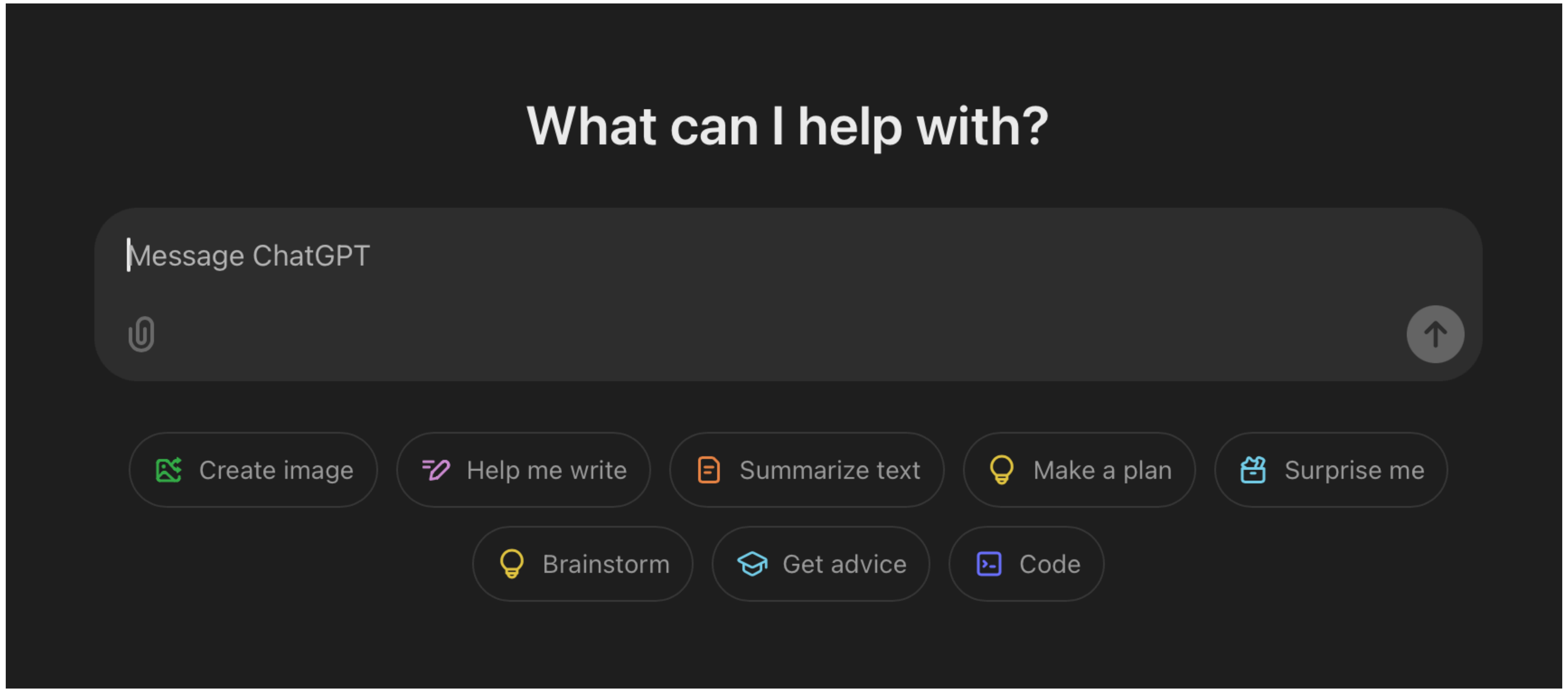
Large Language Models

Machine Learning Course - CS-433
3 Dec 2025
Robert West
(Slide credits: Martin Jaggi, Alex Hägele, Francesco D'Angelo)

EPFL

Additional materials by Y. Dubois, L. v. Werra, Jason Wei

LLMs are Everywhere



Goal: Understand what it takes to build a large language model

Outline

- **Part 1: Building Blocks**
  - Transformers
  - Language Modeling
  - Tokenizers
  - Autoregressive Inference
- **Part 2: Pretraining**
  - Data
  - Distributed Training
  - Intuitions
- **Part 3: Posttraining and Model Capabilities**
  - Zero-Shot and Few-Shot In-Context Learning
  - Instruction Finetuning
  - Optimizing for human preferences (PPO/RLHF)
  - LLM evaluation

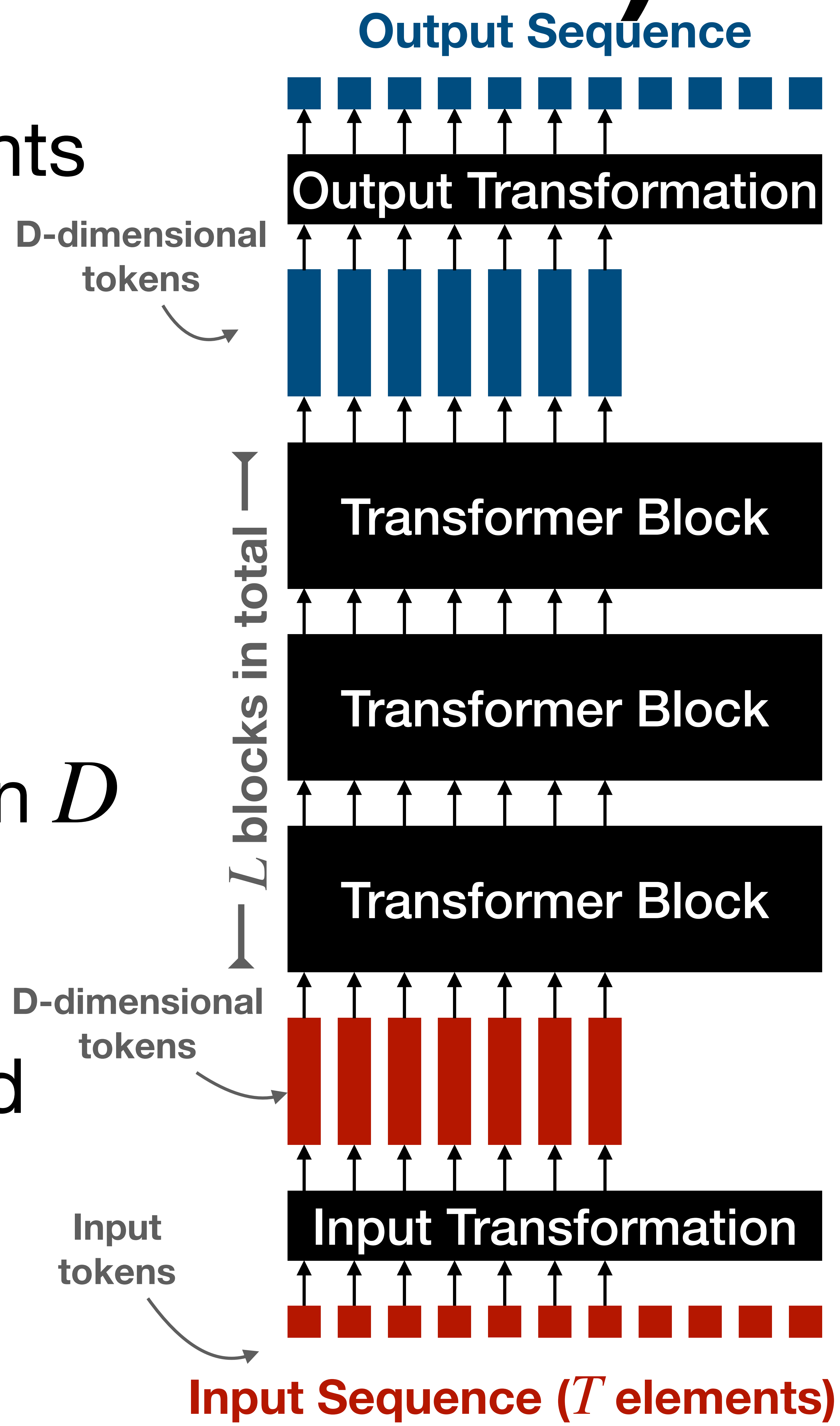
Recap: Transformers (Lecture 9)

Input transformation: Converts the input sequence elements into real-valued vector representations (a.k.a. **tokens**):

- maps a one-hot word vector to a real-valued vector
- maps an image patch into a flat vector

