Deep Learning
Day 13: Transformers and Large Language Models

Eric Ewing

CSCI 1470

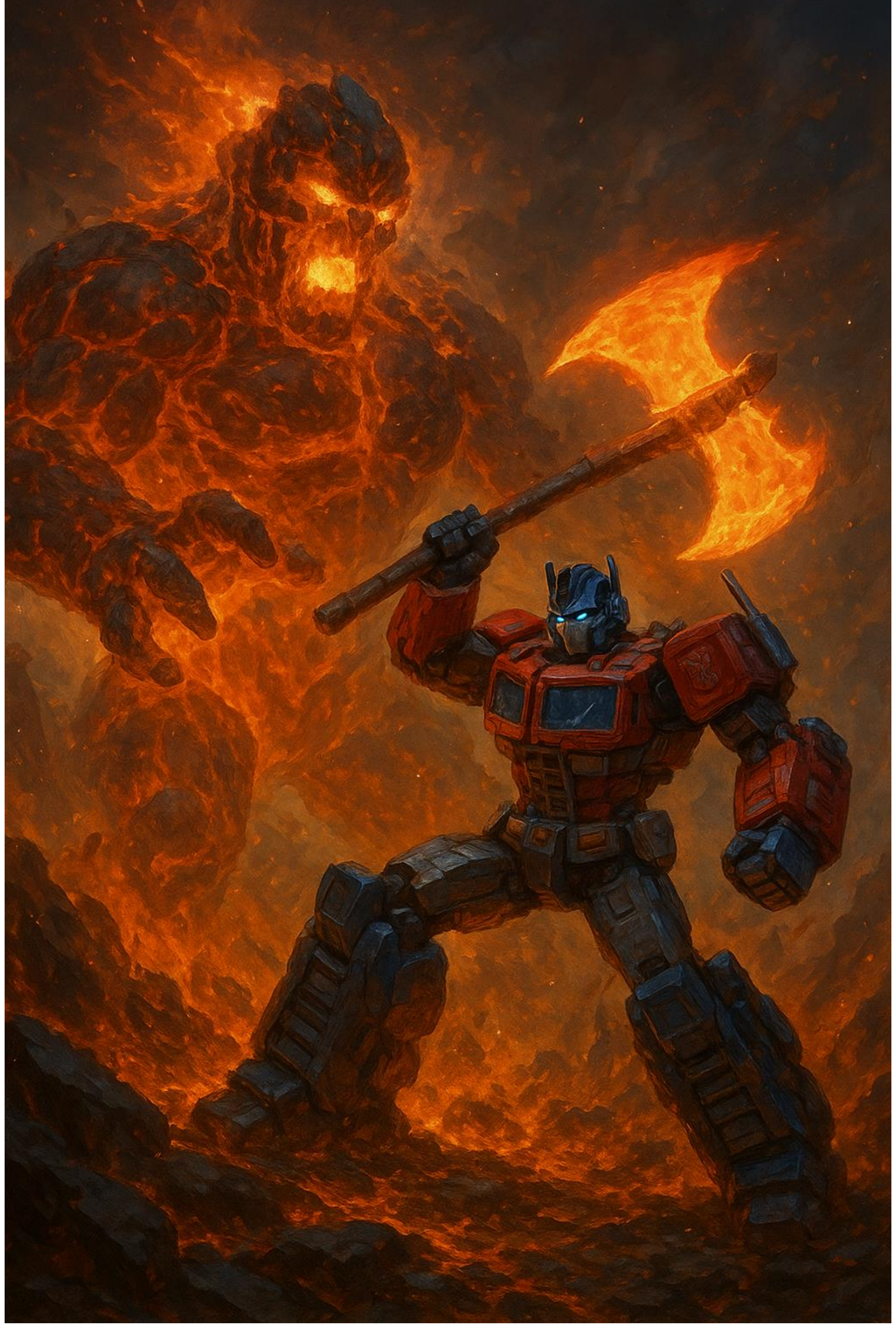


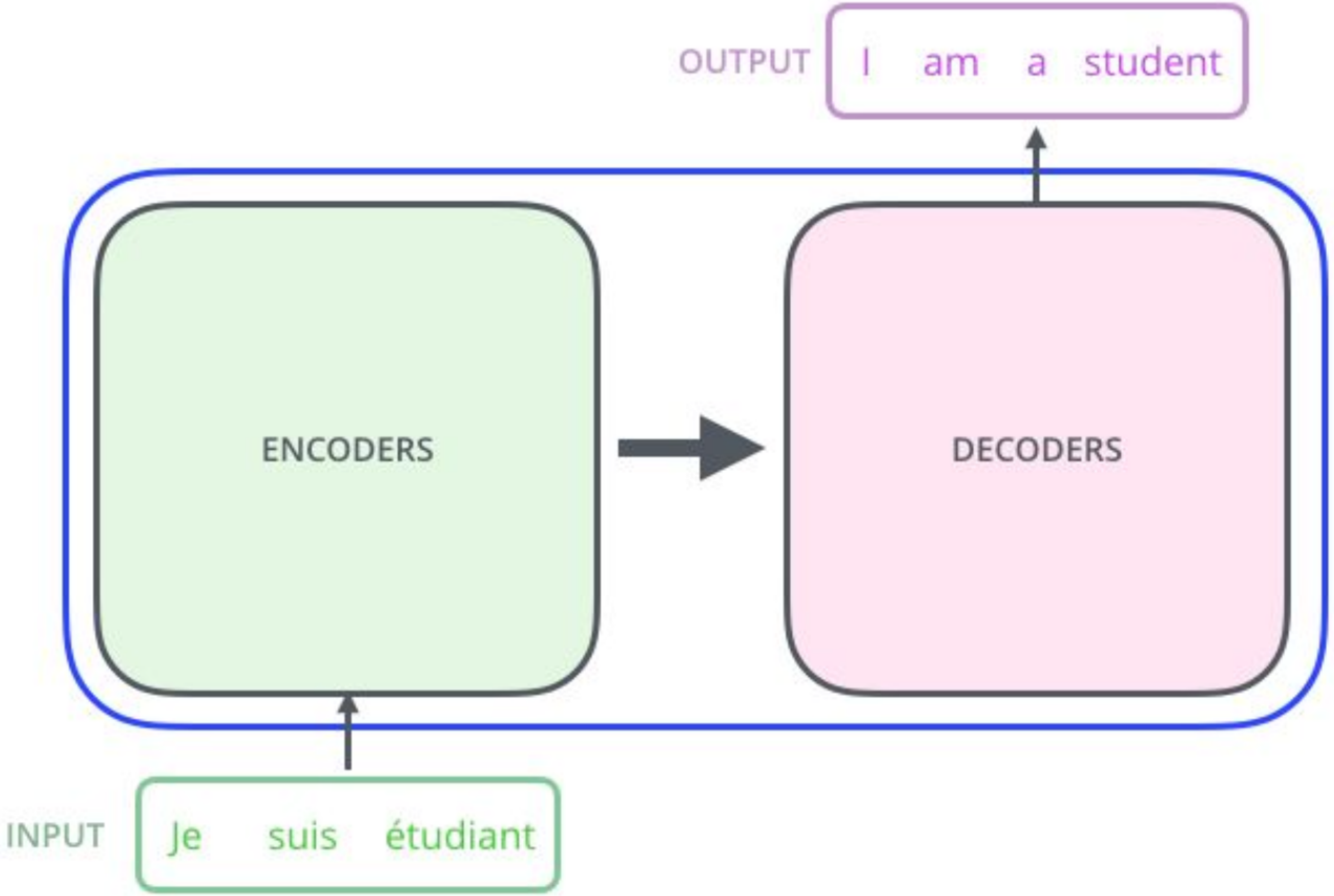
Image generated by ChatGPT 5: Can you make me an image of Optimus Prime fighting a large geologic entity

Transformer Model Overview

- The Transformer model breaks down into Encoder and Decoder blocks.

- At a high level, similar to the seq2seq architecture we've seen already...

- ...but there are no recurrent nets inside the Encoder and Decoder blocks!



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

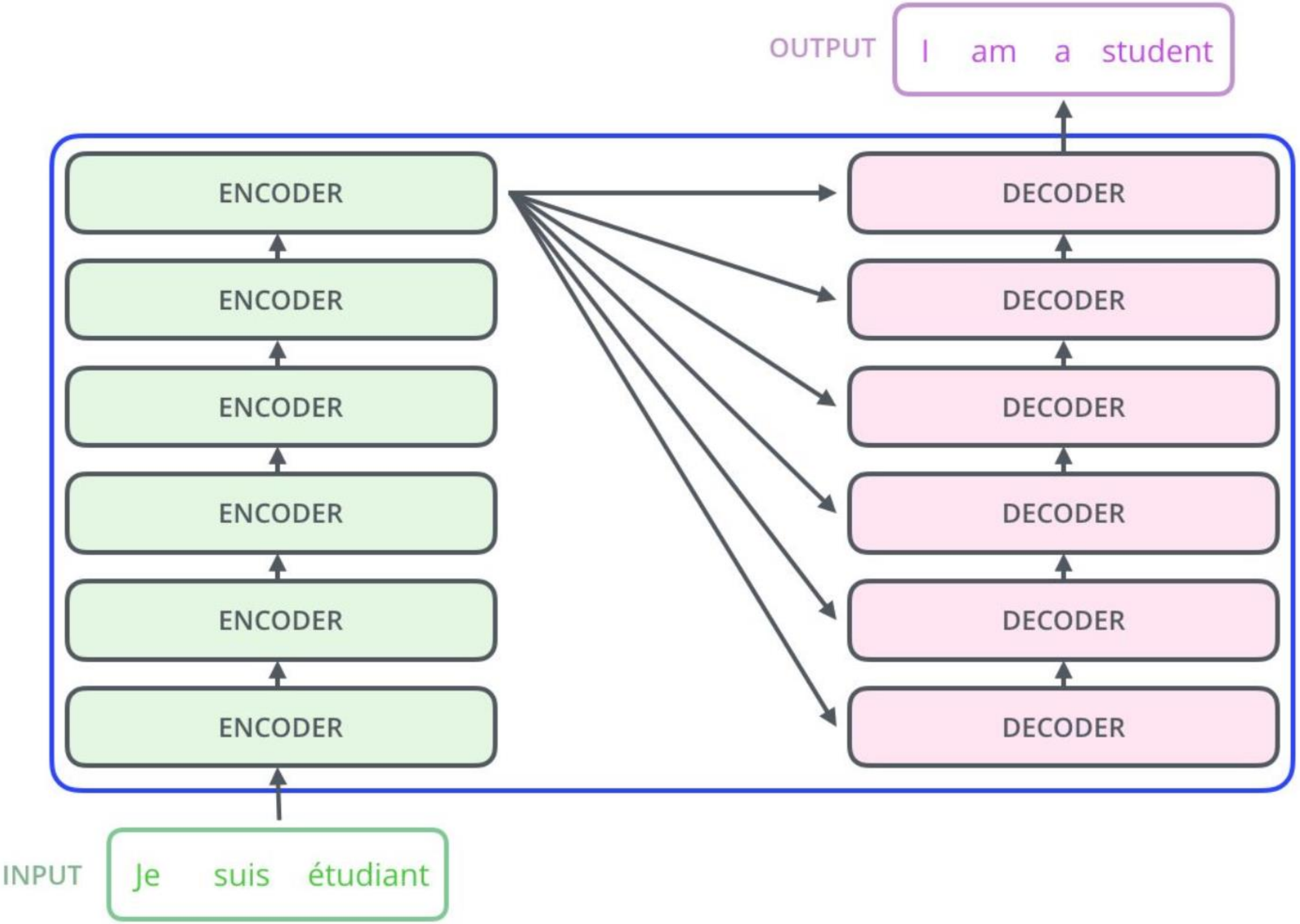
Transformer Model Overview

- The Transformer model breaks down into Encoder and Decoder blocks.

- At a high level, similar to the seq2seq architecture we've seen already...

- ...but there are no recurrent nets inside the Encoder and Decoder blocks!

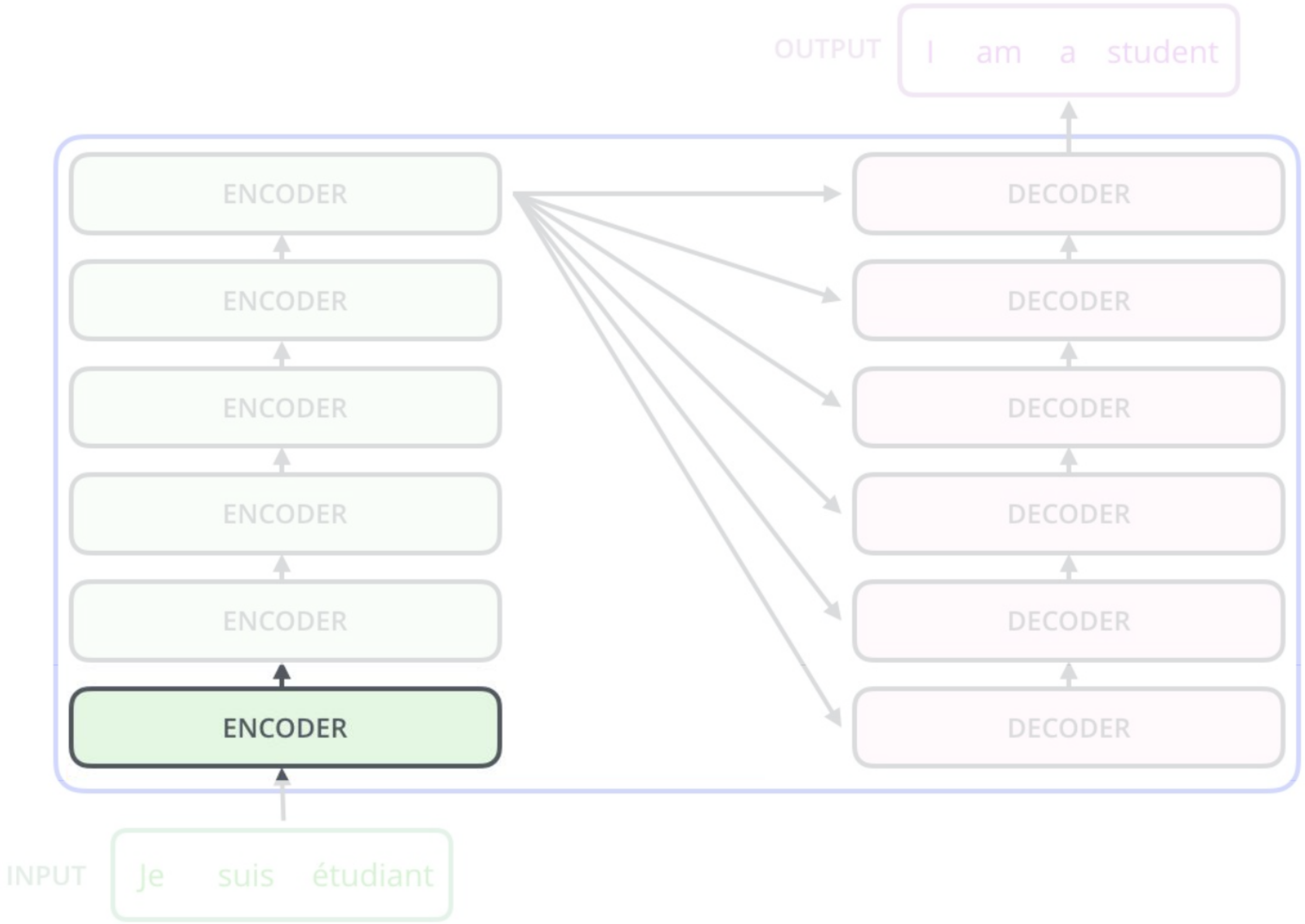
- For better performance, often stack multiple Encoder and Decoder blocks (deeper network)



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Transformer Model Overview

- Let's look at what goes on inside one of these Encoder blocks



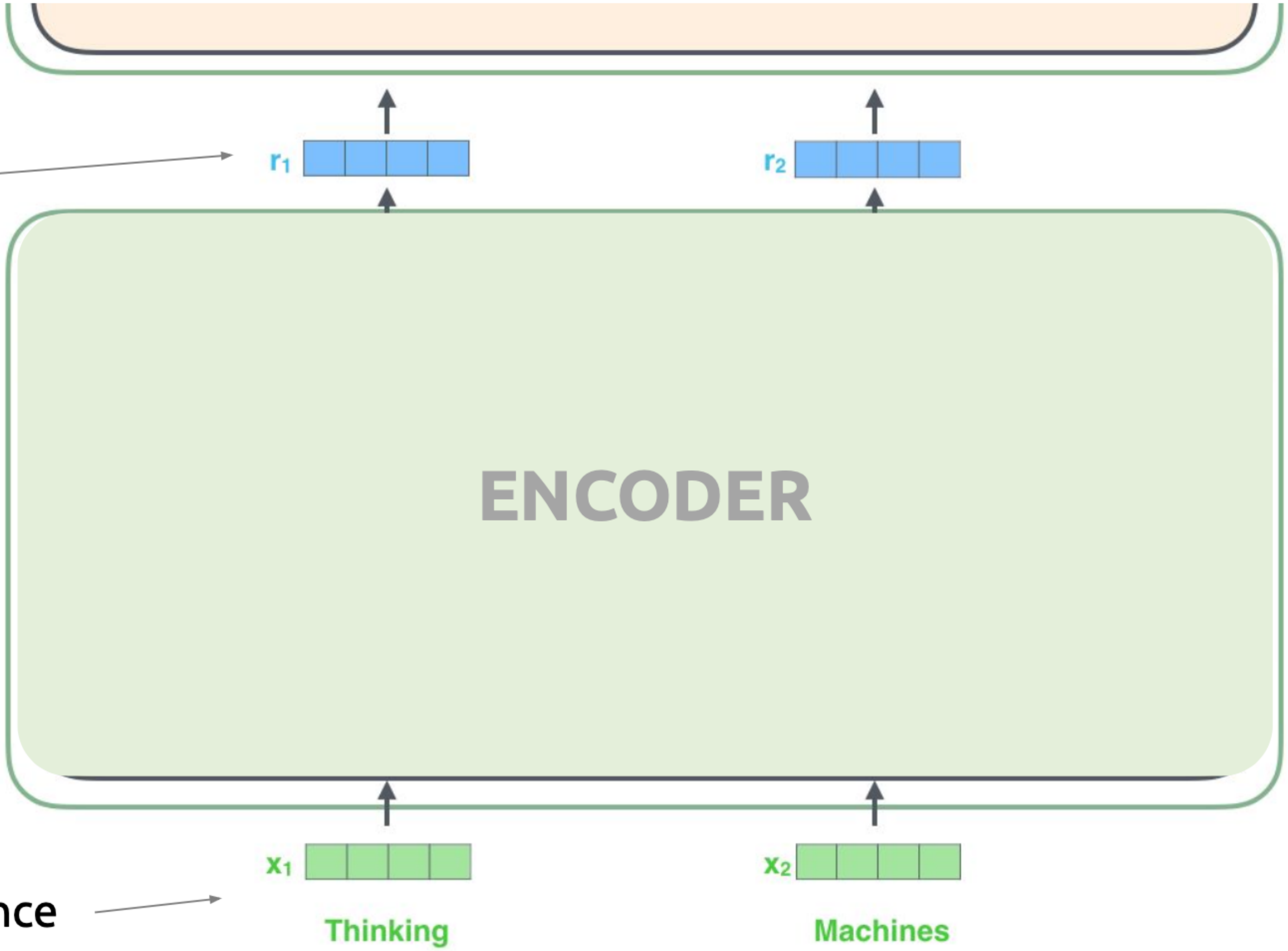
Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Encoder Block Map

These per-word output vectors are analogous to the LSTM hidden states from the seq2seq2 model

- They should capture *"what information about the input sentence is relevant to translating this word?"*

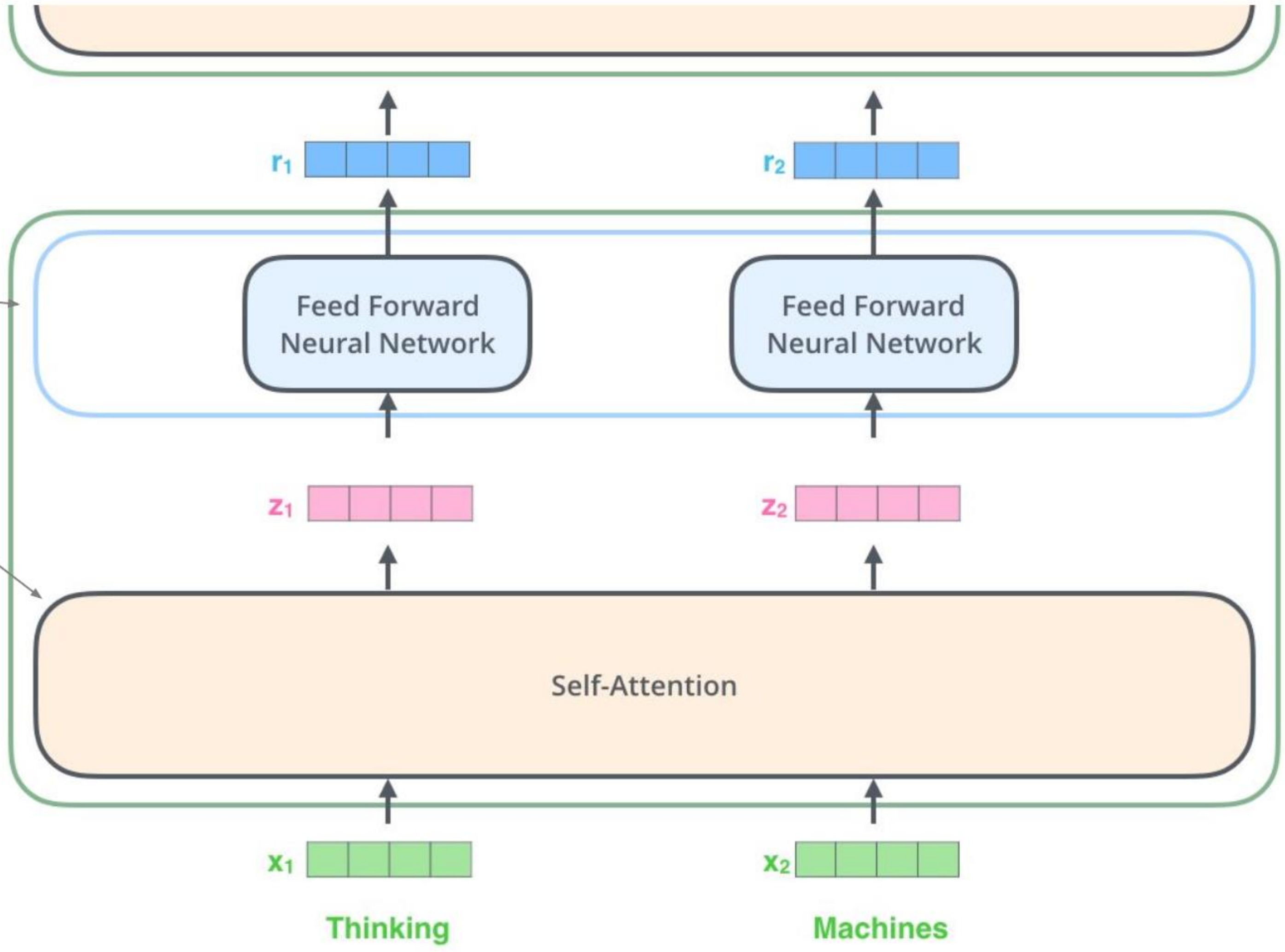
Words in input sentence



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Encoder Block Map

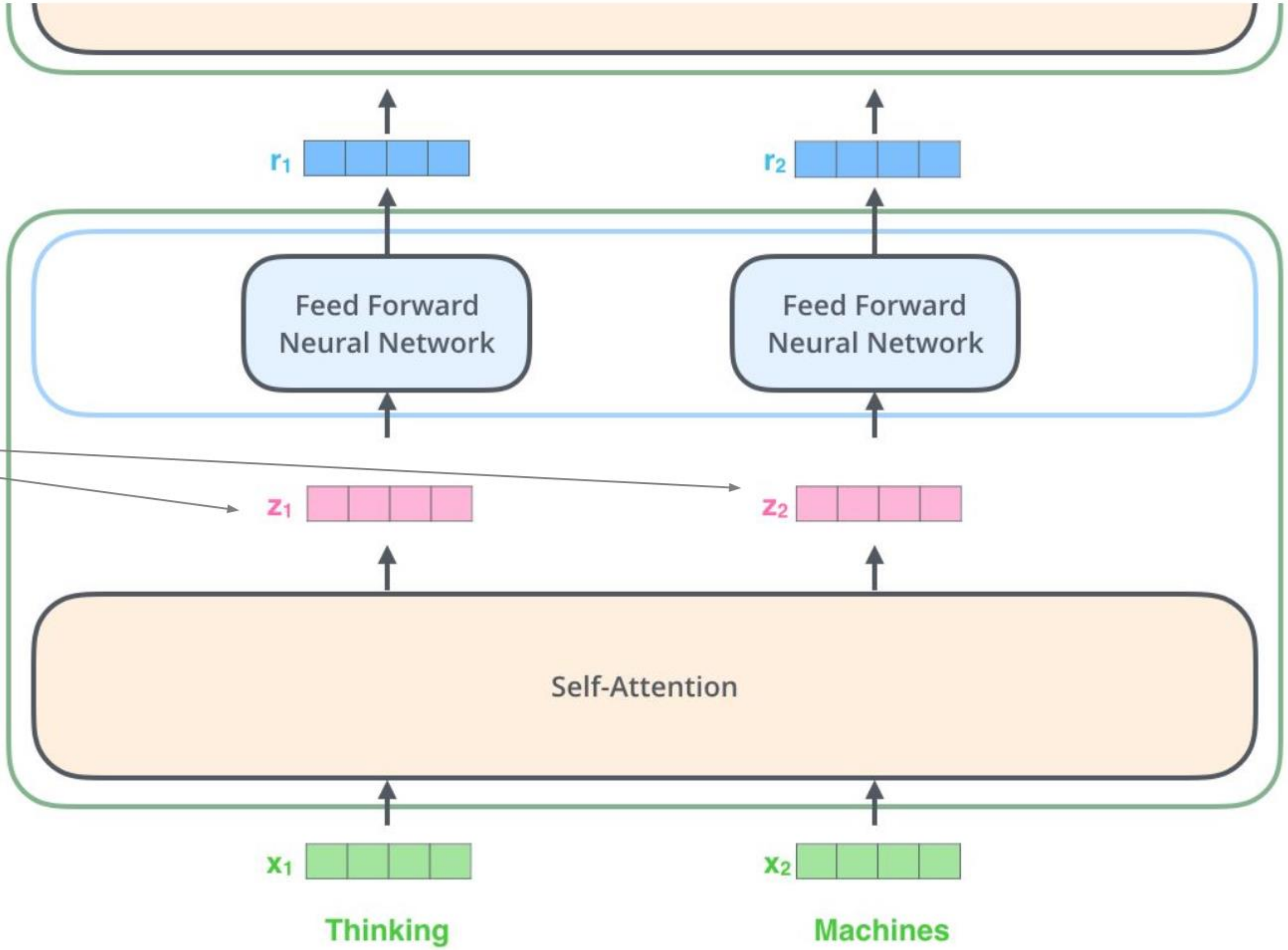
- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

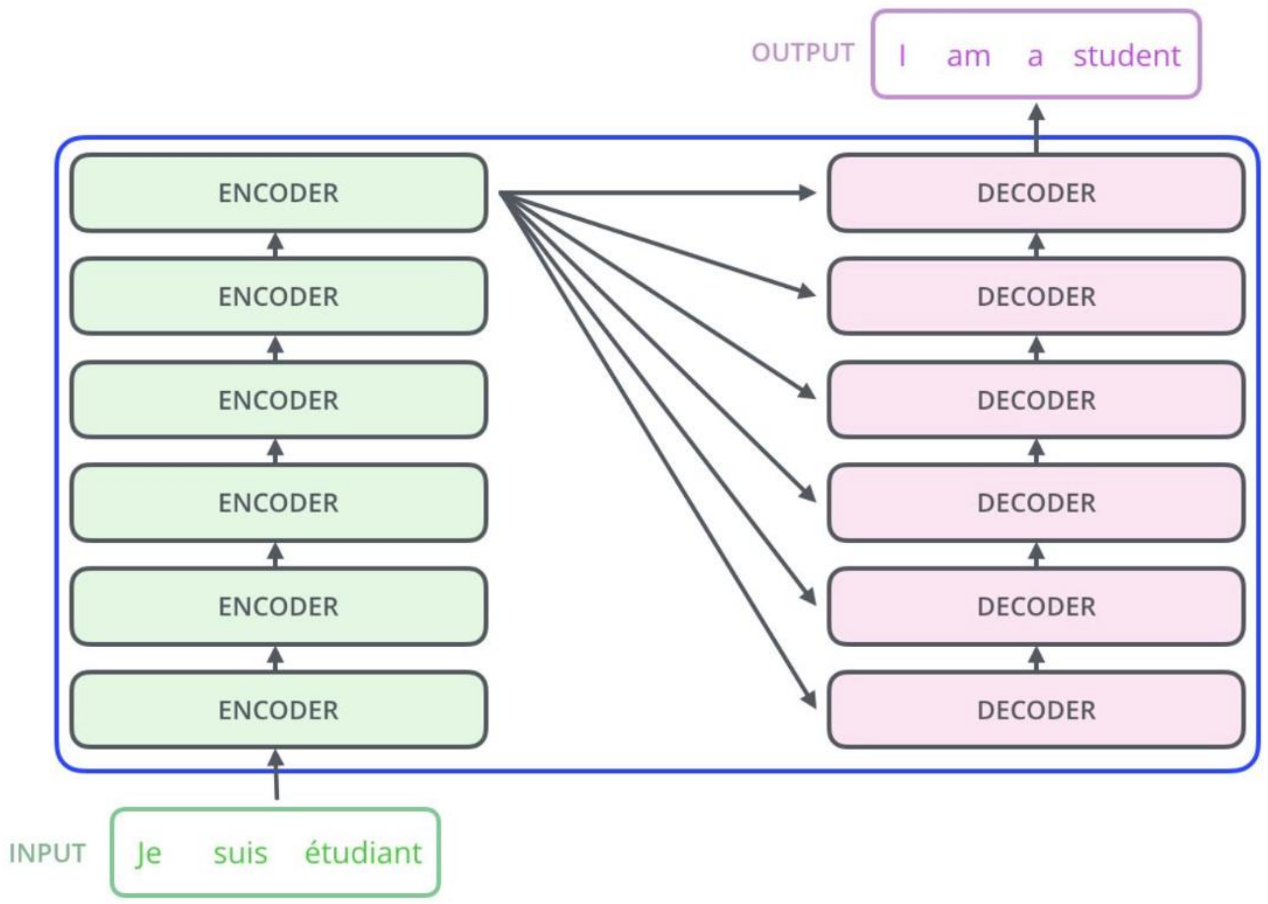
Encoder Block Map

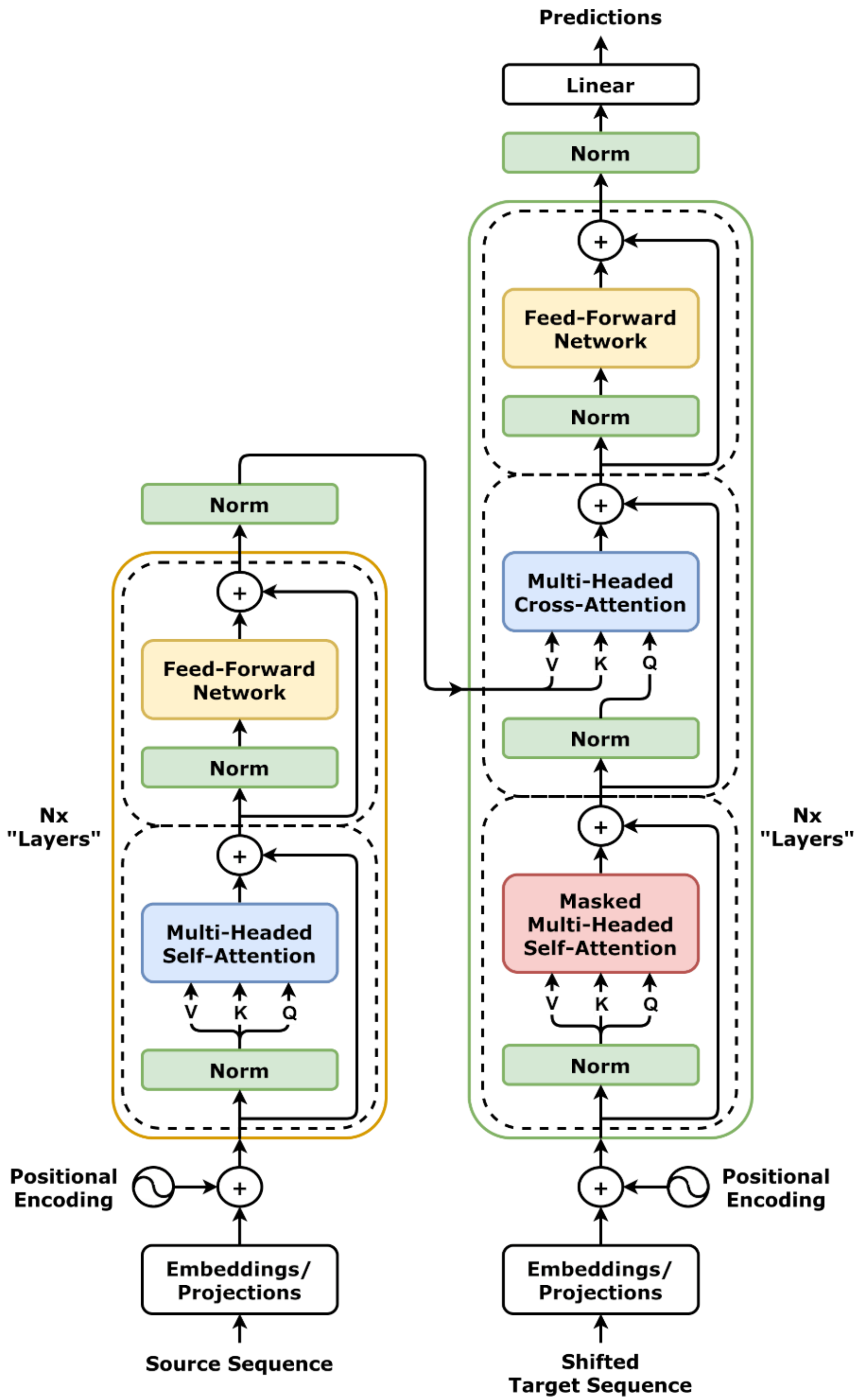
- Encoder block breaks down into two main parts: Self-Attention, and Feed Forward layers.
- Self-Attention layer is applied to each word individually.



