Large Language Models (LLMs)

Aprendizaje Automático

Ingeniería de Robótica Software

Universidad Rey Juan Carlos

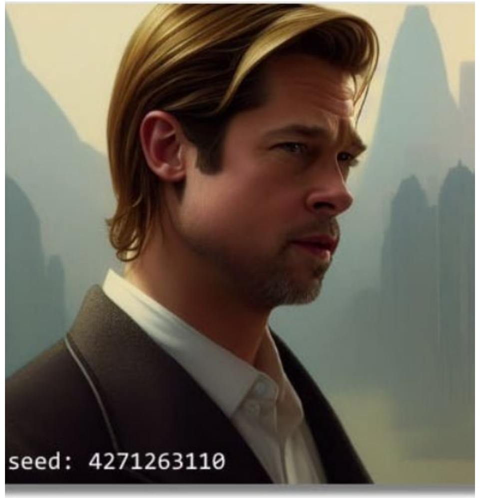
Outline

- Generative AI
- What is a Language Model?
- How large is an LLM?
- How to interact with LLMs
- LLM use cases
- How LLMs work
- Generating text with LLMs
- LLM architecture variations
- Prompting
- Fine-tuning
- Configuration parameters for inference

Generative AI

- **Generative Artificial Intelligence** (AI) is a broad term that can be used for any AI **system** whose primary function is to **generate content** like images, text, code, etc.
- Examples of generative AI systems are:
  - Image generators (such as Midjourney or Stable Diffusion).
  - Text generators (such as GPT-4, Claude, or Llama).
  - Code generators (such as Github Copilot).

Generative AI




```
$> gta brad pitt john
krasinski profile picture by
greg rutkowski, dynamic pose,
intricate, futuristic,
fantasy, elegant, by stanley
artgerm lau, greg rutkowski,
thomas kindkade, alphonse
mucha, loish, norman
rockwell, fantasy lut,
asymmetric, long hair, retro
computer graphics, video
game, fluid lines
```

ST

Please write 10 alternative headlines for this blog post: "10 trends from New York Fashion Week"

Sure, here are 10 alternative headlines for your blog post on New York Fashion Week:

1. "Spring Style Forecast: The Top Trends from Christian Dior's NYFW Line"
2. "The Best Looks from Christian Dior's Spring Line at New York Fashion Week"
3. "Fashion Must-Haves: 10 Pieces from Christian Dior's NYFW Collection"
4. "From the Runway to Your Closet: How to Wear Christian Dior's NYFW Trends"
5. "The Inspiration Behind Christian Dior's Spring Collection at NYFW"
6. "Christian Dior at NYFW: A Masterclass in Effortless Elegance"
7. "Sustainability and Style: Christian Dior's Ethical Fashion at NYFW"
8. "NYFW Recap: Christian Dior's Spring Line Dominates the Runway"
9. "The Future of Fashion: Trends from Christian Dior's NYFW Show"
10. "Breaking Down Christian Dior's NYFW Line: The Standout Pieces and Trends"

```
function manipulateTextBox() {
    const textBox = document.querySelector('input[type="text"]') as HTMLInputElement;
    textBox.value = '';
    textBox.select();
    textBox.focus();
}
```

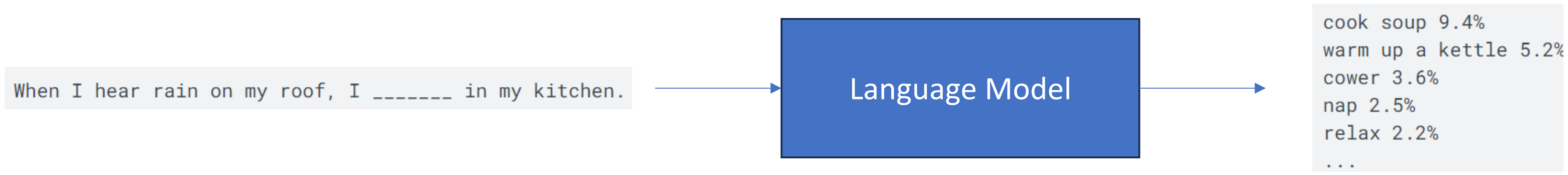
write a typescript function that finds the text box on the screen, removes the text, highlights the box, and places the cursor

Accept Discard

Changed 7 lines

What is a Language Model?

- A **Language Model** is a generative AI system based on a **machine learning** model that aims to **generate plausible language**.
- These models work by **estimating the probability of a token** or **sequence of tokens** occurring within a longer sequence of tokens.



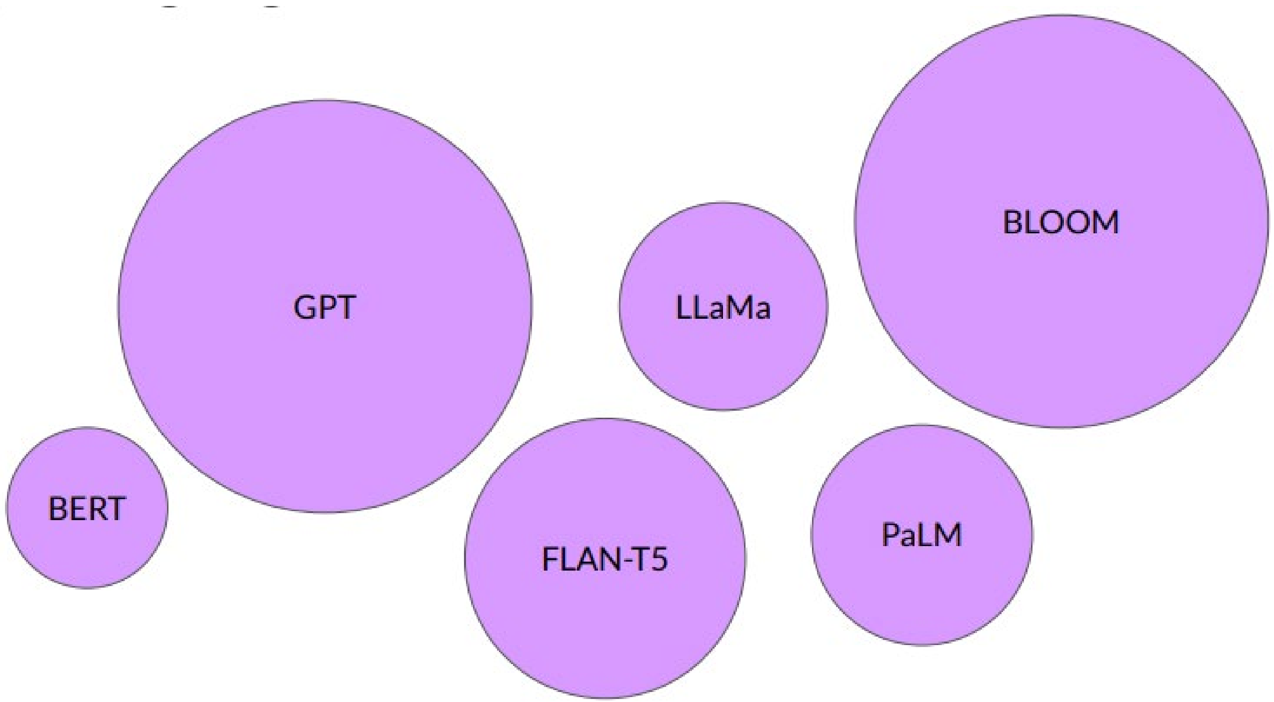
What is an LLM?

- **Early Language Models** were very **limited**, and, for example, they only could predict the probability of a single token.
- However, **modern Language Models** can predict the probability of large sequence of tokens like sentences, paragraphs, or even entire documents.
  - **Modern Language Models** are called "**Large Language Models**" (LLMs) and they refer to Language Models that use **deep learning** techniques with **a large number of parameters** (from millions to even trillions).
  - These models can **capture complex patterns in language** and **generate text** that is often indistinguishable from that written by humans.

How large is an LLM?

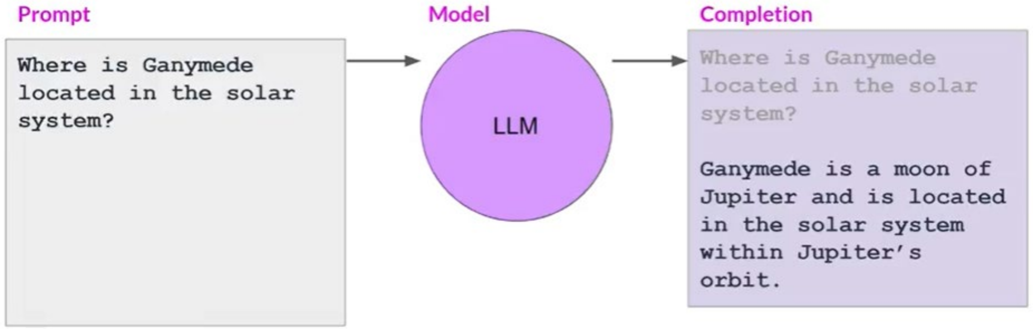
- "**Large**" in Language Models refers to the **number of parameters** in the model (sometimes also to the number of words in the dataset).
- Parameters are the **weights** the deep learning model learned during training.
- **LLMs** have **millions, billions or even trillions** of parameters. For example:
  - BERT: 110M parameters.
  - PaLM 2: 340B parameters.
  - GPT-4: 1T parameters.
- In general, the **more parameters** an LLM has, the **more sophisticated the tasks** it can perform.

How large is an LLM?



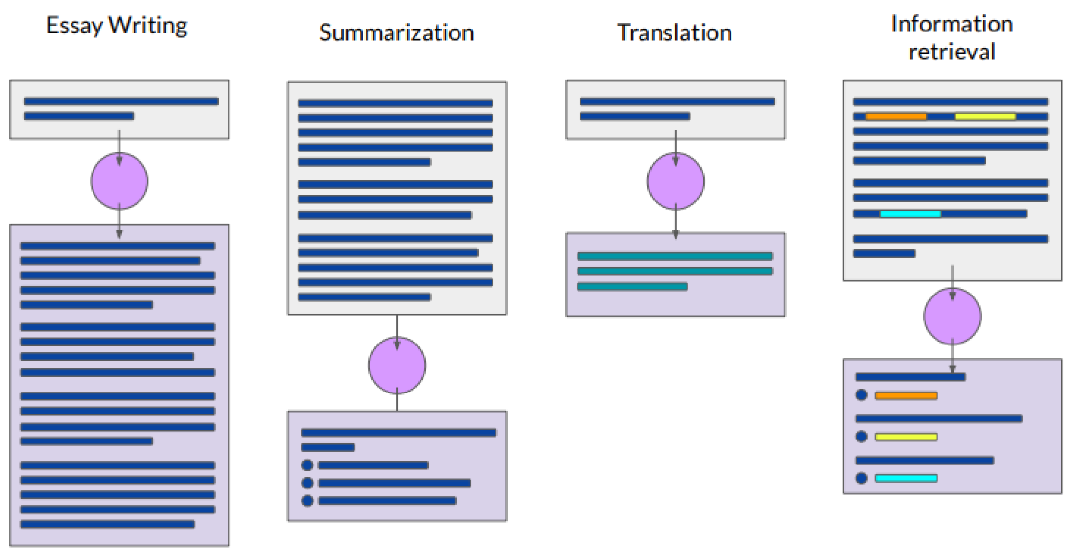
How to interact with LLMs

- The text that you pass to an LLM is known as a **prompt**.
- The space that is available to the prompt is called the **context window**, and this is typically large enough for a few thousand words but differs from model to model.
- The output of the model is called a **completion**.
- The act of using the model to generate text is known as **inference**.



LLM use cases

- **Next word prediction** is the **base concept** behind LLM **text generation**.
- This simple technique is used for a **variety of tasks** in LLMs:
  - Writing information.
  - Summarization.
  - Translation.
  - Information retrieval.

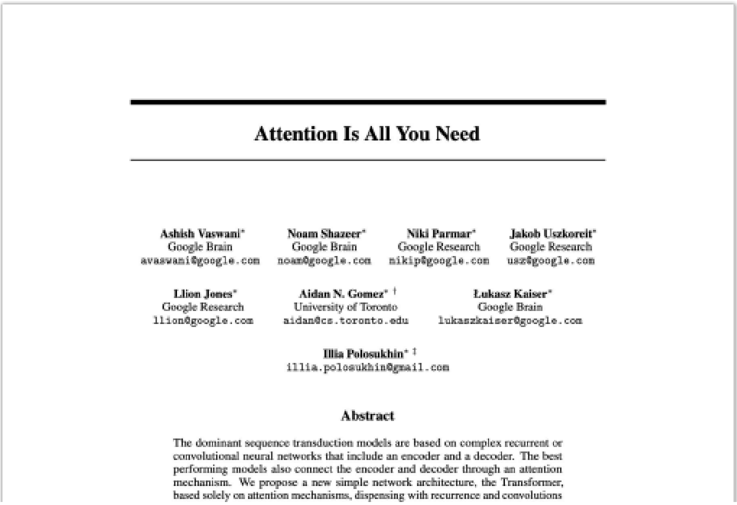


How LLMs work

- Most of the **early Language Models** were created using Recurrent Neural Networks (**RNNs**) models: Long Short-term Memory (LSTM) or Gated Recurrent Units (GRUs) networks.
- However, RNNs were **limited** by the **amount of compute** and **memory** needed to perform well at generative tasks (RNN tries to remember the entire input sequence).
- Also, RNNs suffer from the **vanishing gradient problem** that makes difficult to effectively update weights in the layers of the network, **compromising the ability** to learn and **capacity** of the network.

How LLMs work

- A key development in Language Models was the introduction in 2017 of **transformers**, an architecture designed around the idea of **attention.**
- This made it possible to process longer sequences by focusing on the **most important part of the input** and **solving memory issues** encountered in earlier models.

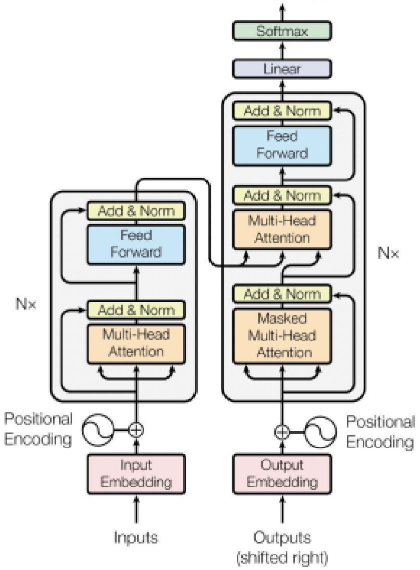


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions

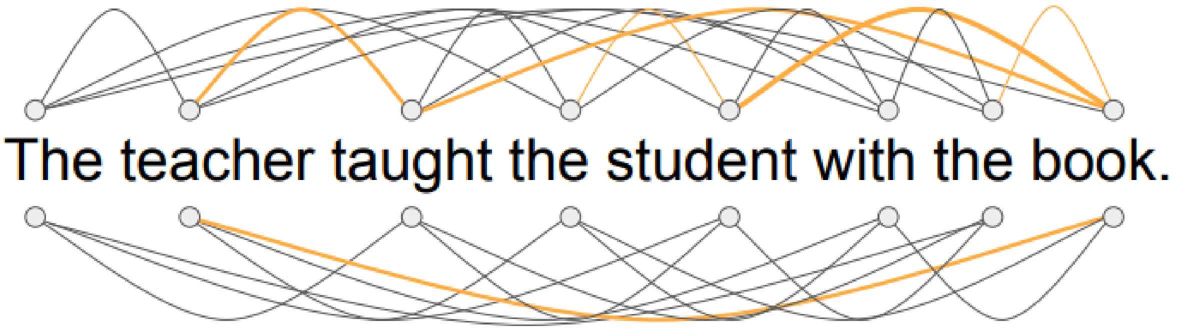


How LLMs work

- The novel **transformer architecture** has produced the **progress** in **generative AI** in general and **LLMs in particular** that we see today.
- Transformer architecture:
  - **can be scaled efficiently** to use multi-core GPUs.
  - **can parallel process input data**, making use of much larger training datasets.
  - is able to **learn to pay attention to the meaning of the words** that are been processed.

