edgerunner AI

Dr. Dylan
Acting Head of Research

Transformers and LoRA

What is a Transformer?

Transformers were first described in the seminal research paper by Google:

Attention Is All You Need

Initially described for language translation, they have become the basis for the majority of LLMs today (although other architectures exist!)

The key breakthrough was the implementation of the *attention* mechanism

Before we jump into the details, we need to understand *tokenizers*

Tokenizers

What are Tokens?

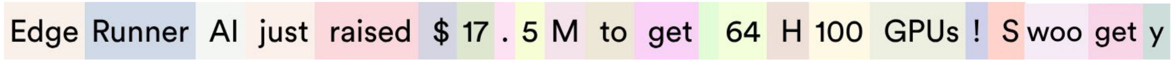
Despite often passing the Turing test, LLMs don’t actually know what words are. Instead, they have a complicated mathematical representation of sub-word units called *tokens*.

Tokens can be anything:

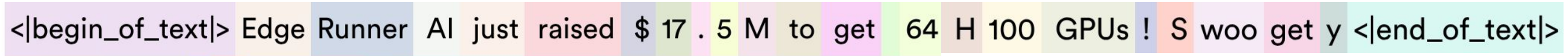
- I n d i v i d u a l [space] l e t t e r s [period]
- Entire words.
- Usually words, but sometimes fractions of long words like photos-ynthesis.
- xxbos xxmaj whacky ways of xxcap tokenizing grammar.xxeos

Tokenizers

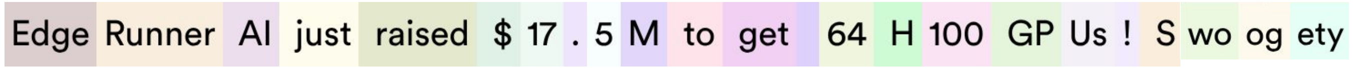
Different LLMs have different tokenizers that *must* be used. Switching them leads to nonsensical results.

- GPT4: 
Edge Runner AI just raised $ 17 . 5 M to get 64 H 100 GPUs ! S woo get y

```
[16577, 26032, 20837, 1327, 15478, 548, 1422, 13, 20, 44, 316, 717, 220, 2220, 487, 1353, 193432, 0, 336, 1338, 479, 6494]
```

- Llama: 
<|begin_of_text|> Edge Runner AI just raised $ 17 . 5 M to get 64 H 100 GPUs ! S woo get y <|end_of_text|>

```
[128000, 11918, 20051, 15592, 1120, 9408, 400, 1114, 13, 20, 44, 311, 636, 220, 1227, 473, 1041, 71503, 0, 328, 49874, 456, 88, 128001]
```

- DeepSeek: 
Edge Runner AI just raised $ 17 . 5 M to get 64 H 100 GP Us ! S wo og ety

```
[0, 35296, 67737, 7703, 1438, 9927, 957, 1002, 16, 23, 47, 304, 1178, 223, 2892, 437, 1457, 21845, 8095, 3, 327, 1015, 520, 1925]
```

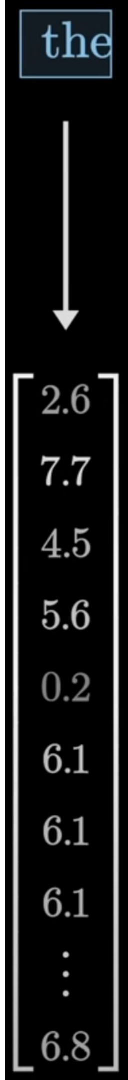
Although tokens are *usually* similar, the mapping from tokens to numbers differs.

Embeddings

Depending on the tokenizer, the model can understand between 32,000 and 200k+ different tokens (multilingual models, generally).

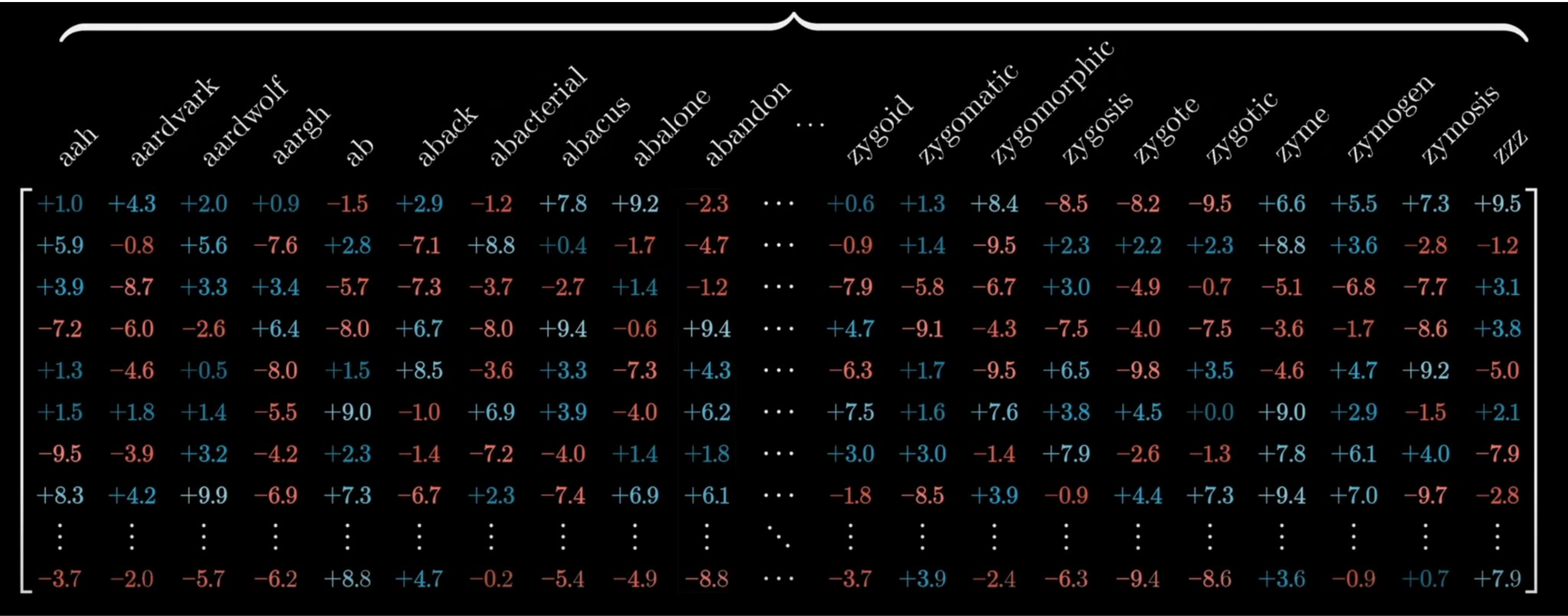
This includes “special tokens” like `<|begin_of_text|><|start_header_id|>system<|end_header_id|>` and wrappers around thinking, images, and tool calls, depending on the model.

Since computers don’t know what a word is, each of these tokens is represented by a (very) high dimensional vector, called an *embedding*.



Embeddings

All the tokens in a model’s vocabulary exist in an embedding matrix.



$$\begin{array}{ccccccccccccccccccccc}
\text{aah} & \text{aardvark} & \text{aardwolf} & \text{aargh} & \text{ab} & \text{aback} & \text{abacterial} & \text{abacus} & \text{abalone} & \text{abandon} & \cdots & \text{zygoid} & \text{zygomatic} & \text{zygomorphic} & \text{zygosis} & \text{zygote} & \text{zygotic} & \text{zyme} & \text{zymogen} & \text{zymosis} & \text{zzz} \\
+1.0 & +4.3 & +2.0 & +0.9 & -1.5 & +2.9 & -1.2 & +7.8 & +9.2 & -2.3 & \cdots & +0.6 & +1.3 & +8.4 & -8.5 & -8.2 & -9.5 & +6.6 & +5.5 & +7.3 & +9.5 \\
+5.9 & -0.8 & +5.6 & -7.6 & +2.8 & -7.1 & +8.8 & +0.4 & -1.7 & -4.7 & \cdots & -0.9 & +1.4 & -9.5 & +2.3 & +2.2 & +2.3 & +8.8 & +3.6 & -2.8 & -1.2 \\
+3.9 & -8.7 & +3.3 & +3.4 & -5.7 & -7.3 & -3.7 & -2.7 & +1.4 & -1.2 & \cdots & -7.9 & -5.8 & -6.7 & +3.0 & -4.9 & -0.7 & -5.1 & -6.8 & -7.7 & +3.1 \\
-7.2 & -6.0 & -2.6 & +6.4 & -8.0 & +6.7 & -8.0 & +9.4 & -0.6 & +9.4 & \cdots & +4.7 & -9.1 & -4.3 & -7.5 & -4.0 & -7.5 & -3.6 & -1.7 & -8.6 & +3.8 \\
+1.3 & -4.6 & +0.5 & -8.0 & +1.5 & +8.5 & -3.6 & +3.3 & -7.3 & +4.3 & \cdots & -6.3 & +1.7 & -9.5 & +6.5 & -9.8 & +3.5 & -4.6 & +4.7 & +9.2 & -5.0 \\
+1.5 & +1.8 & +1.4 & -5.5 & +9.0 & -1.0 & +6.9 & +3.9 & -4.0 & +6.2 & \cdots & +7.5 & +1.6 & +7.6 & +3.8 & +4.5 & +0.0 & +9.0 & +2.9 & -1.5 & +2.1 \\
-9.5 & -3.9 & +3.2 & -4.2 & +2.3 & -1.4 & -7.2 & -4.0 & +1.4 & +1.8 & \cdots & +3.0 & +3.0 & -1.4 & +7.9 & -2.6 & -1.3 & +7.8 & +6.1 & +4.0 & -7.9 \\
+8.3 & +4.2 & +9.9 & -6.9 & +7.3 & -6.7 & +2.3 & -7.4 & +6.9 & +6.1 & \cdots & -1.8 & -8.5 & +3.9 & -0.9 & +4.4 & +7.3 & +9.4 & +7.0 & -9.7 & -2.8 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
-3.7 & -2.0 & -5.7 & -6.2 & +8.8 & +4.7 & -0.2 & -5.4 & -4.9 & -8.8 & \cdots & -3.7 & +3.9 & -2.4 & -6.3 & -9.4 & -8.6 & +3.6 & -0.9 & +0.7 & +7.9
\end{array}$$