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

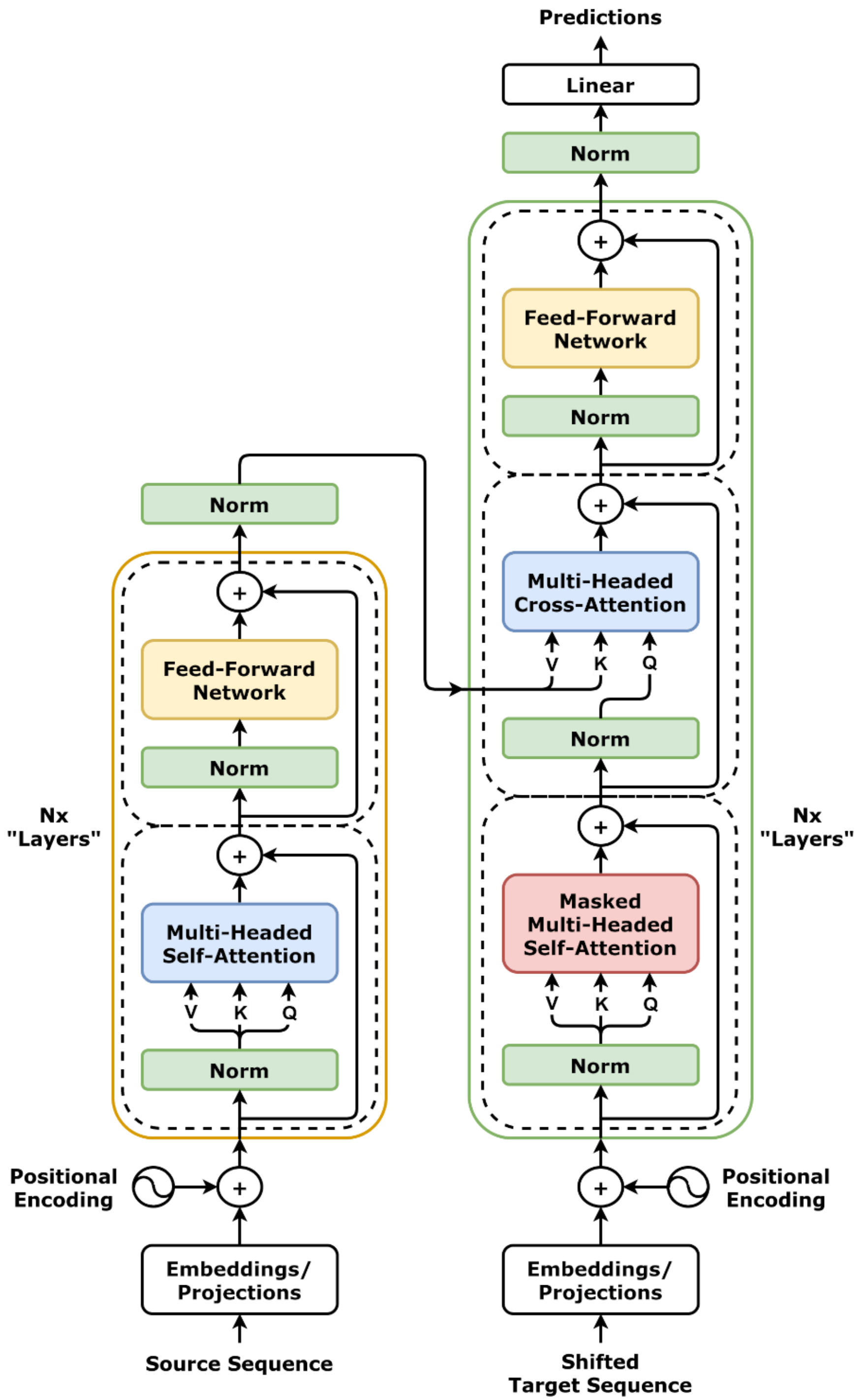
The Transformer



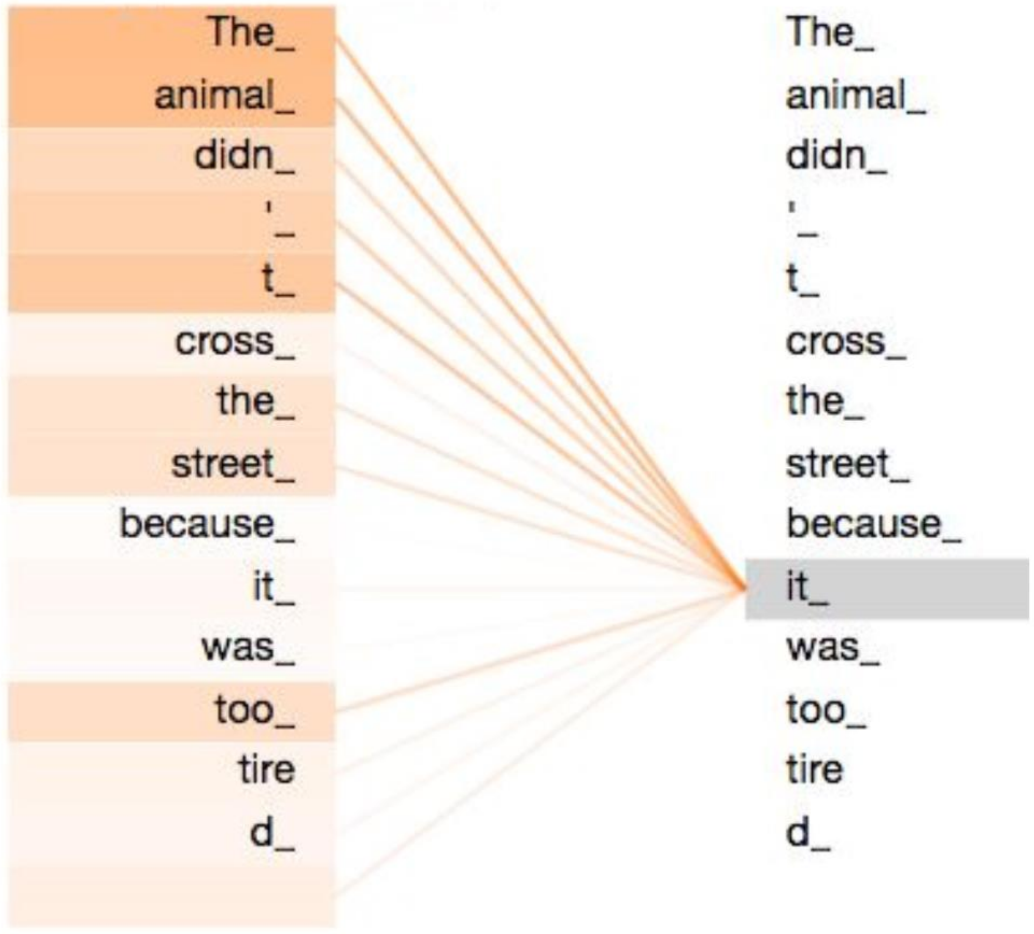


Components

- Self-Attention
- Cross-Attention
- Position Encoding
- Norm



Self-Attention: Input's attention on itself



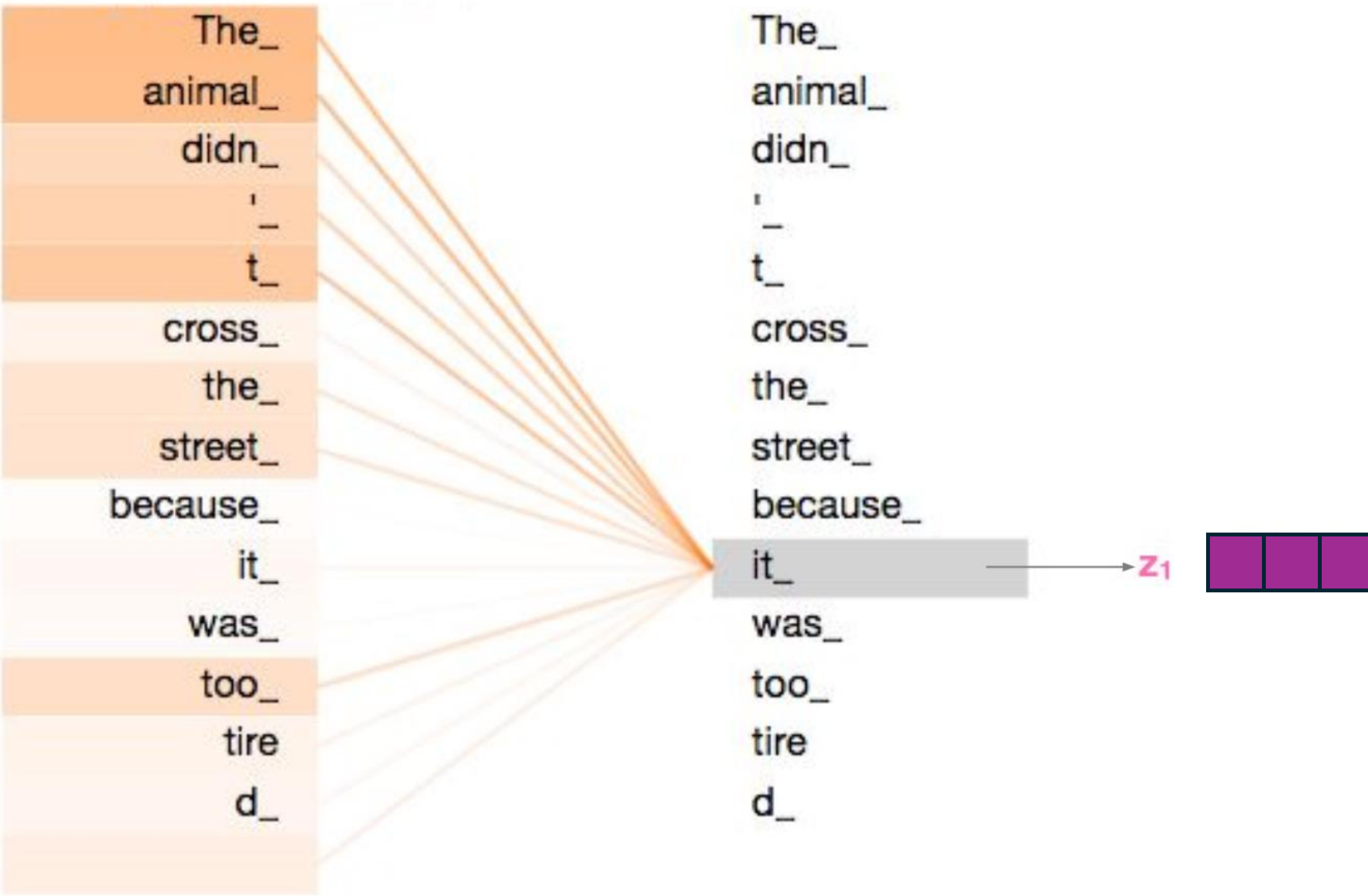
Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Overview

- **The big idea:**
  Self-attention computes the output vector z_i for each word via a weighted sum of vectors extracted from each word in the input sentence

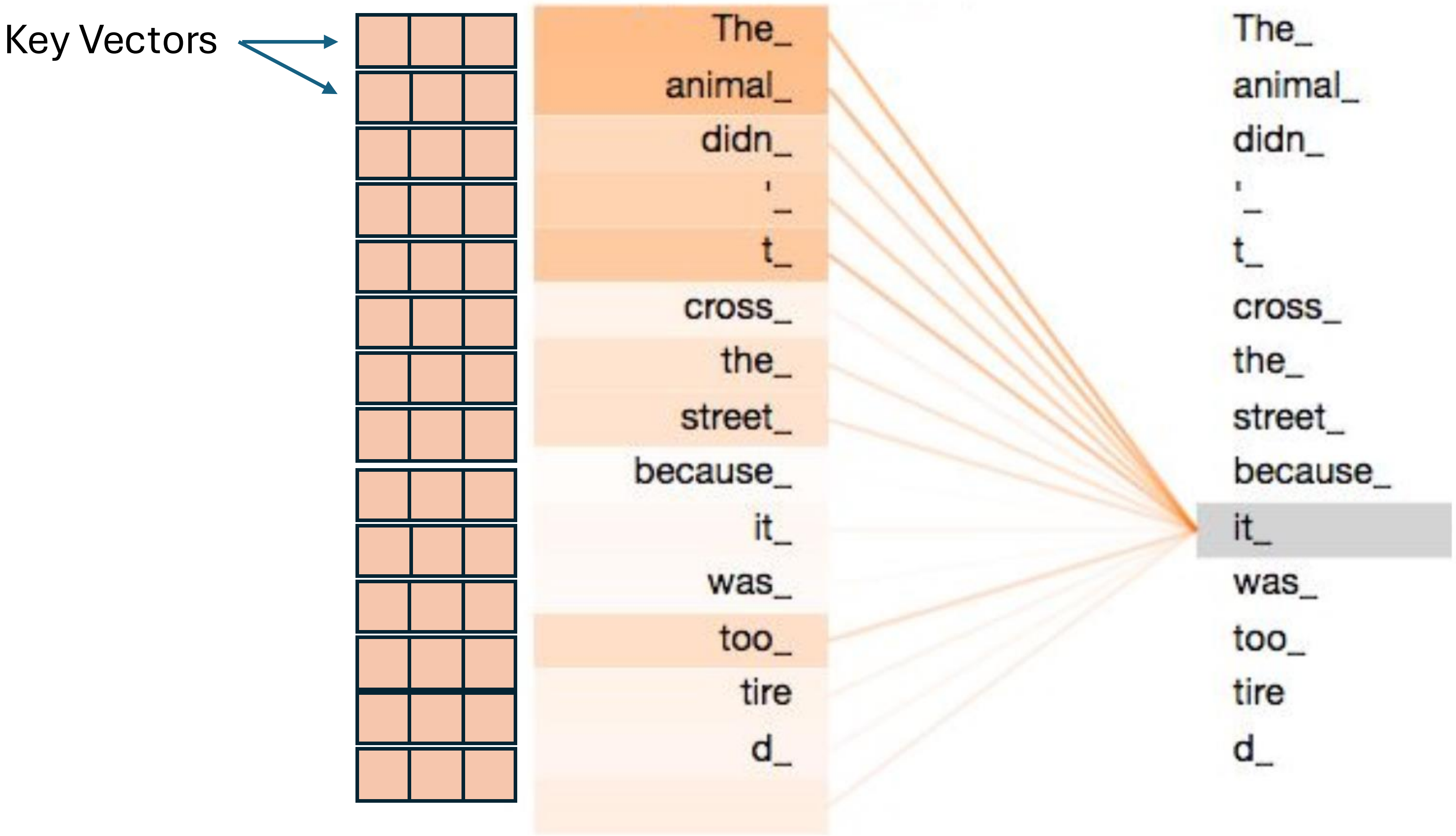
- Here, self-attention learns that "it" should pay attention to "the animal" (i.e. the entity that "it" refers to)

- Why the name ***self***-attention?
  This describes attention that the input sentence pays to itself



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

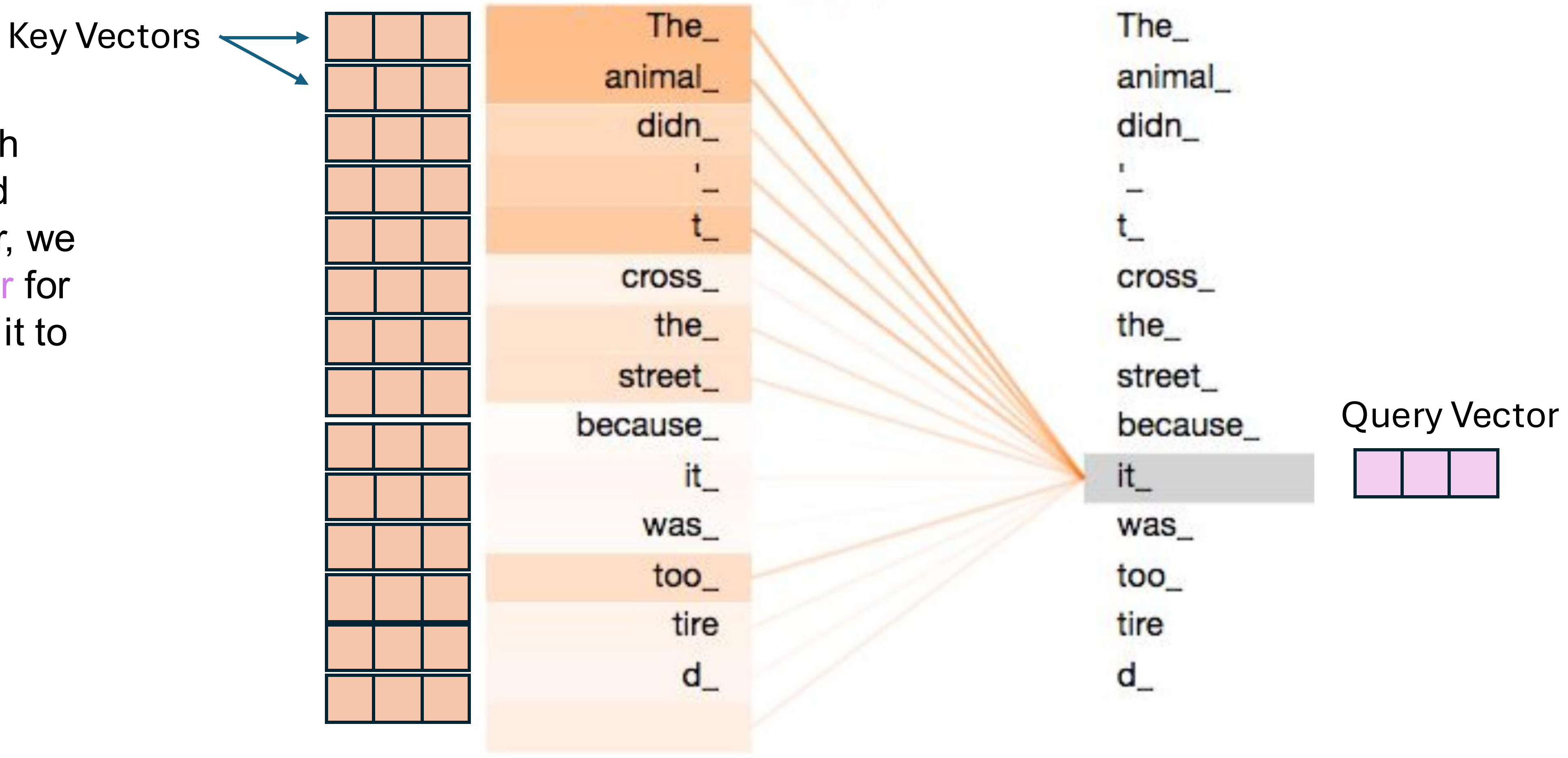
Self-Attention: Input's attention on itself



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input’s attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

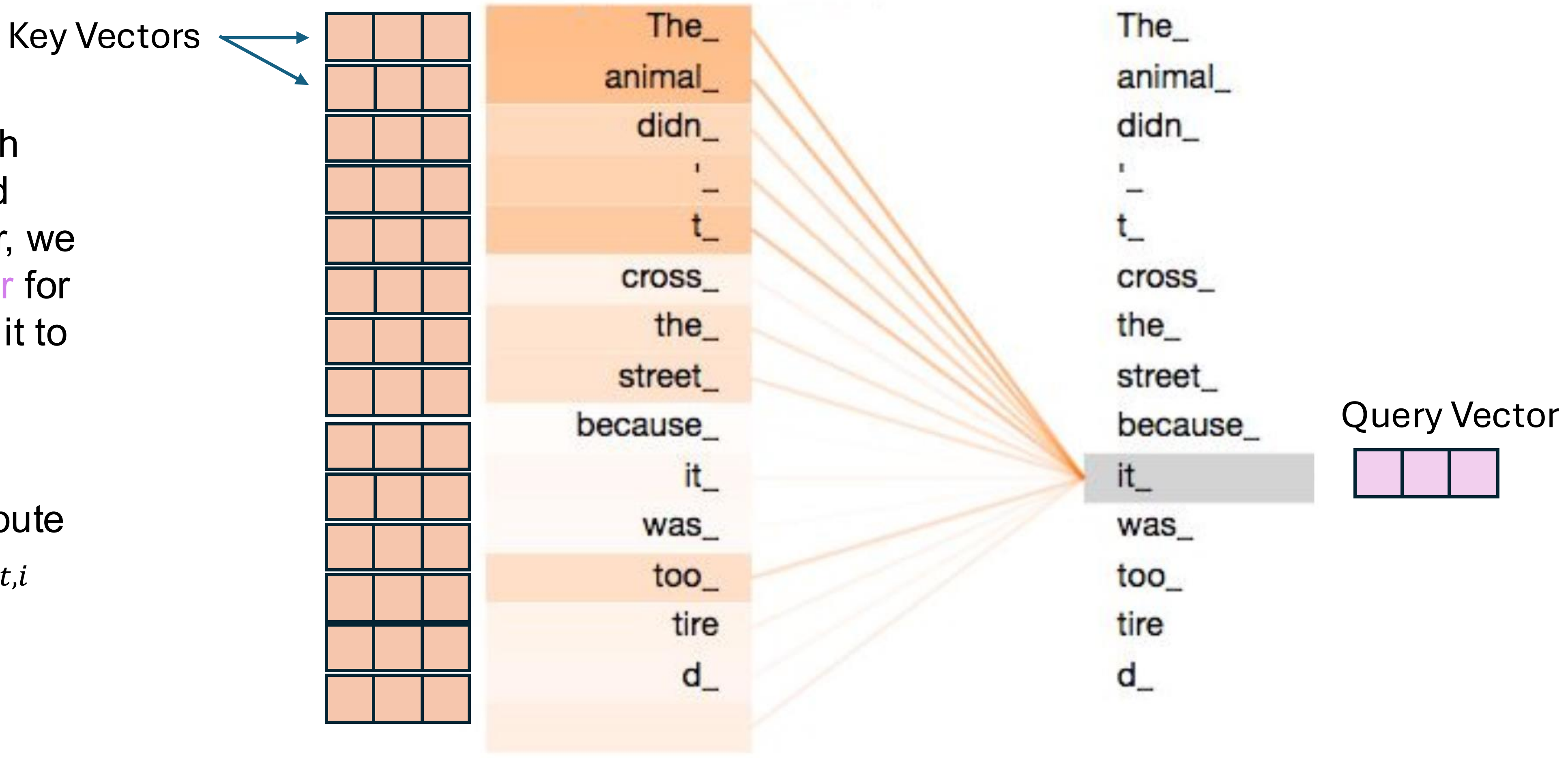


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$

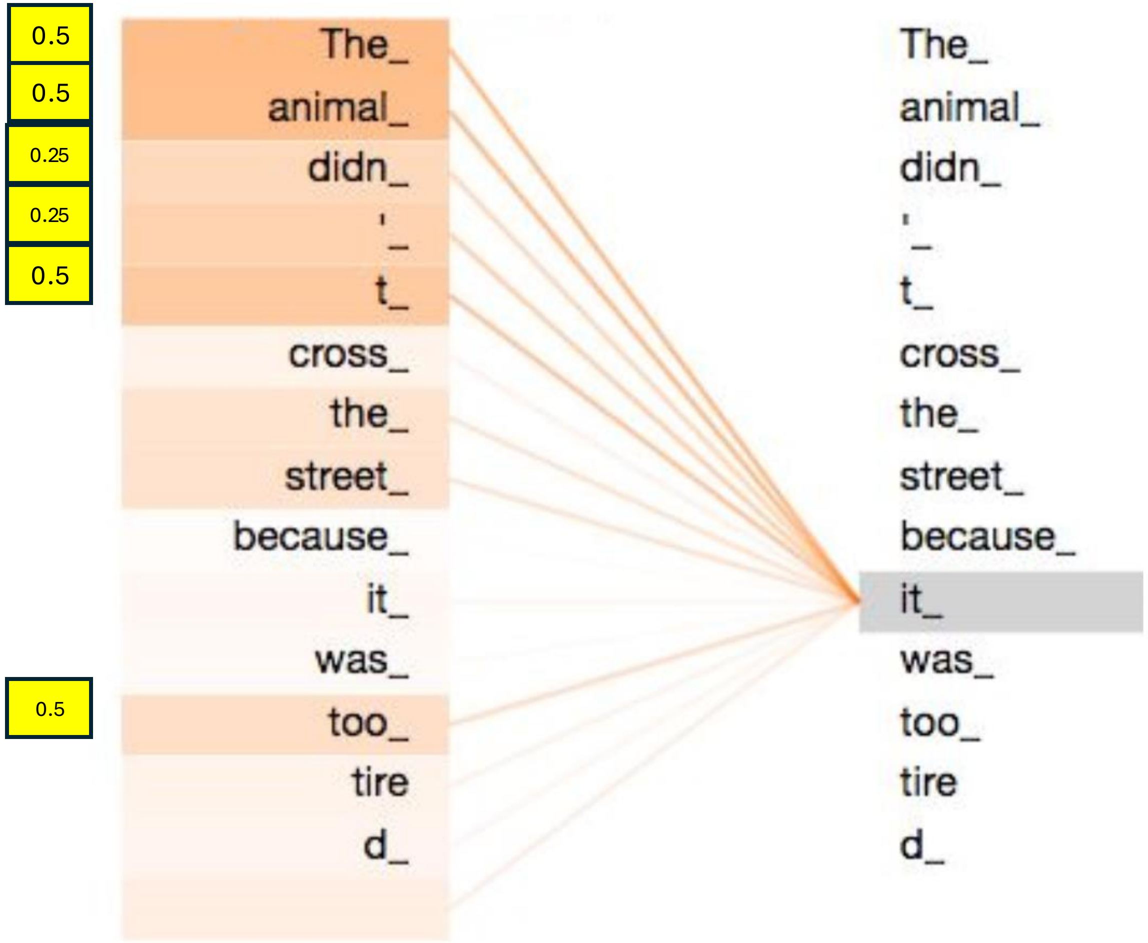


Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$



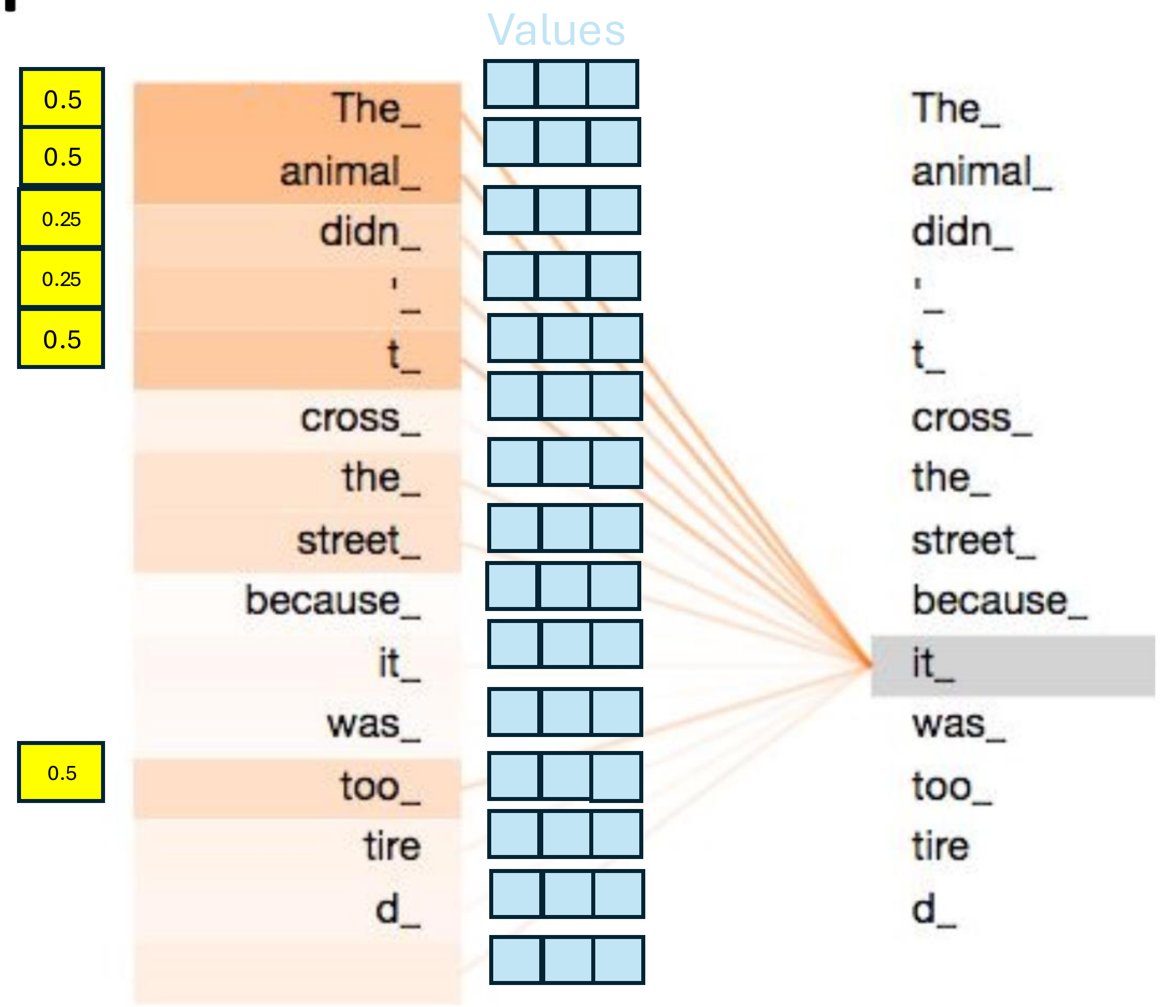
Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input’s attention on itself

What do we do next?

