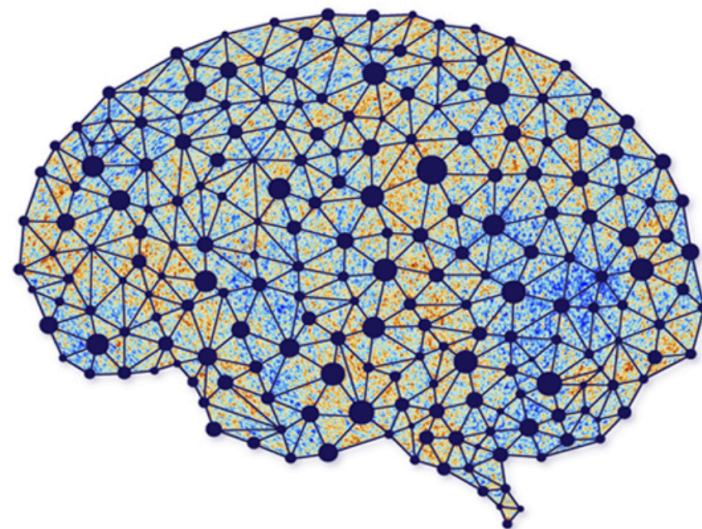


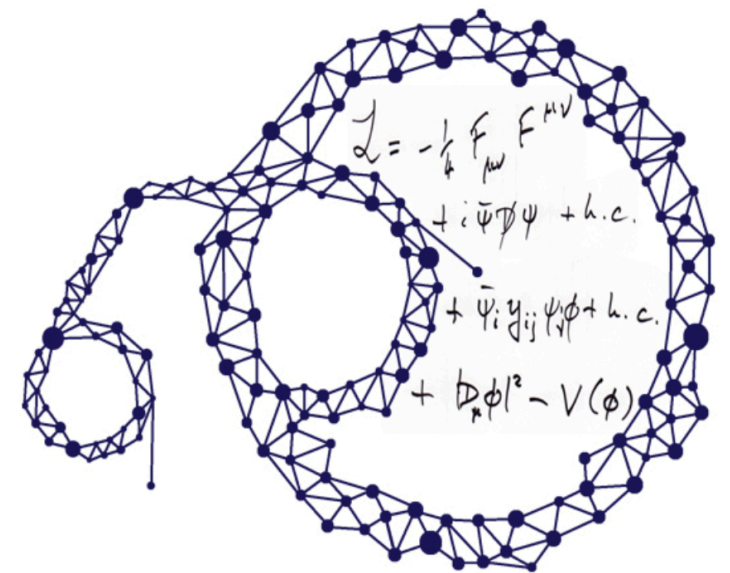
Physics 361 - Machine Learning in Physics

Lecture 16 – Large Language Models

March 13th 2025



AI
∩
Universe



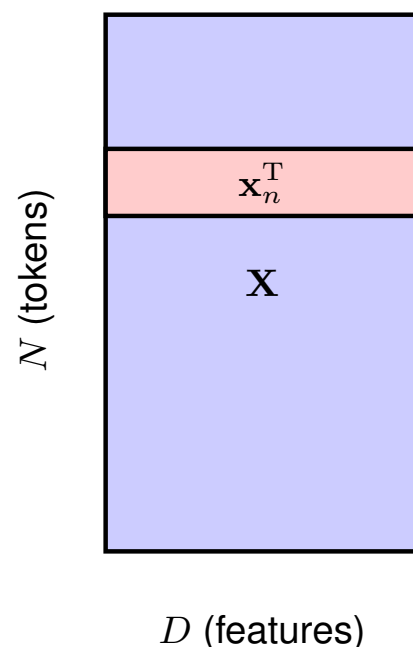
Moritz Münchmeyer

Transformers

Recall: Transformers

Transformer Processing

- Input data is a set of vectors $\{\mathbf{x}_n\}$ of dimensionality D , $n = 1, \dots, N$.
- These data vectors are known as **tokens** (e.g., a word within a sentence, a patch within an image, or an amino acid within a protein).
- The elements x_{ni} of the tokens are called **features**.
- Transformers can handle a mix of different data types by combining the data variables into a joint set of tokens.
- Combining the data vectors into a matrix X of dimensions $N \times D$.



$\tilde{\mathbf{X}} = \text{TransformerLayer}[\mathbf{X}]$

↑
same dimensionality as X

Apply multiple transformer layer
to learn rich internal representations.

Network Parameters

- Define query, key, & value matrices each w/ different transformations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{(q)}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^{(k)}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^{(v)}$$

the weight matrices $\mathbf{W}^{(q)}$, $\mathbf{W}^{(k)}$, $\mathbf{W}^{(v)}$ represent parameters that will be **learned** during the training of the transformer architecture.

- $\mathbf{W}^{(q)}$, $\mathbf{W}^{(k)}$, $\mathbf{W}^{(v)}$ are matrices of dim. $D \times D_k$, $D \times D_q$, $D \times D_v$. Setting $D_k = D_q$ allows for dot-products between query and key while $D_v = D$ allows multiple transformer layers to be stacked. We set $D_k = D_q = D_v = D$.
- Note: E.g. in GPT-3, the embedding dimension is divided among multiple attention heads but the overall dimensional consistency is maintained.

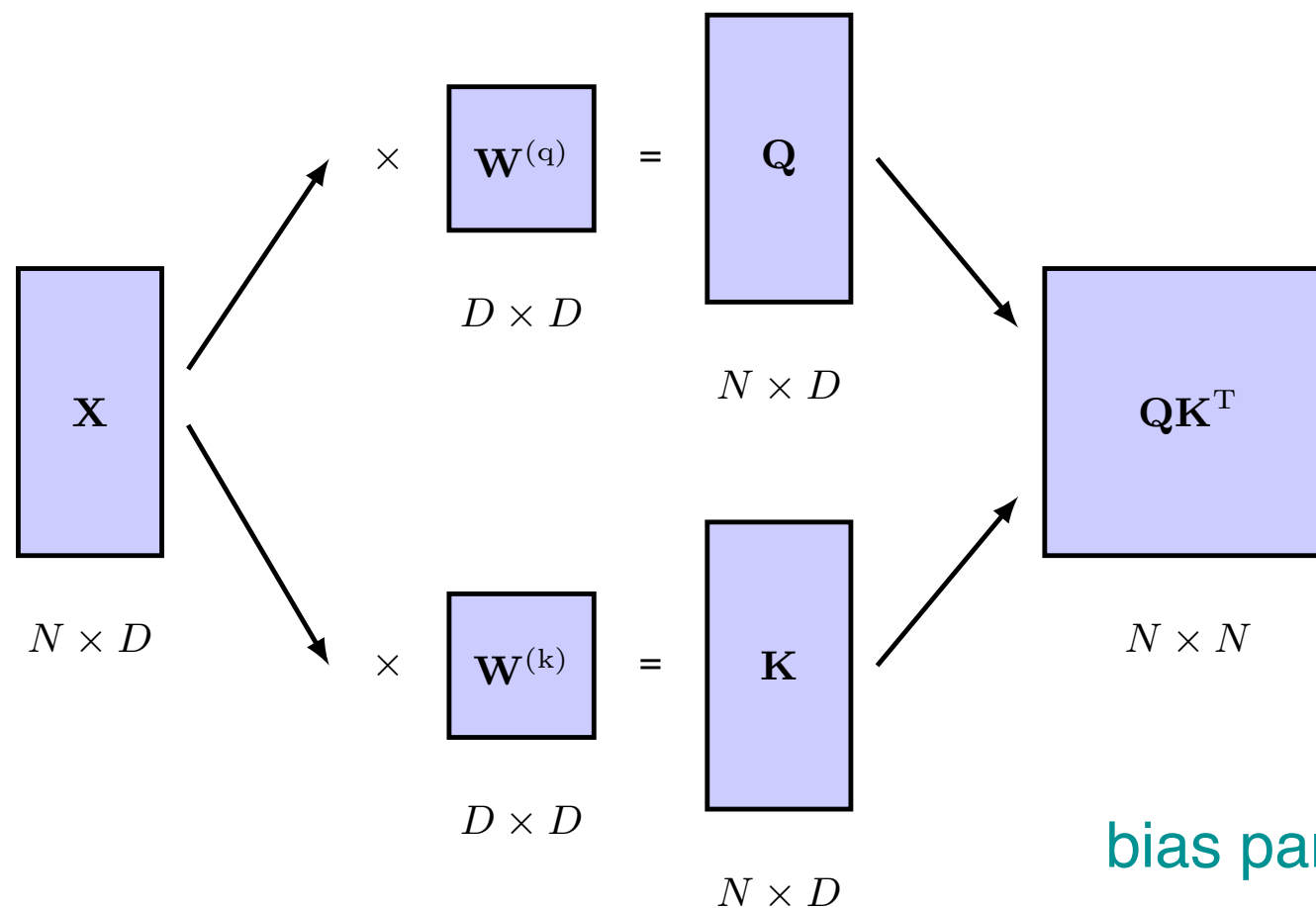
Network Parameters

- The transformation is now generalized to:

$$\mathbf{Y} = \text{Softmax} [\mathbf{QK}^T] \mathbf{V}$$

$N \times D_v$
 $N \times N$
 $N \times D_v$

whereas the dot-product can be computed by:



bias parameters are implicit

Algorithm 12.1: Scaled dot-product self-attention

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$

Weight matrices $\{\mathbf{W}^{(q)}, \mathbf{W}^{(k)}\} \in \mathbb{R}^{D \times D_k}$ and $\mathbf{W}^{(v)} \in \mathbb{R}^{D \times D_v}$

