined vocabulary of tokens.
- Tokens not in the vocabulary will be replaced with a special UNK ("unknown") token.
- Therefore, if your tokens are words, you will be able to process a fixed number of words.
- This is the **fixed vocabulary problem** : you will be getting lot's of unknown tokens, and your model won't translate them properly.
  
- But how can we represent all words, even those we haven't seen in the training data?
- Well, even if you are not familiar with a word, you are familiar with the parts it consists of - **subwords** (in the worst case, symbols). Then why don't we split the rare and unknown words into smaller parts?

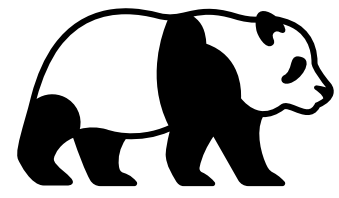




# Subword Segmentation: Byte Pair Encoding

- The original **Byte Pair Encoding (BPE)** (Gage, 1994) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. What we refer to as BPE now is **an adaptation of this algorithm for word segmentation**. Instead of merging frequent pairs of bytes, it merges characters or character sequences.
- BPE algorithm consists of two parts:
  - **training** - learn "BPE rules", i.e., which pairs of symbols to merge;
  - **inference** - apply learned rules to segment a text.

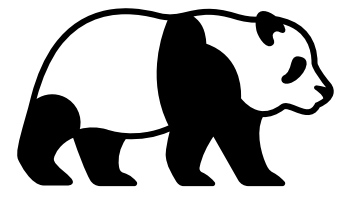




# Training: Learn BPE Rules

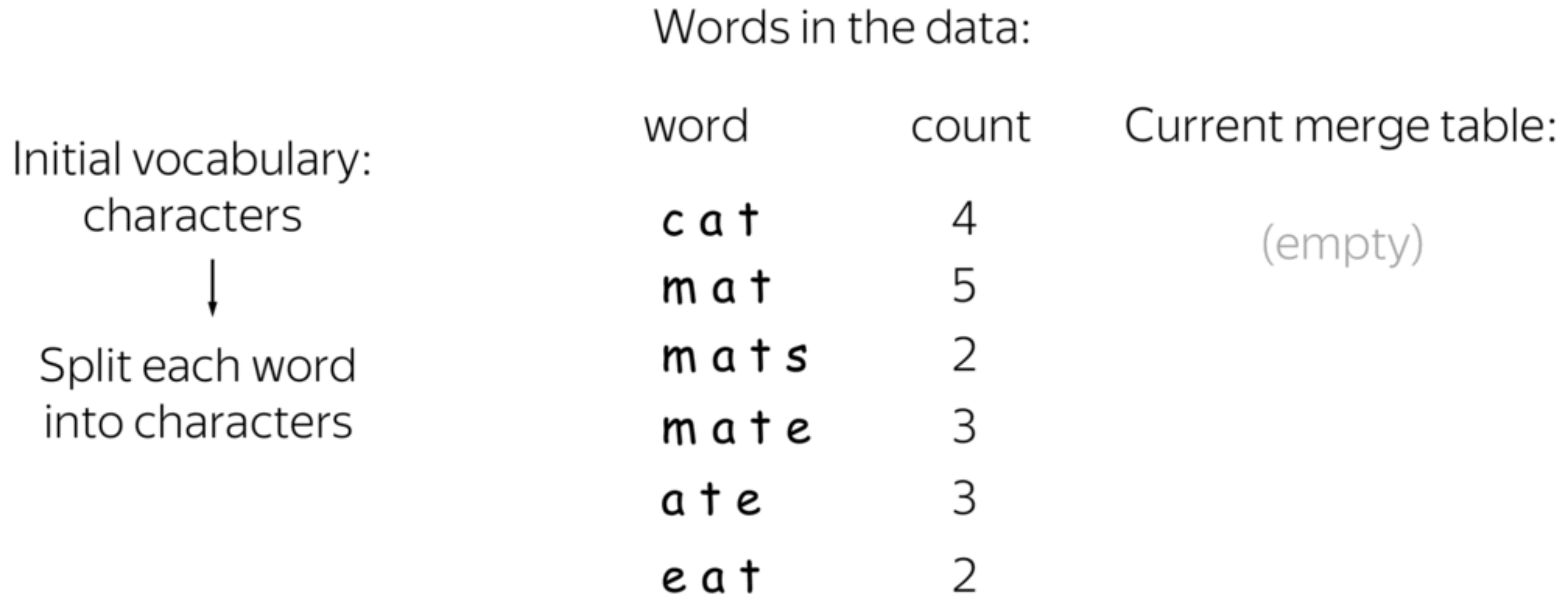
- ⦿ At this step, the algorithm builds a merge table and a vocabulary of tokens. The initial vocabulary consists of characters and an empty merge table. At this step, each word is segmented as a sequence of characters. After that, the algorithm is as follows:
  - count pairs of symbols: how many times each pair occurs together in the training data;
  - find the most frequent pair of symbols;
  - merge this pair - add a merge to the merge table, and the new token to the vocabulary.
- ⦿ In practice, the algorithm first counts how many times each word appeared in the data. Using this information, it can count pairs of symbols more easily. Note also that the tokens do not cross word boundary - everything happens within words.

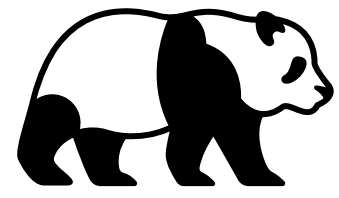




# Training: Learn BPE Rules

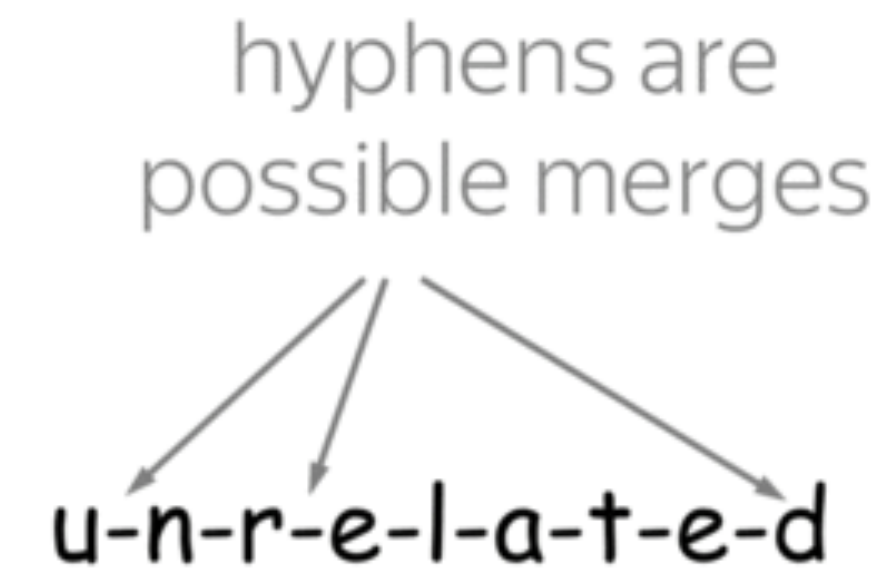
- Here I show you a toy example: here we assume that in training data, we met **cat** 4 times, **mat** 5 times and **mats**, **mate**, **ate**, **eat** 2, 3, 3, 2 times, respectively. We also have to set the maximum number of merges we want; usually, it's going to be about 4k-32k depending on the dataset size, but for our toy example, let's set it to 5.

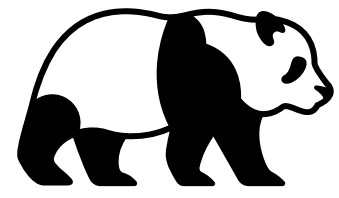




# Inference: Segment a Text

- After learning BPE rules, you have a merge table - now, we will use it to segment a new text.
- The algorithm starts with segmenting a word into a sequence of characters. After that, it iteratively makes the following two steps until no merge is possible:
  - among all possible merges at this step, find the highest merge in the table;
  - apply this merge.
- Note that the merge table is ordered - the merges that are higher in the table were more frequent in the data. That's why in the algorithm, merges that are higher have higher priority: at each step, we merge the most frequent merge among all possible.





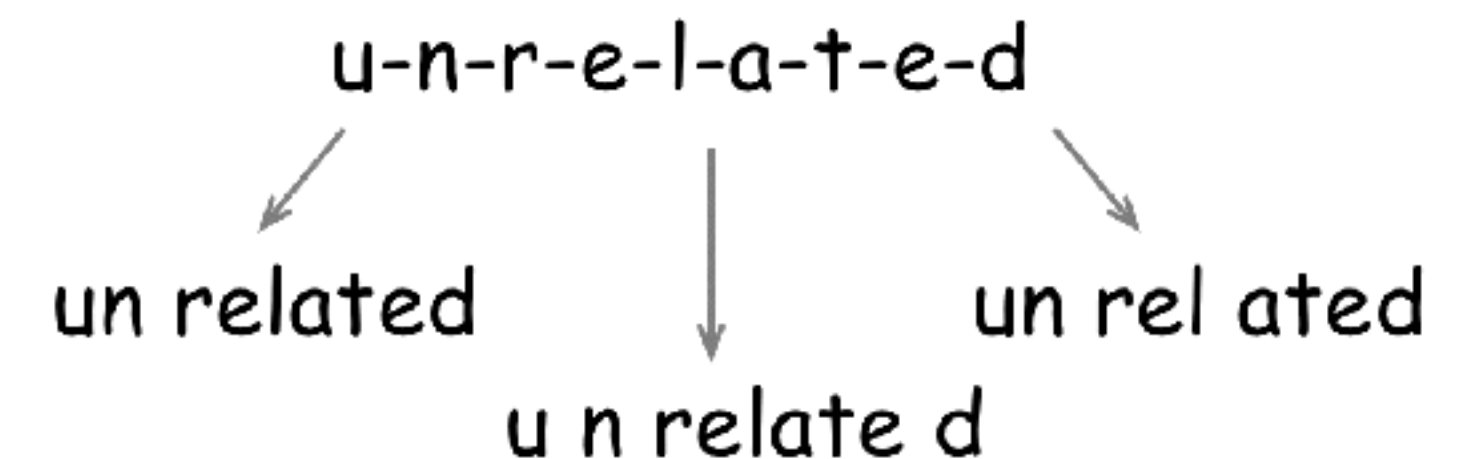
# Make BPE Stochastic: Segment Words Differently

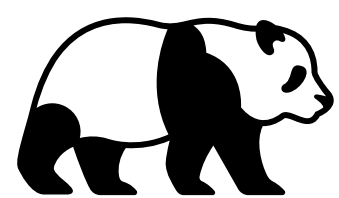
- The **standard BPE segmentation is deterministic**: at each step, it always picks the highest merge in the table. However, even with the same vocabulary, **a word can have different segmentations**, e.g. un relat ed, u n relate d, un rel ated, etc.).
- Possible reasons why showing different segmentations of the same word can help a model are:
  - with different segmentations of a word, a model can better understand the subwords it consists of. Therefore, it can better understand word composition.
  - since only rare and unknown words are split into subwords, a model may not learn representations for subwords very well. With different segmentations, it will see subwords in many different contexts and will understand them better.
  - this may serve as a regularization - a model will learn not to over-rely on individual tokens and to consider a broader context (similar to the standard word dropout).

BPE: deterministic

u-n-r-e-l-a-t-e-d  
u-n re-l-a-t-e-d  
u-n re-l-at-e-d  
u-n re-l-at-ed  
un re-l-at-ed  
un re-l-ated  
un rel-ated  
un related

We want: stochastic





# BPE-Dropout

## Drop Some Merges From The Merge Table

- BPE-Dropout: Simple and Effective Subword Regularization (ACL 2020).
- The idea is very simple: if BPE is deterministic because we pick the highest merge, all we need to do is to (sometimes) pick other merges. For this, the authors **randomly drop some merges** (e.g., 10% of all merges) from the BPE merge table. In this case, the highest merge is sometimes dropped from the table, we'll have to pick the other one, and the segmentation will be different.

### Algorithm 1: BPE-dropout

```

current_split ← characters from input_word;
do
  merges ← all possible merges of tokens
  from current_split;
  for merge from merges do
    /* The only difference
       from BPE */
    remove merge from merges with the
    probability p;
  end
  if merges is not empty then
    merge ← select the merge with the
    highest priority from merges;
    apply merge to current_split;
  end
while merges is not empty;
return current_split;

```

```

u-n-r-e-l-a-t-e-d
u-n re-l-a-t-e-d
u-n re-l-at-e-d
u-n re-l-at-ed
un re-l-at-ed
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
unrelated

```

(a) BPE

```

u-n_r-e-l-a_t-e_d
u-n re-l_a-t-e_d
u-n re_l-at-e_d
un re-l-at-e_d
un re_l-at-ed
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
un re-l-ated
un relate_d

```

(b) BPE-dropout

```

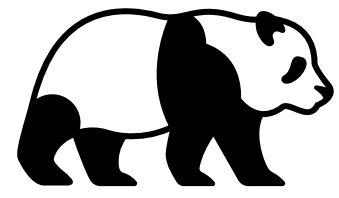
u-n_r_e_l-a_t-e-d
u-n-r_e-l-at-e-d
u-n-r_e-l_at_ed
un-r_e-l-at-ed
un re-l_at-ed
un re-l-ated
un re-l-ated

```

Segmentation process of the word 'unrelated

using (a) BPE, (b) BPE-dropout.

# **Great Results with Transformers**

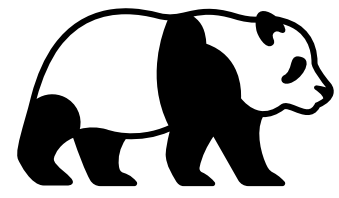


# Great Results with Transformers

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

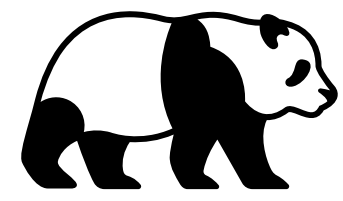
## Machine Translation



# Great Results with Transformers

<b>Model</b>	<b>Test perplexity</b>	<b>ROUGE-L</b>
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	1.90325	38.8

## Document Generation



# Great Results with Transformers

On this popular aggregate benchmark, for example:



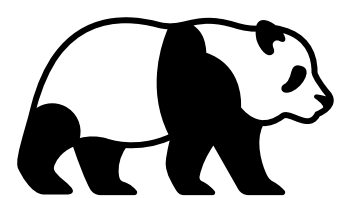
**All top models are Transformer (and pretraining)-based.**

**GLUE**

Rank	Name	Model	URL	Score
1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
2	HFL iFLYTEK	MacALBERT + DKM		90.7
<b>+</b>	3	Alibaba DAMO NLP	StructBERT + TAPT	90.6
<b>+</b>	4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS	90.6
	5	ERNIE Team - Baidu	ERNIE	90.4
	6	T5 Team - Google	T5	90.3



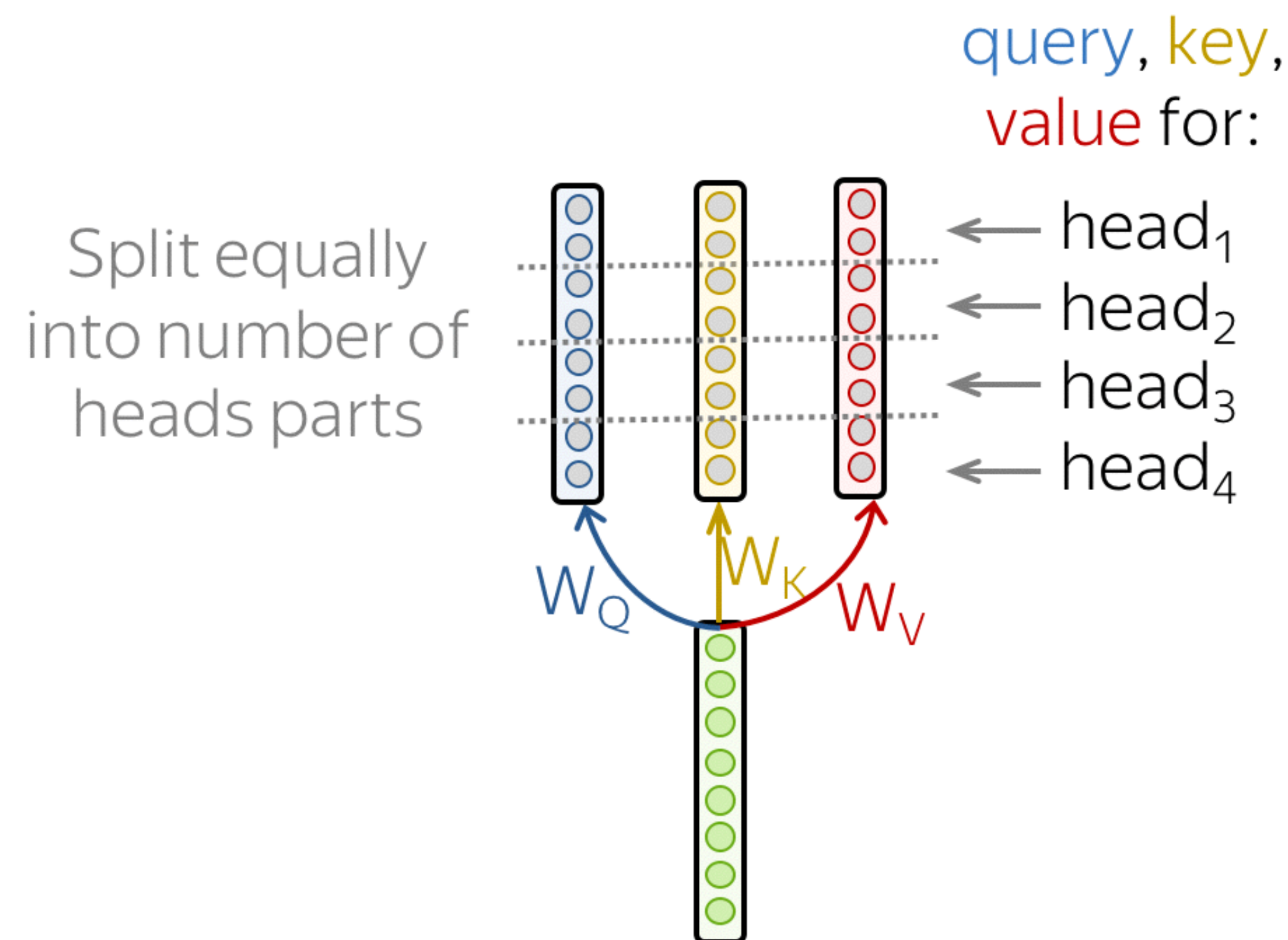
# **Analysis and Interpretability**

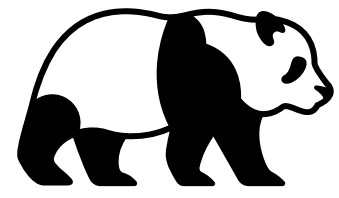


# Multi-Head Self-Attention

## What are these heads doing?

- Multi-head attention is an **inductive bias** introduced in the Transformer.
- When creating an inductive bias in a model, we usually have some kind of intuition for why we think this new model component, inductive bias, could be useful.
- Therefore, it's good to understand how this new thing works - does it learn the things we thought it would? If not, why it helps? If yes, how can we improve it?