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$



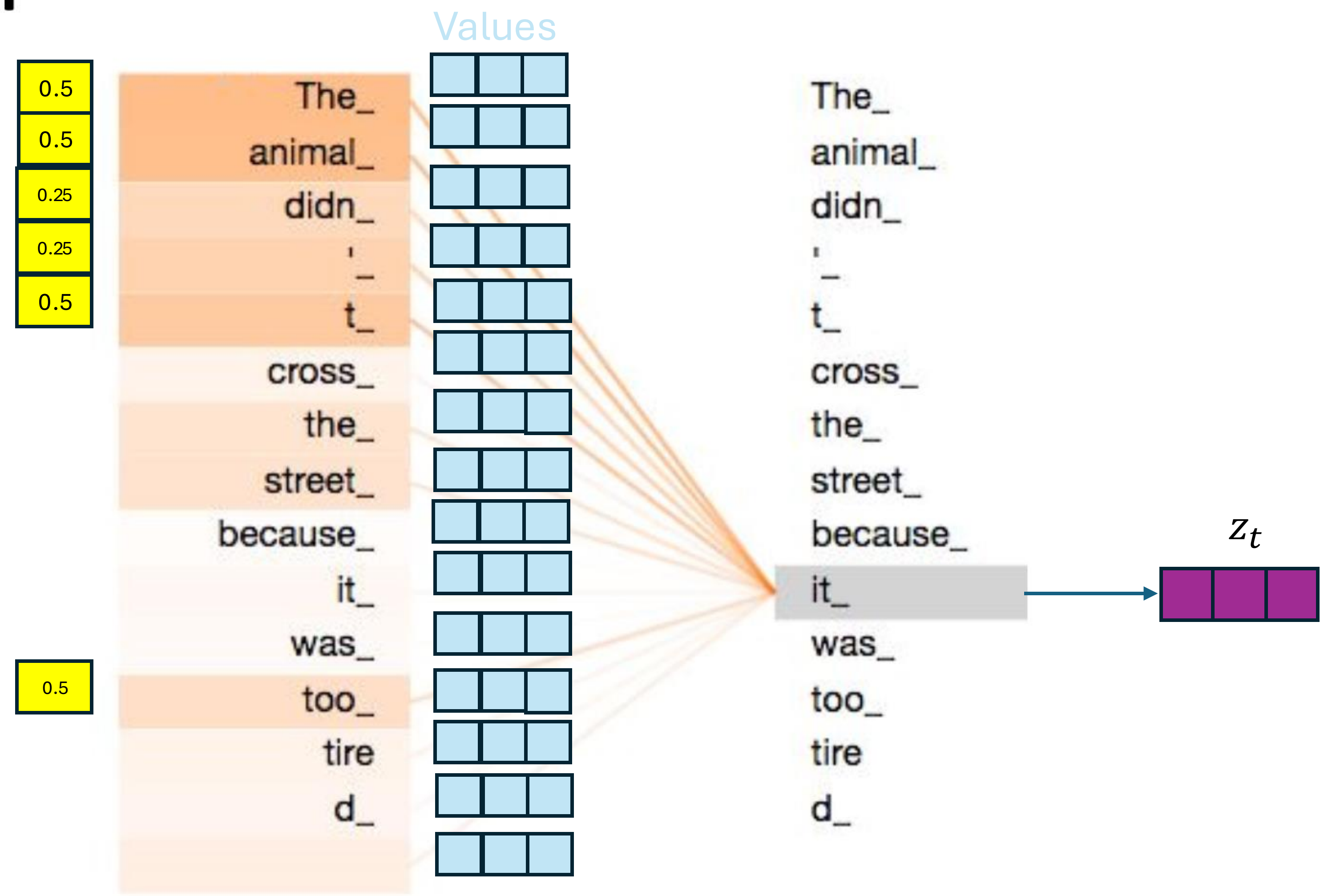
Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Input's attention on itself

To determine how much attention a word should pay to each other other, we compute a query vector for the word and compare it to a key vector for every other word...

Which use use to compute the alignment scores $a_{t,i}$

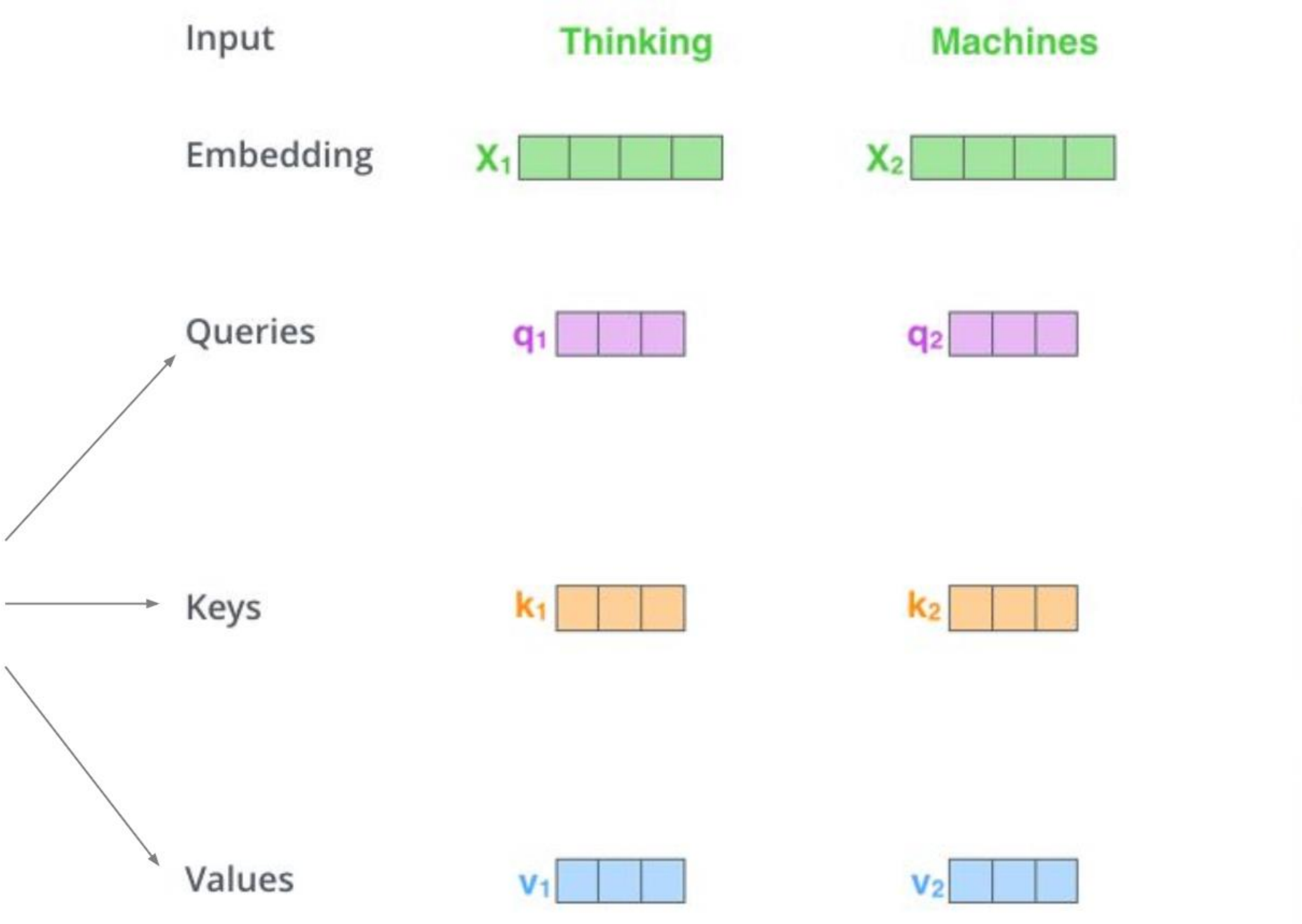
To produce the output vector, we sum up the value vectors for each word, weighted by the score we computed in step 1



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Details

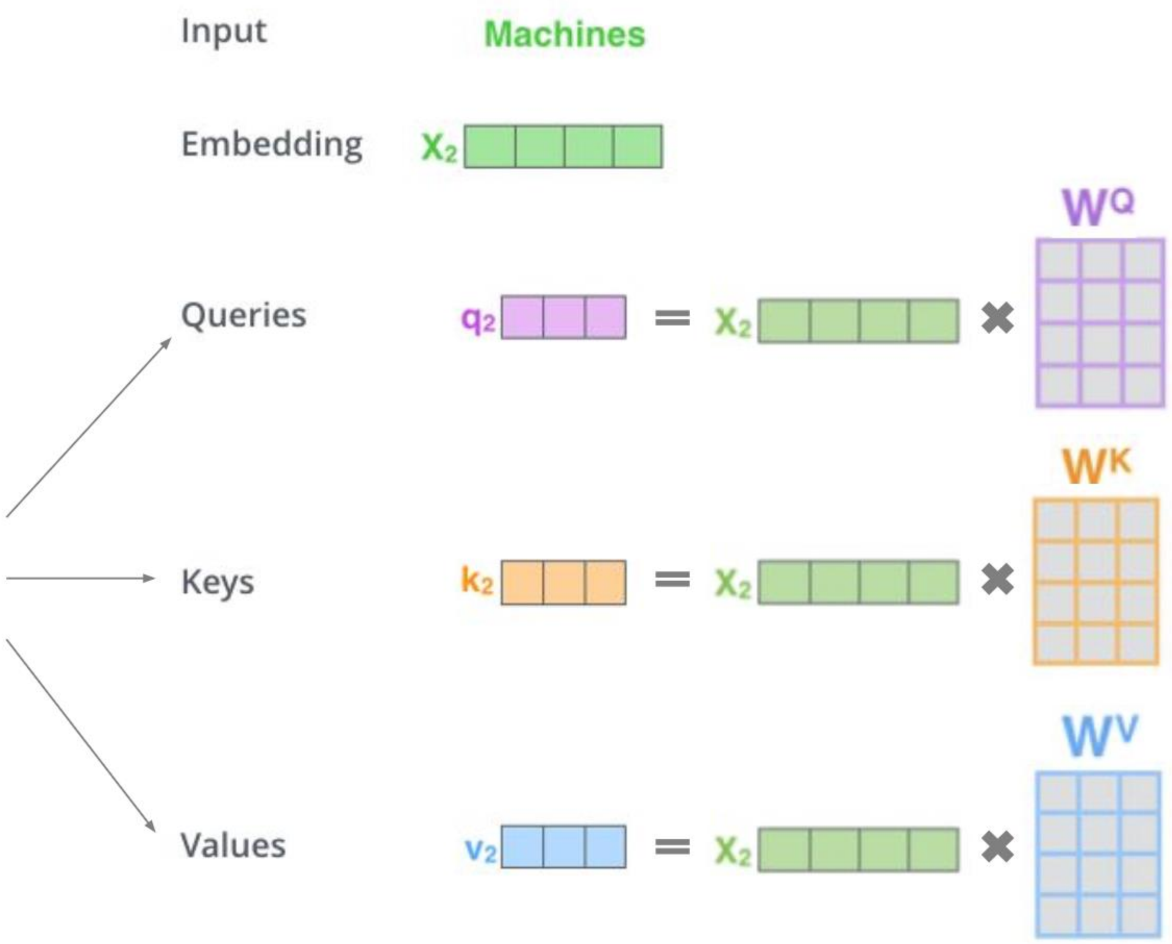
Extract 3 vectors from each word embedding



Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Self-Attention: Details

Extract 3 vectors from each word embedding



Each vector is obtained by multiplying the embedding with the respective weight matrix.

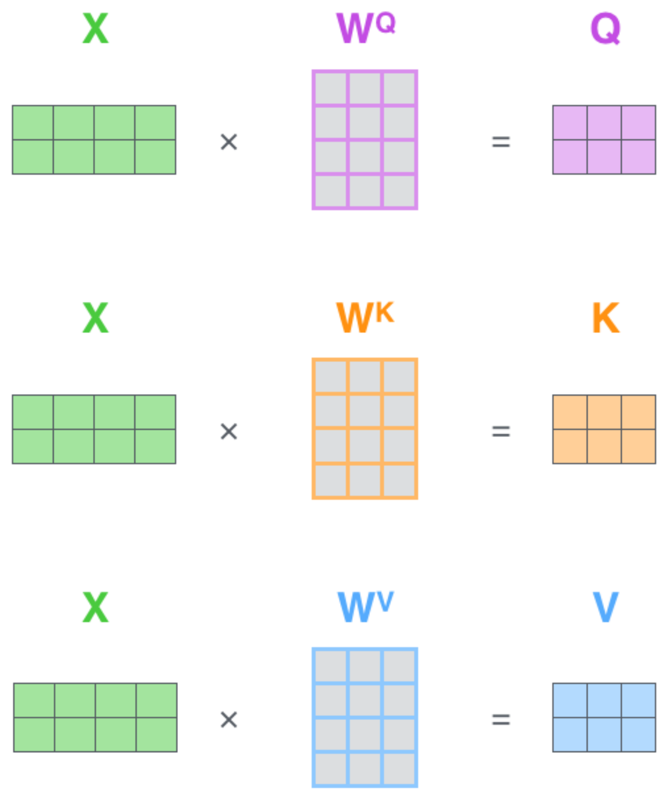
How do we get these weight matrices?

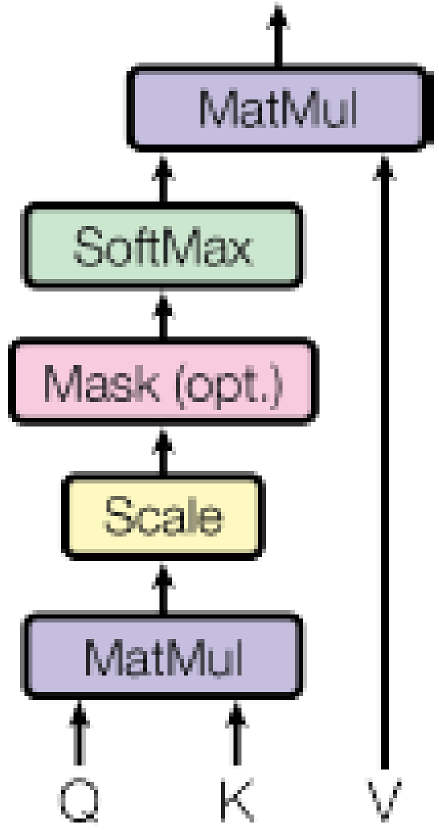
These matrices are the ***trainable parameters*** of the network

Alammar, Jay (2018). The Illustrated Transformer [Blog post]. Retrieved from https://jalammar.github.io/illustrated-transformer/

Scaled Dot Product Attention

Generate Q, K, V, by multiplying word embedding X by weight matrix (i.e., pass through a fully connected layer)





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

Multi-Headed Attention

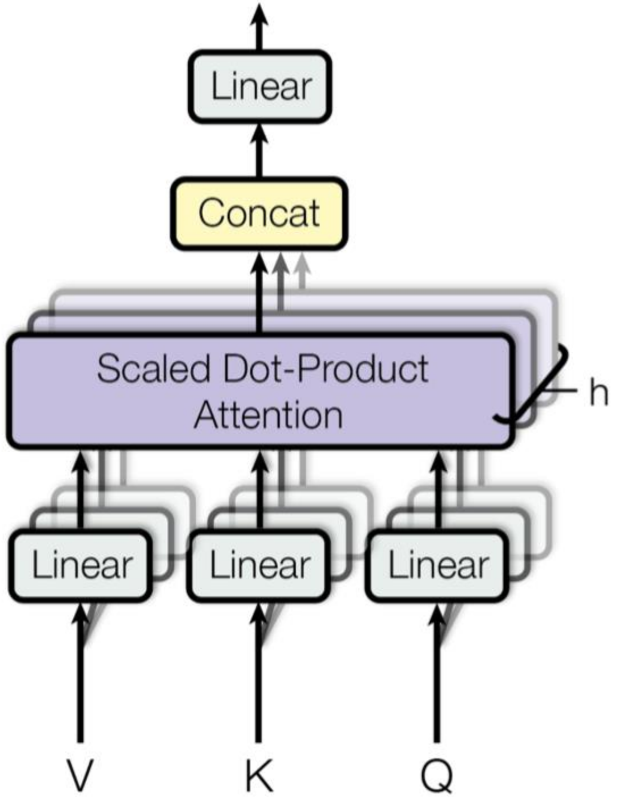
Similar to convolutional layers with multiple filters, we can have “multi-headed attention”

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, \dots, \mathrm{head_h})W^0$$

Where: $head_i = \mathrm{Attention}(\mathrm{QW_i^q}, \mathrm{KW_i^k}, \mathrm{VW_i^v})$

Projected Attention: Project (Q, K, V) with learned parameters W^q, W^k, W^v

Separate learned fully-connected layer for each head *i* and for each of (Q, K, V)



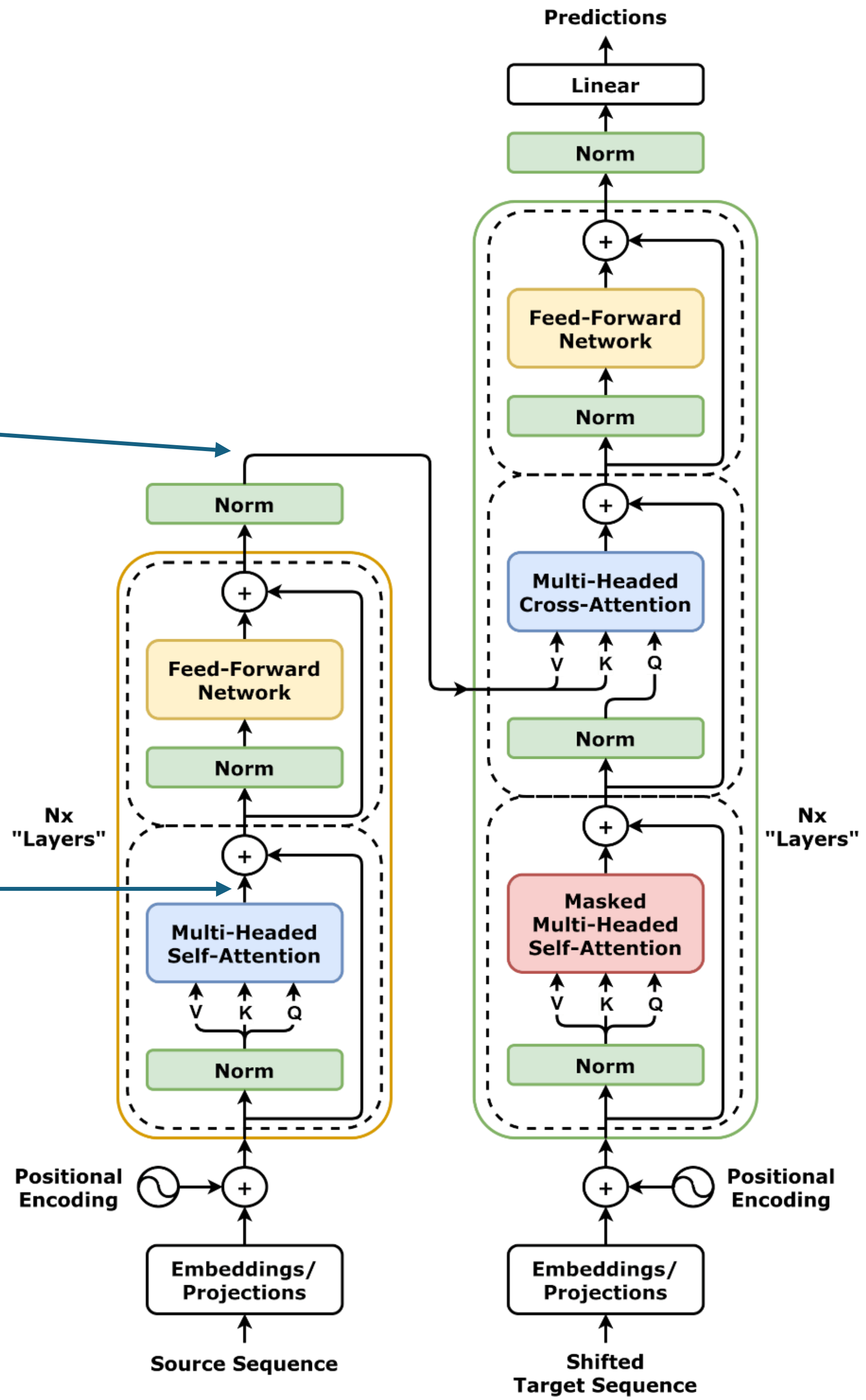
Outputs of Encoder

What does the encoder output?

2. What is this?

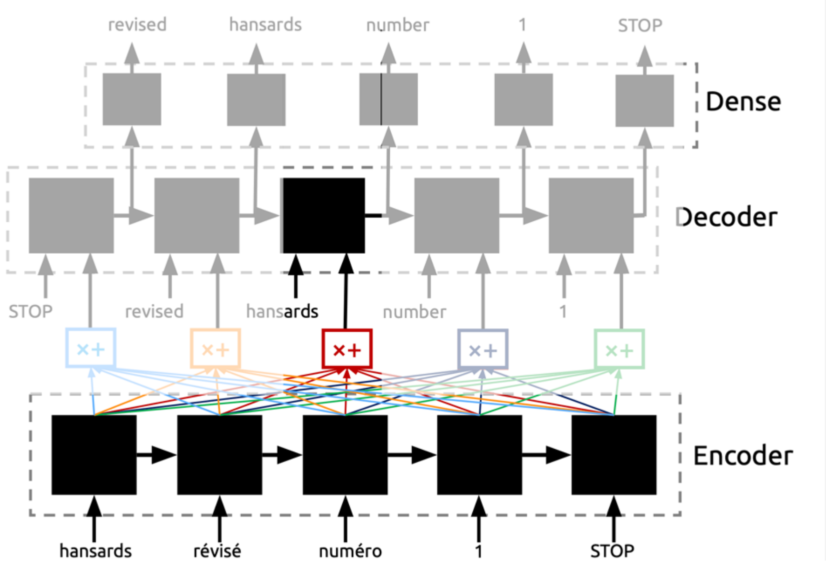
1. What is this?

An embedding (vector) for each word in source sequence!

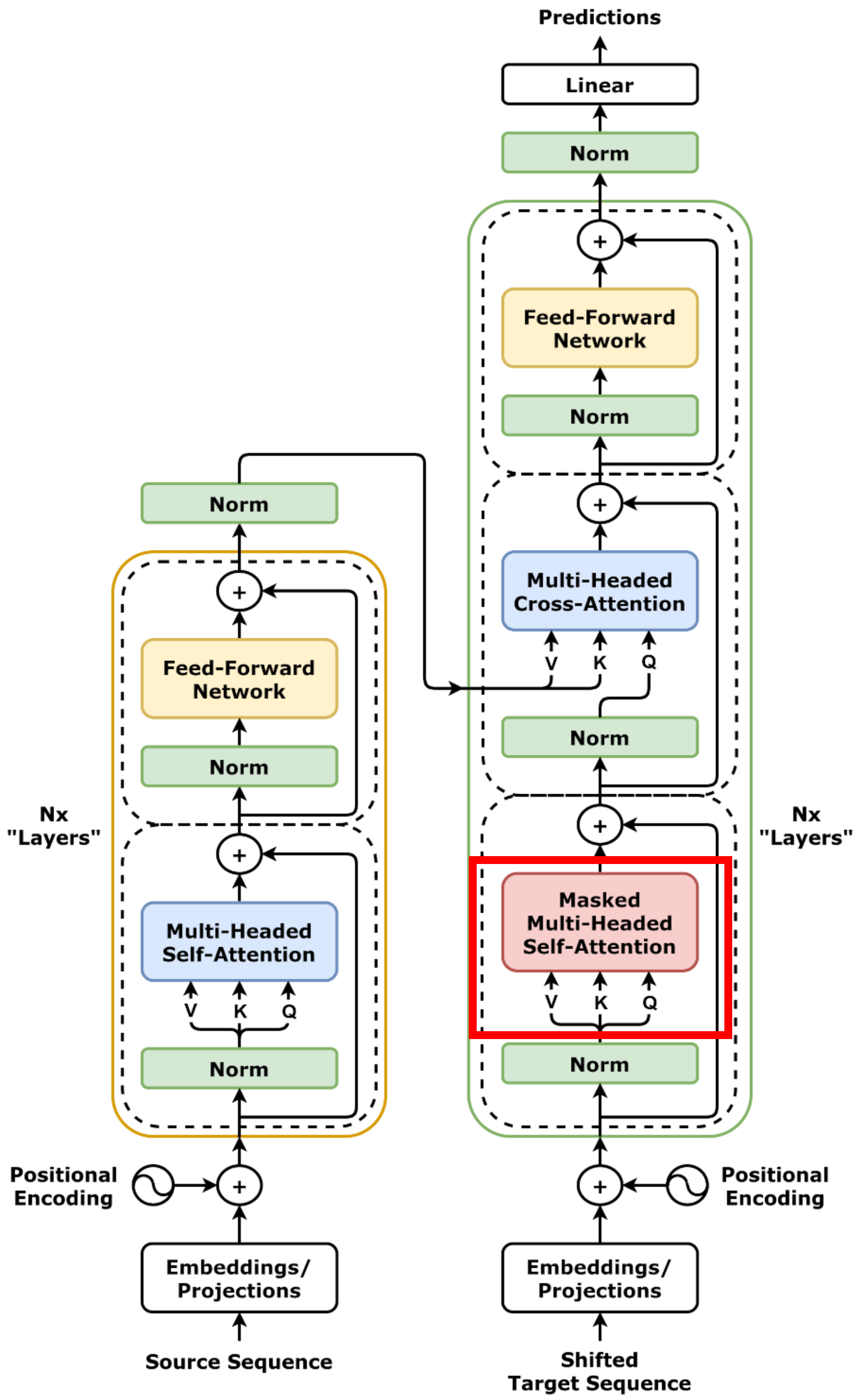


Cross Attention

- Self-Attention is how much each input "attends" to every other input
- Cross-Attention is how much every output "attends" to every input (i.e., our original motivation for attention)