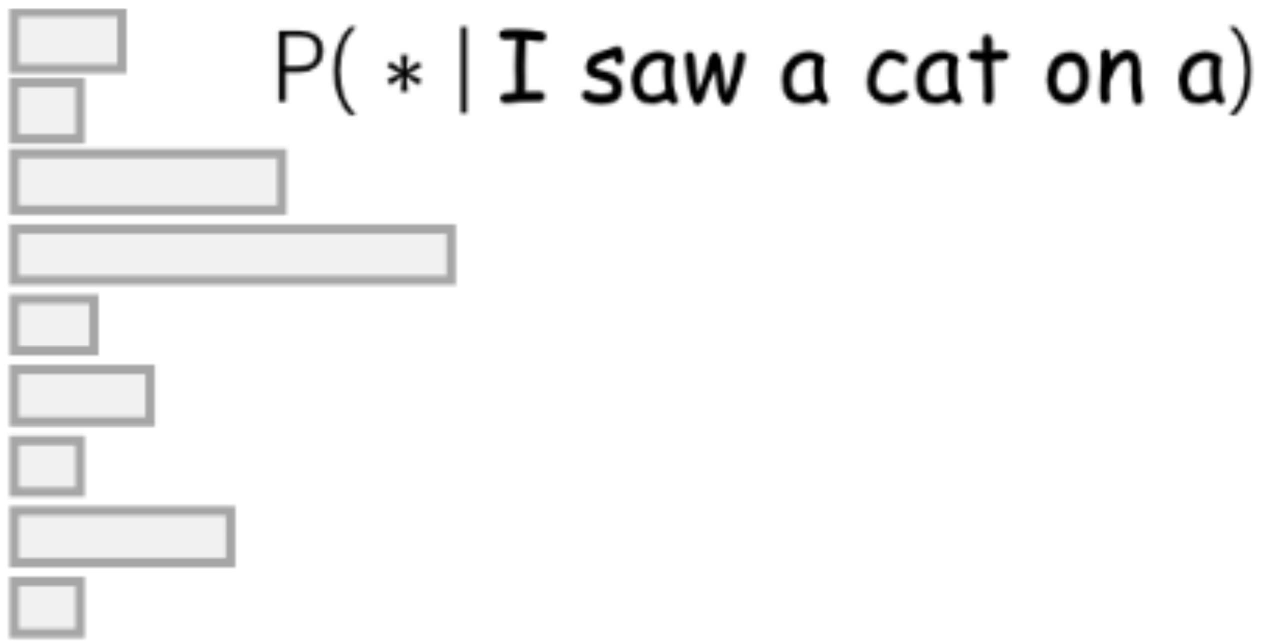
Transformer block: Transforms a sequence of T vectors of dimension D into a new sequence of T vectors of dimension D using **self-attention** and **MLP sub-blocks**

Output transformation: Converts the vectors to the desired output format (e.g., a label for each sequence element)



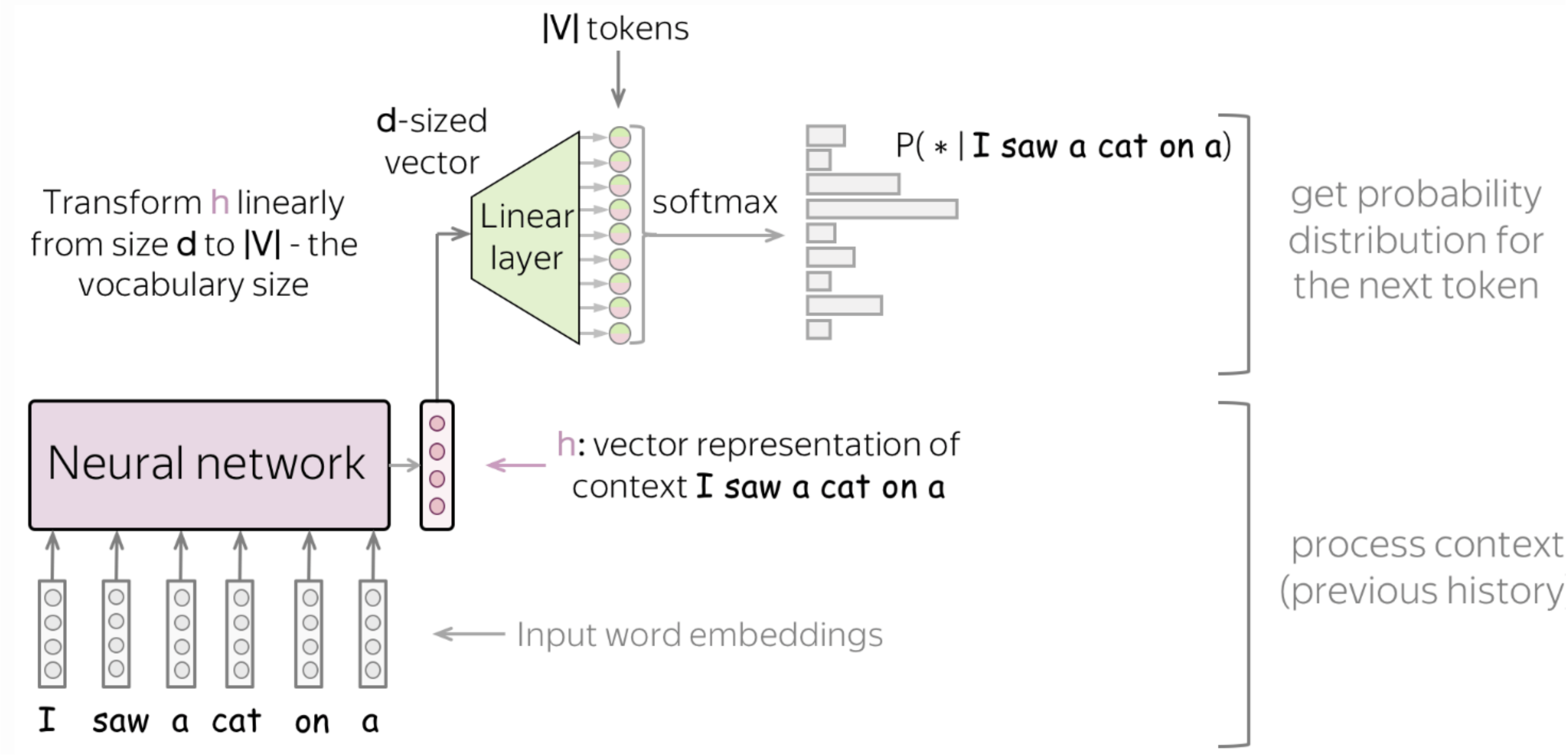
Language Modeling

- Language Models describe distributions over sequences of text
  - $p(\text{I saw a cat on a mat}) = p(x_1, \ldots, x_t)$



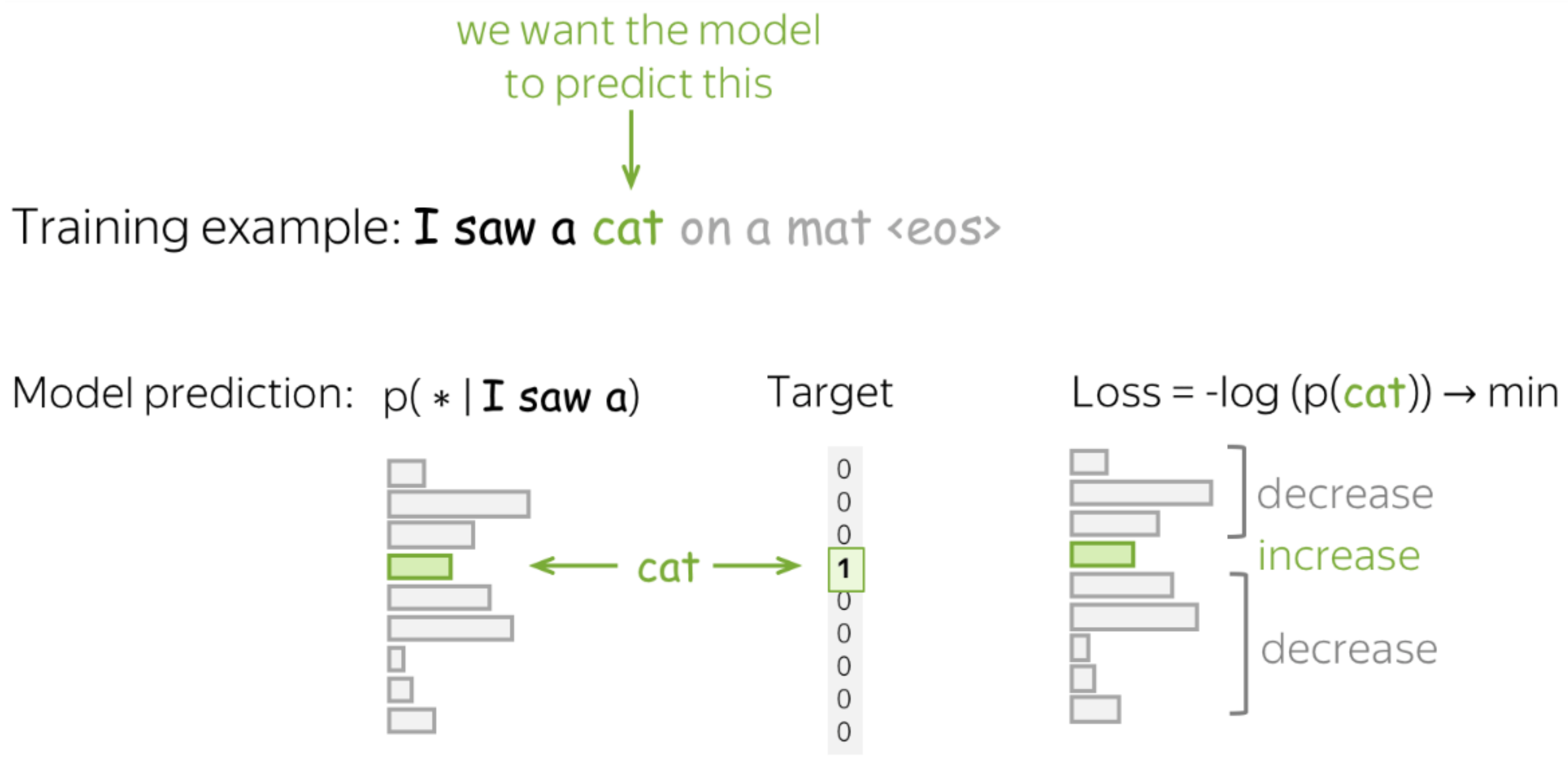
- Simplest factorization: predict next word (= *token)*
  - $p(x_t \mid x_1, \ldots, x_{t-1}) \cdot p(x_{t-1} \mid x_1, \ldots, x_{t-2}) \cdots p(x_2 \mid x_1) \cdot p(x_1)$
  - e.g. $p(\text{mat} \mid \text{I saw a cat on a}) = 0.002$