Model Sizes - GPT1 and GPT2

Parameters	Layers	d_{model}	n_{heads}	$n_{neurons}$
GPT1 - 117M	12	768	12	3072
GPT2 - 117M	12	768	12	3072
GPT2 - 345M	24	1024	16	4096
GPT2 - 762M	36	1280	20	5120
GPT2 - 1542M	48	1600	25	6400

Model Sizes - GPT3

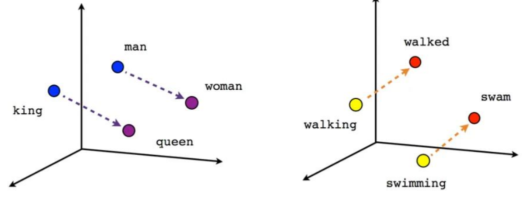
Parameters	Layers	d_{model}	n_{heads}	d_{head}
125M	12	768	12	64
350M	24	1024	16	64
760M	24	1536	16	96
1.3B	24	2048	24	128
2.7B	32	2560	32	80
6.7B	32	4096	32	128
13B	40	5140	40	128
175B	96	12288	96	128

Model Sizes - Llama 3

Parameters	Layers	d_{model}	n_{heads}	$n_{neurons}$*
8B	32	4096	32	14336
70B	80	8192	64	28672
405B	126	16384	128	53248

* in the GPT models, $n_{neurons} = 4 * d_{model}$. This is somewhat close to that.

Embeddings

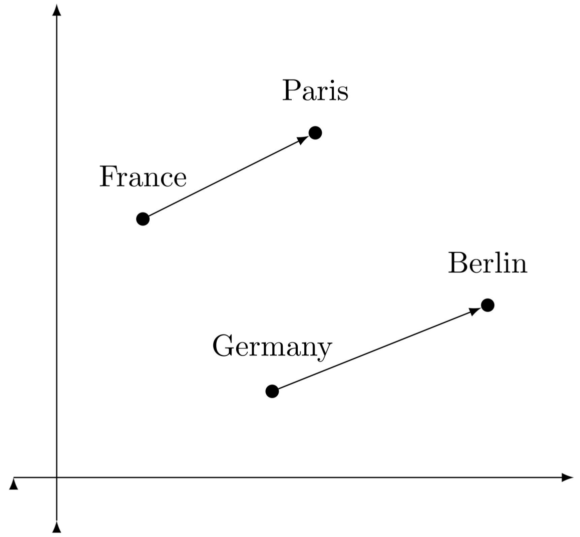


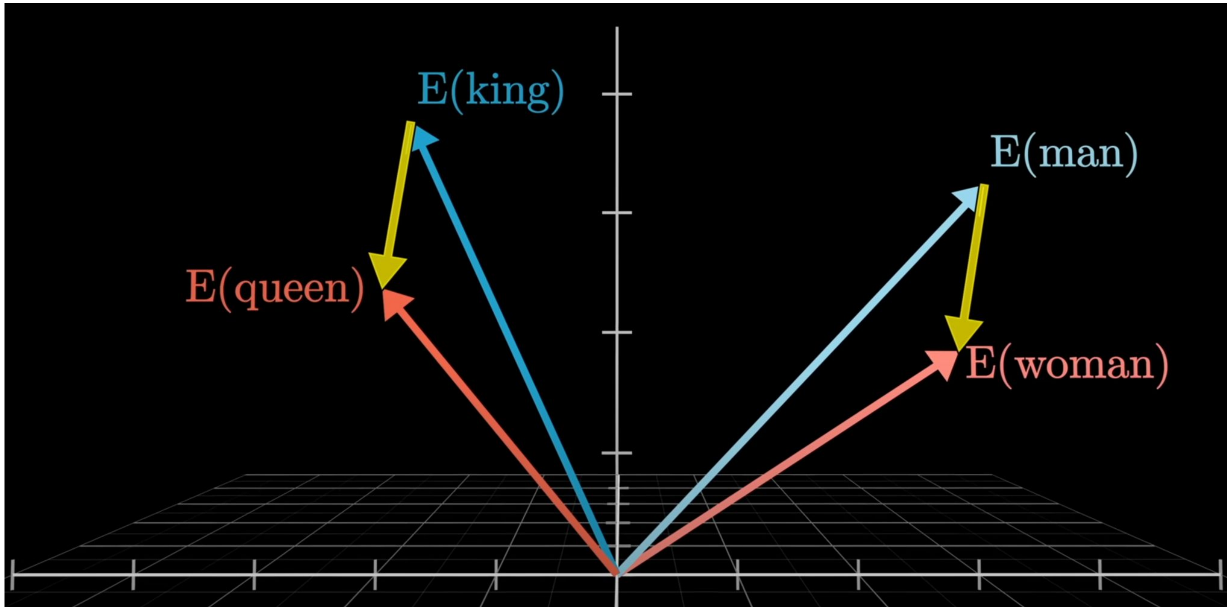
Male-Female

Verb tense

Directions in the high-dimensional embedding space can carry semantic meaning

There may be a unique direction representing gender, size, or “capital-ness”