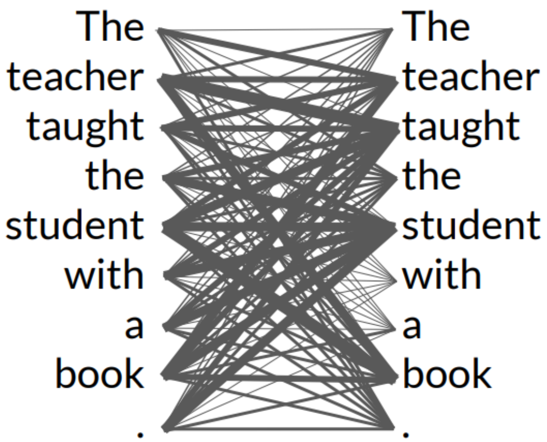
How LLMs work

- The **power of** the **transformer** architecture lies in its ability to learn the **relevance and context** of **all of the words** in a sentence.
  - Not just each word next to its neighbor, but to every other word in a sentence.
- To apply **attention weights** to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input.



How LLMs work

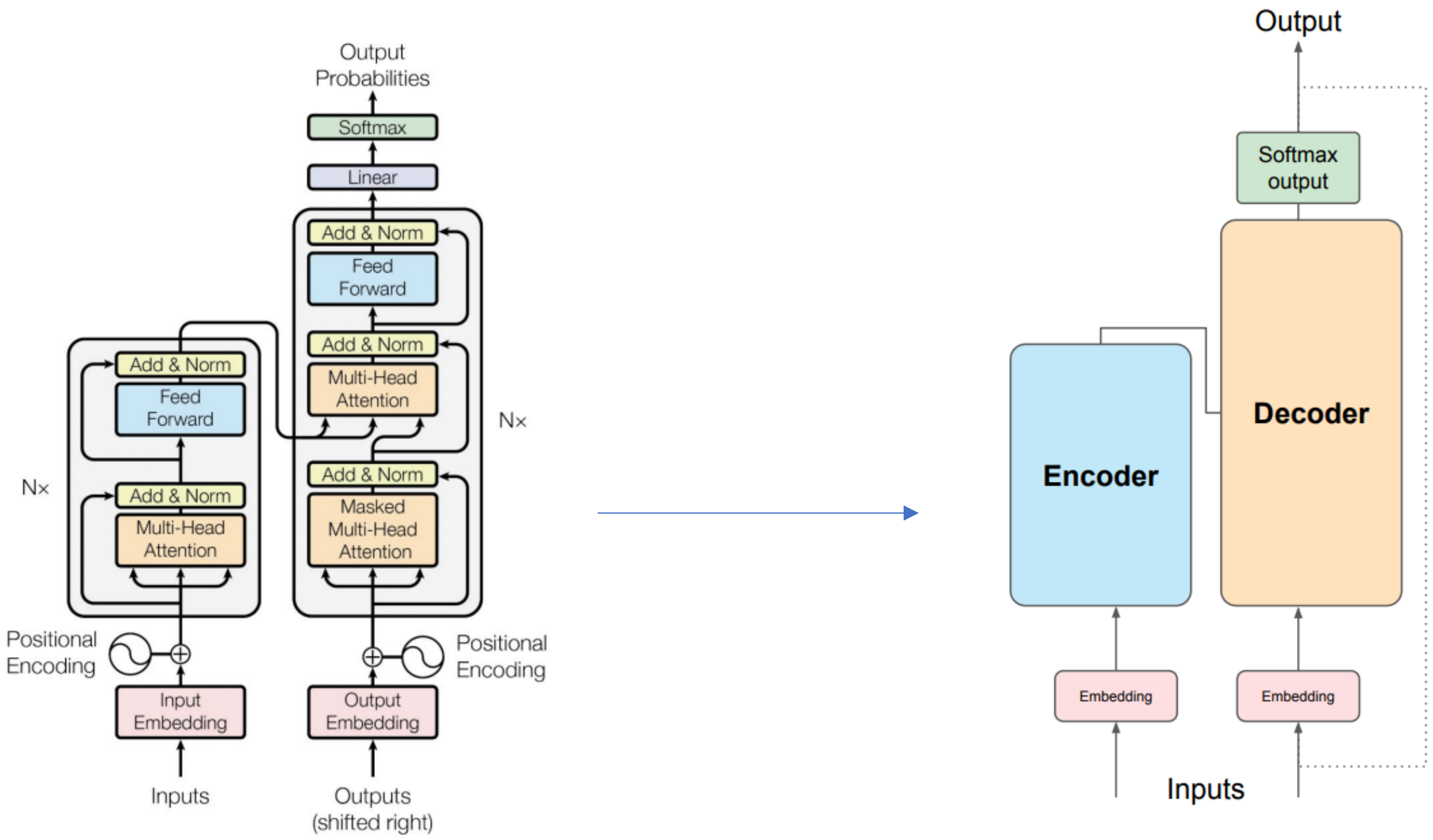
- The **attention weights** are learned during LLM training.
- It is useful to illustrate the attention weights between each word and every other word with an **attention map**:



- This **mechanism** to capture **dependencies and relationships** identifying and weighing the importance within an input sequence is called **self-attention**.
- Self-attention is one of the **key attribute**s of the **transformer** architecture.

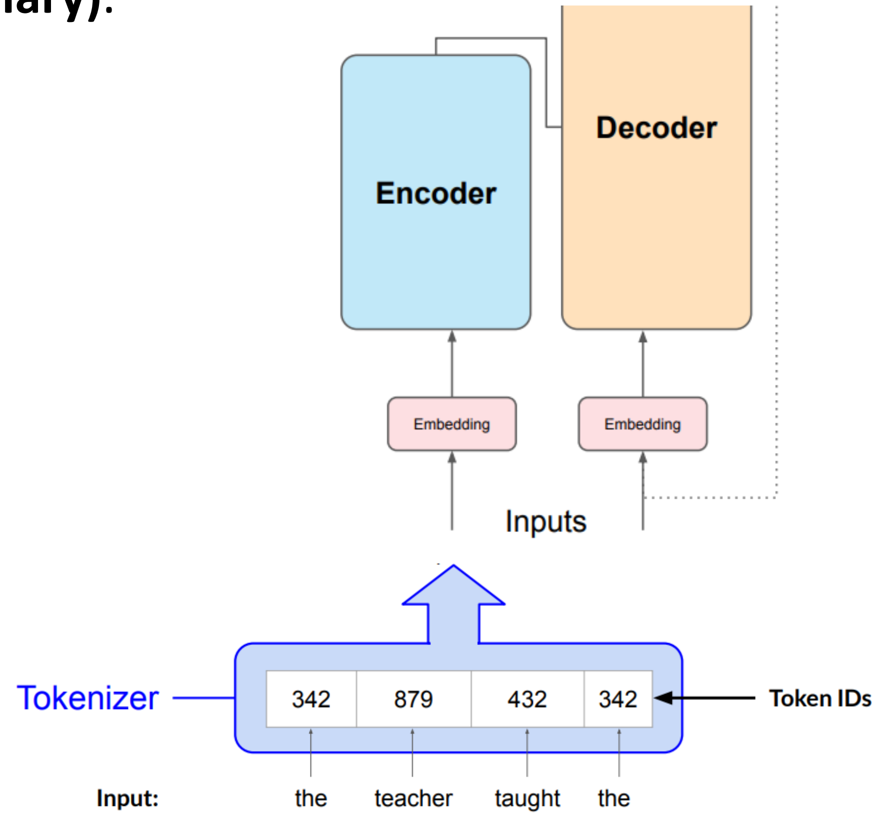
How LLMs work

- Simplified diagram of original **transformer architecture**:



How LLMs work

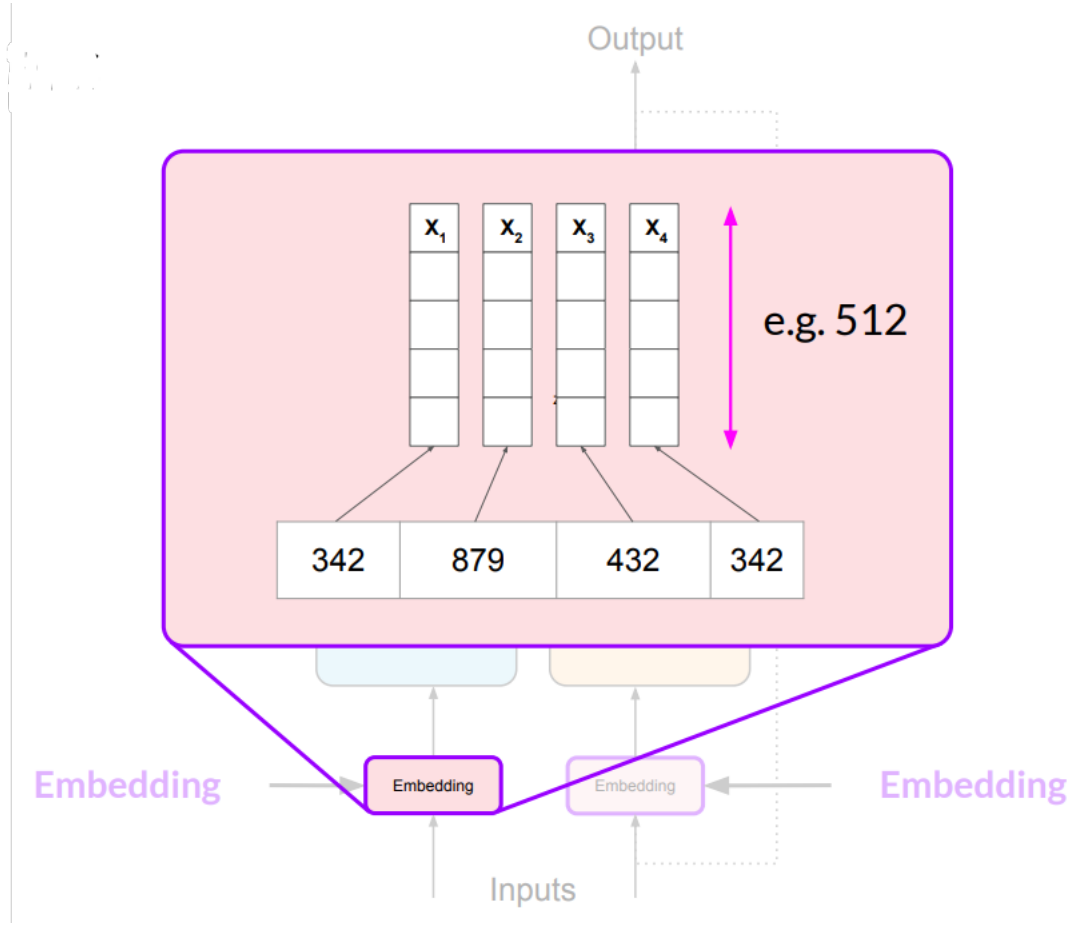
- Before passing texts into the model, **words must be tokenized**.
  - This converts the words into numbers, with each number representing a position in a **dictionary (vocabulary)**.

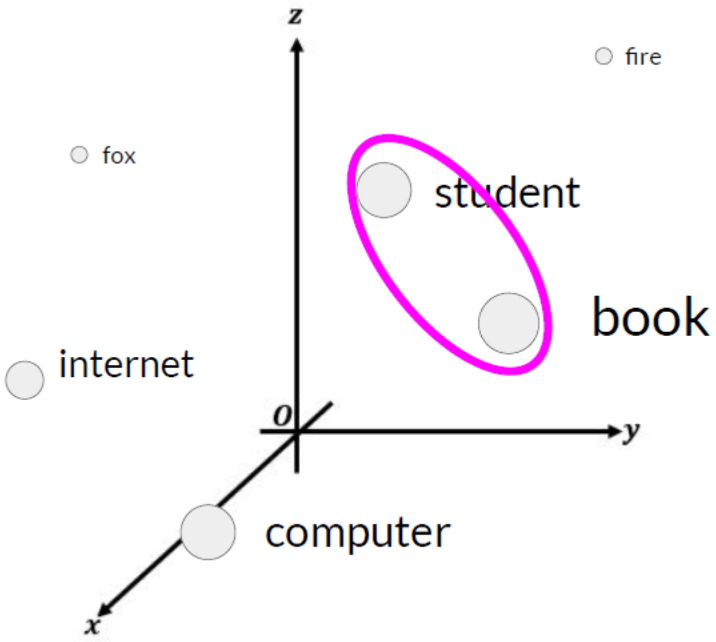


How LLMs work

- Now that input is represented as numbers, it can be passed it to the **embedding layer**.
- This layer is a **high-dimensional trainable vector space** where each token is represented as a multi-dimensional vector and occupies a unique location within that space.
- The intuition is that vectors learn to encode the **meaning and context** of individual tokens in the input sequence.
- It is possible to see that **relate words** are located close to each other in the **embedding space**.