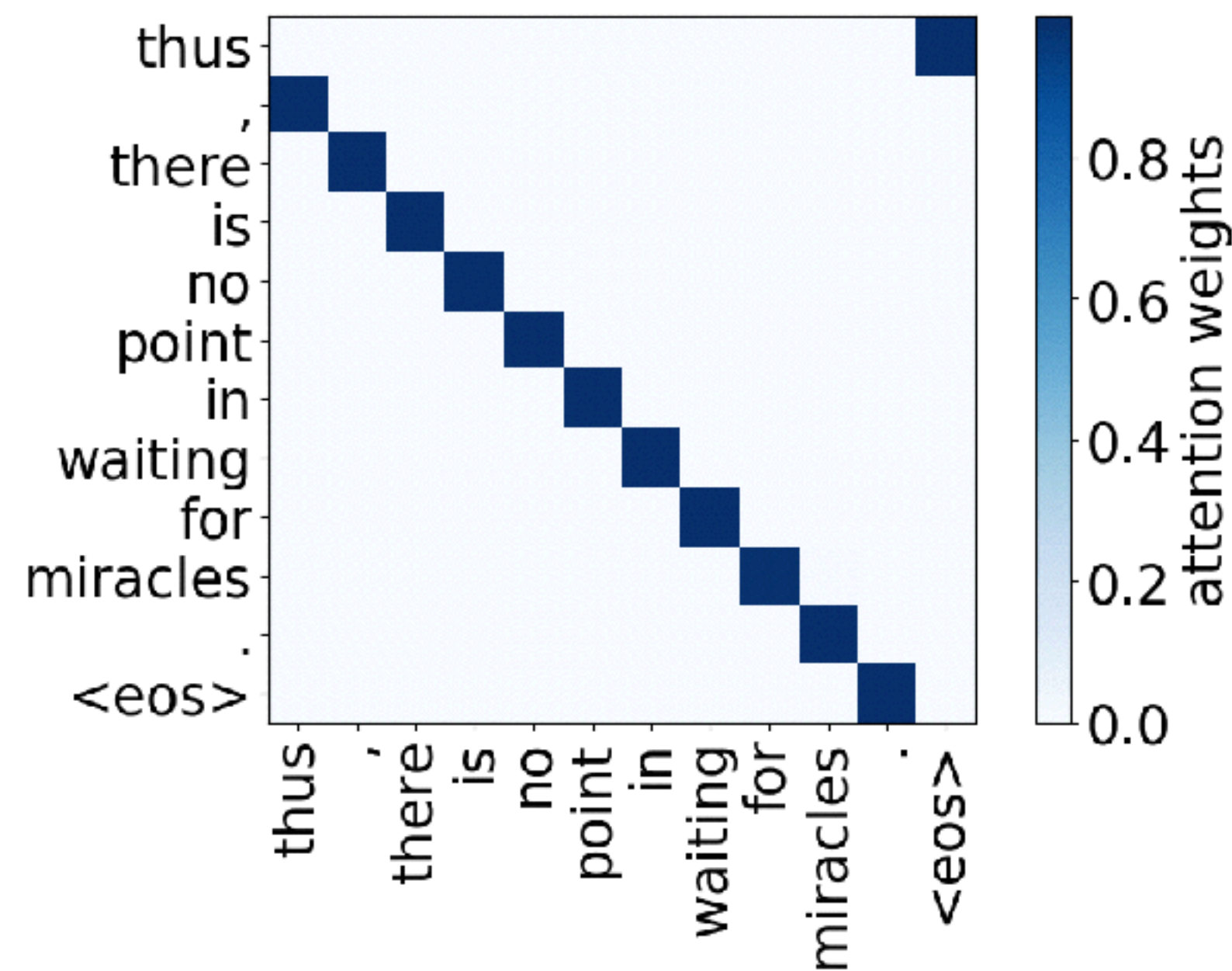
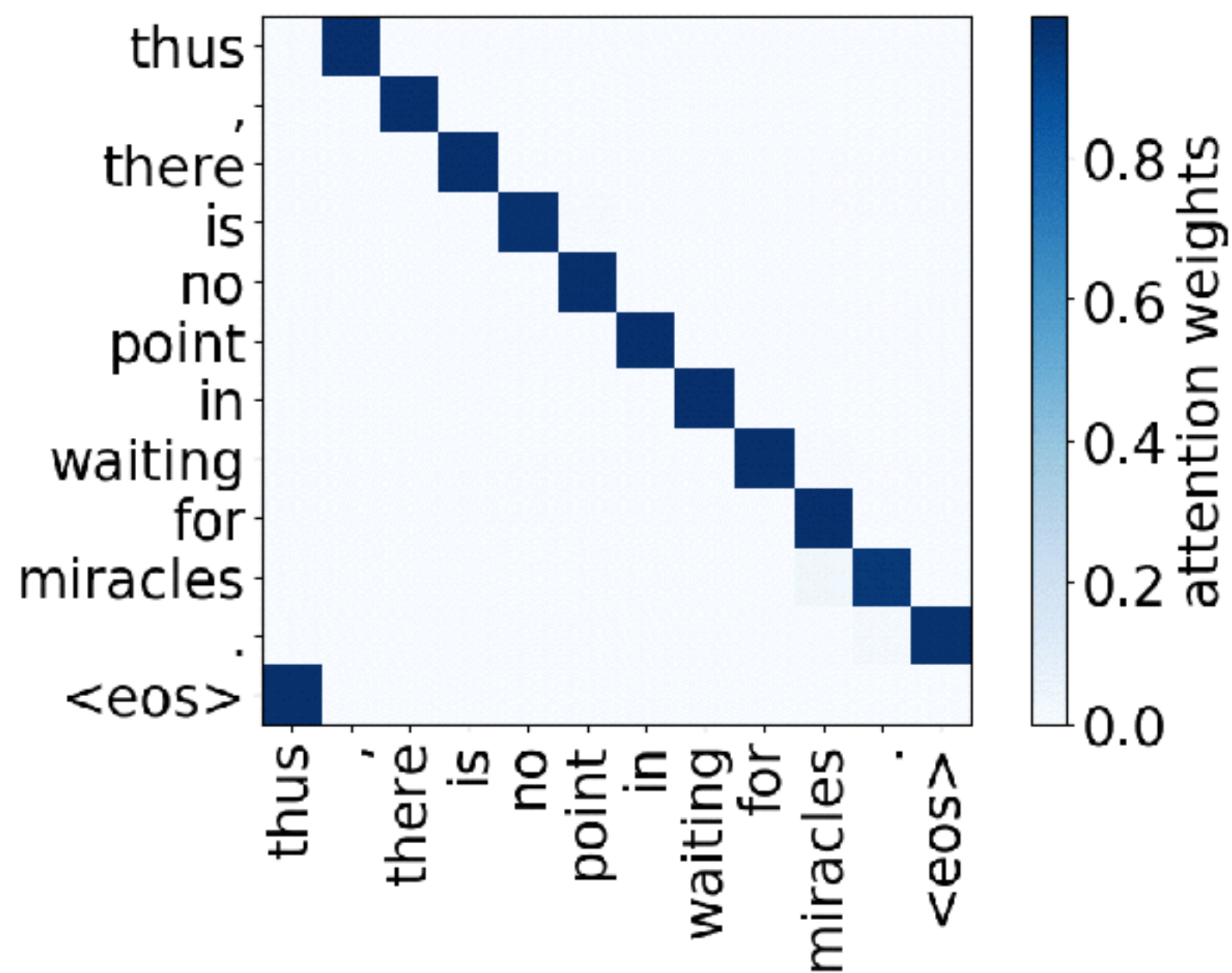
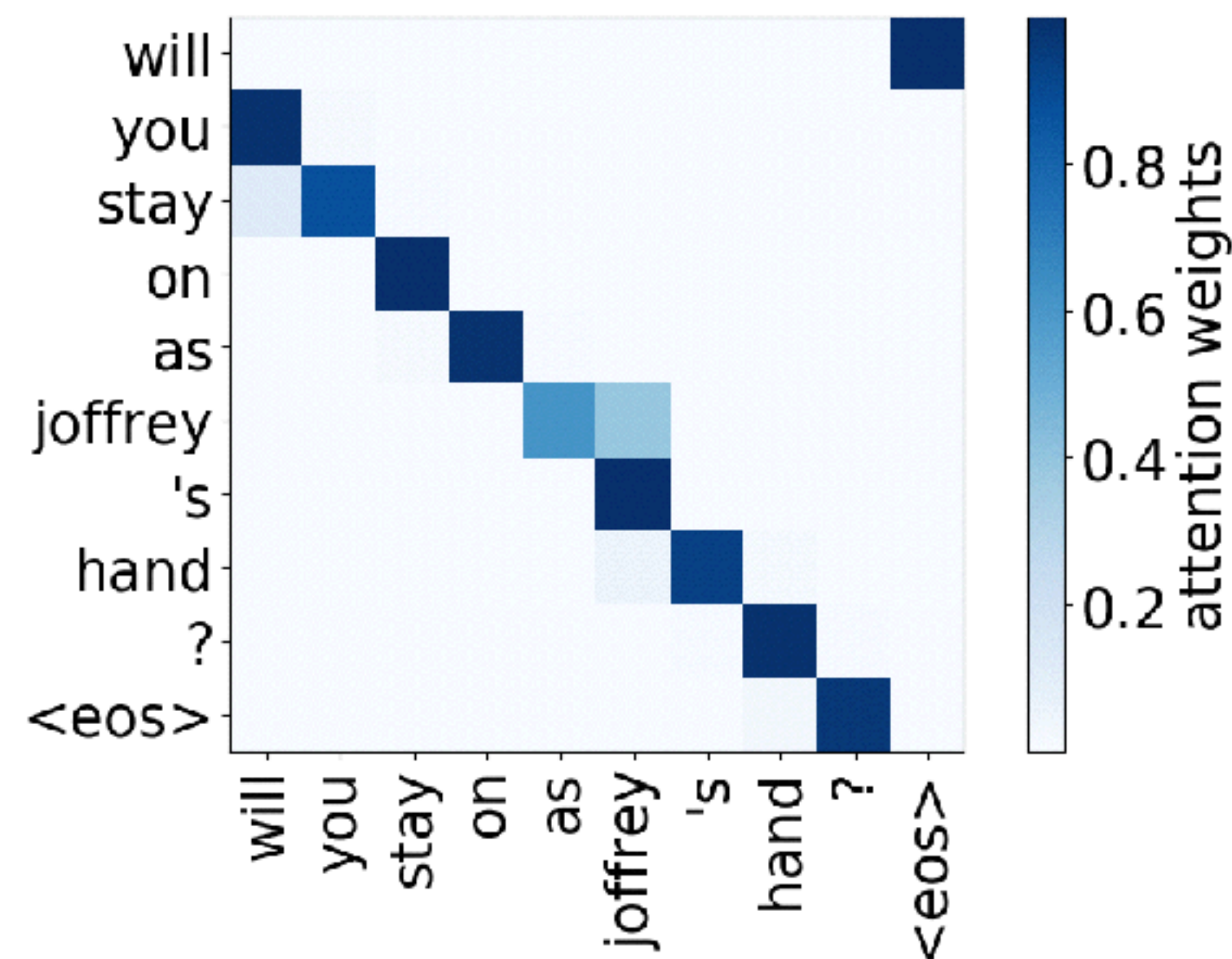
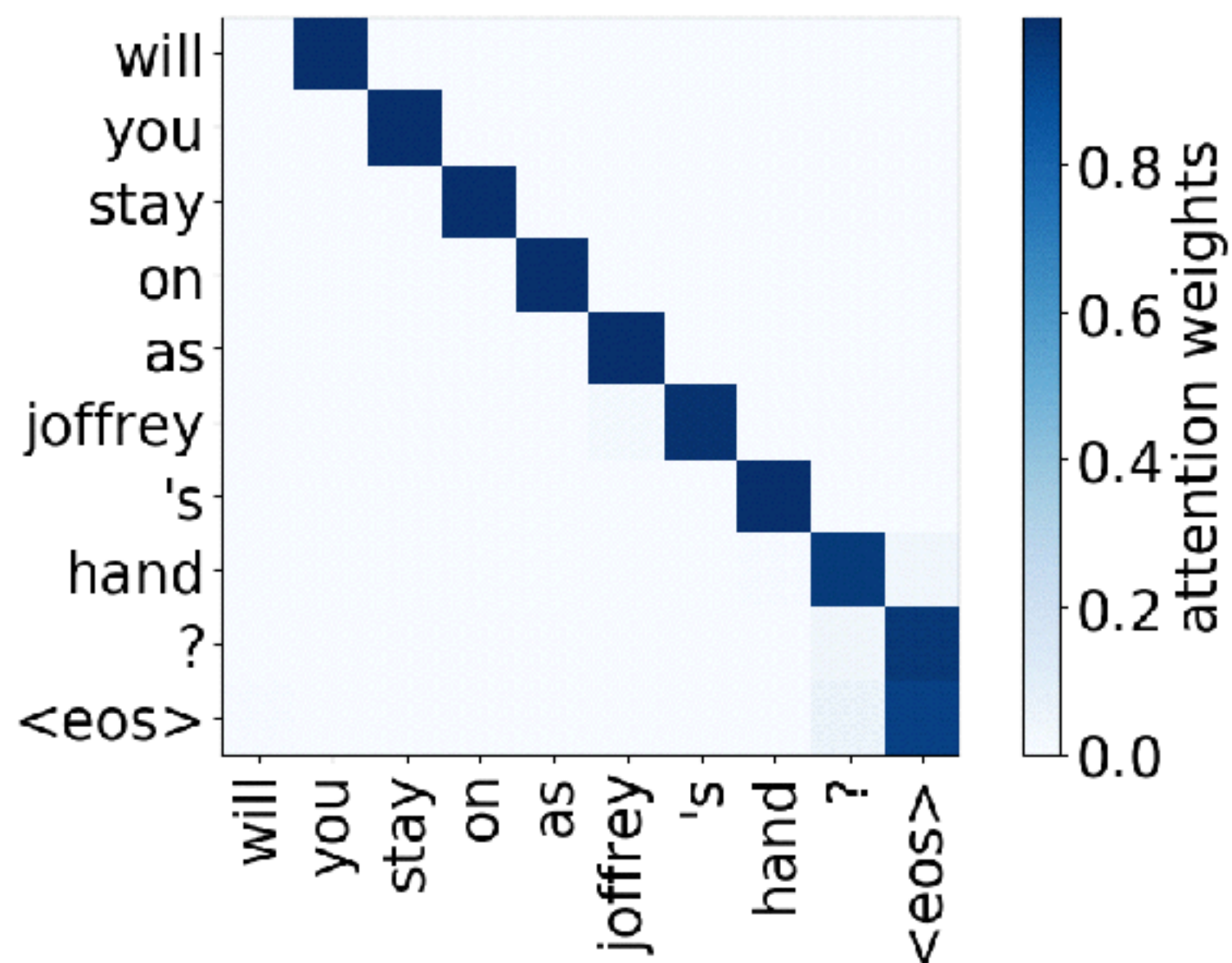




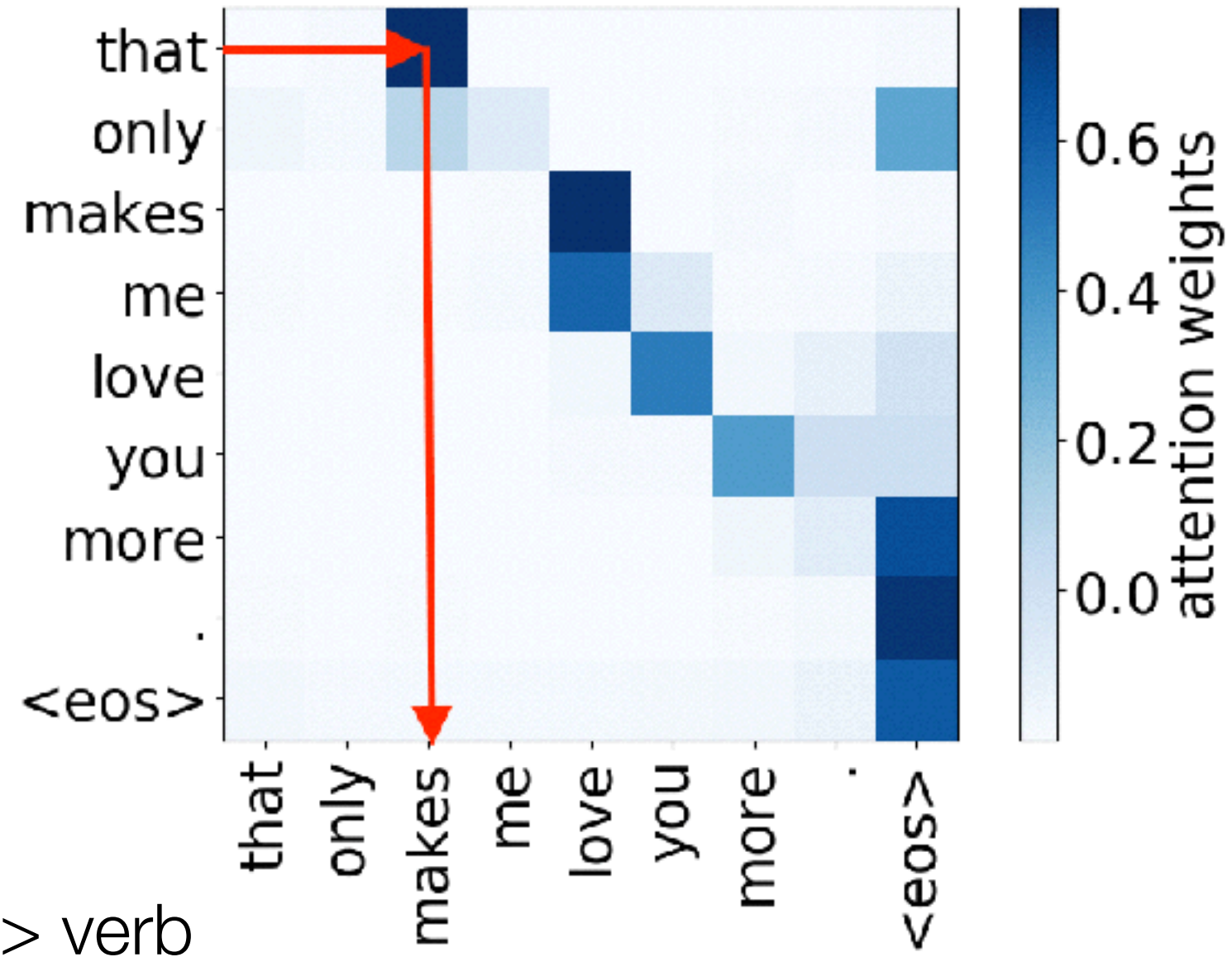
# The Most Important Heads are Interpretable

- Here we'll mention some of the results from the ACL 2019 paper [Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned](#). The authors look at individual attention heads in encoder's multi-head attention and evaluate how much, on average, different heads "contribute" to generated translations (for the details on how exactly they did this, look in the paper or the blog post). As it turns out,
  - only a small number of heads are important for translation,
  - these heads play interpretable "roles".
- These roles are:
  - **positional**: attend to a token's immediate neighbors, and the model has several such heads (usually 2-3 heads looking at the previous token and 2 heads looking at the next token);
  - **syntactic**: learned to track some major syntactic relations in the sentence (subject-verb, verb-object);
  - **rare tokens**: the most important head on the first layer attends to the least frequent tokens in a sentence.

