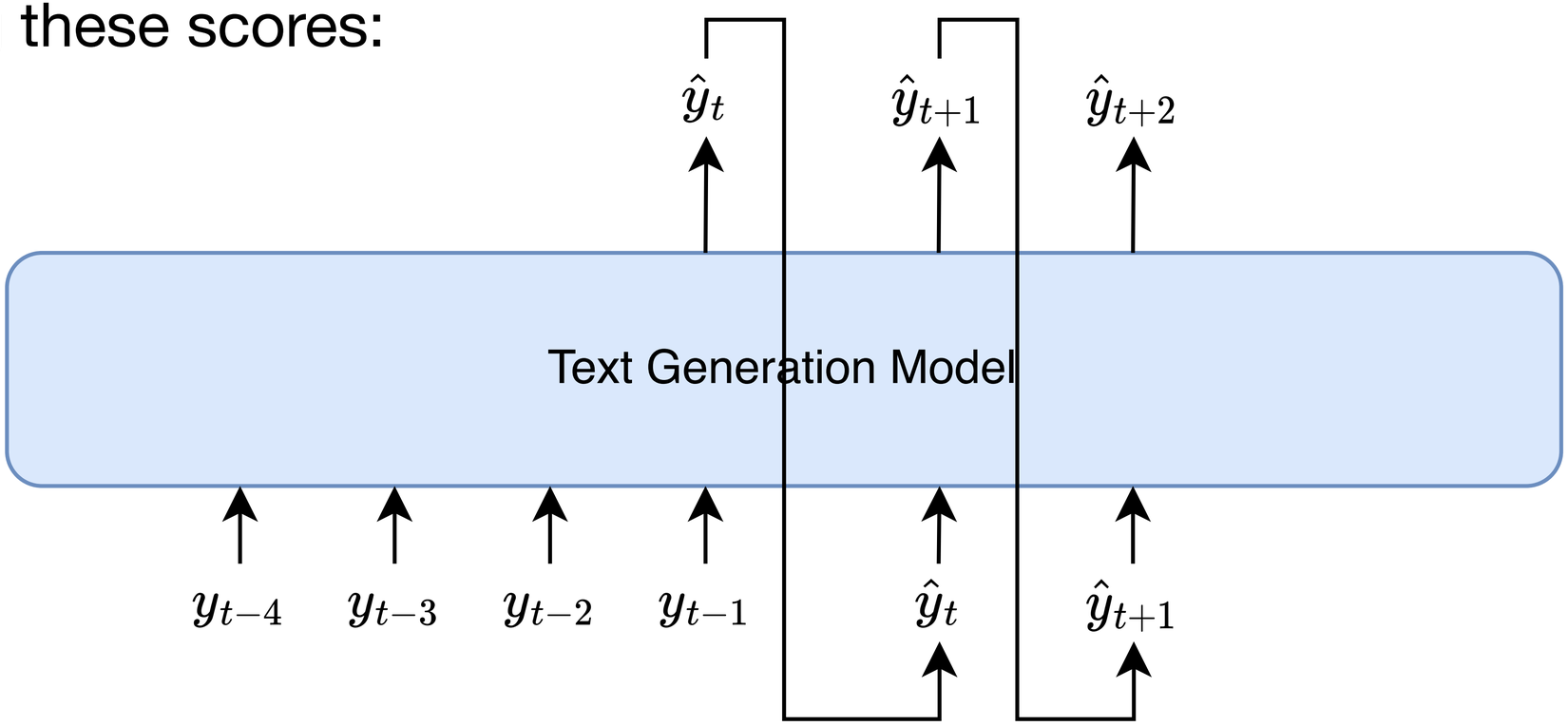
Natural Language Generation II

Kirill Milintsevich | 12.01.2024

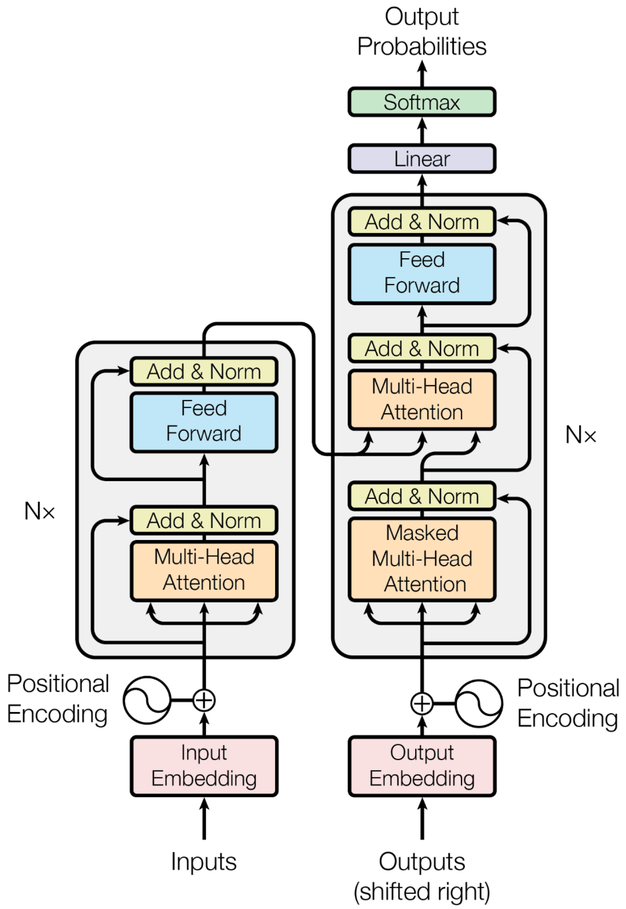
Autoregressive Models (Reminder)

- At each time step t, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$. Then, we compute a probability distribution P over $w \in V$ using these scores:



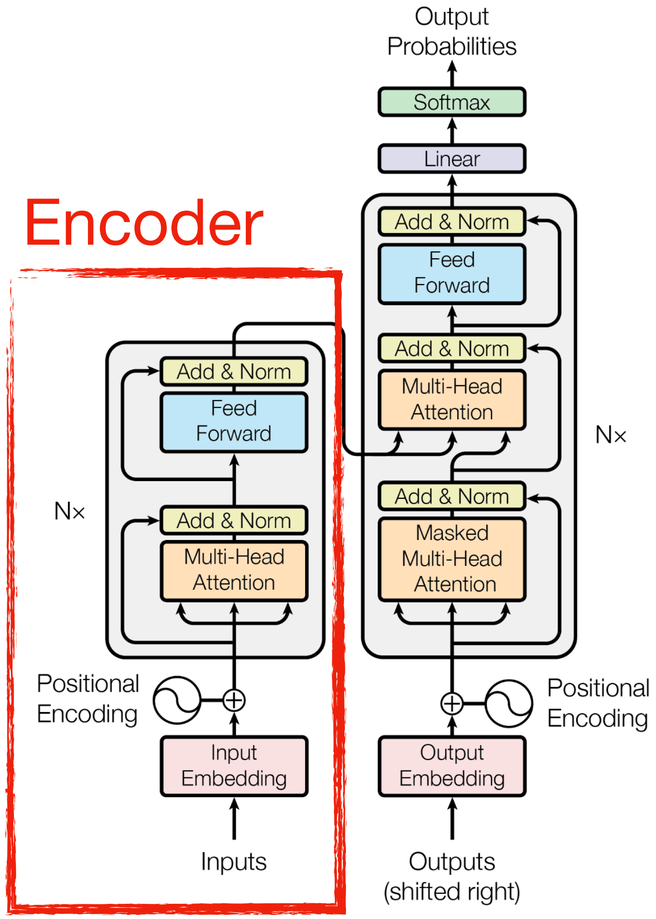
Transformer Architecture

Vaswani, Ashish, et al. "Attention is all you need." (2017)



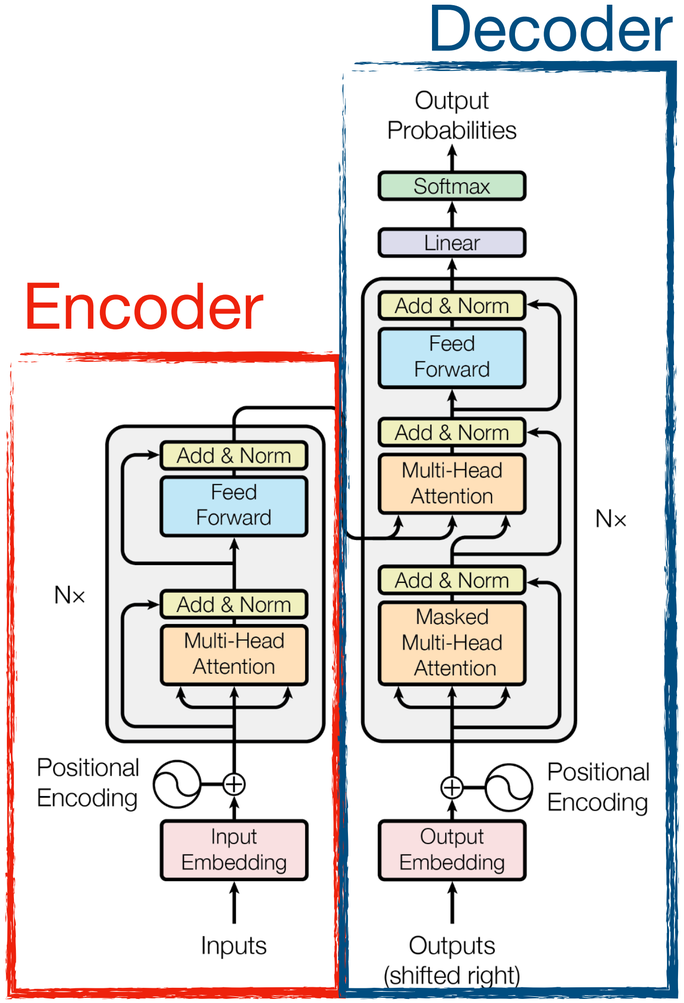
Transformer Architecture

Vaswani, Ashish, et al. "Attention is all you need." (2017)

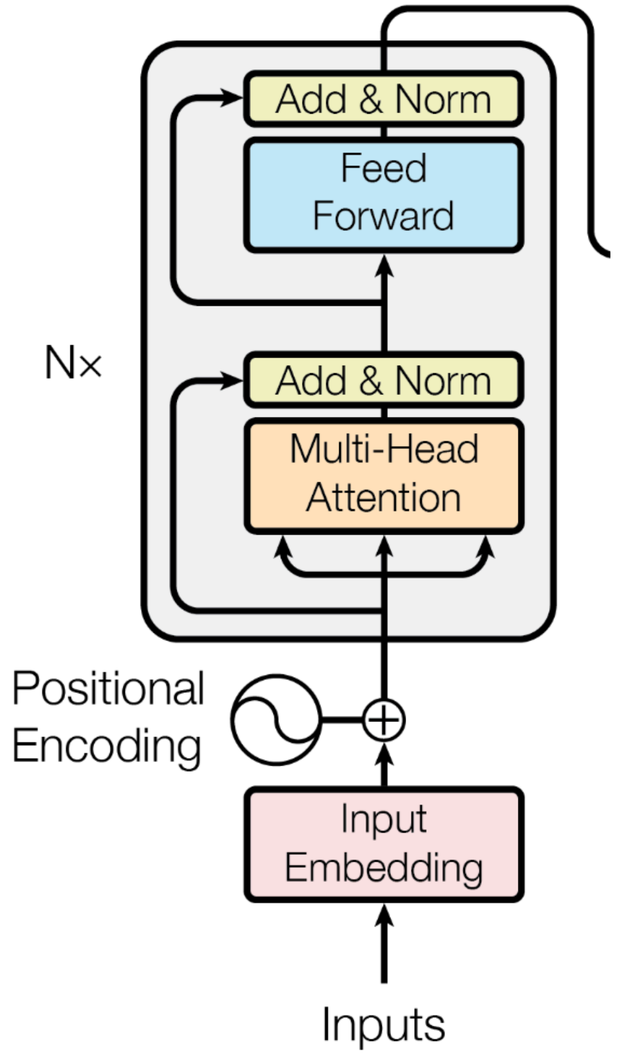


Transformer Architecture

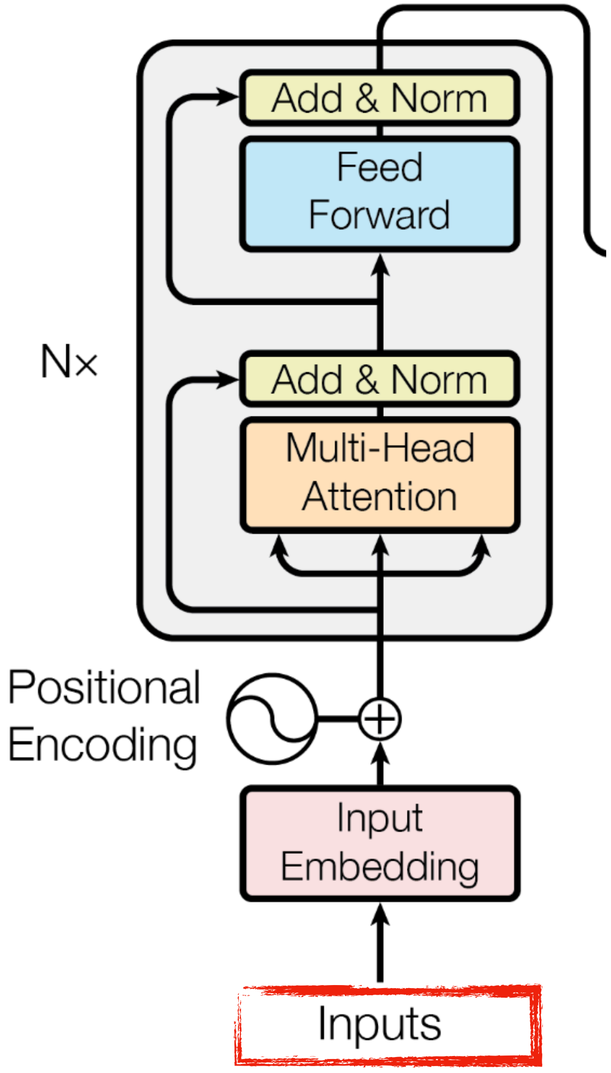
Vaswani, Ashish, et al. "Attention is all you need." (2017)