Output: Attention($\mathbf{Q}, \mathbf{K}, \mathbf{V}$) $\in \mathbb{R}^{N \times D_v} : \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$

$\mathbf{Q} = \mathbf{XW}^{(q)}$ // compute queries $\mathbf{Q} \in \mathbb{R}^{N \times D_k}$

$\mathbf{K} = \mathbf{XW}^{(k)}$ // compute keys $\mathbf{K} \in \mathbb{R}^{N \times D_k}$

$\mathbf{V} = \mathbf{XW}^{(v)}$ // compute values $\mathbf{V} \in \mathbb{R}^{N \times D_v}$

return Attention($\mathbf{Q}, \mathbf{K}, \mathbf{V}$) = $\text{Softmax}\left[\frac{\mathbf{QK}^T}{\sqrt{D_k}}\right] \mathbf{V}$

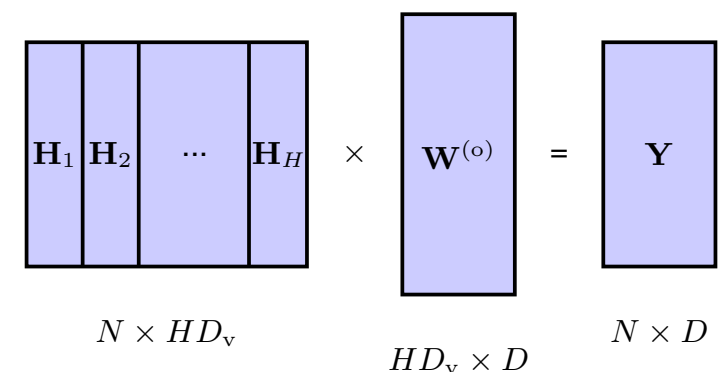
Multi-head attention

- There might be multiple patterns of attention relevant at the same time, e.g., some associated with tenses, some with vocabulary.
- Single “attention head” averages out these effects. Instead use multiple attention heads in parallel; analogous to channels in CNN.
- Suppose we have H heads indexed by $h = 1, \dots, H$:

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)$$

- The heads are concatenated into a single matrix, and the result is then linearly transformed to give a combined output:

$$\mathbf{Y}(\mathbf{X}) = \text{Concat} [\mathbf{H}_1, \dots, \mathbf{H}_H] \mathbf{W}^{(o)}$$



- The matrix $\mathbf{W}^{(o)}$ is learned along with the weight matrices $\mathbf{W}^{(q)}$, $\mathbf{W}^{(k)}$, $\mathbf{W}^{(v)}$.

Algorithm 12.2: Multi-head attention

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$
Query weight matrices $\{\mathbf{W}_1^{(q)}, \dots, \mathbf{W}_H^{(q)}\} \in \mathbb{R}^{D \times D}$
Key weight matrices $\{\mathbf{W}_1^{(k)}, \dots, \mathbf{W}_H^{(k)}\} \in \mathbb{R}^{D \times D}$
Value weight matrices $\{\mathbf{W}_1^{(v)}, \dots, \mathbf{W}_H^{(v)}\} \in \mathbb{R}^{D \times D_v}$
Output weight matrix $\mathbf{W}^{(o)} \in \mathbb{R}^{HD_v \times D}$

Output: $\mathbf{Y} \in \mathbb{R}^{N \times D} : \{\mathbf{y}_1, \dots, \mathbf{y}_N\}$

// compute self-attention for each head (Algorithm 12.1)

for $h = 1, \dots, H$ **do**

$\mathbf{Q}_h = \mathbf{XW}_h^{(q)}, \quad \mathbf{K}_h = \mathbf{XW}_h^{(k)}, \quad \mathbf{V}_h = \mathbf{XW}_h^{(v)}$

$\mathbf{H}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h)$ // $\mathbf{H}_h \in \mathbb{R}^{N \times D_v}$

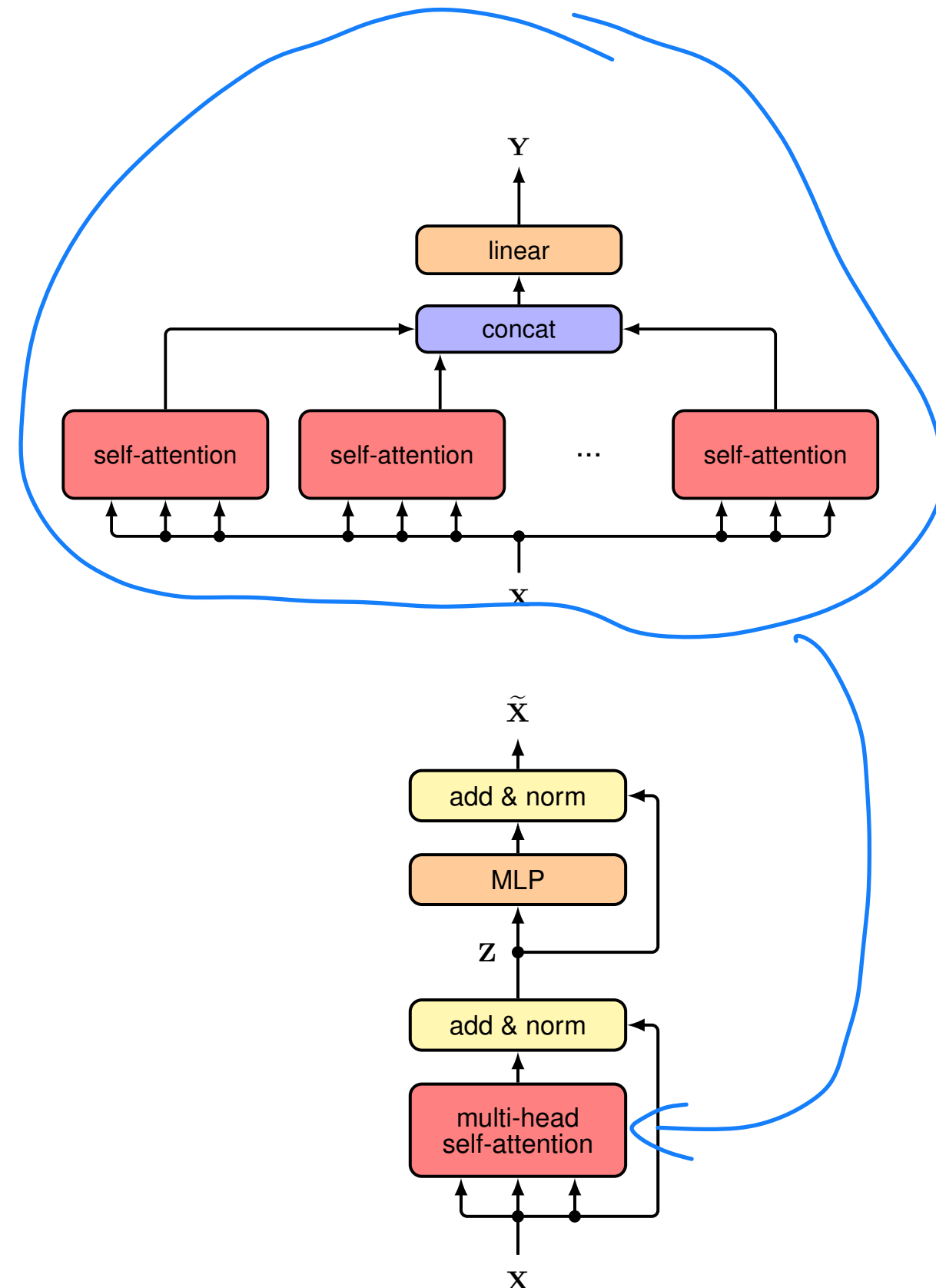
end for

$\mathbf{H} = \text{Concat}[\mathbf{H}_1, \dots, \mathbf{H}_N]$ // concatenate heads

return $\mathbf{Y}(\mathbf{X}) = \mathbf{HW}^{(o)}$

Transformer Layers

- NNs benefit greatly from depth, so we can stack self-attention layers (like the right) on top of each other.
- To improve efficiency, transformer layers are followed by **layer normalization**: <https://arxiv.org/abs/1607.06450>
- Output of an attention layer are constrained to be linear combinations of the inputs, though non-linearities enter through the attention weights.
- Enhance flexibility by post-processing the output of each layer using non-linear network denoted by MLP (same for each vector).