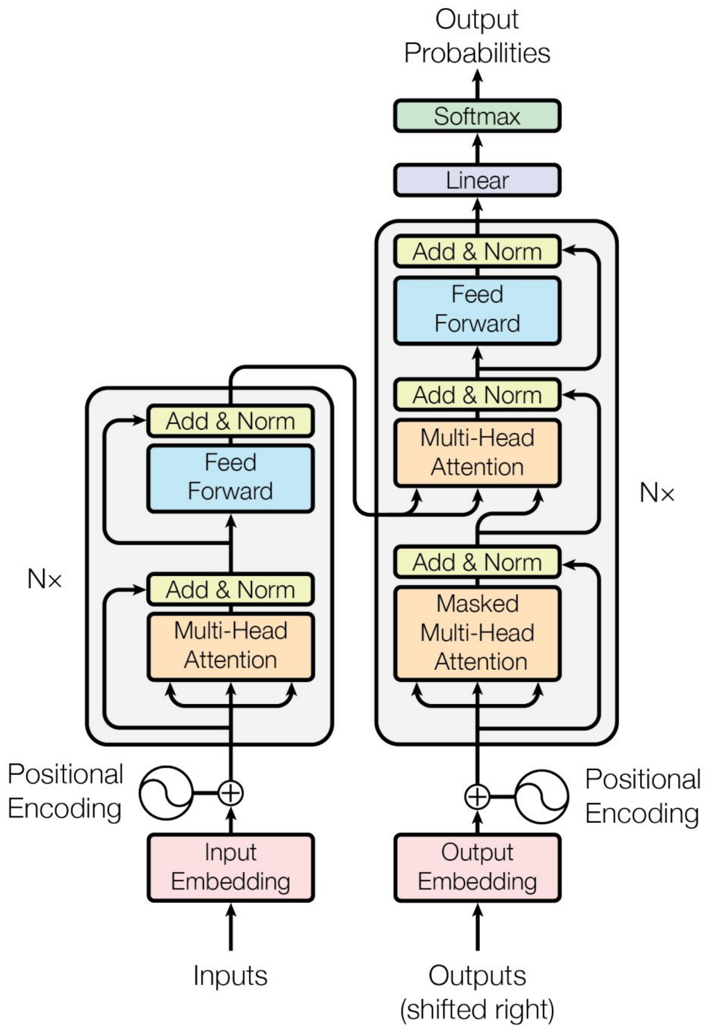




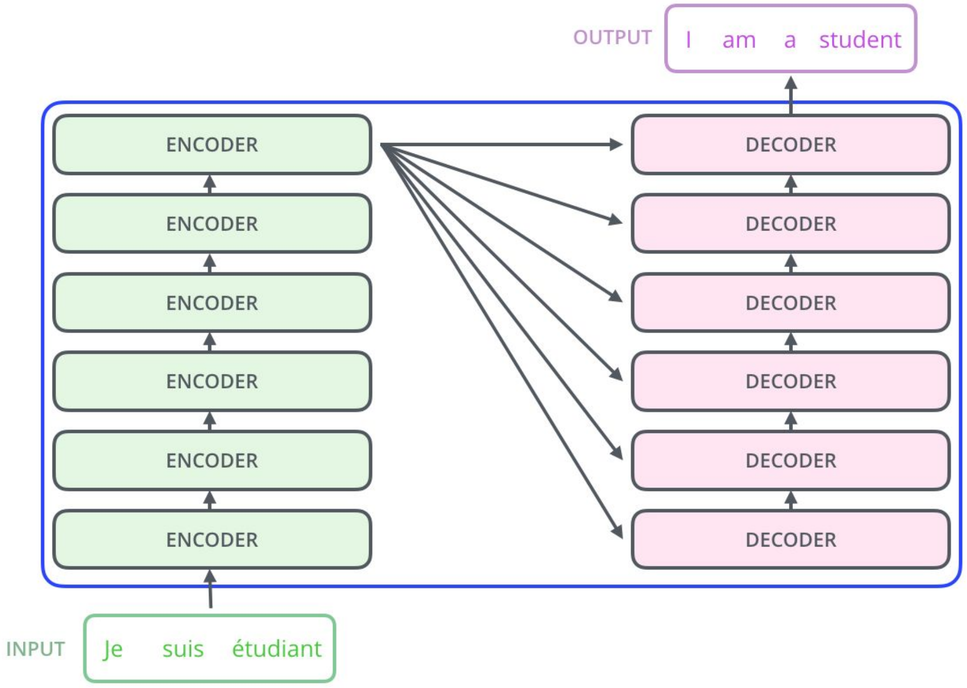
Transformers

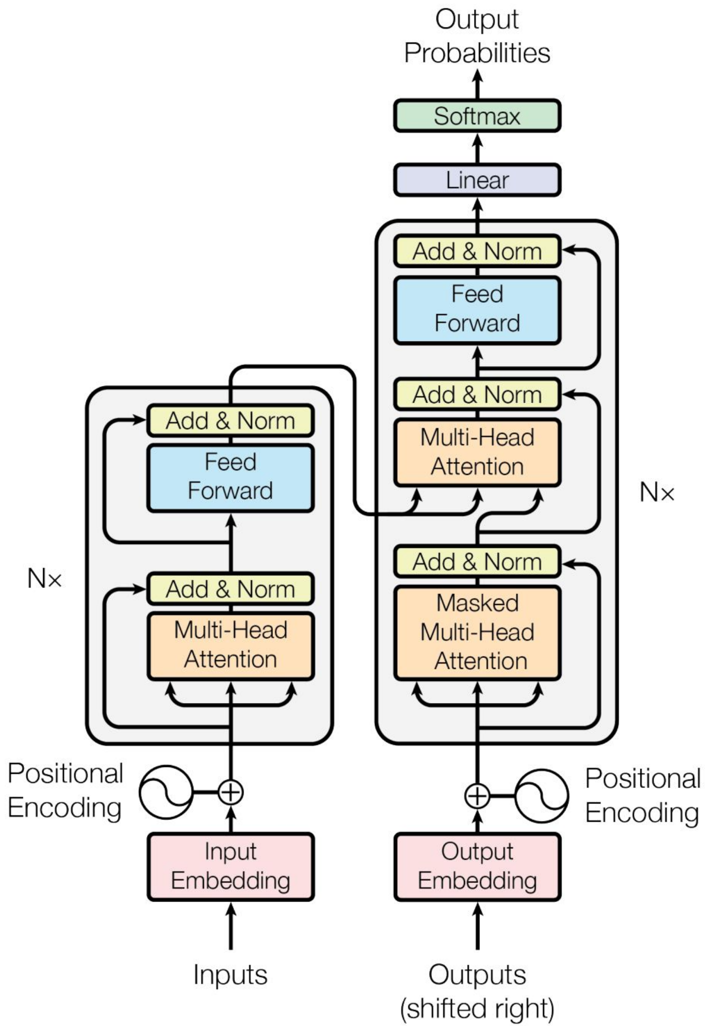
So then: What is a Transformer?

A Transformer is a deep learning architecture making use of the multi-head *attention* mechanism.

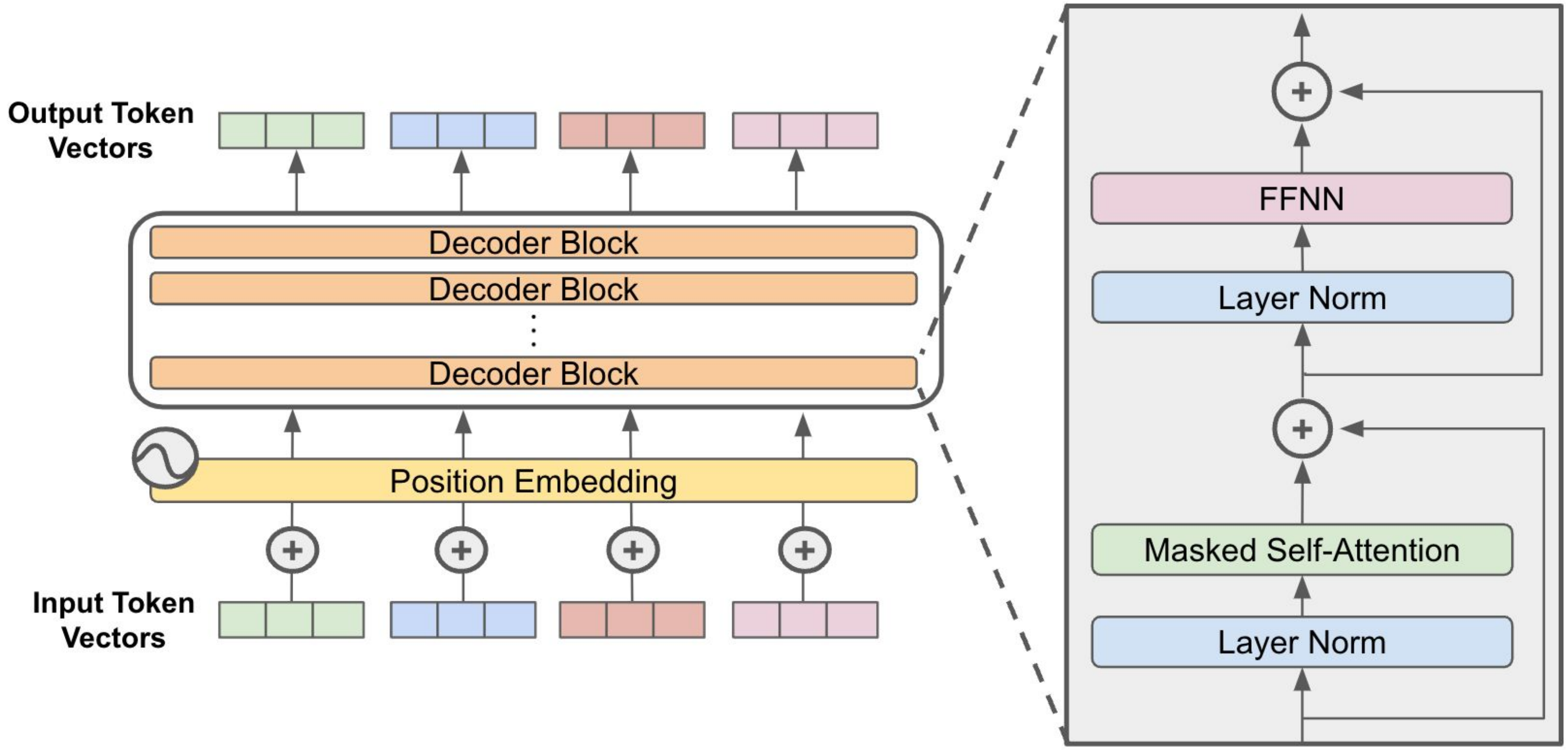


Transformer Architecture





Decoder-Only Architecture (GPTs)



Attention

What is Attention?

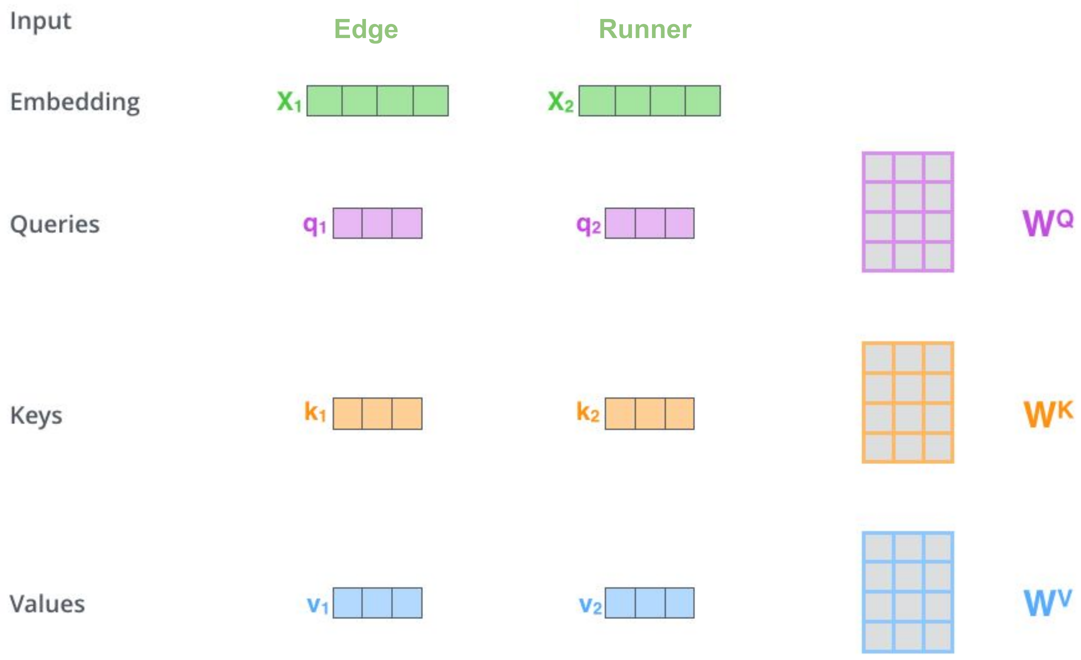
$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q = Query

K = Key

V = Value

d_k = Length of each vector *Q*, *K*, *V*



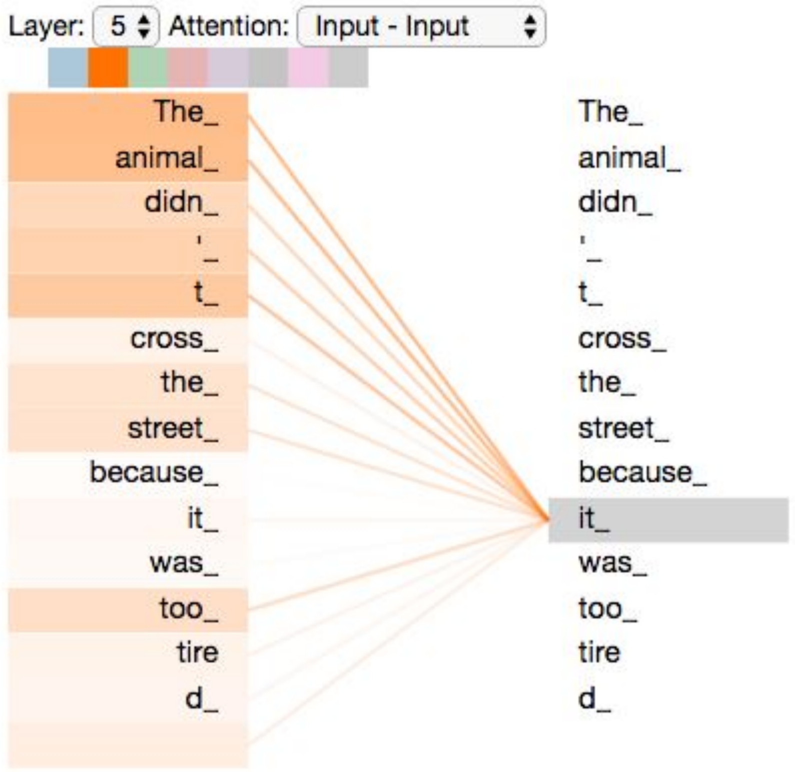
Attention: A Breakdown

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

QK^T: This is just the dot product of the Query and Key matrices, measuring how similar they are.