Transformer Encoder



Transformer Encoder



- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization

Byte-Pair Encoding

Training

- First, we compute the unique set of words in the corpus
- Split each word into characters
- Start merging most frequent pairs

- ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
- ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Byte-Pair Encoding

Training

- First, we compute the unique set of words in the corpus
- Split each word into characters
- Start merging most frequent pairs
- The most frequent pair is `("u", "g")`

- ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
- ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug"]`

Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)`

Byte-Pair Encoding

Training

- First, we compute the unique set of words in the corpus
- Split each word into characters
- Start merging most frequent pairs
- The most frequent pair is `(“u”, “n”)`

- ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
- ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug", "un"]`

Corpus: `("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)`

Byte-Pair Encoding

Training

- First, we compute the unique set of words in the corpus
- Split each word into characters
- Start merging most frequent pairs
- The most frequent pair is `("h", "ug")`

- ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
- ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

Vocabulary: `["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]`

Corpus: `("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)`

Byte-Pair Encoding

Tokenization

1. Normalization
2. Pre-tokenization
3. Splitting the words into individual characters
4. Applying the merge rules learned in order on those splits

Byte-Pair Encoding

Tokenization

1. Normalization
2. Pre-tokenization
3. Splitting the words into individual characters
4. Applying the merge rules learned in order on those splits

Learned rules:

```
("u", "g") -> "ug"

("u", "n") -> "un"

("h", "ug") -> "hug"
```

Byte-Pair Encoding

Tokenization

1. Normalization
2. Pre-tokenization
3. Splitting the words into individual characters
4. Applying the merge rules learned in order on those splits

Learned rules:

```
("u", "g") -> "ug"

("u", "n") -> "un"

("h", "ug") -> "hug"
```

```
“bug” -> “b”, “ug” | “mug” -> “[UNK]”, “ug”
```

WordPiece Tokenization

Training

- Same as BPE but different merging rule

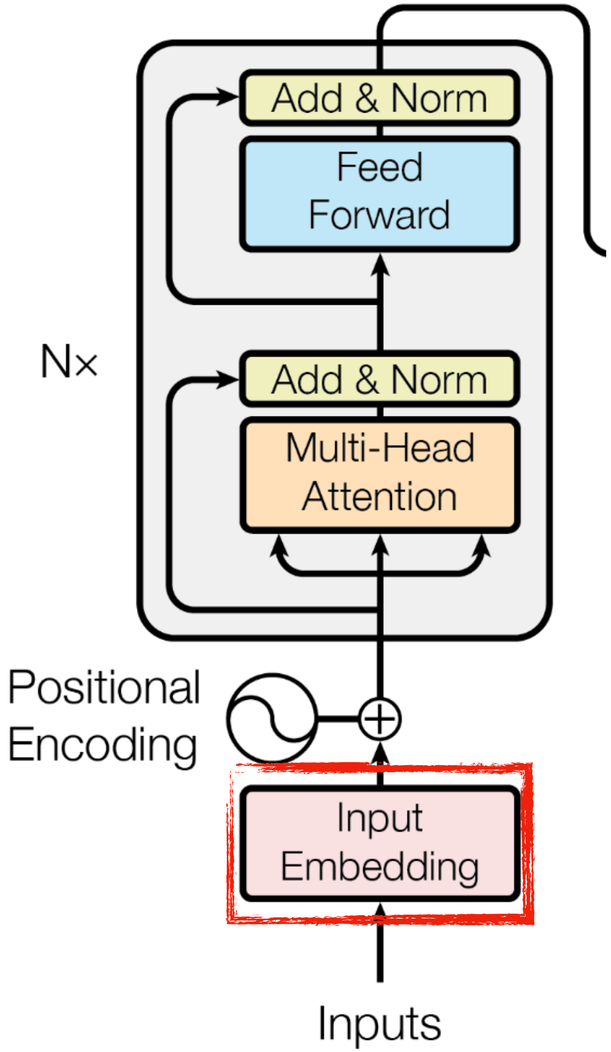
$$\text{score} = \frac{\text{freq\_of\_pair}}{\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element}}$$

WordPiece Tokenization

Tokenization

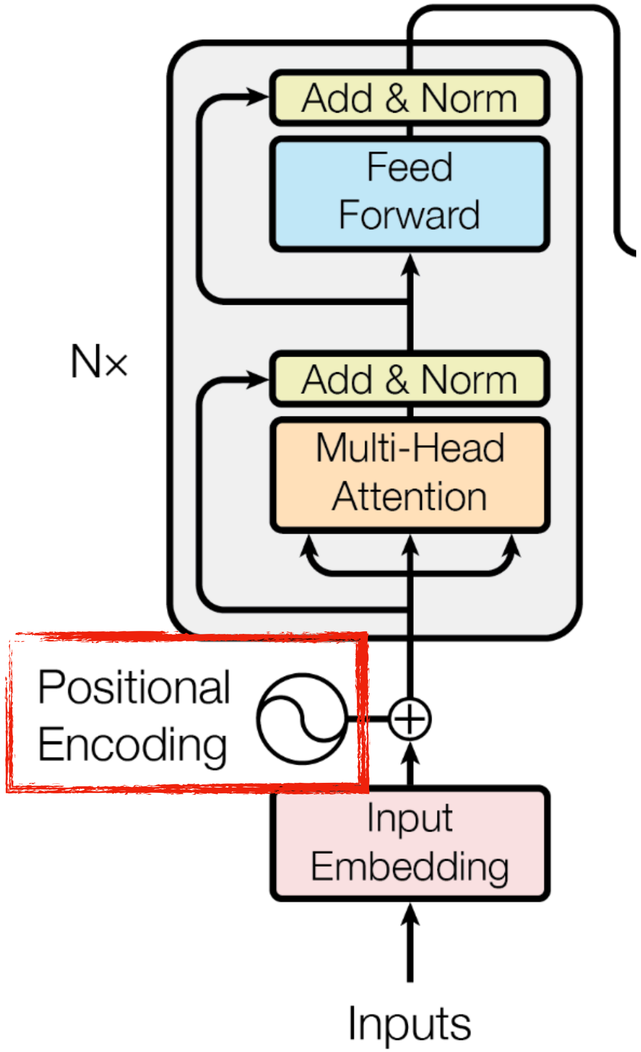
- Same as BPE but the WordPiece starts by searching for the longest sub-token in the vocabulary

Transformer Encoder



- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer

Transformer Encoder



- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence

Positional Encoding

- Since Transformer doesn't have recursion it lacks information about words' positions

Let t be the desired position in an input sentence, $\overrightarrow{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$)
Then $f : \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\overrightarrow{p_t}$ and it is defined as follows:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

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

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

Positional Encoding

Intuition

0 :	0	0	0	0
1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1

8 :	1	0	0	0
9 :	1	0	0	1
10 :	1	0	1	0
11 :	1	0	1	1
12 :	1	1	0	0
13 :	1	1	0	1
14 :	1	1	1	0
15 :	1	1	1	1

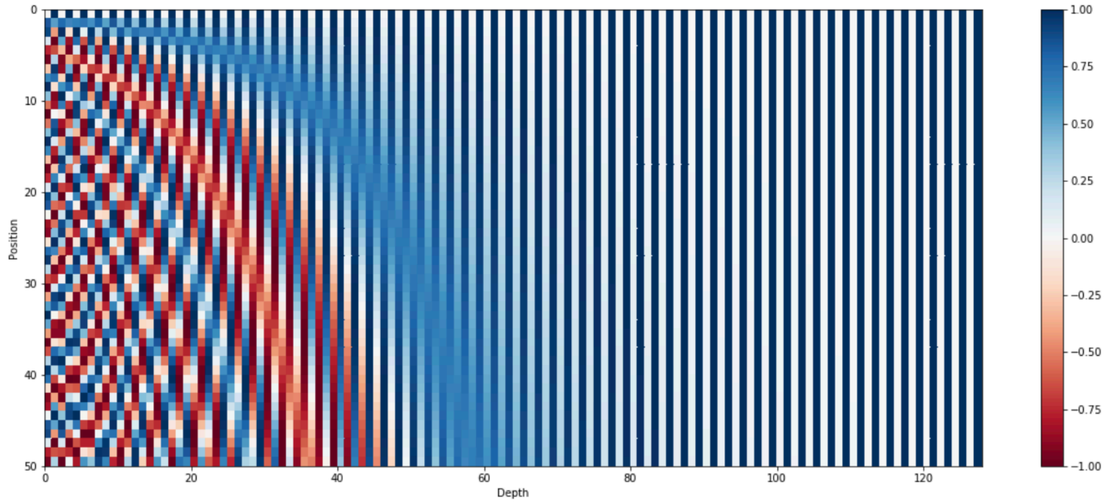
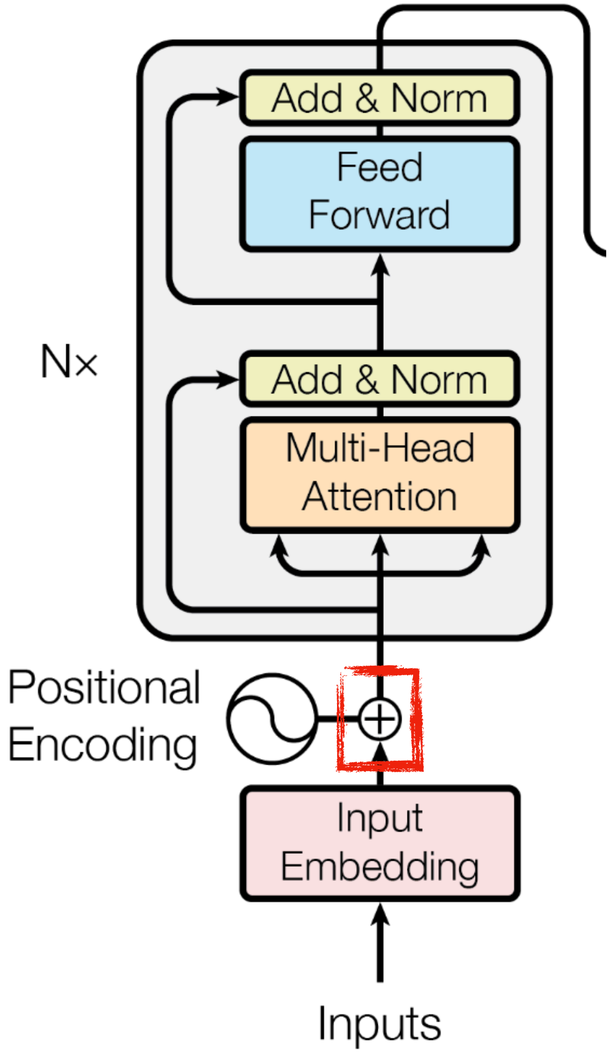