Algorithm 12.3: Transformer layer

Input: Set of tokens $\mathbf{X} \in \mathbb{R}^{N \times D} : \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$

Multi-head self-attention layer parameters

Feed-forward network parameters

Output: $\tilde{\mathbf{X}} \in \mathbb{R}^{N \times D} : \{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N\}$

$\mathbf{Z} = \text{LayerNorm} [\mathbf{Y}(\mathbf{X}) + \mathbf{X}]$ // $\mathbf{Y}(\mathbf{X})$ from Algorithm 12.2

$\tilde{\mathbf{X}} = \text{LayerNorm} [\text{MLP} [\mathbf{Z}] + \mathbf{Z}]$ // shared neural network

return $\tilde{\mathbf{X}}$

Transformers

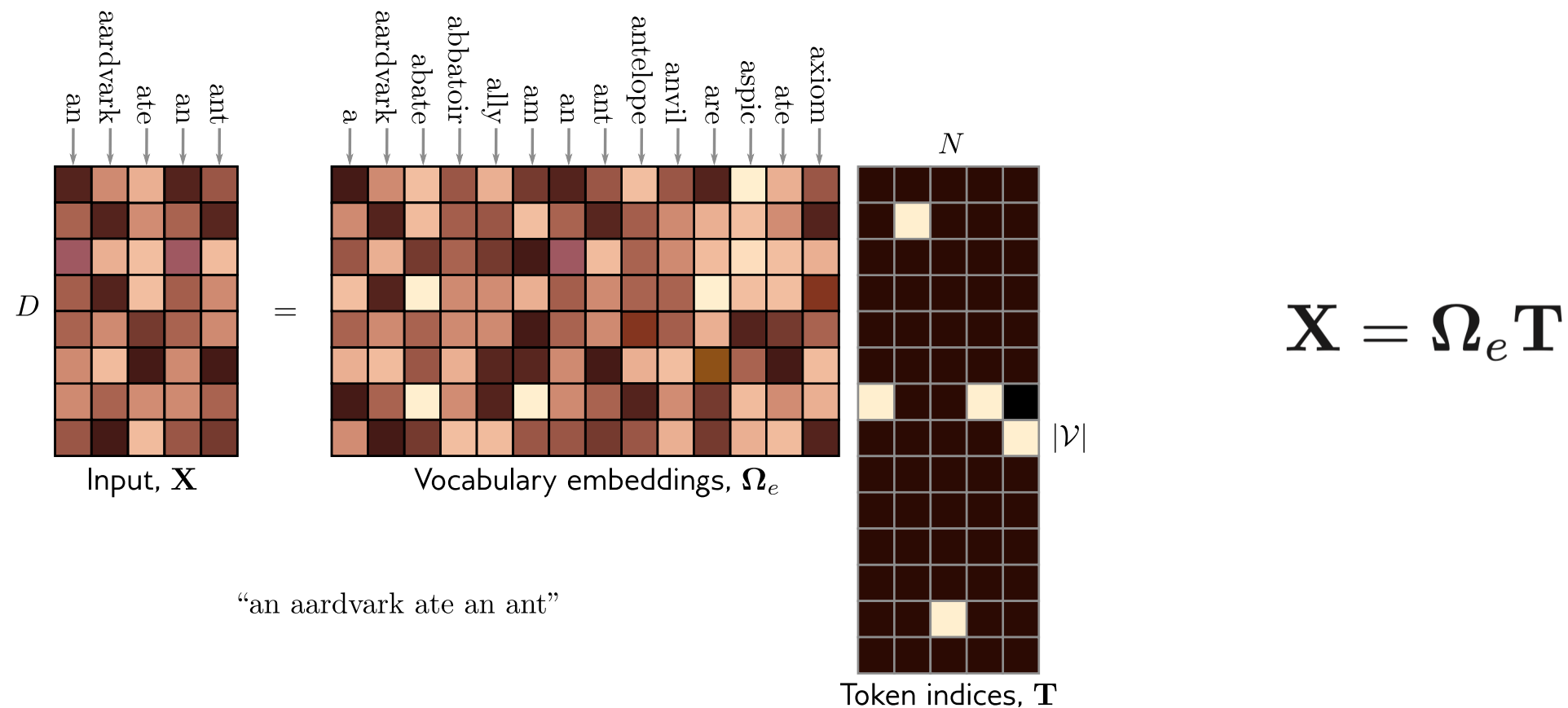
Large Language Models

Transformer for NLP

- A typical NLP pipeline starts with a **tokenizer** that splits the text into words or word fragments.
- Then each of the tokens is mapped to a learned **embedding**.
 - The whole vocabulary is stored in a matrix $\Omega_e \in \mathbb{R}^{D \times |\mathcal{V}|}$ where $|\mathcal{V}|$ is the vocabulary size; this vocabulary matrix is learned.
- These embeddings are passed through a series of **transformer layers**.

Learned Embeddings

- Each token is mapped to a unique word embedding; the embeddings for the whole vocabulary are stored in a matrix $\Omega_e \in \mathbb{R}^{D \times |\mathcal{V}|}$



- The matrix Ω_e can be learned like any other network parameter.
- A typical embedding size D is 1024 and a typical total vocabulary size $|\mathcal{V}|$ is 30,000. Many parameters in Ω_e to learn.

Transformer Encoders and Decoders

- The embedding matrix \mathbf{X} representing the text is passed through a series of K transformer layers, called a **transformer model**.
- Three types of transformer models:
 - An **encoder** transforms the text embeddings into a representation that can support a variety of tasks (e.g., sentiment analysis).
 - A **decoder** predicts the next token to continue the input text.
 - **Encoder-decoder** used in sequence-to-sequence tasks, where one text string is converted into another, e.g., machine translation.
- A hands-on tutorial on transformers in pytorch can be found here: <https://peterbloem.nl/blog/transformers>

Transformers

Large Language Models -
Encoders

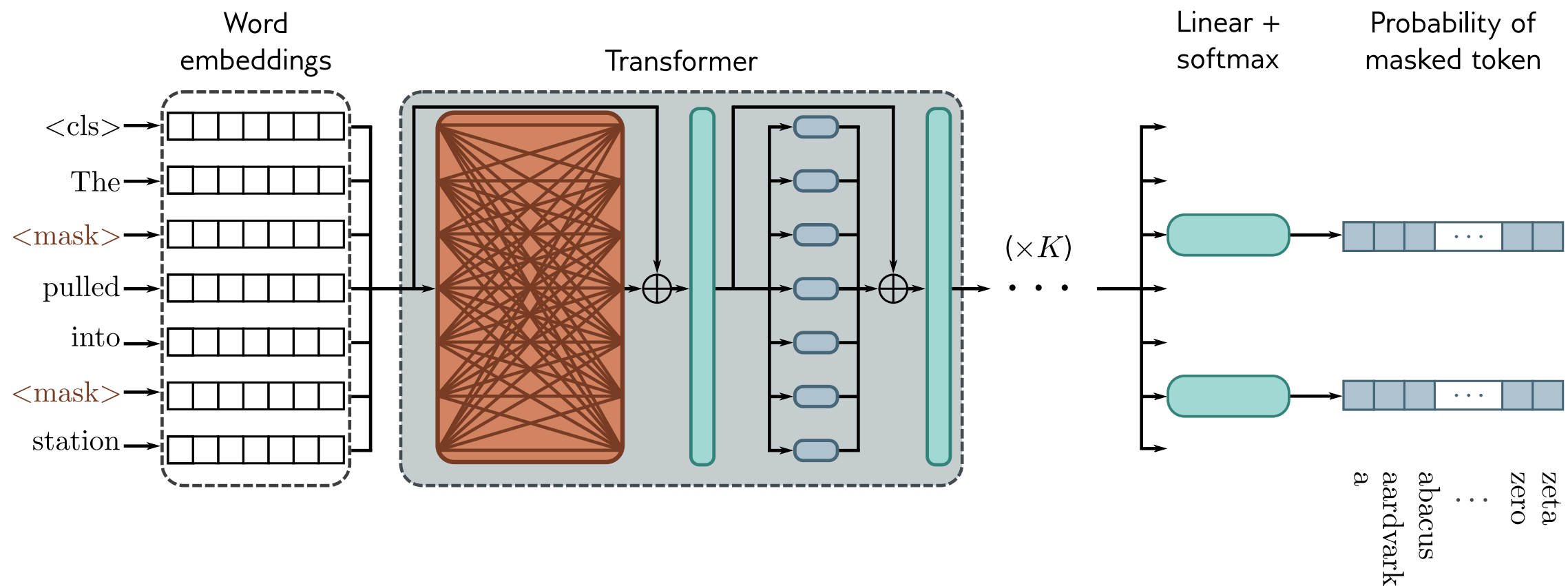
Encoder model example: BERT

<https://arxiv.org/abs/1810.04805v2>

- BERT is an encoder model that uses a vocabulary of 30,000 tokens.
- Input tokens are converted to 1024 dimensional word embeddings and passed through 24 transformer layers.
- Each contains a self-attention mechanism with 16 heads.
- The weight matrices Q_h, K_h, V_h for each head are 1024×64 .
- The total number of parameters is ~ 340 million, but it is now much smaller than state-of-the-art models.
- Encoder models like BERT exploit **transfer learning**: parameters of the ML model are learned during *pre-training* using *self-supervision* from a large corpus of data, followed by a *fine-tuning* stage to adapt for specific task using a smaller body of *supervised training data*.