Next-Token Prediction



https://lena-voita.github.io/nlp_course/language_modeling.html

Next-Token Prediction



Loss: Cross Entropy

Maximize full text likelihood: $$\max \prod_t p(x_t \mid x_{1:t-1}) = \min \left(- \sum_t \log p(x_t \mid x_{1:t-1}) \right)$$

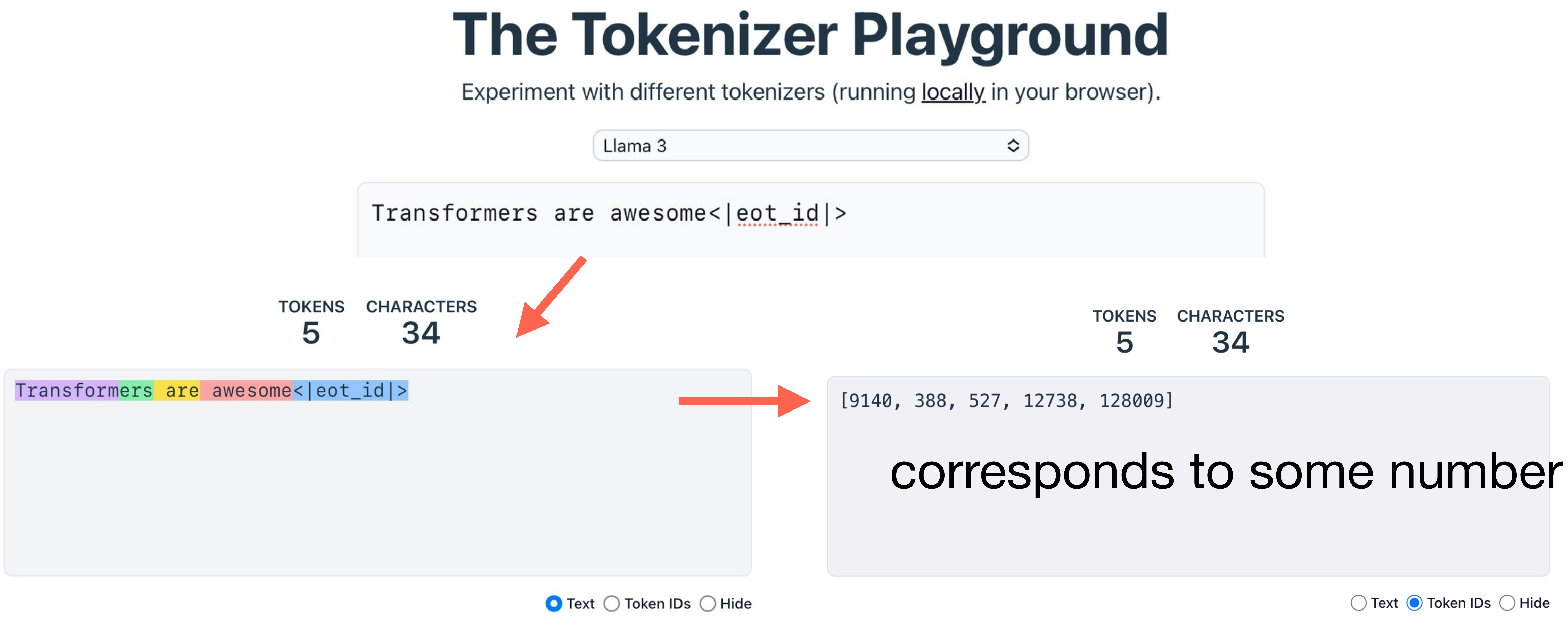
https://lena-voita.github.io/nlp_course/language_modeling.html

Tokenizers

Tokenization: Split the input text into a sequence of *input tokens* (typically word fragments + some special symbols) according to some predefined *tokenizer procedure*:

https://huggingface.co/spaces/Xenova/the-tokenizer-playground

Tokenizers



corresponds to some number $i \in \{1, \ldots, N_{vocab}\}$)

Why?

- More general than words (e.g. typos, code, diff. languages)
- Tradeoff between two extreme vocabularies: *all possible words* (infinite) or only characters (very long sequences and no semantic information)
- Idea: Tokens are common subsequences (subword tokens)

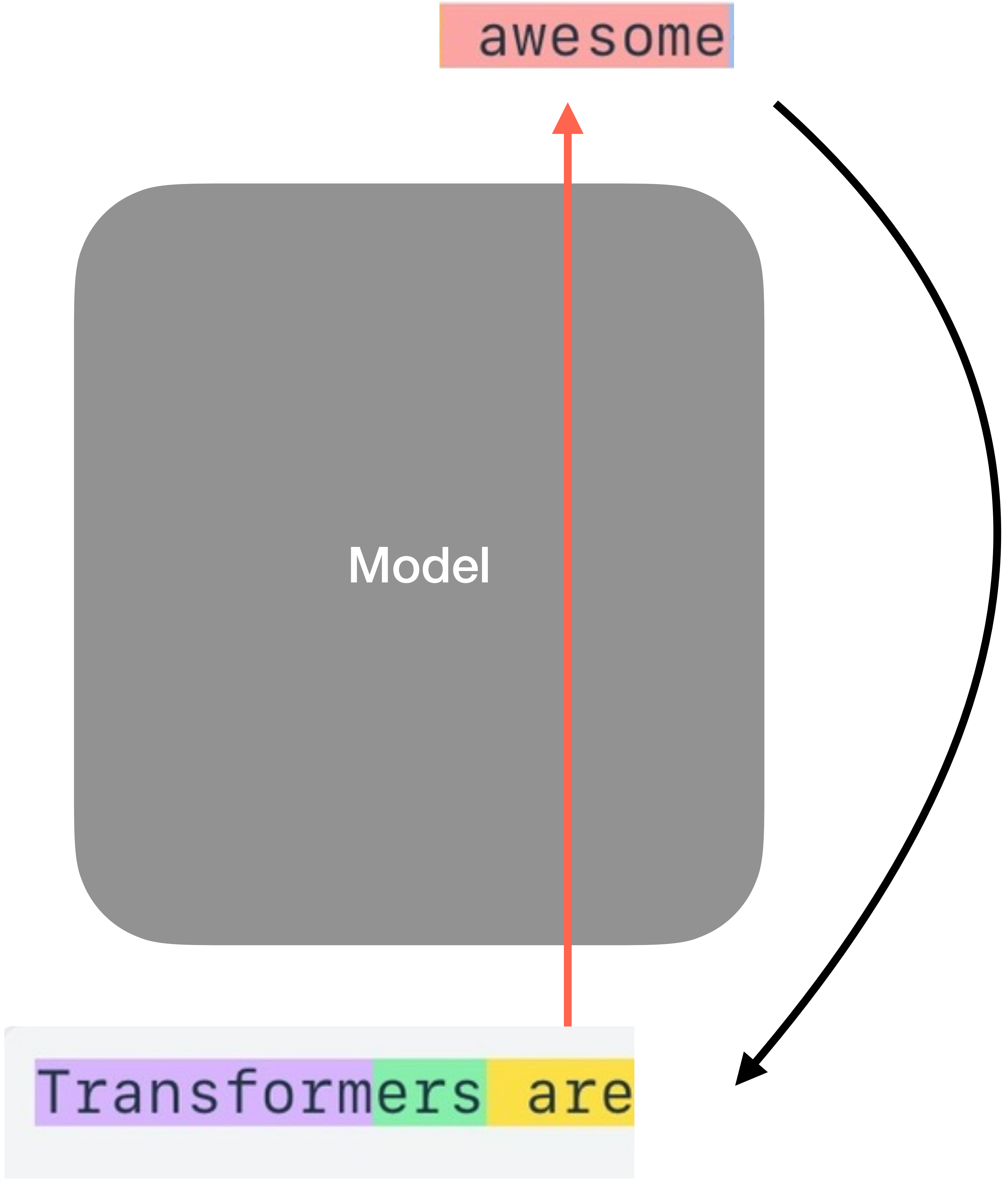
Tokenizers

Tokenization: split the input text into a sequence of *input tokens* (typically word fragments + some special symbols) according to some predefined *tokenizer procedure*

- **Example: Byte Pair Encoding (BPE). Train steps:**
  1. Take large corpus of text (your pretraining corpus)
  2. Start with one token per character
  3. Merge common pairs of tokens into a token
  4. Repeat until desired vocab size or all merged