Query: What information do you have?
Key: I have this information!

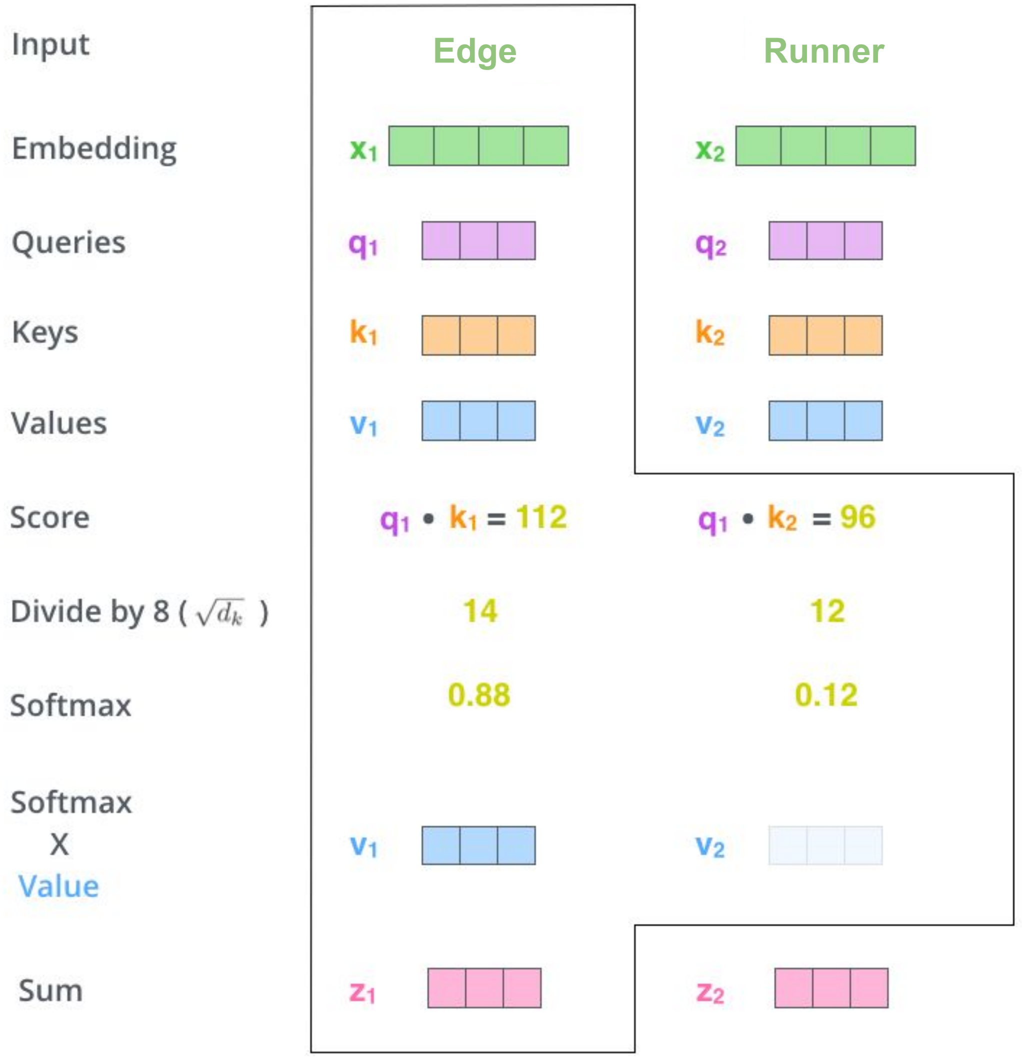
This is large when the Query and Key values are similar.



Attention

$$\mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

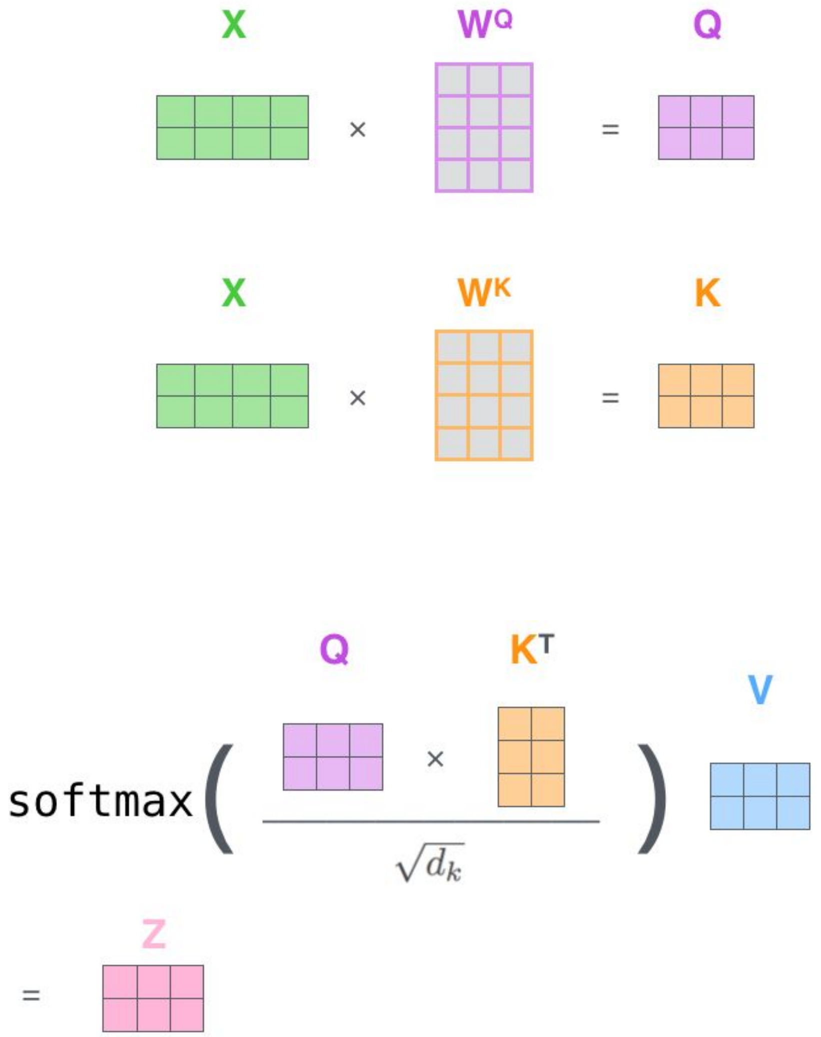
In practice, this is done many tokens at a time by multiplying matrices instead of vectors.



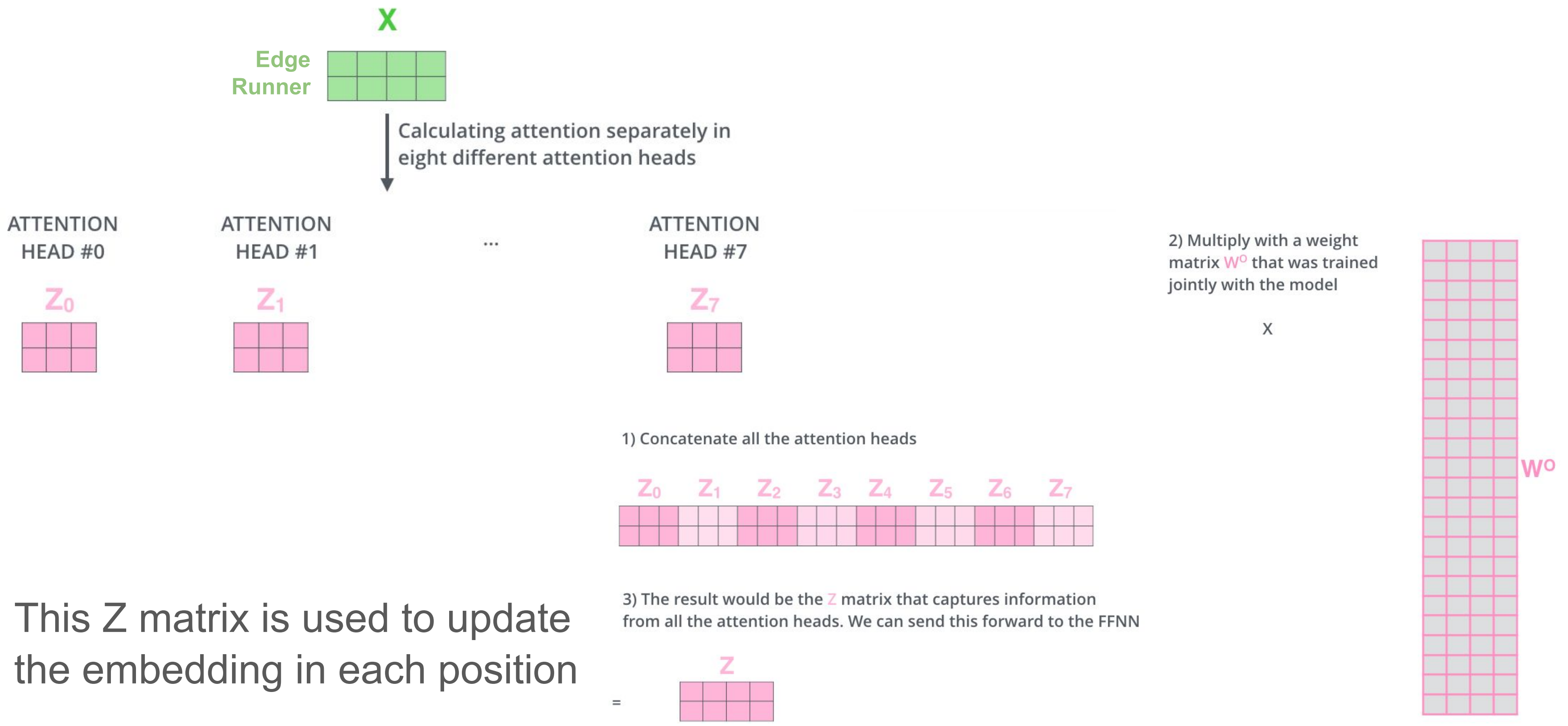
Attention

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

In practice, this is done several tokens at a time by multiplying matrices instead of vectors.