How LLMs work

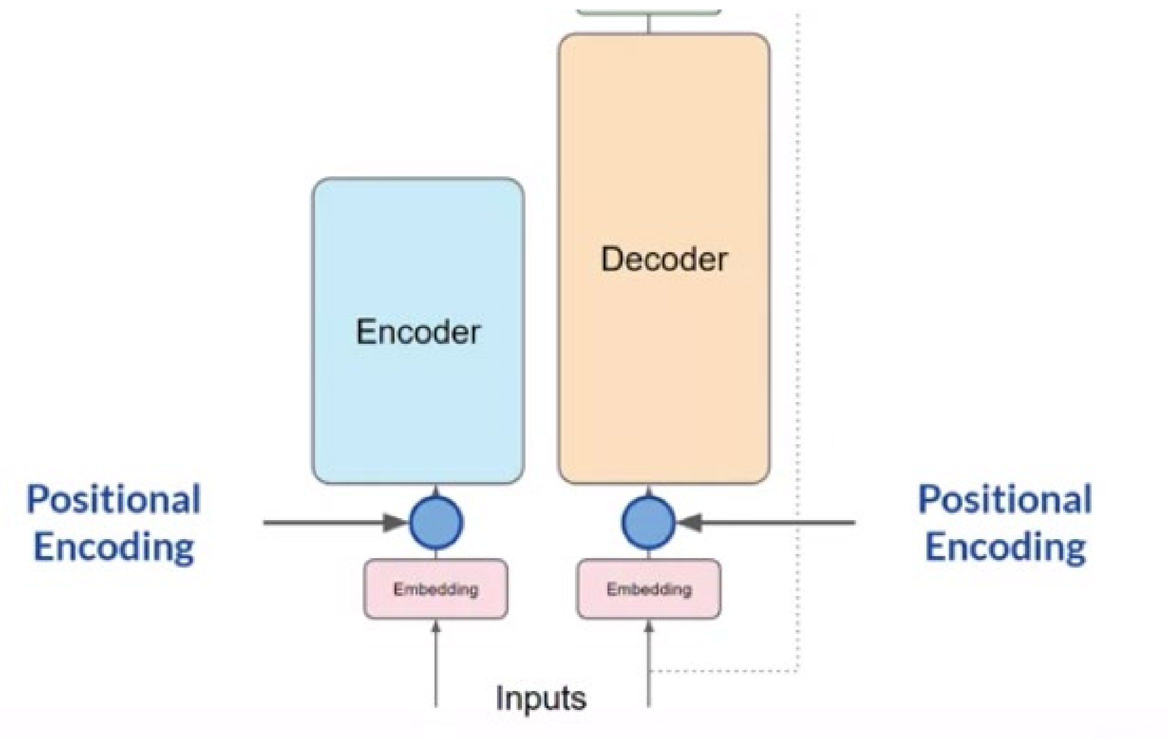




How LLMs work

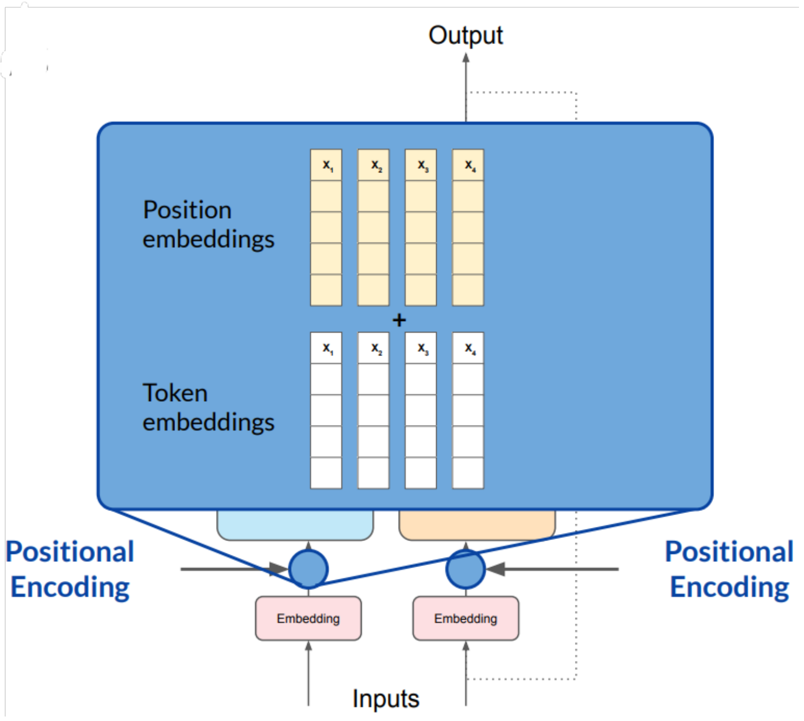
- **Positional information** is added to the model explicitly to retain the information regarding the **order of words in a sentence**.
- **Positional encoding** is the scheme through which the knowledge of the **order** of objects **in a sequence** is maintained.
- The model processes each of the input tokens in parallel, so by **adding the positional encoding**, the information about the **word order is kept** and the **relevance of the position** of the word in the sentence is not lost.

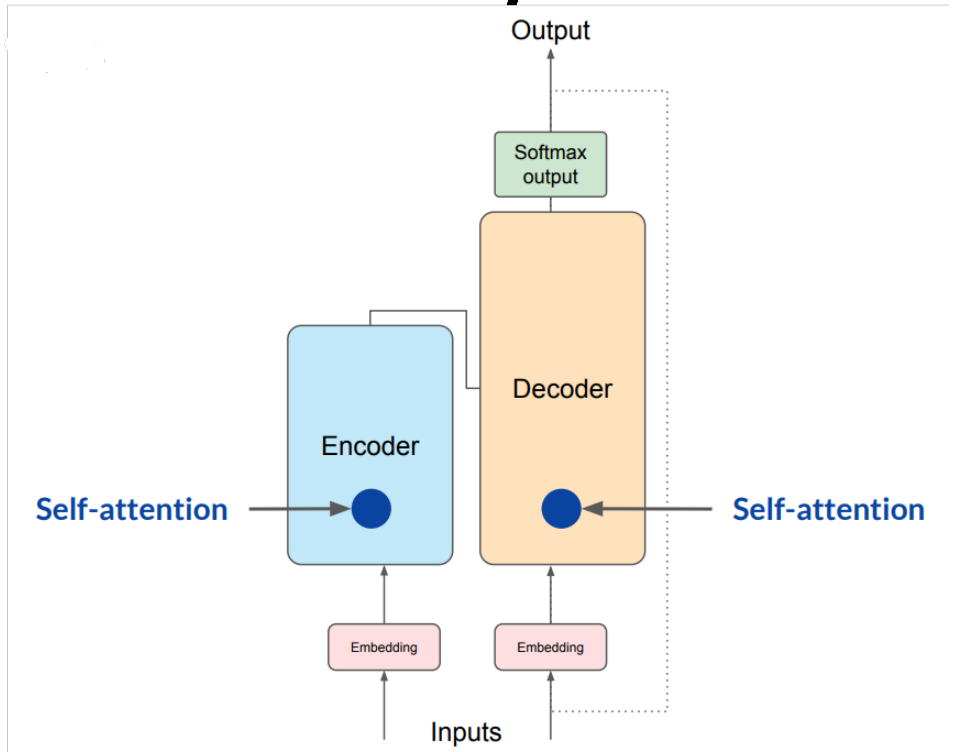
How LLMs work



How LLMs work

- **Token embeddings** and **positional embeddings** are **summed**, and resulting vectors are passed to the **self-attention layer**.

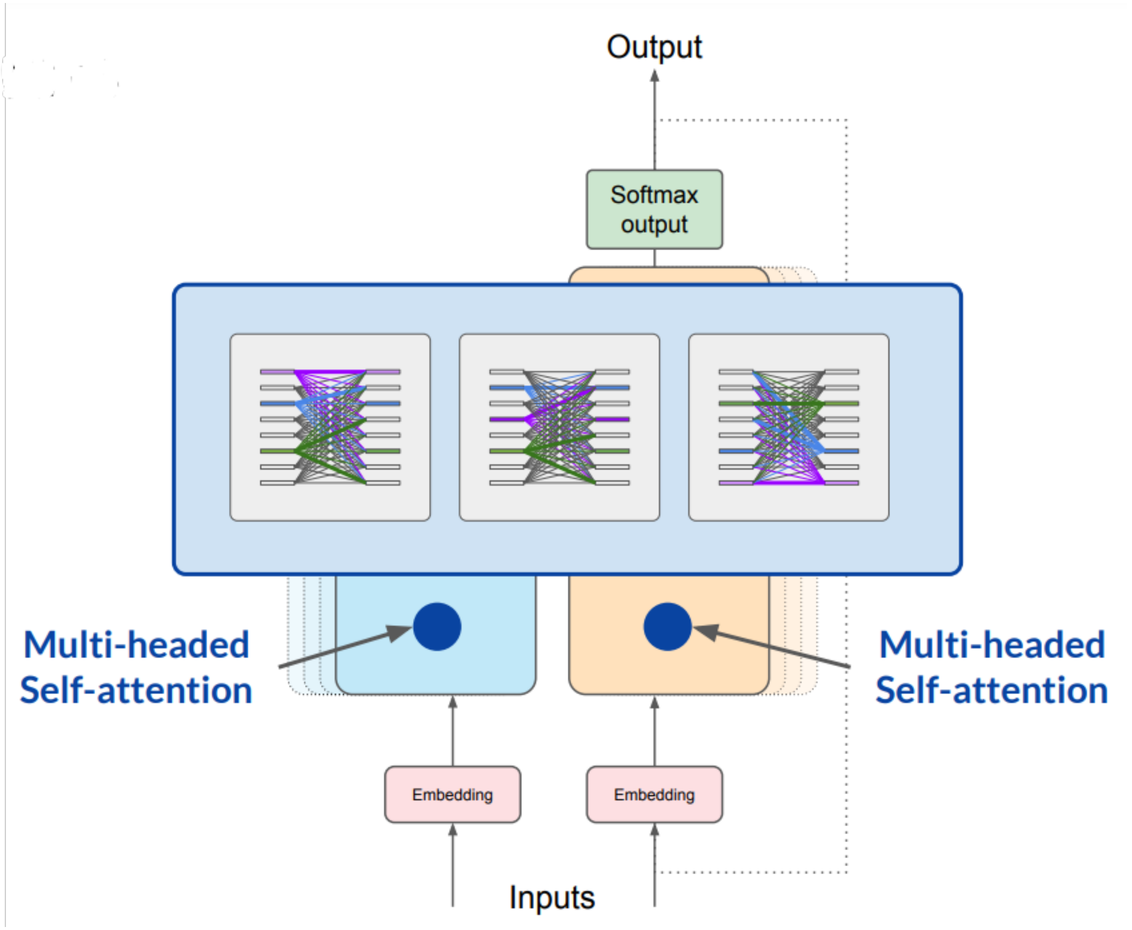




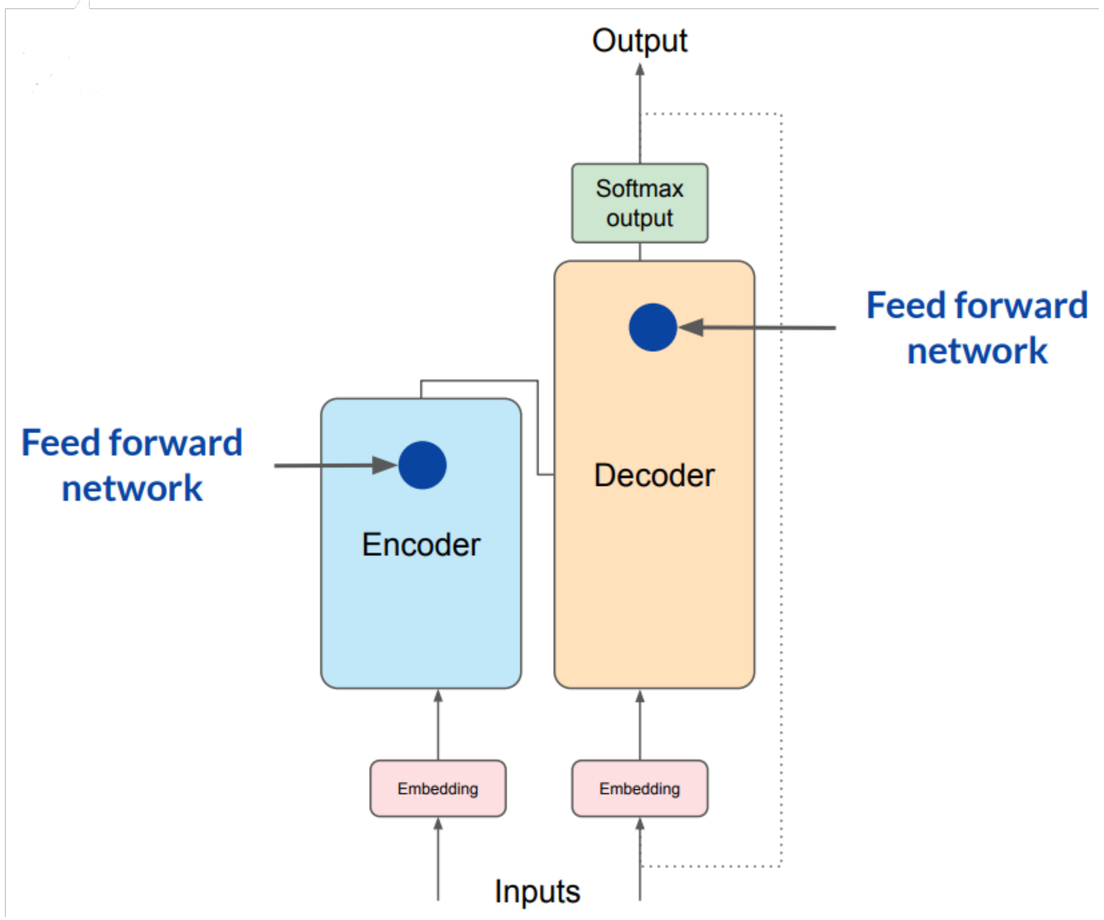
How LLMs work

- In **self-attention layer**, the model analyzes **the relationships between the tokens** in the input sequence and **attends** to different **parts of the input sequence** to better capture the **contextual dependencies** between the words.
- But this does not happen just once: the transformer architecture actually has **multi-headed self-attention**.
  - This means that **multiple sets of self-attention weights** or heads are learned in parallel independently of each other.
  - The intuition is that **each self-attention head** will **learn** a **different aspect of language**.

How LLMs work



How LLMs work



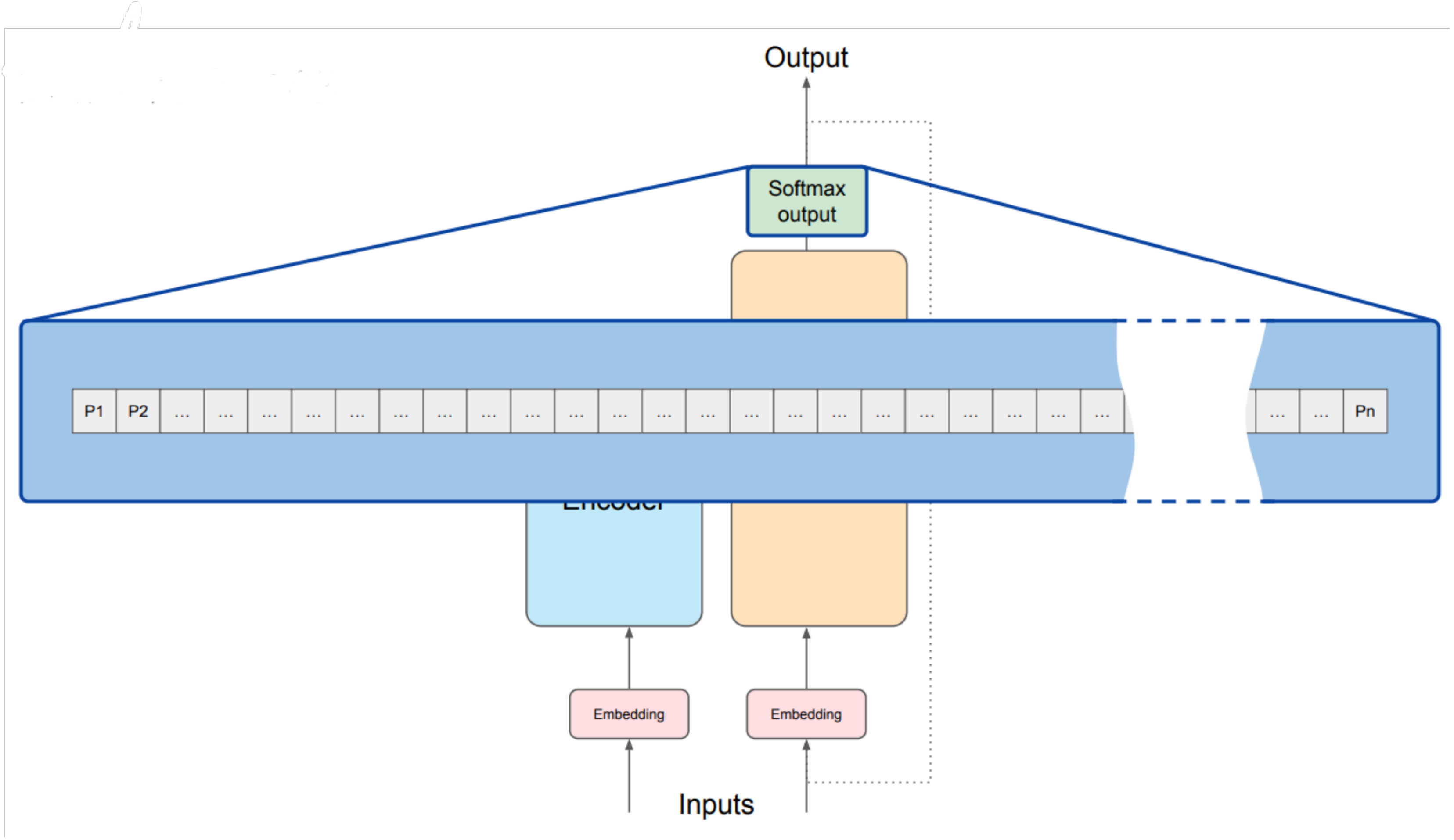
How LLMs work

- After the attention weights have been applied to the input data, the output is processed through a **fully-connected feed-forward** network.
- The output of this layer is a **vector of logits proportional to the probability score** for each token in the dictionary.