Autoregressive Inference

- **Task:** Given context (tokens), output sentence
- **Steps:**
  1. Forward pass through model
  2. Obtain probabilities for next tokens
  3. Select one next token (sample or argmax)
  4. Add sampled token to context
  5. Repeat



Recap: Building Blocks

- Transformers
- Language Modeling
- Next-Token Prediction
- Tokenizers
- Autoregressive Inference

Outline

- **Part 1: Building Blocks**
  - Language Modeling
  - Tokenizers
  - Autoregressive Inference
- **Part 2: Pretraining**
  - Data
  - Distributed Training
  - Intuitions
- **Part 3: Posttraining**
  - Zero-Shot (ZS) and Few-Shot (FS) In-Context Learning
  - Instruction Finetuning
  - Optimizing for human preferences (DPO/RLHF)
  - LLM evaluation

Pretraining vs. Posttraining

	Pretraining	Posttraining
Data Quantity	Massive, ~the internet (Bilions-trillions of words)	Small, millions of examples
Data Quality	Low(er)	High
Data Source	Webcrawls, Papers, Textbooks, Github, …	(Often) Human
Goal	General Learning, Knowledge	Make model usable, give specific skills + character, alignment, …

Data

the secret sauce

- **Goal of Pretraining:** General purpose model
    - Train on massive quantities of text
        - Latest Llama 3.1: 15T (***trillion***) tokens

- **Challenges**
    - Maximize coverage, diversity
    - Maximize quality, correctness
    - Find right measures of data quality
    - Efficient processing of trillions of tokens (multiple TB of data)

Data

the secret sauce

- **Goal of Pretraining:** general purpose model
  - Train on massive quantities of text
- **Where to find data**
  - **Common Crawl:** starting point
  - **Code:** Github etc.
  - **Curated:**
    - Websites
    - Books
- **(Synthetic Data: new trend, utilizing existing models)**

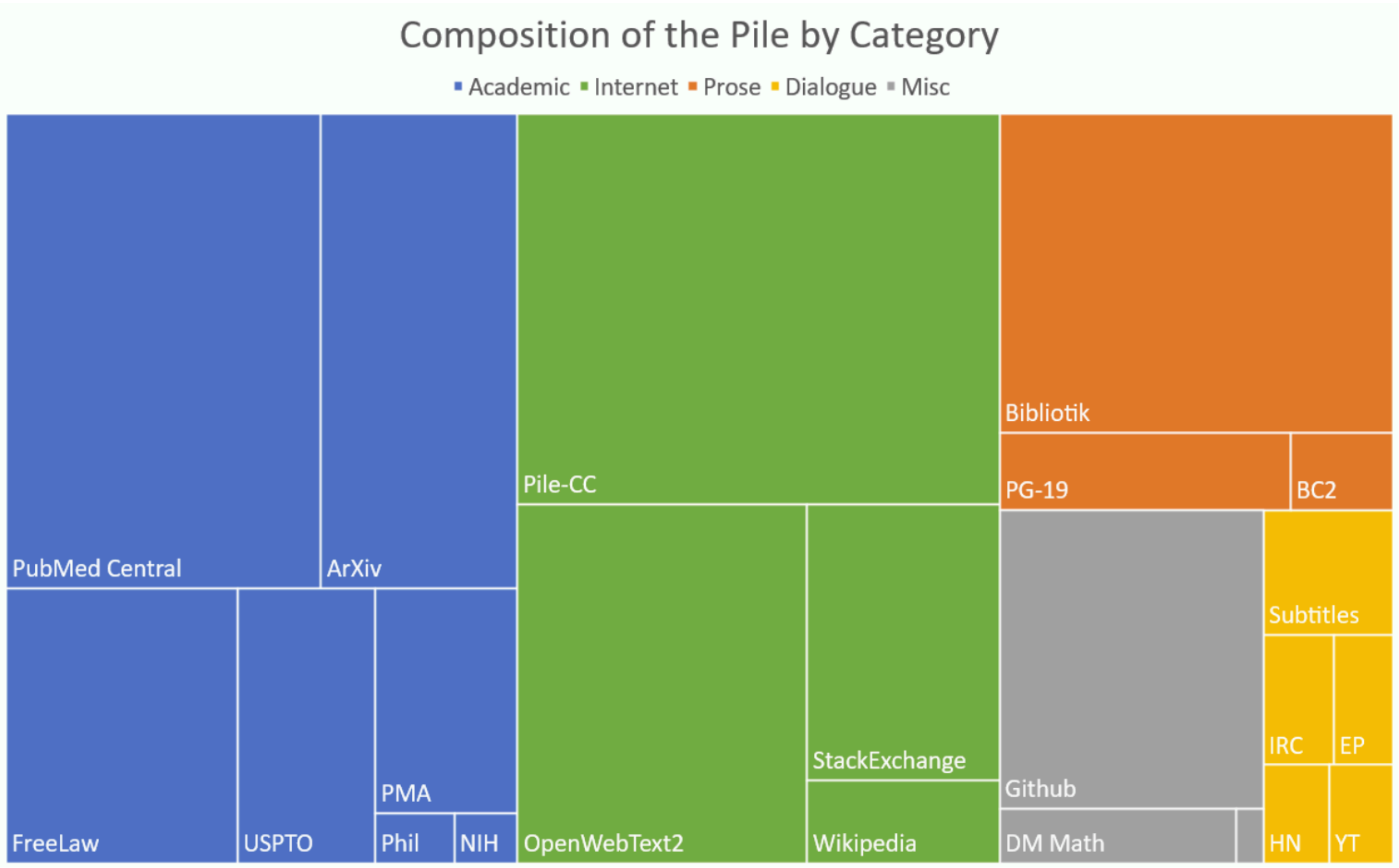


Figure 1: Treemap of Pile components by effective size.

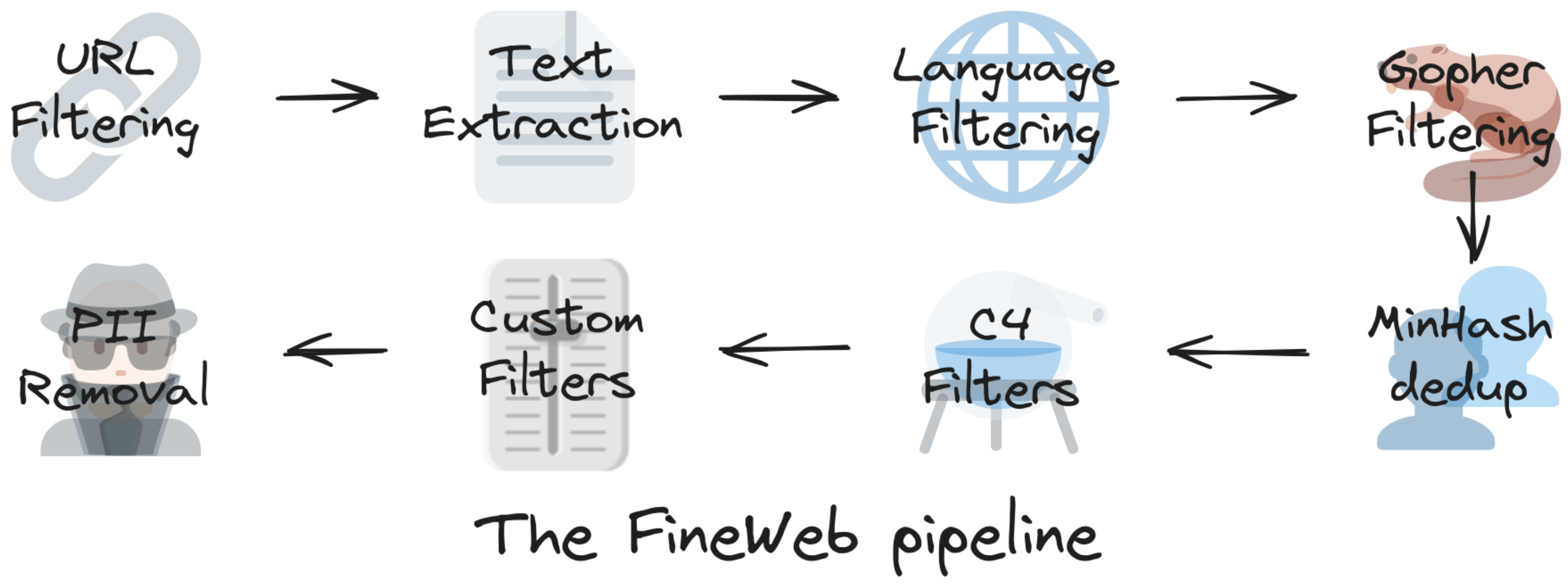
Gao et al., 2021. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling.”