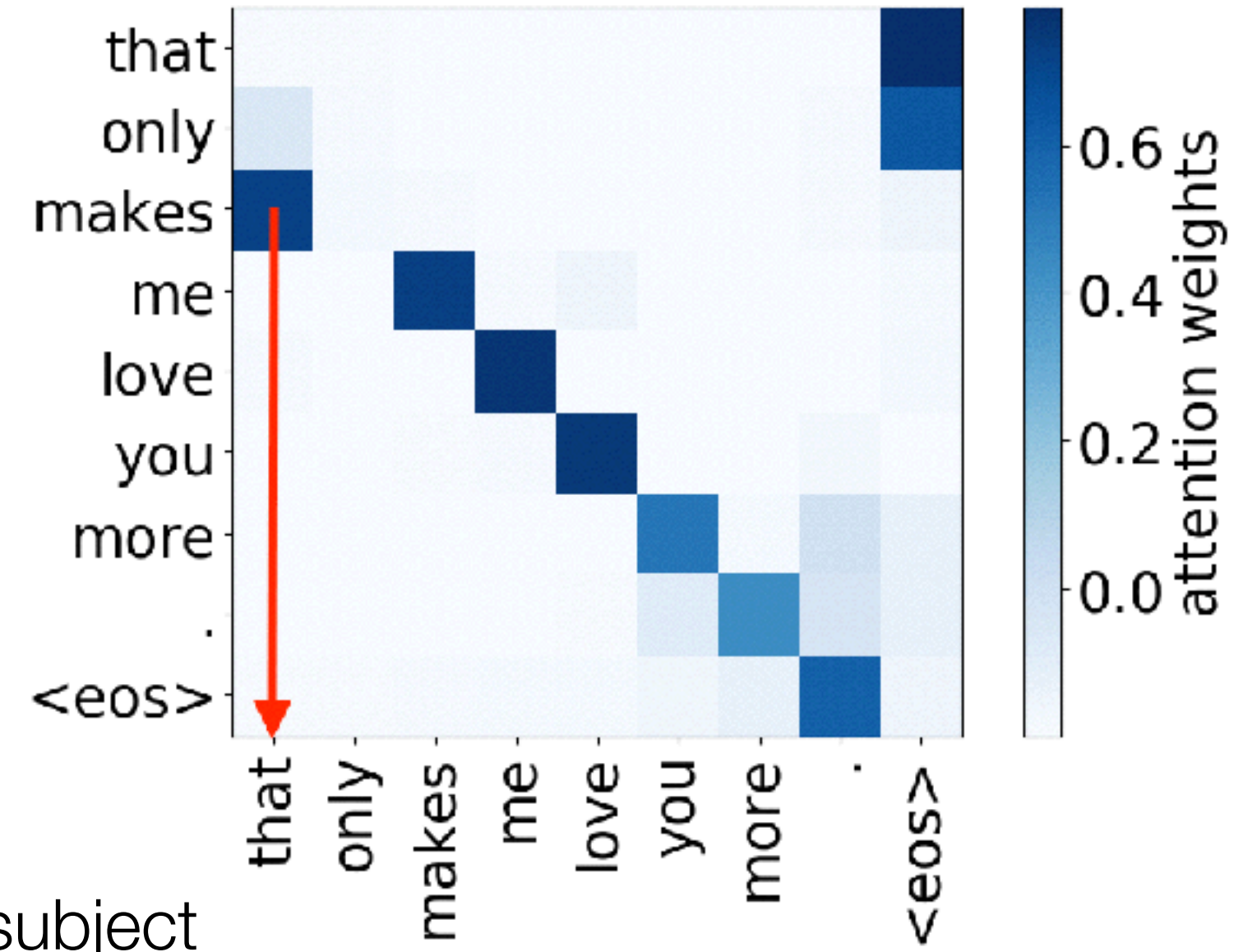
## Positional heads



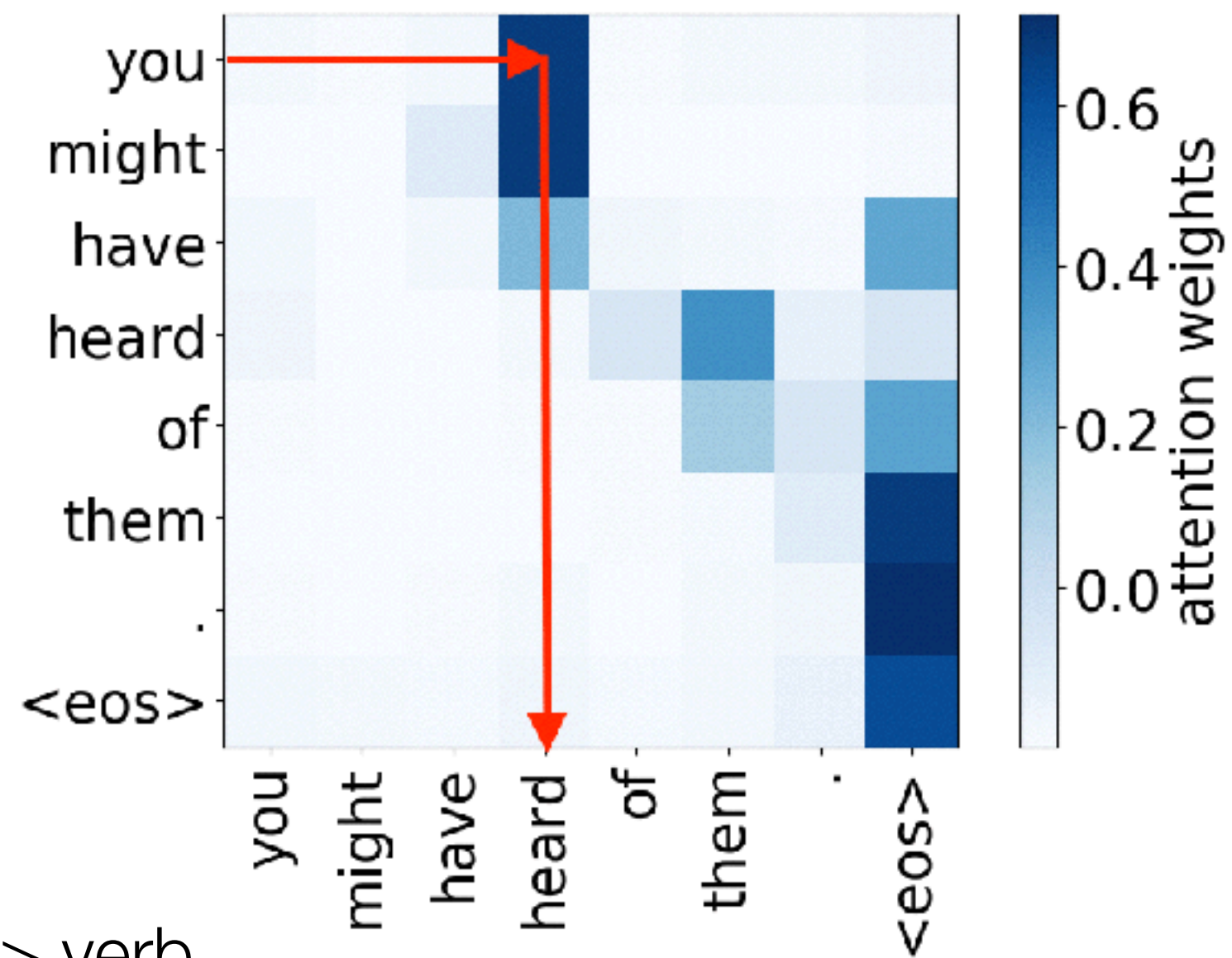
# Syntactic heads



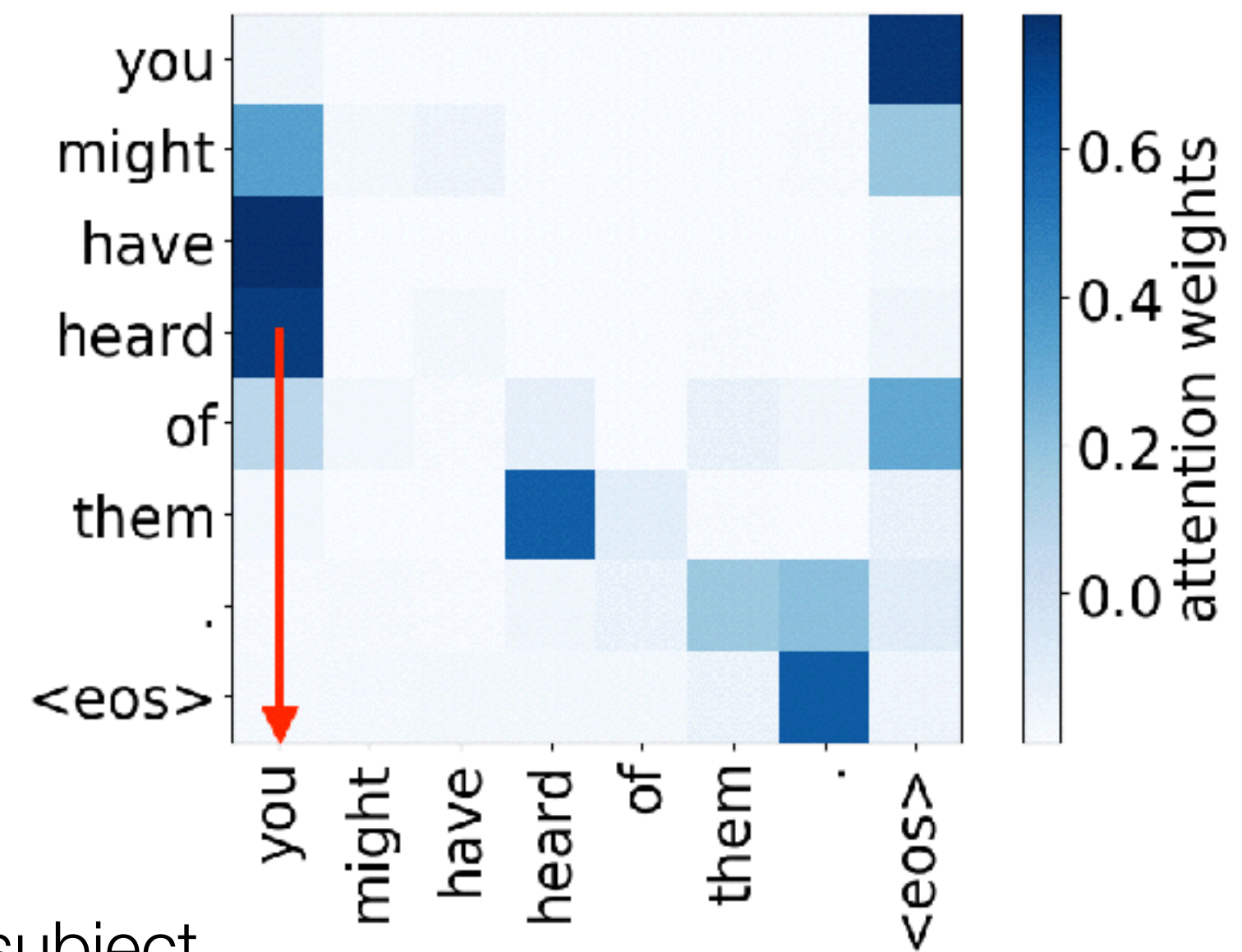
subject -> verb



verb -> subject

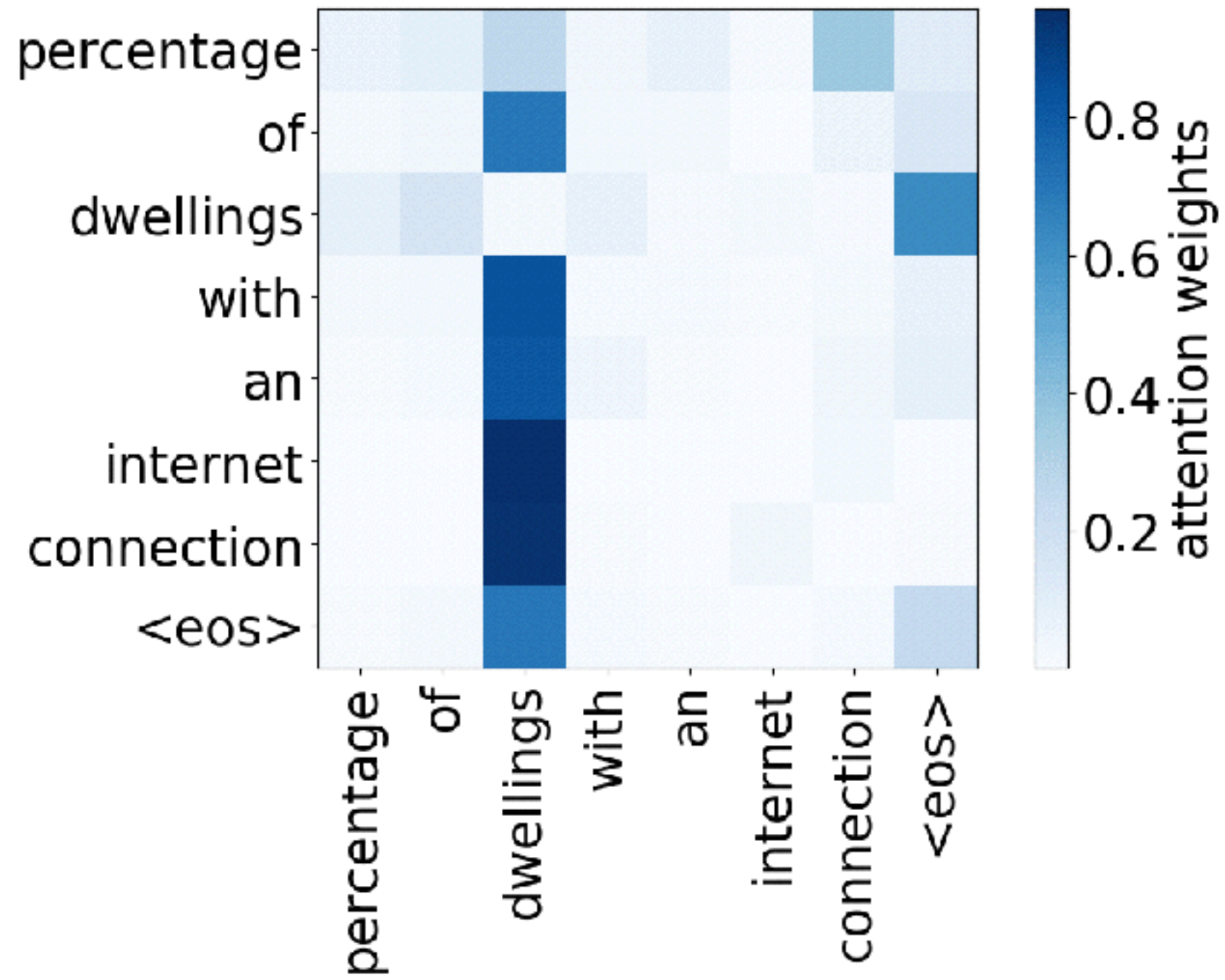
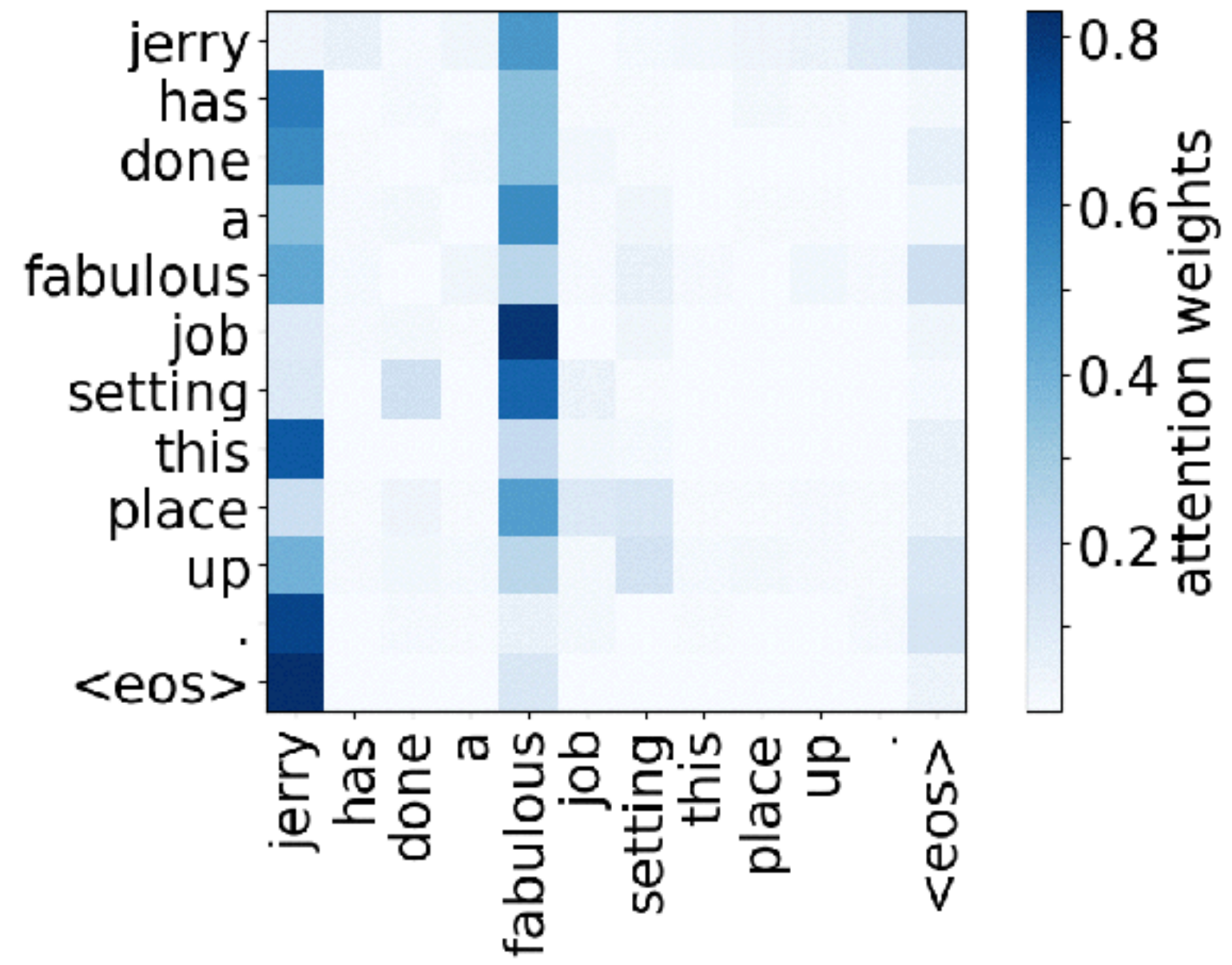
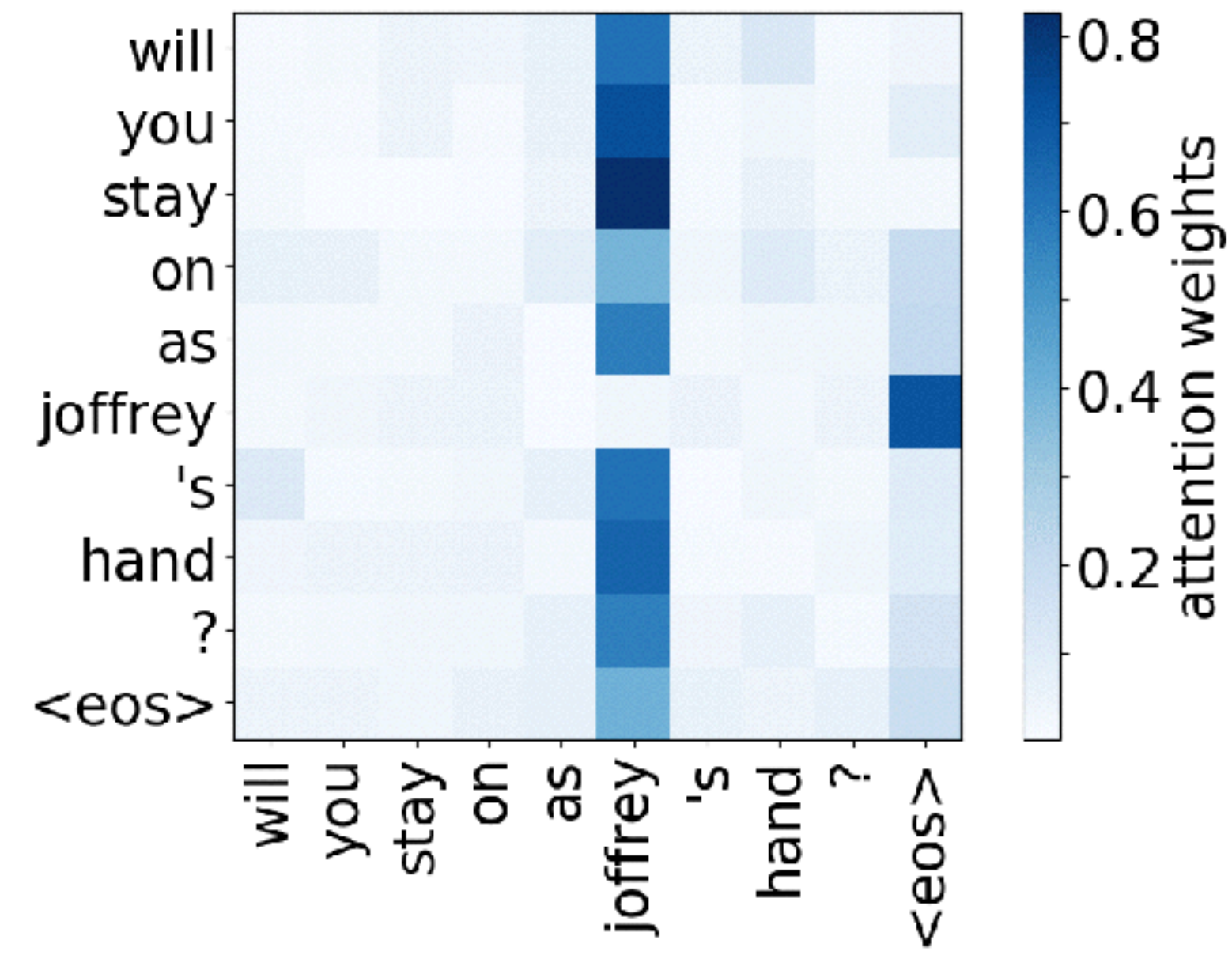
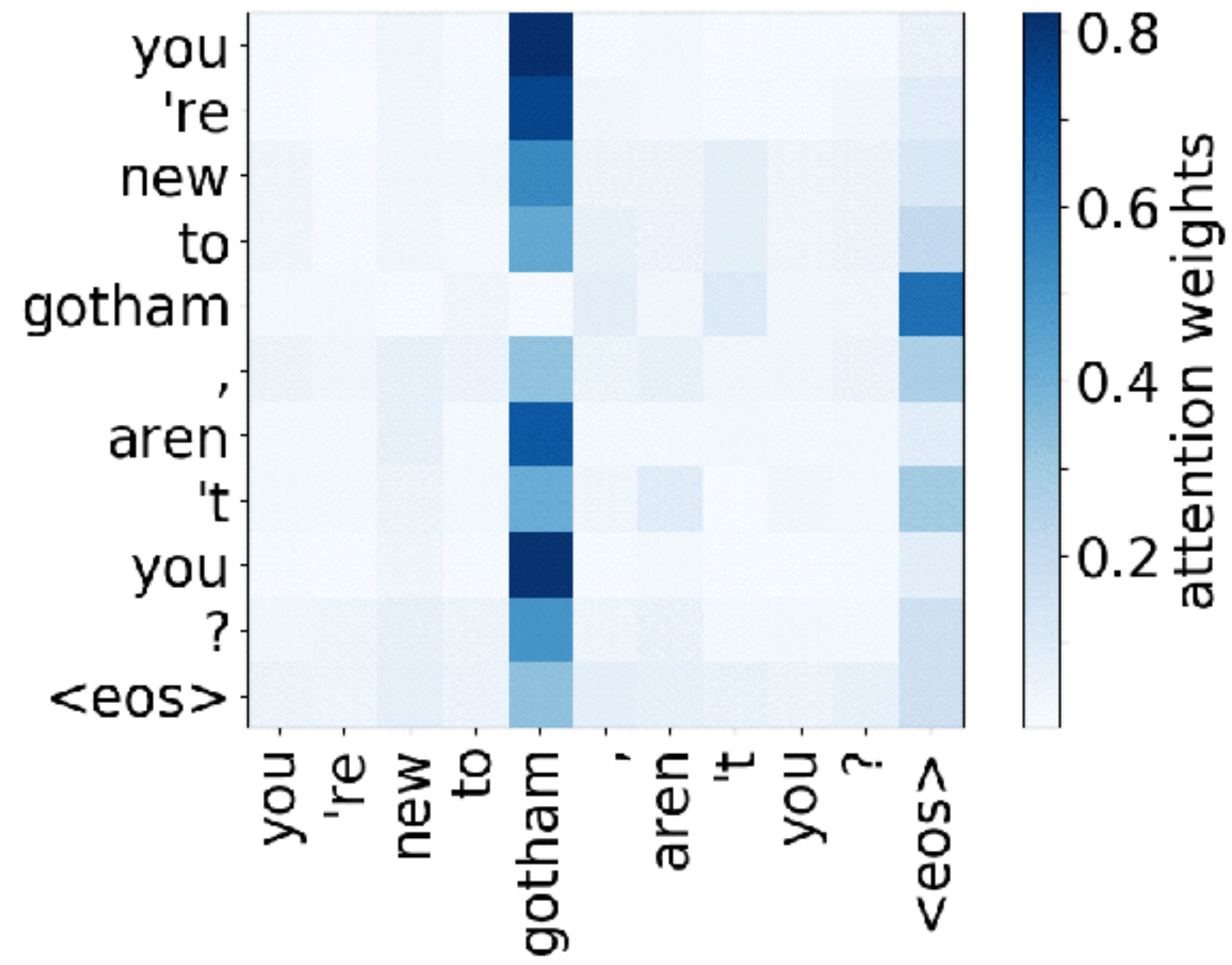


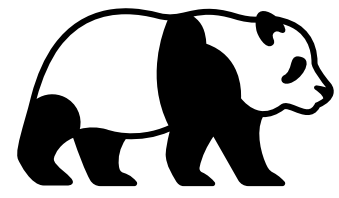
subject -> verb



verb -> subject

# Rare Tokens



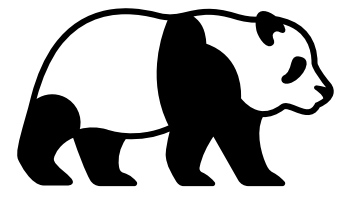


# The Majority of the Heads Can be Pruned

- Later on in the paper, the authors let the model decide which heads it does not need (again, for more details look in the paper or the blog post) and iteratively prunes attention heads, i.e. removes them from the model. In addition to confirming that the specialized heads are the most important (because the model keeps them intact and prunes the other ones), the authors find that **most of the heads can be removed without significant loss in quality.**
- Why don't we train a model with a small number of heads to begin with?
- Well, you can't - the quality will be much lower. You need many heads in training to let them learn all these useful things.

# **Drawbacks and Variants of Transformers**





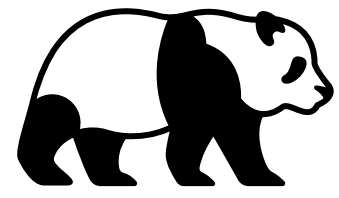
# What Would We Like to Fix about Transformer

- ⦿ **Quadratic compute in self-attention:**

- Computing all pairs of interactions means our computation grows quadratically with the sequence length!
- For recurrent models, it only grew linearly!

- ⦿ **Position representations:**

- Are simple absolute indices the best we can do to represent position?
- Relative linear position attention [[Shaw et al., 2018](#)]
- Dependency syntax-based position [[Wang et al., 2019](#)]



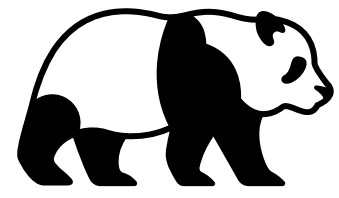
# Quadratic Computation

- Self-attention is highly parallelizable.
- However, its total number of operations grows as  $O(T^2d)$ , where  $T$  is the sequence length, and  $d$  is the dimensionality.
- High complexity for long text.

$$\begin{matrix} \text{XQ} \\ \text{K}^\top \text{X}^\top \end{matrix} = \text{XQK}^\top \text{X}^\top \in \mathbb{R}^{T \times T}$$

Need to compute all pairs of interactions!  
 $O(T^2d)$

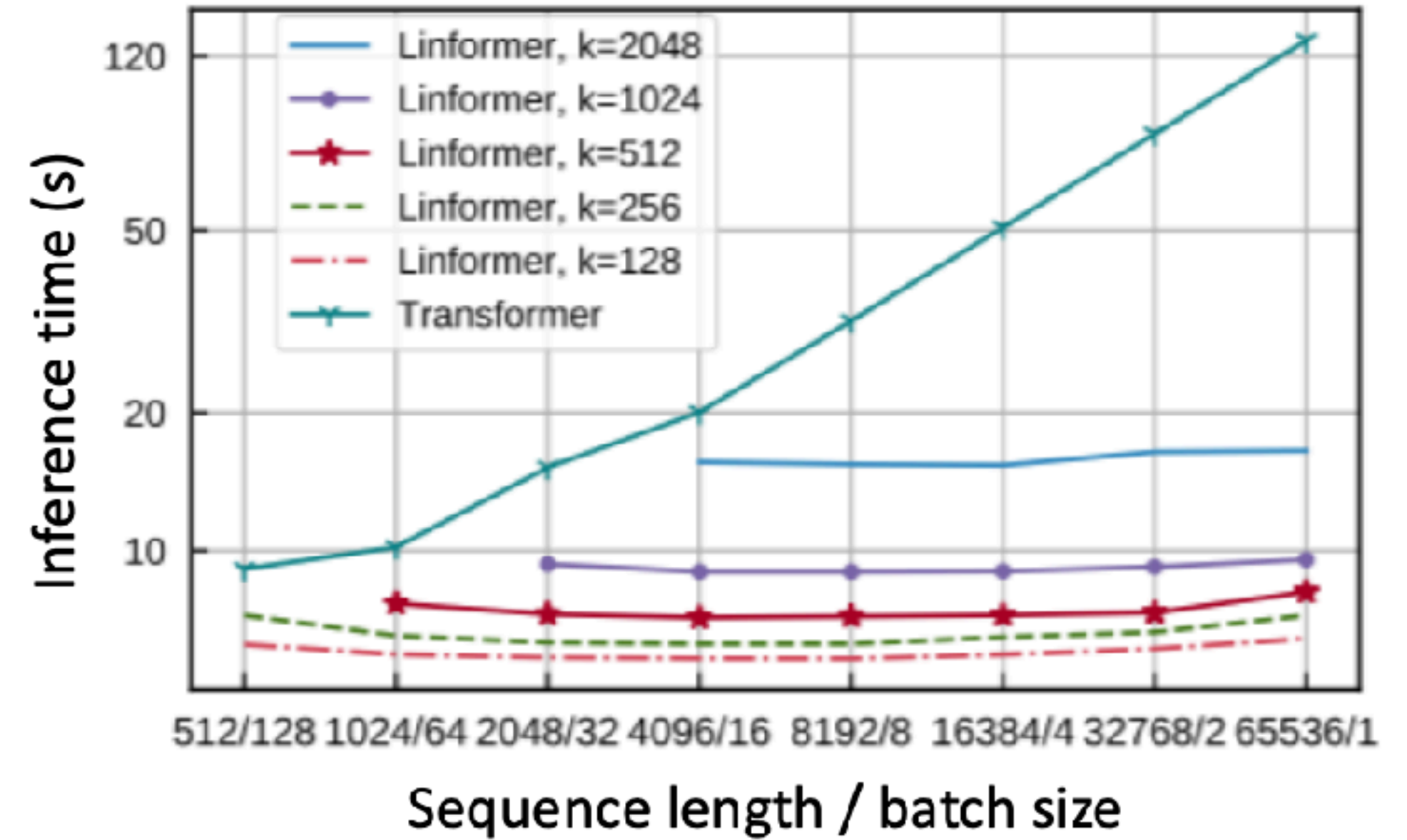
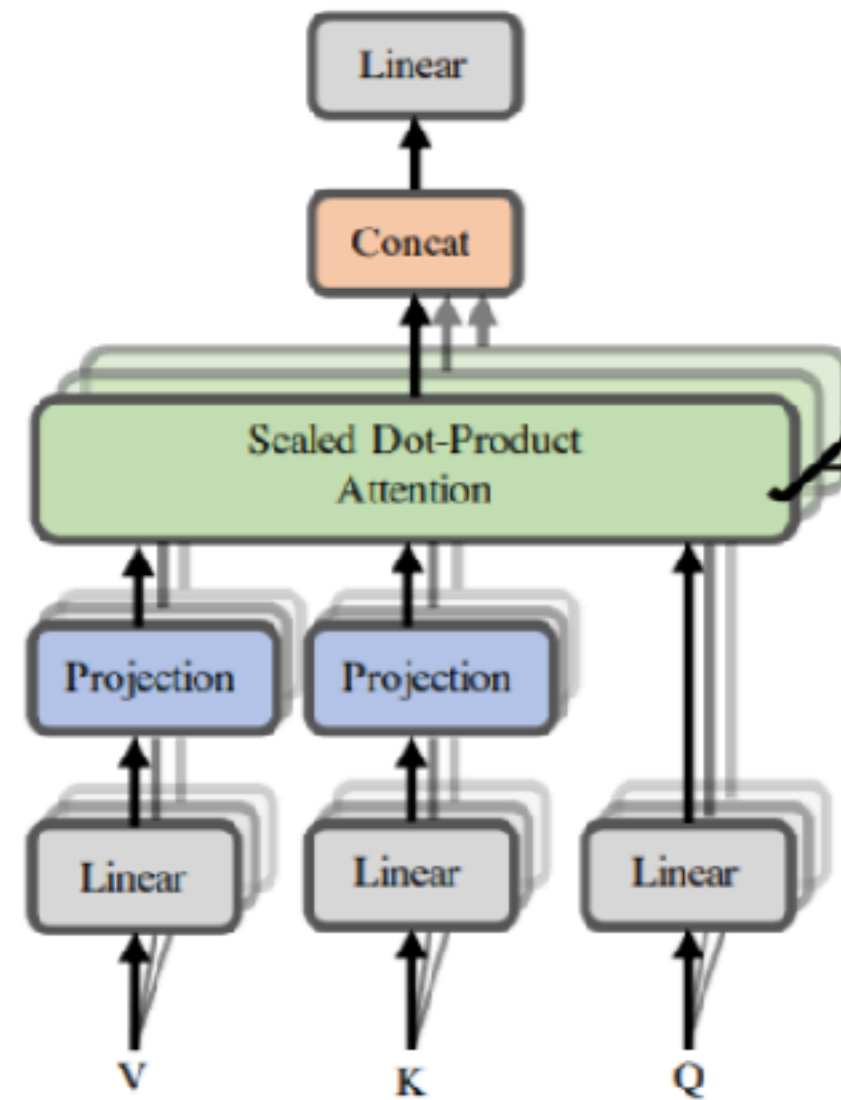
- Can we build models like Transformers without paying the all-pairs self-attention cost?