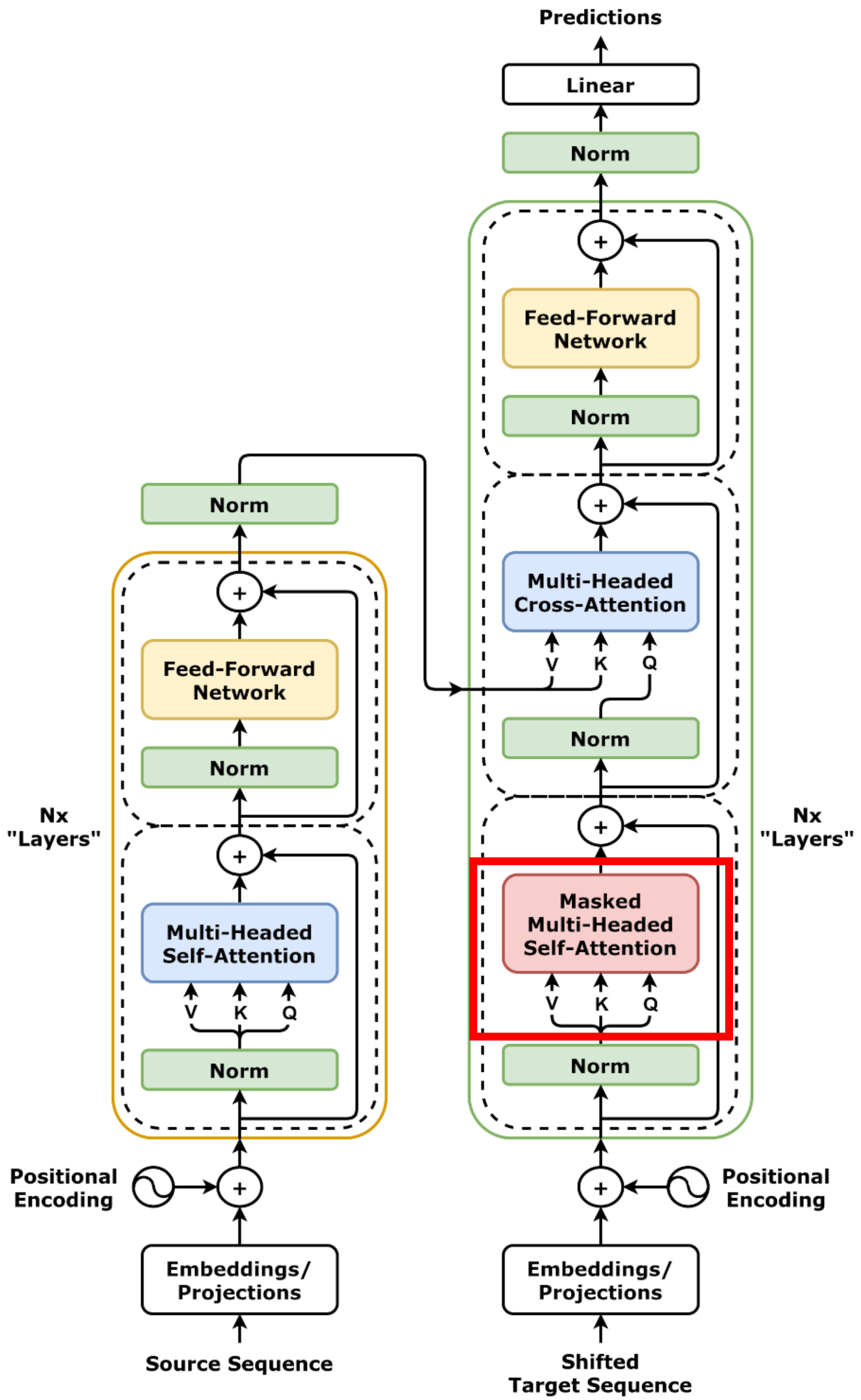


Generating Tokens



Generating Tokens

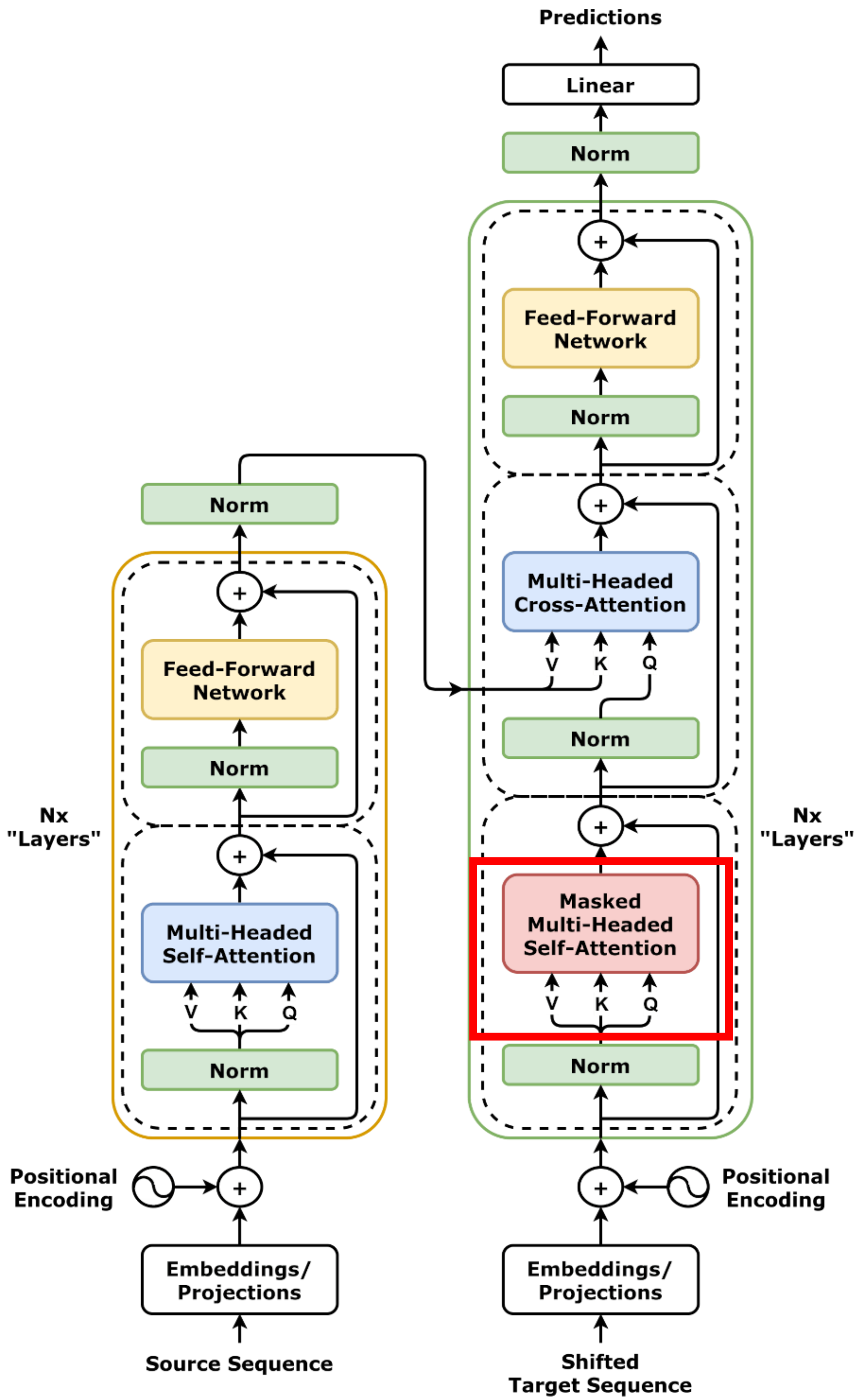
How do we generate new output sentences (English to French)?



Generating Tokens

How do we generate new output sentences (French to English)?

1. Take in source sentence (in French) and one initial token <Start> in target sequence

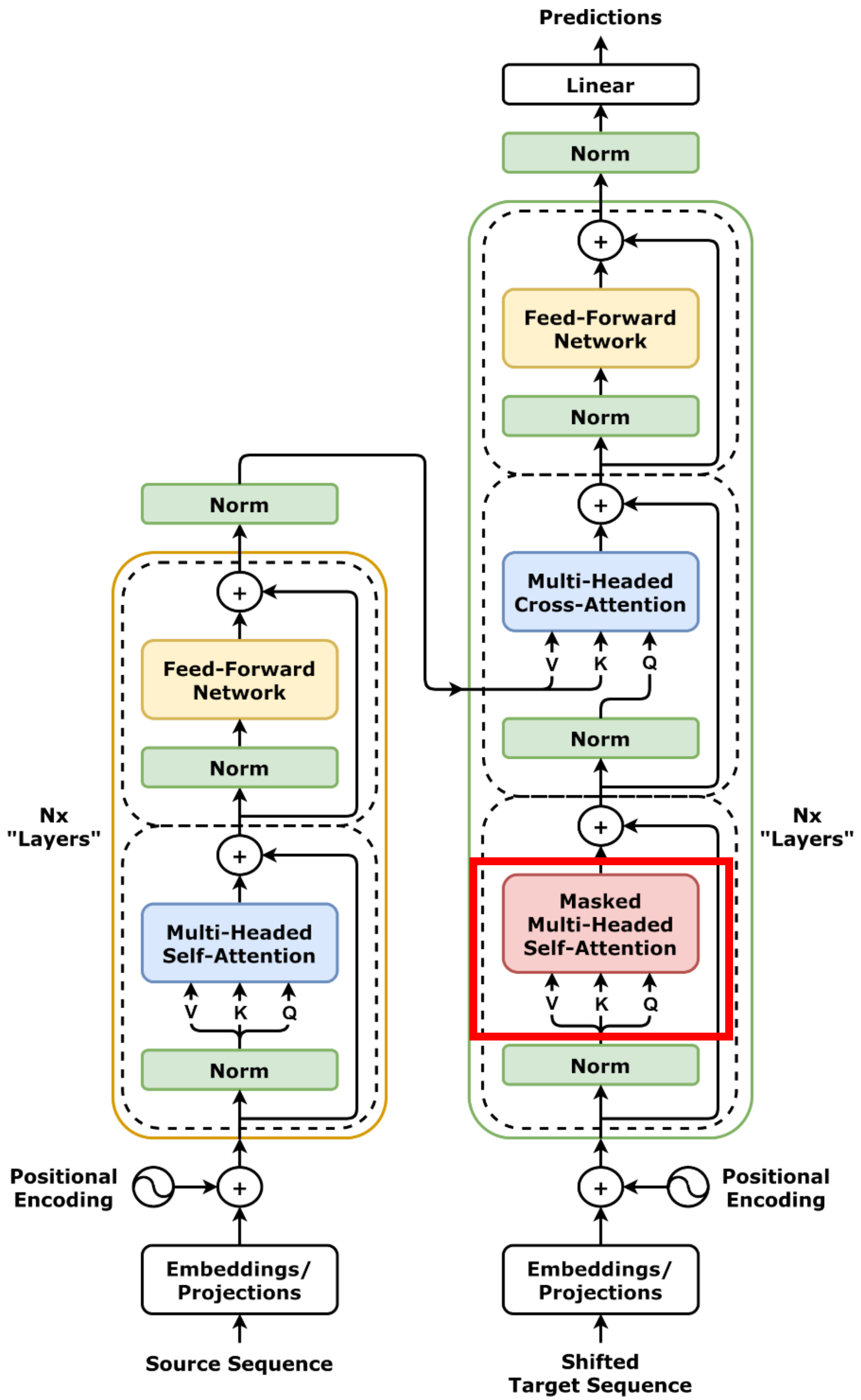


Generating Tokens

How do we generate new output sentences (French to English)?

1. Take in source sentence (in French) and one initial token <Start> in target sequence

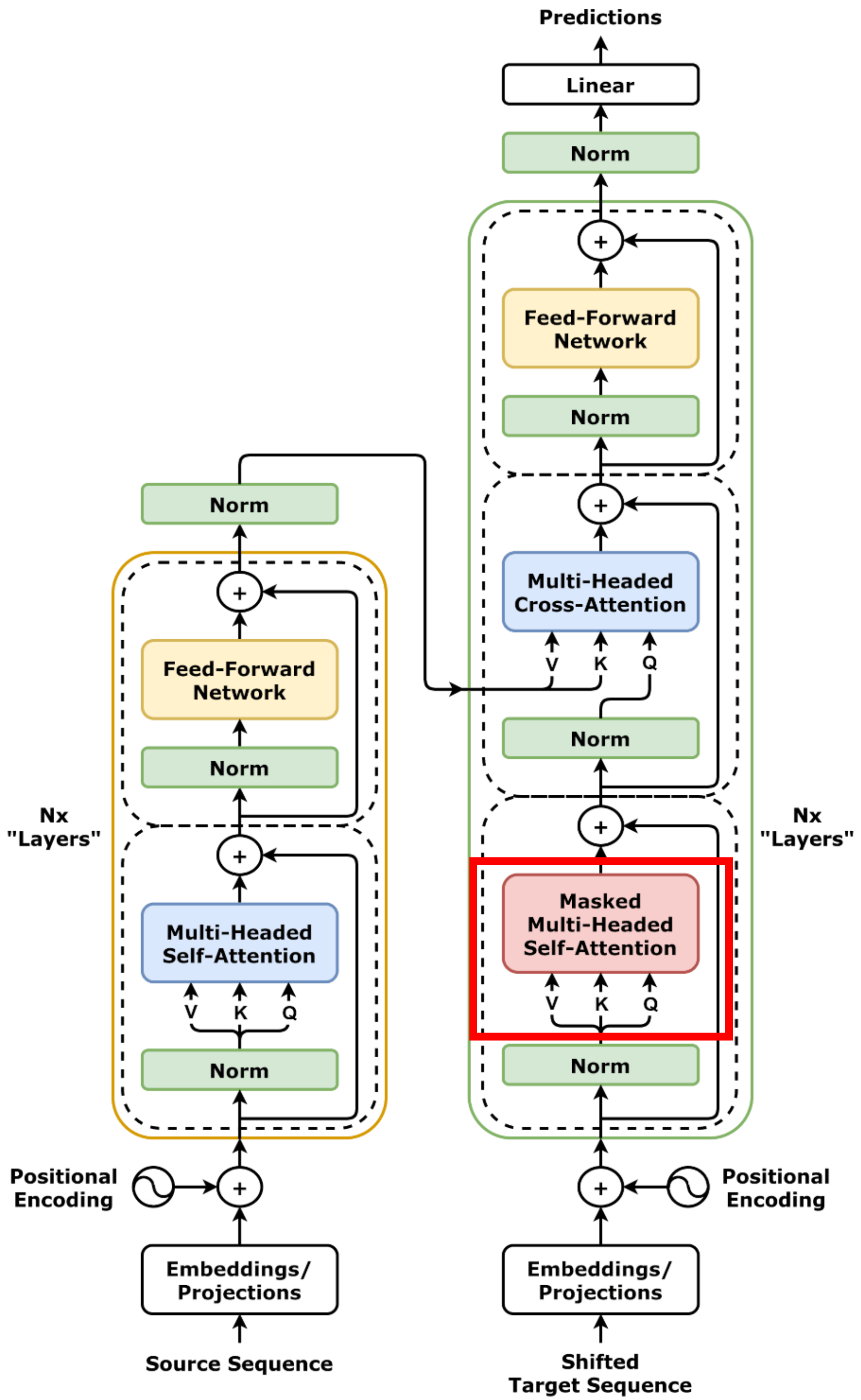
2. Generate one output token (first English word)



Generating Tokens

How do we generate new output sentences (French to English)?

1. Take in source sentence (in French) and one initial token <Start> in target sequence
2. Generate one output token (first English word)
3. Rerun model with target sequence "<Start> + Robot"



(Masked) Self Attention

<Start>

<Start>

Robot

Robot

(Masked) Self Attention

Key Vectors

<Start>

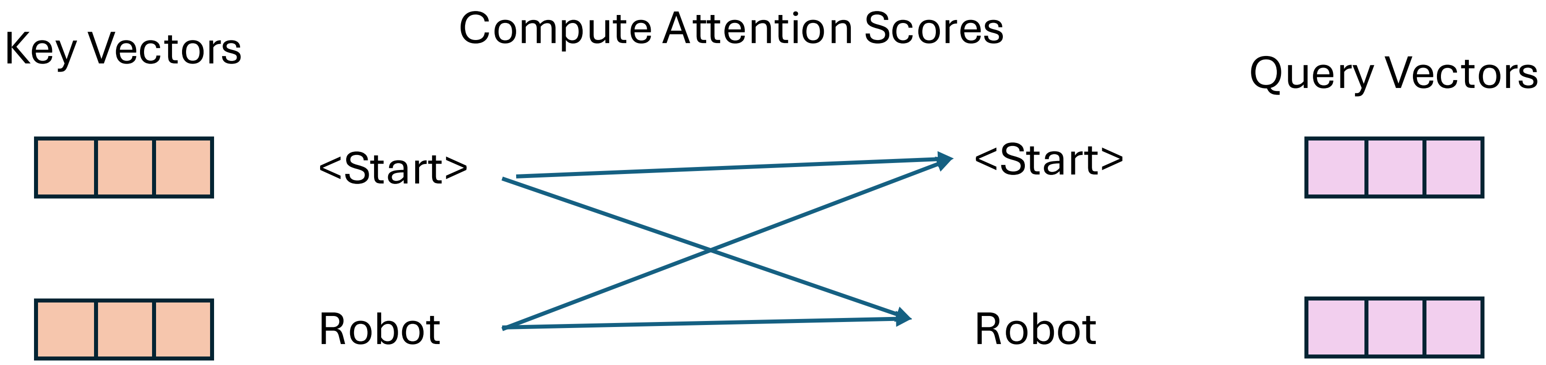
Robot

<Start>

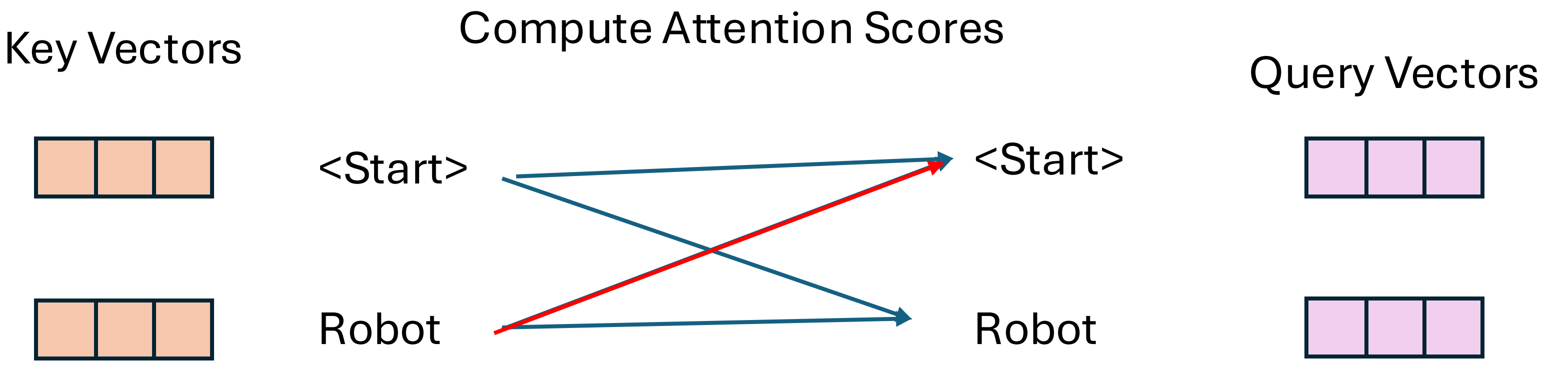
Robot

(Masked) Self Attention

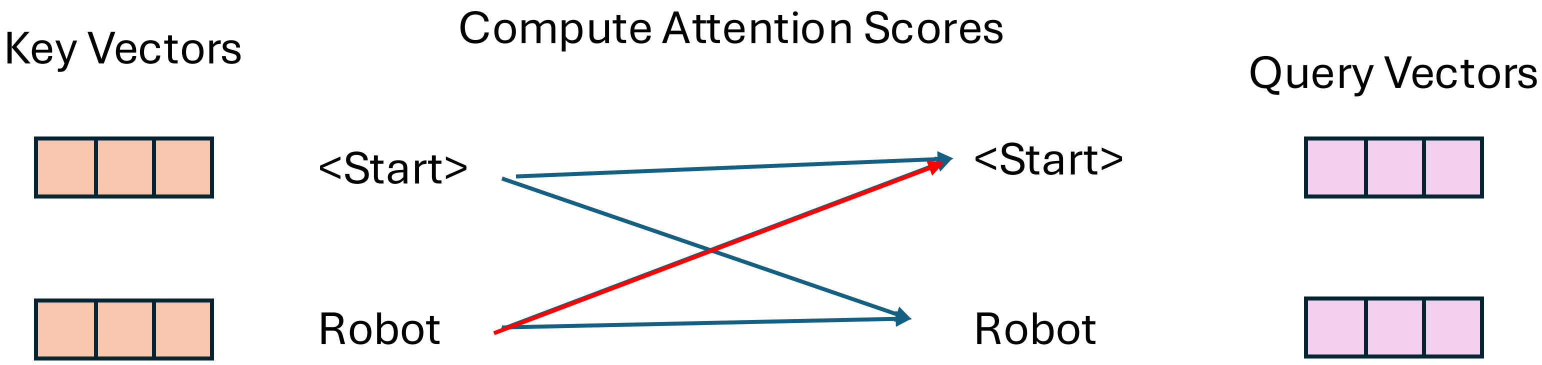
(Masked) Self Attention



(Masked) Self Attention

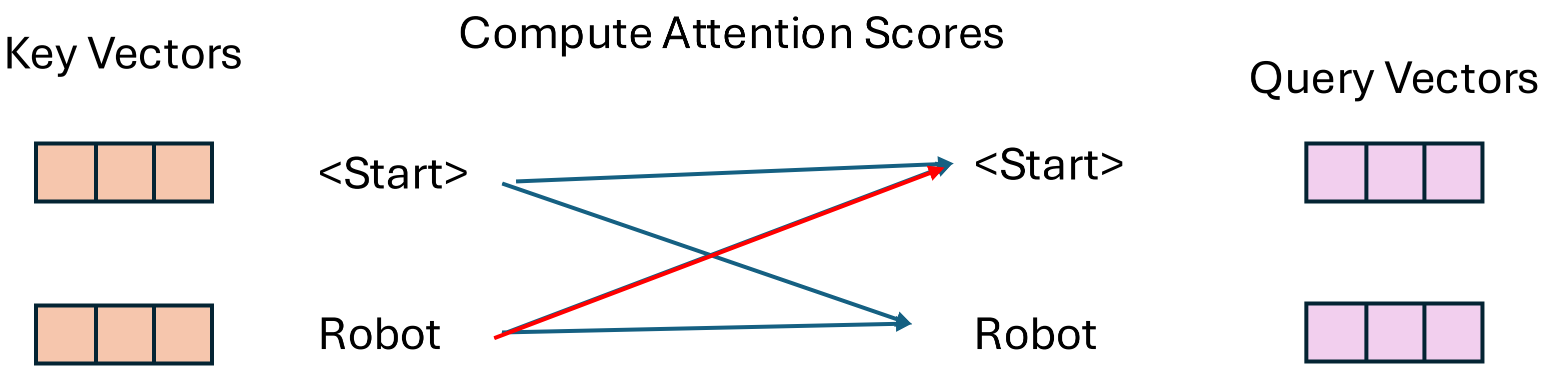


(Masked) Self Attention



Wait… <Start> is paying attention to Robot?
<Start> was generated before Robot, why would it pay attention to Robot?

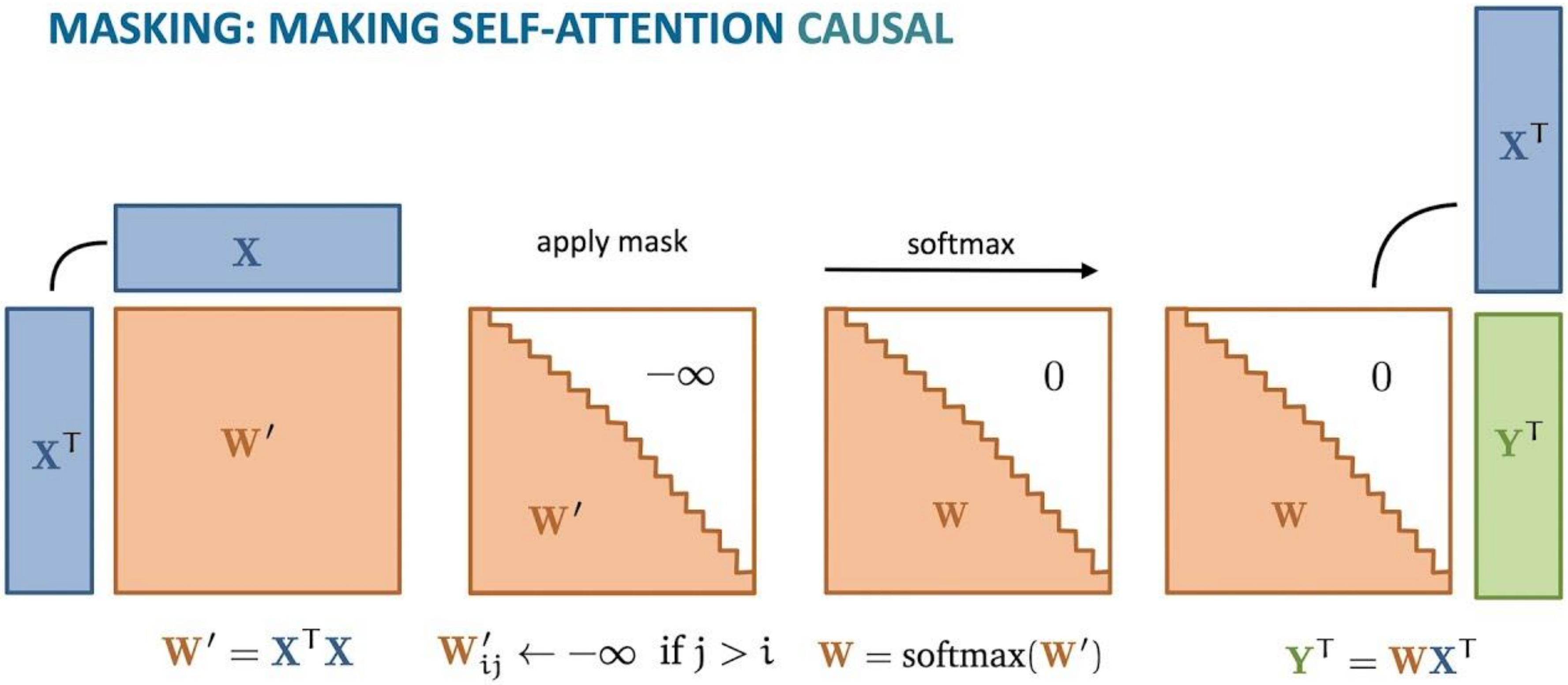
(Masked) Self Attention



Masked Self-Attention: Words in the (**Target**) sequence should not pay attention to words that come after them

Maintain “causality”: later words should not have an effect on the outputs of earlier words

Masked Self-Attention

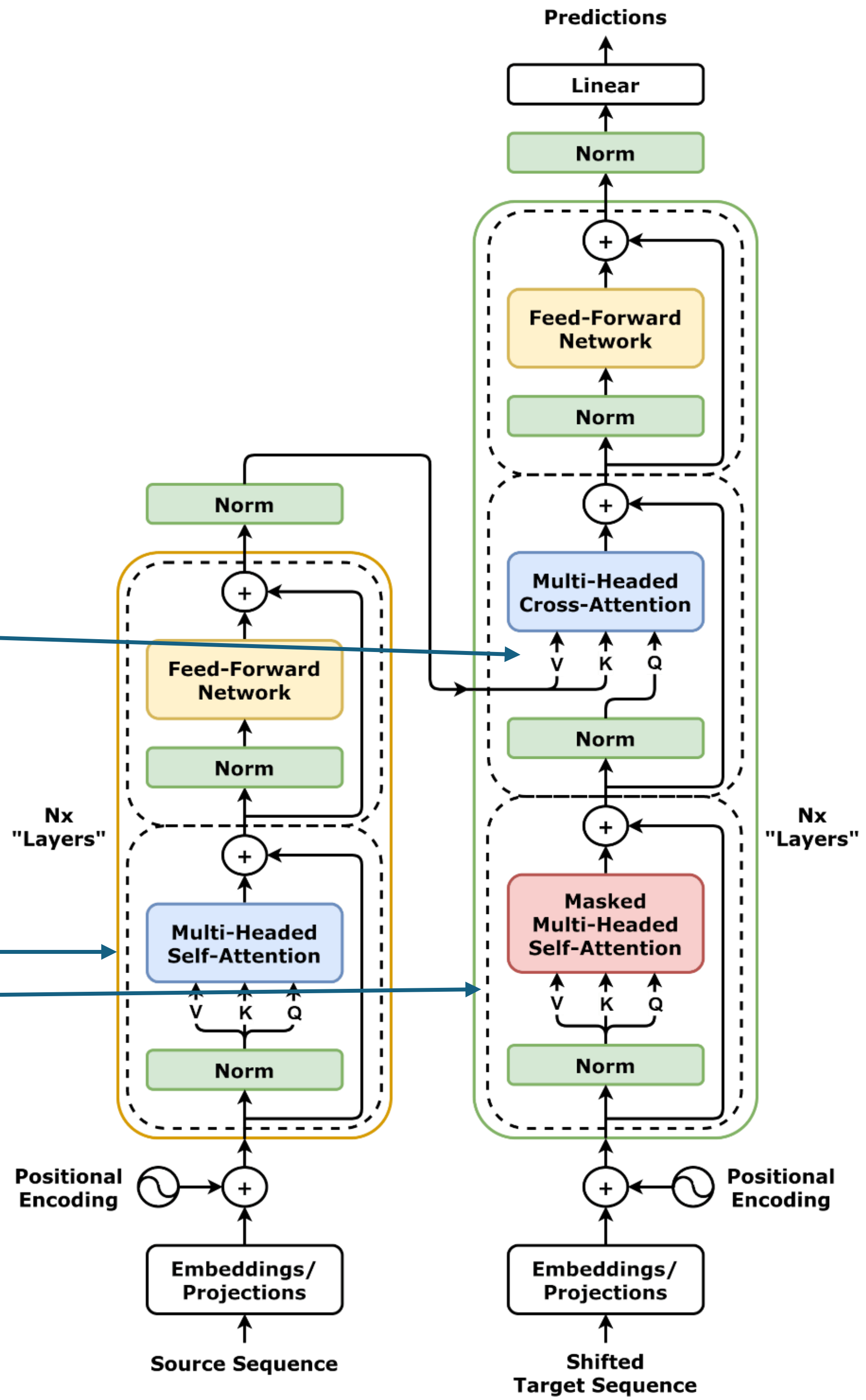


Source: https://www.youtube.com/watch?v=oUhGZMCTHtI&themeRefresh=1

Transformer

Cross-Attention

Self Attention



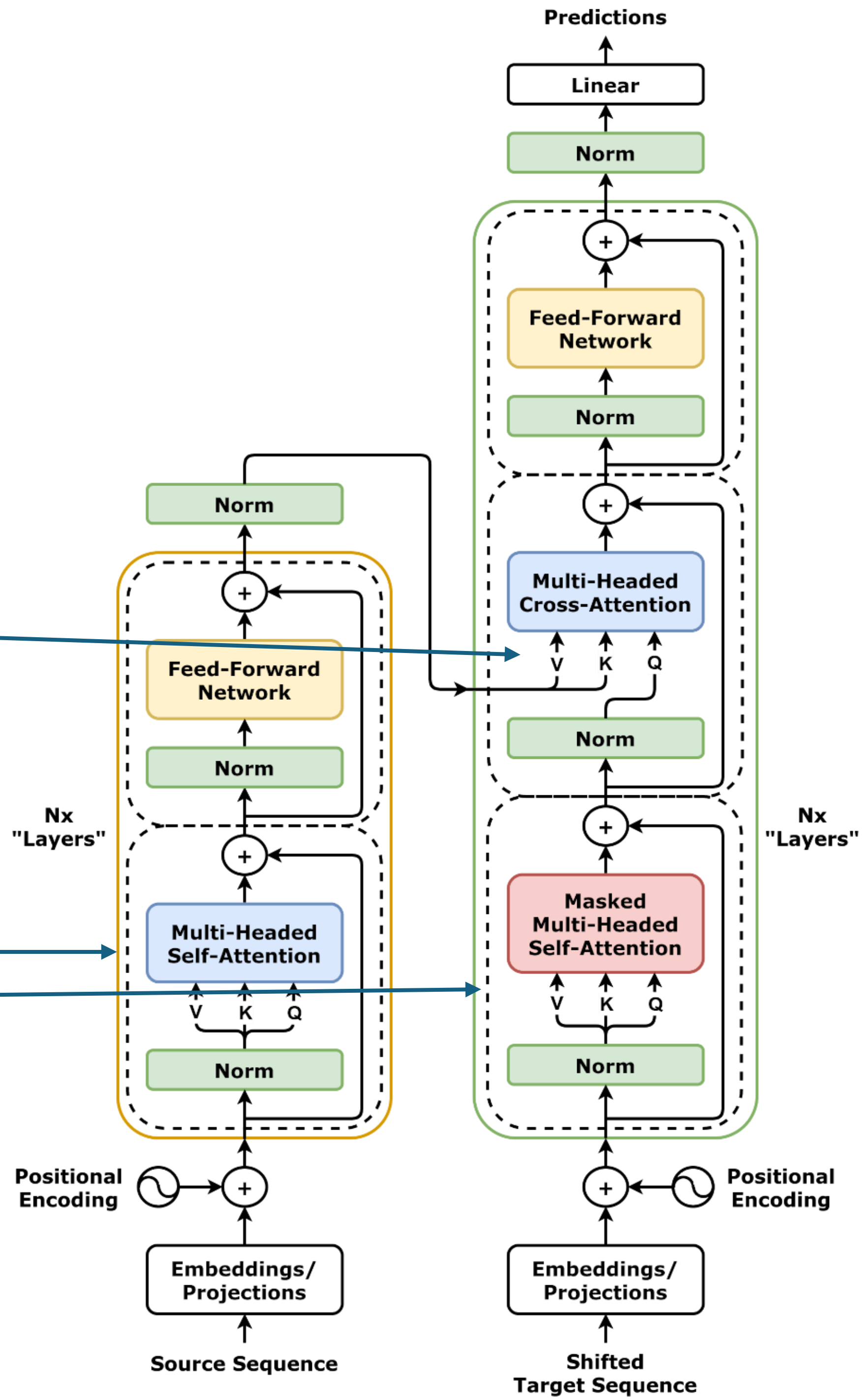
Transformer

What's left?