Multi-Headed Attention


X
Edge Runner
Calculating attention separately in eight different attention heads
ATTENTION HEAD #0
Z_0
ATTENTION HEAD #1
Z_1
...
ATTENTION HEAD #7
Z_7
1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z

This Z matrix is used to update the embedding in each position

Putting it all together

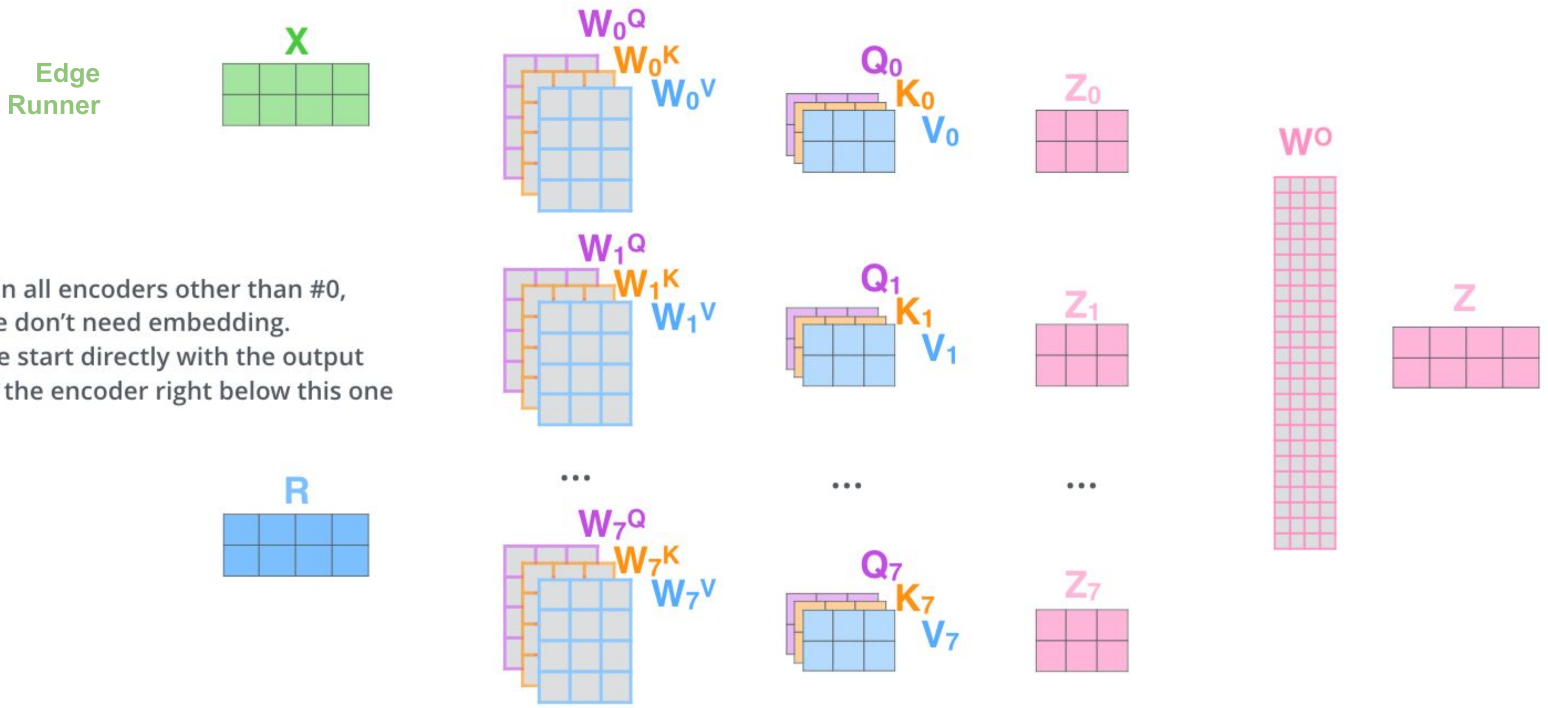
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

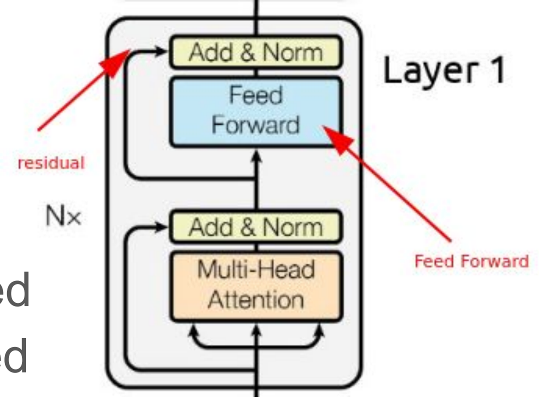
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

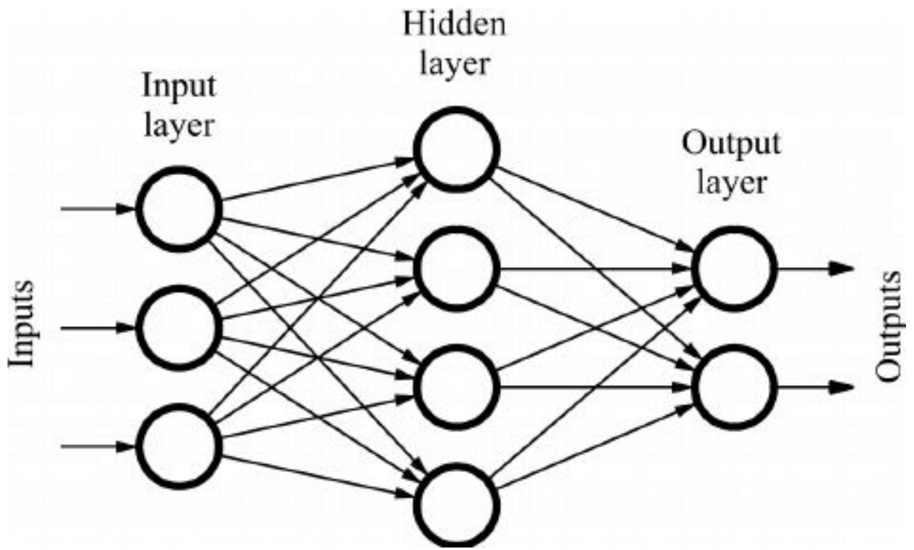


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Linear Layers



After going through the Attention block, the outputs are added to the original embeddings, which are normalized and passed through a standard one-hidden-layer feedforward neural network.



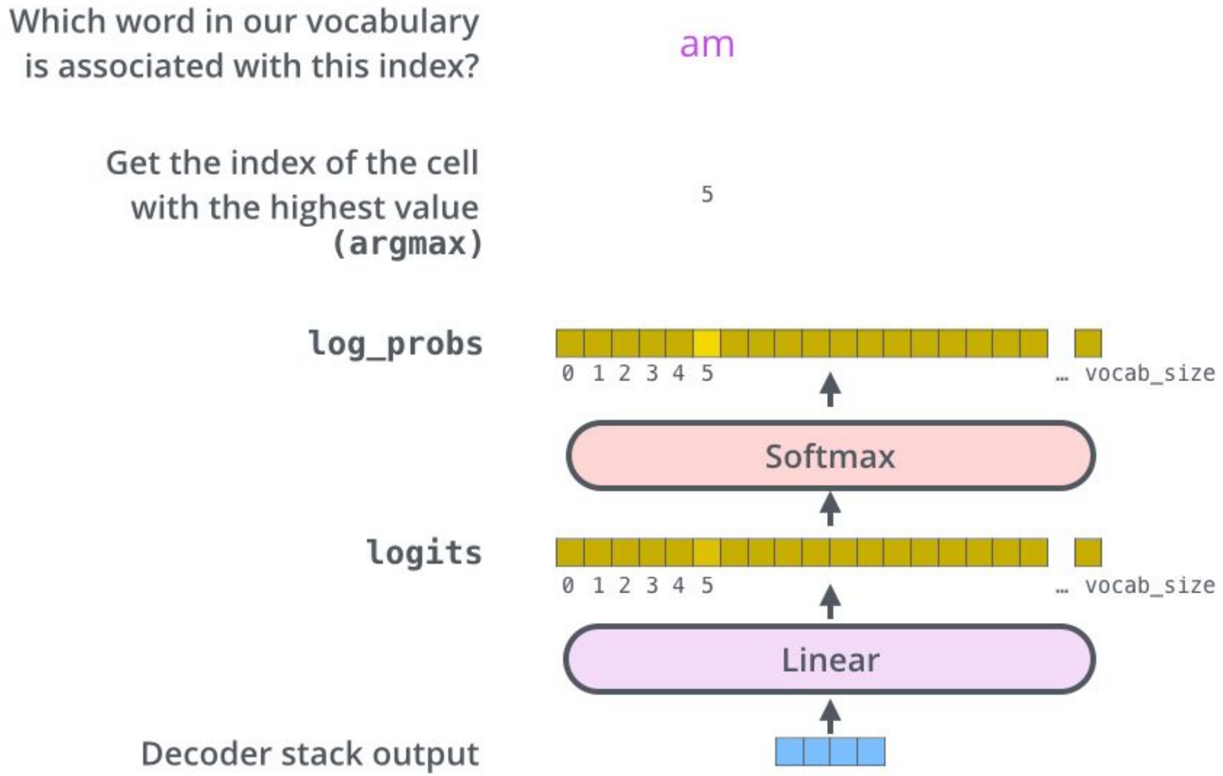
These layers take up ⅔ of the parameters in LLMs, and is where the “facts” in the network are stored.