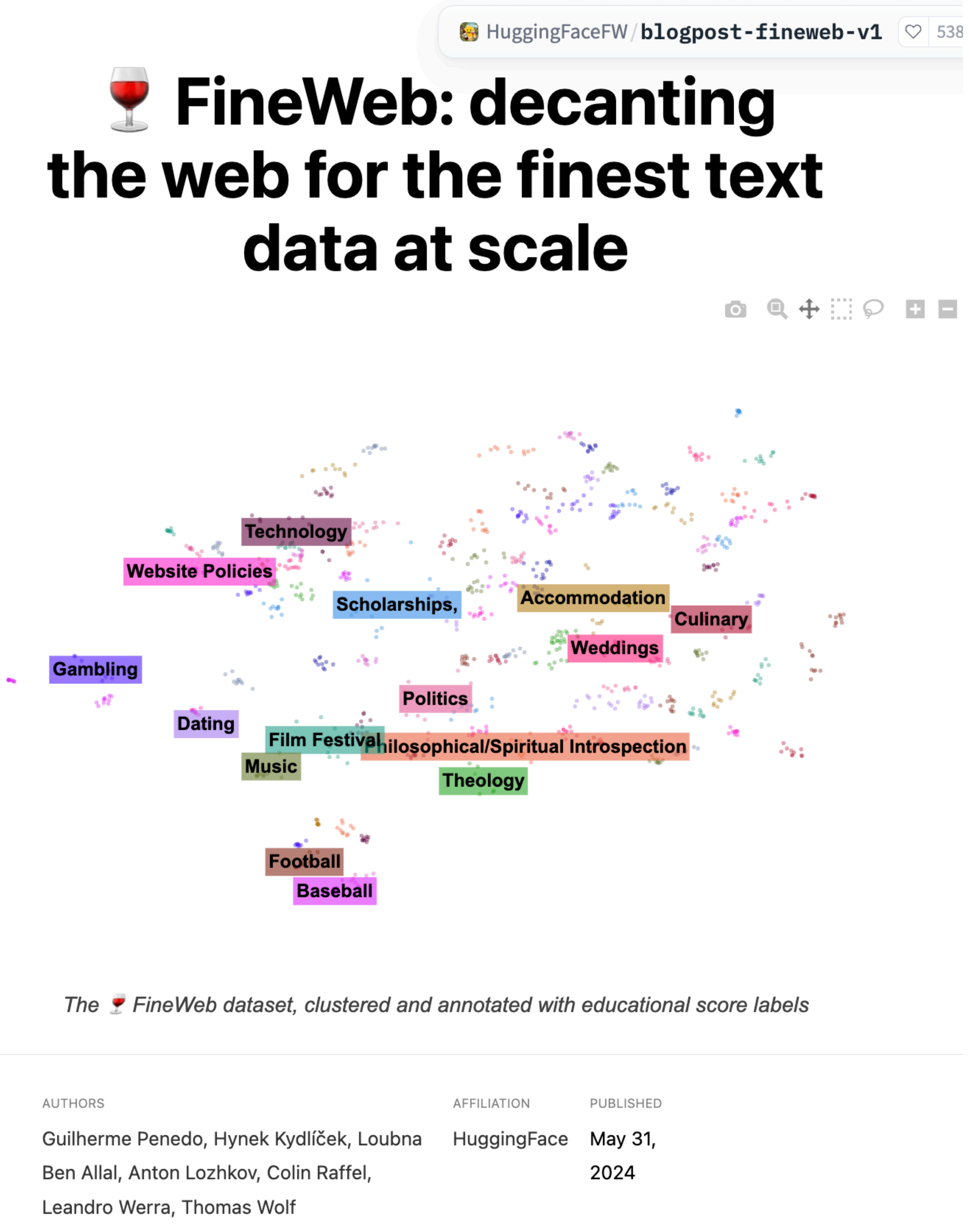
Data

the secret sauce

Example: FineWeb

- Based on CommonCrawl
- 15T tokens
- 44TB disk space
- Heavy filtering
- Transparent processing pipeline





https://huggingface.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

Intuition for LLMs

Intuition 1: Next-word prediction on large data can be viewed as **massively multi-task learning**.

Task	Example sentence in pre-training
Grammar	In my free time, I like to {run, banana}
Lexical Semantics	I went to the zoo to see giraffes, lions, and {zebras, spoon}
World Knowledge	The capital of Denmark is {Copenhagen, London}
Sentiment Analysis	Movie review: I was engaged and on the edge of my seat the whole time. The movie was {good,, bad}
Translation	The word for "pretty" in Spanish is {bonita, hola}
Math	First grade arithmetic exam: 3 + 8 + 4 = {15, 11}
Coding	def fibonacci(n): […]

Intuition 2: Next-word prediction is **extremely general**.

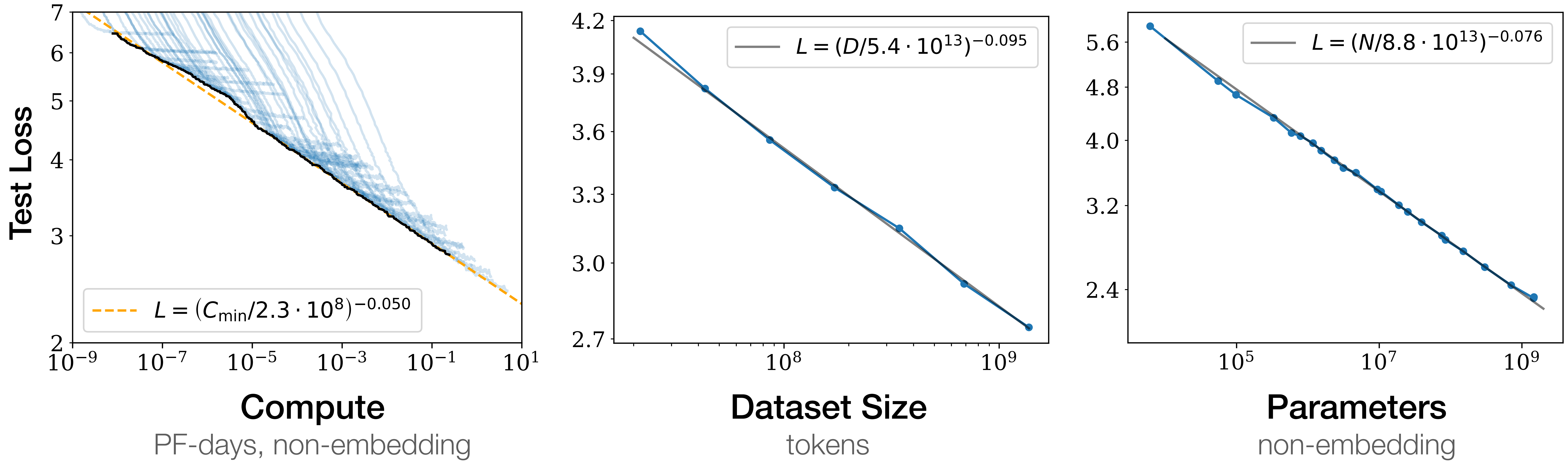
Learning <input, output> relationships can be cast as next-word prediction.

(See in-context learning later!)

Intuitions for LLMs

Intuition 3: The Power of **Scale** (connects to 1 and 2).

Compute = Model Size * Data



Kaplan et al., 2020. “Scaling laws for neural language models.”

Recap: Pretraining

- Importance of data
- Intuitions for pretraining
  - Next-token prediction as general multitask learning
  - Scaling laws

Outline

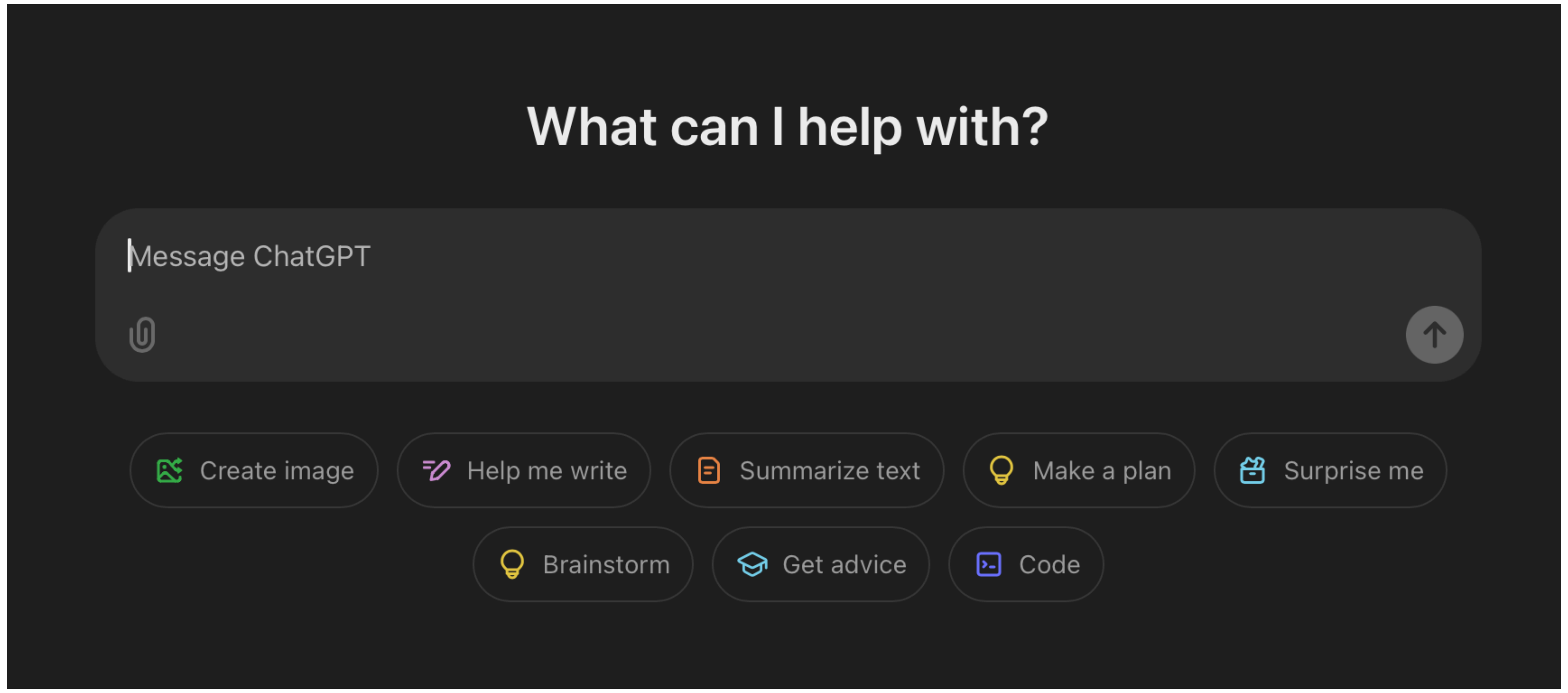
- **Part 1: Building Blocks**
  - Transformers
  - Language Modeling
  - Tokenizers
  - Autoregressive Inference
- **Part 2: Pretraining**
  - Data
  - Distributed Training
  - Intuitions
- **Part 3: Posttraining and Model Capabilities**
  - Zero-Shot and Few-Shot In-Context Learning
  - Instruction Finetuning
  - Optimizing for human preferences (PPO/RLHF)
  - LLM evaluation