1. Position Encoding
2. Norm

Cross-Attention

Self Attention

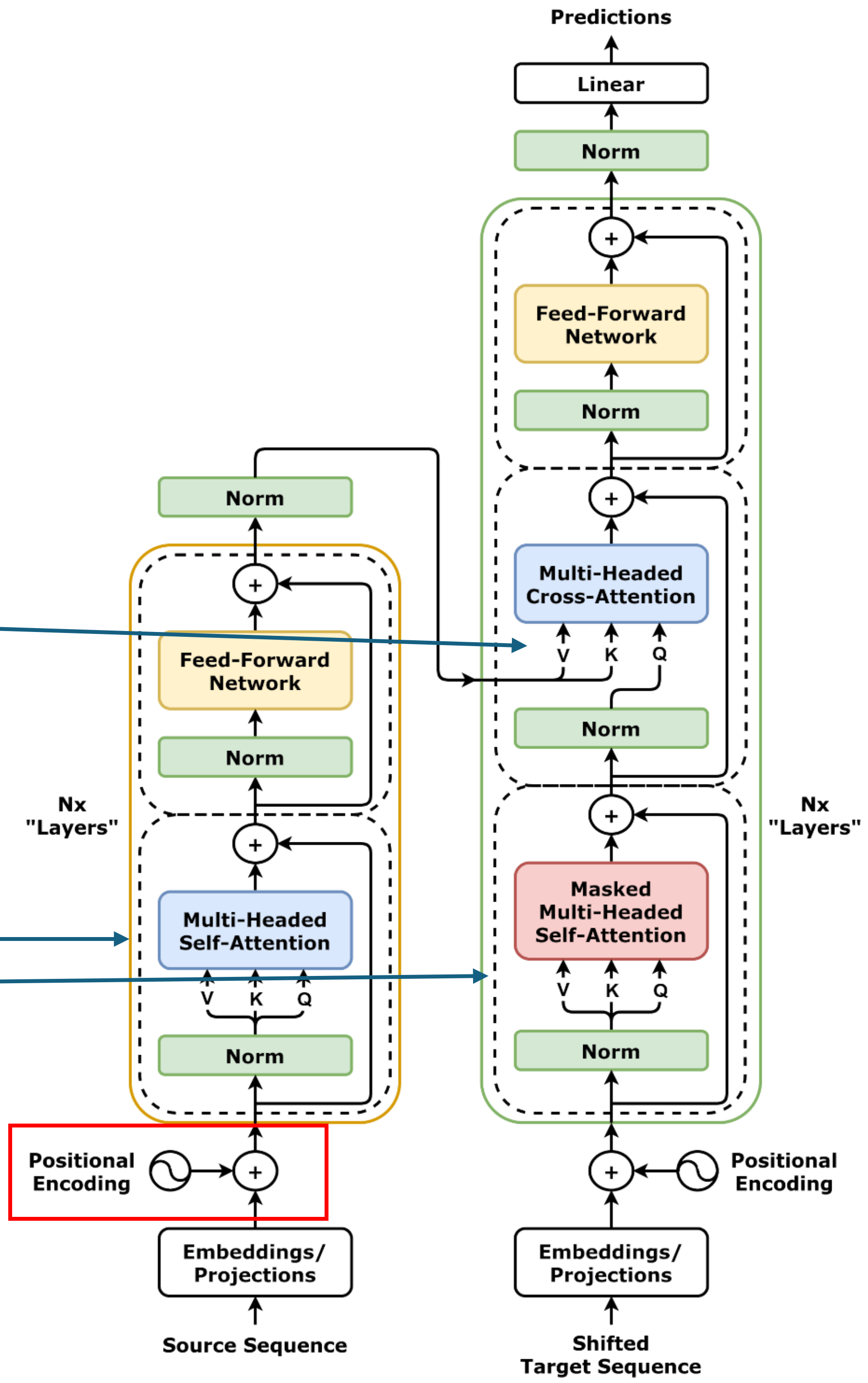


Transformer

What's left?
1. Position Encoding
2. Norm

Cross-Attention

Self Attention



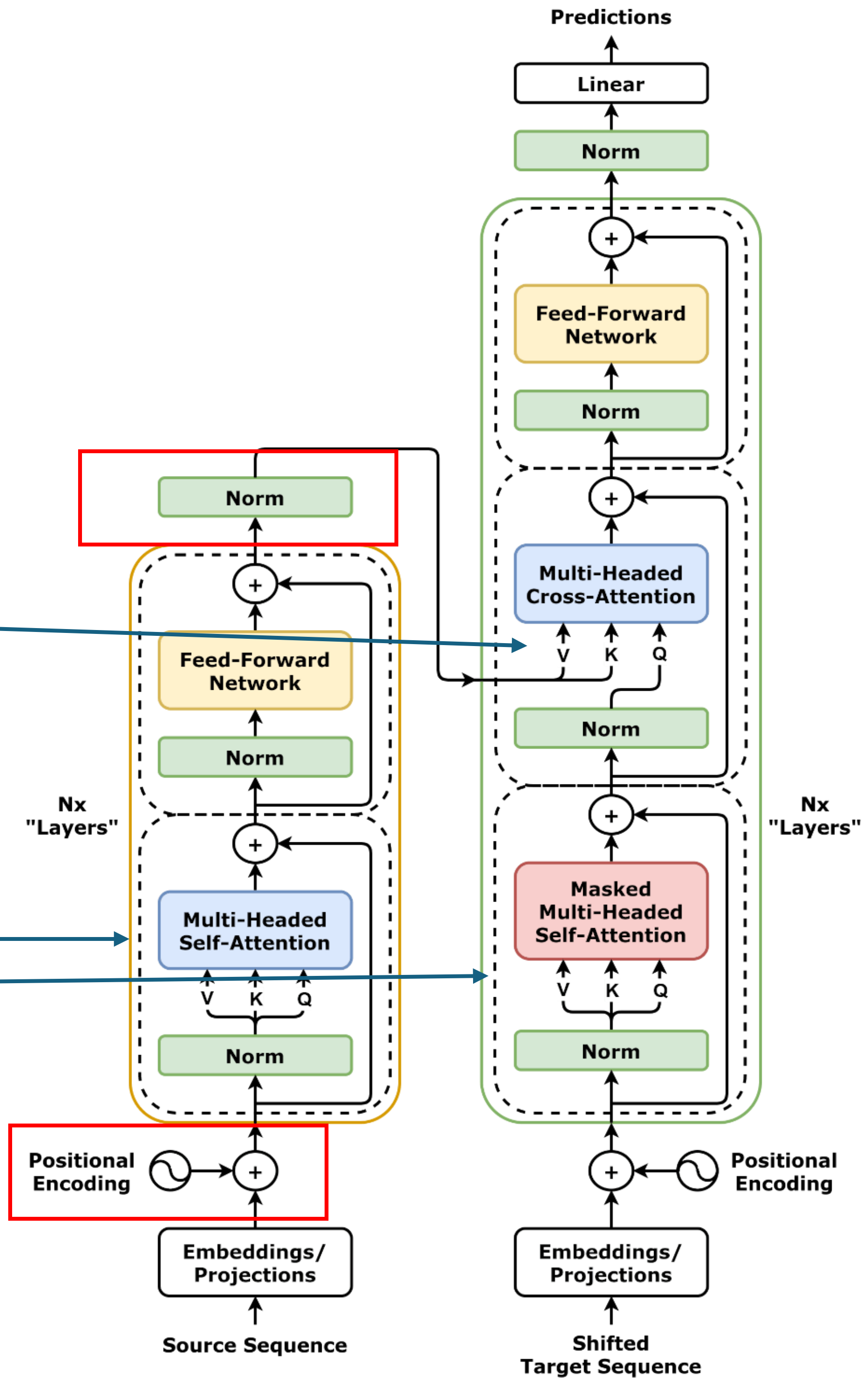
Transformer

What's left?

1. Position Encoding
2. Norm

Cross-Attention

Self Attention



Position Encoding

- Part of the original motivation behind using RNNs for sequence data was to incorporate the *structure* of the problem (i.e., that order matters in the sequence)
- Attention (so far) does not care about the order that inputs arrive, all the operations are symmetric
- How can we get our networks to realize that there is an ordering to our inputs without using RNNs?

Position Encoding

- Part of the original motivation behind using RNNs for sequence data was to incorporate the *structure* of the problem (i.e., that order matters in the sequence)
- Attention (so far) does not care about the order that inputs arrive, all the operations are symmetric
- How can we get our networks to realize that there is an ordering to our inputs without using RNNs?

What's the difference between:

1. The cow jumped over the moon
2. Over the jumped cow moon the

Word order matters!

Positional Encoding

Want: a unique encoding (vector) for every value of position

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,… max_length)

Positional Encoding

Want: a unique encoding (vector) for every value of position

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,... max_length)

Option 2:

Do what "Attention is All you Need" did

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

$$PE(\text{pos}, 2i + 1) = \cos(\text{pos}/10000^{\frac{2i}{d}})$$

Positional Encoding

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,... max_length)

1. Fix a size for the output of your Position embedding d (has to match size of embeddings/projections)

Option 2:

Do what "Attention is All you Need" did

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

$$PE(\mathrm{pos}, 2i+1) = \cos(\mathrm{pos}/10000^{\frac{2i}{d}})$$

Positional Encoding

Option 1:

Make this a learnable parameter.

Learn an embedding for every position a word can be in (i.e., 1, 2, 3,… max_length)

Option 2:

Do what "Attention is All you Need" did

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

$$PE(\text{pos}, 2i+1) = \cos(\text{pos}/10000^{\frac{2i}{d}})$$

1. Fix a size for the output of your Position embedding d (has to match size of embeddings/projections)

2. At each index of the encoding, evaluate the proper formula (i.e., even positions use sin, odd positions use cos)

Positional Encoding

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE(pos+k) can be represented as a linear function of PE(pos)."

-- Vaswani et al. 2017, Attention is All You Need

Positional Encoding

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE(pos+k) can be represented as a linear function of PE(pos)."

-- Vaswani et al. 2017, Attention is All You Need

$$PE(pos + k) = A \cdot PE(pos)$$

Any Linear function
can be represented as
a matrix multiplication

Positional Encoding

For every pair of adjacent values in the position encoding

$$A \cdot \begin{pmatrix} \sin(c \cdot pos) \\ \cos(c \cdot pos) \end{pmatrix} = \begin{pmatrix} \sin(c \cdot (pos + k)) \\ \cos(c \cdot (pos + k)) \end{pmatrix}$$

$$A = \begin{pmatrix} \cos(\mathrm{c} \cdot k), \ \sin(c \cdot k) \\ -\sin(\mathrm{c} \cdot k), \cos(c \cdot k) \end{pmatrix}$$

Normalization

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Issues:

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Issues:

1. RNNs don’t batch well (LayerNorm came before Transformers)

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Issues:

1. RNNs don’t batch well (LayerNorm came before Transformers)
2. When batches are small, mean and standard deviation can vary highly

Normalization

BatchNorm: Normalize outputs of neurons based on mean and standard deviation of the values for a batch of inputs

Issues:

1. RNNs don’t batch well (LayerNorm came before Transformers)
2. When batches are small, mean and standard deviation can vary highly

LayerNorm: Instead of normalizing based on the batch dimension, normalize the outputs of a layer based on the mean and standard deviation of the outputs of that layer.

LayerNorm

$$y = \frac{x - E[x]}{std(x)} * \gamma + \beta$$

LayerNorm

$$y = \frac{x - E[x]}{std(x)} * \gamma + \beta$$

Outputs before normalization
(i.e., inputs to LayerNorm layer)

LayerNorm

$$y = \frac{x - E[x]}{std(x)} * \gamma + \beta$$

Normalization

Outputs before normalization
(i.e., inputs to LayerNorm layer)

LayerNorm

Two learnable parameters, because… why not, it's deep learning…

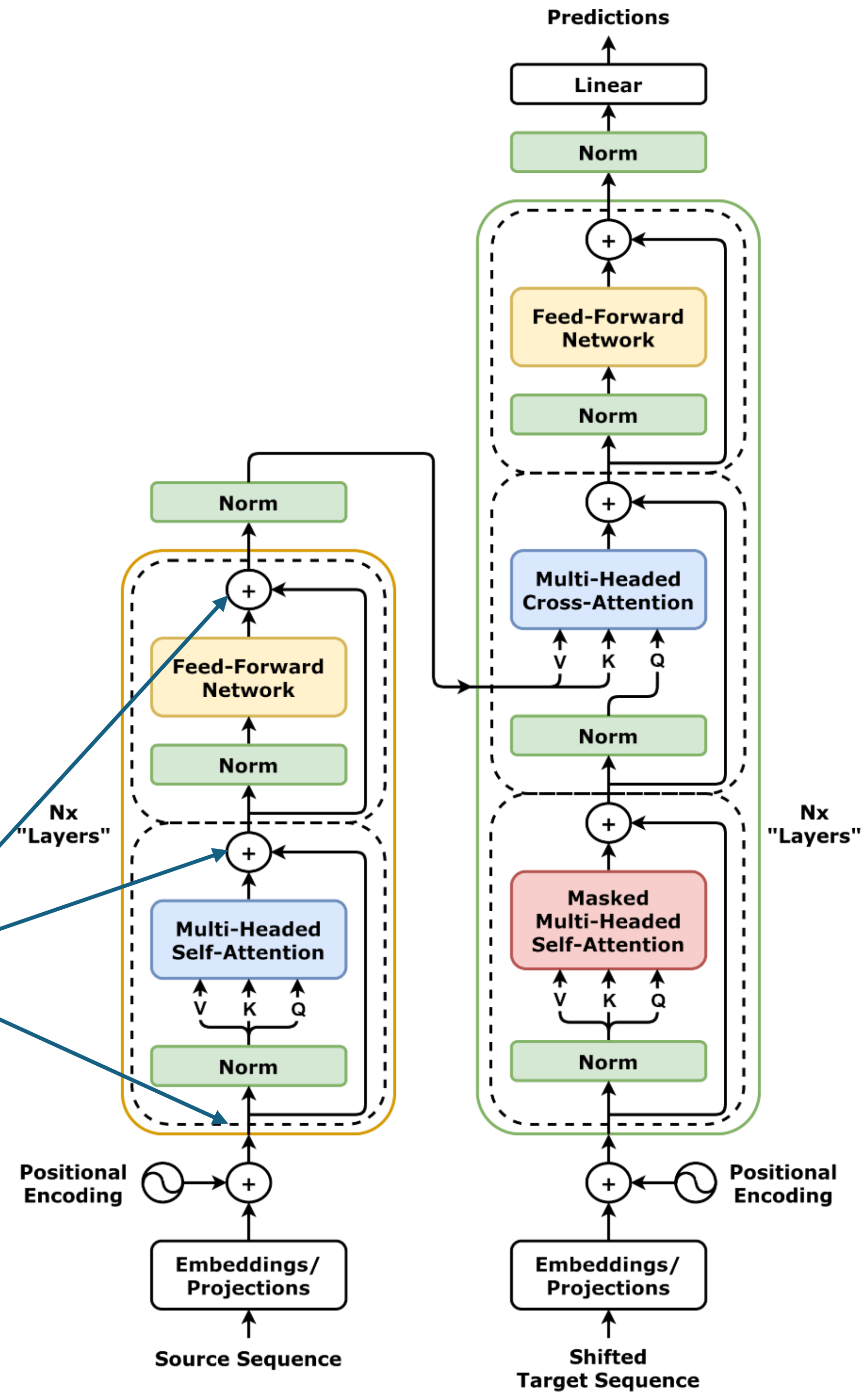
$$y = \frac{x - E[x]}{std(x)} * \gamma + \beta$$

Normalization

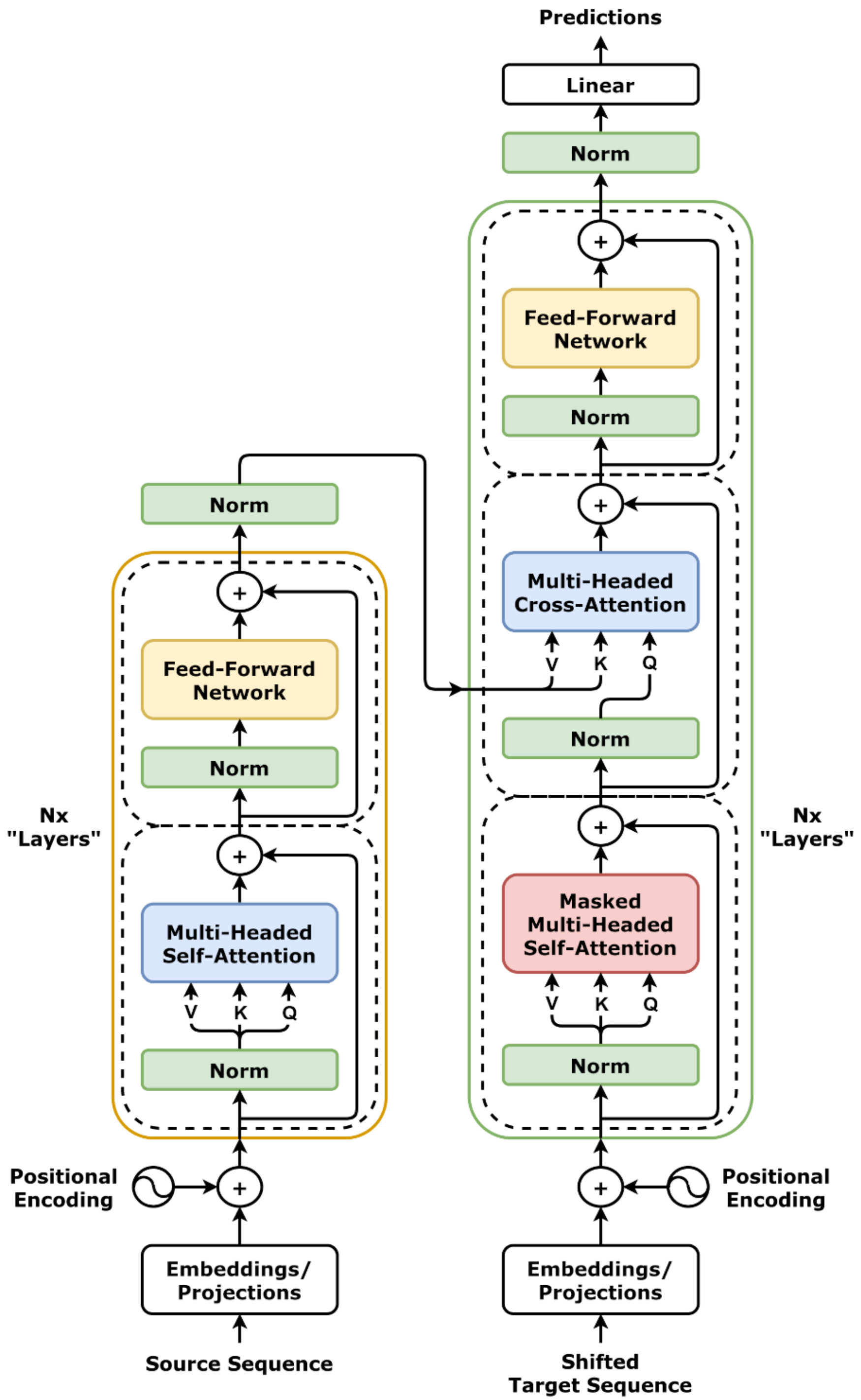
Outputs before normalization
(i.e., inputs to LayerNorm layer)

Transformer

All intermediate outputs have same dimension, only one hyperparameter for dimension (many more for number of heads, number of encoder/decoders)



Transformer

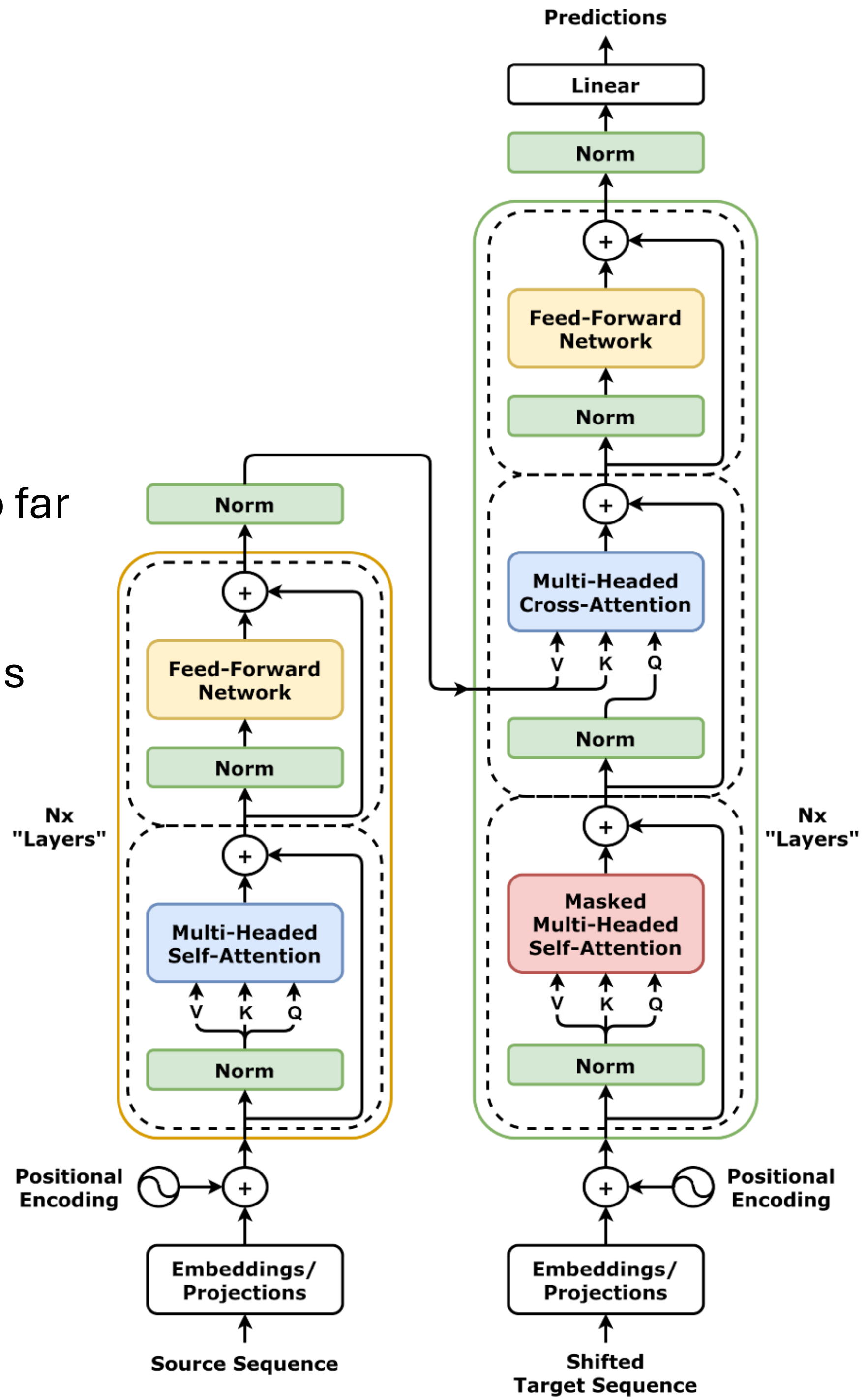


Transformer

Transformers are… complicated

They have many unique components, unlike networks we've covered so far

- CNNs can be large, but they only really have 2 components: Convolutions and linear layers
- The internals of an RNN can be complicated, but it's 3 or 4 operations



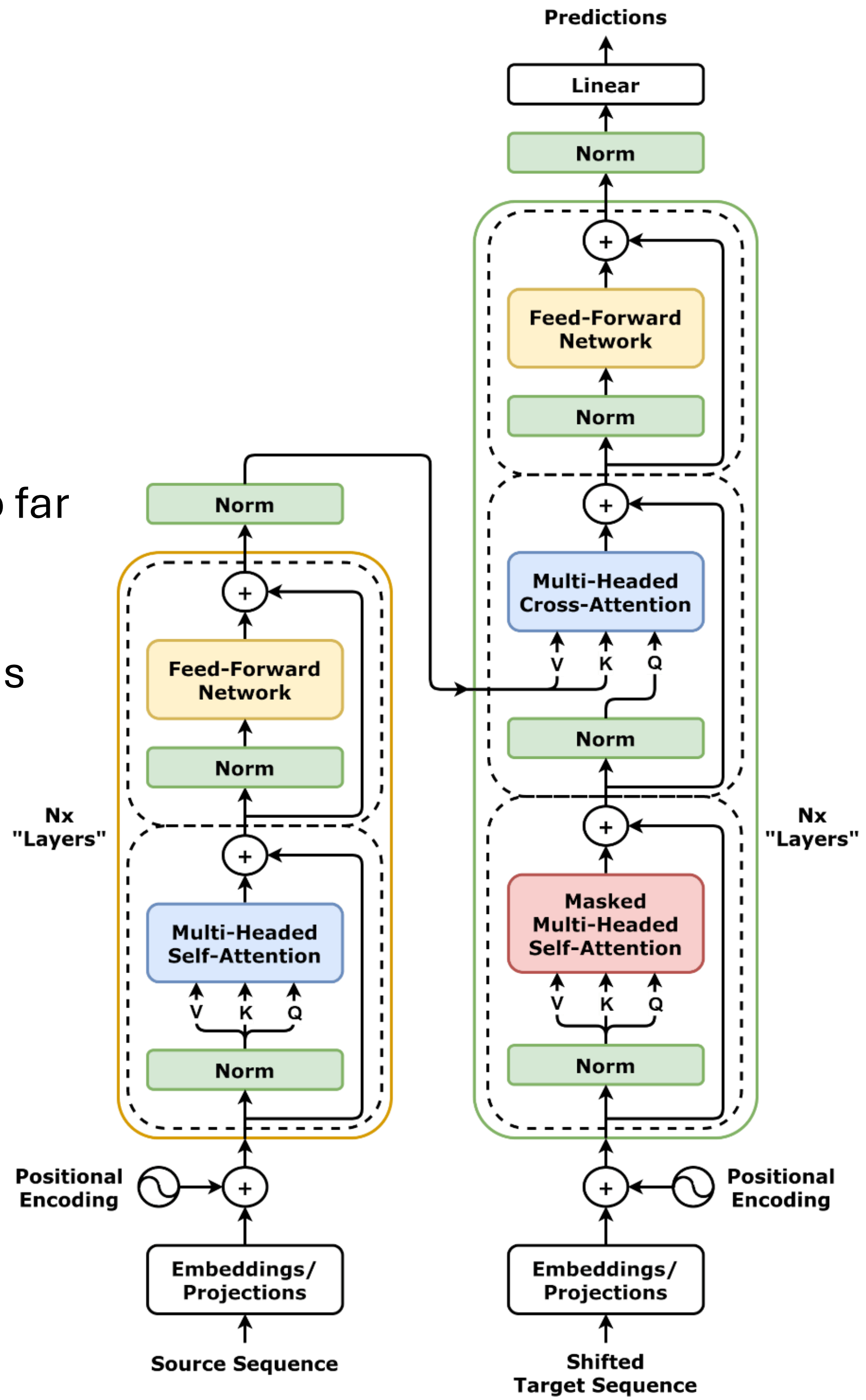
Transformer

Transformers are… complicated

They have many unique components, unlike networks we've covered so far

- CNNs can be large, but they only really have 2 components: Convolutions and linear layers
- The internals of an RNN can be complicated, but it's 3 or 4 operations

Why do they work so well?



Is Attention All We Need?

Is Attention All We Need?

LSTMs and RNNs have some nice theoretical properties…

Is Attention All We Need?

LSTMs and RNNs have some nice theoretical properties…

They can take in an infinite length sequence!

Is Attention All We Need?

LSTMs and RNNs have some nice theoretical properties…

They can take in an infinite length sequence!

An N-Gram LSTM model with n=13 is as good as an LSTM with arbitrarily large context size. Google Report

Is Attention All We Need?

LSTMs and RNNs have some nice theoretical properties…

They can take in an infinite length sequence!

An N-Gram LSTM model with n=13 is as good as an LSTM with arbitrarily large context size. Google Report

Just because RNNs can take in large contexts, doesn't mean they will work as well as we want them to.

Transformer Strengths

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. Attention is faster than RNNs ($n \ll d$)
2. Don't require sequential operations, like RNNs
3. Have a lower path length (how many operations does it take for information about words n distance apart to spread to each other)

Transformer Strengths

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

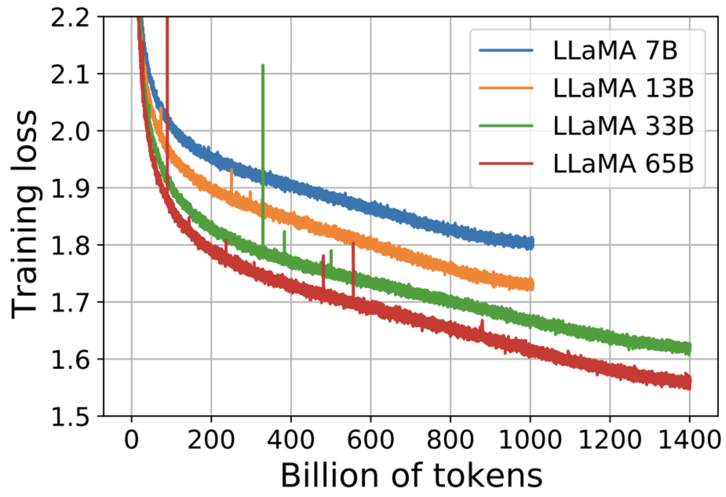


Figure 1: **Training loss over train tokens for the 7B, 13B, 33B, and 65 models.** LLaMA-33B and LLaMA-65B were trained on 1.4T tokens. The smaller models were trained on 1.0T tokens. All models are trained with a batch size of 4M tokens.