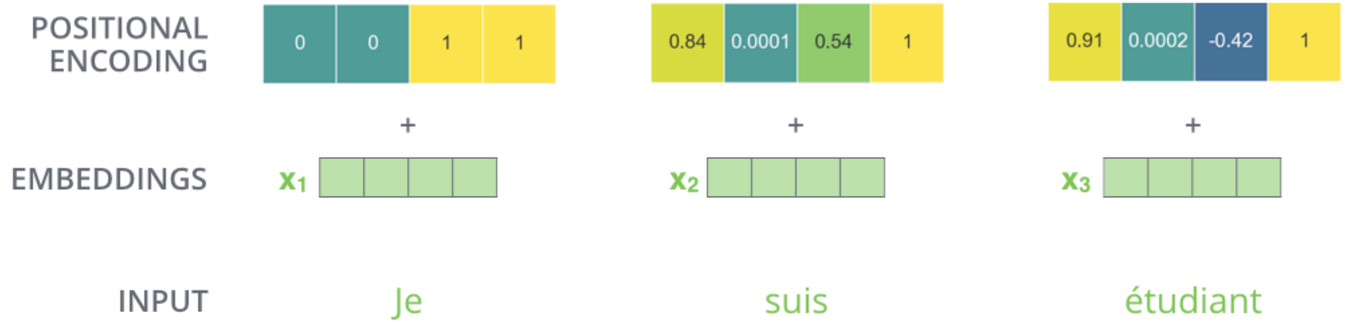
Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$

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

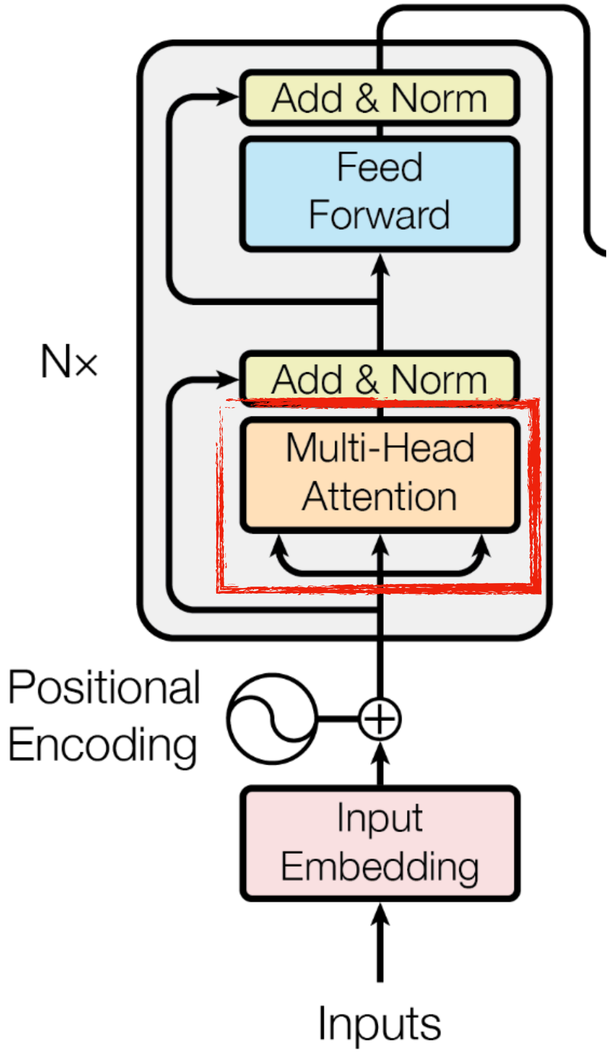
Transformer Encoder



- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding



Transformer Encoder



- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding
- MHA used to redistribute the information among the inputs (contextualization)

Multi-Head Self-Attention

Vaswani, Ashish, et al. "Attention is all you need." (2017)

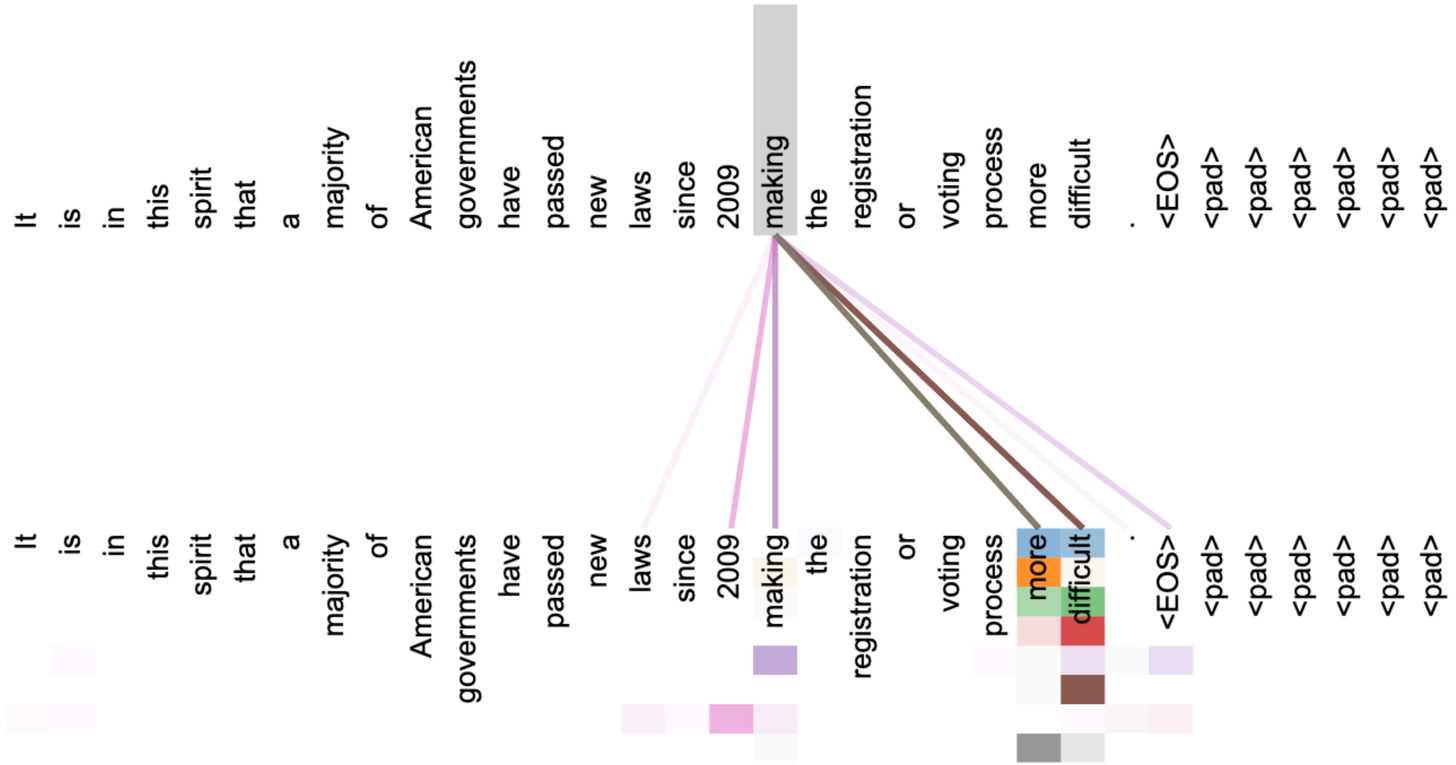
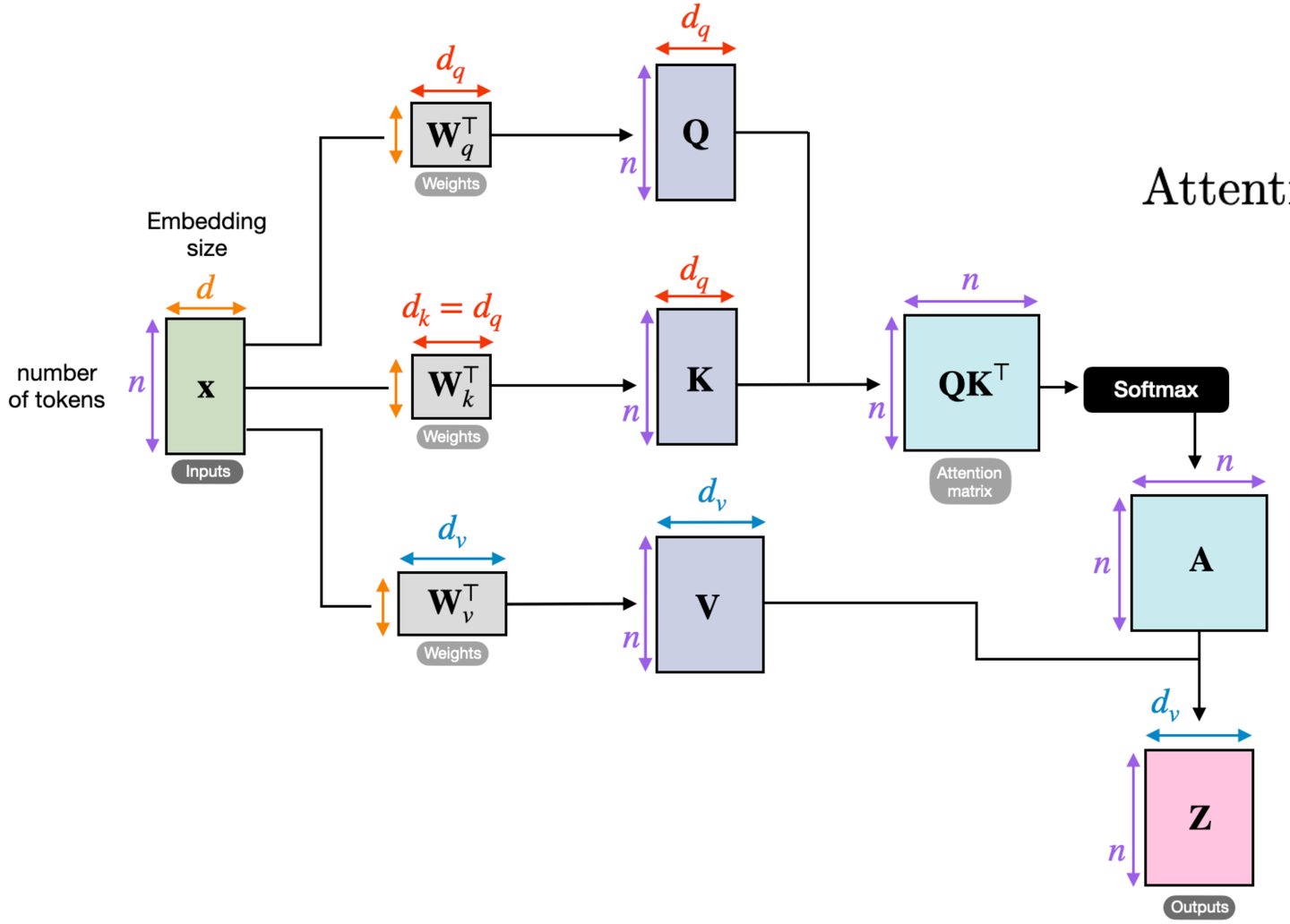


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Multi-Head Self-Attention

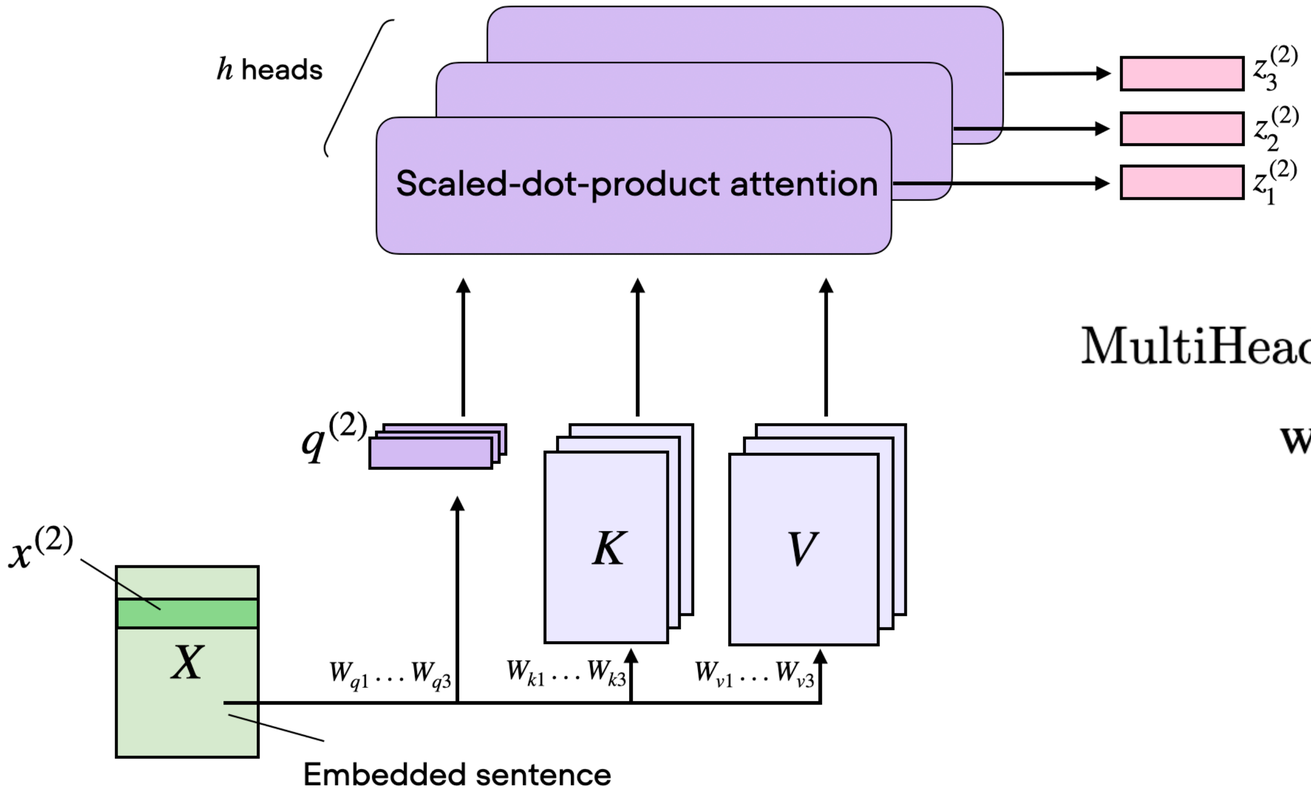
https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html