Pre-training

- For BERT, the self-supervision task consists of predicting missing words from sentences from a large internet corpus.



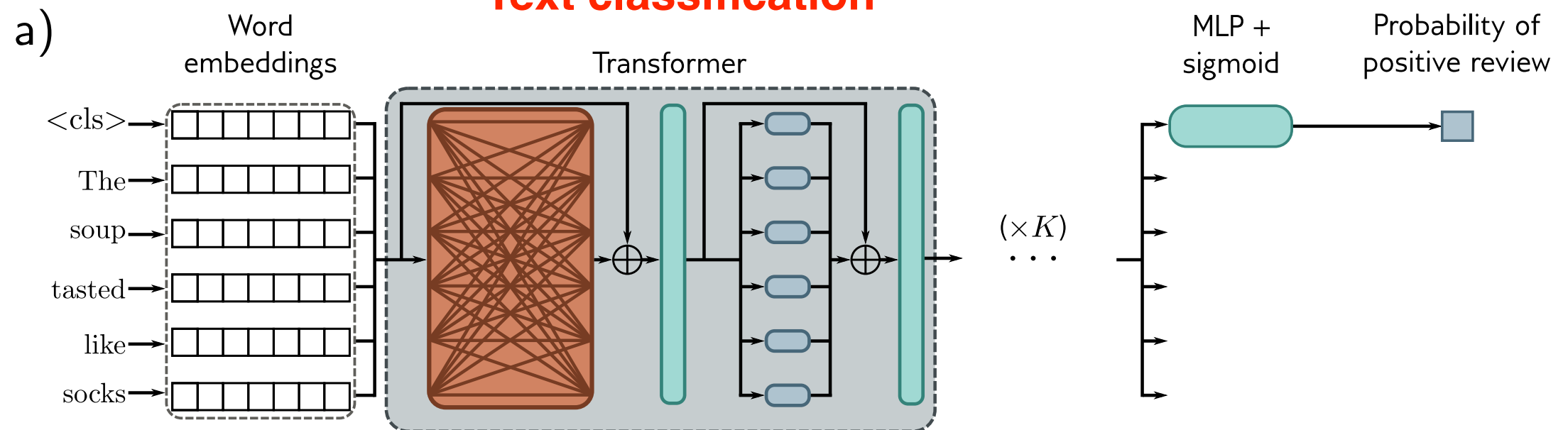
- Predicting missing words forces the transformer model to understand some syntax. For example, **red** is often found before **car** or **dress** than **swim**. In the above example, **train** is more likely than **lasagna**.

Fine-tuning

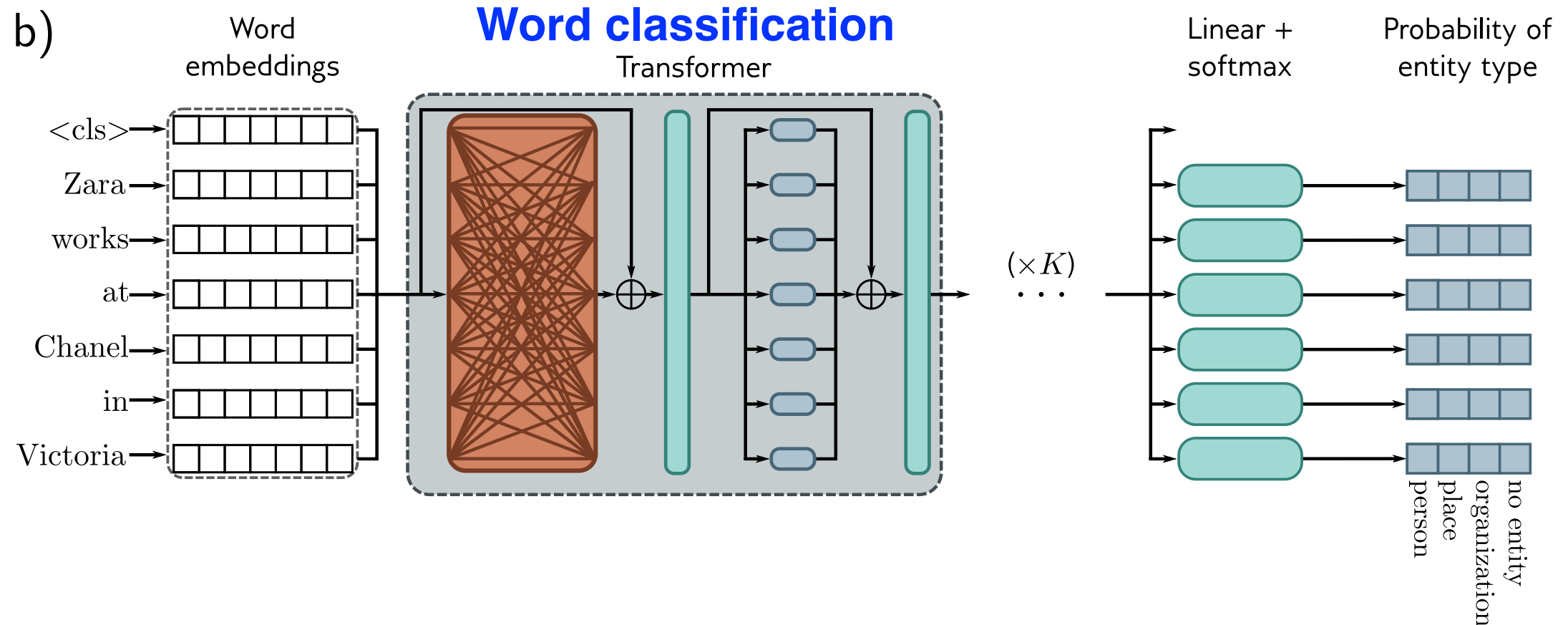
- In the fine-tuning stage, the model parameters are adjusted to specialize the network to a particular task.
- An extra layer is appended onto the transformer network to convert the output vectors to the desired output format.
- Specific tasks include:
 - **Text classification**: <cls> token is added to the start of each string during pre-training. sentiment analysis, the vector associated with <cls> is mapped to a number & passed through a logistic sigmoid.
 - **Word classification**: e.g., to classify a word into entity types (person, place, organization, or no-entry). Input is mapped to a $E \times 1$ vector where E = entry types, then Softmax for probabilities.

Fine-tuning

Text classification



Word classification



Transformers

Large Language Models -
Decoders

Autoregressive text generation

- The basic architecture is similar to the encoder model & comprises a series of transformer layers that operate on learned word embeddings.
- Different goal: to generate the next token in a sequence (and generate a coherent text passage by feeding the sequence back into the model).
- **Autoregressive language model**: factors the joint probability of a sequence of observed tokens into an autoregressive sequence.
- Consider e.g.: “It takes great courage to let yourself appear weak.”

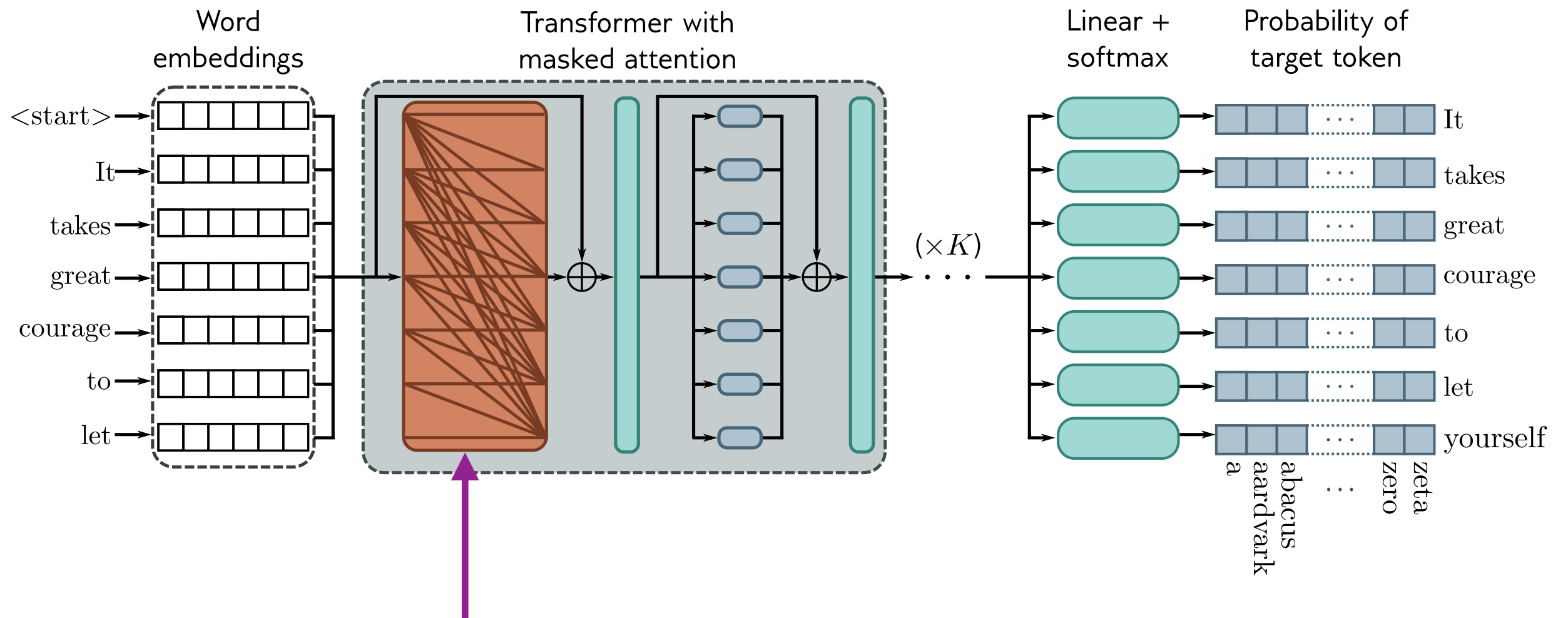
$$\begin{aligned} Pr(\text{It takes great courage to let yourself appear weak}) &= \\ &Pr(\text{It}) \times Pr(\text{takes}|\text{It}) \times Pr(\text{great}|\text{It takes}) \times Pr(\text{courage}|\text{It takes great}) \times \\ &Pr(\text{to}|\text{It takes great courage}) \times Pr(\text{let}|\text{It takes great courage to}) \times \\ &Pr(\text{yourself}|\text{It takes great courage to let}) \times \\ &Pr(\text{appear}|\text{It takes great courage to let yourself}) \times \\ &Pr(\text{weak}|\text{It takes great courage to let yourself appear}). \end{aligned}$$