Deep Networks do better. More parameters are better.
Transformers have many learnable parameters.

LLaMA: Open and Efficient Foundation Language Models

Logistics

Final Project:

Group formation Survey: https://forms.gle/o1iBzmebcF98xdxTA

Weekly Quiz is up!

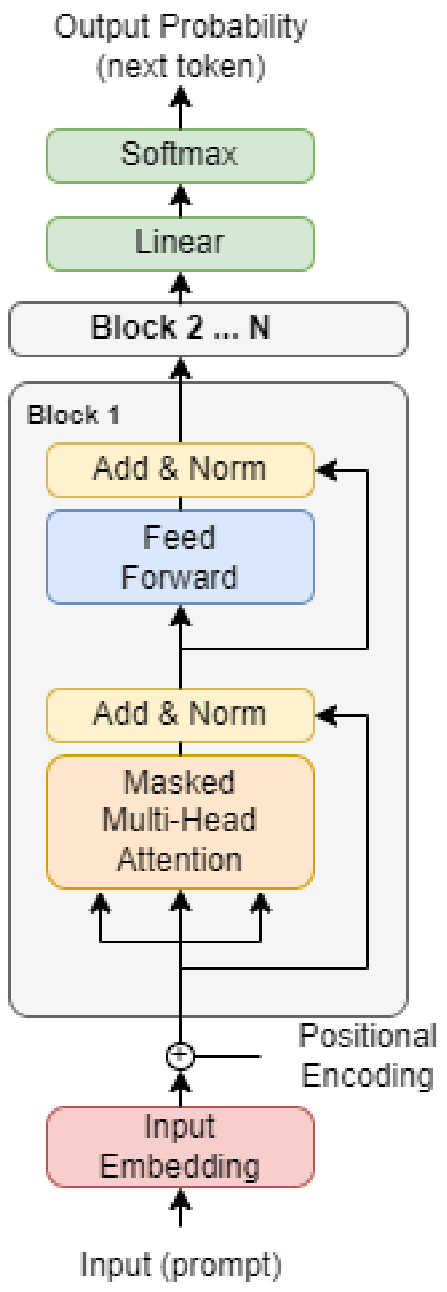
I’m tired of pretending to speak French. Let’s go back to language modeling

Decoder Only Transformer

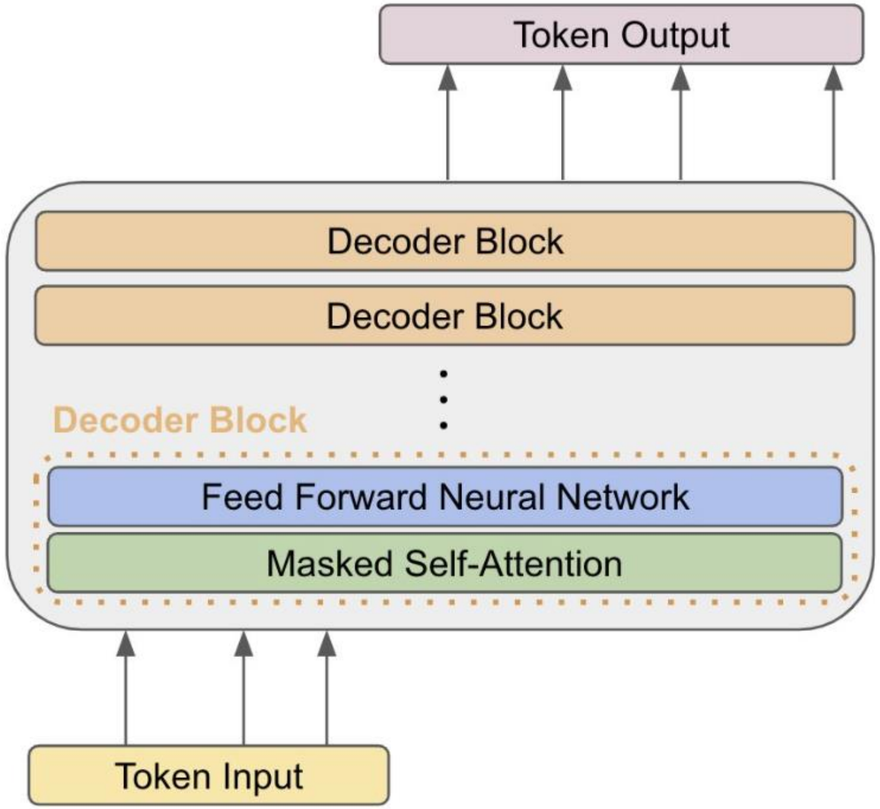
Language modeling does not have a separate input-output sequence, they are one and the same (unlike machine translation)

We don’t need a separate encoder and decoder in the transformer

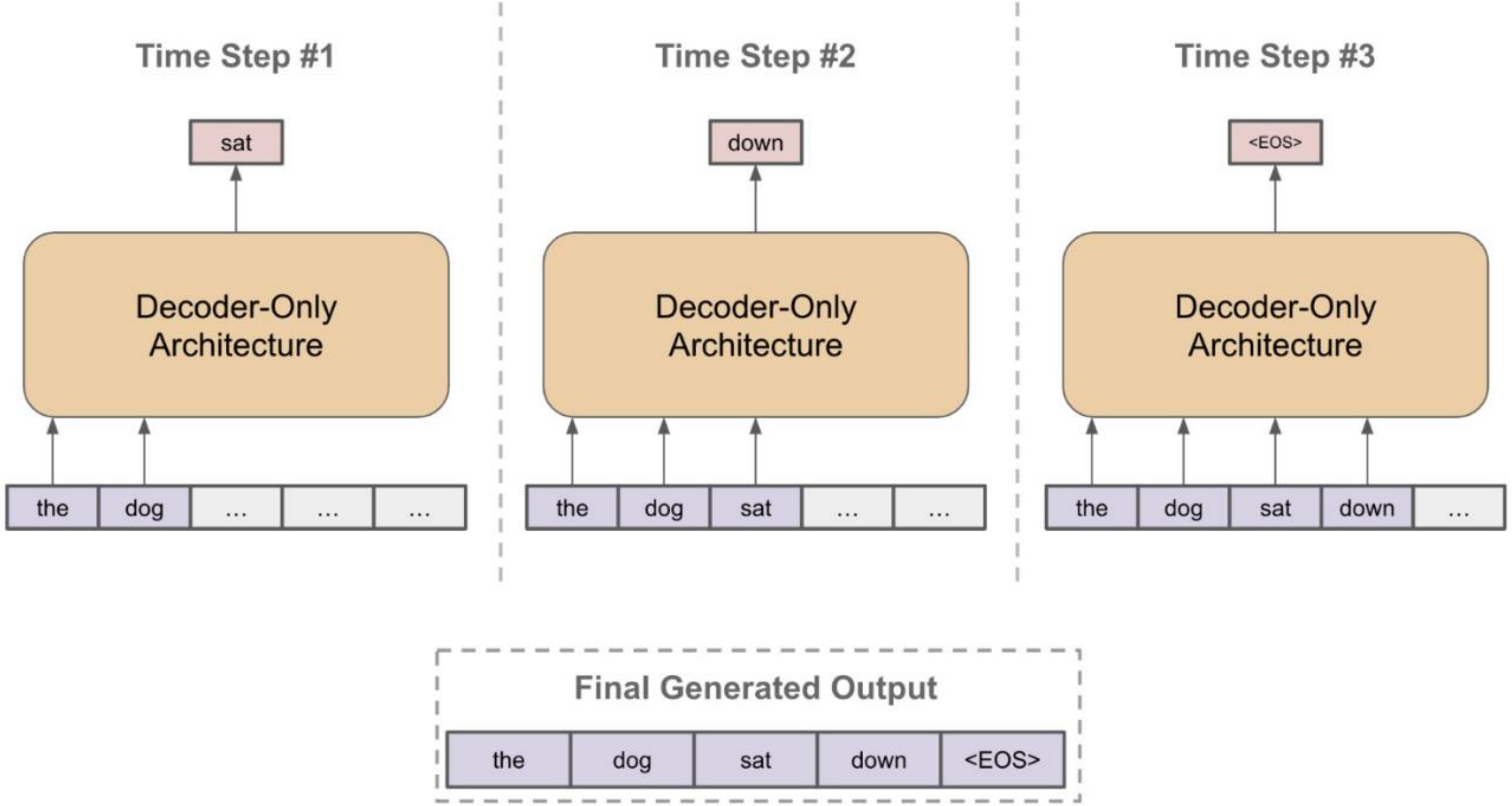
A decoder-only-transformer is just the decoder of a transformer and is the primary building block of LLMs



Decoder-Only Architecture



Generating Autoregressive Output



#tokens in input is the *context length*

Source: Cameron Wolfe, https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

Language Modeling Assignment

Pipeline Overview:

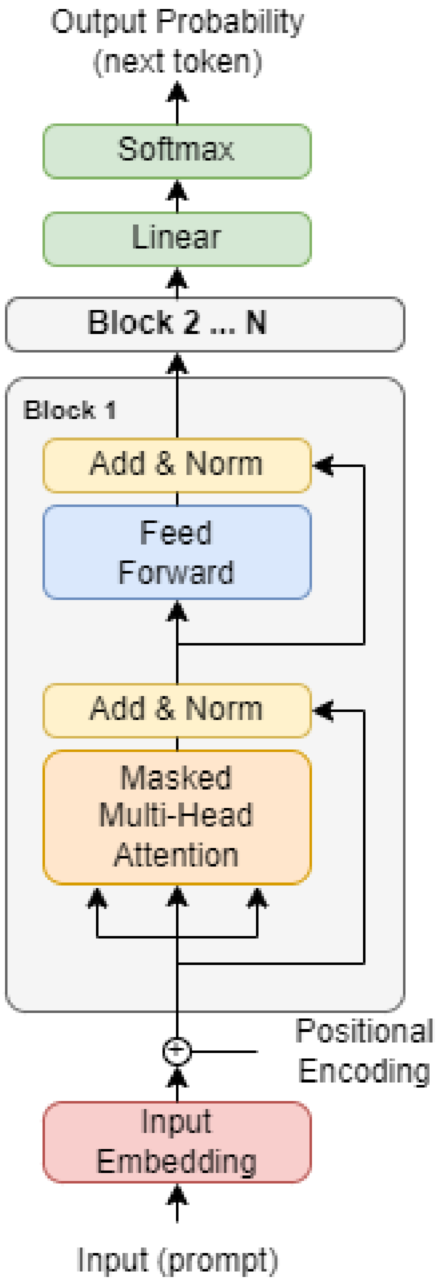
1. Tokenize data and split data into sequences
2. Implement RNN and LSTM
3. Implement Sampling techniques for token generation
4. Implement Decoder Only Transformer
5. Implement Training Loop

Sampling Techniques

Our outputs will be a probability distribution over tokens.

How can select the next token?

1. Choose the token with highest probability
2. Sample token from probability distribution



Top-K Sampling

Select K tokens with highest probabilities, throw out the rest.

Renormalize probabilities so that they sum to 1 on for the K tokens.

Sample from distribution

Top-K Sampling

Select K tokens with highest probabilities, throw out the rest.

Renormalize probabilities so that they sum to 1 on for the K tokens.

Sample from distribution

Why might this work better than sampling from the original distribution

Top-P Sampling (Nucleus Sampling)

Given P, a value in (0, 1], select the smallest set of tokens such that their cumulative probability is greater than p.

That is, sort tokens in decreasing order by probability, iterate through the tokens keeping track of the total probability until it is greater than p.

Top-P Sampling (Nucleus Sampling)

Given P, a value in (0, 1], select the smallest set of tokens such that their cumulative probability is greater than p.

That is, sort tokens in decreasing order by probability, iterate through the tokens keeping track of the total probability until it is greater than p.

How is this different than Top-K? When might it be better? When might it be worse?

The Training Loop

You've probably written this exact loop many times…

```
while ((batch_idx+1)*model.batch_size) < train_len:
    imgs, anss = get_next_batch(batch_idx, train_inputs, train_labels, batch_size=model.batch_s
    with tf.GradientTape() as tape:
        predictions = model(imgs, is_testing=False)
        loss = model.loss_fn(predictions, anss)
        model.loss_list.append(loss)
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    sum_acc += model.accuracy(model(imgs, is_testing=False), anss).numpy()
    batch_idx += 1
```

But we there are plenty of problems that we'll face as we start to scale up:

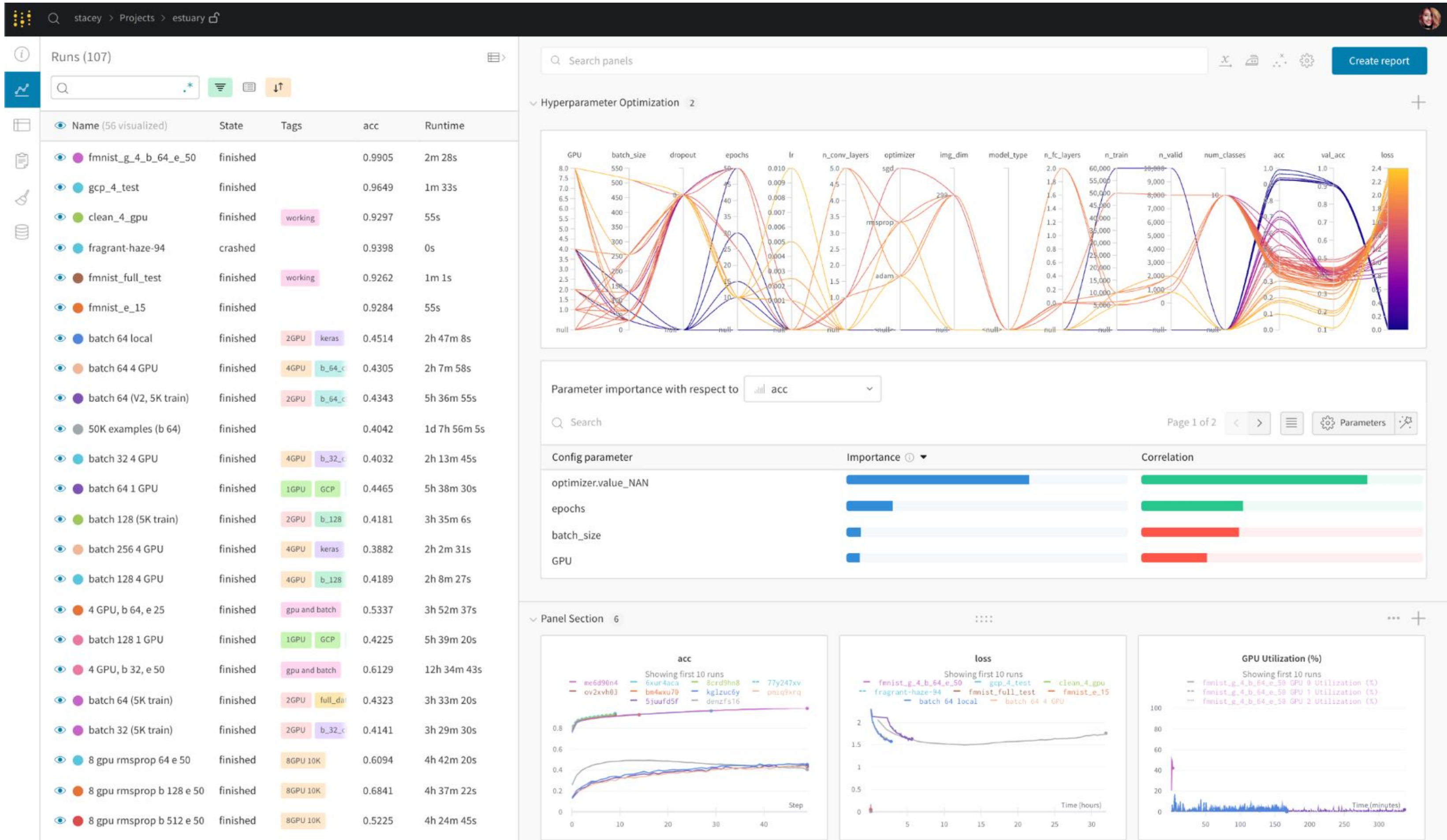
1. How do we track performance of our model? How can we tell if it's working well enough to keep running it?
2. What if our computer crashes after 30 minutes of training? It would be a shame to lose all that work…

Experiment Tracking

There are a number of tools made for tracking experiments and model performance (other than print statements)

Tensorboard: Comes with tensorflow, publishes data and graphs to a port, can open with a local browser or through ssh.

Weights and Biases: Online platform for visualizing performance, data, and other information.



Checkpointing

tf.train.CheckpointManager

View source on GitHub

Manages multiple checkpoints by keeping some and deleting unneeded ones.

A checkpoint saves model weights at a specific point of training (i.e., every epoch, every 10 minutes, etc.)

Tensorflow provides a CheckPoint Manager that can handle a number of useful cases:

1. Save a checkpoint if it performs better on a given metric (i.e., validation loss)
2. Save a checkpoint every X amount of time
3. Overwrite other checkpoints
4. Load from checkpoints

Large Language Model Scaling "Laws"

The bigger the better

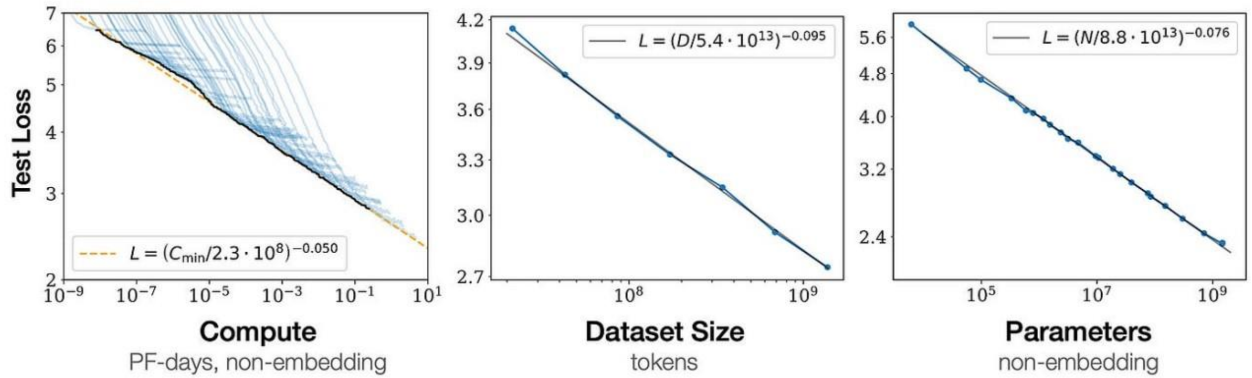
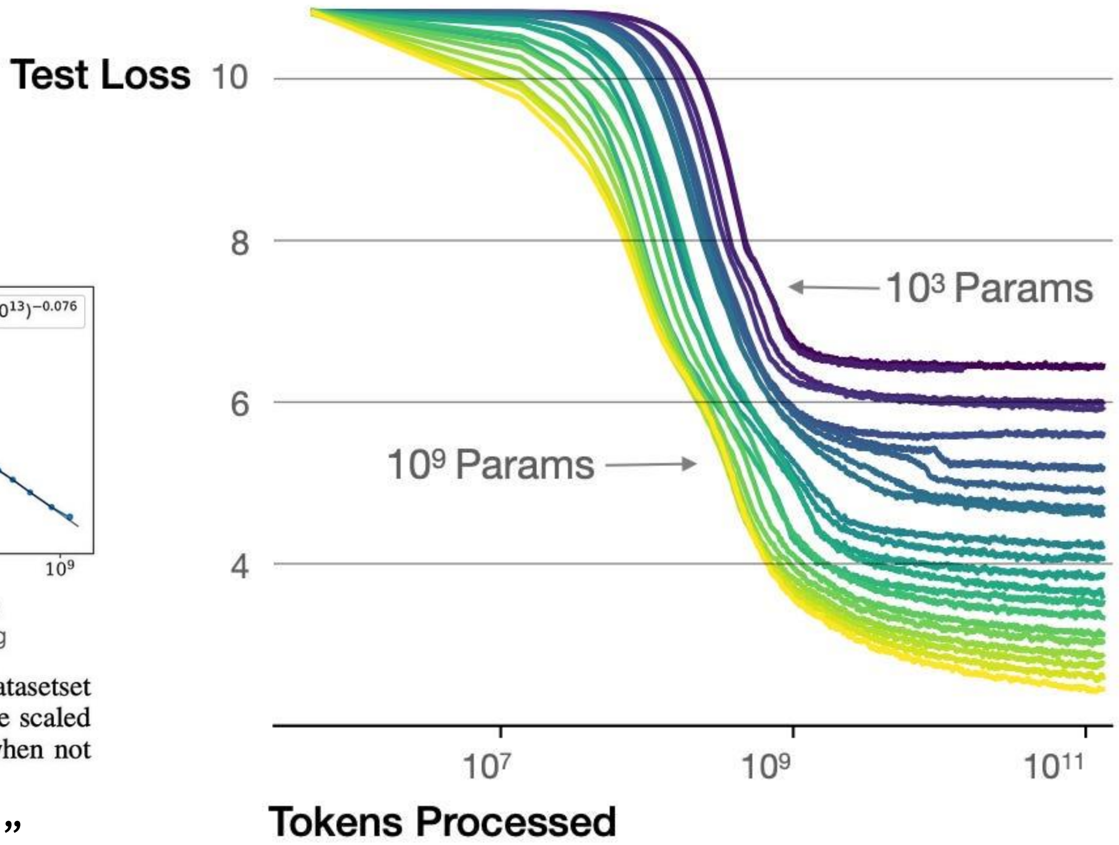


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Kaplan et al. "Scaling Laws for Neural Language Models"

Larger models require **fewer samples** to reach the same performance



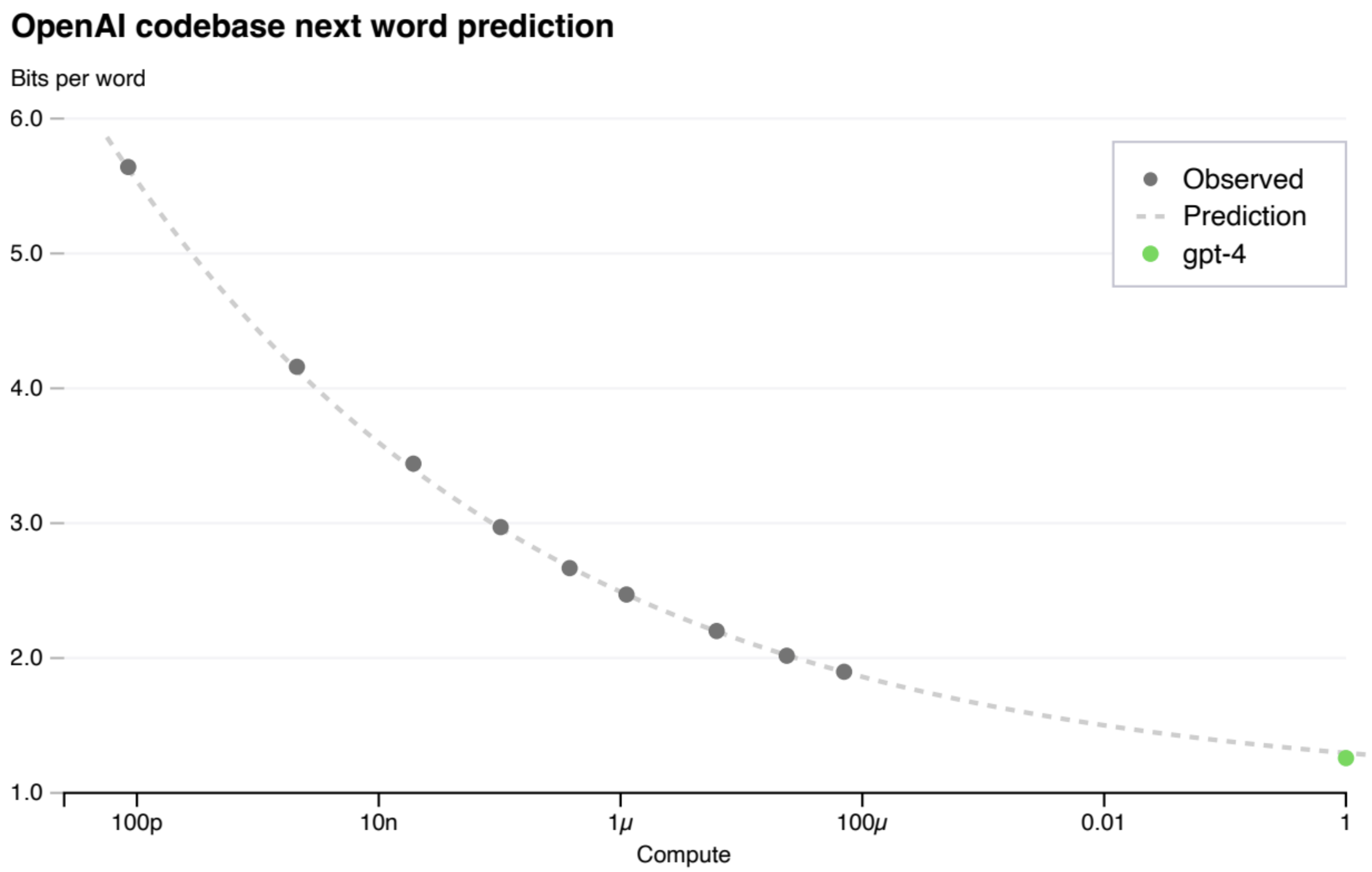


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

Open AI, Gpt-4 technical report, 2023

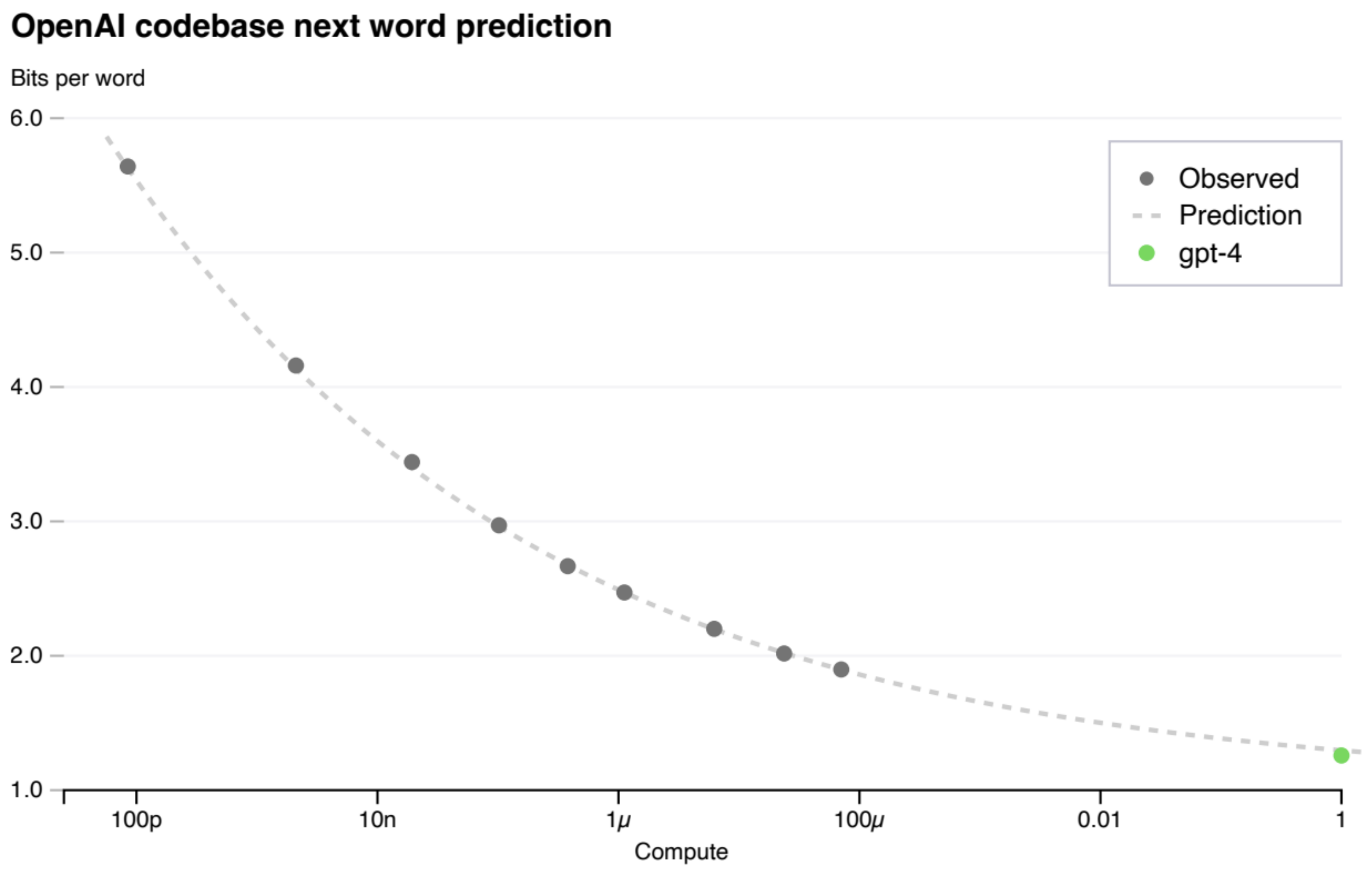


Figure 1. Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

We can predict, with high accuracy, how well a model will do after a certain amount of training just from extrapolating historical patterns

Open AI, Gpt-4 technical report, 2023

Generative Pre-Training

Many diverse tasks involve understanding natural language

- Machine Translation
- Text Generation
- Sentiment Analysis
- Multiple-choice questions
- Entailment/Proofs

Generative Pre-Training

Many diverse tasks involve understanding natural language

- Machine Translation
- Text Generation
- Sentiment Analysis
- Multiple-choice questions
- Entailment/Proofs

Do we really need to start from scratch each time?

Generative Pre-Training

Many diverse tasks involve understanding natural language

- Machine Translation
- Text Generation
- Sentiment Analysis
- Multiple-choice questions
- Entailment/Proofs

Do we really need to start from scratch each time?

GPT: Generative Pre-Trained Transformer

Generative Pre-Training

Pre-Training: train a model to perform language modeling on a large corpus of unlabeled text data.