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

Multi-Head Self-Attention

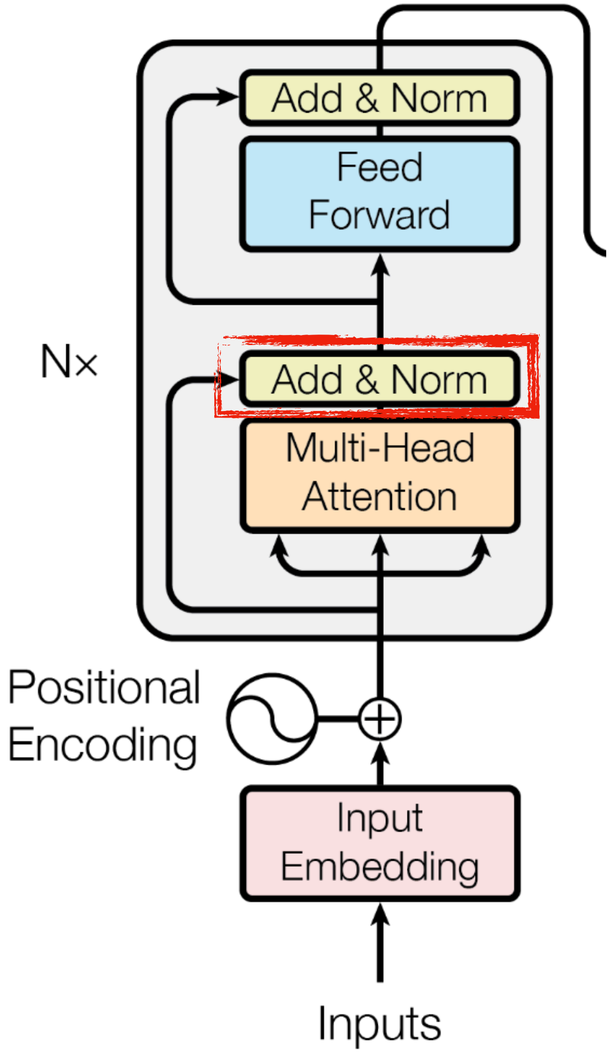
https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

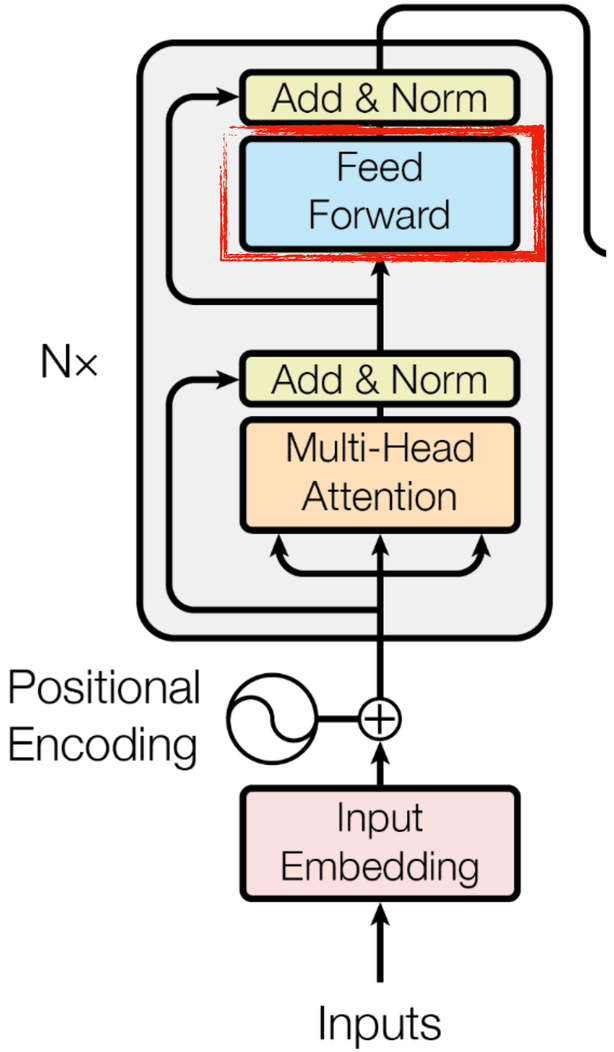
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer Encoder



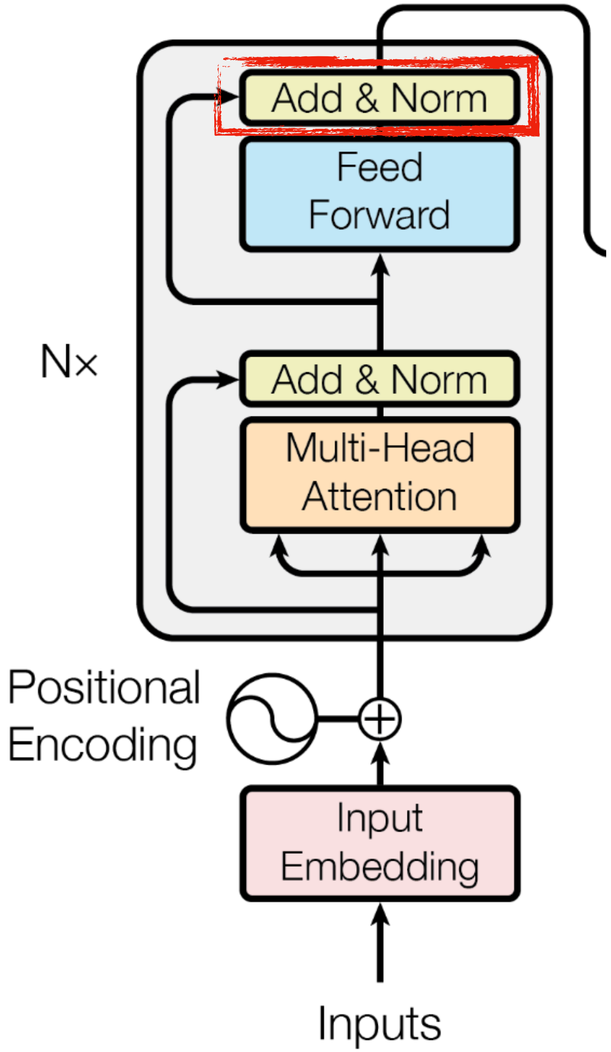
- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding
- MHA used to redistribute the information among the inputs (contextualization)
- Add the input information to the result of MHA and normalize

Transformer Encoder



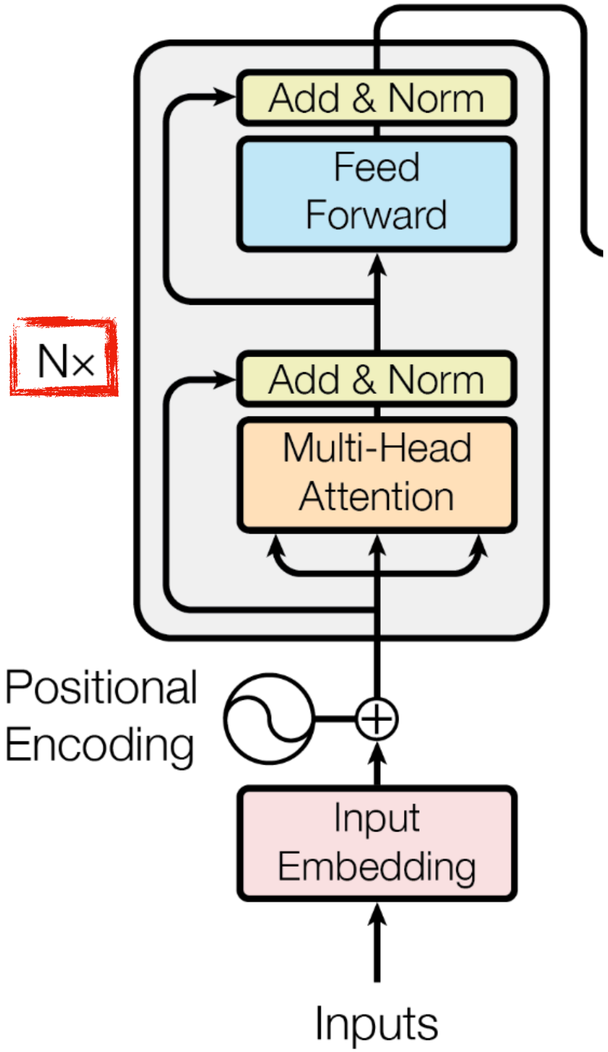
- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding
- MHA used to redistribute the information among the inputs (contextualization)
- Add the input information to the result of MHA and normalize
- Just a regular feed-forward network

Transformer Encoder



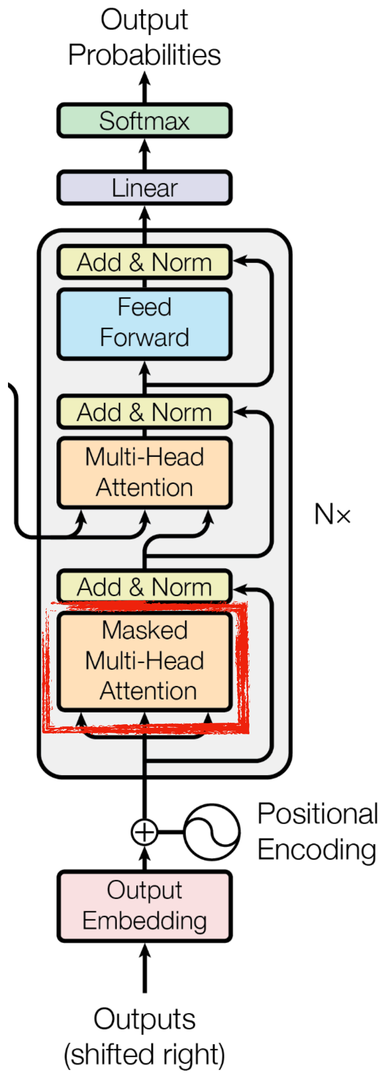
- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding
- MHA used to redistribute the information among the inputs (contextualization)
- Add the input information to the result of MHA and normalize
- Just a regular feed-forward network
- Another residual connection and layer normalization