How LLMs work

- In the **decoder**, these logits are passed to a **final softmax layer**, where they are **normalized** into a **probability** score for each word.
- This output includes a probability for every single word in the vocabulary.
- One single token will have a score higher than the rest. This is the most likely predicted token.

How LLMs work

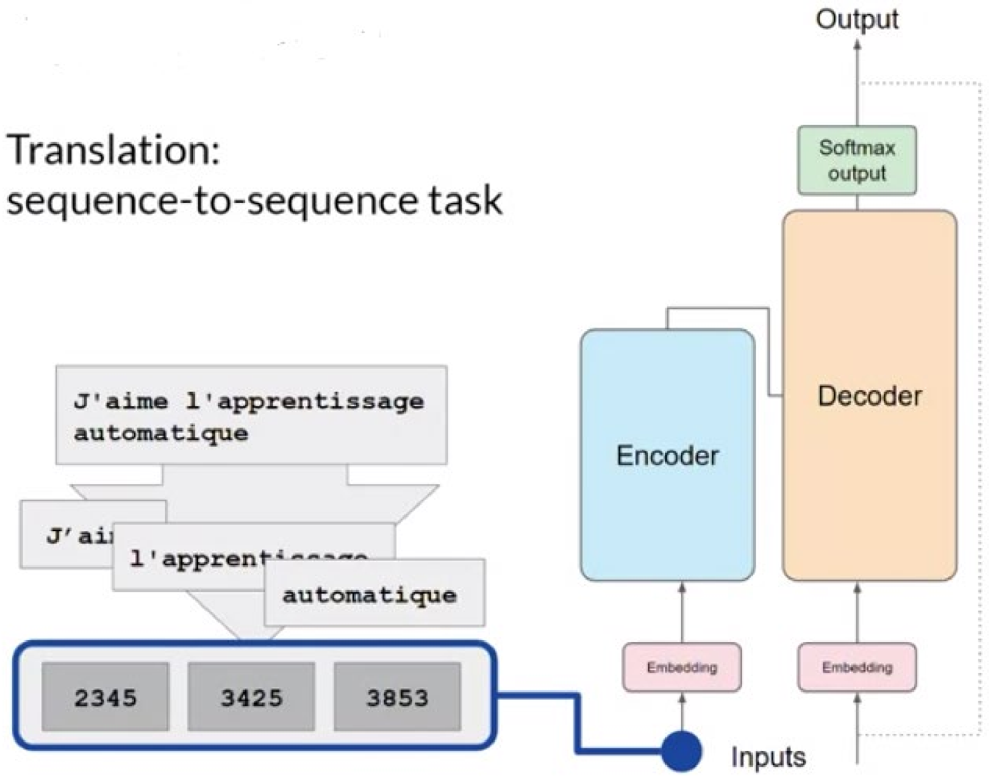


Generating text with LLMs

- Let's look at an **example** of overall prediction process from end to end in a transformer architecture:
  - This example is a **sequence-to-sequence** task (specifically a **translation task**), which was the original objective of the transformer architecture designers.

Generating text with LLMs

- First, the input **words are tokenized**, and these tokens are inserted into the **input on the encoder** side of the network.

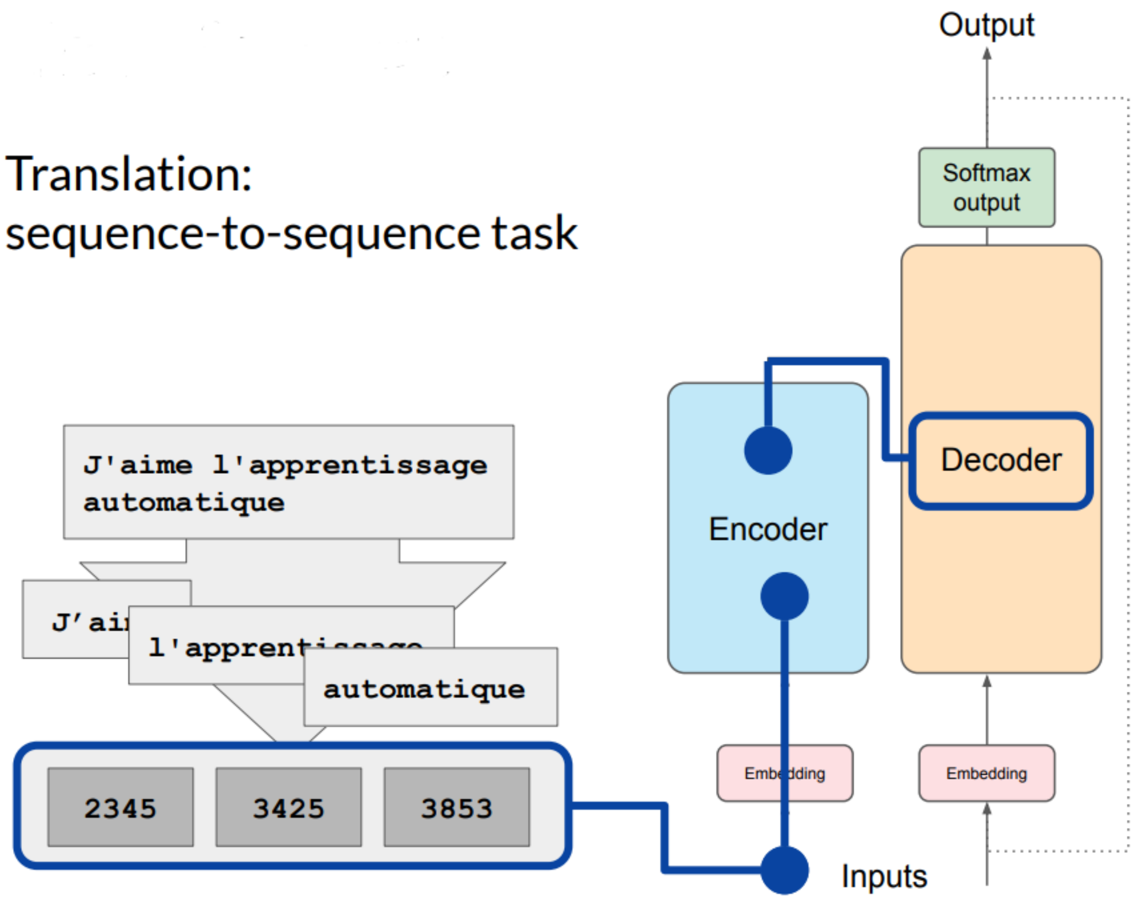


Generating text with LLMs

- **Tokens** are passed **through the embedding layer**, and then **fed into** the **multi-headed attention layers**.
- The **outputs of the multi-headed attention layers** are **fed through** a **feed-forward network** to the output of the encoder.
- At this point, the **data** that leaves the encoder is a deep **representation of the structure and meaning of the input sequence**.
- This **representation** is **inserted** into the middle of the **decoder** to influence the decoder's self-attention mechanisms.

Generating text with LLMs

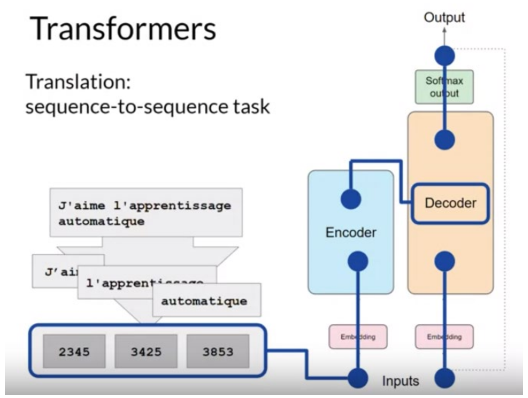
Translation:
sequence-to-sequence task

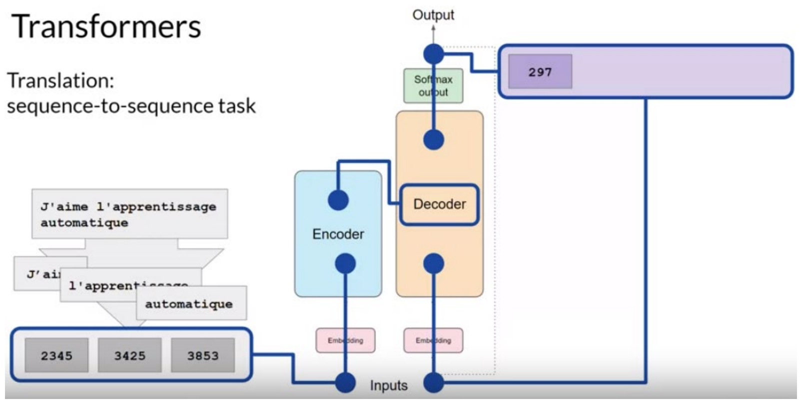


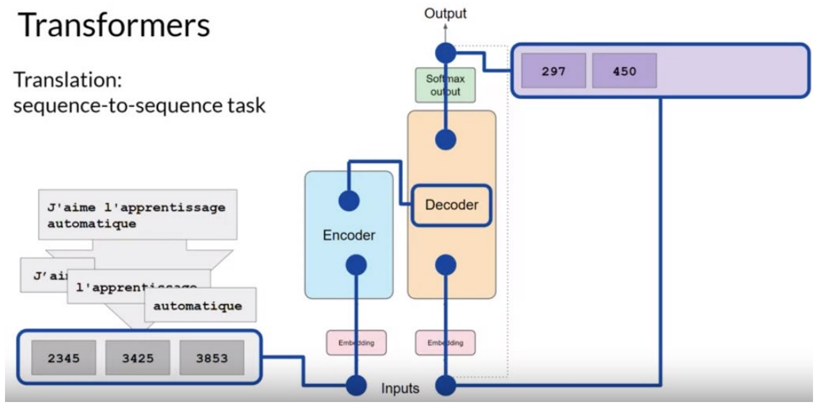
Generating text with LLMs

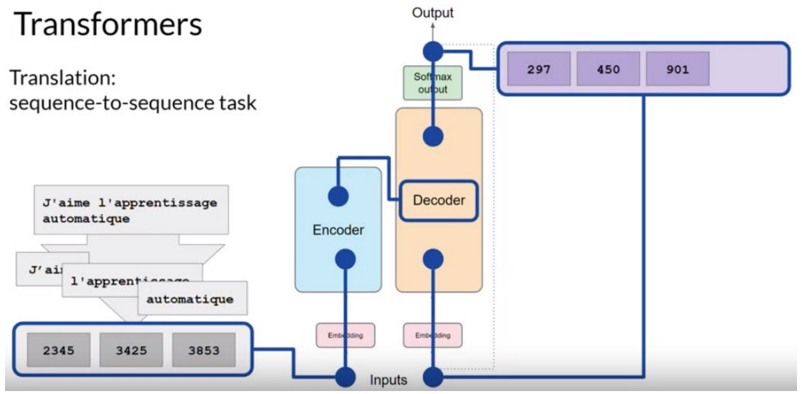
- Next, **a start of sequence token** is added to the **input of the decoder**.
- This triggers the decoder to predict the next token, which it does based on the contextual understanding that it is being provided from the encoder.
- The **output of the decoder's** self-attention layers gets passed through the decoder feed-forward network and through a final softmax output layer. At this point, we have our **first token**.
- This **loop continues**, passing the **output token back to the input decoder** to trigger the generation of the next token, **until the model predicts an end-of-sequence token**.
- At this point, the final sequence of tokens can be detokenized into words, and you have your output.

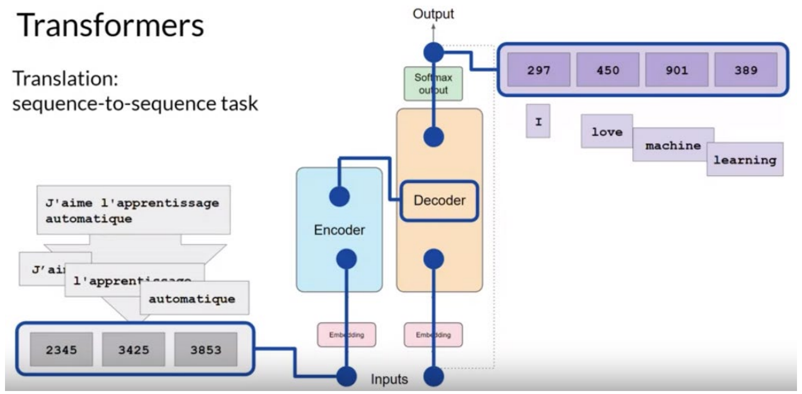
Generating text with LLMs







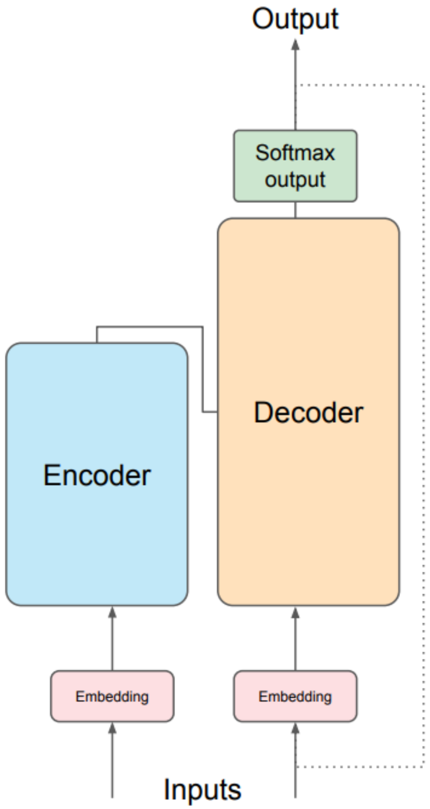




Generating text with LLMs

- Summary:

Encoder
Encodes inputs ("prompts") with contextual understanding and produces one vector per input token.

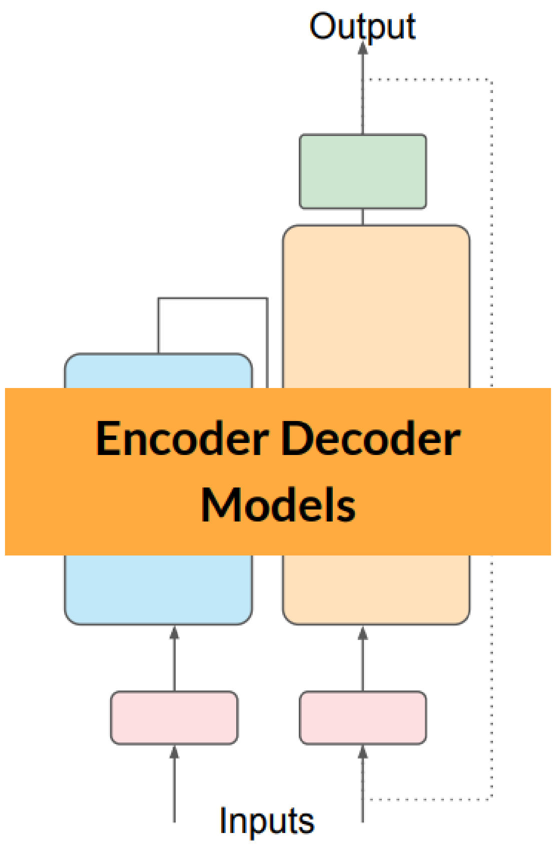


Decoder
Accepts input tokens and generates new tokens.