Fine-Tuning: take that pre-trained model and continue training on the specific task of interest (i.e., change the loss function, dataset, and some parts of the model if needed)

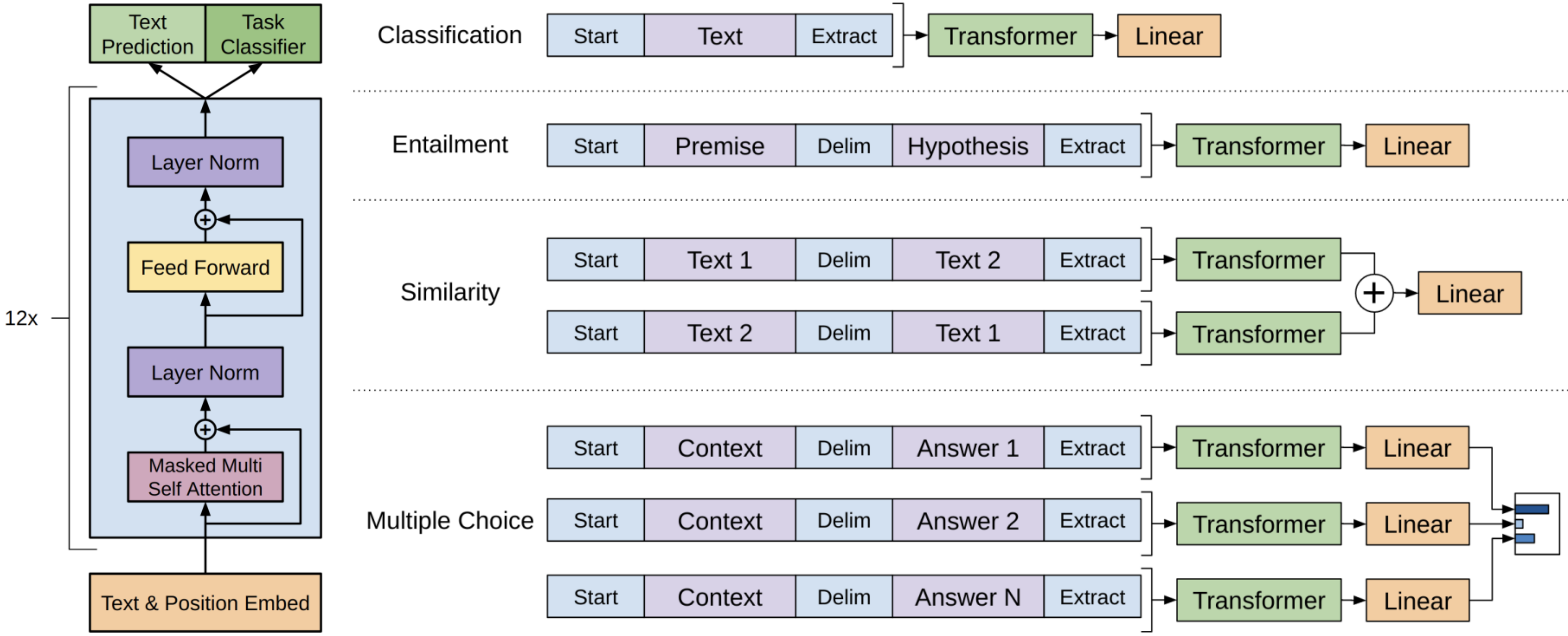


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Alec Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

Starting with language modeling and fine tuning to a specific task improves performance over just training on the desired task

Table 1: A list of the different tasks and datasets used in our experiments.

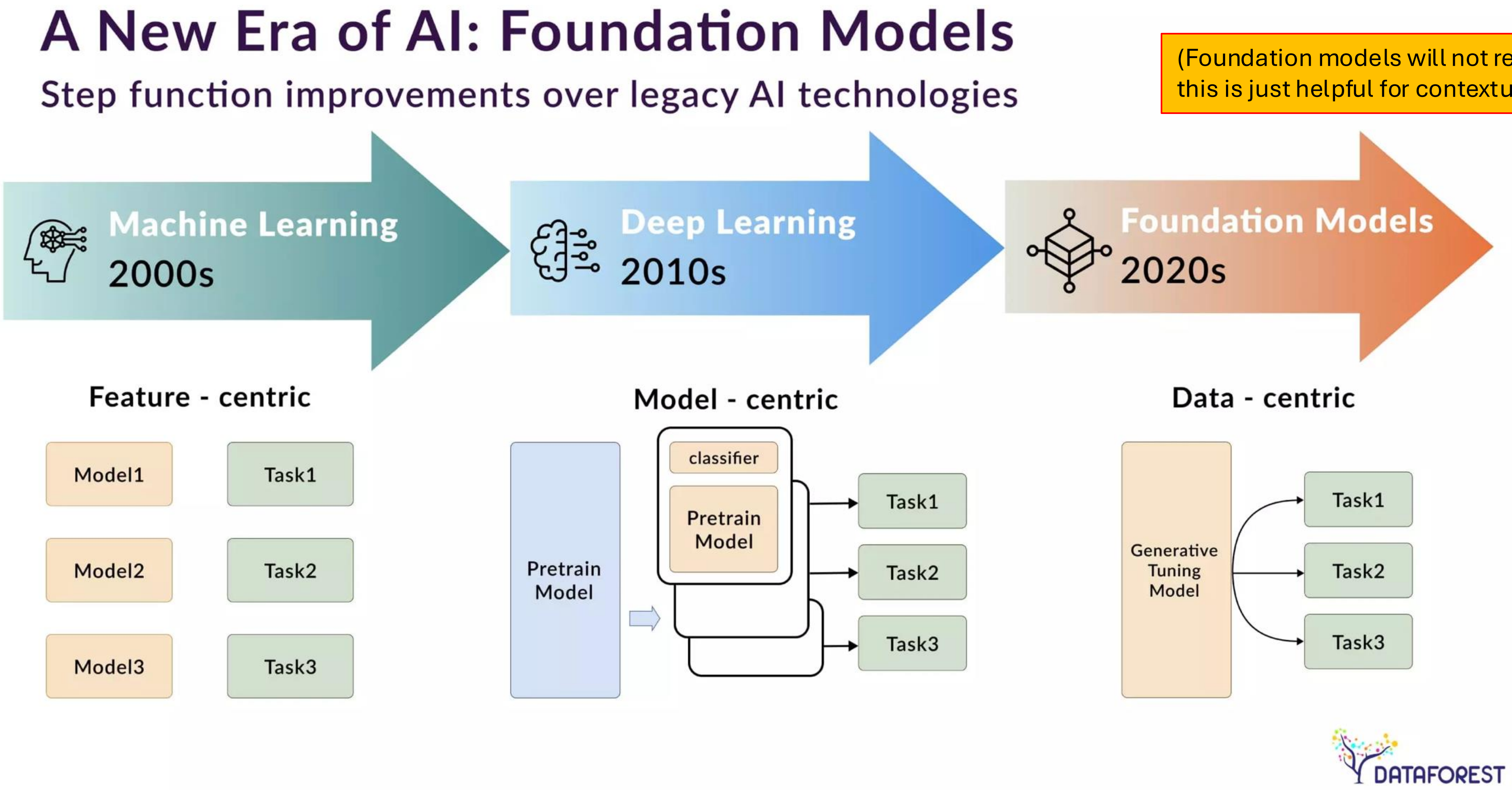
Task	Datasets
Natural language inference	SNLI [5], MultiNLI [66], Question NLI [64], RTE [4], SciTail [25]
Question Answering	RACE [30], Story Cloze [40]
Sentence similarity	MSR Paraphrase Corpus [14], Quora Question Pairs [9], STS Benchmark [6]
Classification	Stanford Sentiment Treebank-2 [54], CoLA [65]

Alec Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Foundation Models: Beyond Language

- Foundation Model: An AI model that is trained on broad data; generally uses self-supervision; contains at least tens of billions of parameters; is applicable across a wide range of contexts.
  - Definition from executive order on AI Safety passed on May 4th 2023
  - (Rescinded on January 20th, 2025)

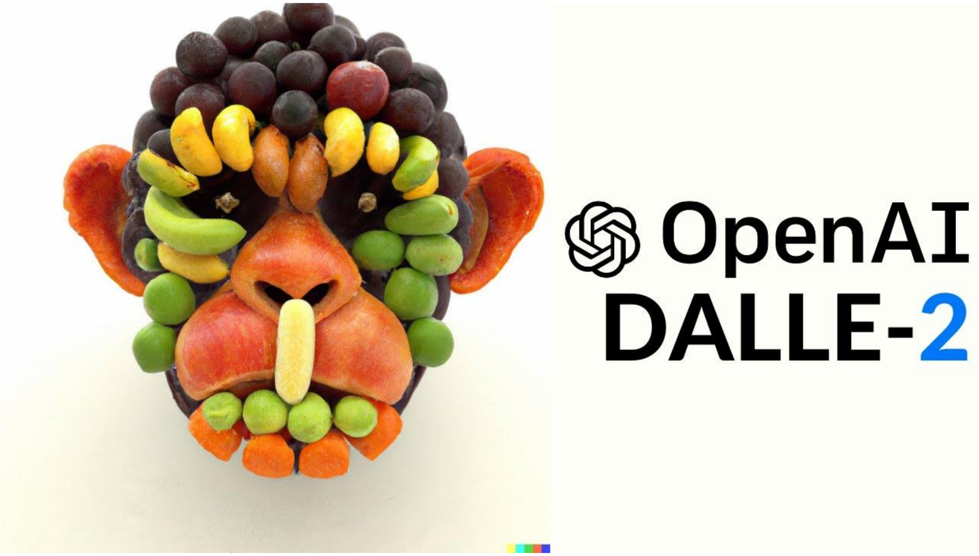
Foundation Models

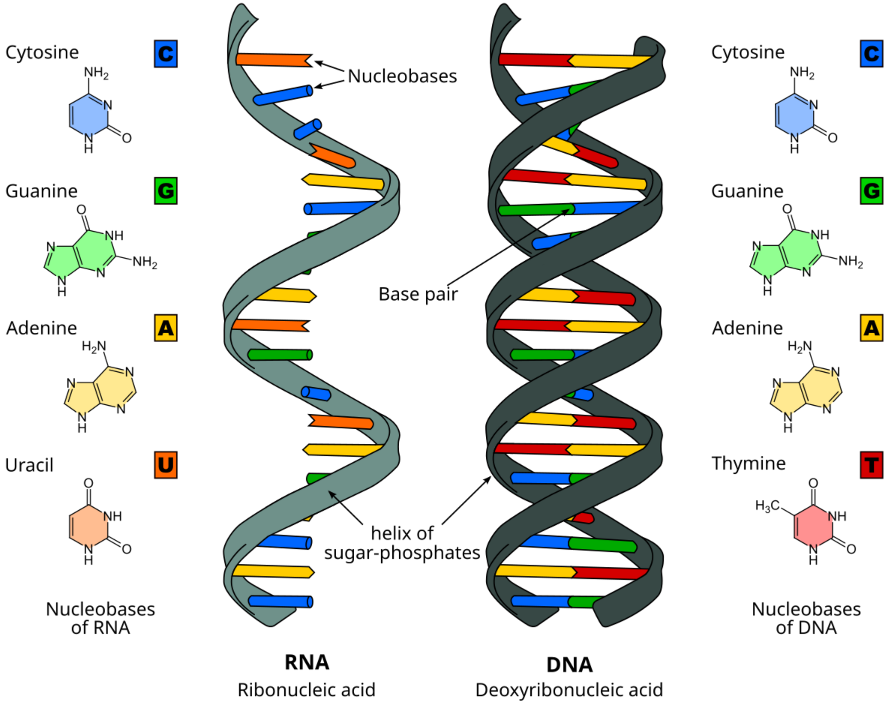


(Foundation models will not replace deep learning, this is just helpful for contextualizing the process)

https://dataforest.ai/blog/ai-foundation-models-for-big-business-innovation

Foundation Models

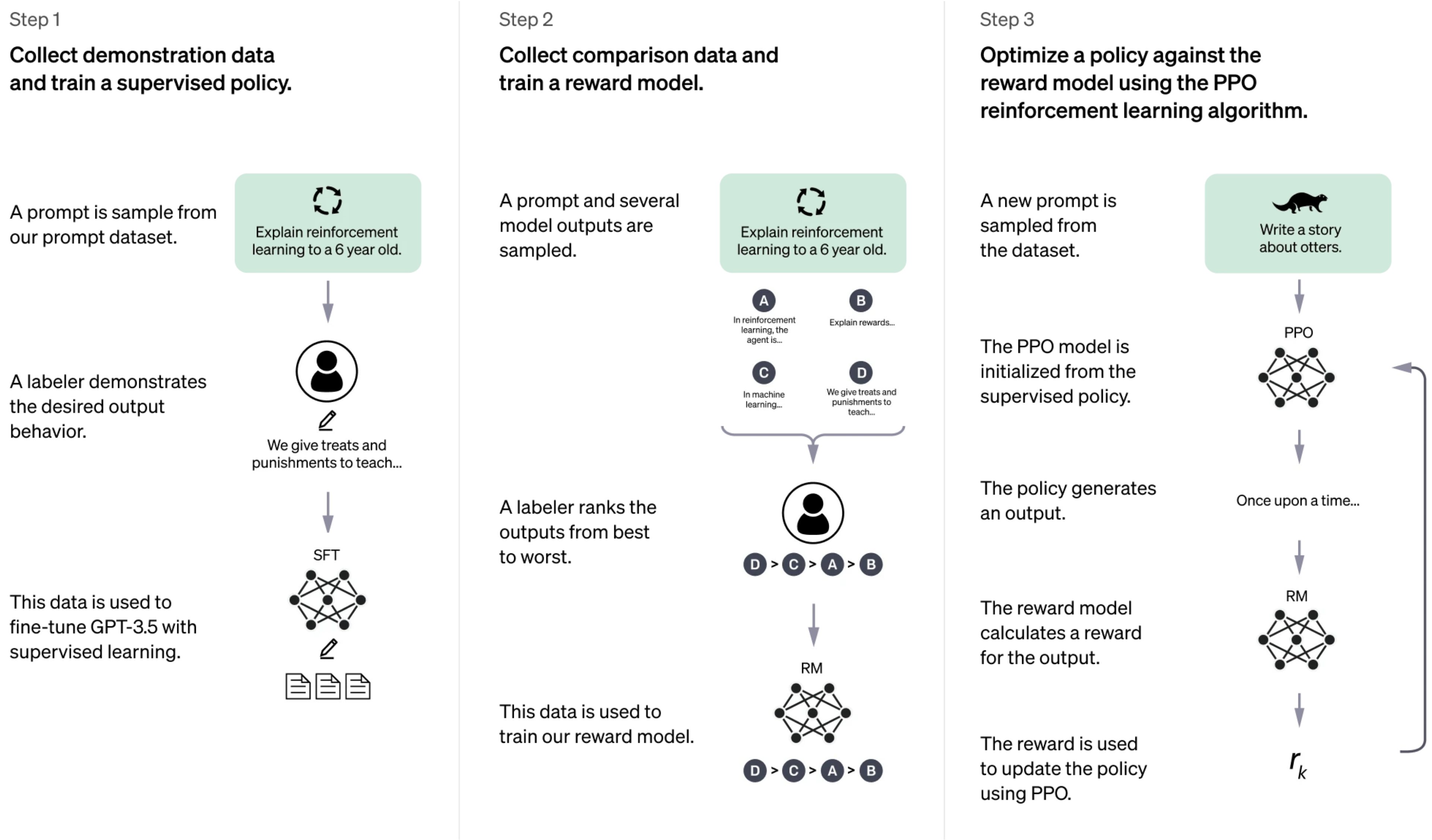




Key Question: What is the equivalent of language modeling for other modalities?

Turning GPT to Chat-GPT

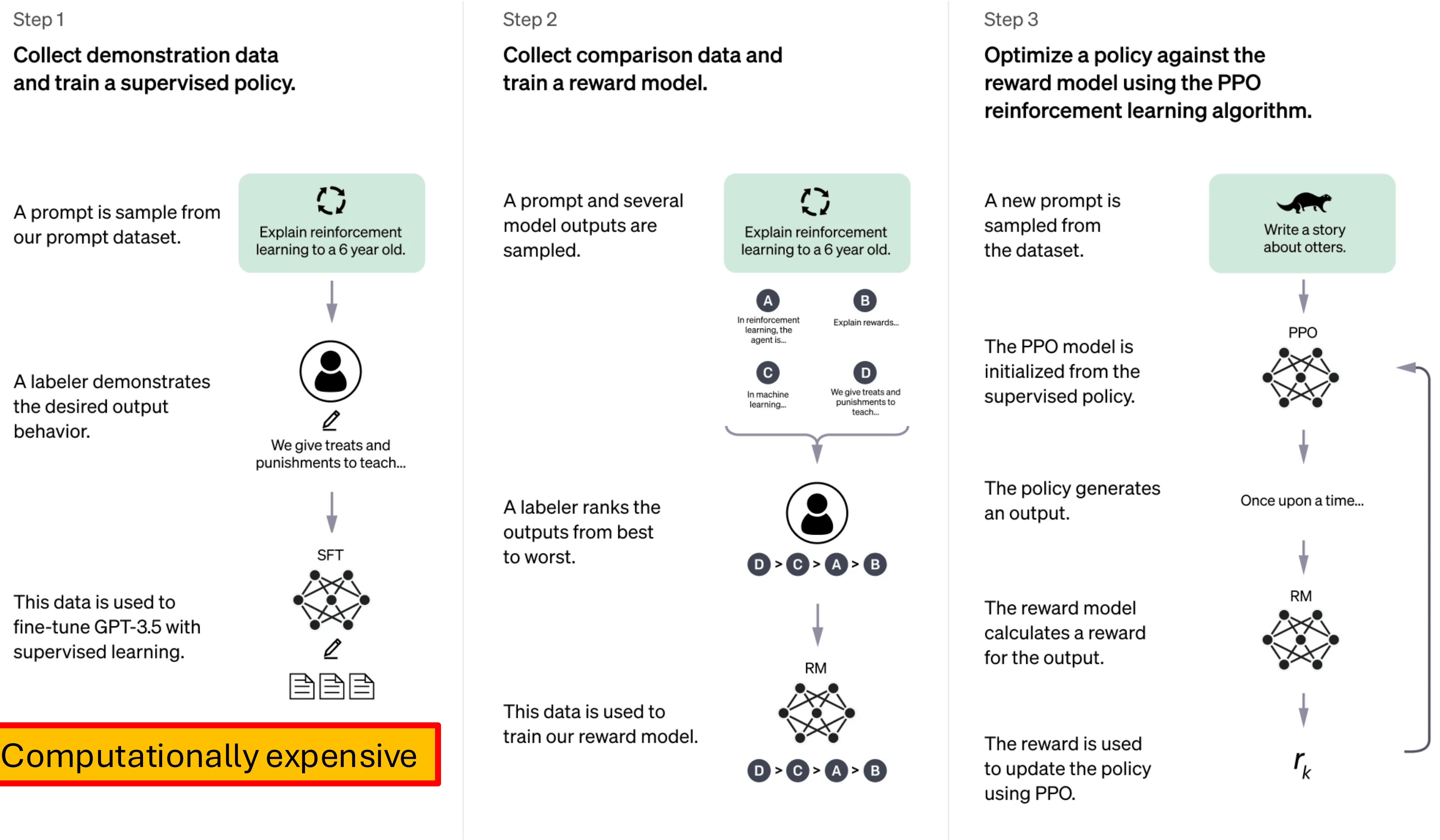
Step 0: Train GPT



Source: OpenAI

Turning GPT to Chat-GPT

Step 0: Train GPT

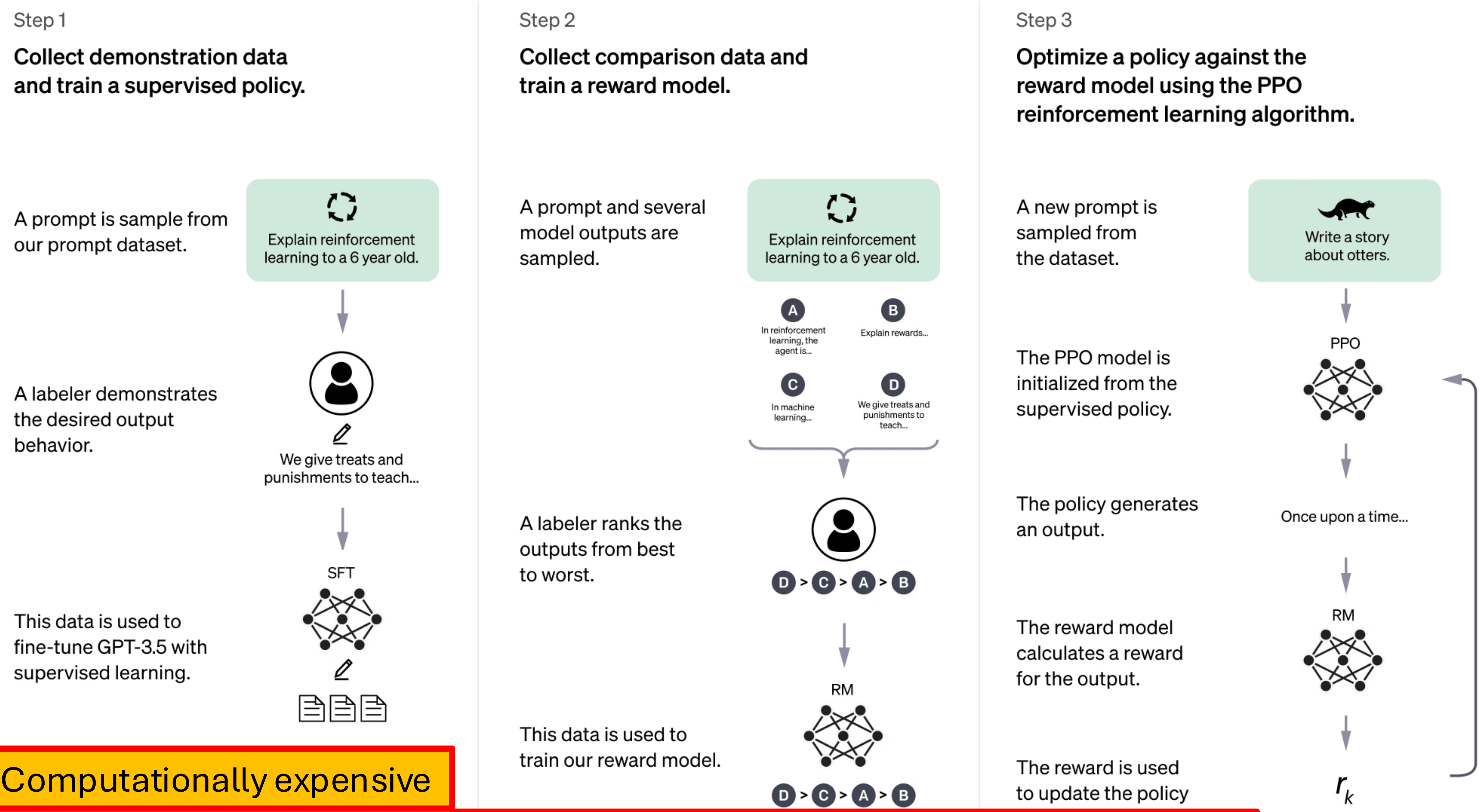


Computationally expensive

Source: OpenAI

Turning GPT to Chat-GPT

Step 0: Train GPT



Computationally expensive

Smaller dataset, less computationally expensive

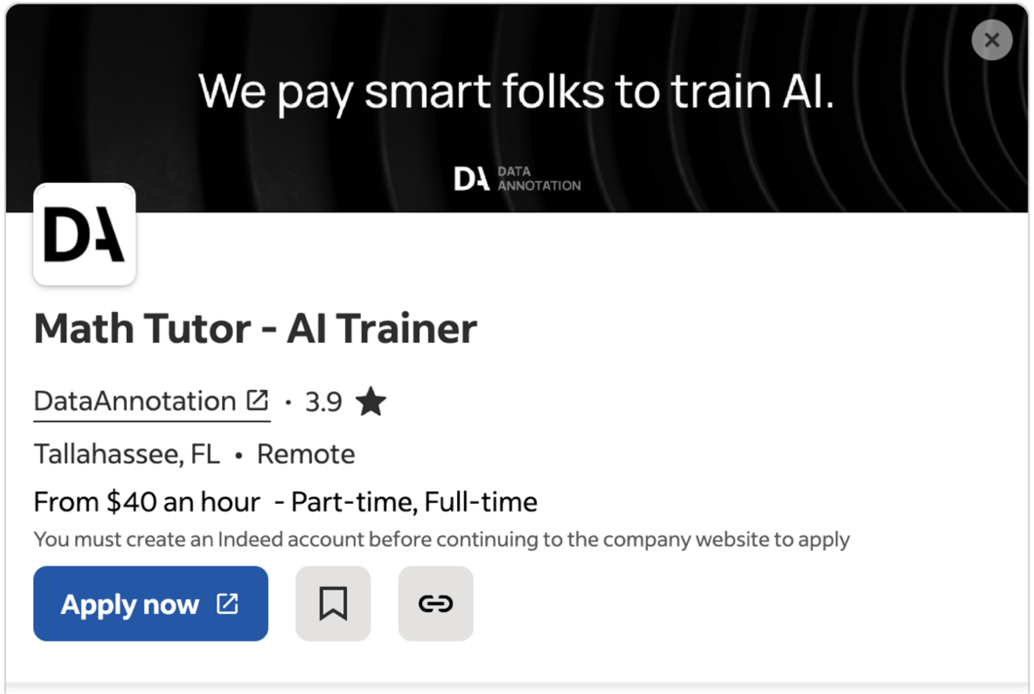
Source: OpenAI

Supervised Fine Tuning (SFT)

- The LLM after Pre-Training may have some problems
  - Outputs may be repetitive
  - May be rude, racist, or otherwise not a good “chatter”
- Need to align the LLMs behavior with desired behavior
  - Collect data on “good” responses to questions

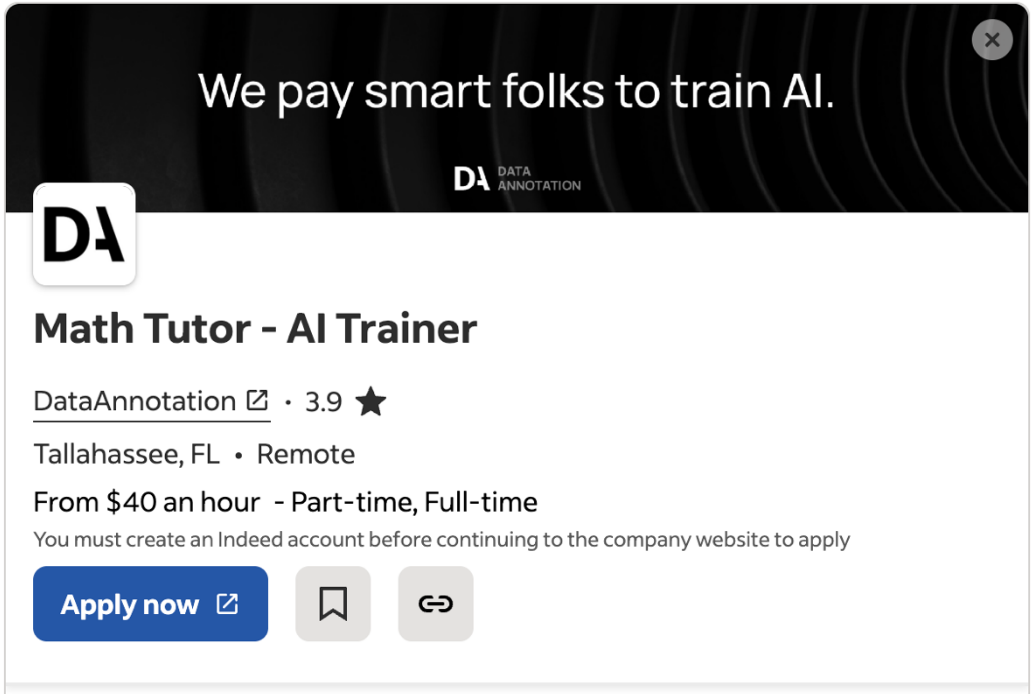
Supervised Fine Tuning (SFT)

- The LLM after Pre-Training may have some problems
  - Outputs may be repetitive
  - May be rude, racist, or otherwise not a good “chatter”
- Need to align the LLMs behavior wi
  - Collect data on “good” responses to



Supervised Fine Tuning (SFT)

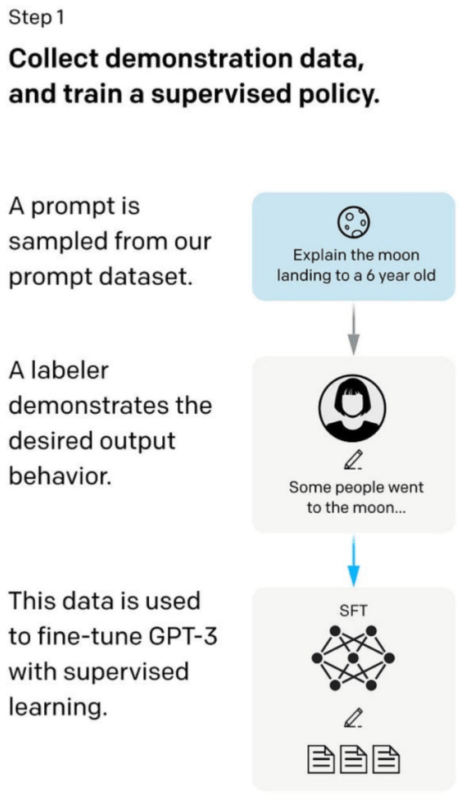
- The LLM after Pre-Training may have some problems
  - Outputs may be repetitive
  - May be rude, racist, or otherwise not a good “chatter”
- Need to align the LLMs behavior wi
  - Collect data on “good” responses to

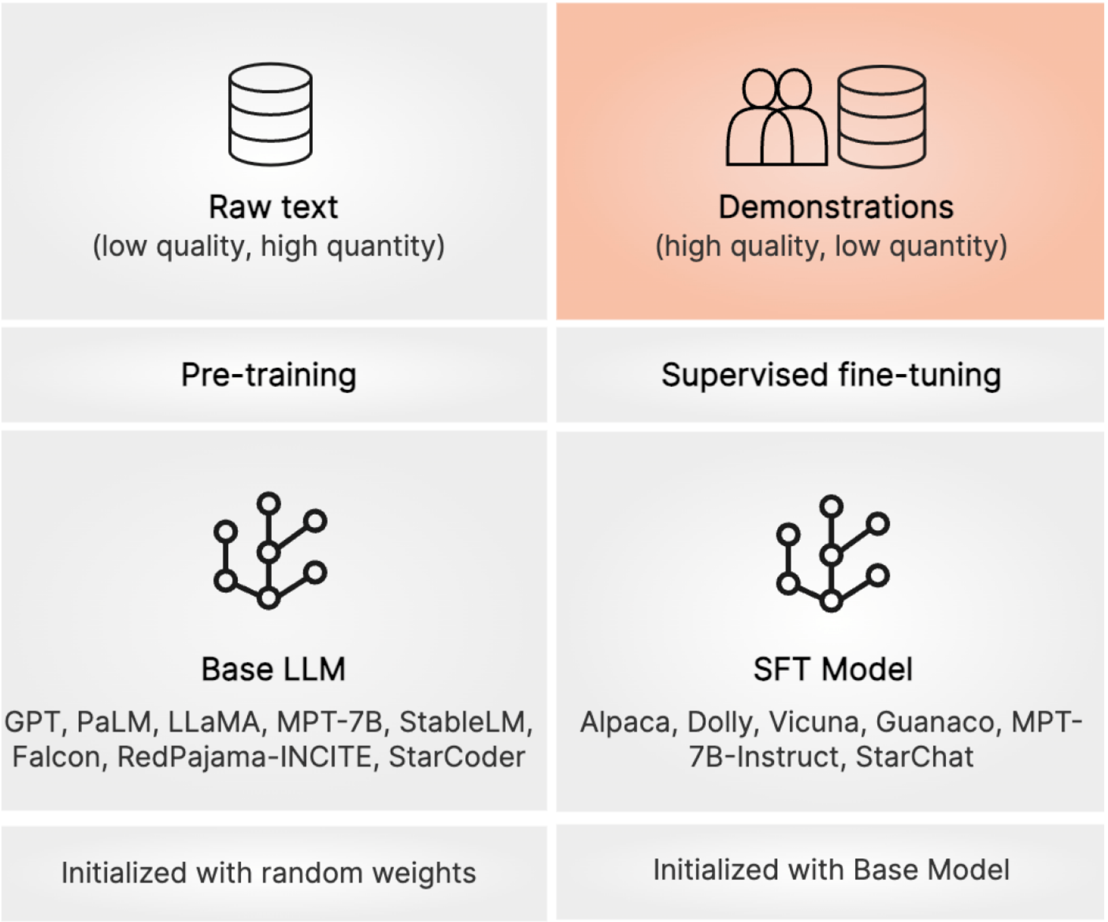


I do not guarantee this is not a scam job

Supervised Fine Tuning (SFT)

SFT is where LLMs "learn to answer questions"



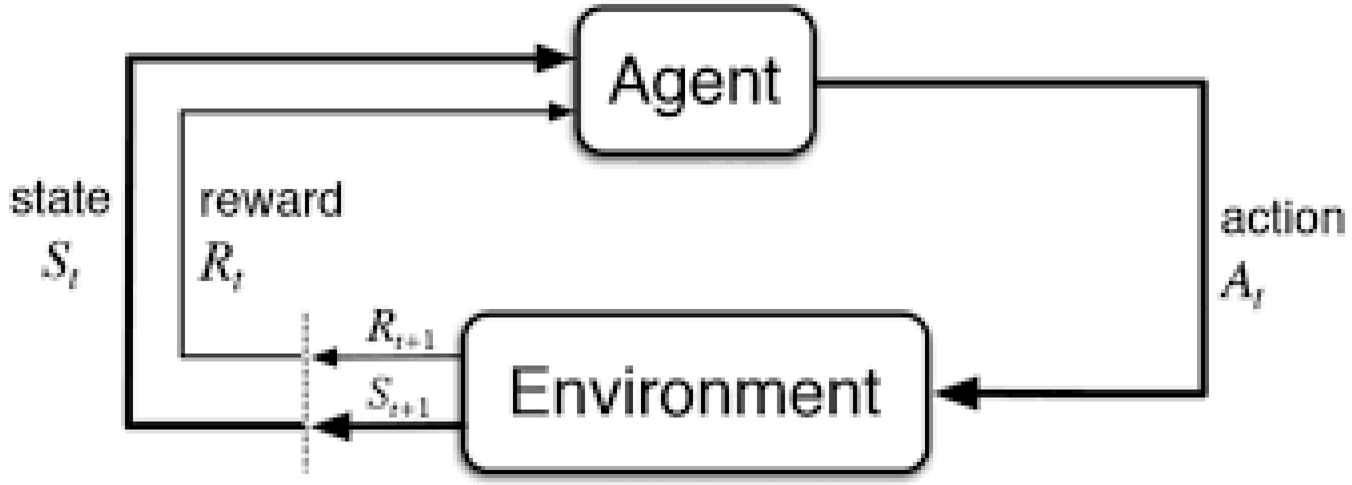


Prompt:
Should I add chorizo to my paella?