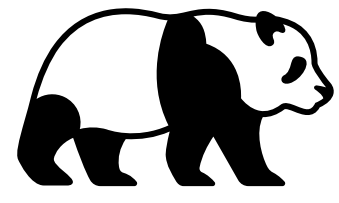


# Linformer

## Linformer: Self-Attention with Linear Complexity

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

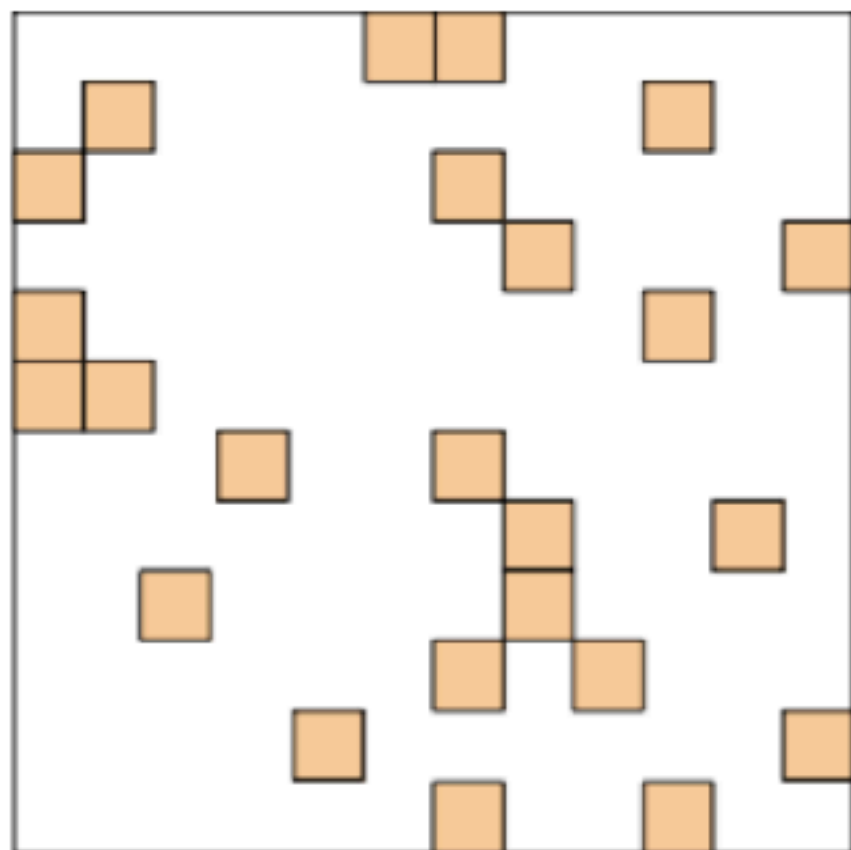




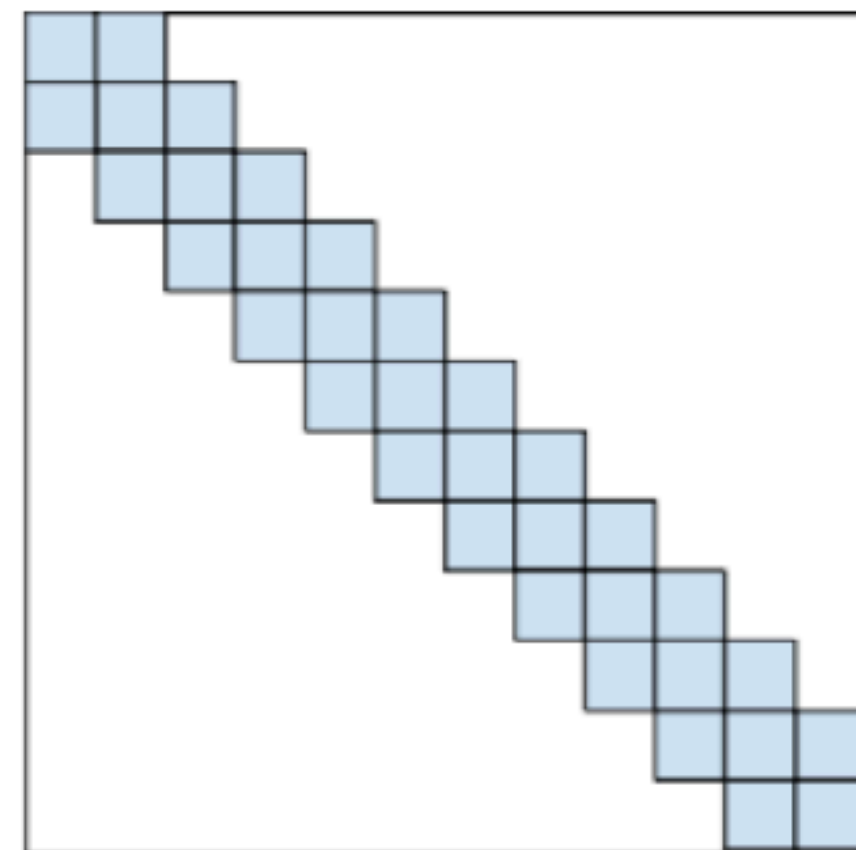
# Big Bird

- Big Bird: Transformers for Longer Sequences

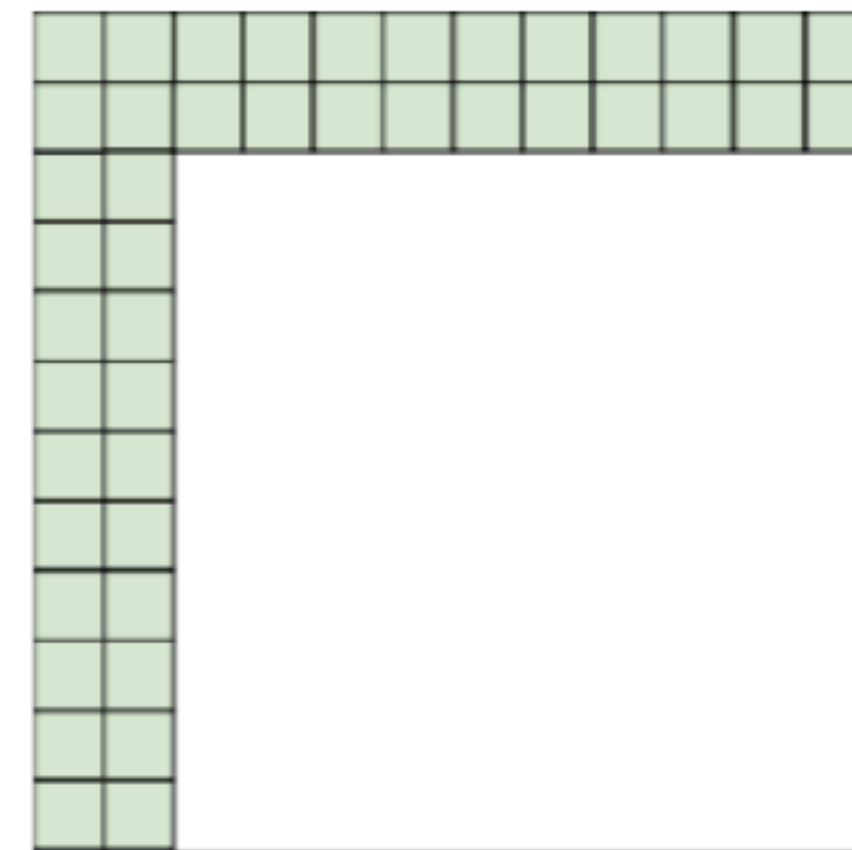
Key idea: replace all-pairs interactions with a family of other interactions, **like local windows, looking at everything, and random interactions.**



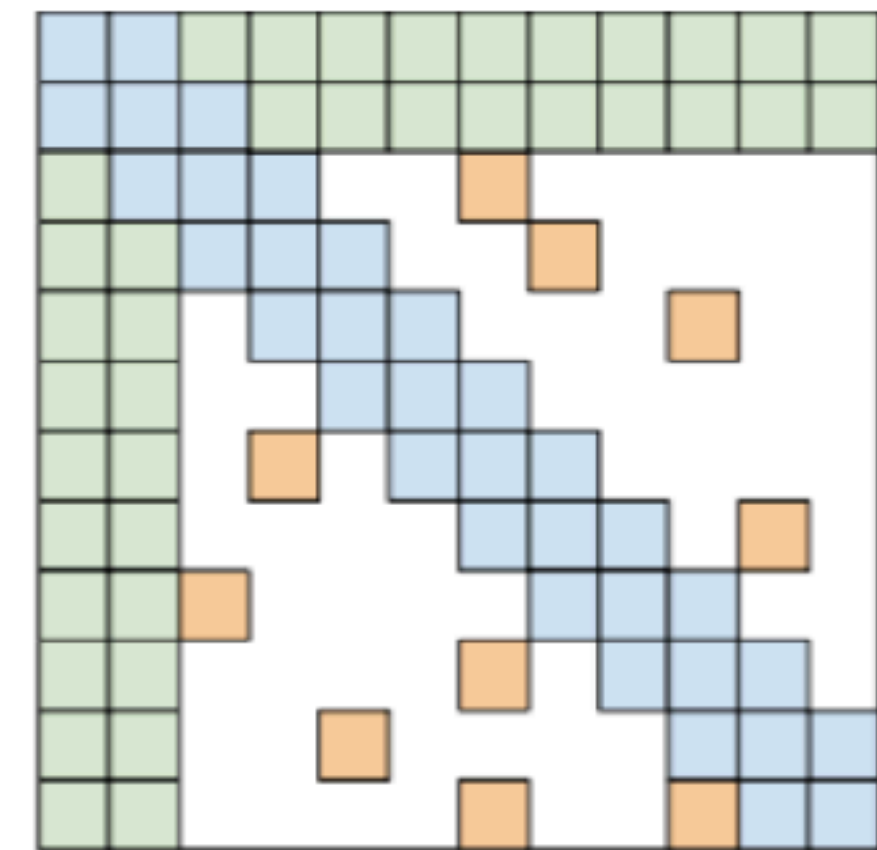
(a) Random attention



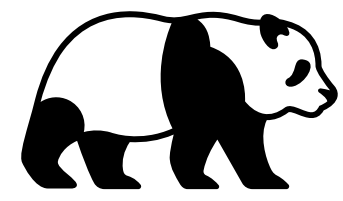
(b) Window attention



(c) Global Attention



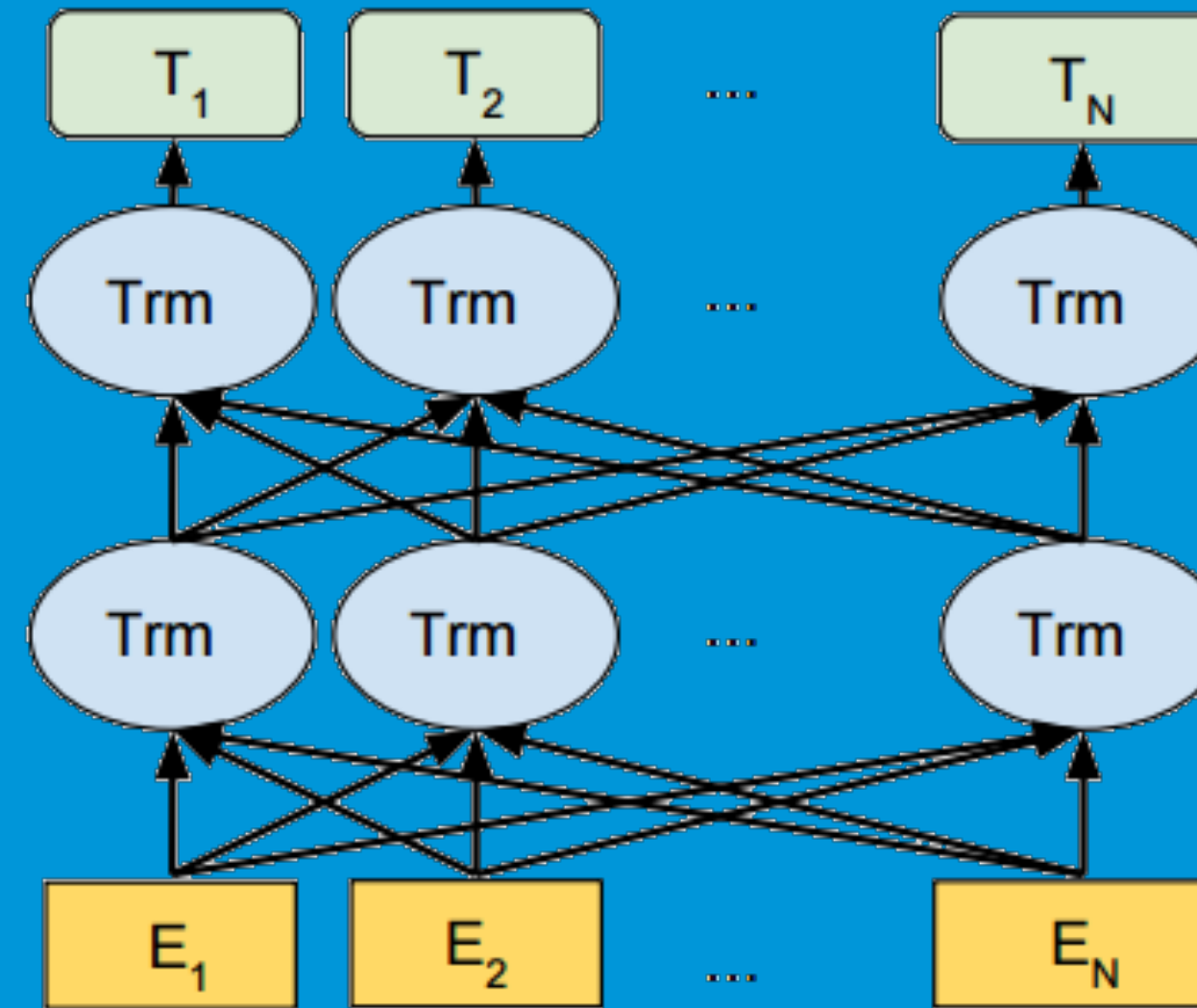
(d) BIGBIRD



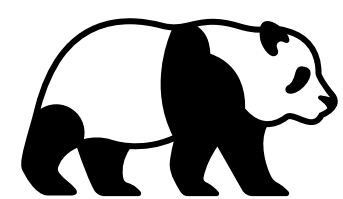
# More Transformer Variants

- (**Survey**) Efficient Transformers: a Survey, 2020
- (**AAAI'21 Best Paper**) Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting

# BERT



# History and Background



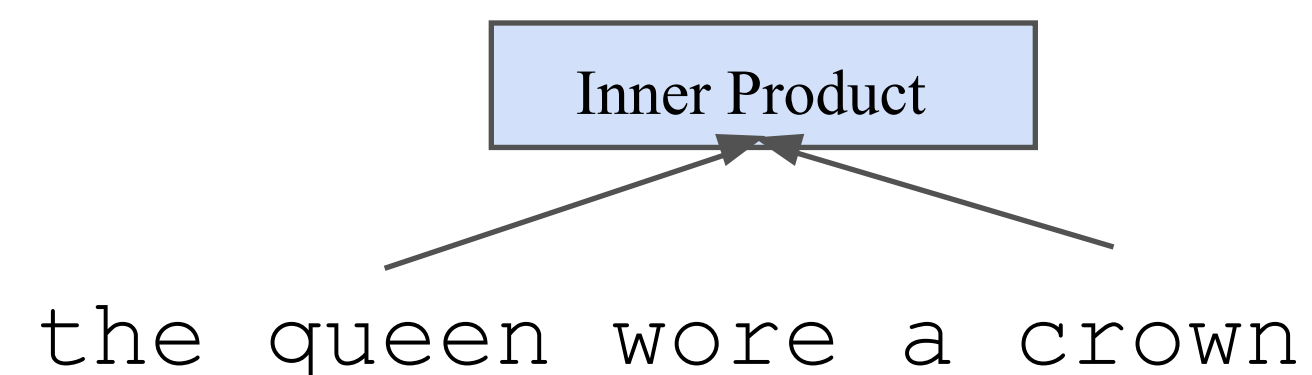
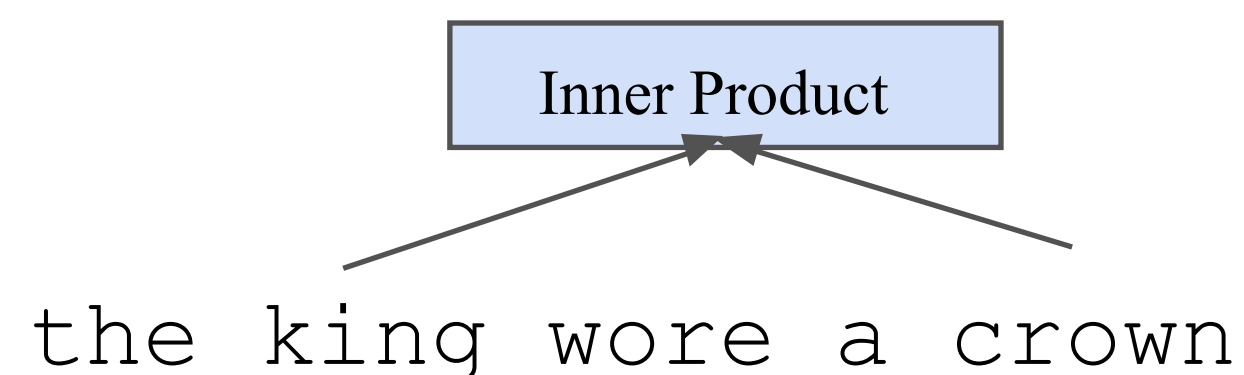
# Pre-training in NLP

- Word embeddings are the basis of deep learning for NLP

king  
↓  
[-0.5, -0.9, 1.4, ...]

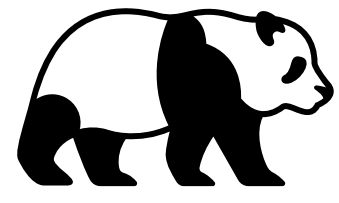
queen  
↓  
[-0.6, -0.8, -0.2, ...]

- Word embeddings (word2vec, GloVe) are often pre-trained on text corpus from co-occurrence statistics