LLM architecture variations

- **Encoder-decoder models** perform **sequence-to-sequence tasks** such as translation, where the input sequence and the output sequence can be different lengths.
- It can be **trained** this type of models to **perform text generation tasks**.
- Example: BART and T5 models.

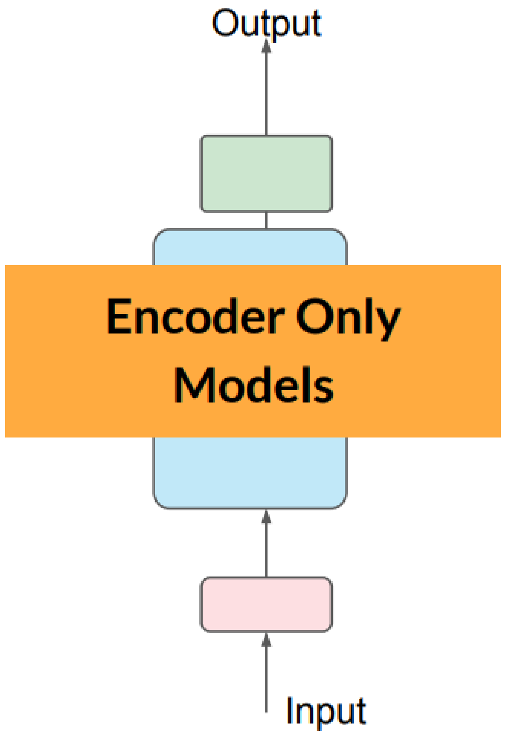
LLM architecture variations



LLM architecture variations

- **Encoder-only models** are **sequence-to-sequence** models (the input sequence and the output sequence have the same length) **use to extract the relevant information from the input** and generate embeddings for other posterior tasks like classification or sentiment analysis.
- Example: BERT model.

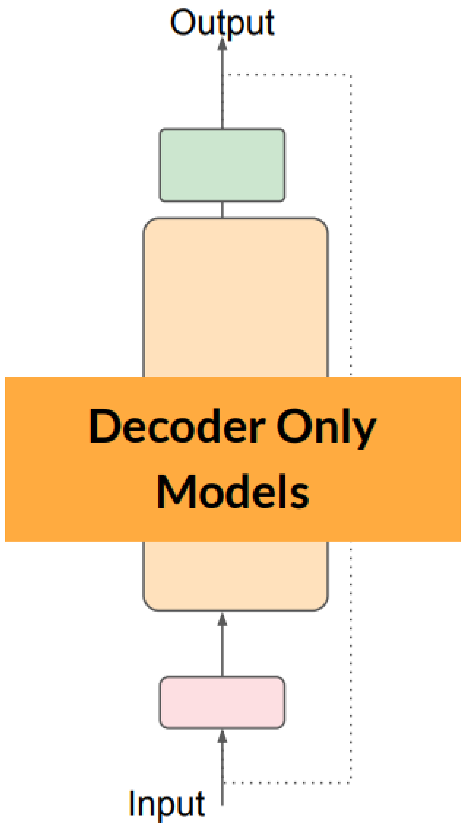
LLM architecture variations



LLM architecture variations

- **Decoder-only models** are some of the **most commonly used today**. They are similar to the encoder-decoder models, but the input information are not encoded explicitly.
- The **are trained to predict the next token** in a sequence given the previous tokens.
- They can generalize to most tasks.
- Example: GPT and LlaMA family models

LLM architecture variations

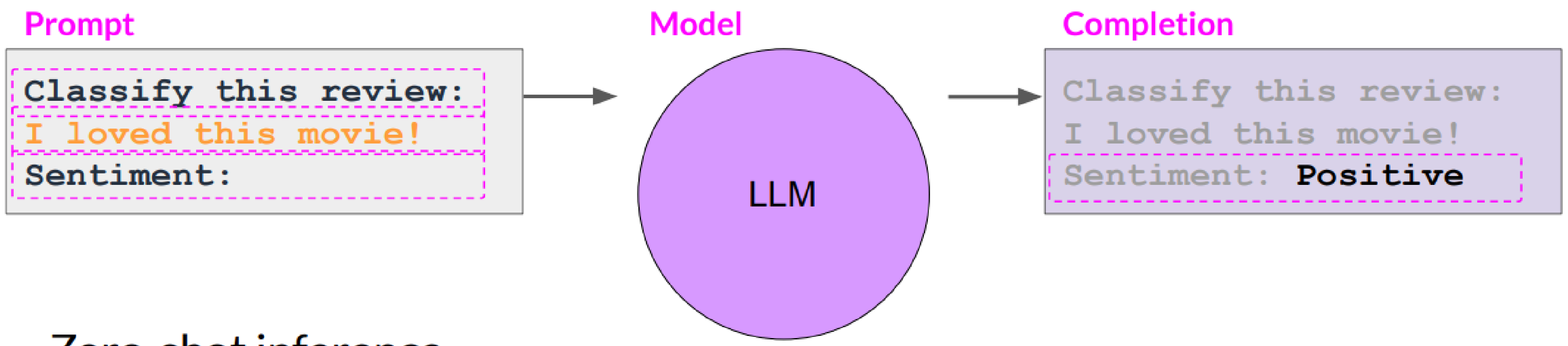


Prompting

- The work to develop and improve the prompt is known as **prompt engineering**.
- A powerful **strategy** to get the model **to produce better outcomes** is to **include examples of the task** that you want the model to carry out inside the prompt. Providing examples inside the context window is called **in-context learning**.
- So, with in-context learning, you can **help LLMs learn more about the task** being asked by including examples or additional data in the prompt.

Prompting

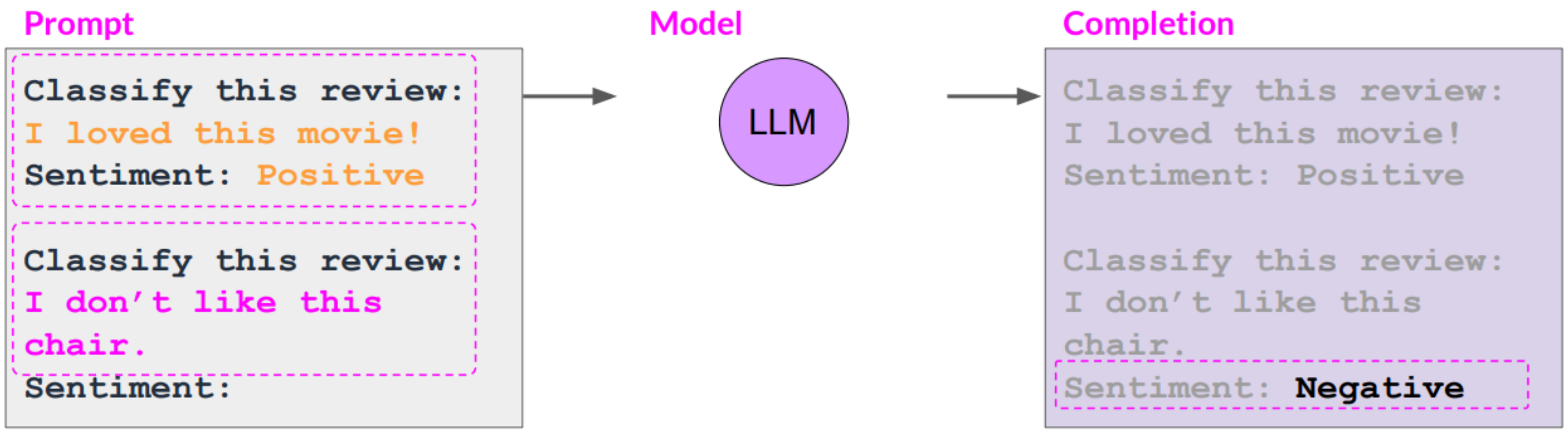
In-context learning (ICL) - zero shot inference



Zero-shot inference

Prompting

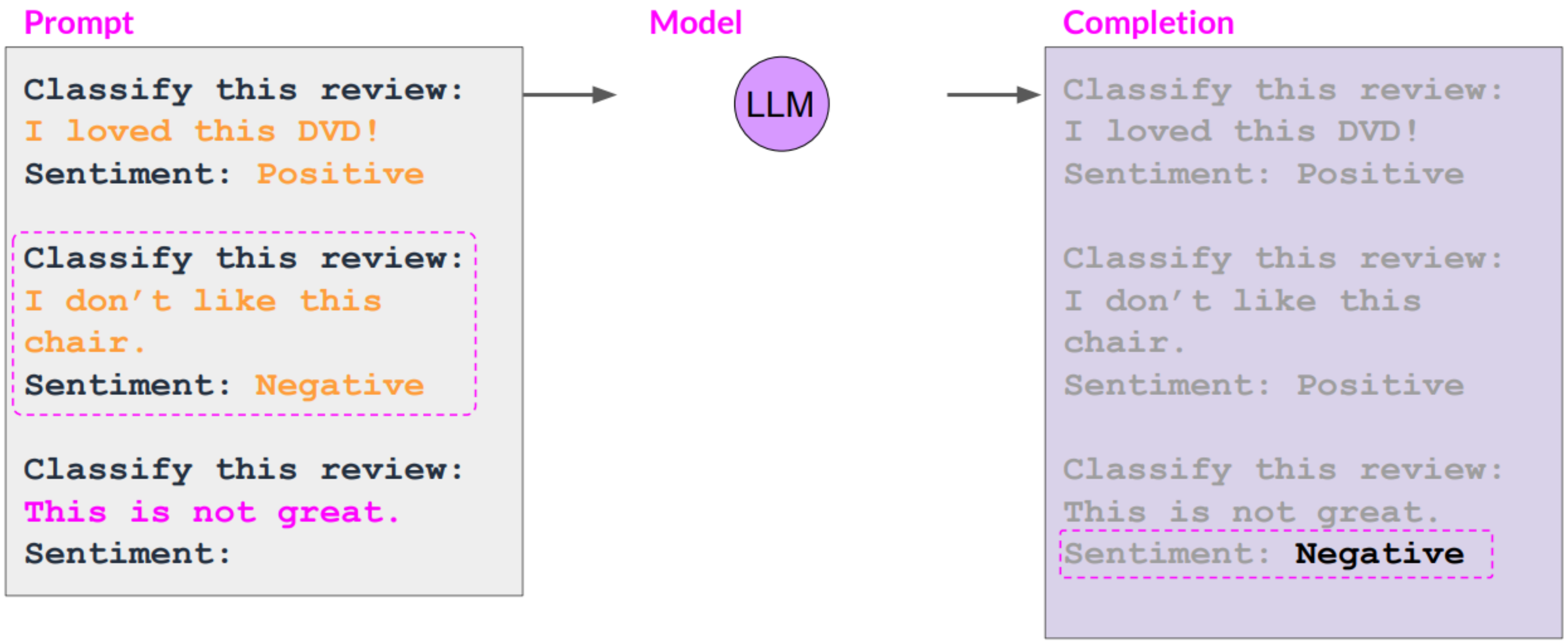
In-context learning (ICL) - one shot inference



One-shot inference

Prompting

In-context learning (ICL) - few shot inference



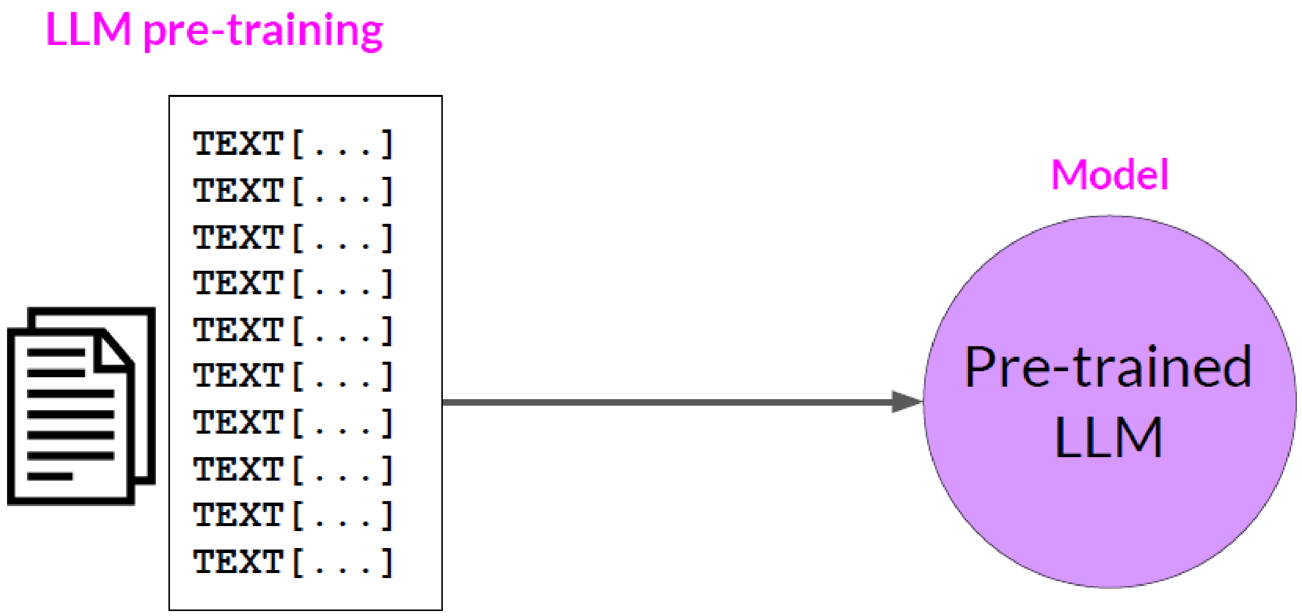
Prompting

- Prompts can be engineered to encourage the model to learn by examples.
- While the **largest models are good at zero-shot inference** with no examples, **smaller models can benefit from one-shot or few-shot inference** that include examples of the desired behavior.
- However, the **context window must be remembered** because **there is a limit on the amount of in-context learning** can be passed into the model.

Fine-tuning

- **Fine-tuning** is a method that it can be used to **improve** the **performance** of an existing LLM model for a **specific use case**.
- Firstly, there is a **pre-training** step where **LLM** is trained using vast amounts of **unstructured textual** data via **self-supervised learning**.