This process is repeated n_{layers} times.

What's the output?

After a final linear layer, the embeddings are passed through an "unembedding" matrix.

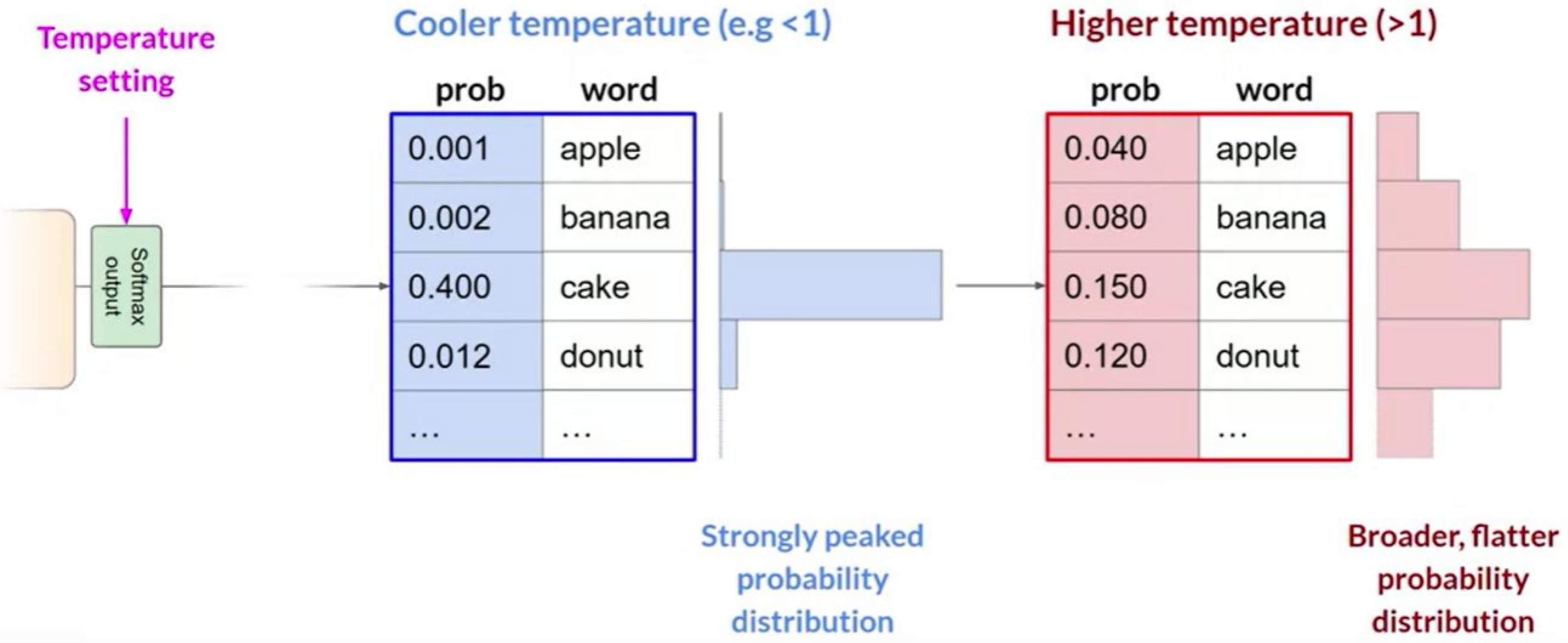
This produces a probability distribution of most likely next tokens.


Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5 … vocab_size
Softmax
logits
0 1 2 3 4 5 … vocab_size
Linear
Decoder stack output

We sample a token from the distribution, append it to the text, and repeat the whole process with the new (slightly longer) input.

Hallucinations

They're a fundamental part the way the next token is selected:


Temperature setting
Softmax output
Cooler temperature (e.g <1)
Higher temperature (>1)
Strongly peaked probability distribution
Broader, flatter probability distribution

Cooler temperature (e.g <1)

prob	word
0.001	apple
0.002	banana
0.400	cake
0.012	donut
...	...

Higher temperature (>1)

prob	word
0.040	apple
0.080	banana
0.150	cake
0.120	donut
...	...

Further Optimizations

Reducing the size of the attention scores to reduce memory/compute and increase sequence length: sparse attention, ring attention, blockwise attention

Linformers, Structured State Space Sequence models, Mamba/Jamba

KV-caching - storing previously computed key and value matrices

Mixture of Experts (MoE) - fewer active parameters at a time

Multi-Token Prediction (MTP) - what it says on the tin

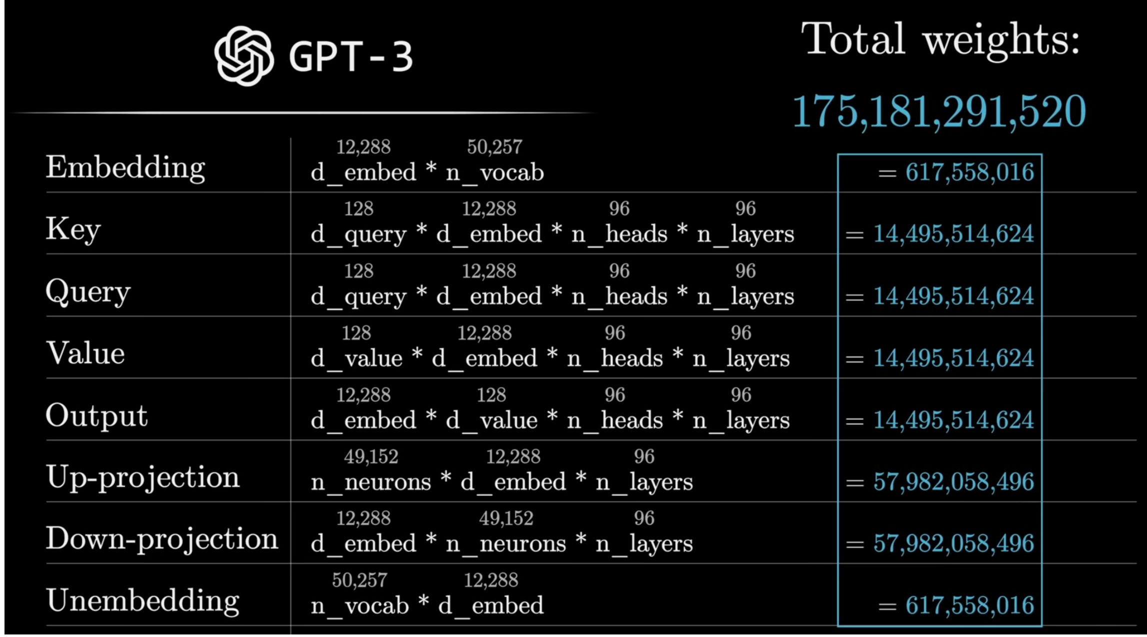
Multi-head Latent Attention (MLA) - low-dimensional representation of attention

Rotary Position Embeddings (RoPE) - enables longer sequences

Training a Transformer

Unsurprisingly, all of the weights in a LLM are stored as matrices.

During backpropagation, we update the individual weights so the outputs better match the actual next word in our corpus.



GPT-3

Total weights: 175,181,291,520

Embedding	$\overset{12,288}{\text{d\_embed}} * \overset{50,257}{\text{n\_vocab}}$	= 617,558,016
Key	$\overset{128}{\text{d\_query}} * \overset{12,288}{\text{d\_embed}} * \overset{96}{\text{n\_heads}} * \overset{96}{\text{n\_layers}}$	= 14,495,514,624
Query	$\overset{128}{\text{d\_query}} * \overset{12,288}{\text{d\_embed}} * \overset{96}{\text{n\_heads}} * \overset{96}{\text{n\_layers}}$	= 14,495,514,624
Value	$\overset{128}{\text{d\_value}} * \overset{12,288}{\text{d\_embed}} * \overset{96}{\text{n\_heads}} * \overset{96}{\text{n\_layers}}$	= 14,495,514,624
Output	$\overset{12,288}{\text{d\_embed}} * \overset{128}{\text{d\_value}} * \overset{96}{\text{n\_heads}} * \overset{96}{\text{n\_layers}}$	= 14,495,514,624
Up-projection	$\overset{49,152}{\text{n\_neurons}} * \overset{12,288}{\text{d\_embed}} * \overset{96}{\text{n\_layers}}$	= 57,982,058,496
Down-projection	$\overset{12,288}{\text{d\_embed}} * \overset{49,152}{\text{n\_neurons}} * \overset{96}{\text{n\_layers}}$	= 57,982,058,496
Unembedding	$\overset{50,257}{\text{n\_vocab}} * \overset{12,288}{\text{d\_embed}}$	= 617,558,016

However, updating (hundreds of) billions of parameters can be expensive and time-consuming - not to mention requiring a shitload of data to do a good job.