Transformer Encoder

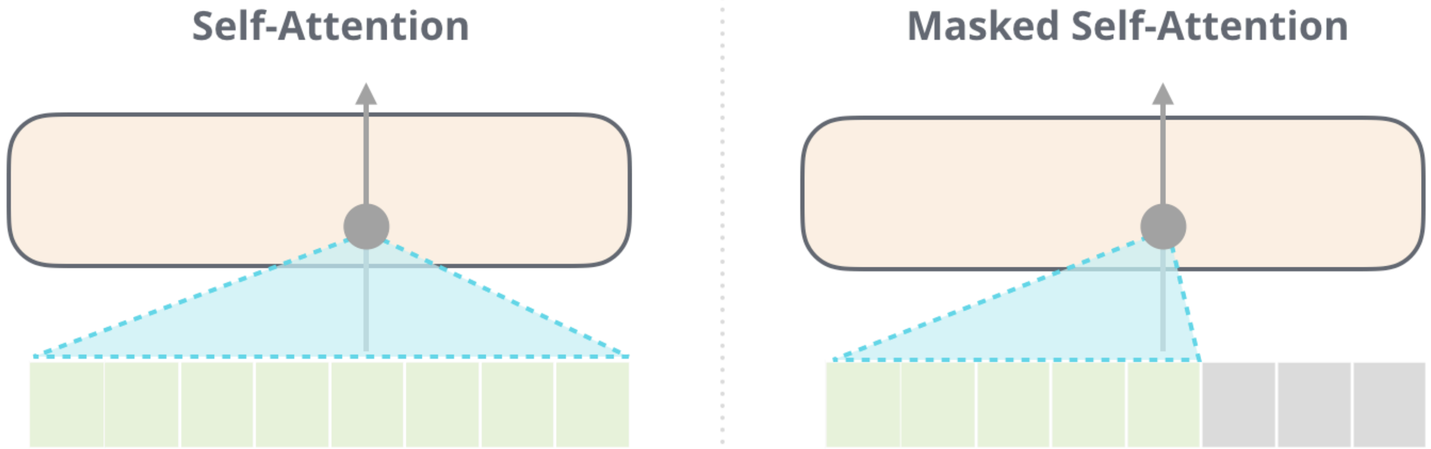


- Byte-pair Encoding (GPT, GPT-2, RoBERTa, BART, and DeBERTa) or WordPiece (BERT) tokenization
- Randomly initialized learnable embedding layer
- Positional encoding to add information about words' position in the sequence
- Positional encoding is added to the embedding
- MHA used to redistribute the information among the inputs (contextualization)
- Add the input information to the result of MHA and normalize
- Just a regular feed-forward network
- Another residual connection and layer normalization
- Repeat N times

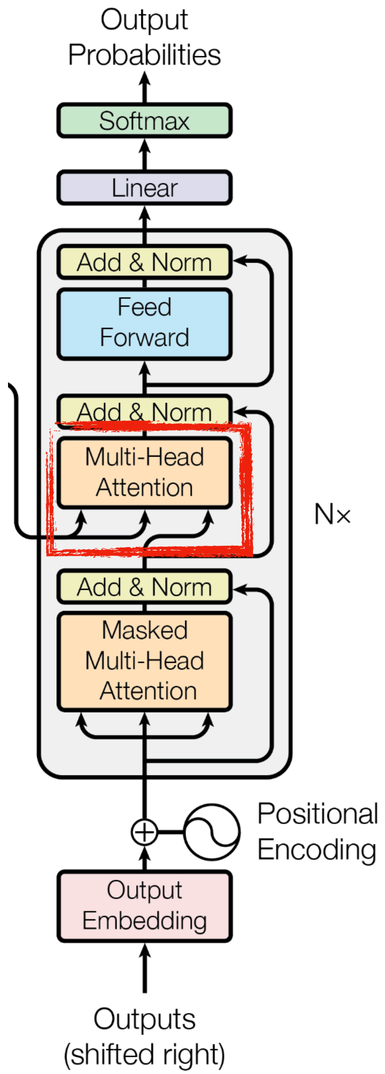
Transformer Decoder



- Has the same elements as encoder
- Masked MHA is used so that the model cannot “look into the future”

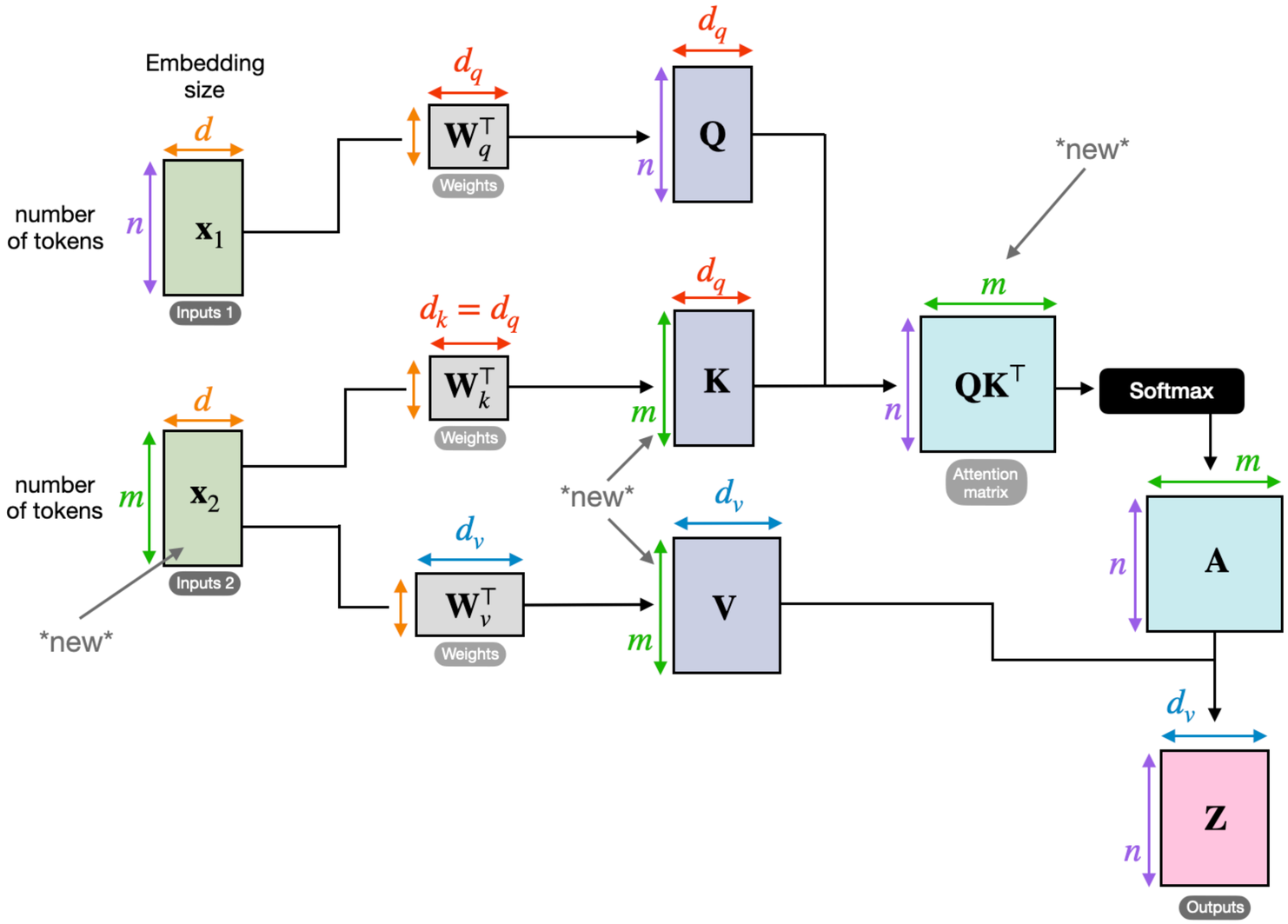


Transformer Decoder

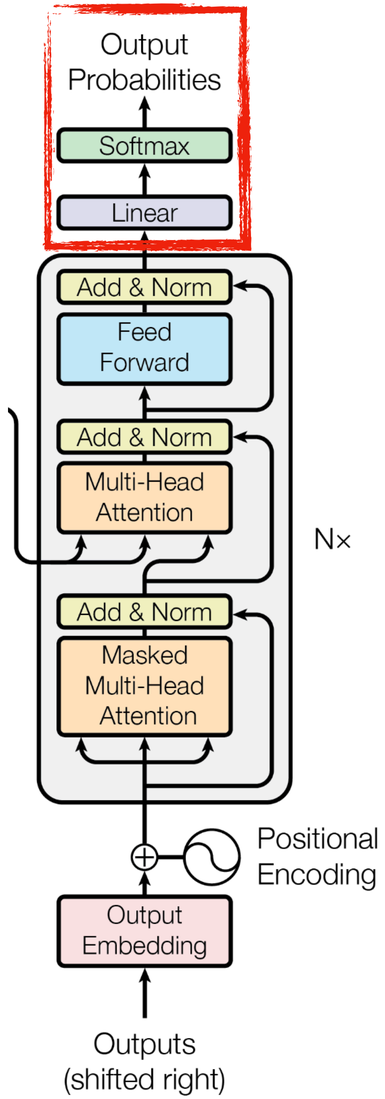


- Has the same elements as encoder
- Masked MHA is used so that the model cannot “look into the future”
- Cross-attention with encoder outputs

Cross-Attention



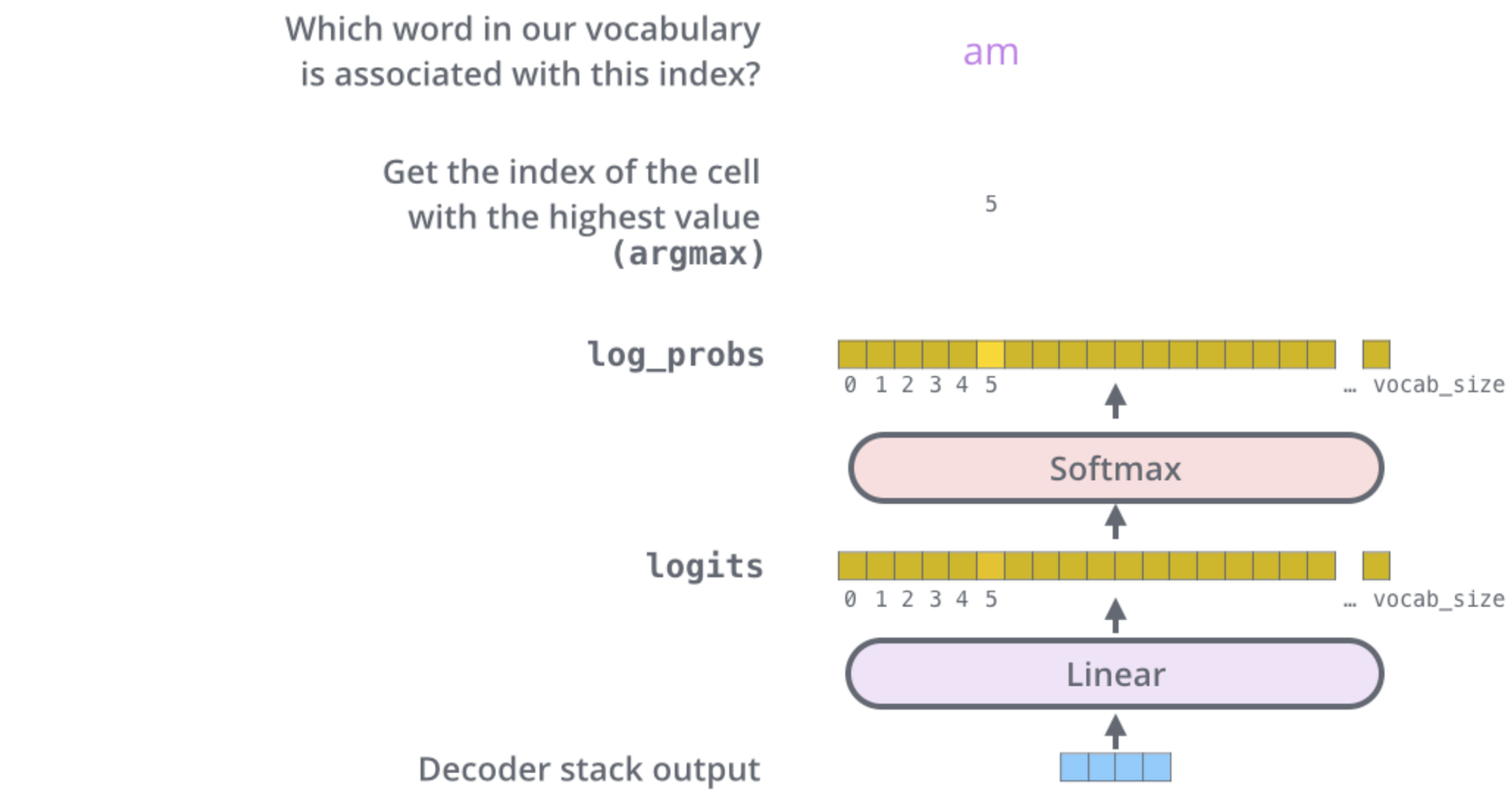
Transformer Decoder



- Has the same elements as encoder
- Masked MHA is used so that the model cannot “look into the future”
- Cross-attention with encoder outputs
- Final layer to output the next token probabilities