Fine-tuning

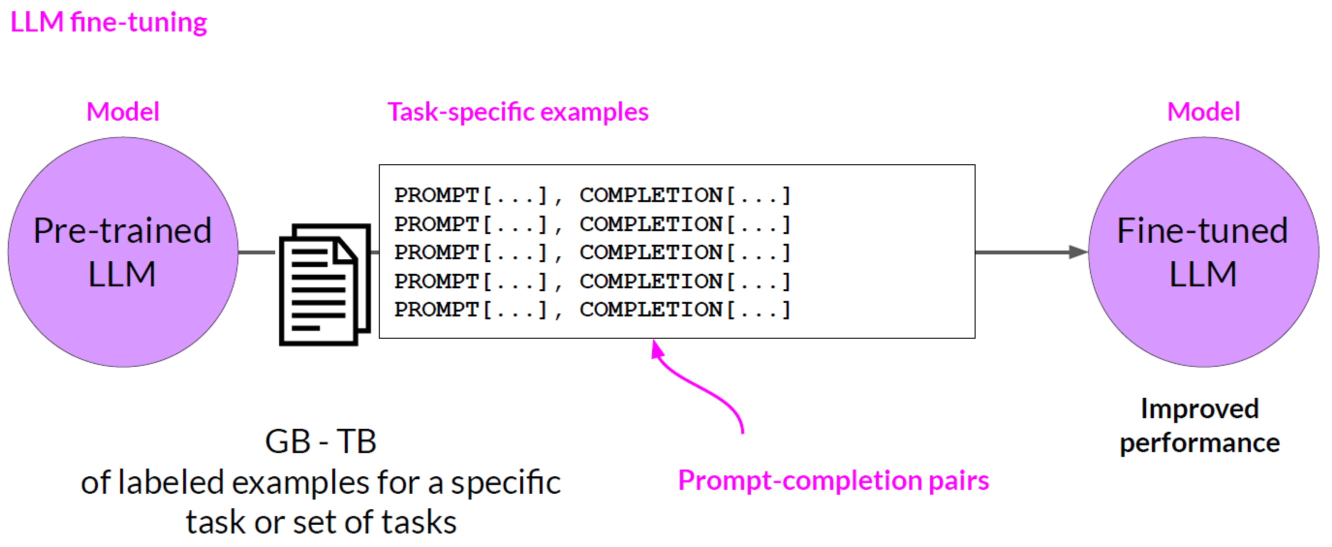


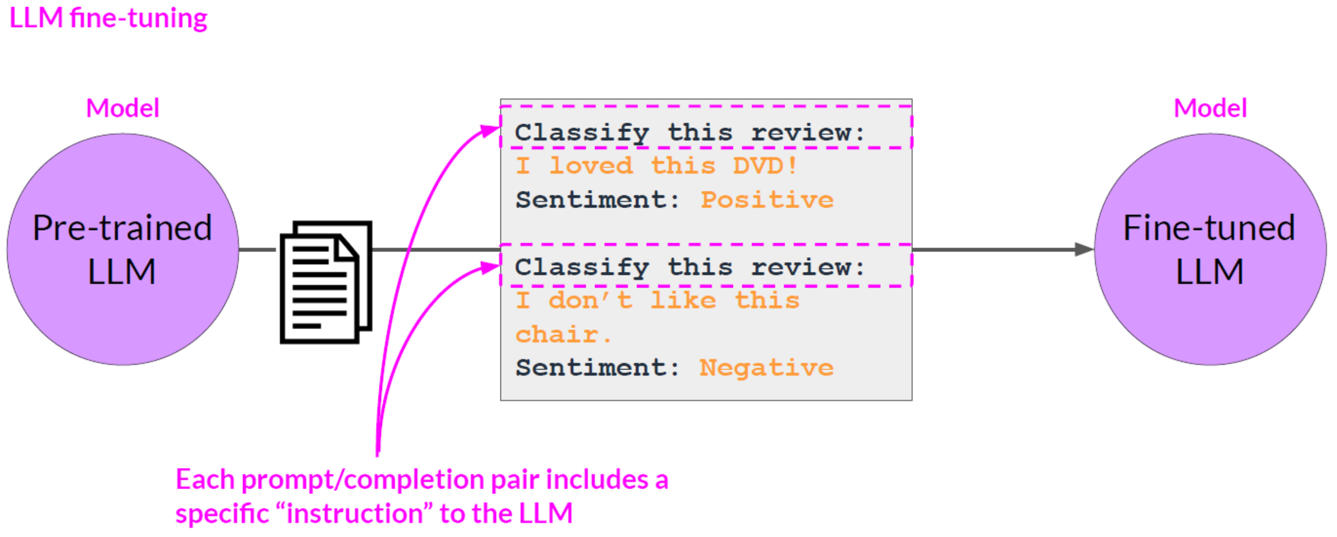
GB - TB - PB
of unstructured textual data

Fine-tuning

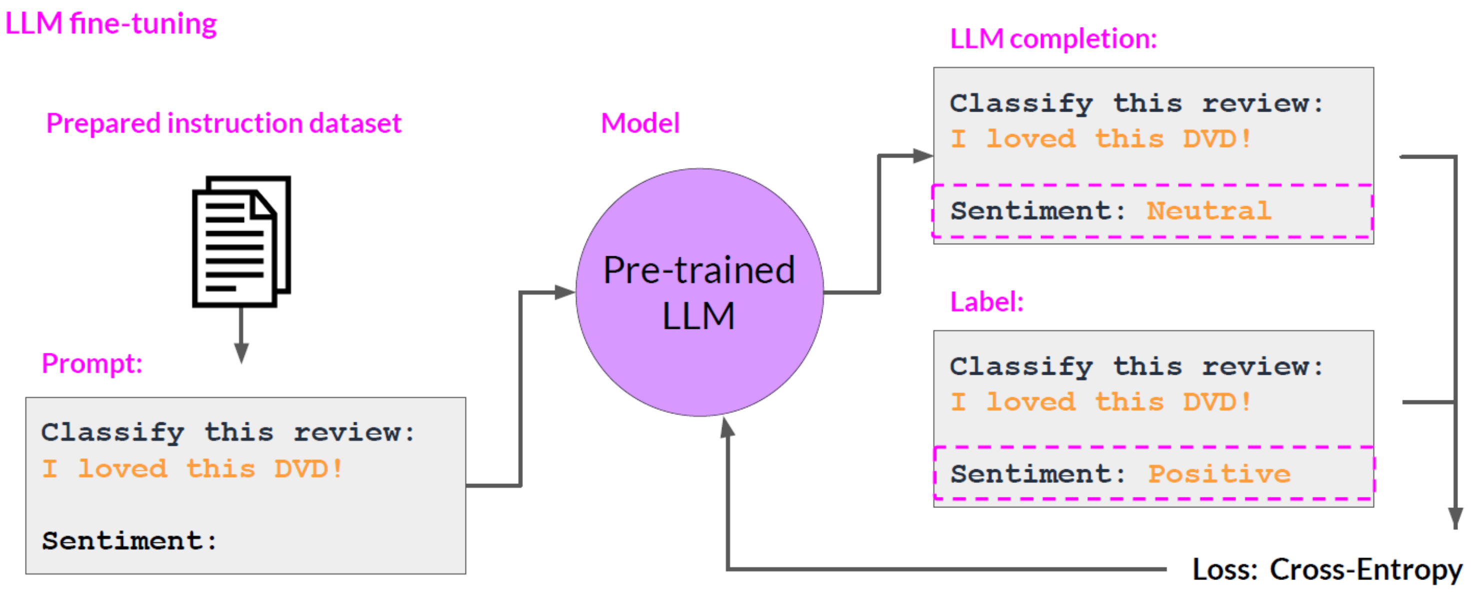
- After that, **fine-tuning** is used: it is a **supervised learning process** where you use a **data set of labeled** examples to update the weights of the LLM.
- The fine-tuning process extends the training of the model to **improve** its ability to generate good completions for a **specific task**.
- One strategy is known as **instruction fine-tuning**: it trains the model using **examples** that **demonstrate how it should respond to a specific instruction**.
  - The labeled examples are prompt-completion pairs.
- Interestingly, **good results can be achieved with** relatively **few examples**: often just 500-1,000 examples can result in good performance (in contrast to the billions of pieces of texts that the model saw during pre-training).

Fine-tuning


LLM fine-tuning
Model
Pre-trained LLM
Task-specific examples
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
Model
Fine-tuned LLM
Improved performance
GB - TB
of labeled examples for a specific task or set of tasks
Prompt-completion pairs


LLM fine-tuning
Model
Pre-trained LLM
Classify this review:
I loved this DVD!
Sentiment: Positive
Classify this review:
I don't like this chair.
Sentiment: Negative
Model
Fine-tuned LLM
Each prompt/completion pair includes a specific "instruction" to the LLM

Fine-tuning



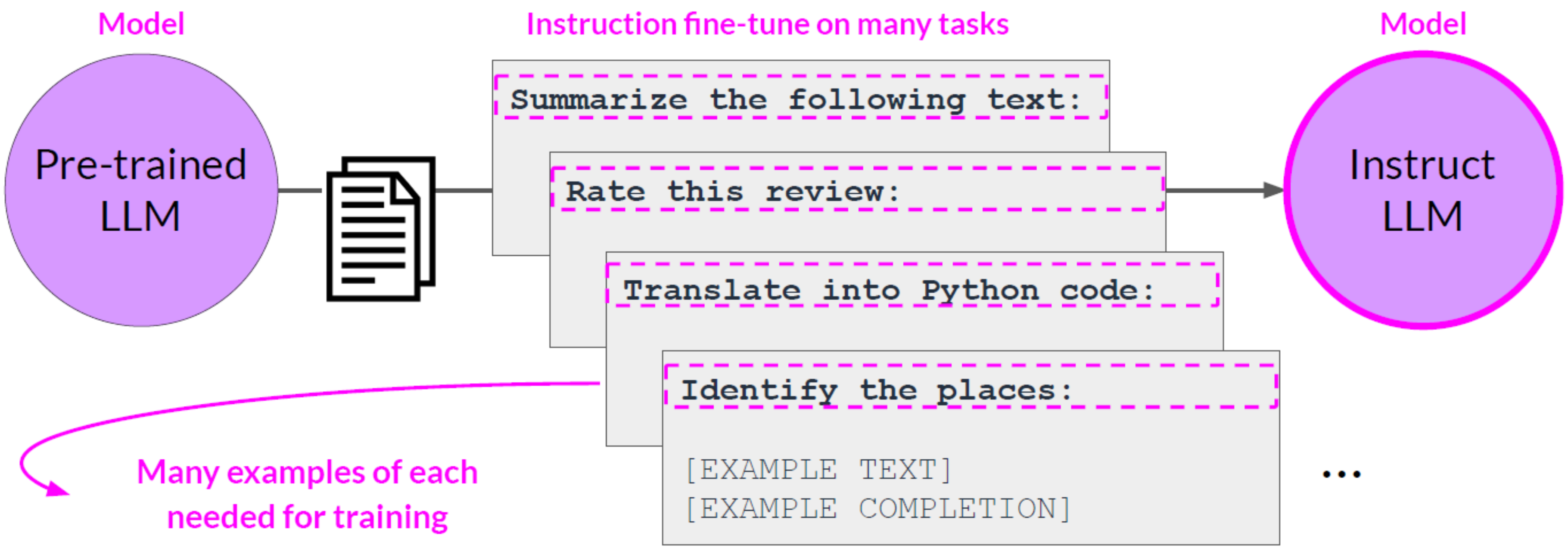
LLM fine-tuning
Prepared instruction dataset
Model
Prompt:
Classify this review:
I loved this DVD!
Sentiment:
Pre-trained LLM
LLM completion:
Classify this review:
I loved this DVD!
Sentiment: Neutral
Label:
Classify this review:
I loved this DVD!
Sentiment: Positive
Loss: Cross-Entropy

Fine-tuning

- However, there is a **potential downside** to **fine-tuning** on a single task: the process may lead to a phenomenon called **catastrophic forgetting**.
- This happens because the full **fine-tuning process modifies the weights** of the original LLM.
- While this leads to **great performance on the single fine-tuning task**, it can **degrade performance on other tasks**.
- This can be avoided with:
  - **Multi-task** instruction **fine-tuning:** fine-tuning on multiple tasks at the same time.
  - **Parameter Efficient Fine-tuning (PEFT):** a set of techniques that preserves the weights of the original LLM training only a small number of task-specific adapter layers and parameters.

Fine-tuning

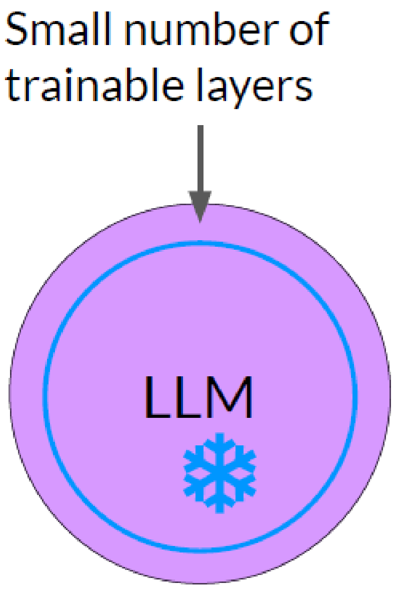
- **Multi-task** instruction **fine-tuning**



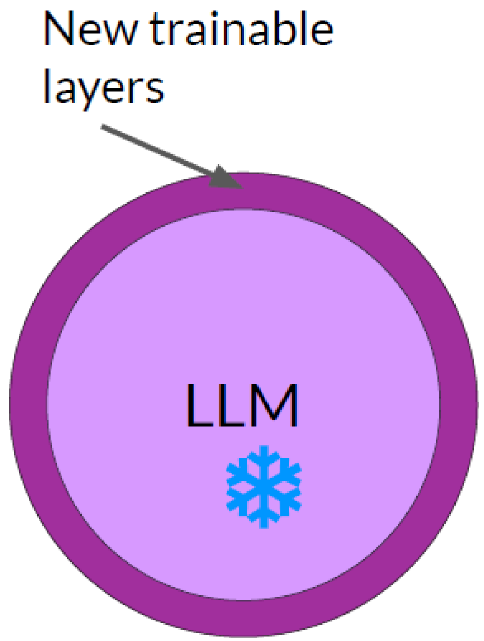
Fine-tuning

- In **full fine-tuning**, **every model weight** is **updated** during supervised learning process.
- In **PEFT**, methods only update a **small subset of parameters**.
  - Some techniques freeze most of the model weights and focus on **fine-tuning a subset of existing model parameters**, for example, particular layers or components.
  - Other techniques do not touch the original model weights at all, and instead **add a small number of new parameters or layers and fine-tune** only the new components.

Fine-tuning



LLM with most layers frozen



LLM with additional layers for PEFT

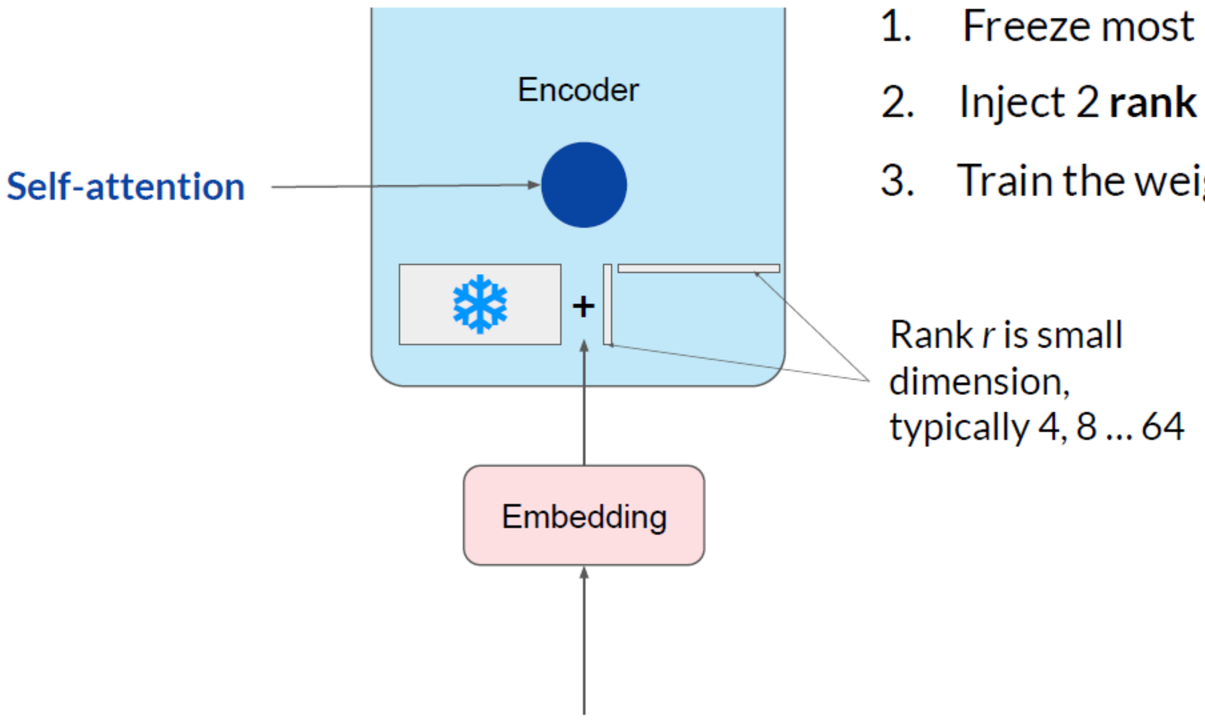
Fine-tuning

- Specifically, **PEFT** methods are:
  - **Selective methods** are those that **fine-tune only a subset of the original LLM parameters** with several approaches to identify which parameters you want to update. There is the option to train only certain components of the model or specific layers, or even individual parameter types.
    - A subgroup called **reparameterization methods** that also work with the original LLM parameters but reducing the number of parameters to train by creating new low rank transformations of the original network weights. A commonly used technique of this type is Low-Rank Adaptation of Large Language Models (**LoRA**).
  - **Additive methods** carry out fine-tuning by **keeping all of the original LLM weights frozen and introducing new trainable components**.
    - **Adapter methods** add new trainable layers to the architecture of the model, typically inside the encoder or decoder components after the attention or feed-forward layers.
    - **Soft prompt methods**, on the other hand, keep the model architecture fixed and frozen, and focus on manipulating the input to achieve better performance. This can be done by adding trainable parameters to the prompt embeddings or keeping the input fixed and retraining the embedding weights.