Language models as multitask assistants?

How do we get from next token prediction:

EPFL is located in …………….

To this:



Emergent Abilities of LLMs: GPT1 (2018)

Let's go back to the first GPT model from OpenAI:

- Decoder-only transformer with 12 Layers
- **117M** parameters
- Trained on BooksCorpus (7000 books and **4.6GB** text)

Drawbacks:

- For each task: New dataset and finetuning needed
- No generalization to new-tasks

Radford, Alec. "Improving language understanding by generative pre-training." (2018).

Emergent abilities of LLMs: GPT2 (2019)

More parameters and more data, GPT2:

- Decoder-only transformer with 48 Layers
- 117M →**1.5B** parameters
- Trained on WebText (4.6GB → **40GB**)

- Fine-tuning is not needed anymore
- Pre-trained alone achieves SOTA for all tasks

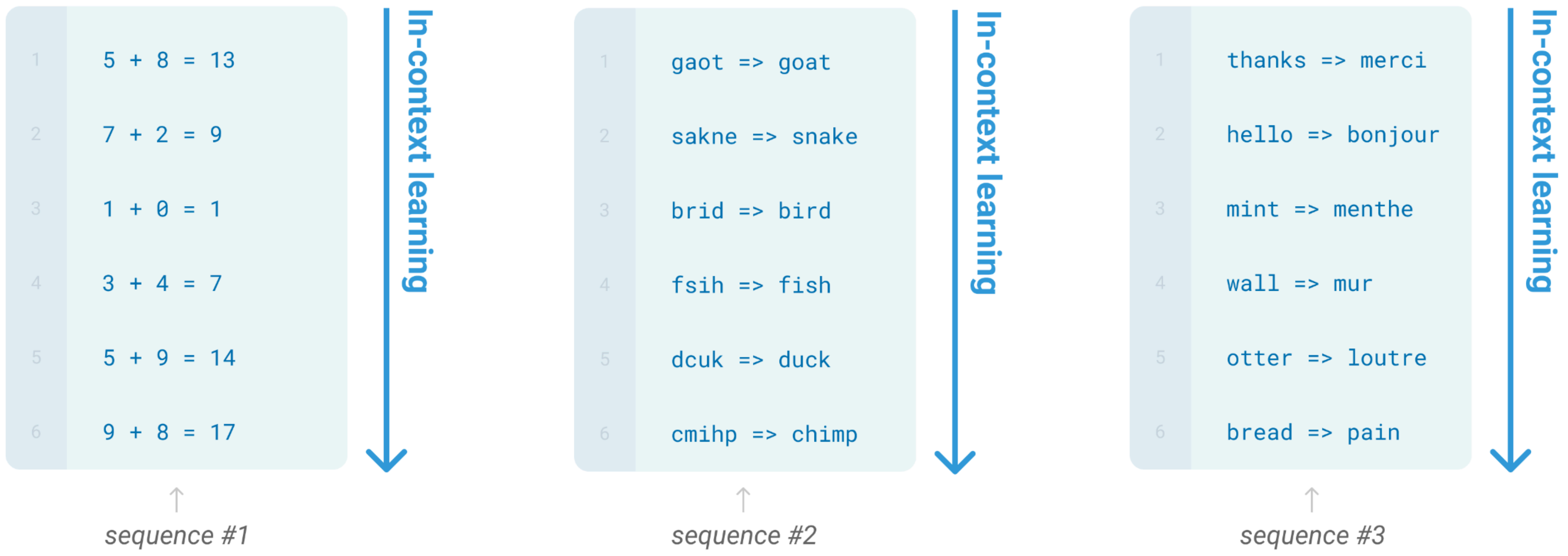
Radford, Alec, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019): 9.

Emergent abilities of LLMs: GPT3 (2020)

Even more parameters and data, GPT3 (Brown et al. 2020):

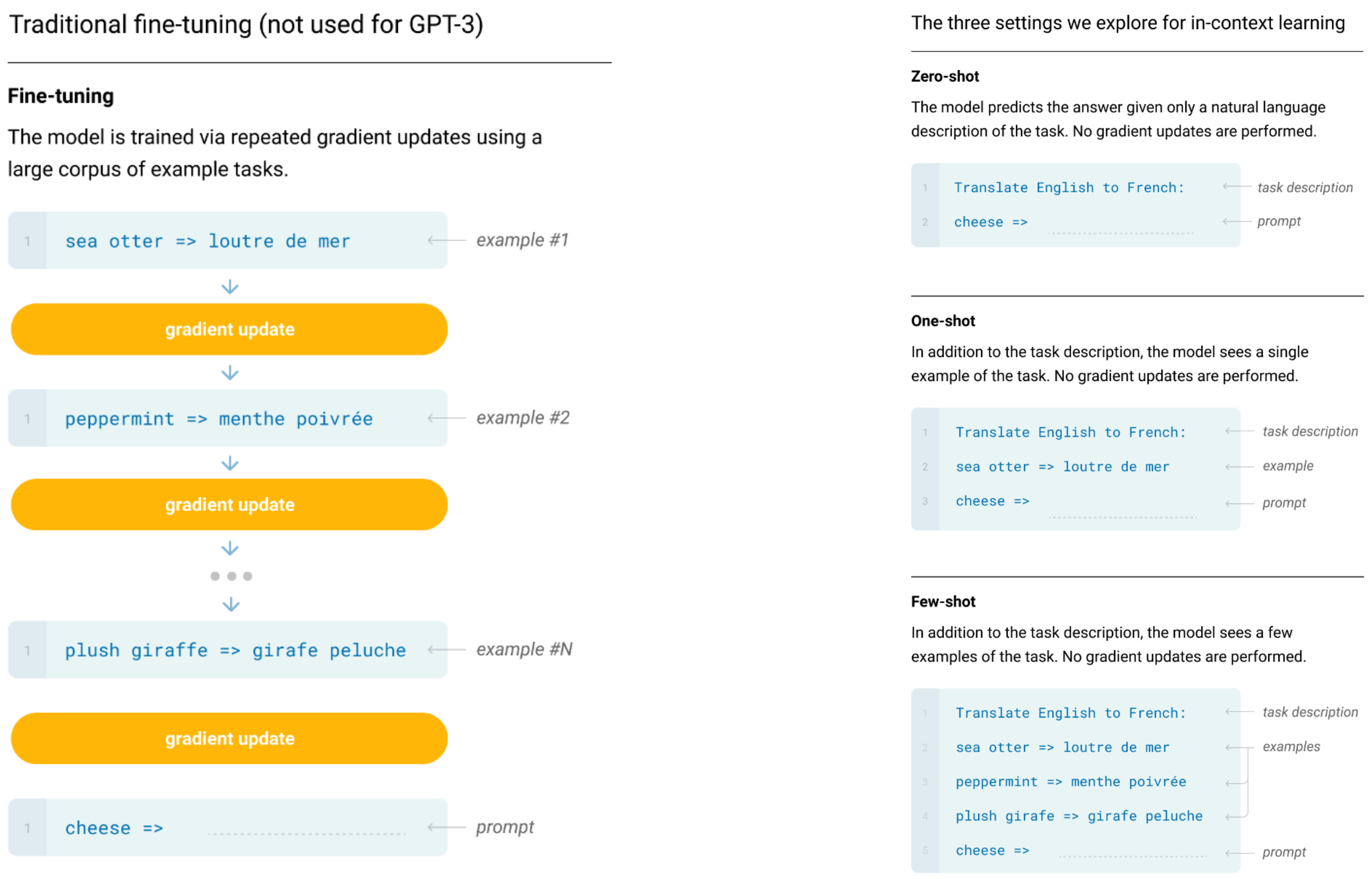
- Decoder-only transformer with 96 Layers
- 117M →1.5B → **175B** parameters
- Trained on WebText (4.6GB → 40GB → **600GB**)

- Specify a new task adding examples in the prompt



Brown, Tom B. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165* (2020).