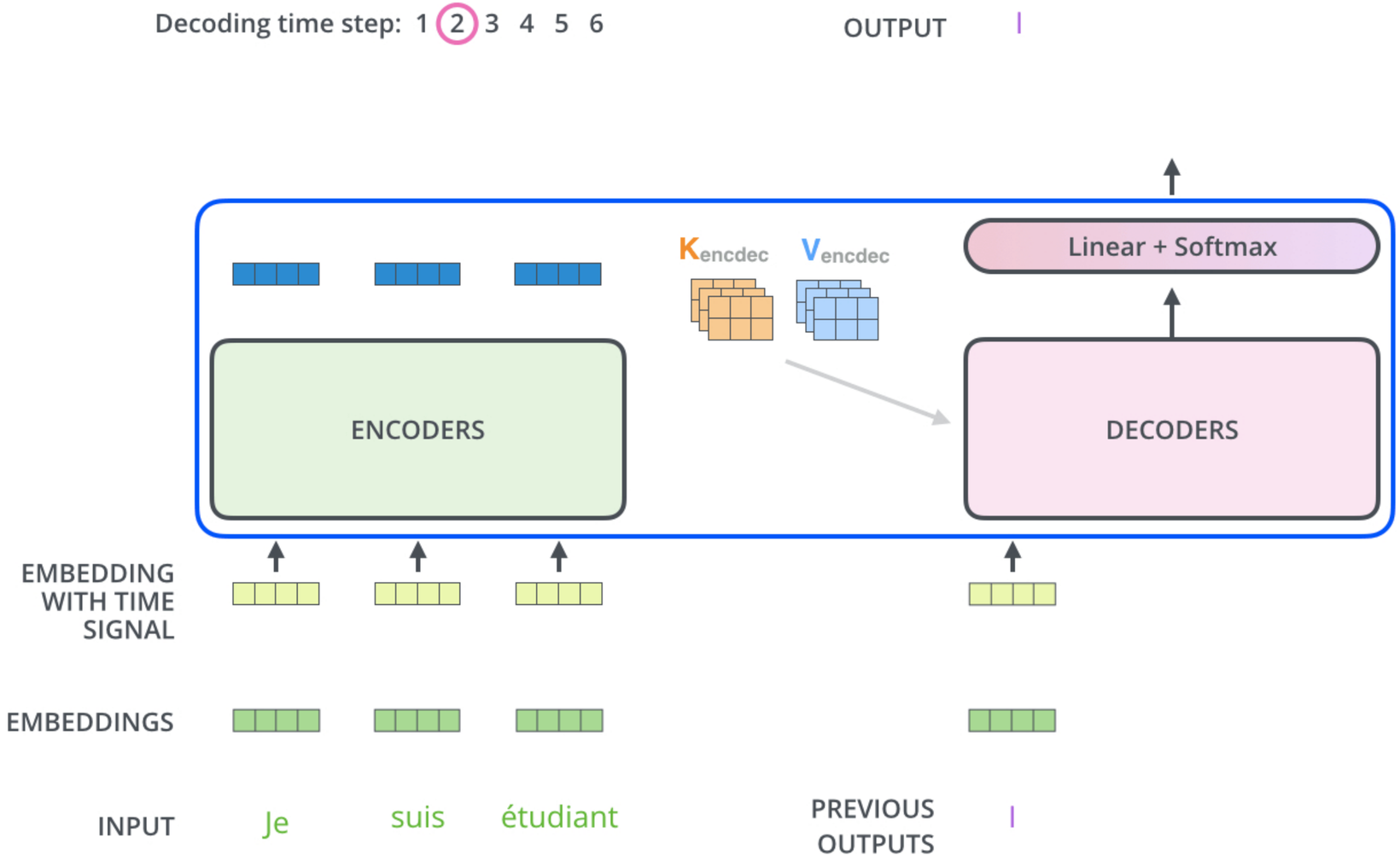
Transformer Final Layer

https://jalammar.github.io/illustrated-transformer/



Encoder-Decoder in Practice

https://jalammar.github.io/illustrated-transformer/

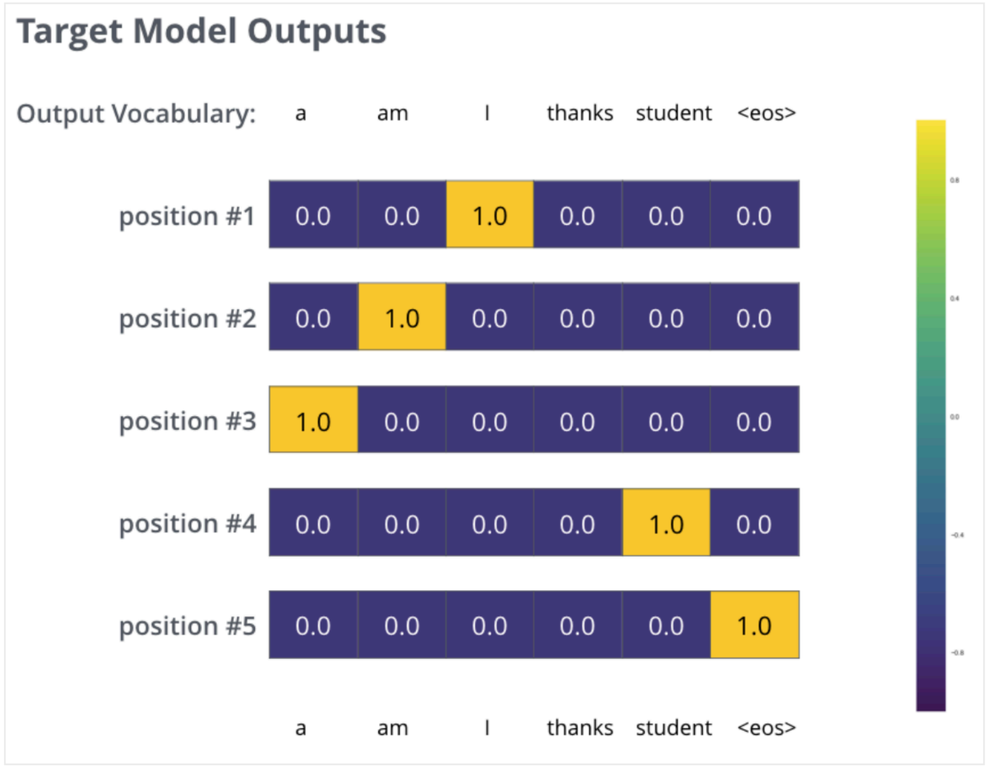


Training

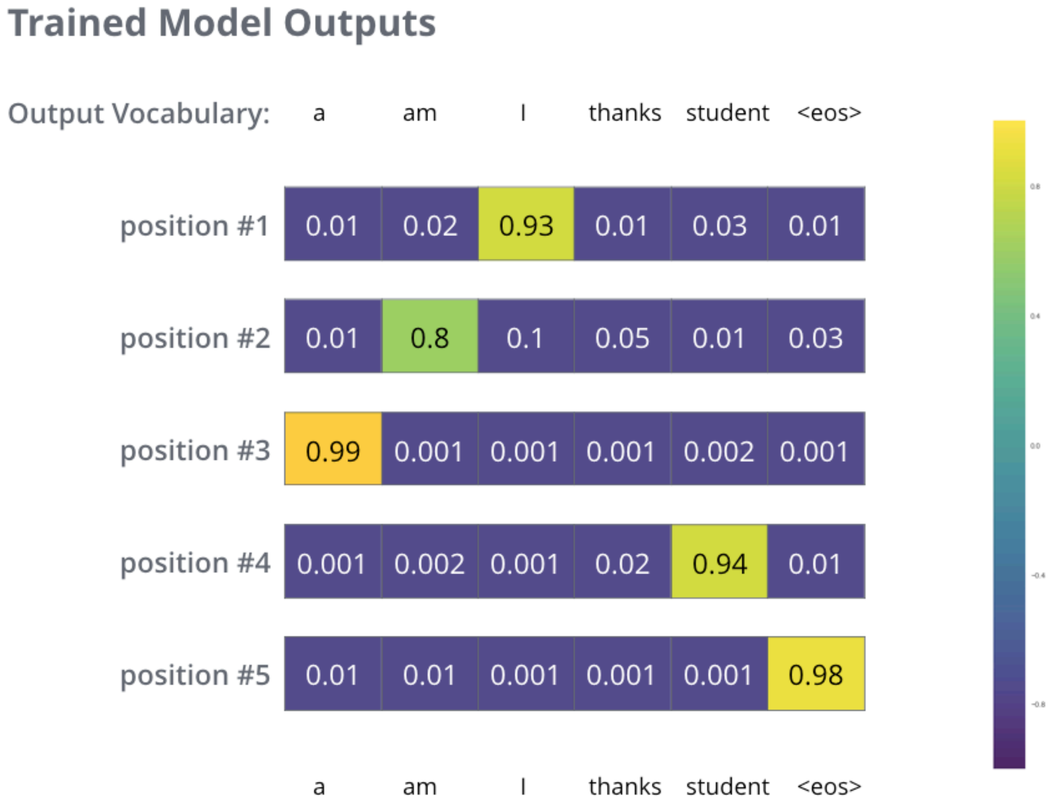
Training

Cross-Entropy Loss

$$H(P^* \mid P) = -\sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTIRBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTIRBUTION}}$$



The targeted probability distributions we'll train our model against in the training example for one sample sentence.



Training

- Usually involves pre-training and fine-tuning
- Pre-training is done on large amount of general purpose texts
- Fine-tuning is on a smaller but more specific corpus

Evaluation

BLEU Score

Mathematically, the BLEU score is defined as:

$$\text{BLEU} = \underbrace{\min\Big(1, \exp\big(1 - \frac{\text{reference-length}}{\text{output-length}}\big)\Big)}_{\text{brevity penalty}} \underbrace{\Big(\prod_{i=1}^{4} precision_i\Big)^{1/4}}_{\text{n-gram overlap}}$$

with

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'} \in \text{Cand-Corpus}} \sum_{i' \in \text{snt'}} m^{i'}_{cand}}$$

where

- m^i_{cand} is the count of i-gram in candidate matching the reference translation
- m^i_{ref} is the count of i-gram in the reference translation
- w^i_t is the total number of i-grams in candidate translation

BLEU Score

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

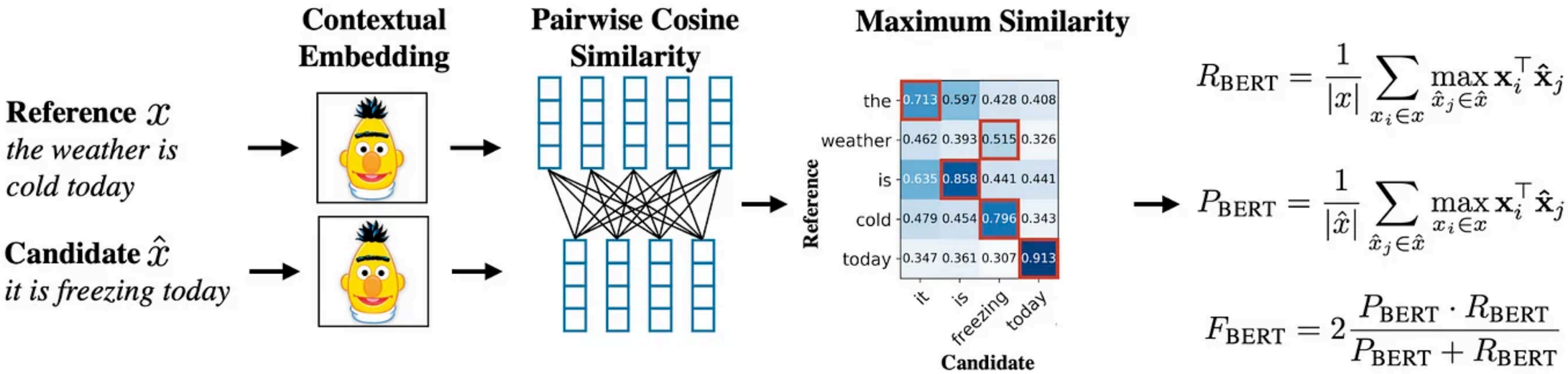
ROUGE Score

$$RECALL = \frac{Overlapping\ number\ of\ n-grams}{Number\ of\ n-grams\ in\ the\ reference}$$

$$PRECISION = \frac{Overlapping\ number\ of\ n-grams}{Number\ of\ n-grams\ in\ the\ candidate}$$

$$F1 = \frac{2 \times Precision \times Recall}{(Precision + Recall)}$$

BERT Score



Source: Bertscore: Evaluating text generation with bert

Examples

Vanilla Transformer

Vaswani, Ashish, et al. "Attention is all you need.” (2017)

- Trained for machine translation
  - WMT 2014 English-German dataset (4.5 million sentence pairs)
  - WMT 2014 English-French dataset (36 million sentence pairs)