Fine-tuning - LoRA

- In **encoder and decoder** parts of the transformer there are two kinds of neural networks: **self-attention** and **feed-forward** networks.
- The **weights** of these networks are **learned** during **pre-training**.
- **LoRA** freezes all of the original model parameters and then **injecting a pair of rank decomposition matrices** alongside the original weights.
  - The dimensions of the smaller matrices are set so that their product is a matrix with the same dimensions as the weights they're modifying.
- Researchers have found that applying LoRA to just the self-attention layers of the model is often enough to fine-tune for a task.

Fine-tuning - LoRA

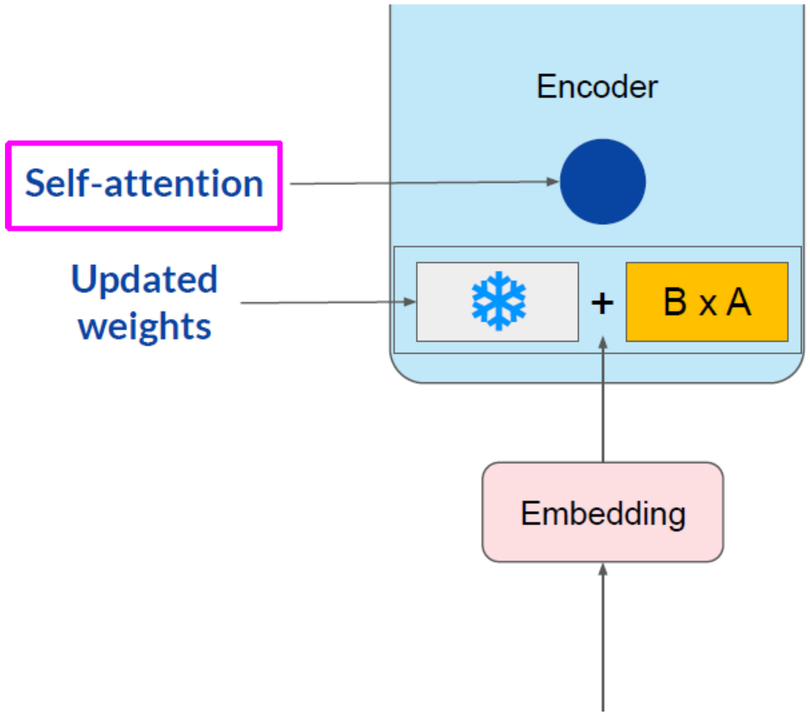


1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Fine-tuning - LoRA

- For **inference**, the two **low-rank matrices** are **multiplied** together to create a matrix with the same dimensions as the frozen weights.
- Then they are **added this to the original weights** and **replace them** in the model with these updated values.
- There is now a LoRA fine-tuned model that can carry out a specific task.
- There is little to **no impact** on **inference latency**.

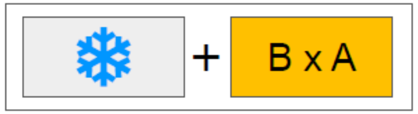
Fine-tuning - LoRA



Steps to update model for inference:

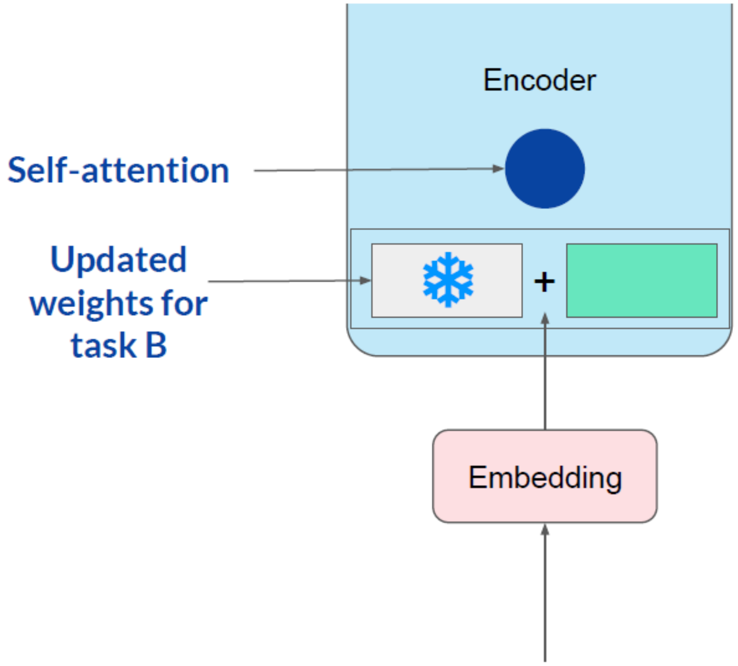
1. Matrix multiply the low rank matrices

2. Add to original weights

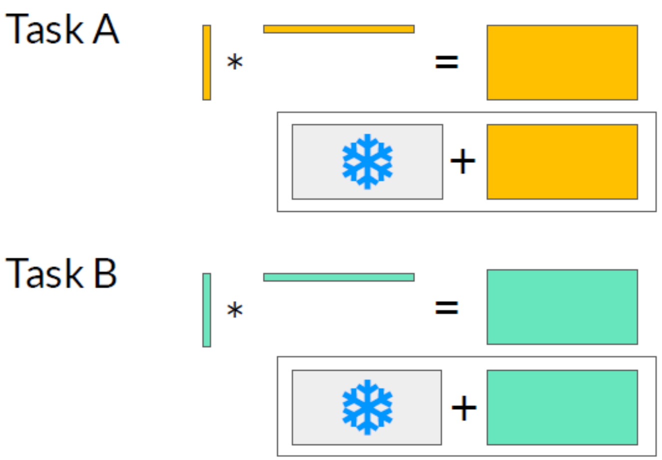


Fine-tuning - LoRA

- It can be fine-tuned a different set of LoRA matrices for different tasks and then switch them out at inference time by updating the weights.



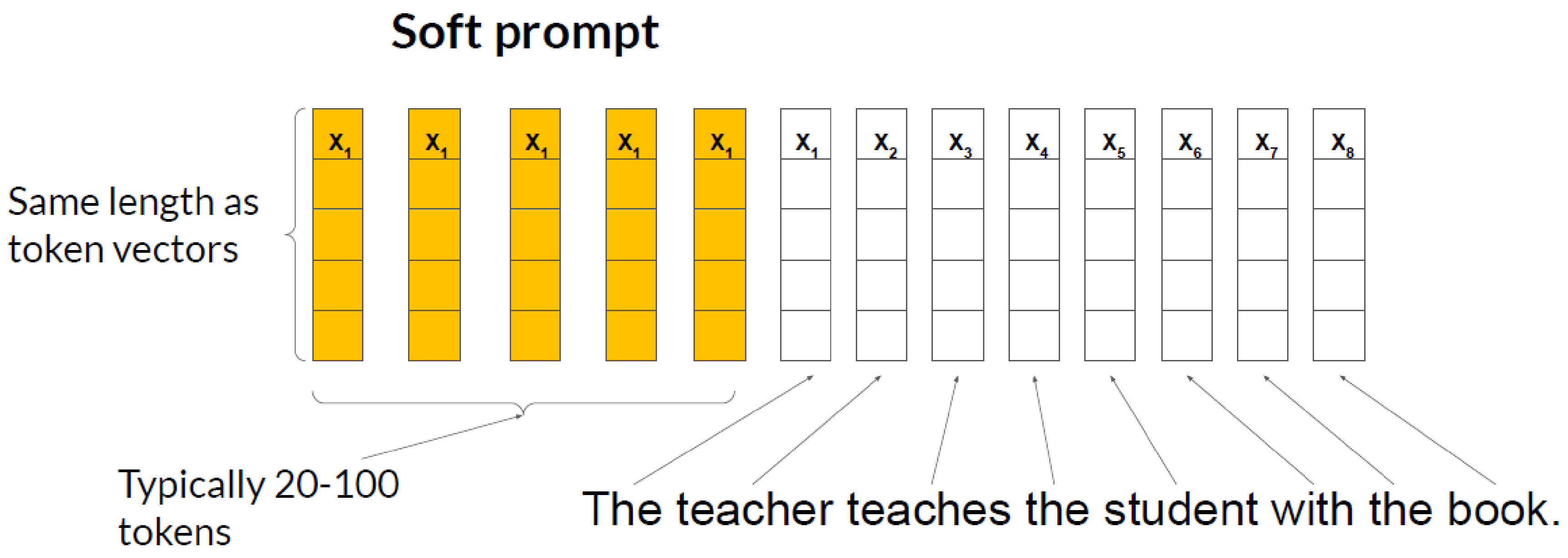
1. Train different rank decomposition matrices for different tasks
2. Update weights before inference



Fine-tuning - Soft prompts

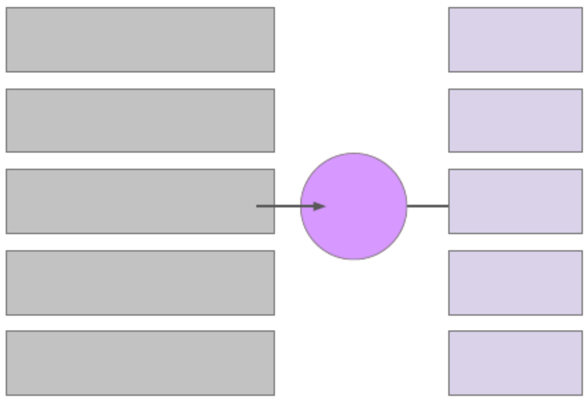
- With **prompt tuning**, additional trainable **tokens are added to the prompt** and leave it up to the supervised learning process to determine their optimal values.
- The set of trainable tokens is called **soft prompts** and is **added to the embedding vectors** representing the input text.
- The soft prompt vectors have the same length as the embedding vectors of the language tokens.

Fine-tuning - Soft prompts



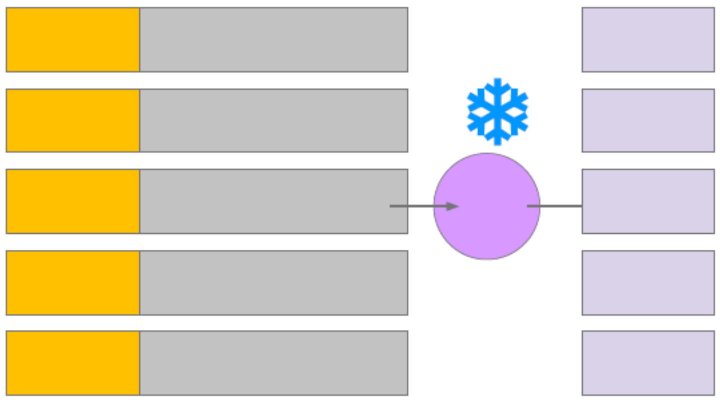
Fine-tuning - Soft prompts

Weights of model updated
during training



Millions to Billions of
parameter updated

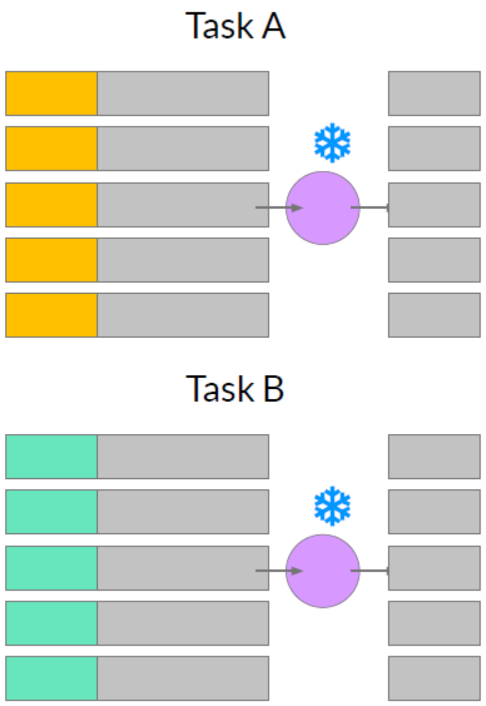
Weights of model frozen and
soft prompt trained



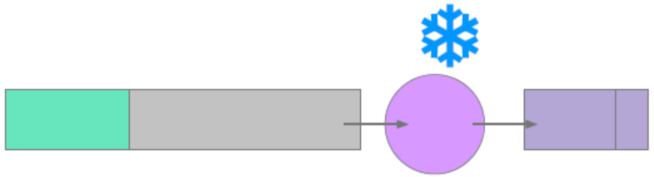
10K - 100K of parameters
updated

Fine-tuning - Soft prompts

- A different set of soft prompts for each task can be trained and then easily swap them out at inference time.