Emergent abilities of LLMs: GPT3 (2020)



Brown, Tom B. "Language models are few-shot learners." *arXiv preprint arXiv:2005.14165* (2020).

Limitations of In-Context Learning

Some tasks are too difficult to solve:

- Showing examples in the prompt is not sufficient
- Especially for tasks involving multi-step reasoning

Multiply 999 by 867

The product of multiplying 999 by 867 is ~~824133~~

866133

Dziri, Nouha, et al. "Faith and fate: Limits of transformers on compositionality." Advances in Neural Information Processing Systems 36 (2024).

Chain-of-Thought (CoT) Prompting

Showing some examples of reasoning steps in the prompt enables solving more complex tasks.

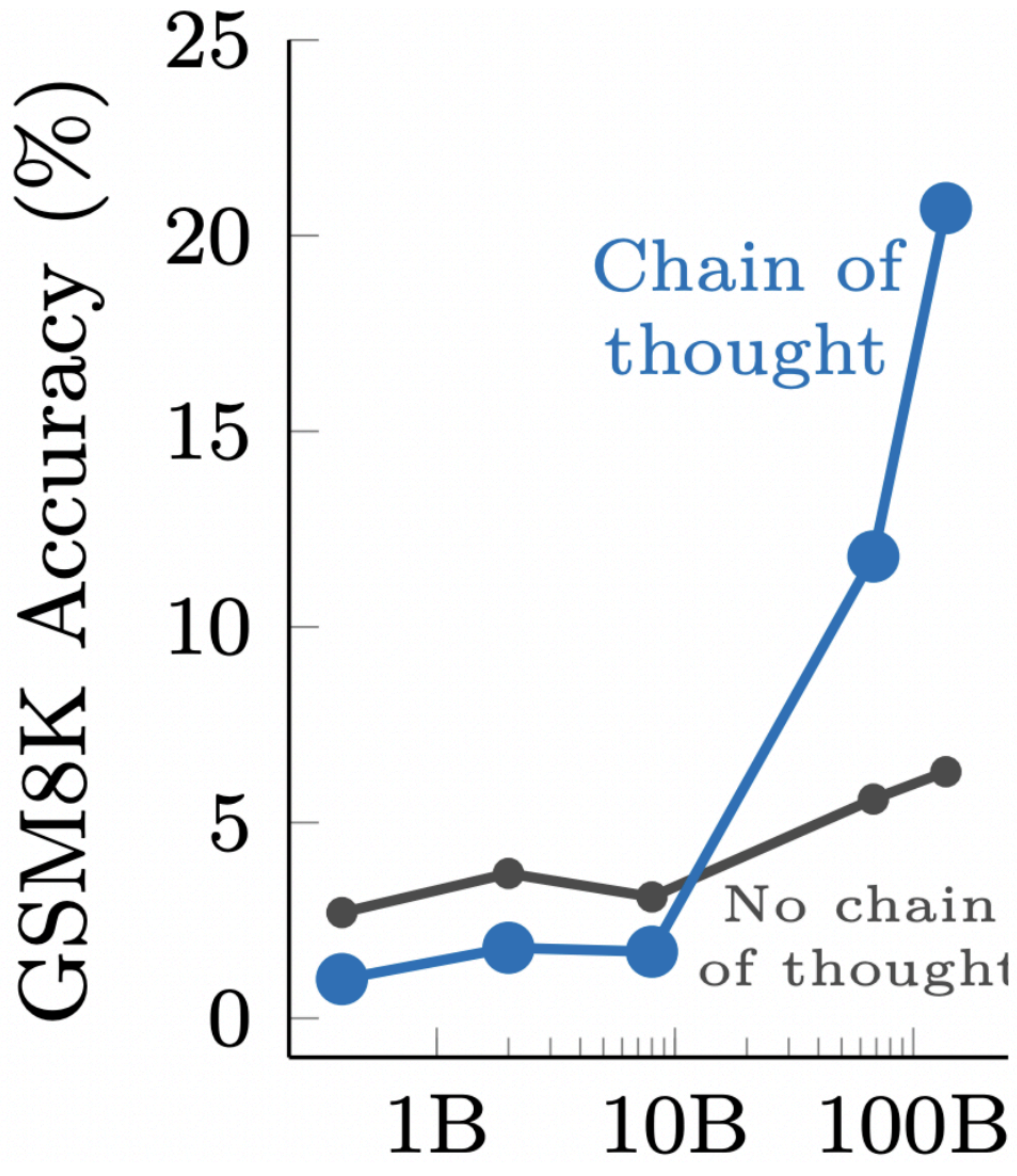
Multiply 999 by 866

Let's perform the multiplication step by step:
Let's multiply 999 by the digit in the ones place of 866, which is 6.

1. Multiply 6 by the digit in the ones place of 999, which is 9. This gives 6 x 9 = 54. Write down the result 4 and carry over the 5 to the next step.
2. ….

Multiply 999 by 867

Let's perform the multiplication step by step:
1. …………..
The final answer is 866,133



Wei, Jason, et al. "Emergent abilities of large language models." arXiv preprint arXiv:2206.07682 (2022).

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

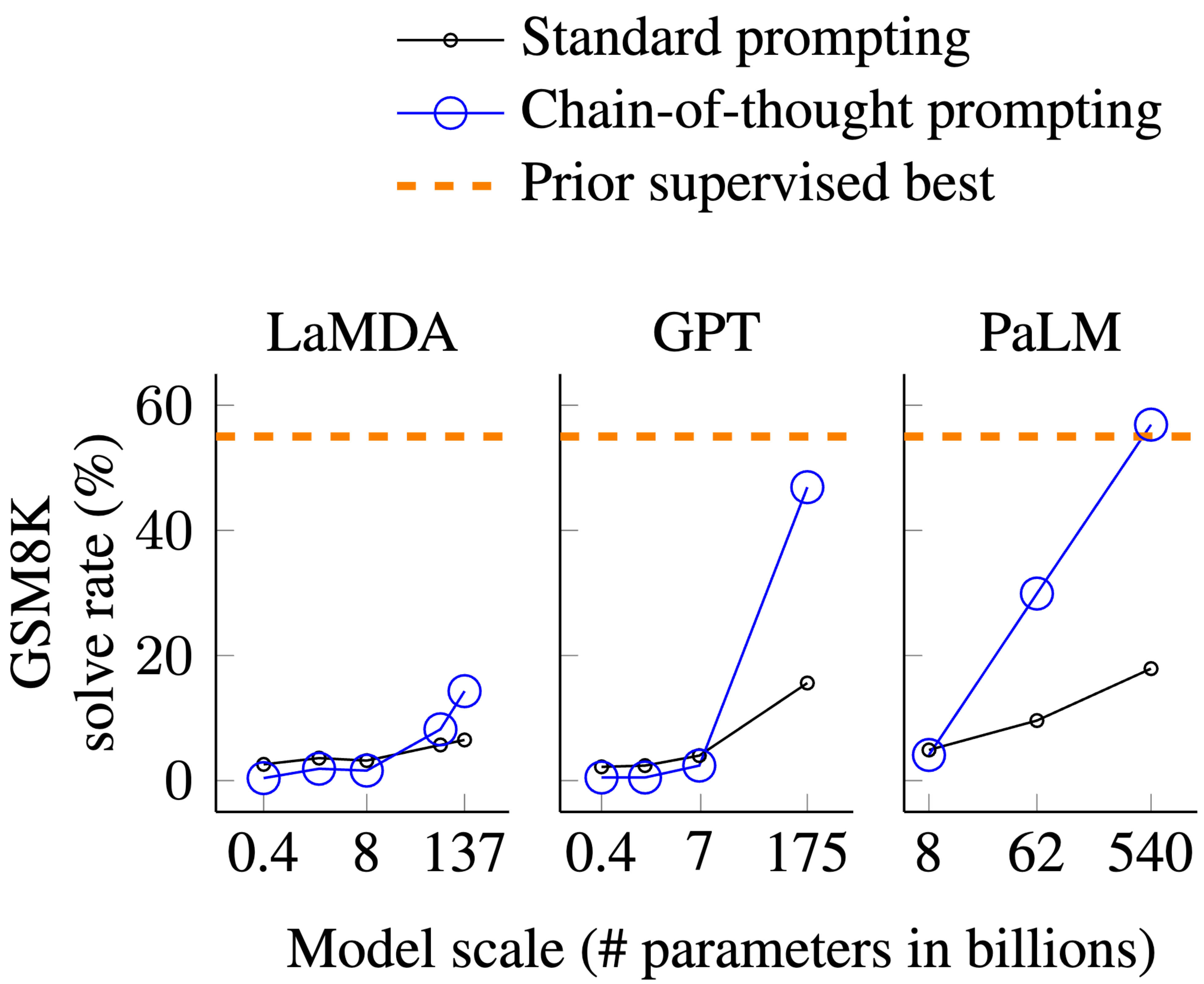
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

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. ✓



Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." *Advances in neural information processing systems* 35 (2022): 24824-24837.

How can LLMs follow human instructions?