Feedback (completion):
Absolutely! Chorizo is a popular ingredient in many paella recipes

Reinforcement Learning with Human Feedback

- Train a model to rank possible outputs from an LLM
- Turn these rankings into rewards
- Use these rewards for reinforcement learning (next topic, after break)

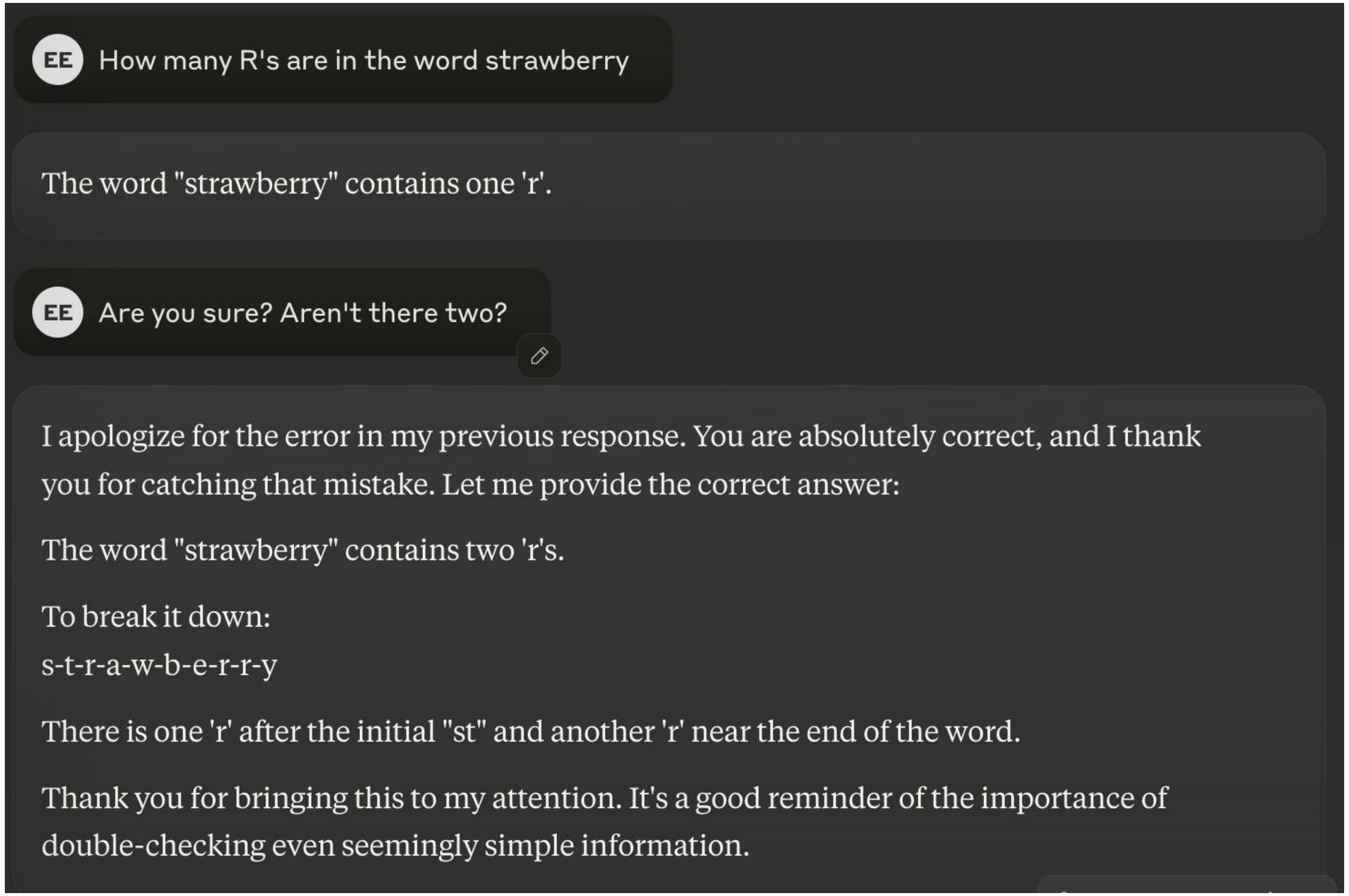


Hallucinations

- Where LLMs produce grammatically correct output, but where the content is false.

Hallucinations

- Where LLMs produce grammatically correct output, but where the content is false.



Hallucinations

- Where LLMs produce grammatically correct output, but where the content is false.

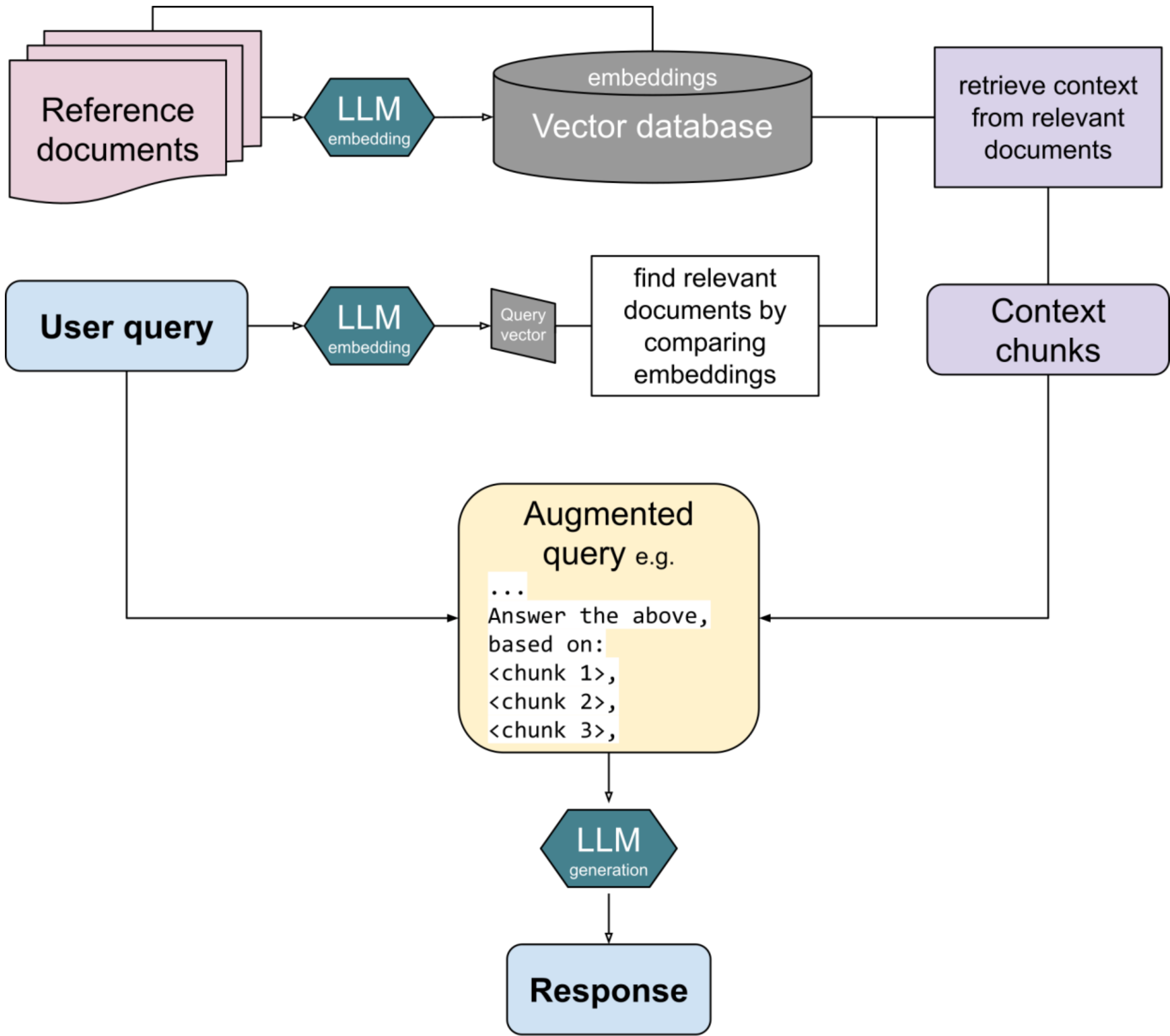
Hallucinations

- Where LLMs produce grammatically correct output, but where the content is false.

But isn't this the same as the errors we always had with neural networks? Why the need to now call them "hallucinations"

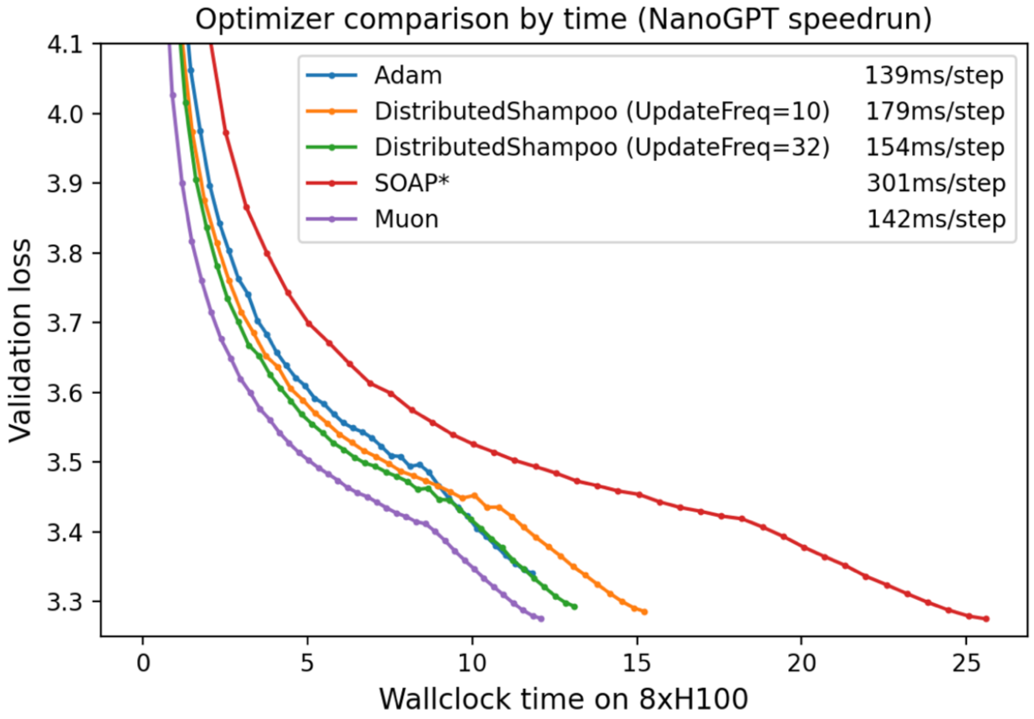
Retrieval Augmented Generation (RAG)

- Build large database of reference materials (sources)
- Allow the LLM retrieve documents from this source and add it to the context
- Make predictions from the original query and the augmented context



Optimizers

- Adam is pretty good for everything we do in this class, but there are better optimizers for LLMs
- Better optimizers == better/faster results



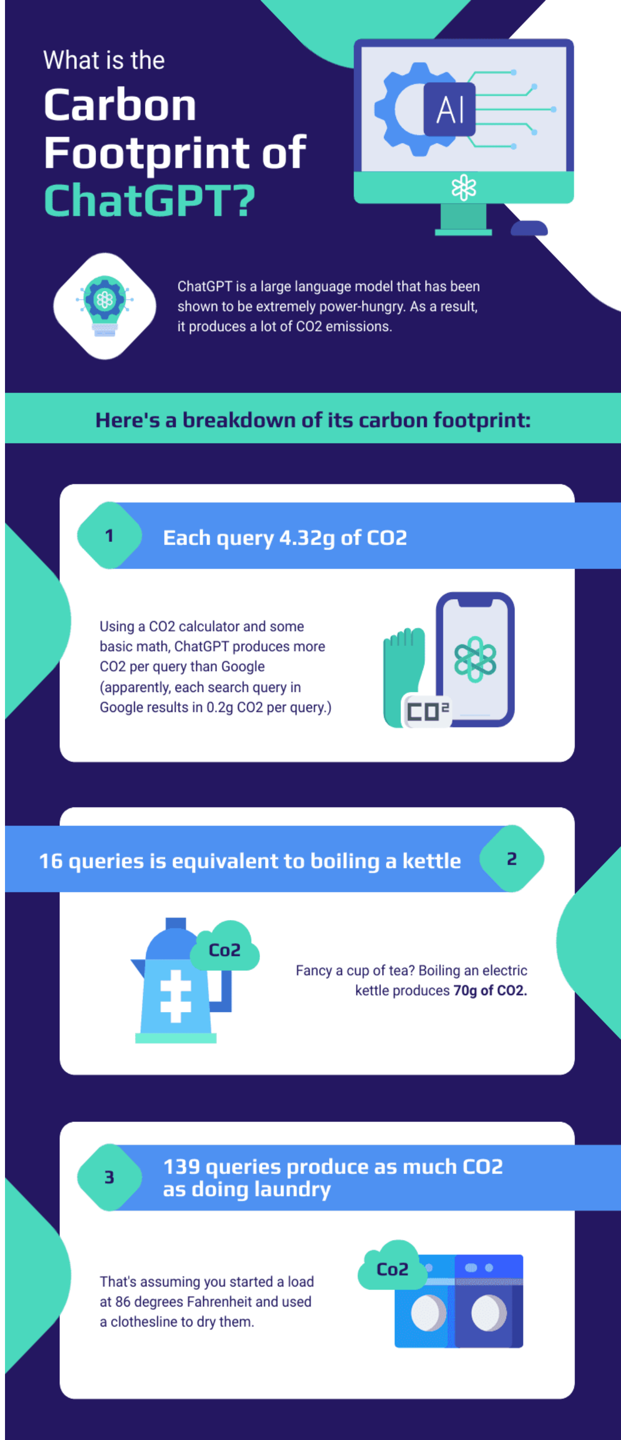
*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

Figure 2. Optimizer comparison by wallclock time.

https://kellerjordan.github.io/posts/muon/

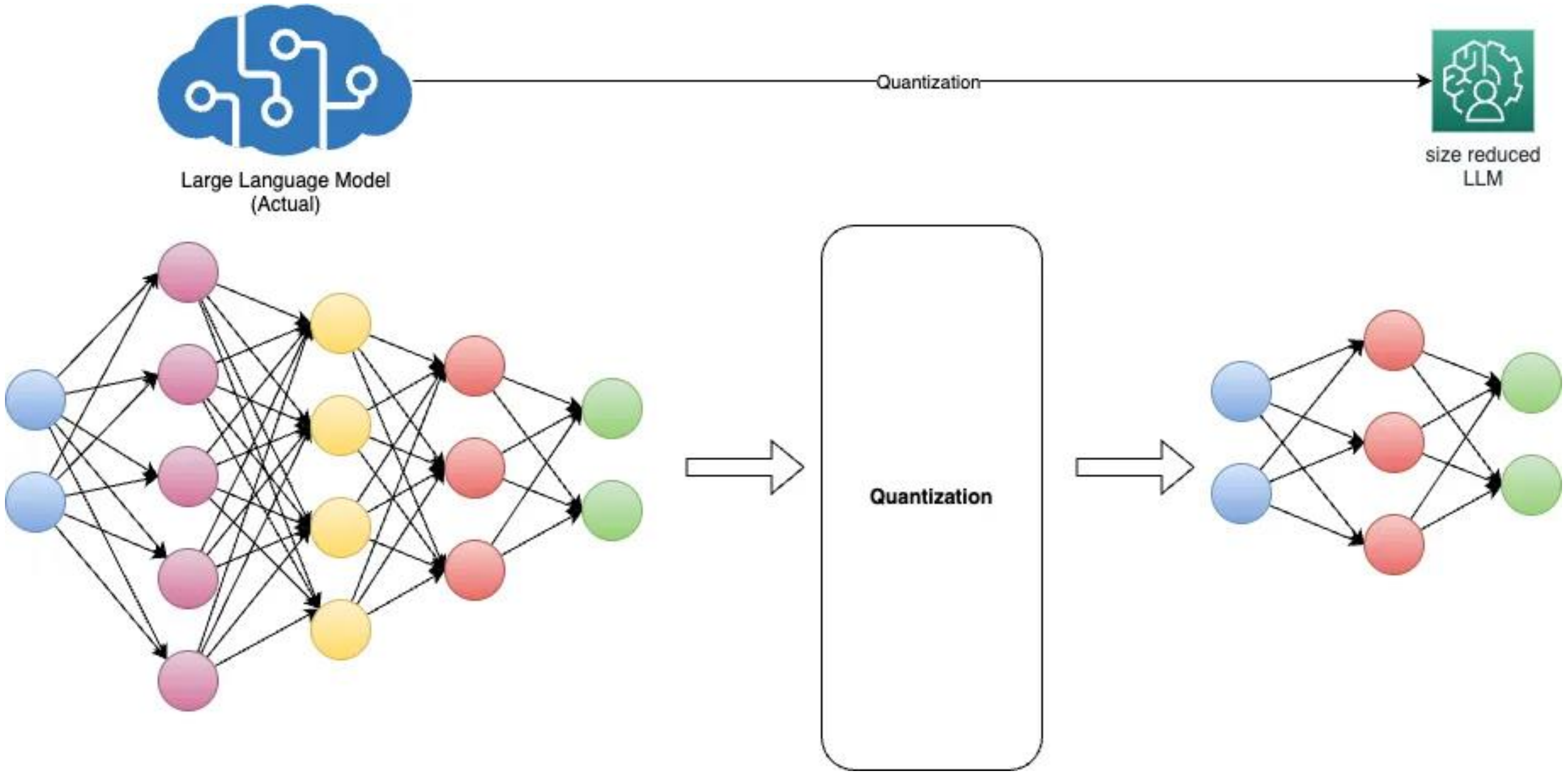
Reducing Climate Impact

- These models take a lot of electricity to train and run inference (make responses)
- This can have costly environmental impacts
- Concerns for both the amount of CO_2 generated and the amount of water required for cooling data centers.



Reducing Climate Impact

Can we achieve similar results with smaller models?

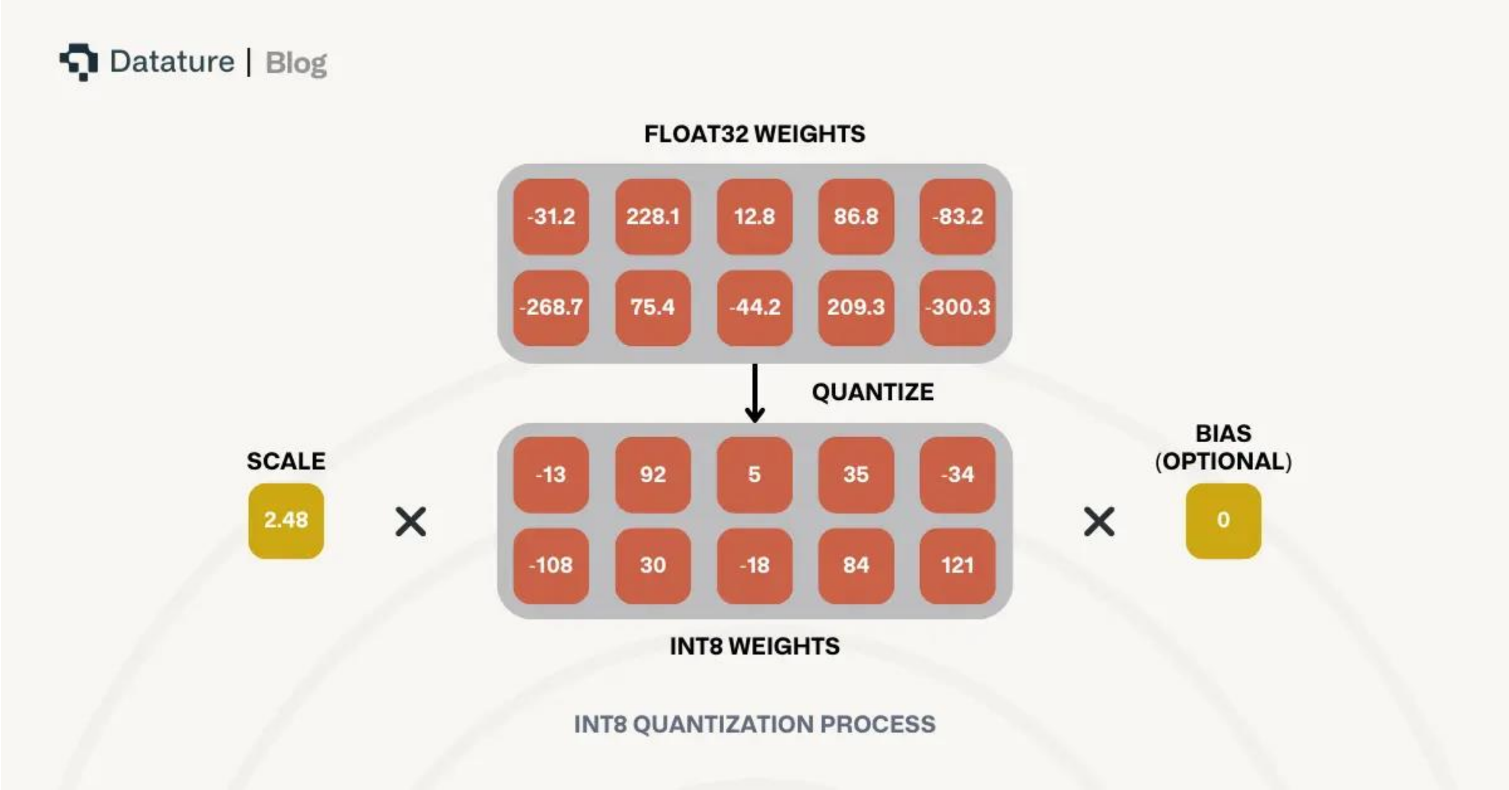


Quantization

Can we use smaller representation of parameters?

DeepSeek was able to create distilled and quantized models that only used 4 bits per parameter

https://huggingface.co/neuralmagic/DeepSeek-R1-Distill-Llama-8B-quantized.w4a16

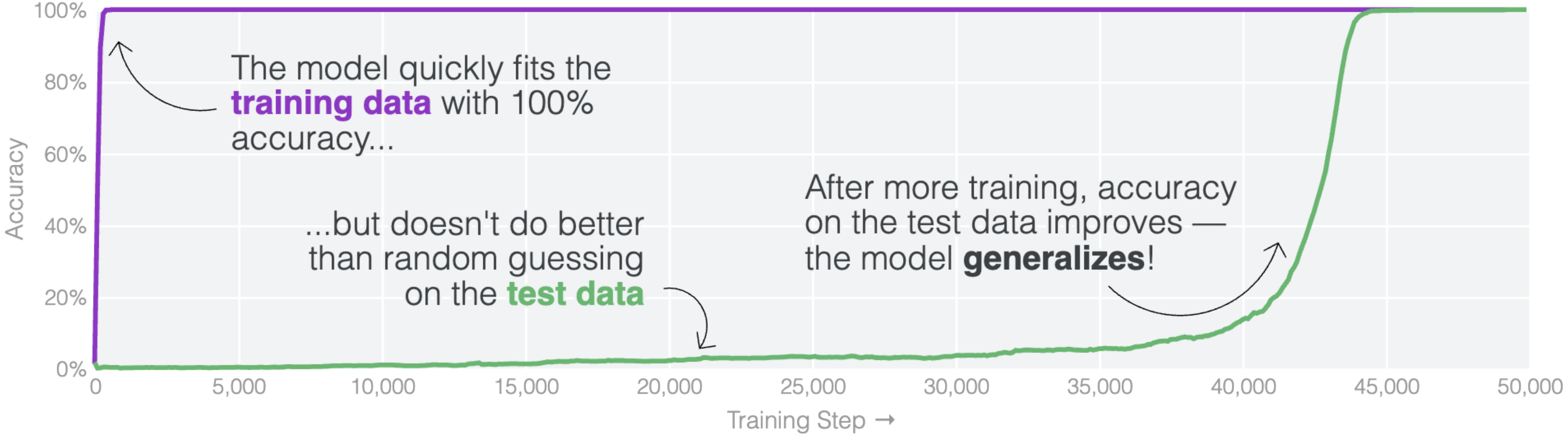


Memorization or Generalization?

Do LLMs "just memorize the training data"?

Grokking: The network suddenly generalizes well after initially overfitting the training data

An Example Of Grokking: Memorization Followed By Sudden Generalization



https://pair.withgoogle.com/explorables/grokking/

Memorization or Generalization?

Do LLMs “just memorize the training data”?

Why this **really** matters:

- If a language model is memorizing its inputs, it should not fall under fair use
- If a language model uses its training data to train and generalize, it probably falls under fair use

Fair use: under certain circumstances, the use of copyrighted materials without permission is allowed

One key consideration: The use must be ***transformative***

Chain of Thought (CoT)

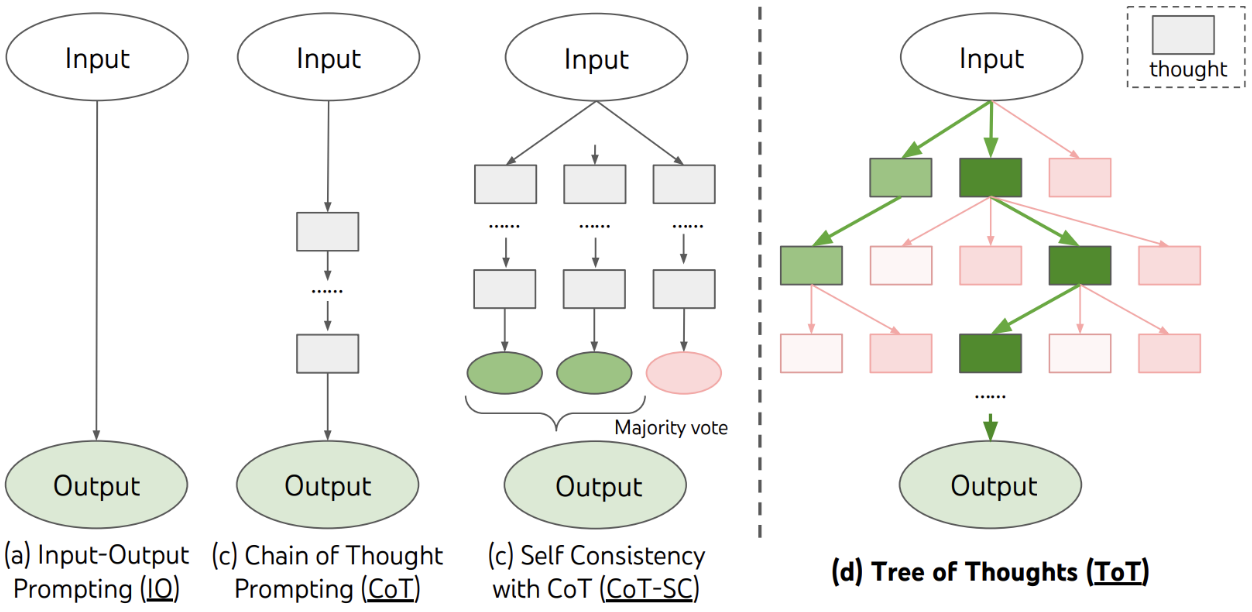


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a *thought*, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2,4,6.