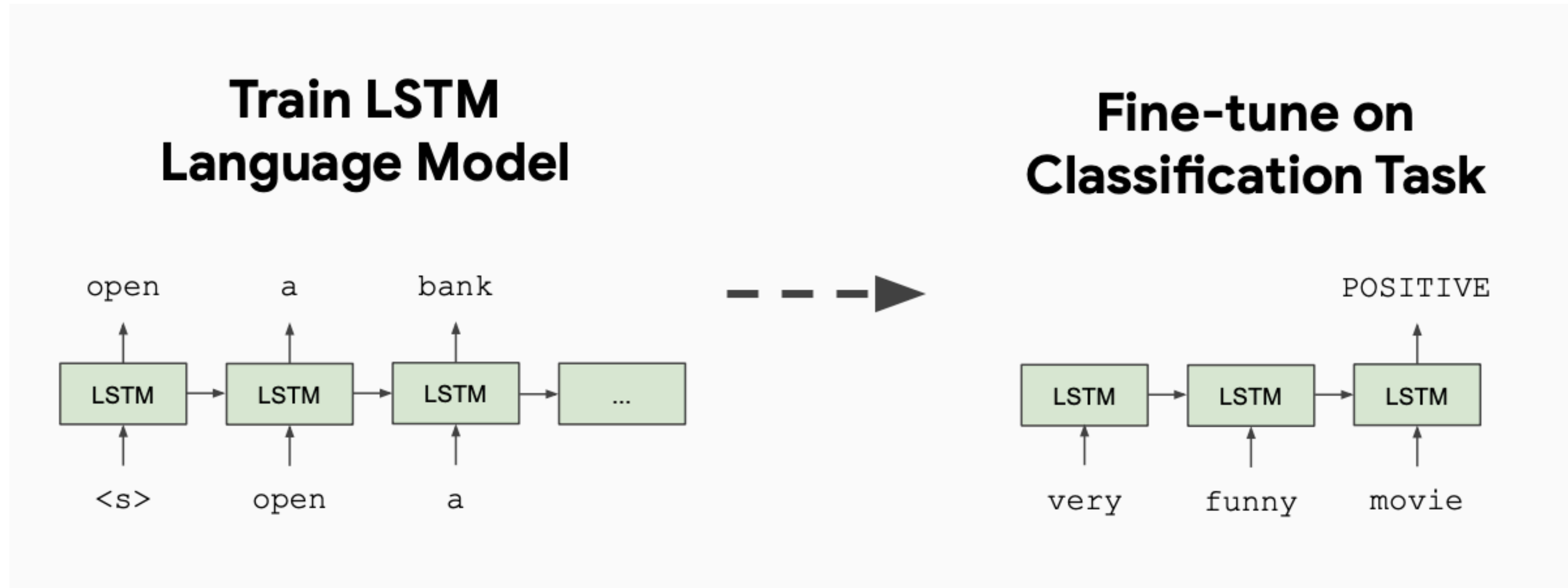


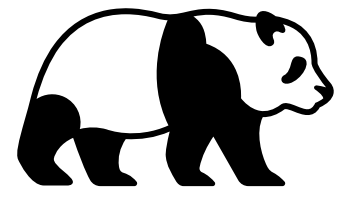




# History of Contextual Representations

- Semi-Supervised Sequence Learning, Google, 2015

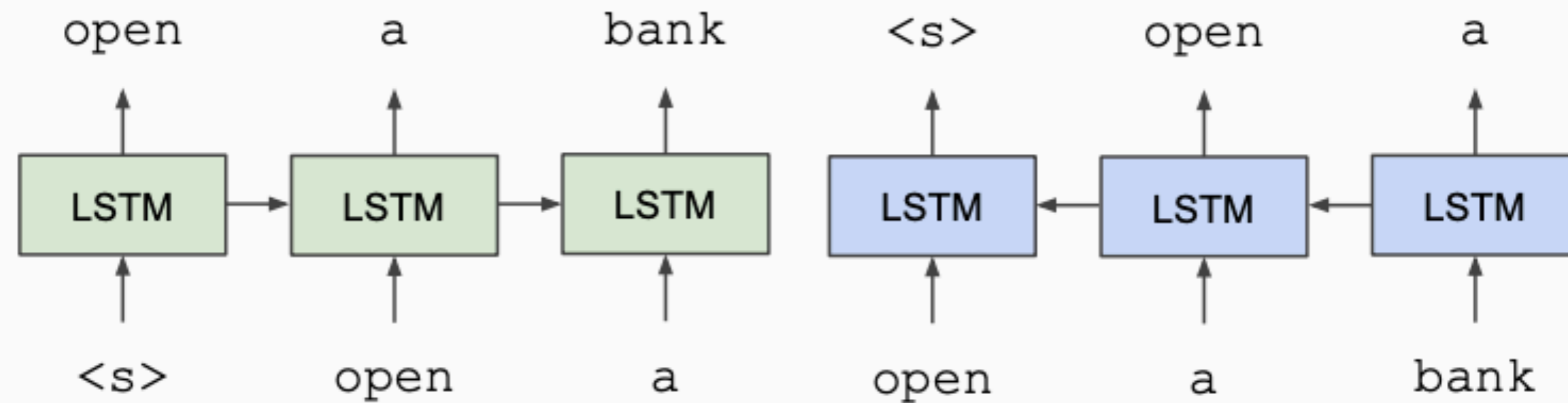




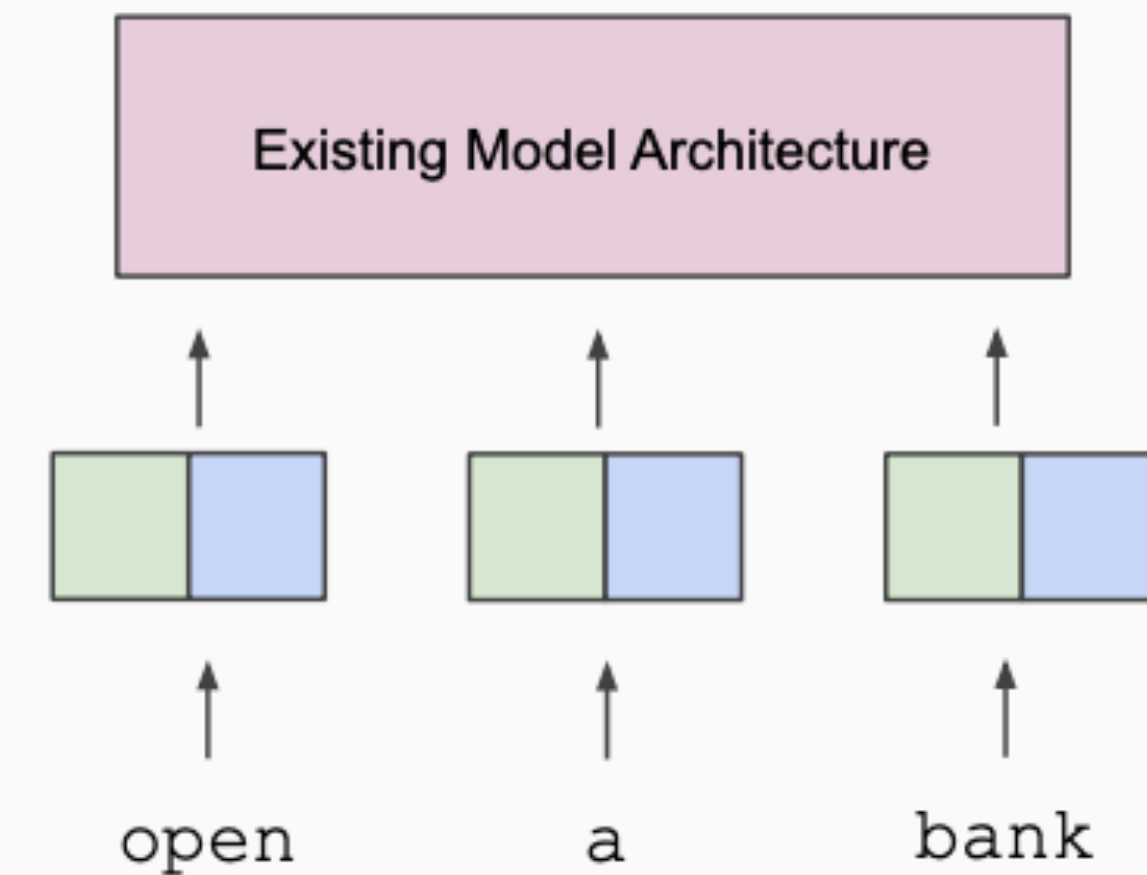
# History of Contextual Representations

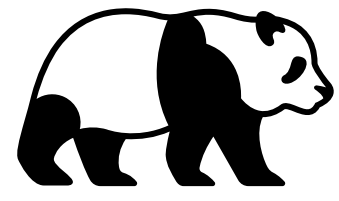
- ELMo: Deep Contextual Word Embeddings, AI2 & University of Washington, 2017

## Train Separate Left-to-Right and Right-to-Left LMs



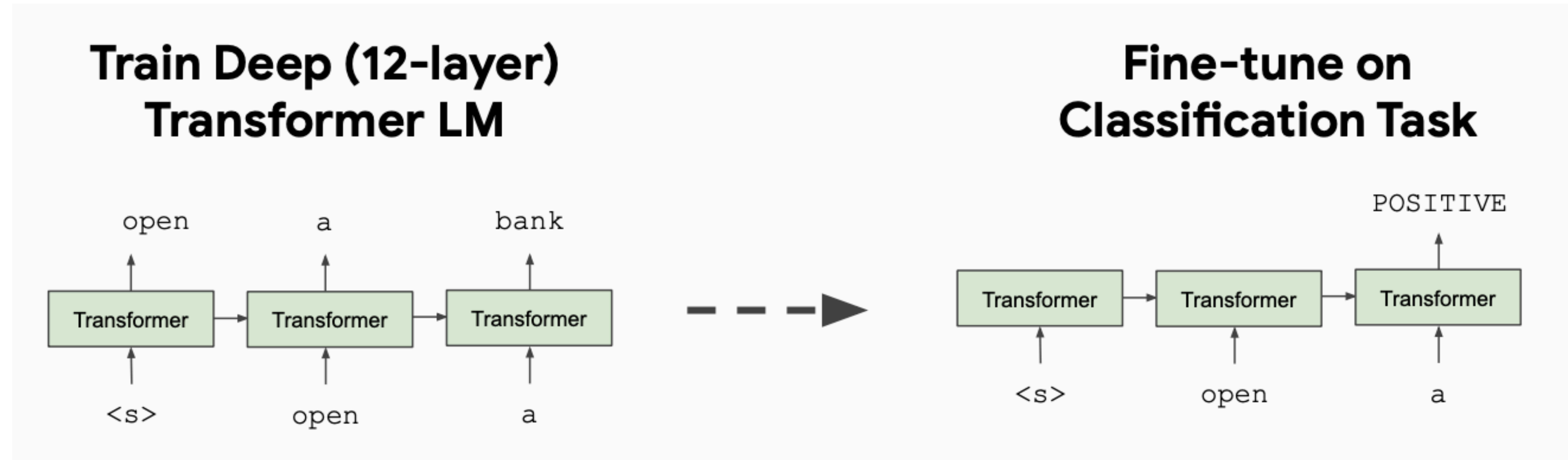
## Apply as “Pre-trained Embeddings”

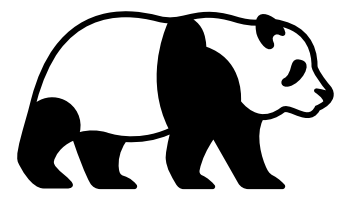




# History of Contextual Representations

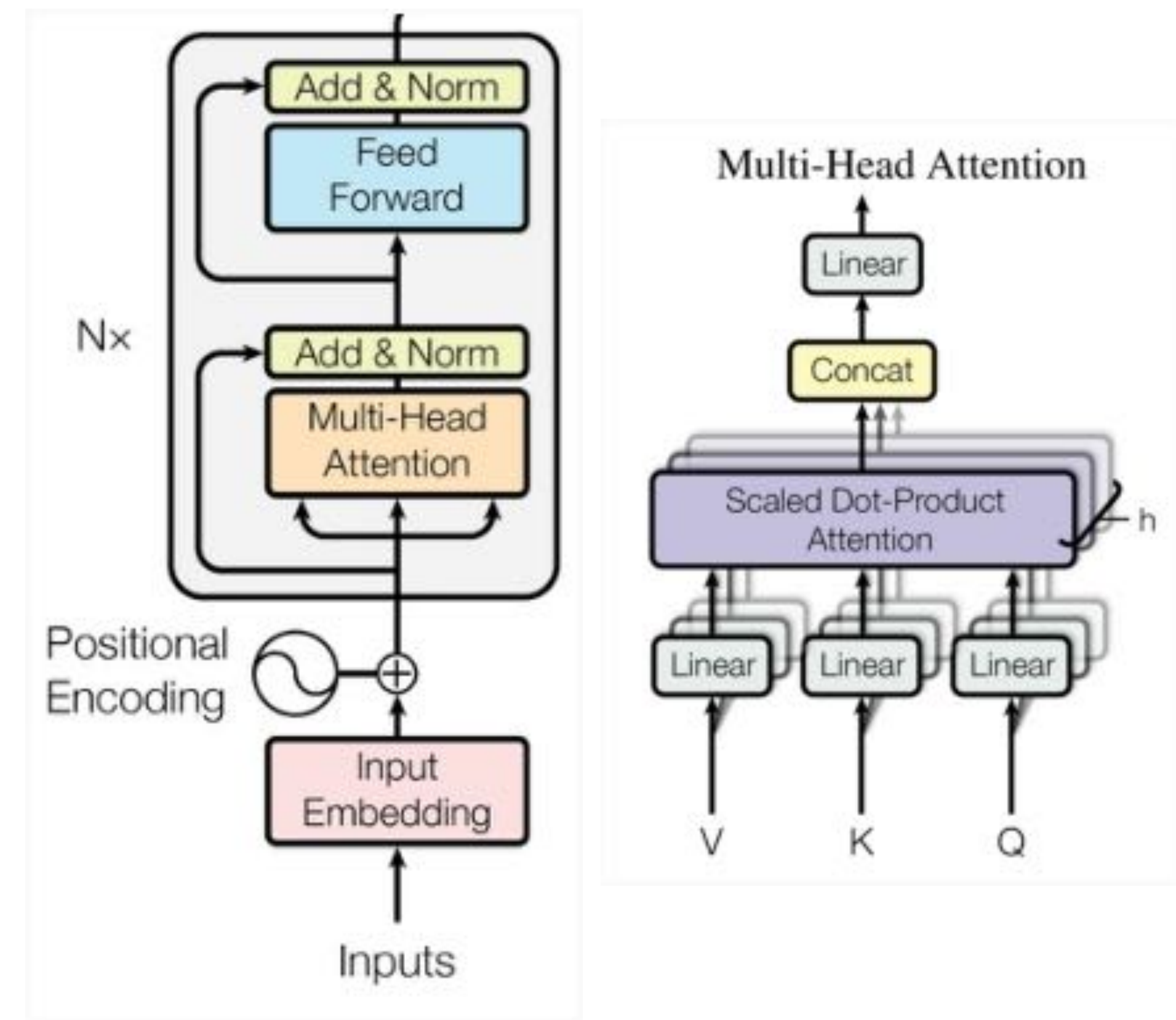
- Improving Language Understanding by Generative Pre-Training, OpenAI, 2018 (GPT)

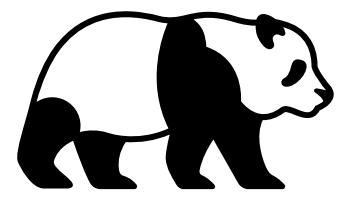




# Model Architecture: Transformer Encoder

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning

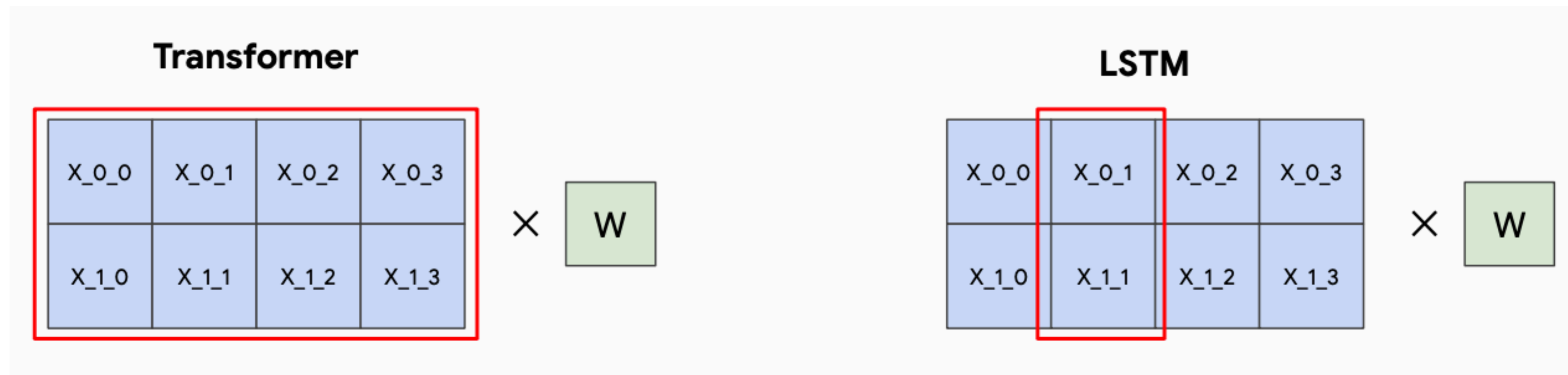




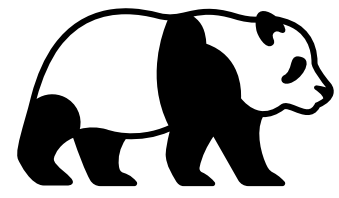
# Model Architecture: Transformer Encoder

## Empirical advantages of Transformer vs. LSTM:

- Self-attention == no locality bias
  - Long-distance context has “equal opportunity”
- Single multiplication per layer == efficiency on GPU/TPU
  - Effective batch size is number of words, not sequences



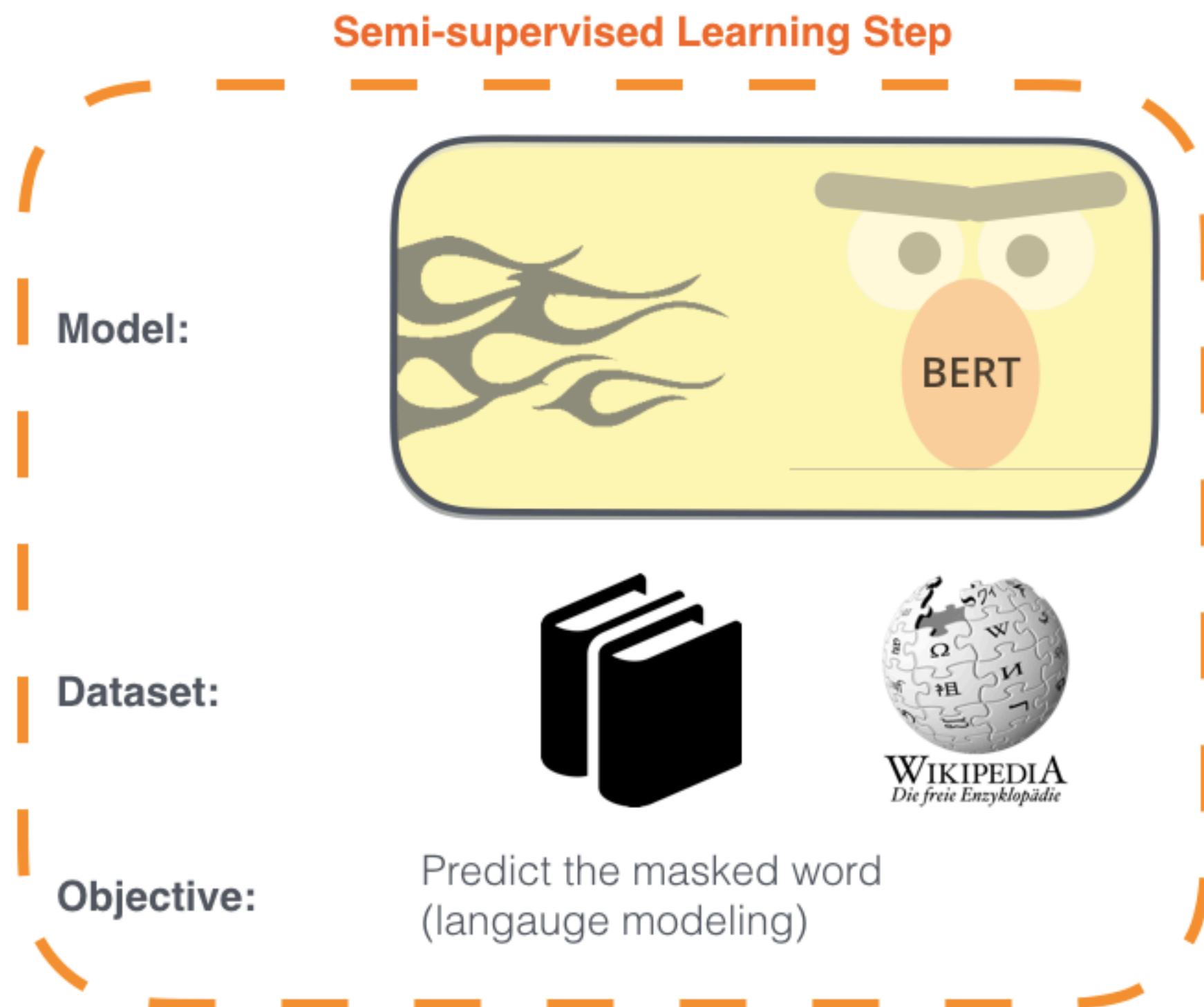
**BERT:**  
**B**idirectional **E**ncoder  
**R**epresentations from  
**T**ransformers



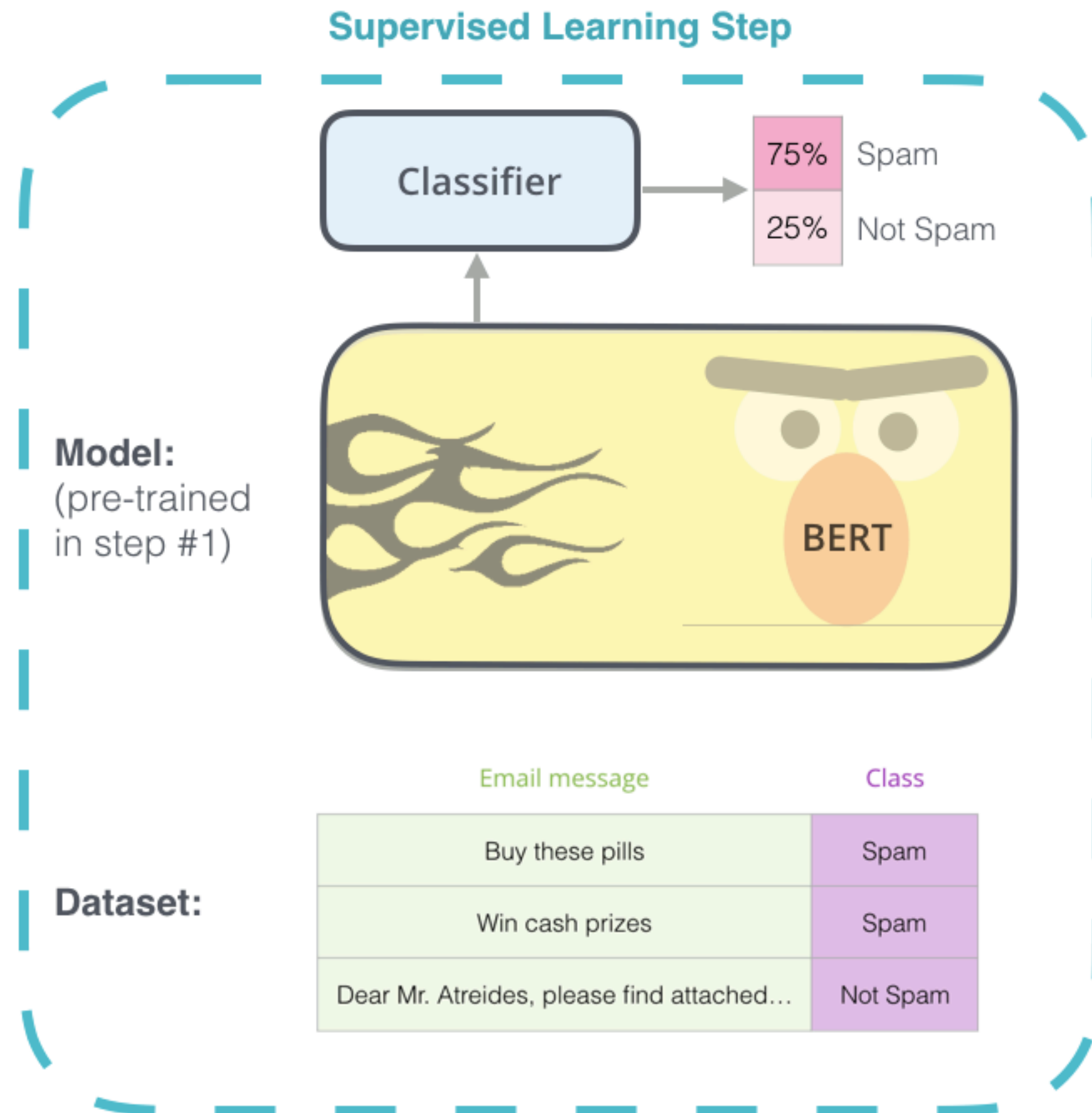
# BERT Overview

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

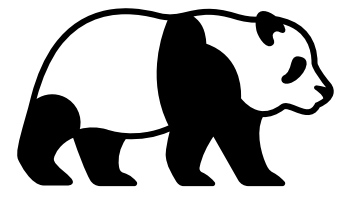


2 - **Supervised** training on a specific task with a labeled dataset.



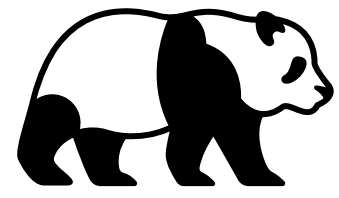
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [[Source](#) for book icon].





# Problem with Previous Methods

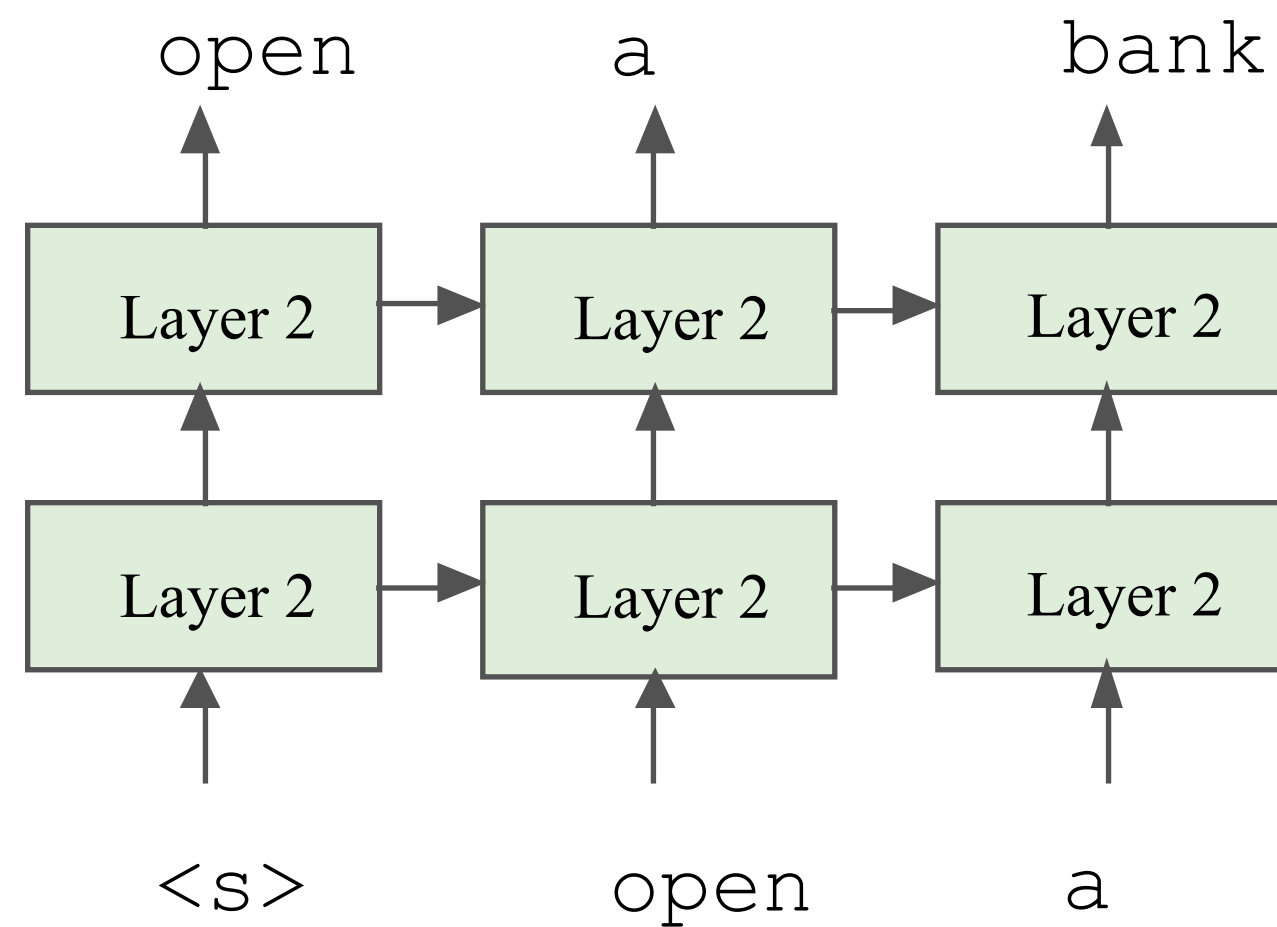
- **Problem:** Language models only use left context or right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
  - Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - Reason 2: Words can “see themselves” in a bidirectional encoder.