LoRA

Low-Rank Adaptation (LoRA)

- Fine-tune models to be better at a specific domain or task
- Can train models with less VRAM, less time, and less data
- Popular for Language and Diffusion models
- Easier to transmit and download
- Can store and swap LoRAs

Because full fine-tunes are so expensive, I’ve only ever made (and merged) LoRAs

A little linear algebra

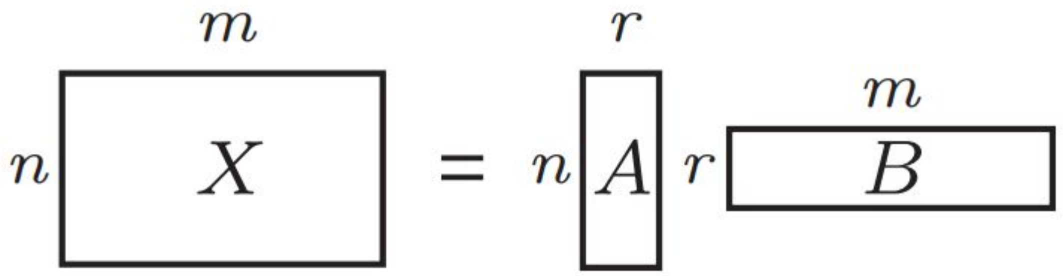
The *rank* of a matrix is the number of columns (or rows) that are linearly independent

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

What is the rank of this matrix?

If the rank of a matrix is less than min(*m*,*n*) it is *low-rank*

We can decompose a large matrix into two matrices with lower rank.



The idea of Low-Rank Adaptation (LoRA)

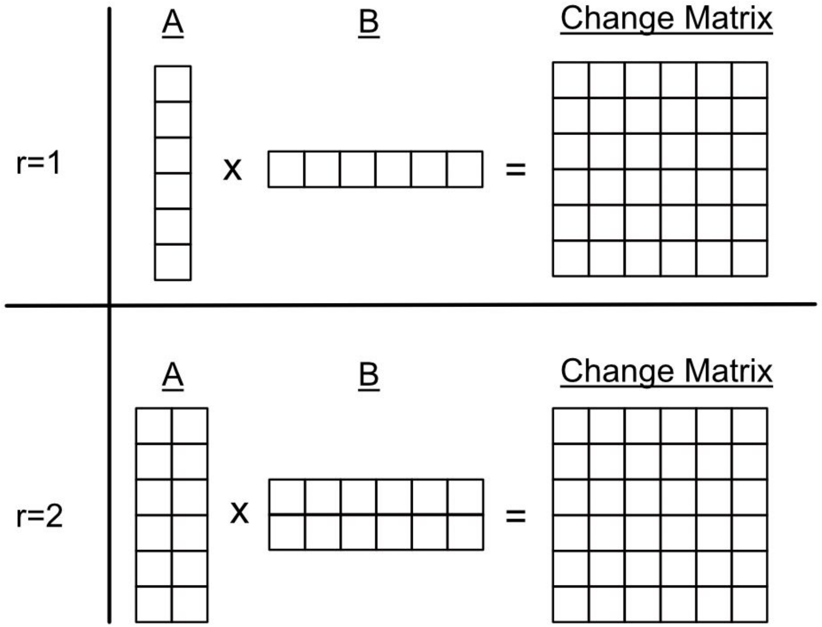
Instead of updating all 49,152 × 12,288 ≈ 604 million parameters in one of the linear layers, we can instead update as few as 49,152 × 1 + 1 × 12,288 = 61,440

$$m \text{ x } n \approx (m \times r) \times (r \times n)$$

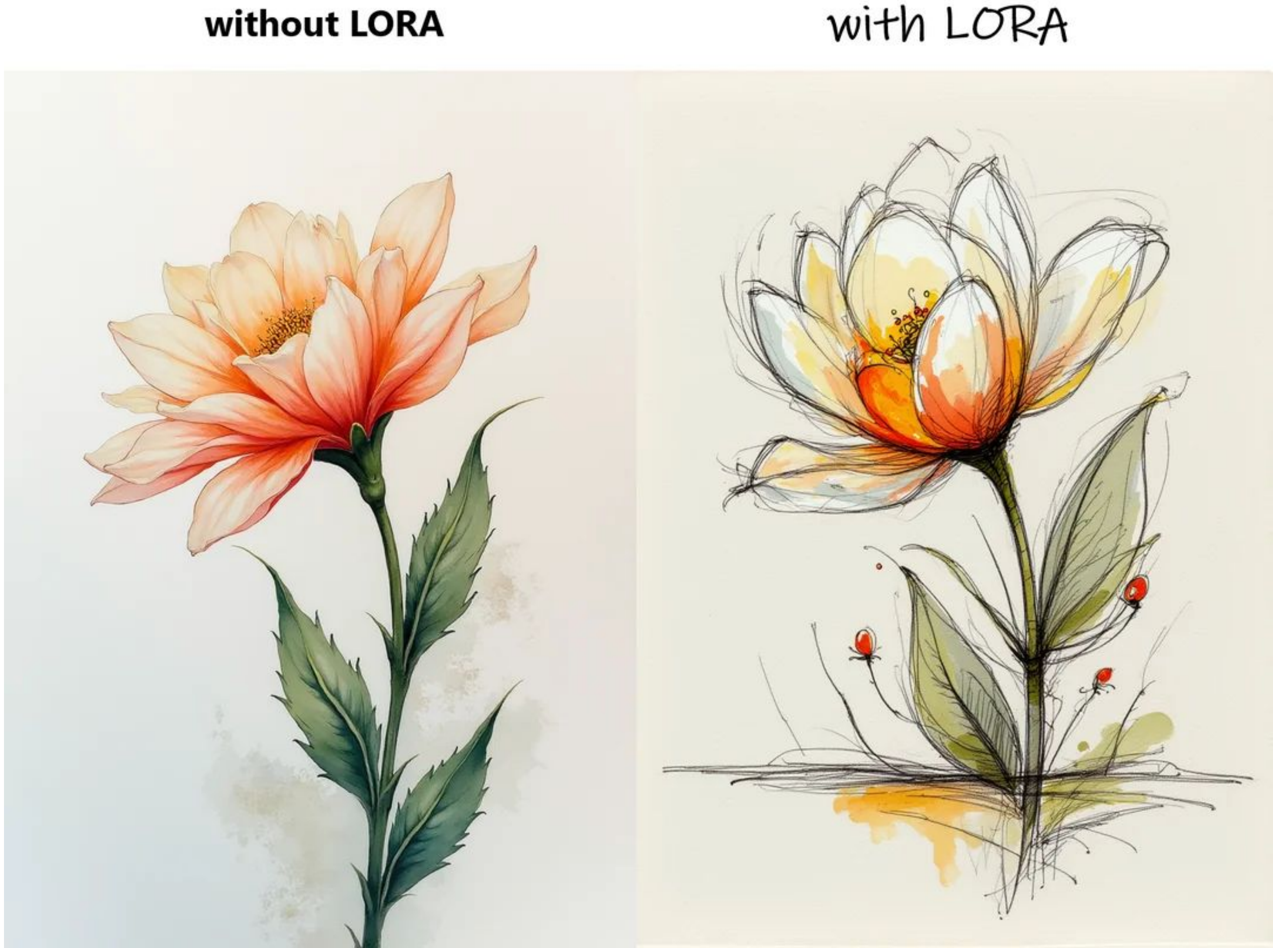
$$r << (m,n)$$

$$W_0 + \Delta W = W_0 + AB$$

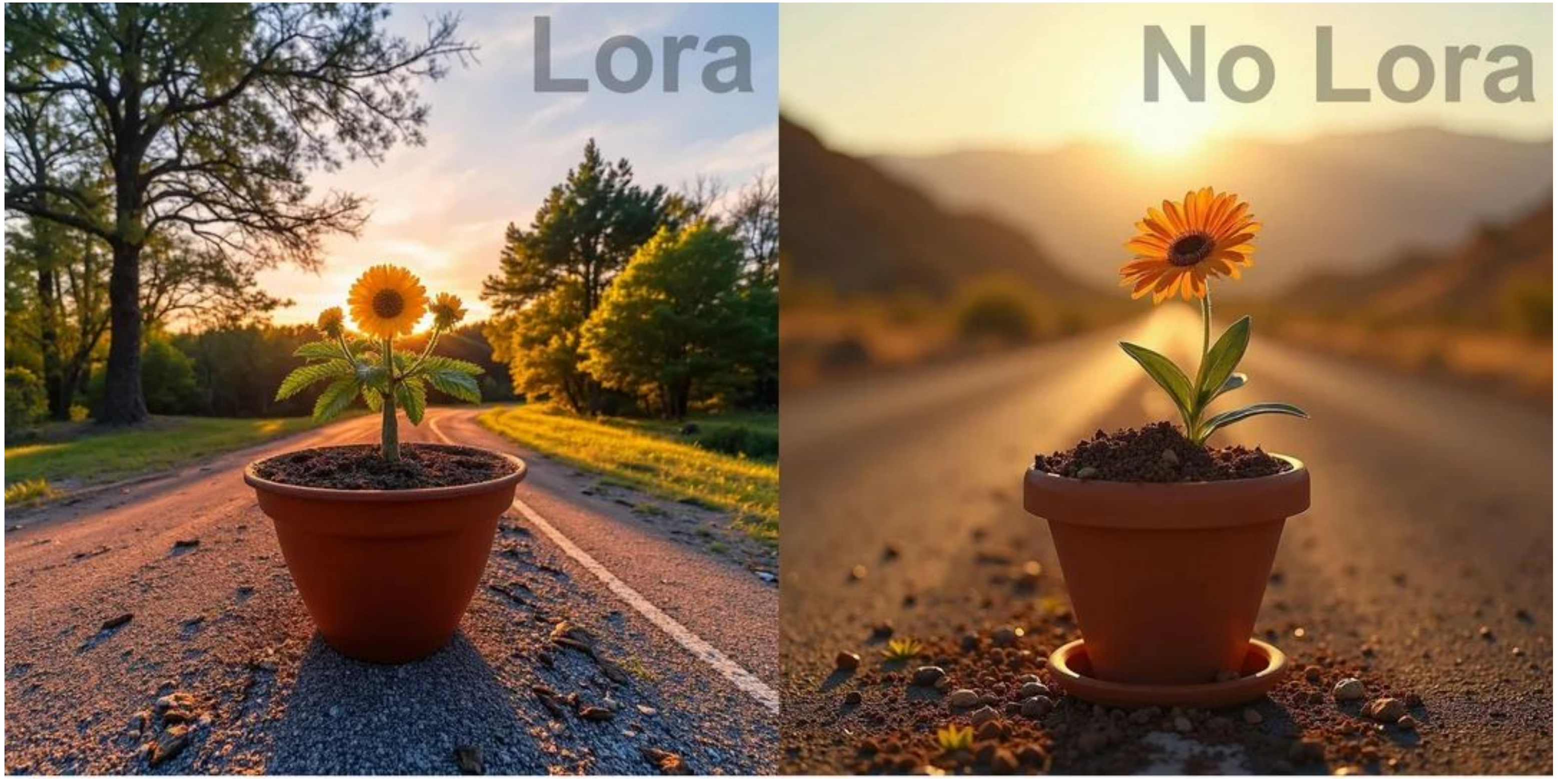
Usually, we set $r = 16$ up to $r = 256$.