Vanilla Transformer

Results

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

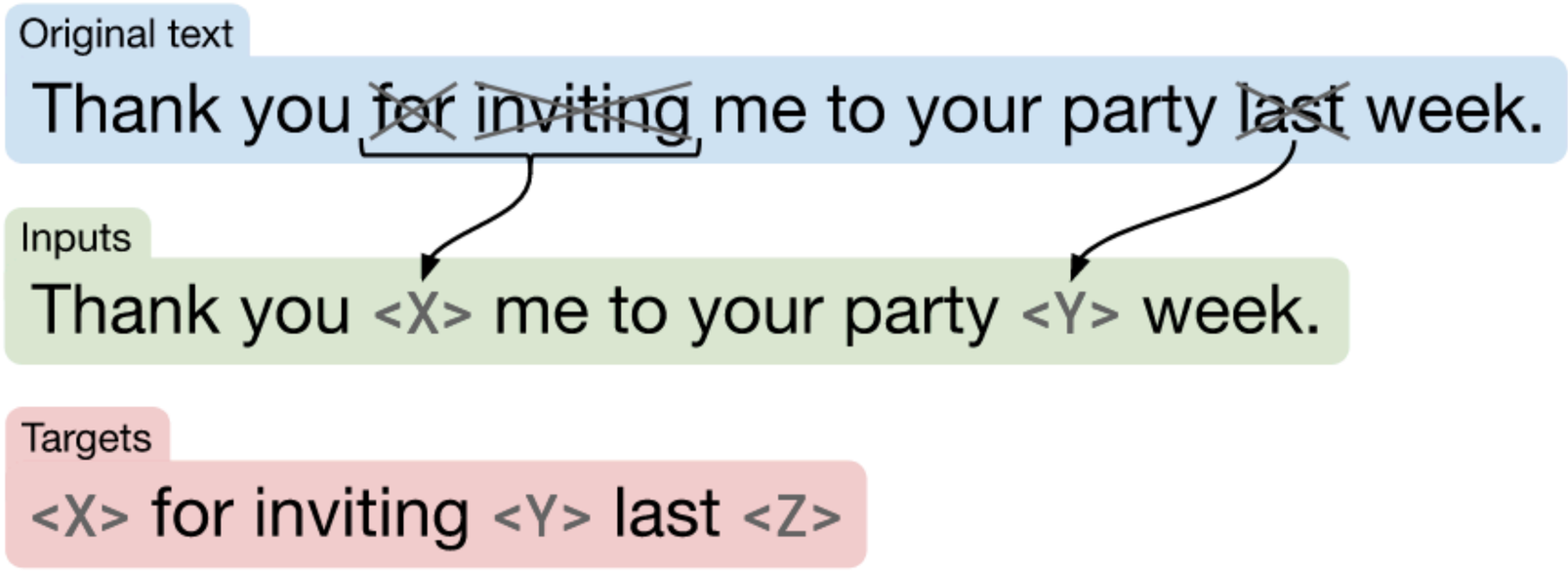
T5

Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer.” (2020)

- Encoder-decoder based large language model (LLM)
- Pre-trained on Colossal Clean Crawled Corpus (C4) - about 750 GB of cleaned English text (about 34 billion tokens)
- Use span-corruption objective for pre-training
- Fine-tuning on GLUE and SuperGLUE tasks

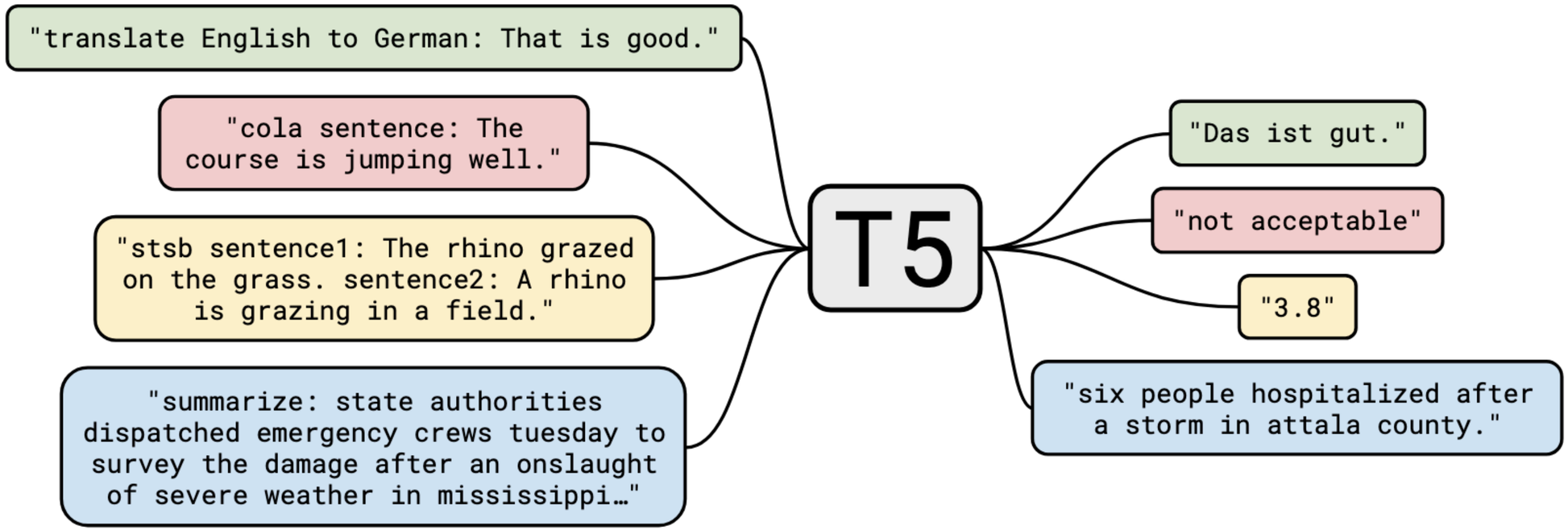
T5

Pre-training



T5

Fine-tuning



T5

Results

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	74.8[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	**75.1**	90.6	**92.2**	**91.9**	96.9	**92.8**	**94.5**

Model	SQuAD EM	SQuAD F1	SuperGLUE Average	BoolQ Accuracy	CB F1	CB Accuracy	COPA Accuracy
Previous best	90.1[a]	95.5[a]	84.6[d]	87.1[d]	90.5[d]	95.2[d]	90.6[d]
T5-Small	79.10	87.24	63.3	76.4	56.9	81.6	46.0
T5-Base	85.44	92.08	76.2	81.4	86.2	94.0	71.2
T5-Large	86.66	93.79	82.3	85.4	91.6	94.8	83.4
T5-3B	88.53	94.95	86.4	89.9	90.3	94.4	92.0
T5-11B	**91.26**	**96.22**	**88.9**	**91.2**	**93.9**	**96.8**	**94.8**

Model	MultiRC F1a	MultiRC EM	ReCoRD F1	ReCoRD Accuracy	RTE Accuracy	WiC Accuracy	WSC Accuracy
Previous best	84.4[d]	52.5[d]	90.6[d]	90.0[d]	88.2[d]	69.9[d]	89.0[d]
T5-Small	69.3	26.3	56.3	55.4	73.3	66.9	70.5
T5-Base	79.7	43.1	75.0	74.2	81.5	68.3	80.8
T5-Large	83.3	50.7	86.8	85.9	87.8	69.3	86.3
T5-3B	86.8	58.3	91.2	90.4	90.7	72.1	90.4
T5-11B	**88.1**	**63.3**	**94.1**	**93.4**	**92.5**	**76.9**	**93.8**

Model	WMT EnDe BLEU	WMT EnFr BLEU	WMT EnRo BLEU	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-L
Previous best	**33.8**[e]	**43.8**[e]	**38.5**[f]	43.47[g]	20.30[g]	40.63[g]
T5-Small	26.7	36.0	26.8	41.12	19.56	38.35
T5-Base	30.9	41.2	28.0	42.05	20.34	39.40
T5-Large	32.0	41.5	28.1	42.50	20.68	39.75
T5-3B	31.8	42.6	28.2	42.72	21.02	39.94
T5-11B	32.1	43.4	28.1	**43.52**	**21.55**	**40.69**

FlanT5

Chung, Hyung Won, et al. "Scaling instruction-finetuned language models." (2022)

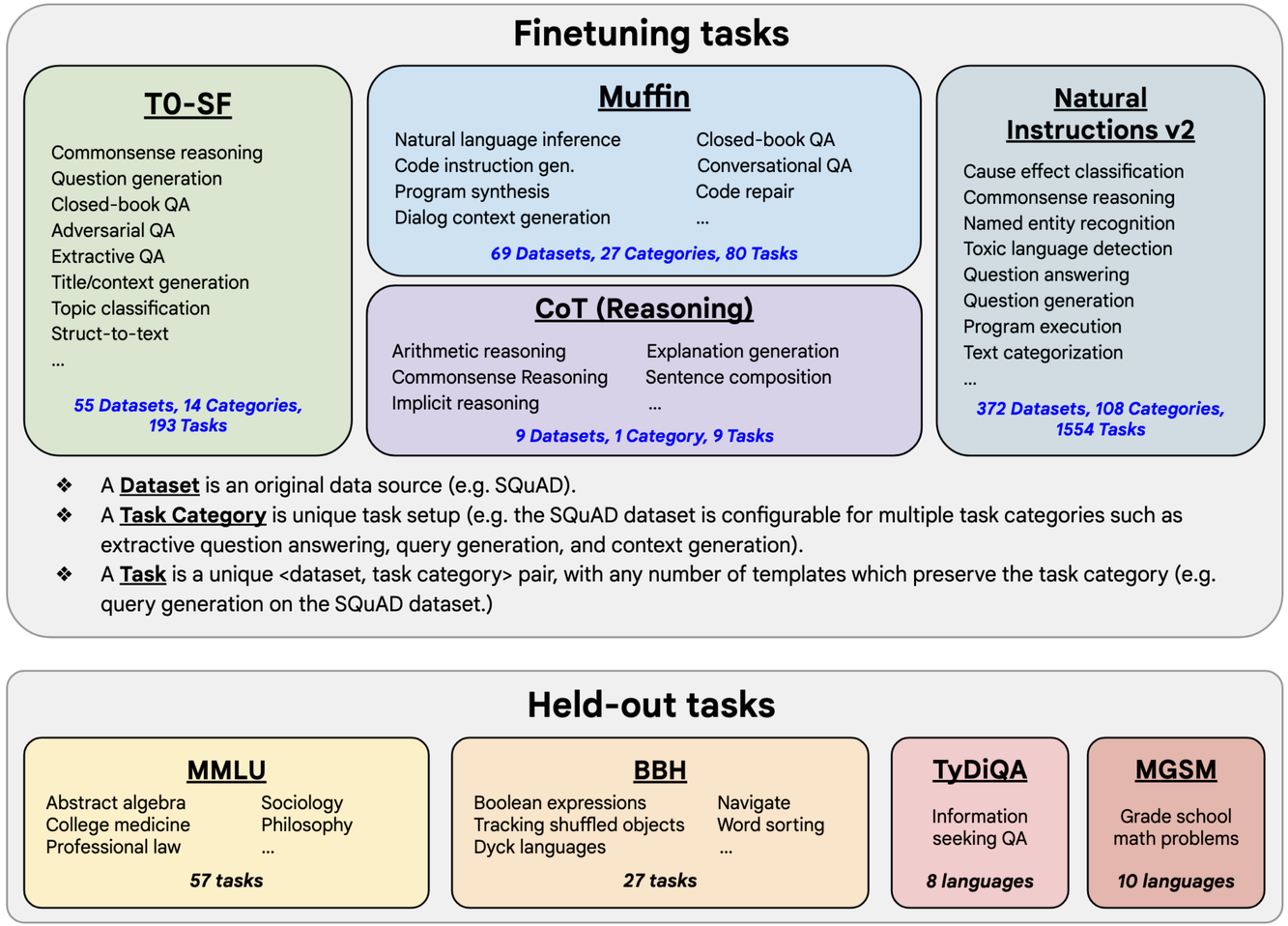


Figure 2: Our finetuning data comprises 473 datasets, 146 task categories, and 1,836 total tasks. Details for the tasks used in this paper is given in Appendix F.