# Unidirectional vs. Bidirectional Models

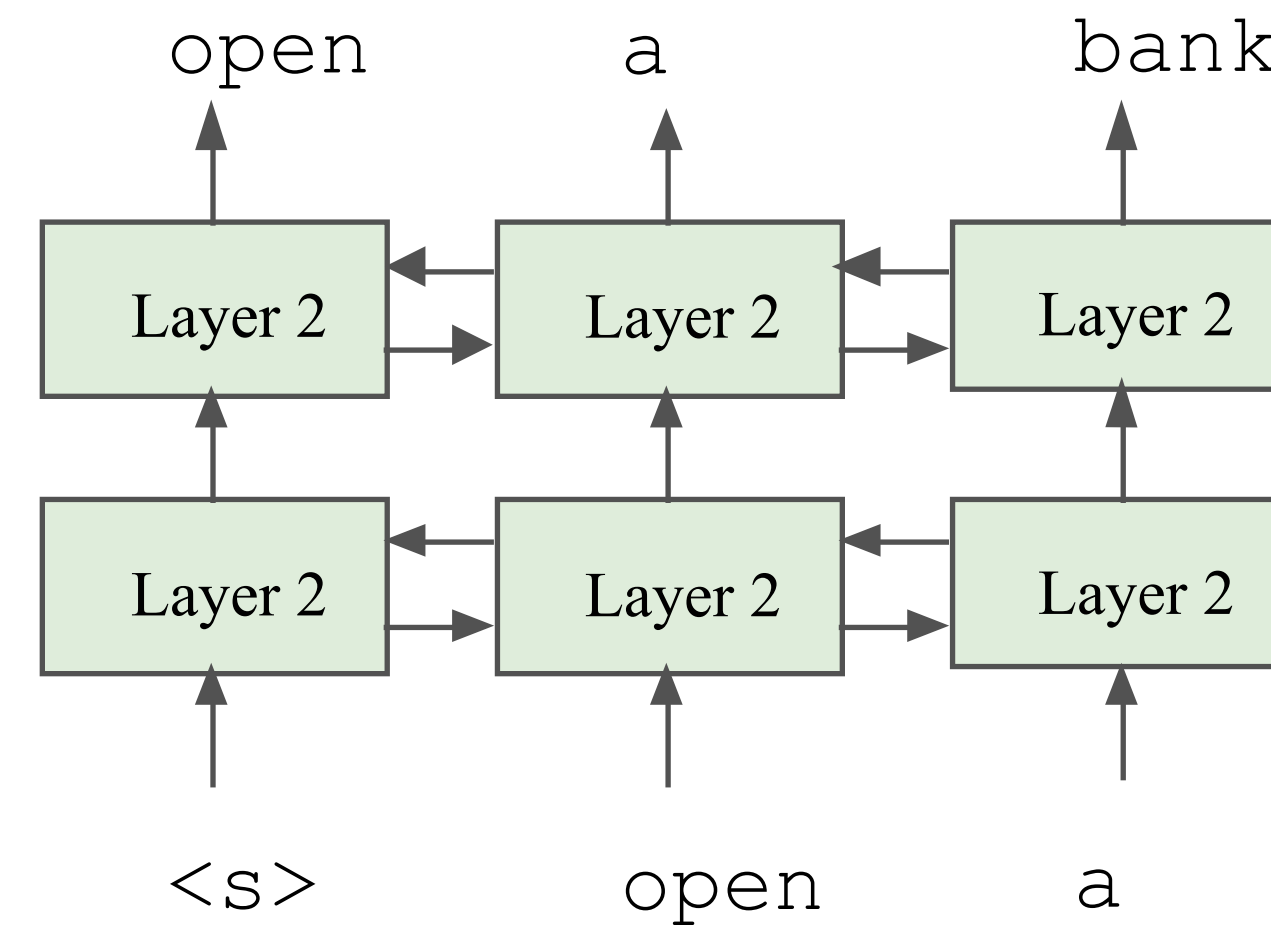
## Unidirectional context

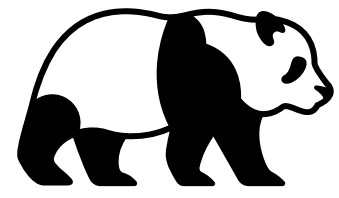
Build representation incrementally



## Bidirectional context

Words can “see themselves”





# Masked LM

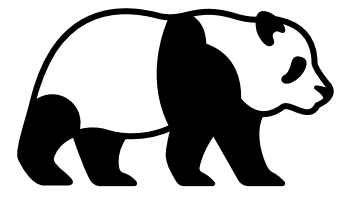
- **Solution:** Mask out  $k\%$  of the input words, and then predict the masked words (use 15%)

the man went to the [MASK] to buy a [MASK] of milk

store                      gallon

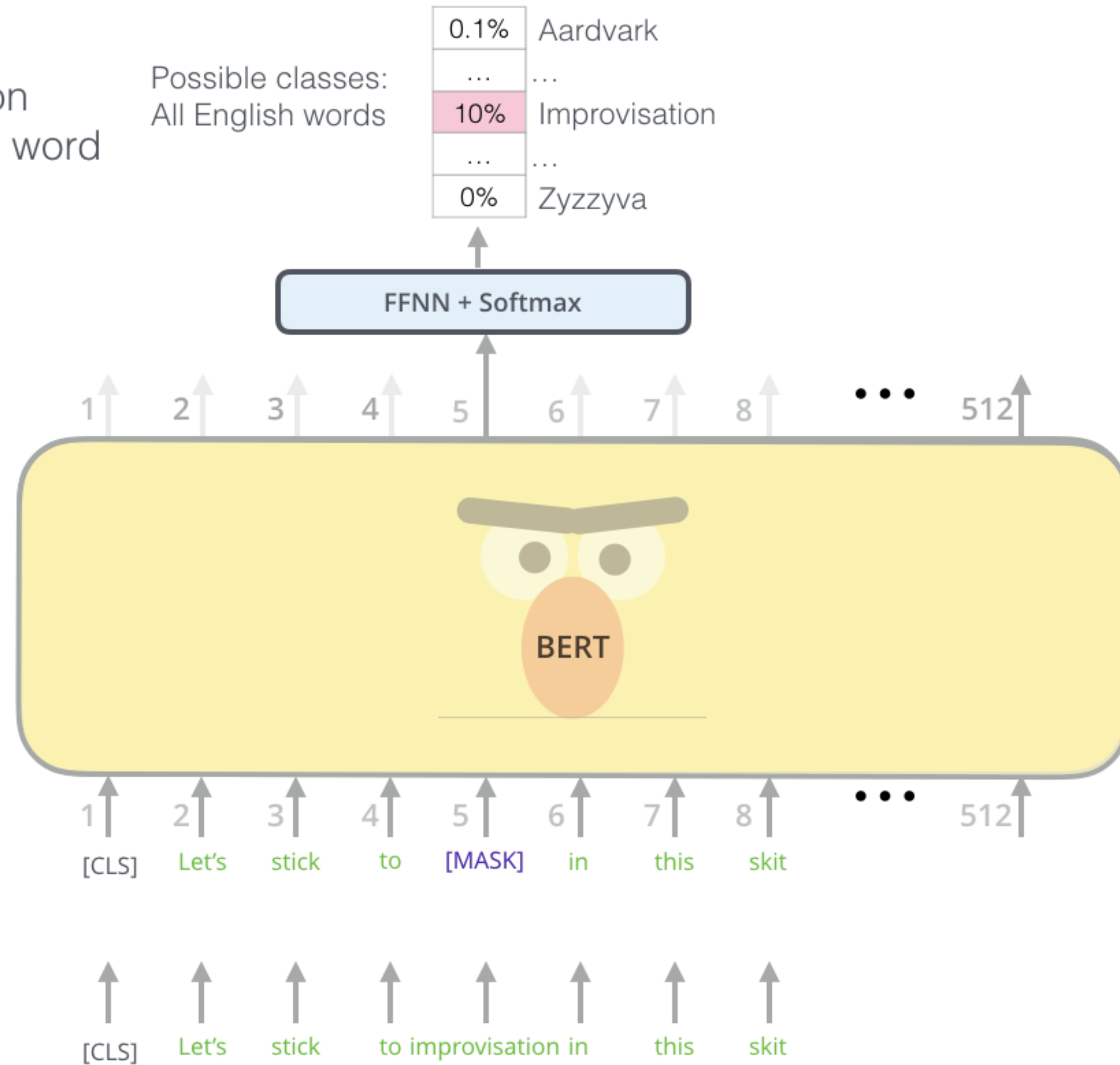
↑                              ↑

- Too little masking: Too expensive to train
- Too much masking: Not enough context



# Masked LM

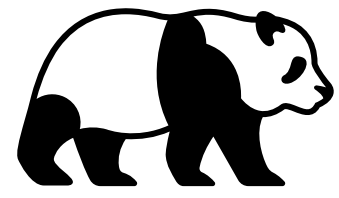
Use the output of the masked word's position to predict the masked word



Randomly mask 15% of tokens

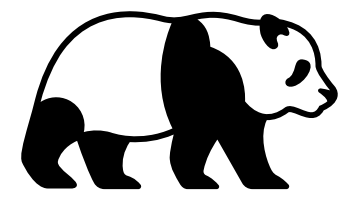
Input

BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.



# Masked LM

- **Problem:** Mask token never seen at fine-tuning
- **Solution:** 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, replace with [MASK]  
went to the store → went to the [MASK]
  - 10% of the time, replace random word  
went to the store → went to the running
  - 10% of the time, keep same  
went to the store → went to the store

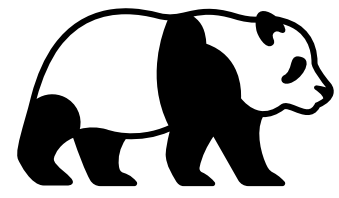


# Next Sentence Prediction

- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

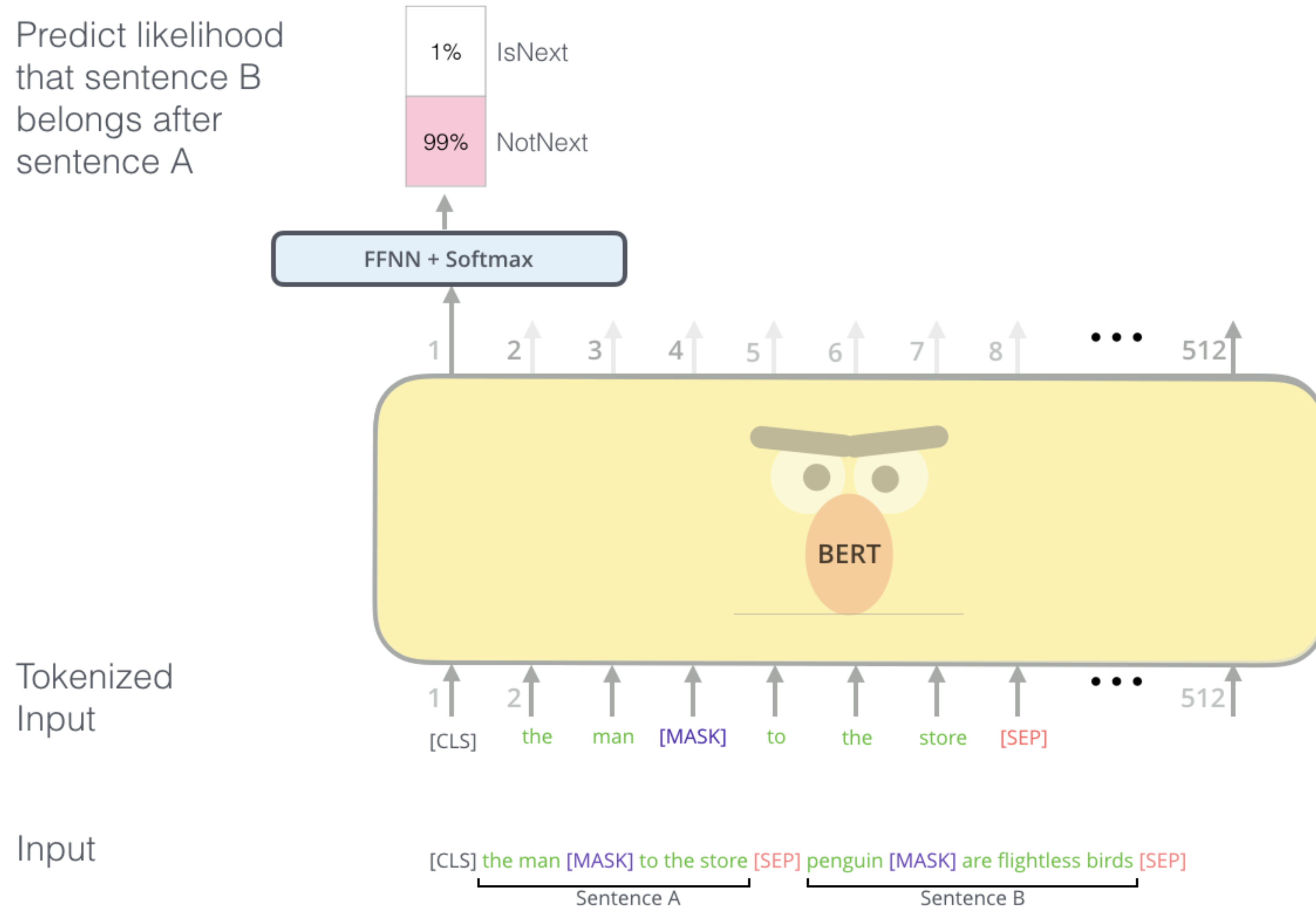
**Sentence A** = The man went to the store.  
**Sentence B** = He bought a gallon of milk.  
**Label** = IsNextSentence

**Sentence A** = The man went to the store.  
**Sentence B** = Penguins are flightless.  
**Label** = NotNextSentence

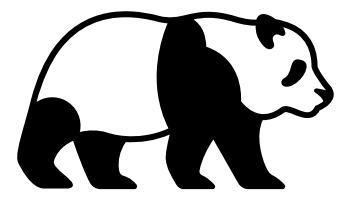


# Next Sentence Prediction

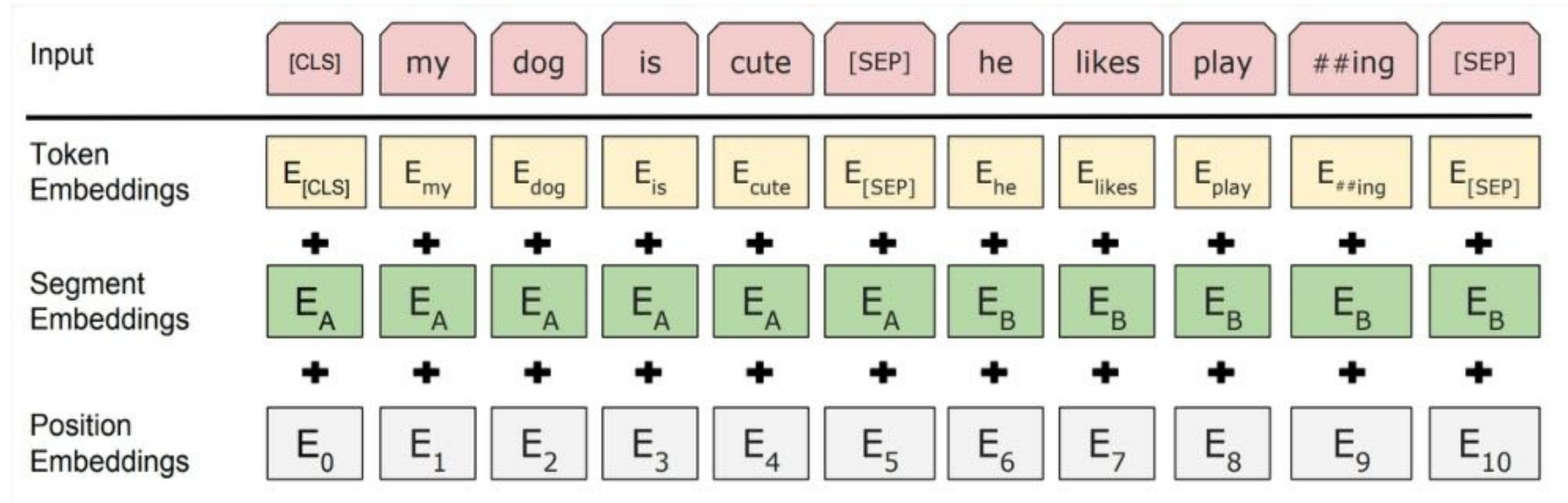
Predict likelihood that sentence B belongs after sentence A



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

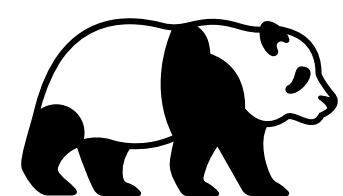


# Input Representation



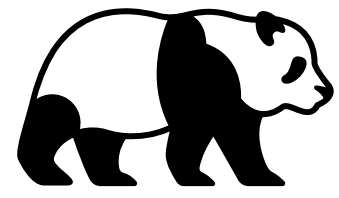
- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings.



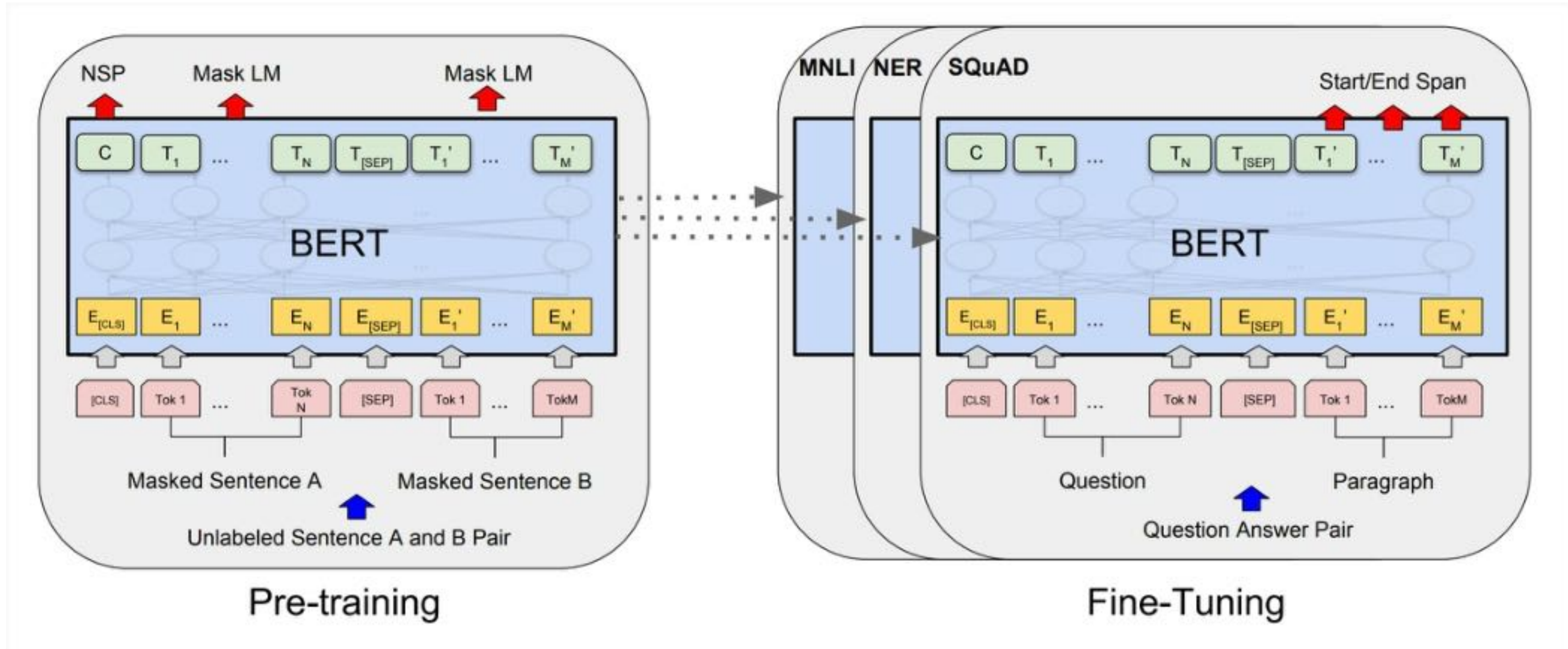


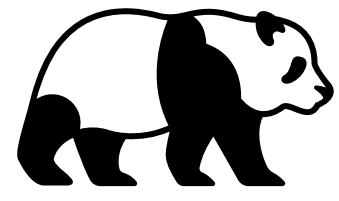
# More Details

- **Data:** Wikipedia (2.5B words) + BookCorpus (800M words)
- **Batch Size:** 131,072 words (1024 sequences \* 128 length or 256 sequences \* 512 length)
- **Training Time:** 1M steps (~40 epochs)
- **Optimizer:** AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- **Trained on** 4x4 or 8x8 TPU slice for 4 days