Generally:
$$Pr(t_1, t_2, \dots, t_N) = Pr(t_1) \prod_{n=2}^N Pr(t_n | t_1, \dots, t_{n-1}).$$

Decoder model example: GPT3

- To train a decoder, we maximize the log probability of the input text under the autoregressive model defined above.
- This poses a problem: if we pass the full sentence, the term computing $\log | Pr(\text{great} | \text{It takes})$ has access to the rest of the sentence.
- The system can cheat rather than learn to predict, and thus will not train properly.
- **Masked self-attention**: setting the dot products with future tokens in the self-attention computation to $-\infty$ before passing through softmax.
- The transformer layers use masked self-attention so that only attention to the current and previous tokens are allowed.
- During training, we aim to maximize the sum of the log probabilities of the next token using a standard multiclass cross-entropy loss.

Masked self-attention



attend only to the current and previous tokens

Sampling: Generating text from a decoder

- The autoregressive language model is a **generative model**.
- Start with an input sequence of text, beginning with a <start> token.
- The outputs are the probabilities over possible subsequent tokens. **We can either pick the most likely token or sample from this probability distribution.**
- The new extended sequence can be fed back into the decoder network that outputs the probability distribution over the next token.
- At each step, **the decoder takes the entire sequence generated so far (including all previous tokens) as input and produces a probability distribution for the next token.** Then, you typically select one token from that distribution and append it to the sequence. This updated sequence, now containing the newly generated token, is then fed back into the decoder for the next prediction.
- Other strategies (instead of greedy search): **beam search and top-k sampling**, etc.

Transformers

Large Language Models -
Encoder-Decoder
Transformer (briefly)

Encoder-decoder model example: machine translation

- Translation between languages is a **sequence-to-sequence** task.
- **An encoder computes a good representation of the source sentence. A decoder generates the sentence in the target language.**
- Consider a **encoder-decoder model for English-French** translation.
 - The encoder receives the sentence in English and process it through a series of transformer layers to create an output rep. for each token.
 - During training, the decoder receives the ground truth translation in French and passes it through a series of transformer layers that use masked self-attention and predict the following word at each position.
 - However, the decoder layers also attend to the output of the encoder. Each French output word is conditioned on the previous output words and the source English sentence.
- This is the **original setup which invented the transformer.**



Attention is All you Need

by A Vaswani · Cited by 171064 — We propose a new simple network architecture, the Transformer, based solely on **attention** mechanisms, dispensing with recurrence and convolutions entirely.

11 pages

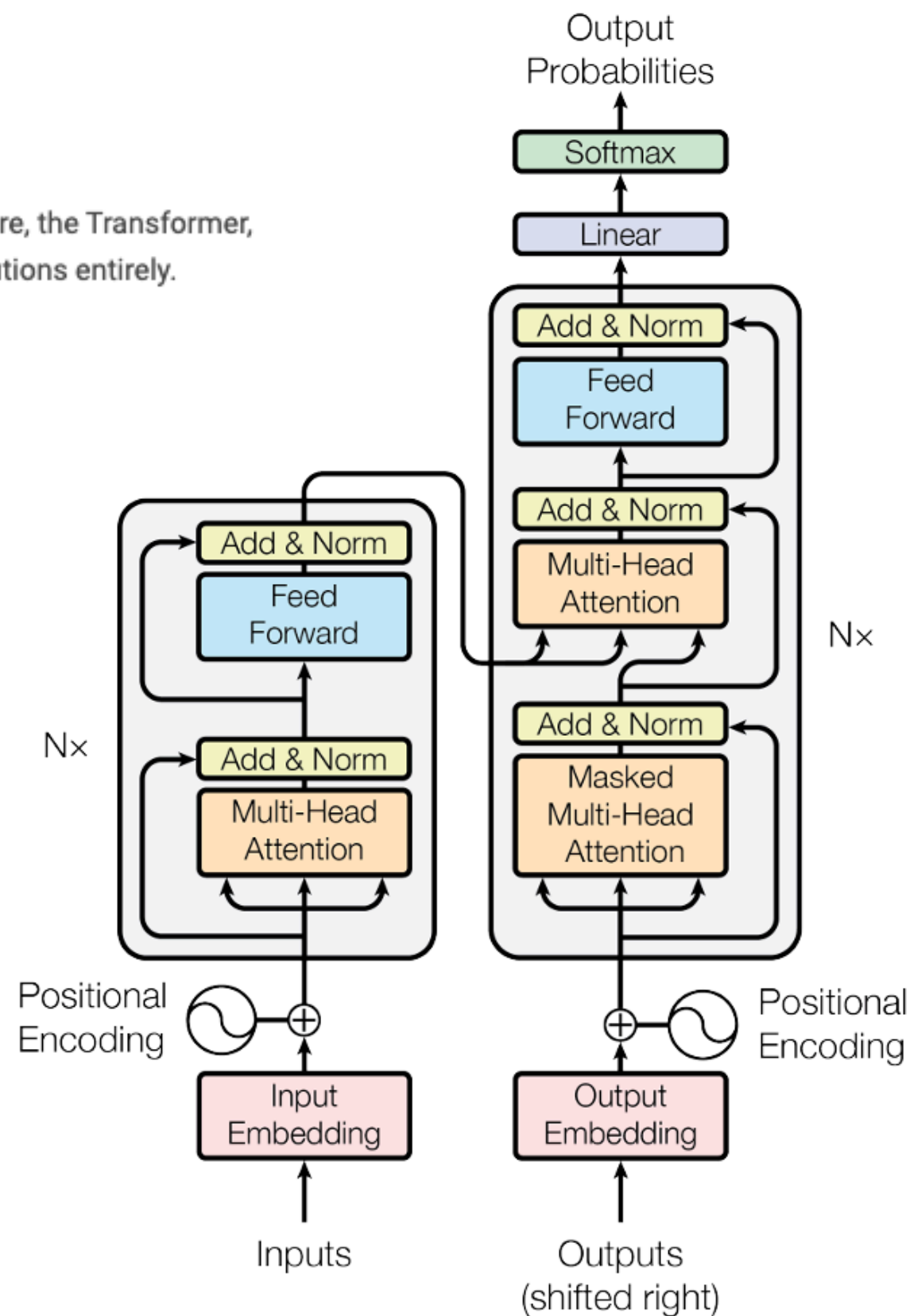


Figure 1: The Transformer - model architecture.