FlanT5

Fine-tuning

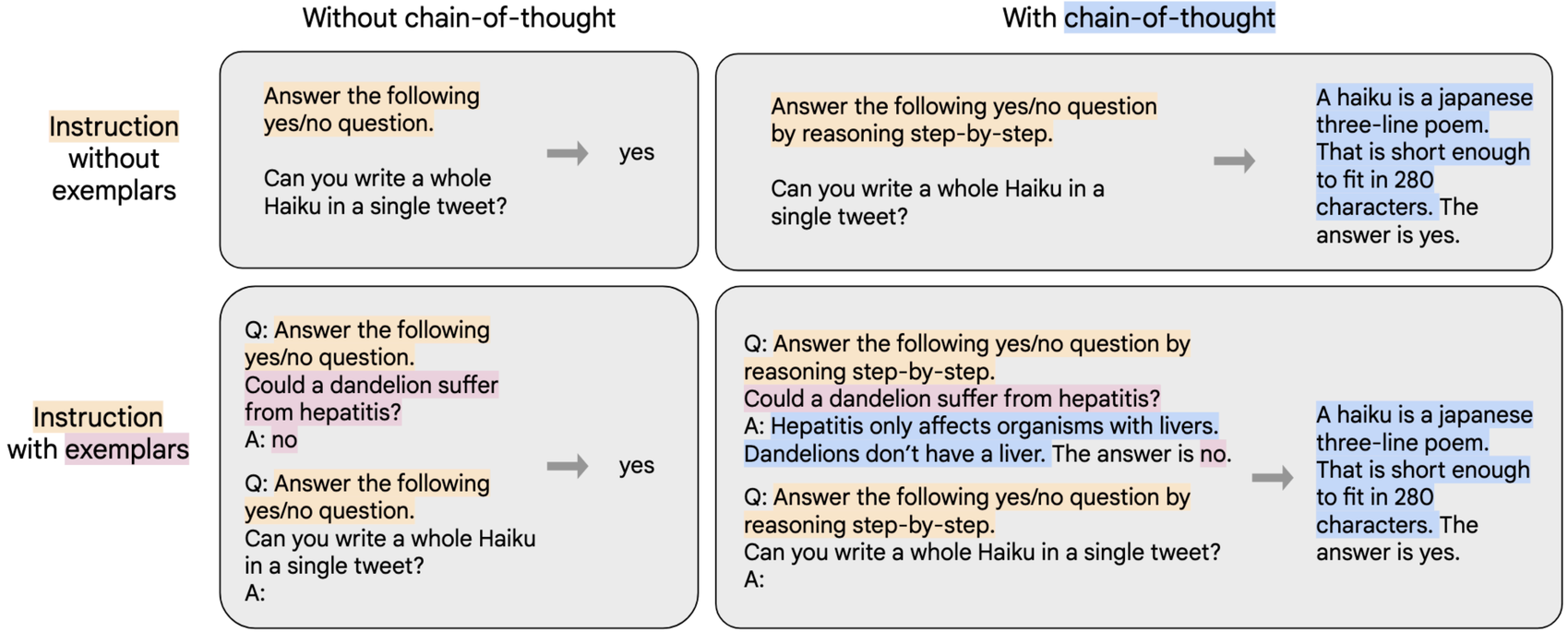
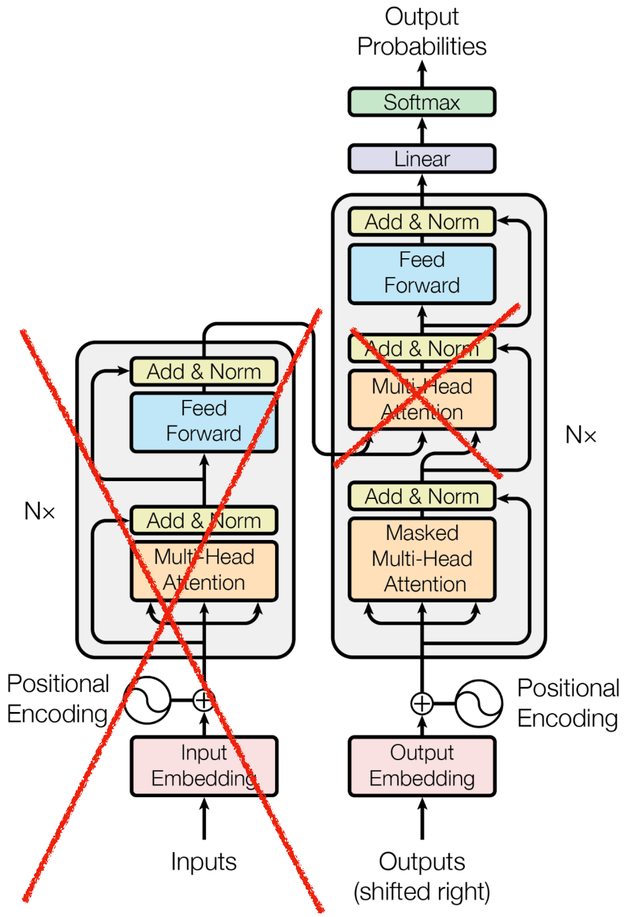


Figure 3: Combinations of finetuning data formats in this work. We finetune with and without exemplars, and also with and without chain-of-thought. In addition, we have some data formats without instructions but with few-shot exemplars only, like in Min et al. (2022) (not shown in the figure). Note that only nine chain-of-thought (CoT) datasets use the CoT formats.

GPT-2

Radford, Alec, et al. "Language models are unsupervised multitask learners.” (2019)



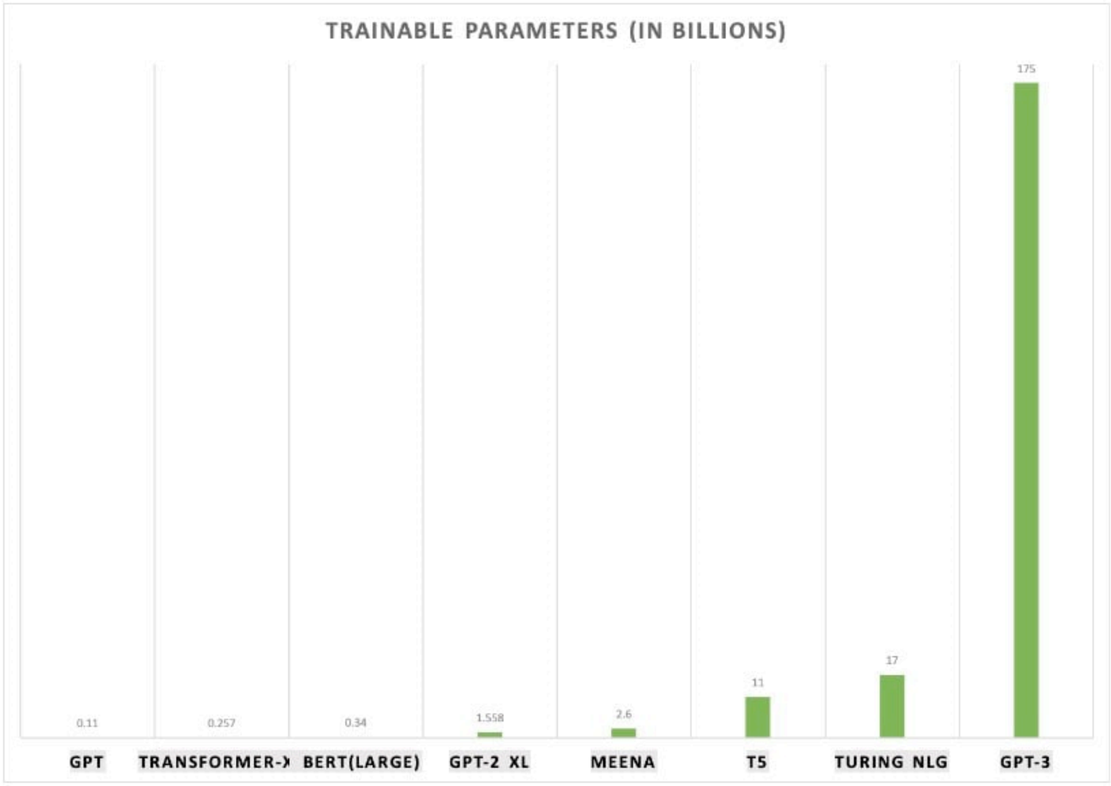
GPT-2

- Decoder-only large language model (LLM)
- Pre-trained on WebText (8 million documents ~ 40 GB of text)
- Some differences with the Vanilla Transformer
  - Layer normalization was moved to the input of each sub-block
  - Additional layer normalization was added after the final self-attention block

GPT-3

Brown, Tom, et al. "Language models are few-shot learners.” (2020)

- Same architecture as GPT-2 but much (much!) bigger
- A huge amount of training data (from 550GB to



Instruct GPT

Ouyang, Long, et al. "Training language models to follow instructions with human feedback." (2022)

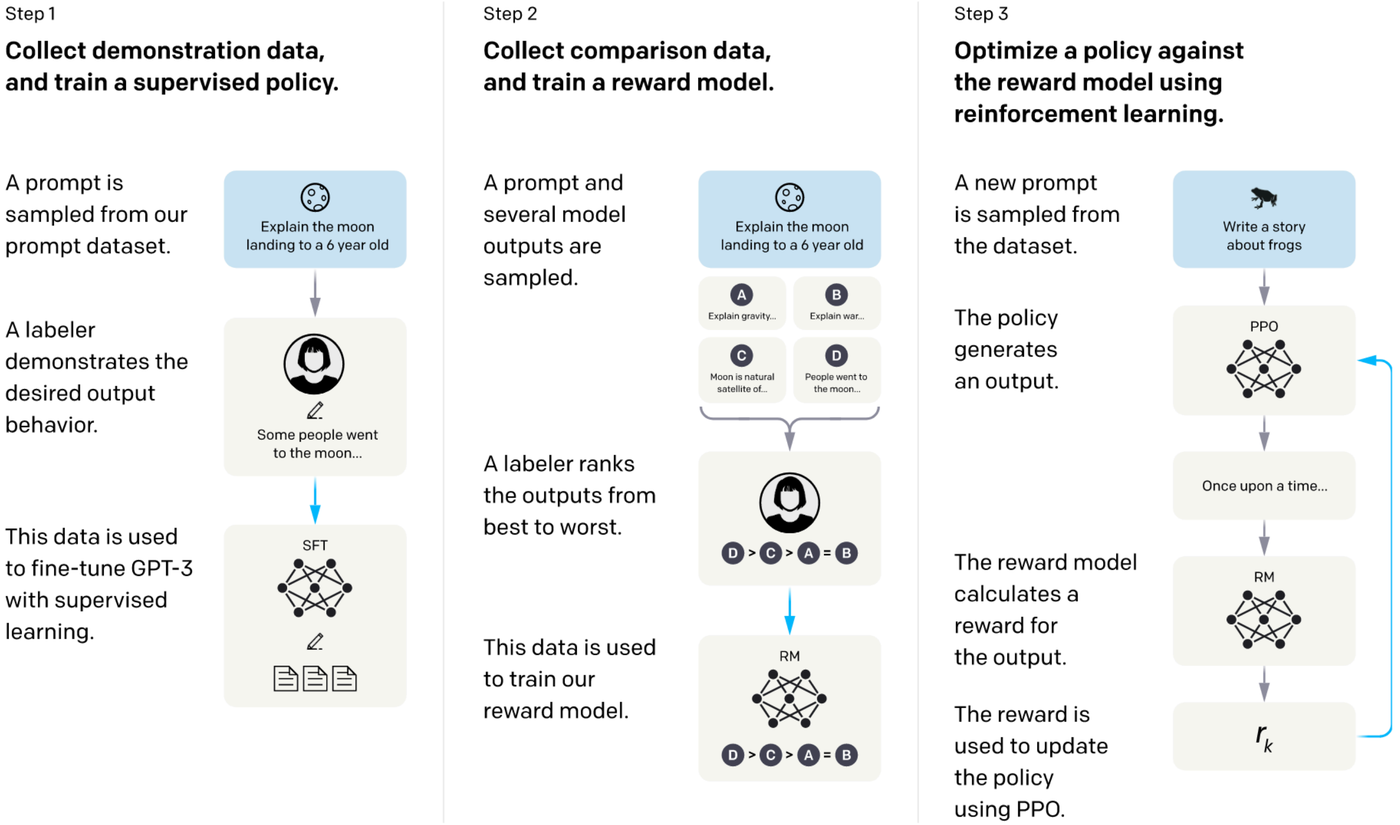


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

LLaMA

- Large training dataset (1.4 trillion tokens)
- Differences with Vanilla Transformer:
  - Pre-normalization (like GPT-3)
  - SwiGLU activation function instead of ReLU
  - Rotary embeddings instead of positional embeddings

Mistral

- Dataset is unknown
- Architecture similar to LLaMA with several changes:
  - Sliding window attention that allows input and output sequences up to 131K tokens
  - Various optimization techniques

Conclusion

- NLG is an active research field
- A lot of current NLP tasks can be solved with text generation
- Task-specific models are still better for many tasks but large language models are more versatile and generalizable
- Entry threshold to the LLM is very high (required a lot of recourses to train and use)