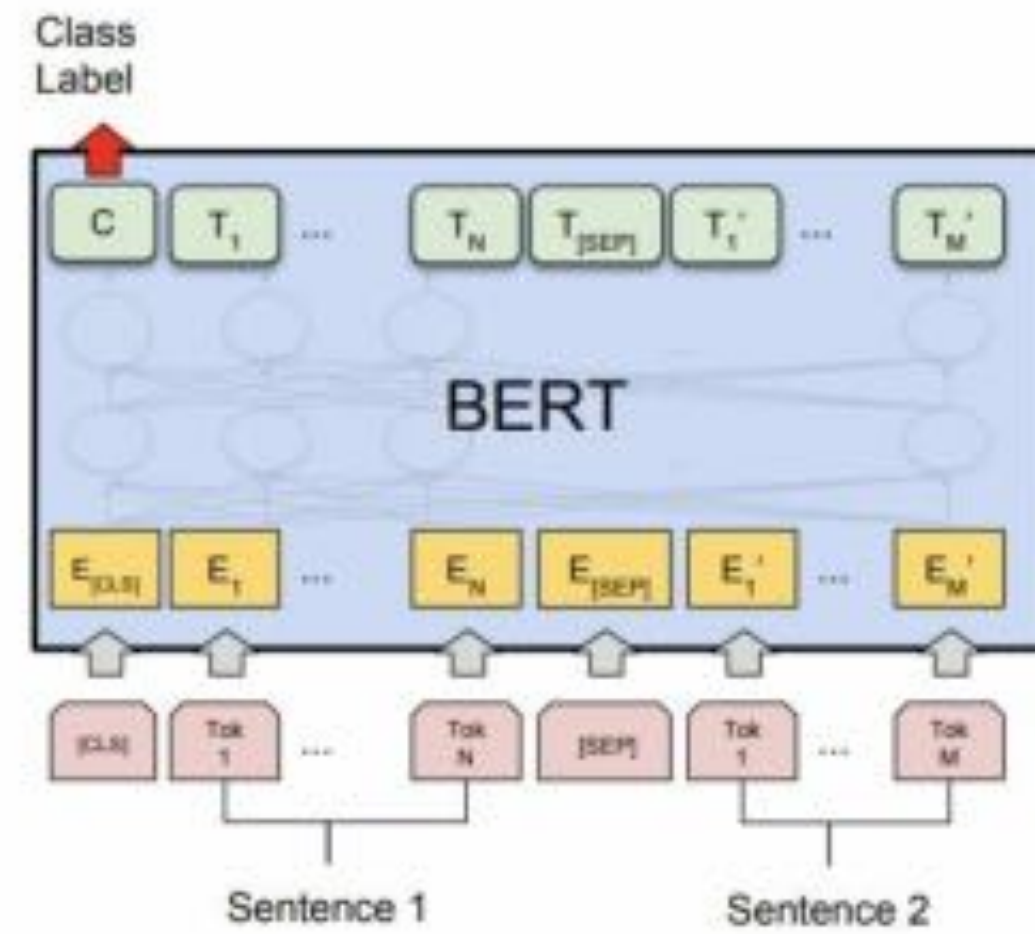


# Fine-Tuning Procedure

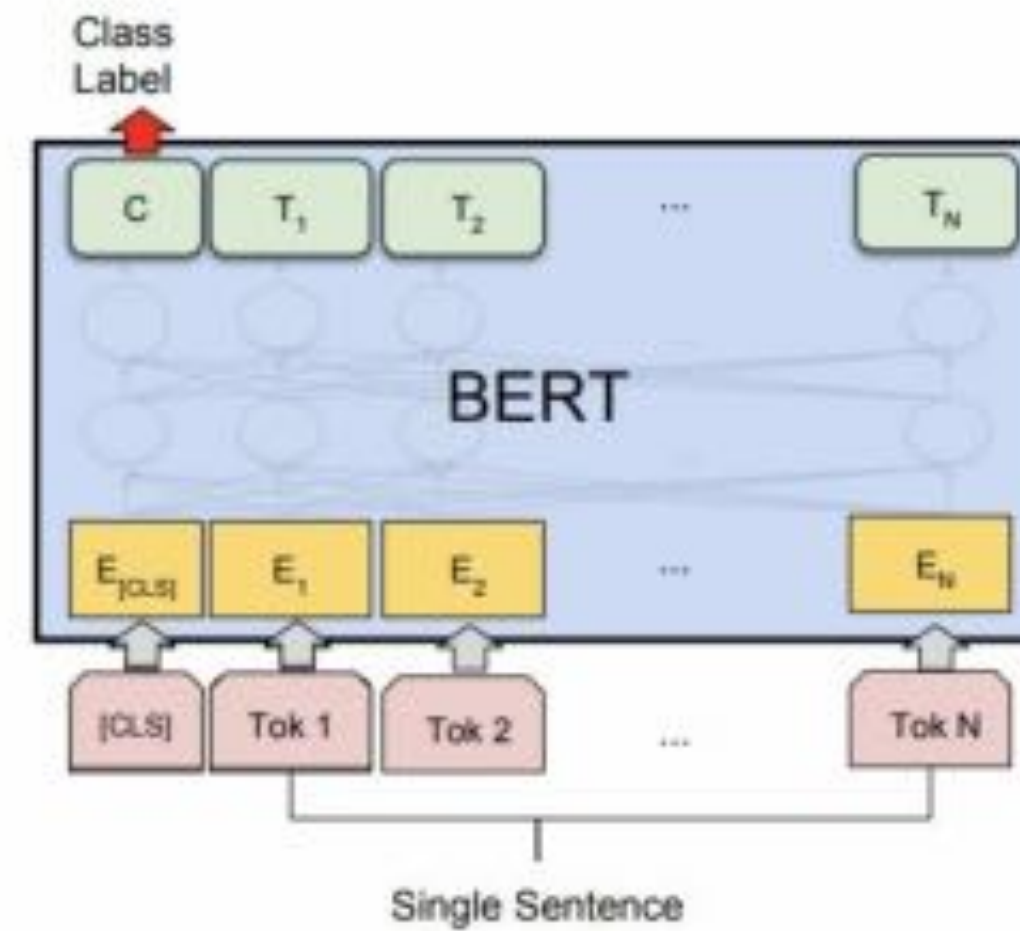




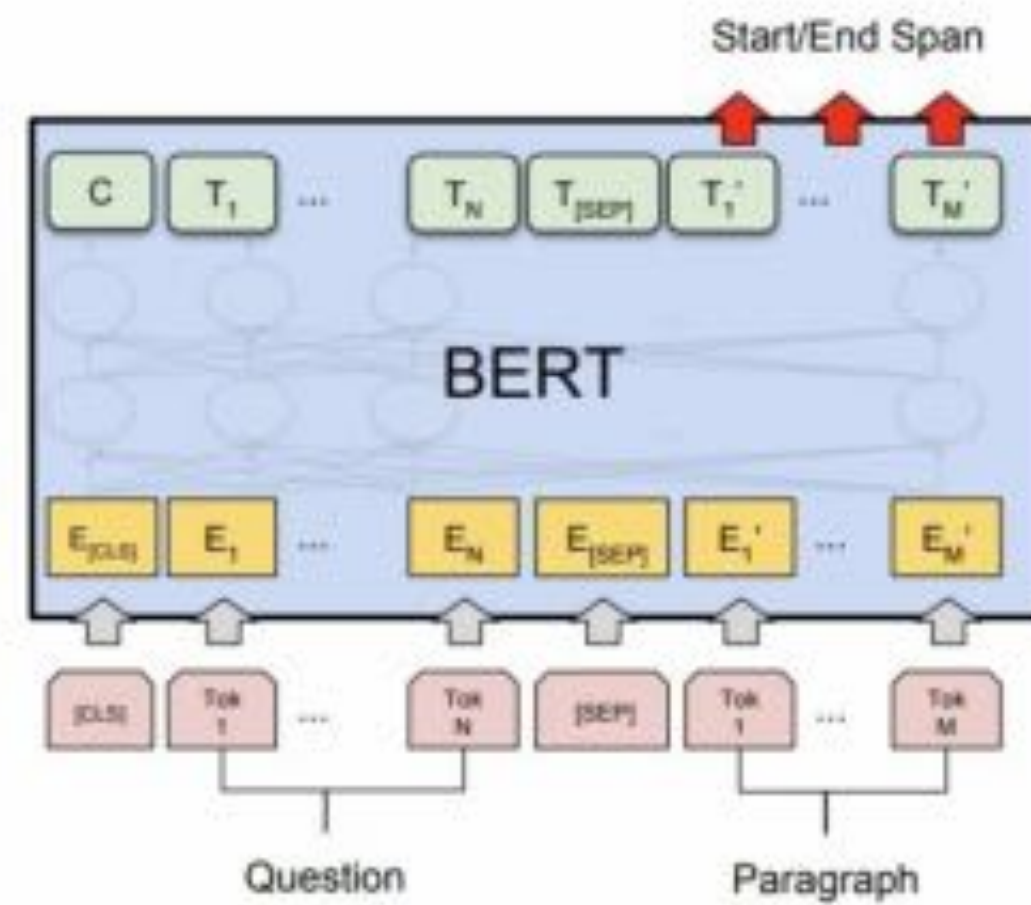
# Fine-Tuning Procedure



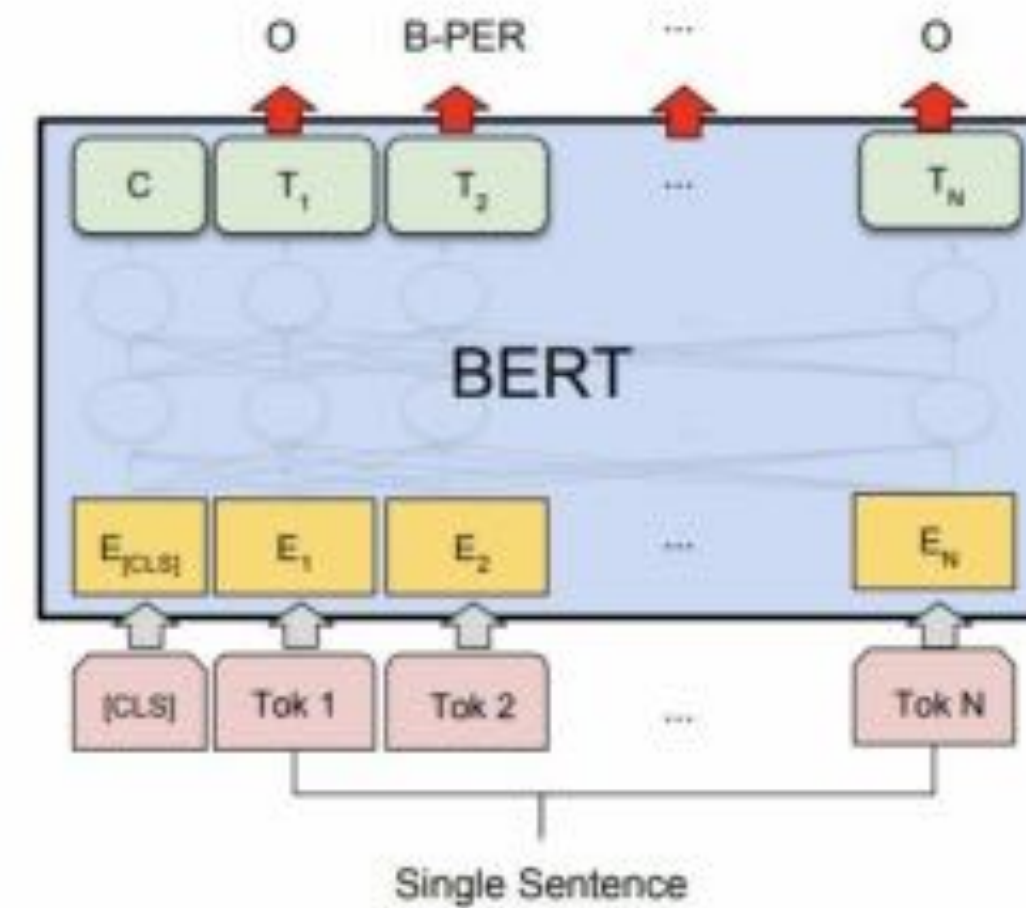
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA

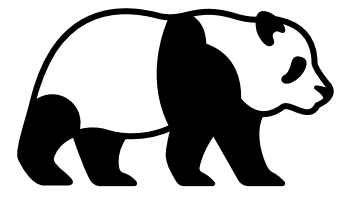


(c) Question Answering Tasks:  
SQuAD v1.1



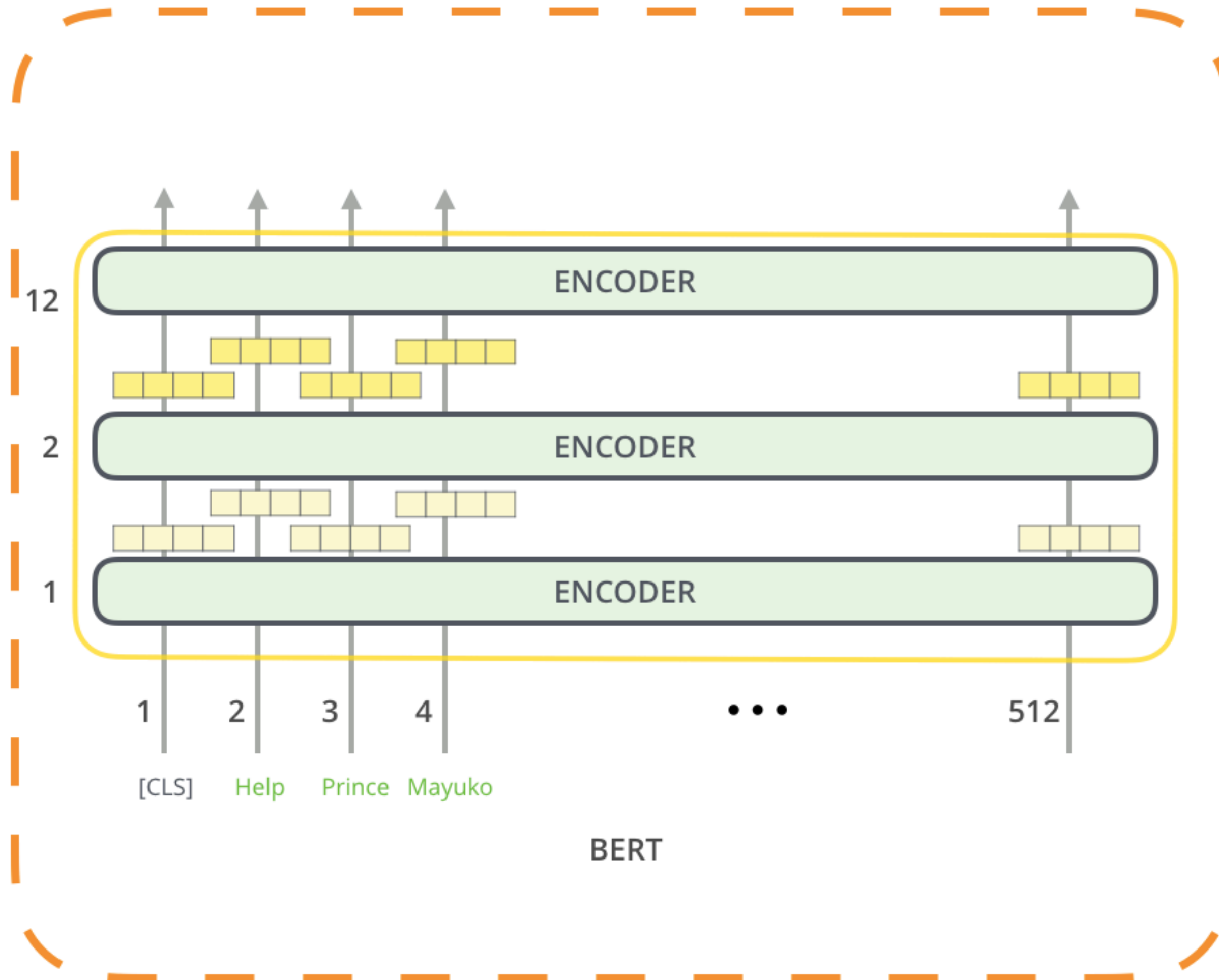
(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

Jacob Devlin,

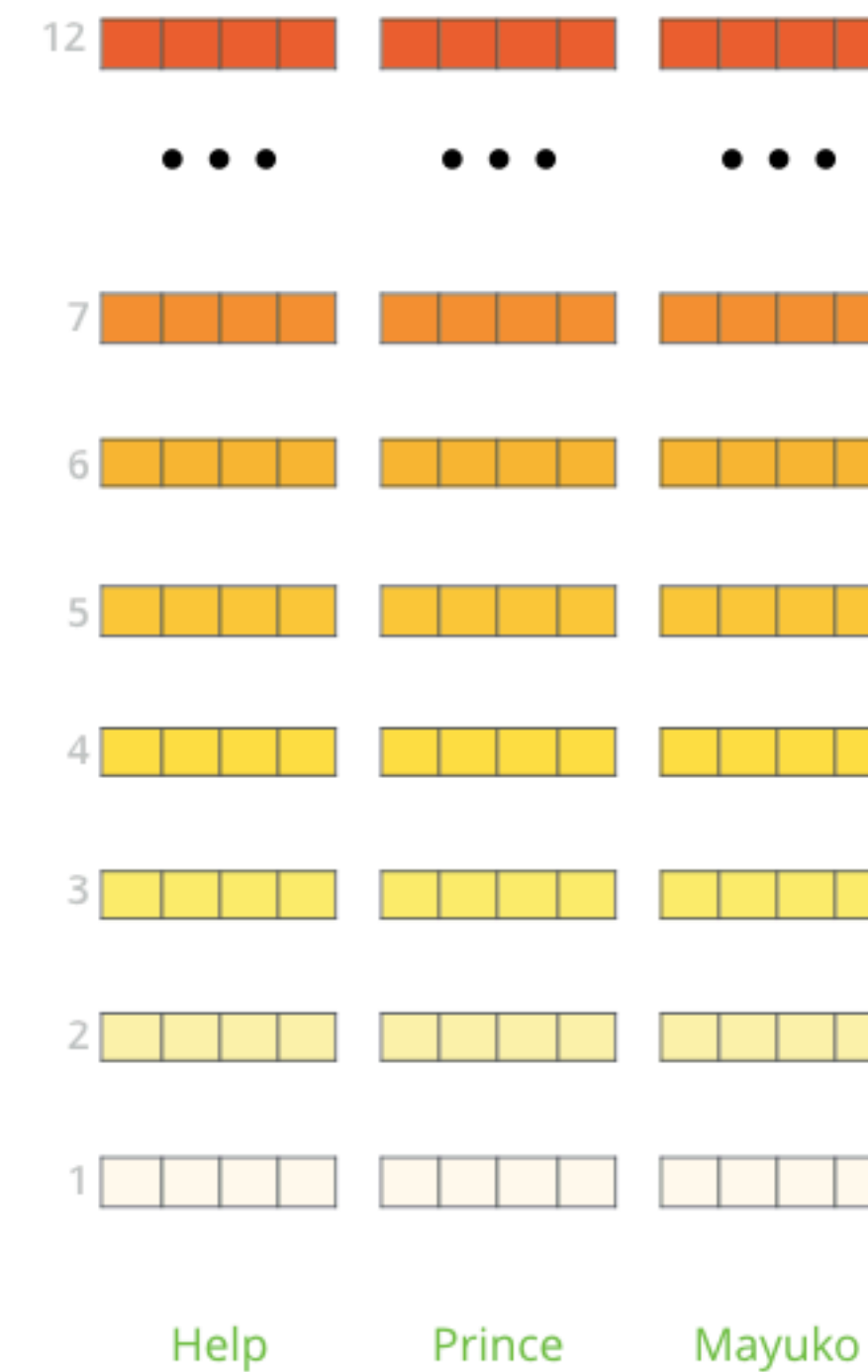


# BERT for Feature Extraction

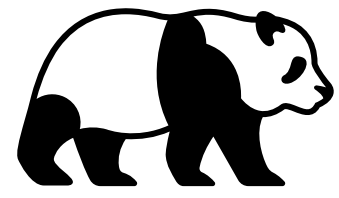
## Generate Contextualized Embeddings



The output of each encoder layer along each token's path can be used as a feature representing that token.

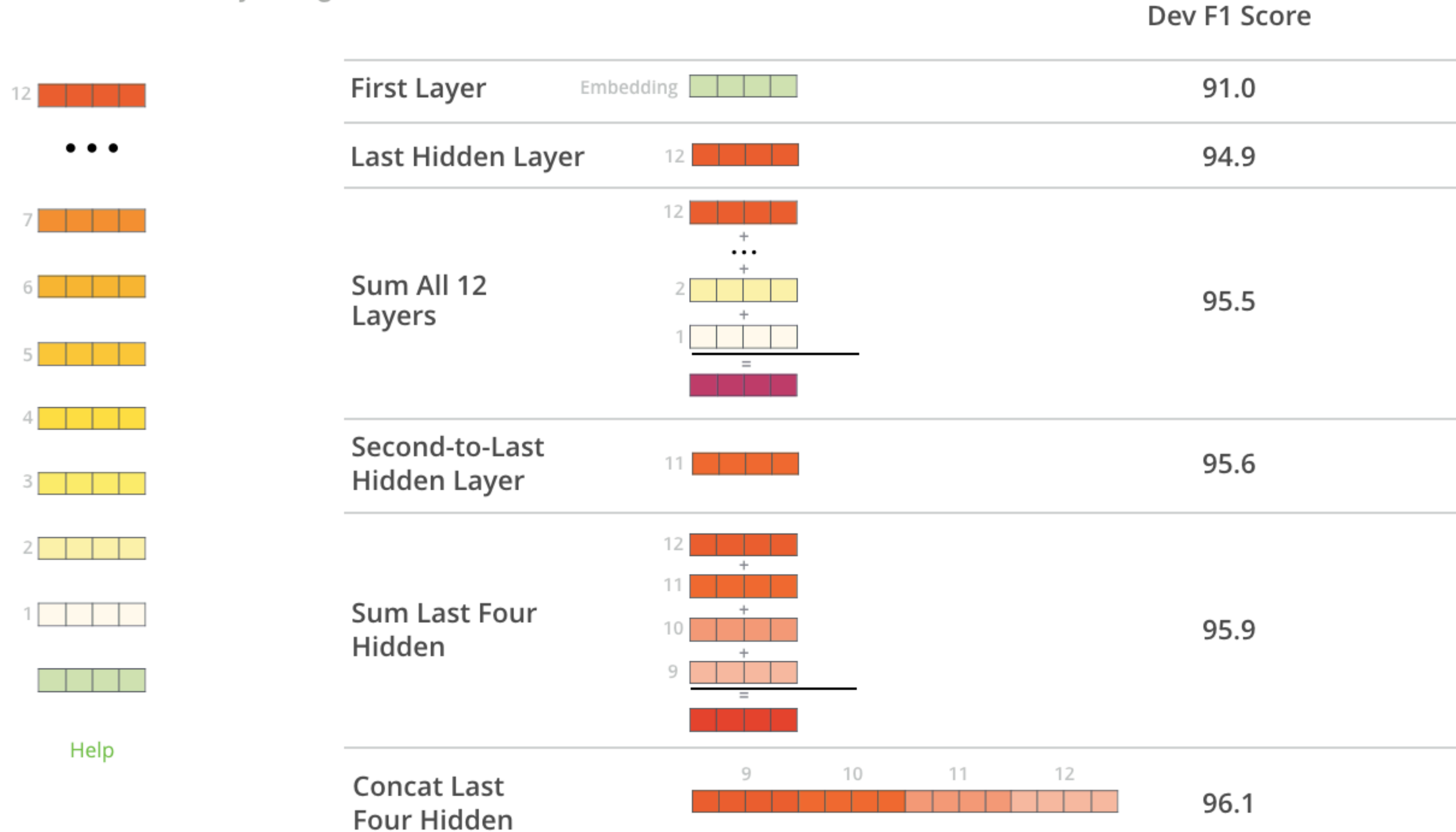


But which one should we use?

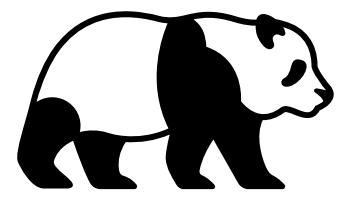


# BERT for Feature Extraction

What is the best contextualized embedding for “Help” in that context?  
For named-entity recognition task CoNLL-2003 NER



Help



# Performance: GLUE



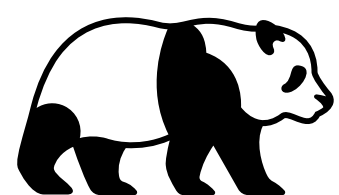
ML<sup>2</sup>



DeepMind

The General Language Understanding Evaluation (GLUE) benchmark is a collection of resources for training, evaluating, and analyzing natural language understanding systems. GLUE consists of:

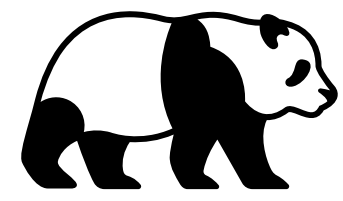
- A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty,
- A diagnostic dataset designed to evaluate and analyze model performance with respect to a wide range of linguistic phenomena found in natural language, and
- A public leaderboard for tracking performance on the benchmark and a dashboard for visualizing the performance of models on the diagnostic set.



# Performance: GLUE

Corpus	Train	Test	Task	Metrics	Domain
Single-Sentence Tasks					
CoLA	8.5k	<b>1k</b>	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
Similarity and Paraphrase Tasks					
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	<b>391k</b>	paraphrase	acc./F1	social QA questions
Inference Tasks					
MNLI	393k	<b>20k</b>	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	<b>146</b>	coreference/NLI	acc.	fiction books

Table 1: Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form.

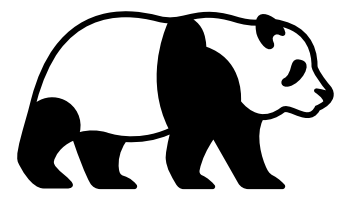


# Performance: GLUE

## GLUE Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>91.1</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>81.9</b>





# Performance: SQuAD 2.0

What action did the US begin that started the second oil shock?