Transformers

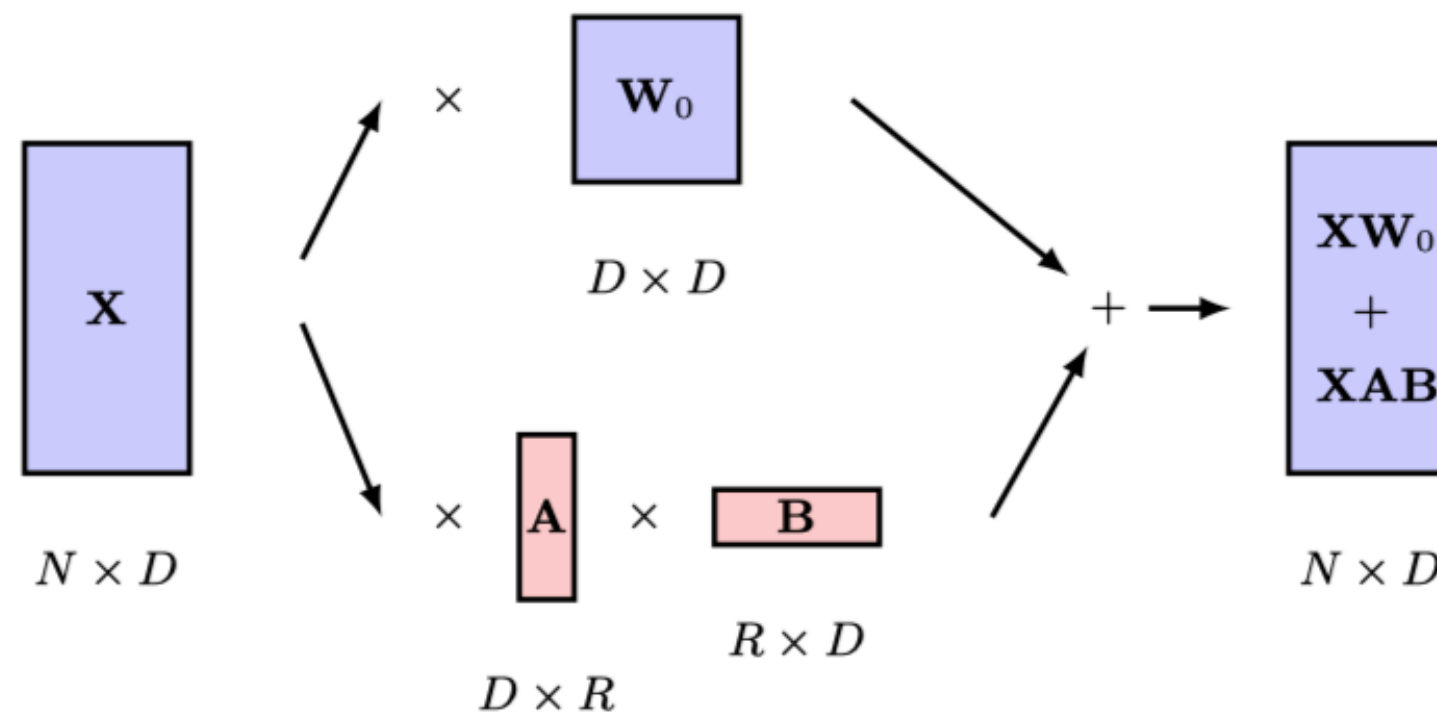
Large Language Models -
Fine Tuning

Fine Tuning Foundation Models

- A pre-trained LLM (or large transformer) is also called a “**Foundation Model**”.
- We can then use supervised fine tuning for specific applications, often called “**downstream tasks**”.
- The fine-tuning **can be done by adding extra layers to the outputs of the network or by replacing the last few layers** with fresh parameters and then using the labelled data to train these final layers.
- During the fine-tuning stage, **the weights and biases in the main model can either be left unchanged or be allowed to undergo small levels of adaptation**. Typically the cost of the fine-tuning is small compared to that of pretraining.

Fine Tuning with LoRA

- One very efficient approach to fine-tuning is called **low-rank adaptation or LoRA** (Hu et al., 2021). This approach is inspired by results which show that a trained overparameterized model has a low intrinsic dimensionality with respect to fine-tuning.
- LoRa exploits this by freezing the weights of the original model and adding additional learnable weight matrices into each layer of the transformer in the form of low-rank products.



Schematic illustration low-rank adaptation showing a weight matrix W_0 from one of the attention layers in a pre-trained transformer. Additional weights given by matrices A and B are adapted during fine-tuning and their product AB is then added to the original matrix for subsequent inference.

Fine Tuning and RLHF

- After training a decoder model model will “babble” text, trying to complete sequences. e.g. if you give it a question it might follow up with more questions.
- Chat Bots are then fine tuned in several steps to make them more useful:
- <https://openai.com/index/chatgpt/>
 - Step 1: Fine tuning
 - They have thousands of question and answer pairs in a curated data set
 - Step 2: Humans rank different answers (Reinforcement Learning with Human Feedback RLHF). Train a reward model.
 - Step 3: Use Reinforcement Learning using this reward model.
- Roughly, this brings the model **from a “document completer” to a “question answerer”**

Step 1

Collect demonstration data and train a supervised policy.

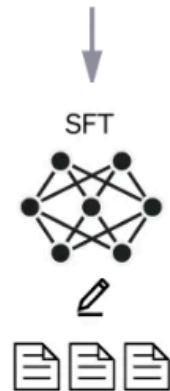
A prompt is sample from our prompt dataset.



A labeler demonstrates the desired output behavior.



This data is used to fine-tune GPT-3.5 with supervised learning.



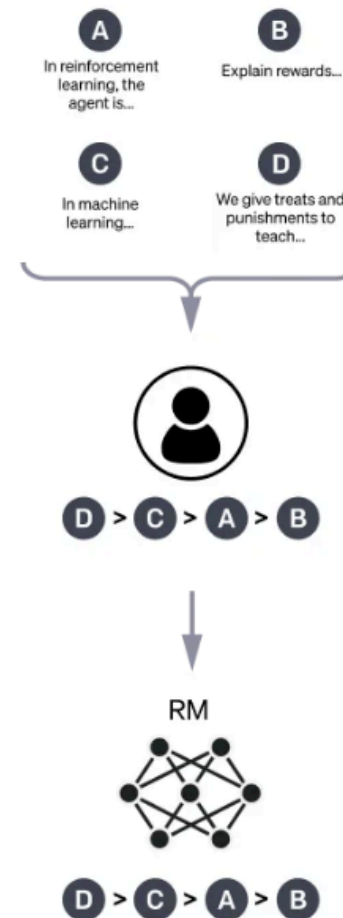
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

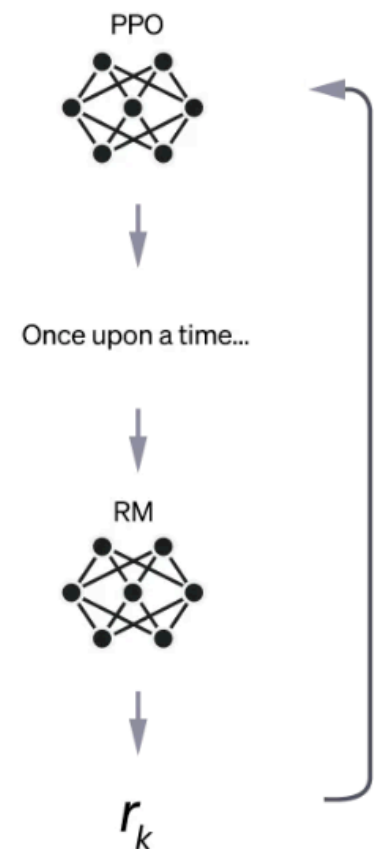


The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



We will talk more about Reinforcement Learning later.

Transformers

Using LLMs

LLM Model Zoo

- <https://lmarena.ai/>

🏆 Chatbot Arena LLM Leaderboard: Community-driven Evaluation for Best LLM and AI chatbots

[Discord](#) | [Twitter](#) | [小红书](#) | [Blog](#) | [GitHub](#) | [Paper](#) | [Dataset](#) | [Kaggle Competition](#)

Chatbot Arena is an open platform for crowdsourced AI benchmarking, developed by researchers at UC Berkeley [SkyLab](#) and [LMArena](#). With over 1,000,000 user votes, the platform ranks best LLM and AI chatbots using the Bradley-Terry model to generate live leaderboards. For technical details, check out our [paper](#).

Chatbot Arena thrives on community engagement — cast your vote to help improve AI evaluation!

New Launch! WebDev Arena: [web.lmarena.ai](#) - AI Battle to build the best website!

[Language](#) [Overview](#) [Price Analysis](#) [WebDev Arena](#) [Vision](#) [Text-to-Image](#) [Copilot Arena Leaderboard](#) [Arena-Hard-Auto](#)

Total #models: 212. Total #votes: 2,768,389. Last updated: 2025-03-10.

Code to recreate leaderboard tables and plots in this [notebook](#). You can contribute your vote at [lmarena.ai](#)!

Category

Overall

Apply filter

☐ Style Control

☐ Show Depreciated

Overall Questions

#models: 212 (100%) #votes: 2,768,389 (100%)

Rank★ (UB)	▲	Rank (StyleCtrl)	▲	Model	▲	Arena Score	▲	95% CI	▲	Votes	▲	Organization	▲	License	▲
1		2		Grok-3-Preview-02-24		1407		+7/-7		7580		xAI		Proprietary	
1		1		GPT-4.5-Preview		1404		+7/-9		6024		OpenAI		Proprietary	
3		6		Gemini-2.0-Flash-Thinking-Exp-01-21		1384		+5/-5		19837		Google		Proprietary	
3		3		Gemini-2.0-Pro-Exp-02-05		1380		+4/-4		17695		Google		Proprietary	
3		2		ChatGPT-4o-latest (2025-01-29)		1375		+4/-5		19587		OpenAI		Proprietary	
6		4		DeepSeek-R1		1361		+5/-6		10474		DeepSeek		MIT	
6		10		Gemini-2.0-Flash-001		1355		+4/-5		15416		Google		Proprietary	
6		3		o1-2024-12-17		1353		+4/-4		22010		OpenAI		Proprietary	
9		10		Gemma-3-27B-it		1339		+9/-11		3870		Google		Gemma	
9		10		Qwen2.5-Max		1338		+5/-5		14258		Alibaba		Proprietary	
9		7		o1-preview		1335		+4/-4		33195		OpenAI		Proprietary	
9		10		o3-mini-high		1328		+6/-5		11409		OpenAI		Proprietary	
13		12		DeepSeek-V3		1319		+4/-4		23079		DeepSeek		DeepSeek	