Examples of LoRA

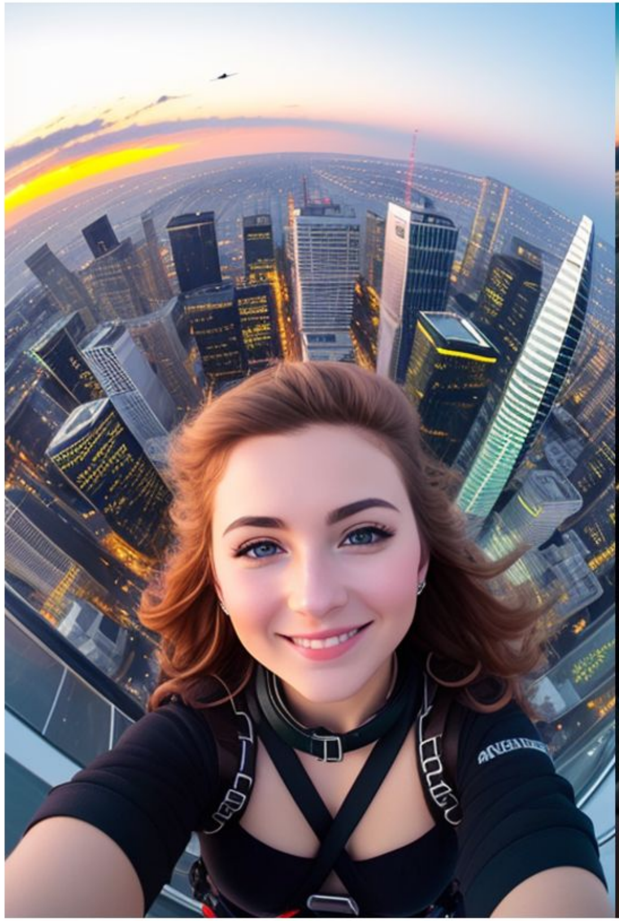


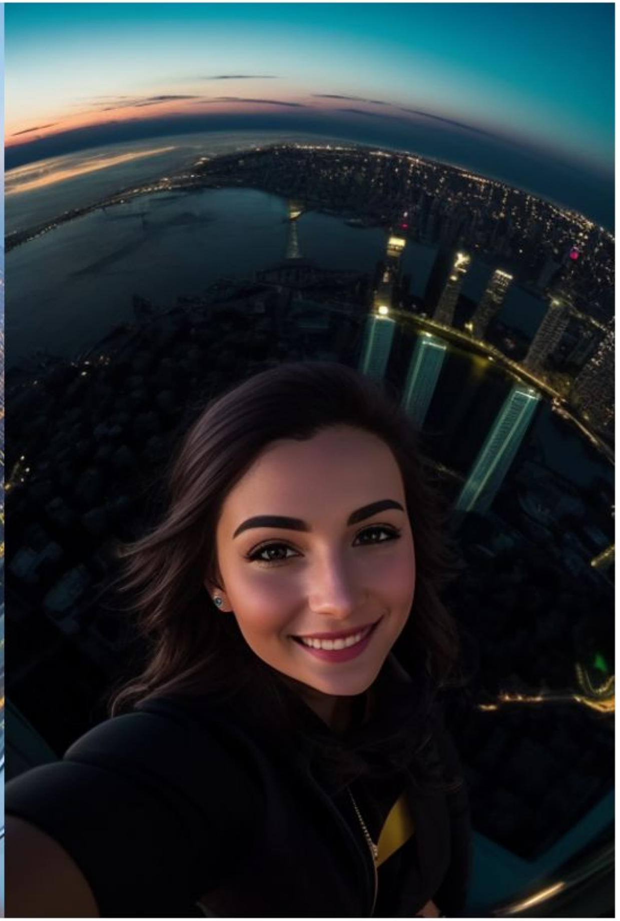
Examples of LoRA



clay pot full of dirt with a beautiful daisie planted in it, shining in the autumn sun on an abandoned, wallpaper, no blur

Examples of LoRA


Without LoRA


With LoRA

LLM Examples of LoRA

In the same vein, we can modify the outputs of language models by training them on a particular type of text.

For example, you could create a Shakespeare LoRA, or a Trump LoRA.

In practice, we’ve been building LoRAs for task-specific or role-specific purposes:

- Acquisitions agent
- Logistics officer
- Scriptwriting assistant

This post-training LoRA requires data with “instruction” “input” and “output”

Practical LoRA Training - How to do it?

There are several frameworks that allow you to train LoRAs:

- HuggingFace PEFT
- Unsloth
- Axolotl

Images only:

- Kohya-trainer
- Dreambooth

You can get away with just a .yaml file and data

Finetuning with Axolotl

I wrote a whole document on training with Axolotl.

You need:

- A dataset (local or HuggingFace)
- A `config.yaml` file like this
- At least one GPU

Then type:

`axolotl train my_config.yaml`

```yaml
base_model: meta-llama/Llama-3.3-70B-Instruct
model_type: LlamaForCausalLM
tokenizer_type: AutoTokenizer

load_in_8bit: false
load_in_4bit: true # Note, we load the model in 4bits for training
strict: false

chat_template: llama3

datasets:
  - path: json
    type: alpaca_chat.load_qa
    ds_type: json
    data_files: Acquisitions_Data_Full.jsonl

dataset_prepared_path: last_run_prepared
val_set_size: 0.02
output_dir: ./outputs/70B_v1

sequence_len: 4096
sample_packing: true
eval_sample_packing: false # For multi-GPU setups, this has to be false
pad_to_sequence_len: true

# Same settings as last time
adapter: lora
lora_model_dir:
lora_r: 256
lora_alpha: 128
lora_dropout: 0.05
lora_target_linear: true
```

Axolotl Optimizations

When we were GPU poor (or when training 70B+ models… ever), you will OOM

- `load_in_8bit`: easy, recommended
- `load_in_4bit`: easy, not recommended unless you have to
- Reduce `sequence_len`: 2048 should be good enough for most models.
- LoRA Optimizations: Can use with 4bit, but not 8bit for some reason.
    - `lora_mlp_kernel: true`
    - `lora_qkv_kernel: true`
    - `lora_o_kernel: true`
- Single GPU only: unsloth optimizations
    - Requires pytorch < 2.6, which introduces other conflicts with the latest transformers, which is needed for training Mistral. Should be directly incorporated into axolotl soon.
- Decrease `lora_r` (and `lora_alpha` proportionally). I don't think this makes a huge difference.
- Reduce `micro_batch_size`. Increase `gradient_accumulation_steps` proportionally.
- Switch optimizer to `adamw_bnb_8bit`
- Enable `flash_attention` (you should be doing this anyway - not for Mistral)
- Increase DeepSpeed level (need to download new json file)

Quantized LoRA

By setting `load_in_4bit: True` and `adapter: qlora`, you can quantize a LLM down to 4-bits and freeze the weights, reducing VRAM usage by ~75%.

QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model using LoRA, getting nearly the same results as LoRA or a full fine-tune, but on considerably smaller (cheaper) hardware.

If you have the horsepower, I’d avoid it. But it’s very useful when you need it.

Weight-Decomposed LoRA

DoRA first decomposes the weight matrix W into magnitude/direction components:

$$W = m\frac{V}{||V||_c}$$

It then applies LoRA fine-tuning to the direction only:

$$W' = \underline{m}\frac{V + \Delta V}{||V + \Delta V||_c} = \underline{m}\frac{W_0 + \underline{BA}}{||W_0 + \underline{BA}||_c}$$

Although this extra math results in slightly more compute (longer training), empirical results show the output is similar (or better than) full fine-tunes.

You can use the flag `peft_use_dora: true` to activate DoRA training.

You can combine Quantized LoRA with Weight-Decomposed LoRA □ QDoRA.

Helpful References

Attention is All You Need, Vaswani et. al. (Original Transformers paper)

Language Models are Unsupervised Multitask Learners, Radford et. al. (GPT2)

Language Models are Few-Shot Learners, Brown et. al. (GPT3)

The Illustrated Transformer, Jay Alammar (Visuals from this presentation)

The Llama 3 Herd of Models, Llama Team

Neural Networks, 3Blue1Brown (specifically videos 5-8)

LoRA: Low-Rank Adaptation of Large Language Models, Hu et. al.

QLoRA: Efficient Finetuning of Quantized LLMs, Dettmers et. al.

DoRA: Weight-Decomposed Low-Rank Adaptation, Liu et. al.