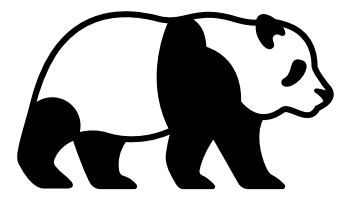
Ground Truth Answers: <No Answer>

Prediction: <No Answer>

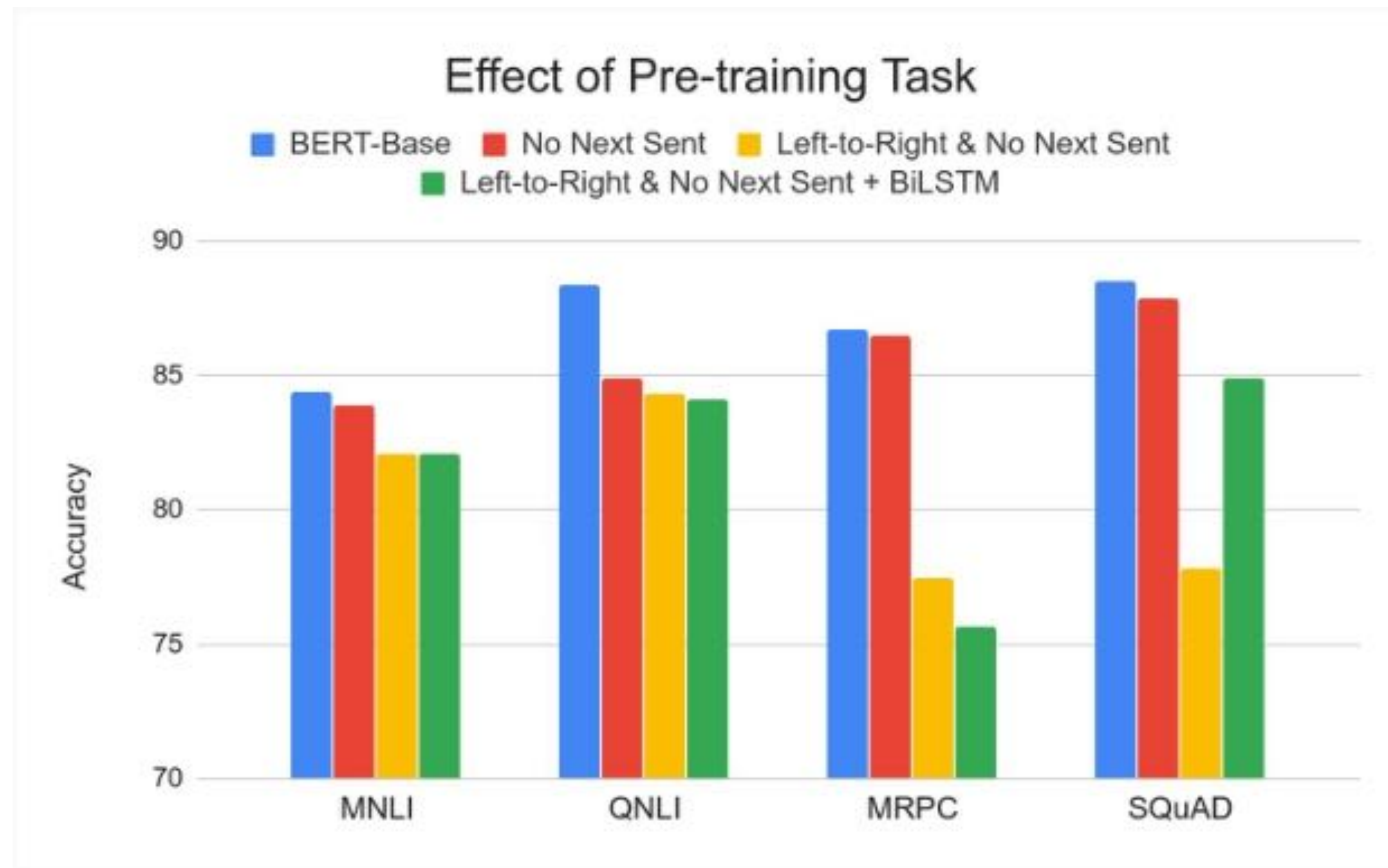
The 1973 **oil crisis** began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an **oil** embargo. By the end of the embargo in March 1974, the price of **oil** had risen from US\$3 per barrel to nearly \$12 globally; US prices were significantly higher. The embargo caused an **oil crisis**, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the "**first oil shock**", followed by the 1979 **oil crisis**, termed the "second **oil** shock."

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
12 Nov 08, 2018	BERT (single model) Google AI Language	80.005	83.061
20 Sep 13, 2018	nlnet (single model) Microsoft Research Asia	74.272	77.052

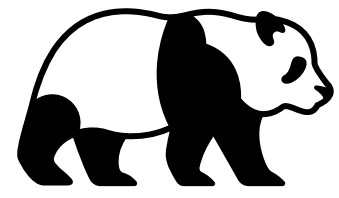
- Use token 0 ([CLS]) to emit logit for "no answer".
- "No answer" directly competes with answer span.
- Threshold is optimized on dev set.



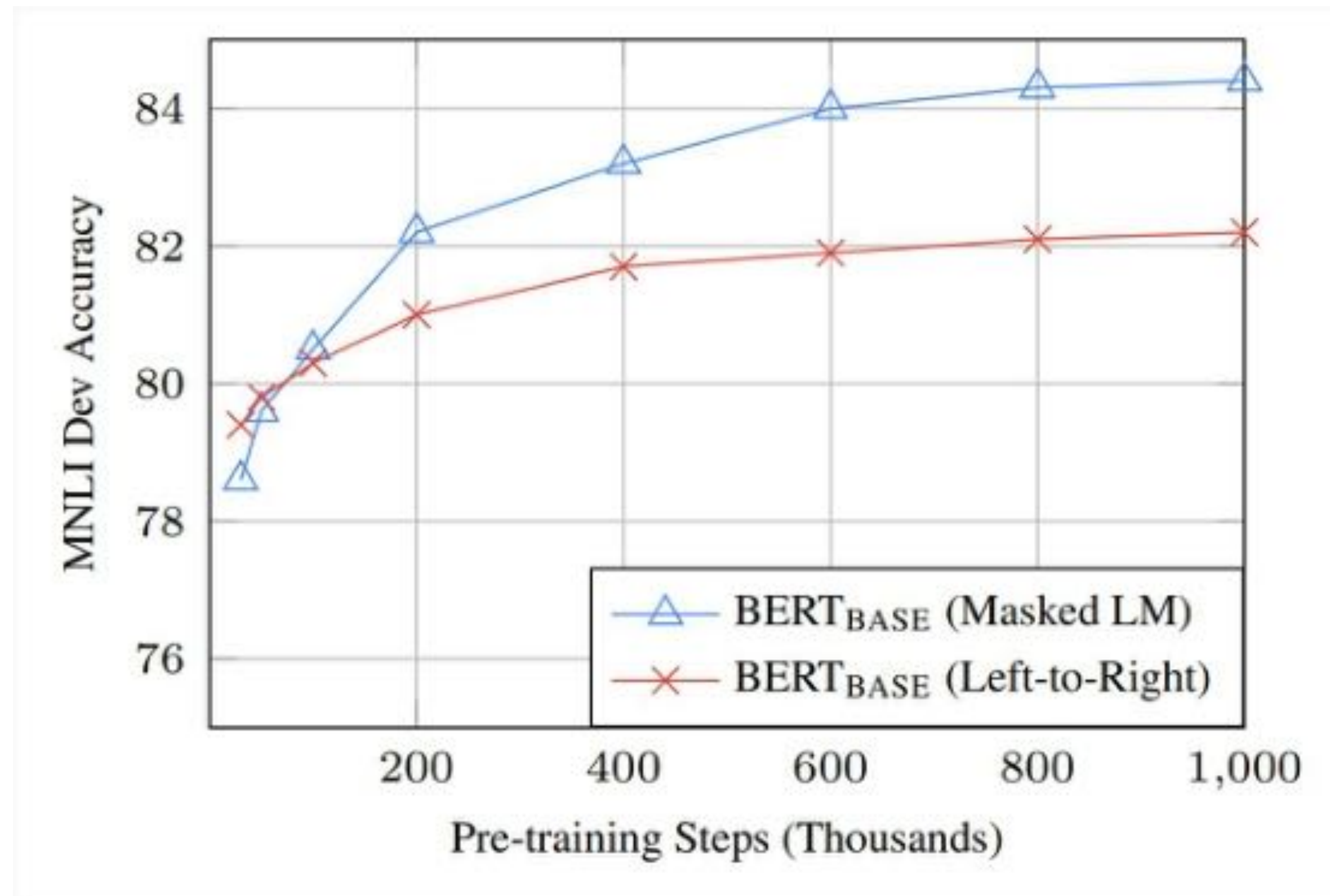
# Effect of Pre-training Task



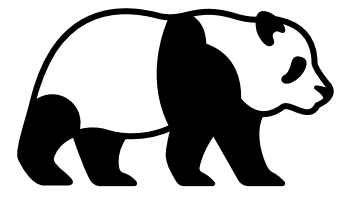
- Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks.
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM.



# Effect of Directionality and Training Time



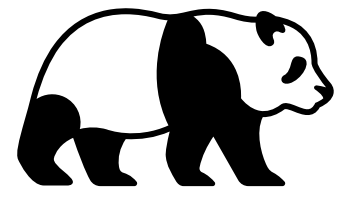
- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%
- But absolute results are much better almost immediately



# Effect of Model Size



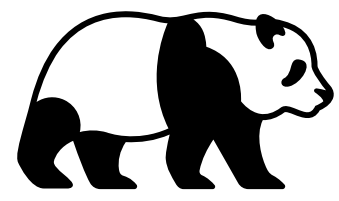
- Big models help a lot
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have not asymptoted



# Open Source Release

- One reason for BERT's success was the open source release
  - Minimal release (not part of a larger codebase)
  - No dependencies but TensorFlow (or PyTorch)
  - Abstracted so people could including a single file to use model
  - End-to-end push-button examples to train SOTA models
  - Thorough README
  - Idiomatic code
  - Well-documented code
  - Good support (for the first few months)

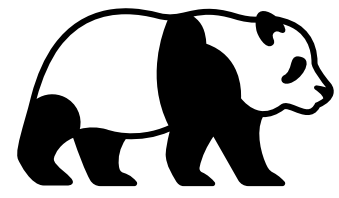
# **A Few Post-BERT Pre-training Advancements**



# RoBERTA

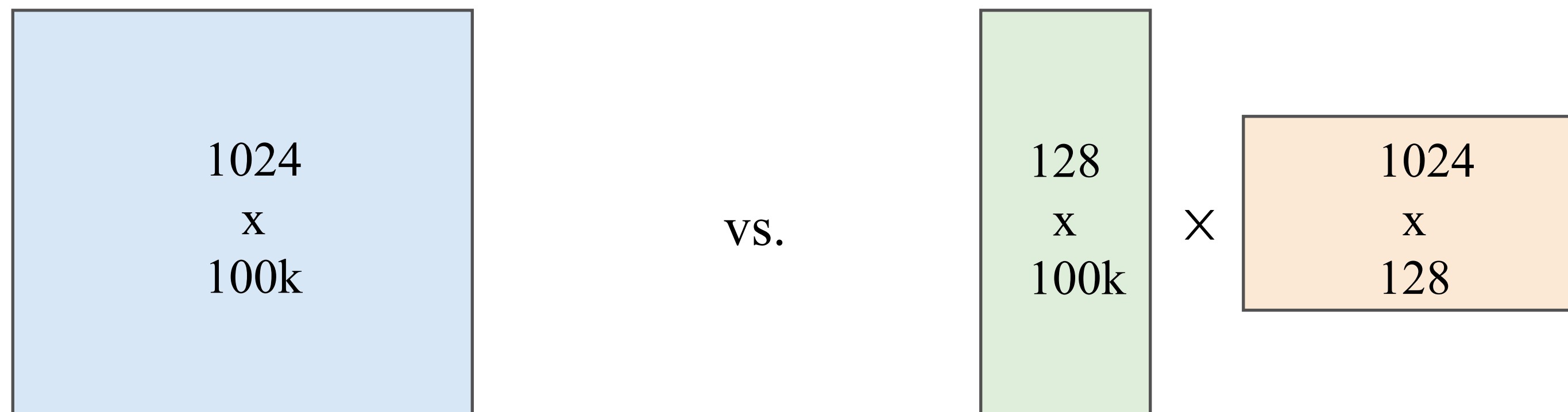
- RoBERTa: A Robustly Optimized BERT Pretraining Approach (Liu et al, University of Washington and Facebook, 2019)
- Trained BERT for more epochs and/or on more data
  - Showed that **more epochs alone helps**, even on same data
  - **More data also helps**
- Improved masking and pre-training data slightly

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT <sub>LARGE</sub>	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet <sub>LARGE</sub>	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	<b>90.2/90.2</b>	<b>94.7</b>	<b>92.2</b>	<b>86.6</b>	<b>96.4</b>	<b>90.9</b>	<b>68.0</b>	<b>92.4</b>	<b>91.3</b>	-

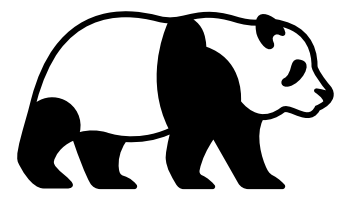


# ALBERT

- ALBERT: A Lite BERT for Self-supervised Learning of Language Representations (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: **Factorized embedding parameterization**
  - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix
- Innovation #2: **Cross-layer parameter sharing**
  - Share all parameters between Transformer layers







# ALBERT

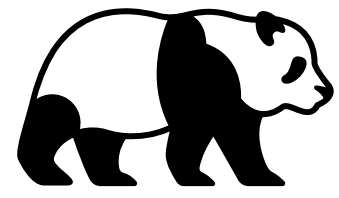
- Results:

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
<i>Single-task single models on dev</i>								
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	<b>92.2</b>	86.6	96.4	<b>90.9</b>	68.0	92.4
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7
ALBERT (1.5M)	<b>90.8</b>	<b>95.3</b>	<b>92.2</b>	<b>89.2</b>	<b>96.9</b>	<b>90.9</b>	<b>71.4</b>	<b>93.0</b>

- ALBERT is light in terms of parameters, not speed

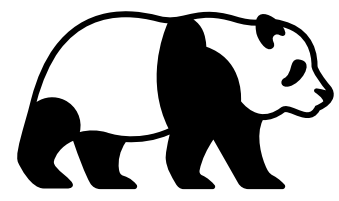
Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	<b>94.1/88.3</b>	<b>88.1/85.1</b>	<b>88.0</b>	<b>95.2</b>	<b>82.3</b>	<b>88.7</b>	0.3x

Jacob Devlin,



# T5

- Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (Raffel et al, Google, 2019)
- **Ablated many aspects of pre-training:**
  - Model size
  - Amount of training data
  - Domain/cleanness of training data
  - Pre-training objective details (e.g., span length of masked text)
  - Ensembling
  - Finetuning recipe (e.g., only allowing certain layers to finetune)
  - Multi-task training



# T5

Conclusions:

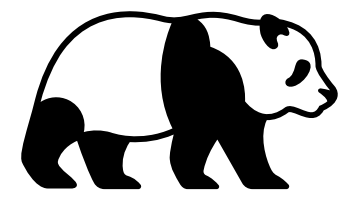
- **Scaling up model size and amount of training data** helps a lot
- Best model is 11B parameters (BERT-Large is 330M), trained on 120B words of cleaned common crawl text
- Exact masking/corruptions strategy doesn't matter that much
- Mostly negative results for better finetuning and multi-task strategies

T5 results on SuperGLUE: <https://super.gluebenchmark.com/leaderboard/>

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultIRC	ReCoRD	RTE	WIC	WSC	AX-b	AX-g	
+	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	66.7	93.3/93.8
+	2	Zirui Wang	T5 + Meena, Single Model (Meena Team - Google Brain)		90.2	91.3	95.8/97.6	97.4	88.3/63.0	94.2/93.5	92.7	77.9	95.9	66.5	88.8/89.9
	3	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	4	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
+	5	Huawei Noah's Ark Lab	NEZHA-Plus		86.7	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	58.0	87.1/74.4

Jacob Devlin,

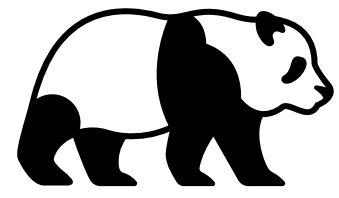
# Distillation



# Applying Models to Production Services

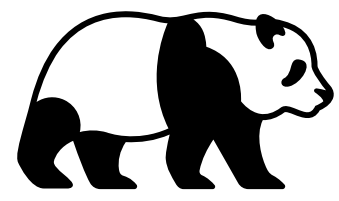
- BERT and other pre-trained language models are extremely large and expensive
- **How are companies applying them to low-latency production services?**





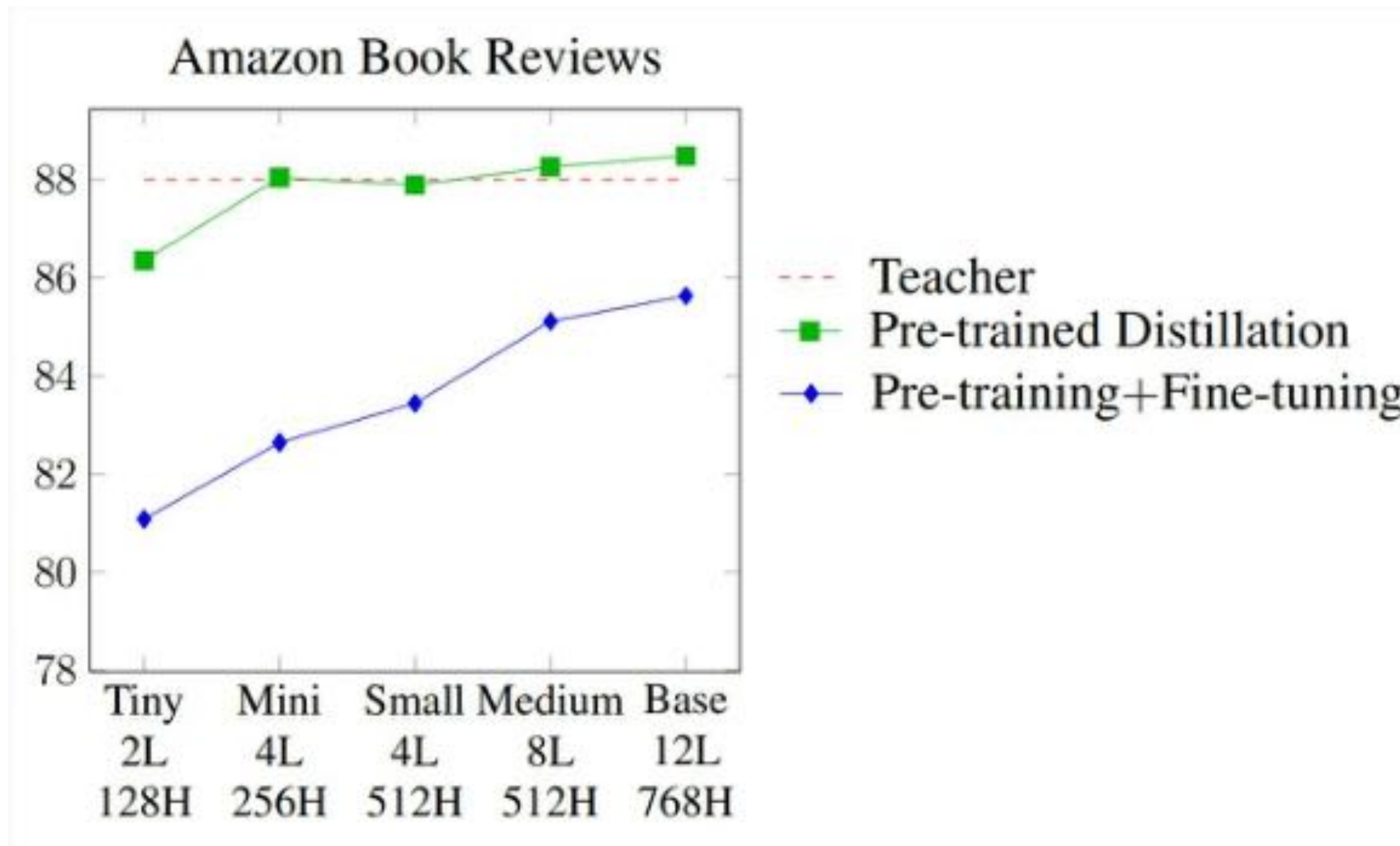
# Distillation

- Answer: **Distillation** (a.k.a., **model compression**)
- Idea has been around for a long time:
  - Model Compression (Bucila et al, 2006)
  - Distilling the Knowledge in a Neural Network (Hinton et al, 2015)
- Simple technique:
  - Train “Teacher”: Use SOTA pre-training + fine-tuning technique to train model with maximum accuracy
  - Label a large amount of unlabeled input examples with Teacher
  - Train “Student”: Much smaller model (e.g., 50x smaller) which is trained to mimic Teacher output
  - Student objective is typically Mean Square Error or Cross Entropy

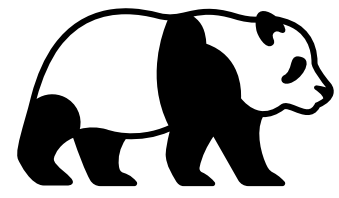


# Distillation

- Example distillation results
  - 50k labeled examples, 8M unlabeled examples
- Distillation works much better than pre-training + fine-tuning with smaller model



*Well-Read Students Learn Better: On the Importance of Pre-training Compact Models (Turc et al, 2020)*

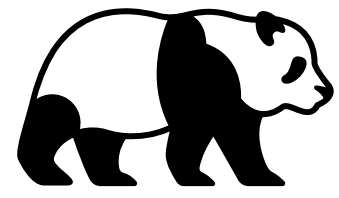


# Distillation

- Why does distillation work so well? A hypothesis:
  - Language modeling is the “ultimate” NLP task in many ways
    - I.e., a perfect language model is also a perfect question answering/entailment/sentiment analysis model
  - Training a massive language model learns millions of latent features which are useful for these other NLP tasks
  - Fine-tuning mostly just picks up and tweaks these existing latent features
  - This requires an oversized model, because only a subset of the features are useful for any given task
  - Distillation allows the model to only focus on those features
  - Supporting evidence: Simple self-distillation (distilling a smaller BERT model) doesn't work

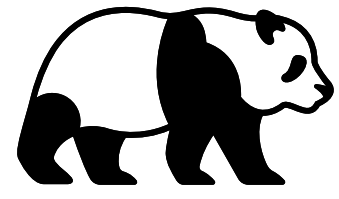


# Conclusions



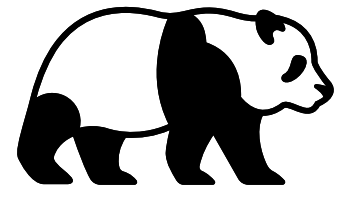
# Conclusions

- Pre-trained bidirectional language models work incredibly well
- However, the models are extremely expensive
- Improvements (unfortunately) seem to mostly come from even more expensive models and more data
- The inference/serving problem is mostly “solved” through distillation



# Todo

- **Suggested Readings: (No reading assignments for this week and next week)**
  - [The original transformer paper] Attention Is All You Need (<https://arxiv.org/pdf/1706.03762.pdf>)
  - [BERT] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (<https://arxiv.org/abs/1810.04805>)
  - Pre-trained Models for Natural Language Processing: A Survey (<https://arxiv.org/abs/2003.08271>)
  - Efficient Transformers: A Survey (<https://arxiv.org/pdf/2009.06732.pdf>)



# References

1. [https://lena-voita.github.io/nlp\\_course/seq2seq\\_and\\_attention.html](https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html)
2. **Stanford CS224N, Winter 2021**: <http://web.stanford.edu/class/cs224n/>, lecture: Transformers and Self-Attention For Generative Models
3. Talk of Jacob Devlin: Contextual Word Representations with BERT and Other Pre-trained Language Models
4. <http://jalammar.github.io/illustrated-bert/>

# Thanks! Q&A

**Bang Liu**

**Email:** [bang.liu@umontreal.ca](mailto:bang.liu@umontreal.ca)

**Homepage:** <http://www-labs.iro.umontreal.ca/~liubang/>