Running them online

- Example: <https://www.together.ai/>

The screenshot displays the Together AI playground interface. At the top, the navigation bar includes links for DASHBOARD, PLAYGROUNDS (selected), GPU CLUSTERS, MODELS, JOBS, ANALYTICS, and DOCS. A user profile icon is visible on the right. A blue banner below the navigation bar contains a warning: "AI models may provide inaccurate information. Verify important details." The main chat area shows a conversation with the model "meta-llama/Llama-3.3-70B-Instruct-Turbo". The chat input field has tabs for "UI" and "API", and a "What is LoRA?" button. The chat content includes a definition of LoRA (Low-Rank Adaptation) and its benefits. The right sidebar contains a "MODEL" dropdown set to "Meta Llama 3.3 70B Instruct Turbo" and a "PARAMETERS" section with various settings: System Prompt (Default), Auto-set output length (checked), Output Length (2048), Temperature (0.7), Top-P (0.7), Top-K (50), Repetition Penalty (1), and Safety Models (None). At the bottom of the chat area, there are icons for share, copy, and a token count: "13 IN | 387 OUT | 145.38 TOKENS/S".

together.ai

DASHBOARD PLAYGROUNDS GPU CLUSTERS MODELS JOBS ANALYTICS DOCS

AI models may provide inaccurate information. Verify important details.

CHAT meta-llama/Llama-3.3-70B-Instruct-Turbo

UI API

What is LoRA?

LoRA (Low-Rank Adaptation) is a technique used in artificial intelligence, specifically in the field of natural language processing (NLP) and computer vision. It's a method for fine-tuning pre-trained models, such as language models or image classification models, to adapt to new tasks or datasets.

The main idea behind LoRA is to update only a small subset of the model's parameters, rather than retraining the entire model from scratch. This is done by adding low-rank matrices to the original model's weights, which allows the model to adapt to new information without requiring significant changes to the underlying architecture.

LoRA has several benefits, including:

1. **Efficient adaptation:** LoRA allows models to adapt to new tasks or datasets with minimal additional training data and computational resources.
2. **Preservation of pre-trained knowledge:** By updating only a small subset of the model's parameters, LoRA preserves the knowledge and representations learned during pre-training.
3. **Flexibility:** LoRA can be applied to a wide range of models and tasks, making it a versatile technique for transfer learning and domain adaptation.

LoRA has been successfully applied in various areas, including:

1. **Language modeling:** LoRA has been used to adapt language models to new languages, domains, or tasks, such as sentiment analysis or question answering.
2. **Computer vision:** LoRA has been used to adapt image classification models to new datasets, such as adapting a model trained on ImageNet to a new dataset with different classes or distributions.
3. **Multimodal learning:** LoRA has been used to adapt models to new modalities, such as adapting a model trained on text data to work with image or audio data.

Overall, LoRA is a powerful technique for adapting pre-trained models to new tasks and datasets, and has the potential to improve the efficiency and effectiveness of AI systems in a wide range of applications.

13 IN | 387 OUT | 145.38 TOKENS/S

MODEL

Meta Llama 3.3 70B Instruct Turbo

PARAMETERS

System Prompt

Default

☒ Auto-set output length

Output Length 2048

Temperature 0.7

Top-P 0.7

Top-K 50

Repetition Penalty 1

Safety Models

None

Running them locally

- Large Library of models and data sets: <https://huggingface.co/>

The screenshot displays the Hugging Face homepage. At the top, there's a navigation bar with the Hugging Face logo, a search bar, and links to Models, Datasets, Spaces, Posts, Docs, Enterprise, and Pricing. A yellow banner below the navigation bar promotes joining an organization. The main content area is divided into a left sidebar and a central grid of model cards.

Left Sidebar:

- Tasks:** Libraries, Datasets, Languages, Licenses, Other. A search bar "Filter Tasks by name" is present.
- Multimodal:** Audio-Text-to-Text, Image-Text-to-Text, Visual Question Answering, Document Question Answering, Video-Text-to-Text, Visual Document Retrieval, Any-to-Any.
- Computer Vision:** Depth Estimation, Image Classification, Object Detection, Image Segmentation, Text-to-Image, Image-to-Text, Image-to-Image, Image-to-Video, Unconditional Image Generation, Video Classification, Text-to-Video, Zero-Shot Image Classification, Mask Generation, Zero-Shot Object Detection, Text-to-3D, Image-to-3D, Image Feature Extraction, Keypoint Detection.
- Natural Language Processing:** Text Classification, Token Classification, Table Question Answering, Question Answering, Zero-Shot Classification, Translation, Summarization, Feature Extraction, Text Generation, Text2Text Generation, Fill-Mask, Sentence Similarity.
- Audio:** Text-to-Speech, Text-to-Audio, Automatic Speech Recognition, Audio-to-Audio, Audio Classification, Voice Activity Detection.
- Tabular:** Tabular Classification, Tabular Regression, Time Series Forecasting.

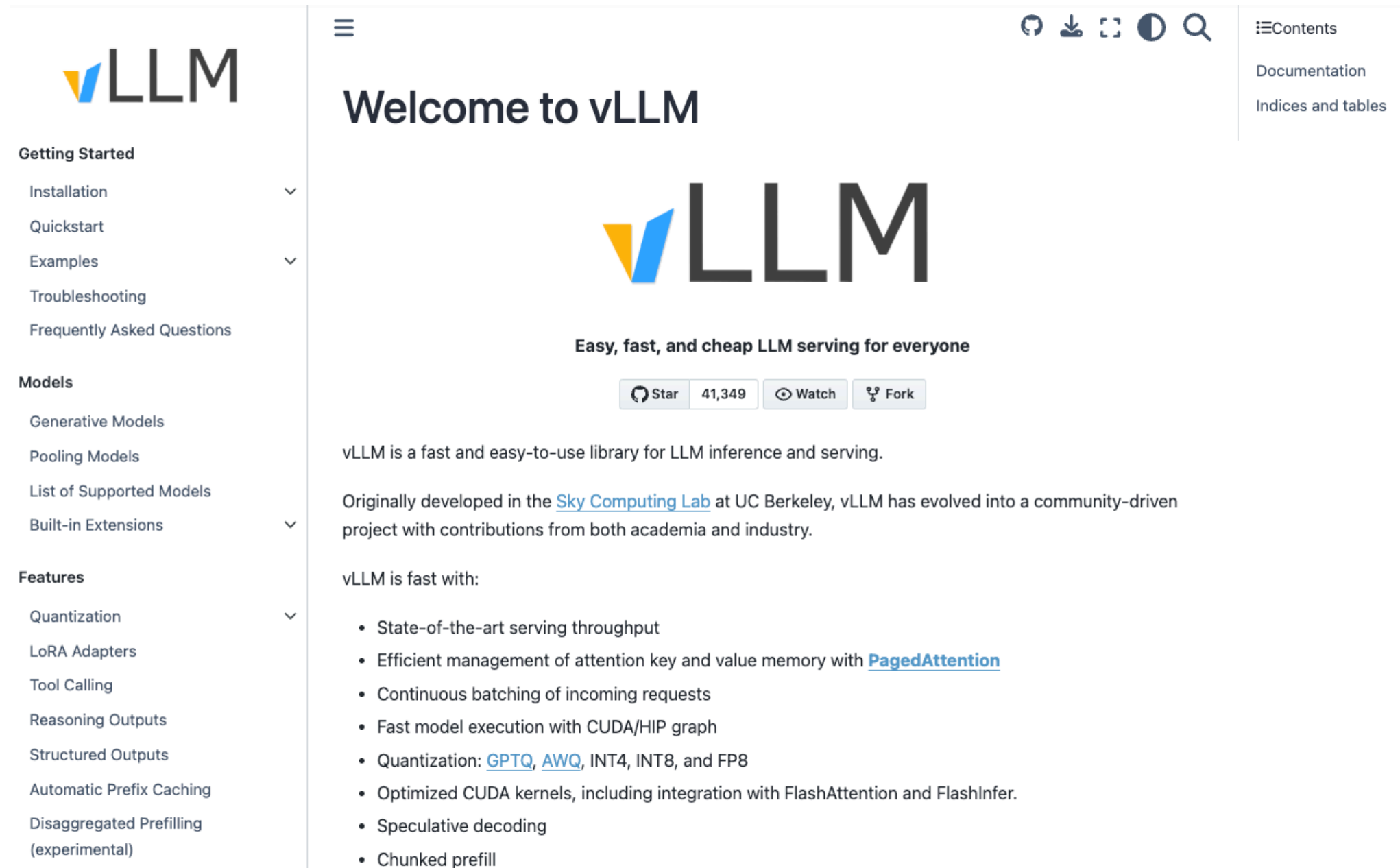
Central Grid:

The grid shows a list of models with filters and sorting options. The "Models" filter is set to "1,500,036". The sorting is set to "Trending".