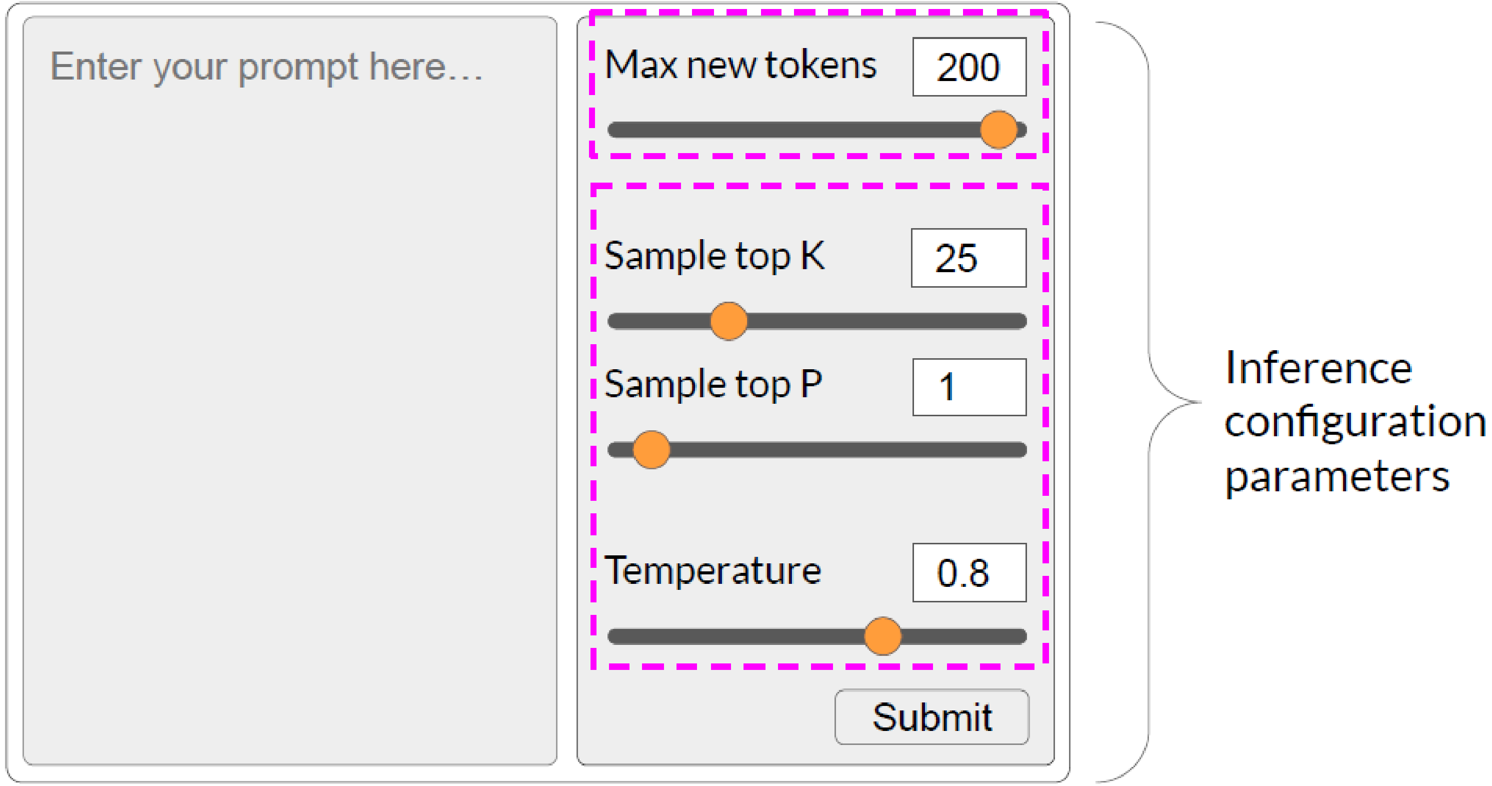
Switch out soft prompt at inference time to change task!



Configuration parameters for inference

- LLMs playgrounds are usually presented with some **controls** to adjust a set of **configuration parameters** to change how the LLM behaves.
- These set of configuration parameters influences the model's output during **inference**.
- Configuration parameters are invoked at inference time and give you control over things.
- Some parameters are the maximum number of tokens in the completion or others that influence in how creative the output is.

Configuration parameters for inference



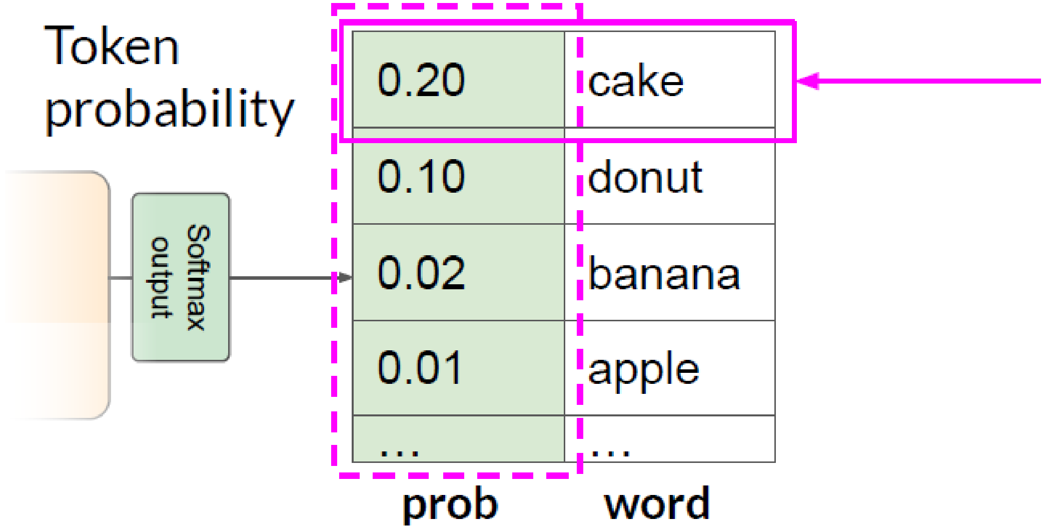
Configuration parameters for inference

- **Max new tokens** is used to **limit the number of tokens** that the model will generate.
- It is the **maximum new tokens**, not a hard number of new tokens generated (if a stop condition is reached, the model end the generation).

Configuration parameters for inference

- The **output** from the transformer's softmax layer is a **probability distribution** across the entire **dictionary of words** that the model uses.
- The **simplest** form of **next-word prediction**, where the model will always choose **the word with the highest probability**, is called **greedy sampling** or greedy decoding.
- This method can **work very well for short generation** but is **susceptible to repeated words** or repeated sequences of words.
- If it want to **generate text that's more natural**, more creative and avoids repeating words, it is necessary to **use other methods**.

Configuration parameters for inference

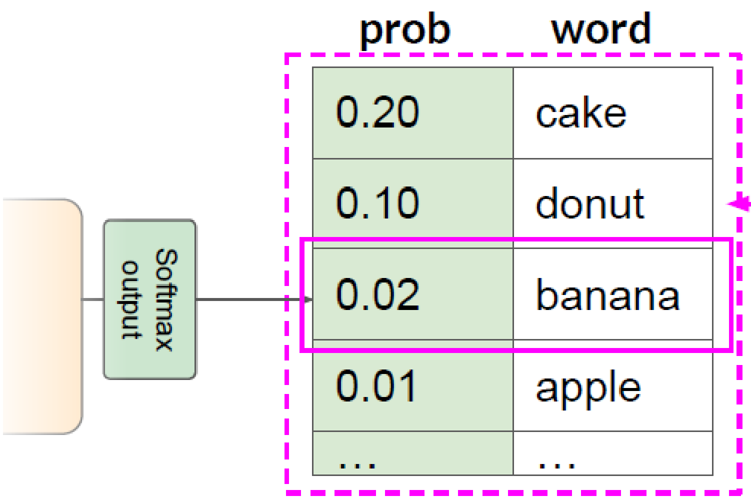


***greedy*:** The word/token with the highest probability is selected.

Configuration parameters for inference

- **Random sampling** is one way to **introduce some variability**.
- With random sampling, the model chooses an output word at random using the **probability distribution to weight the selection**.
- By using this technique, the **likelihood that words will be repeated is reduced**.

Configuration parameters for inference



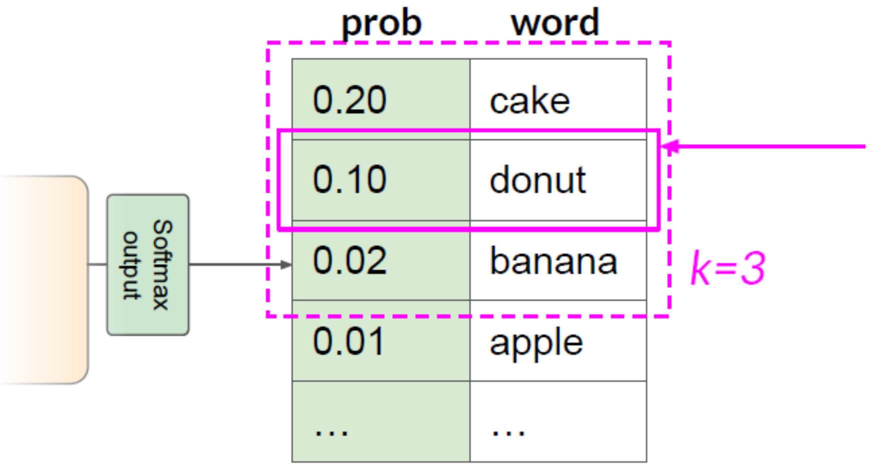
random(-weighted) sampling: select a token using a random-weighted strategy across the probabilities of all tokens.

Here, there is a 20% chance that 'cake' will be selected, but 'banana' was actually selected.

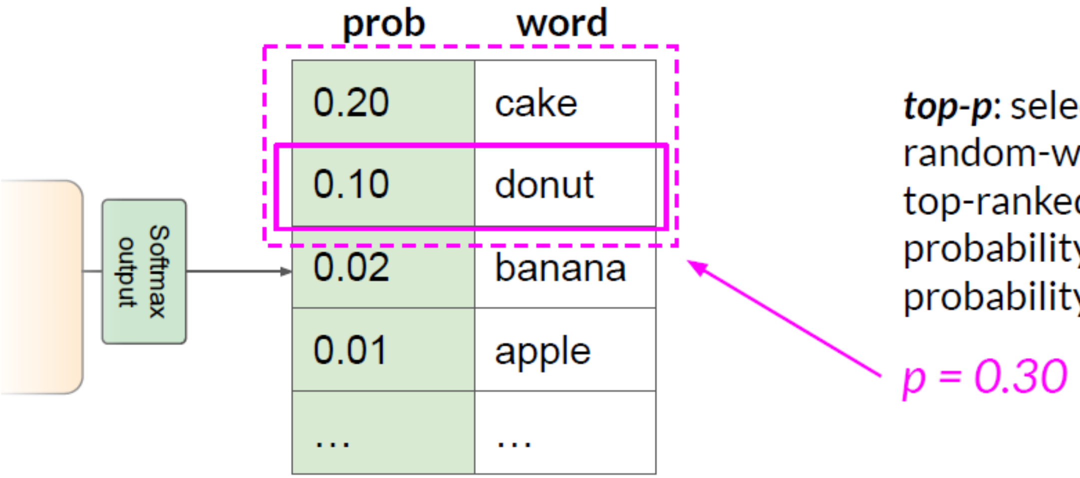
Configuration parameters for inference

- **Top K** and **Top P** are **sampling techniques** that can be used to **limit random sampling** and increase the probability of a **reasonable outcome**.
- **Top K** limits the options while allowing for some variability by specifying that the **model** chooses only among the **K tokens with the highest probability** using random sampling technique.
- A greedy decoding setting is equivalent to Top K=1.
- Alternatively, **Top P** limits random sampling to tokens whose **combined probabilities do not exceed P**.
- Top P is not used unless you set the Top P parameter value to something other than the default value of 1.

Configuration parameters for inference



top-k: select an output from the top-k results after applying random-weighted strategy using the probabilities



top-p: select an output using the random-weighted strategy with the top-ranked consecutive results by probability and with a cumulative probability <= *p*.

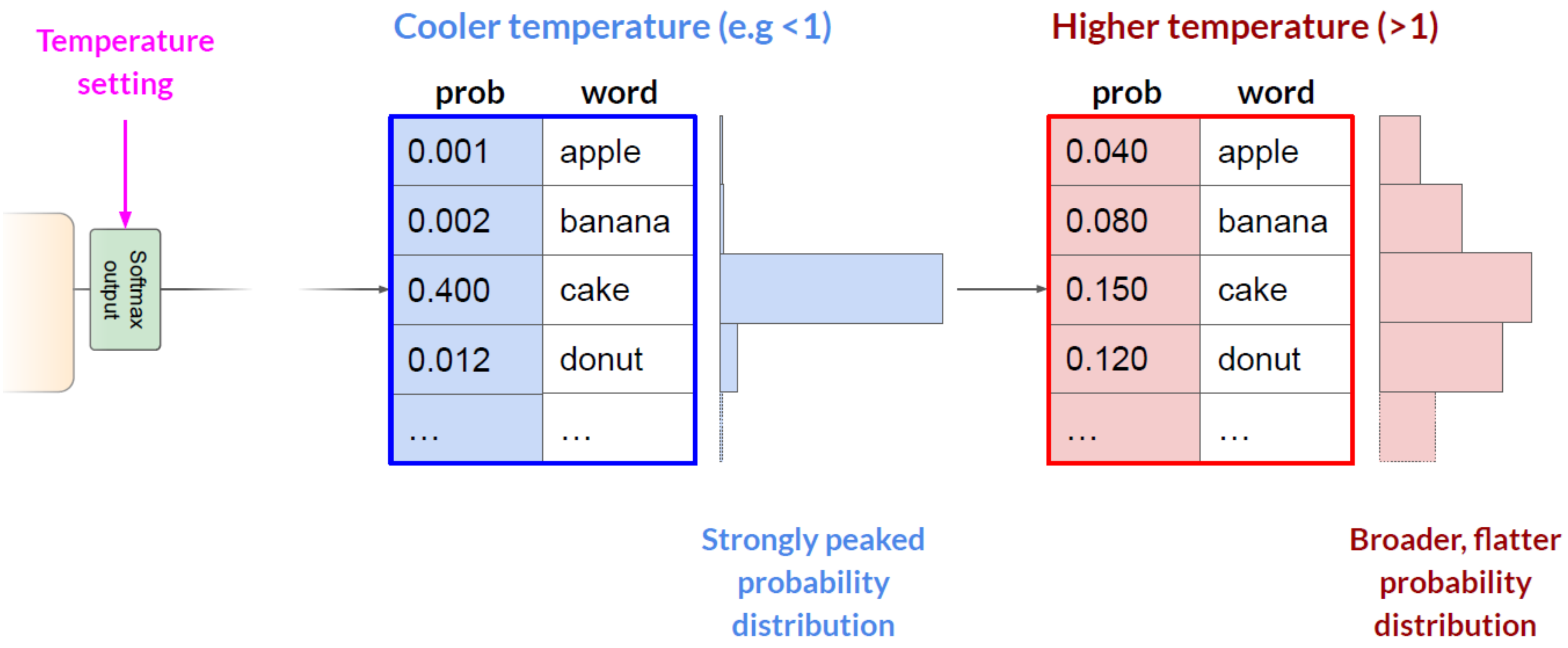
Configuration parameters for inference

- When both parameters (K and P) are specified, Top K is applied first, and any tokens below the cutoff that is set by Top K are considered to have a probability of zero when Top P is computed.

Configuration parameters for inference

- Other parameter that it is used to control the randomness of the model output is the **temperature**.
- This parameter **influences the shape of the probability distribution** of the softmax output layer of the word dictionary used by the model.
- In general, the **higher the temperature, the higher the randomness** and and the **lower the temperature, the lower the randomness**.
- The temperature value is a **scaling factor** that is applied to the **probability distribution of the softmax** output layer and affects the shape of the distribution.

Configuration parameters for inference



Resources

- https://www.coursera.org/learn/generative-ai-with-llms
- https://developers.google.com/machine-learning/resources/intro-llms
- https://www.analyticsvidhya.com/blog/2023/03/an-introduction-to-large-language-models-llms/
- https://stanford-cs324.github.io/winter2022/lectures
- https://docs.cohere.com/docs/intro-large-language-models
- https://towardsdatascience.com/a-practical-introduction-to-llms-65194dda1148
- https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/
- https://www.linkedin.com/pulse/transformer-architectures-dummies-part-2-decoder-only-qi6vc/
- https://www.baeldung.com/cs/large-language-models#general_architecture_of_llms
- https://www.ibm.com/docs/en/watsonx-as-a-service?topic=lab-model-parameters-prompting

Large Language Models (LLMs)

Luis Bote Curiel