Model Name	Task	Updated	Downloads	Stars	Hearts
Qwen/QwQ-32B	Text Generation	Updated 2 days ago	256k	2.11k	
google/gemma-3-27b-it	Image-Text-to-Text	Updated 1 day ago	38.5k	451	
deepseek-ai/DeepSeek-R1	Text Generation	Updated 18 days ago	2.75M	11.3k	
RekaAI/reka-flash-3		Updated about 4 hours ago	1.3k	220	
tencent/HunyuanVideo-I2V	Image-to-Video	Updated about 8 hours ago	2.1k	244	
google/gemma-3-12b-it	Image-Text-to-Text	Updated 1 day ago	7.94k	140	
black-forest-labs/FLUX.1-dev	Text-to-Image	Updated Aug 16, 2024	2.73M	9.32k	
google/gemma-3-1b-it	Text Generation	Updated 1 day ago	9.78k	116	
perplexity-ai/r1-1776	Text Generation	Updated 15 days ago	55k	2.12k	
CohereForAI/aya-vision-8b	Image-Text-to-Text	Updated 9 days ago	147k	250	
open-r1/OlympicCoder-7B	Text Generation	Updated about 3 hours ago	641	76	
lodestones/Chroma	Text-to-Image	Updated about 13 hours ago		133	
stabilityai/stable-diffusion-3.5-large	Text-to-Image	Updated Oct 22, 2024	160k	2.47k	
GSAI-ML/LLaDA-8B-Instruct	Text Generation	Updated 15 days ago	26.3k	211	
SparkAudio/Spark-TTS-0.5B	Text-to-Speech	Updated 7 days ago	7.82k	378	
microsoft/Phi-4-multimodal-instruct	Automatic Speech Recognition	Updated about 18 hours ago	441k	1.12k	
Wan-AI/Wan2.1-T2V-14B	Text-to-Video	Updated 1 day ago	207k	1.01k	
google/gemma-3-4b-it	Image-Text-to-Text	Updated 1 day ago	12.6k	136	
CohereForAI/c4ai-command-a-03-2025	Text Generation	Updated about 1 hour ago		128	
allenai/olmoOCR-7B-0225-preview	Image-Text-to-Text	Updated 17 days ago	193k	541	
hexgrad/Kokoro-82M	Text-to-Speech	Updated 10 days ago	1.58M	3.65k	
Qwen/QwQ-32B-GGUF	Text Generation	Updated about 8 hours ago	99.2k	132	
Comfy-Org/Wan_2.1_ComfyUI_repackaged		Updated 6 days ago		278	
bartowski/Qwen_QwQ-32B-GGUF	Text Generation	Updated 8 days ago	159k	139	
microsoft/Phi-4-mini-instruct	Text Generation	Updated 3 days ago	156k	351	
microsoft/OmniParser-v2.0	Image-Text-to-Text	Updated 24 days ago	9.42k	1.15k	

Running them locally

- My students use VLLM <https://docs.vllm.ai/en/latest/>



The screenshot shows the vLLM documentation website. On the left is a sidebar with navigation links. The main content area has a large 'Welcome to vLLM' header with the vLLM logo and a tagline. Below this is a GitHub repository summary showing 41,349 stars. The text describes vLLM as a fast and easy-to-use library for LLM inference and serving, originally developed at UC Berkeley. It lists several features that make vLLM fast.

Getting Started

- Installation
- Quickstart
- Examples
- Troubleshooting
- Frequently Asked Questions

Models

- Generative Models
- Pooling Models
- List of Supported Models
- Built-in Extensions

Features

- Quantization
- LoRA Adapters
- Tool Calling
- Reasoning Outputs
- Structured Outputs
- Automatic Prefix Caching
- Disaggregated Prefilling (experimental)

Welcome to vLLM

Easy, fast, and cheap LLM serving for everyone

Star 41,349 Watch Fork

vLLM is a fast and easy-to-use library for LLM inference and serving.

Originally developed in the [Sky Computing Lab](#) at UC Berkeley, vLLM has evolved into a community-driven project with contributions from both academia and industry.

vLLM is fast with:

- State-of-the-art serving throughput
- Efficient management of attention key and value memory with [PagedAttention](#)
- Continuous batching of incoming requests
- Fast model execution with CUDA/HIP graph
- Quantization: [GPTQ](#), [AWQ](#), INT4, INT8, and FP8
- Optimized CUDA kernels, including integration with FlashAttention and FlashInfer.
- Speculative decoding
- Chunked prefill

Contents

- Documentation
- Indices and tables

Quantization

- Models have large GPU memory requirements.
- For example, on the 24GB memory GPUs in my group we can run (inference, not training) 7B Llama models.
 - That's because with 2 byte (16bit) precision the required memory for 7B parameters is 14 GB (roughly speaking).
 - We also need some memory for the “**KV Cache**”, which depends on the sequence length.
- One can compress these models using a technique called “quantization”. This decreases their performance somewhat.
- **Quantization in large language models (LLMs) is a technique used to reduce the memory footprint and computational cost of inference by representing model parameters with lower-precision data types, such as 8-bit integers (INT8) or even lower, instead of the standard 16-bit (FP16) or 32-bit floating-point (FP32) representations.**
- In this way we can use models that are two or four times larger.

Prompt Engineering

- Prompt engineering is **the practice of crafting effective prompts to guide AI models**, such as large language models (LLMs).
- It involves **structuring inputs in a way that optimizes the model's performance** for specific tasks, such as text generation, code writing, or problem-solving.
- Example: **Chain-of-Thought (CoT) Prompting** – Encouraging step-by-step reasoning improves logical accuracy.
- <https://arxiv.org/abs/2201.11903> Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
- Some people joke that AI research has been reduced to prompt engineering.

Chain-of-Thought Prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Few-Shot Learning

- **Few shot learning** - The model is given a few examples of input-output pairs before making a prediction.
 - Few-shot learning enables models to adapt to new tasks without extensive retraining.
- **One-Shot Learning** – A special case of few-shot learning where only one example is provided.
- **Zero-Shot Learning** – The model makes predictions without any examples, relying solely on prior knowledge.
- <https://arxiv.org/abs/2005.14165> Language Models are Few-Shot Learners

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French:
2 cheese => .....
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French:
2 sea otter => loutre de mer
3 cheese => .....
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1 Translate English to French:
2 sea otter => loutre de mer
3 peppermint => menthe poivrée
4 plush girafe => girafe peluche
5 cheese => .....
```

Multi-Agent Frameworks

- Many works explore combining LLMs to solve tasks.
- An example of this line of research is
 - <https://arxiv.org/abs/2409.15254> Archon: An Architecture Search Framework for Inference-Time Techniques

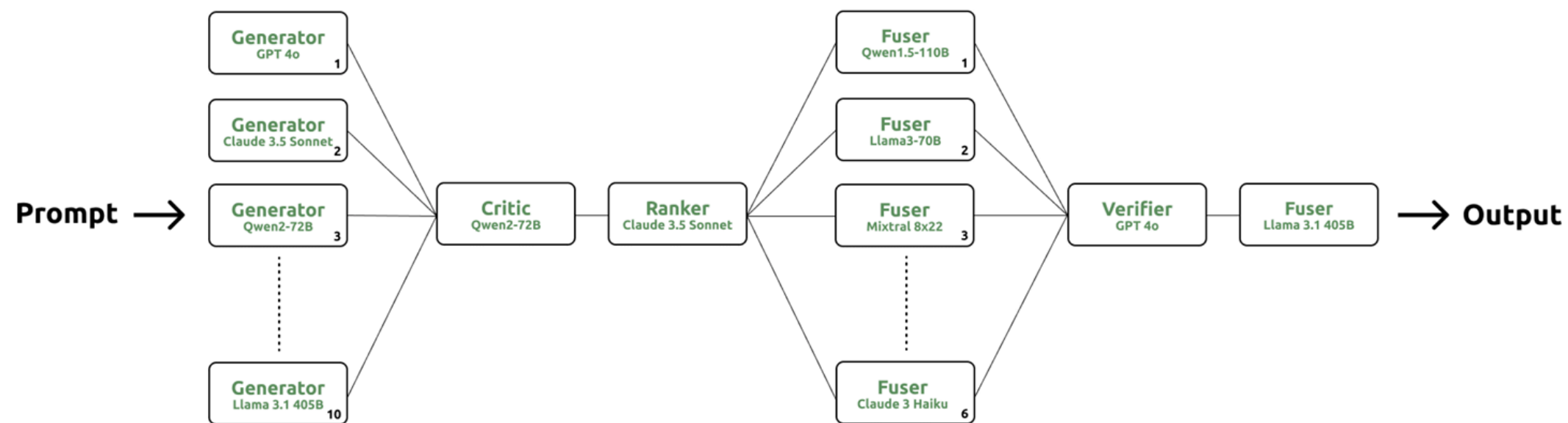


Figure 2: **Example ARCHON Architecture:** The architecture starts with ten generator models, followed by a critic model, a ranker model, one layer of six fuser models, a verifier model, and finishing with a fuser model.

Course logistics

- **Reading for this lecture:**
 - This lecture was based in part on the books by Bishop and Prince, linked on the website. Many figures were taken from these books.