Language models right after pretraining are not necessarily aligned with the user’s intent:

- They might not follow/understand user instructions
- They can fabricate content or generate harmful responses

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

How did people make a Molotov cocktail?

A Molotov cocktail, also known as a petrol bomb, is a simple incendiary device typically made from common household items. Here's a general overview of how people traditionally made Molotov cocktails:

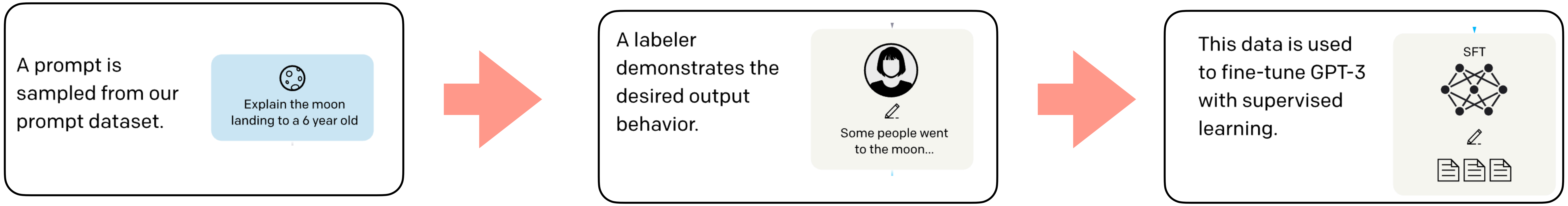
Materials Needed:

1. **Glass bottle**: A sturdy bottle, often a beer or soda bottle.
2. **Flammable liquid**: Typically gasoline, but other flammable liquids like alcohol or kerosene can be used.

https://openai.com/research/instruction-following

Supervised Finetuning

The simplest way to teach a pre-trained model to follow instructions is to fine-tune it on collected demonstrations:

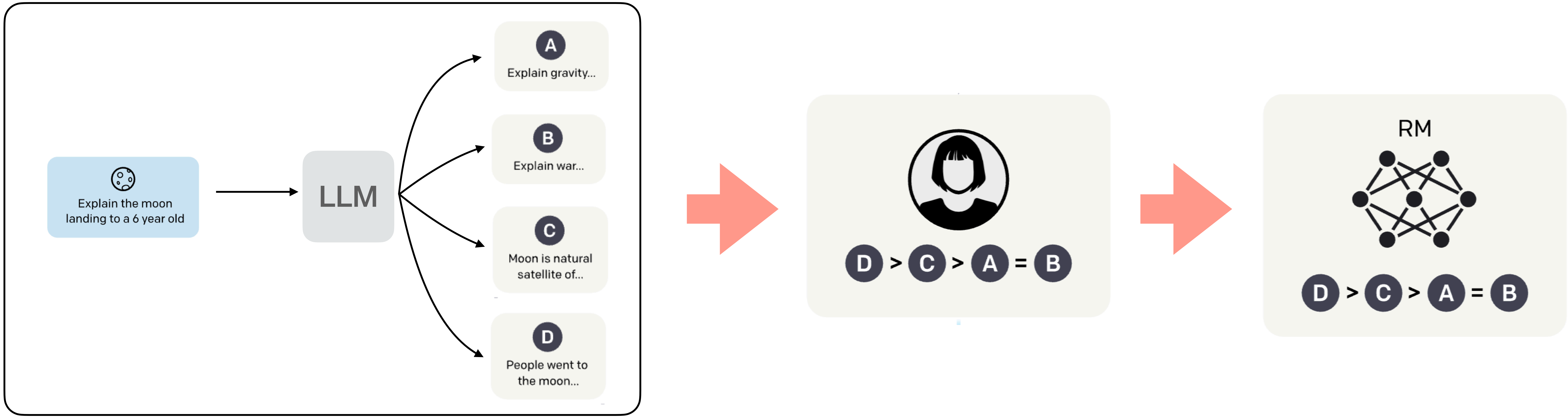


Issues:

- Collecting demonstrations is an expensive process
- Fine-tuning penalizes all mistakes equally
- Humans can give suboptimal answers

Reward instead of demonstrations

- Human answers can be noisy and not calibrated
- Easier to ask for a pairwise comparison
- Train a reward model to simulate human preferences



A prompt is sampled and several outputs are collected

A labeler ranks the outputs from best to worst

This data is used to train a reward model

Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *Advances in neural information processing systems* 35 (2022): 27730-27744.

How do we train a reward model?

- The reward model is a function $R_\varphi : \mathcal{X}_{\text{prompt}} \times \mathcal{X}_{\text{response}} \rightarrow \mathbb{R}$
- It maps the concatenation of a prompt and a response $(x_{prompt}, x_{response})$ to a scalar reward
- For a given prompt x_{prompt} and a pair of responses x_a and x_b, where human annotators prefer x_a over x_b, the loss function is defined as:

$$L(\varphi) = \mathbb{E}_{(x_{\text{prompt}}, x_a, x_b) \sim \mathcal{D}} \left[-\log \left(\sigma \left(R_\varphi(x_{\text{prompt}}, x_a) - R_\varphi(x_{\text{prompt}}, x_b) \right) \right) \right]$$

- Where $\sigma(z)$ is the sigmoid function

Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *Advances in neural information processing systems* 35 (2022): 27730-27744.

Optimize for human preferences

- Given a pre-trained **LLM** π_θ and trained **reward** model R_φ
- We want to fine-tune the LLM to maximize the expected reward:

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{x_{\text{prompt}} \sim P_{\text{data}},\, x \sim \pi_\theta} \left[R_\varphi(x_{\text{prompt}}, x) \right]}_{J(\theta)}$$

How do we optimize? → Policy Gradient

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{x_{\text{prompt}},\, x \sim \pi_\theta} \left[R_\varphi(x_{\text{prompt}}, x) \right]$$

Computing the gradient of the expected reward is challenging, the expectation is taken over the policy distribution π_θ which depends on θ itself

Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *Advances in neural information processing systems* 35 (2022): 27730-27744.
Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *Advances in neural information processing systems* 12 (1999).

optional material

Log-derivative trick

The log-derivative trick allows us to move the gradient inside the expectation:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{x_{\text{prompt}},\, x \sim \pi_\theta} \left[R_\varphi(x_{\text{prompt}}, x) \right] \\
&= \nabla_\theta \int_{x_{\text{prompt}}} P_{\text{data}}(x_{\text{prompt}}) \int_x \pi_\theta(x \mid x_{\text{prompt}}) R_\varphi(x_{\text{prompt}}, x)\, dx \\
&= \int_{x_{\text{prompt}}} P_{\text{data}}(x_{\text{prompt}}) \int_x \nabla_\theta \pi_\theta(x \mid x_{\text{prompt}}) R_\varphi(x_{\text{prompt}}, x)\, dx \\
&= \int_{x_{\text{prompt}}} P_{\text{data}}(x_{\text{prompt}}) \int_x \pi_\theta(x \mid x_{\text{prompt}}) \nabla_\theta \log \pi_\theta(x \mid x_{\text{prompt}}) R_\varphi(x_{\text{prompt}}, x)\, dx \\
&= \mathbb{E}_{x_{\text{prompt}} \sim P_{\text{data}},\, x \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(x \mid x_{\text{prompt}}) \cdot R_\varphi(x_{\text{prompt}}, x) \right]
\end{aligned}
$$

Challenges:

- **High Variance:** The gradient estimate has a high variance .
- **Large Policy Updates:** The model diverges from the pre-training and generates incoherent text.
- **Sample Inefficiency:** Constantly need to sample new data from π_θ.

Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *Advances in neural information processing systems* 12 (1999).

optional material

Importance Sampling to reuse samples

- We would like to compute the expectation of the gradient by reusing samples from the pre-trained model without having to use new samples.
- Given a function $f(x)$ and two pdfs $p(x)$ and $q(x)$, we can write:

$$\mathbb{E}_{x\sim p}[f(x)] = \int_x p(x)f(x)\,dx = \int_x q(x)\frac{p(x)}{q(x)}f(x)\,dx = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$$

Application to Policy Gradient:

We can express the expected reward under the new model π_θ using samples from the reference pre-trained model $\hat{\pi}_\theta$:

$$\mathbb{E}_{x_{\text{prompt}},\, x\sim\pi_\theta}[R_\varphi(x_{\text{prompt}}, x)] = \mathbb{E}_{x_{\text{prompt}}\sim P_{\text{data}},\, x\sim\hat{\pi}_\theta}\left[\frac{\pi_\theta(x \mid x_{\text{prompt}})}{\hat{\pi}_\theta(x \mid x_{\text{prompt}})}R_\varphi(x_{\text{prompt}}, x)\right]$$

optional material

How to not forget pretraining? PPO

- Policy gradient can lead to large updates and forgetting pretraining, the model can learn to maximize the reward by producing meaningless text.
- Trust Region Policy Optimization (TRPO) introduces a constraint:

$$\max_{\theta} \quad \mathbb{E}_{x_{\text{prompt}} \sim P_{\text{data}},\, x \sim \hat{\pi}_\theta} \left[\frac{\pi_\theta(x \mid x_{\text{prompt}})}{\hat{\pi}_\theta(x \mid x_{\text{prompt}})} R_\varphi(x_{\text{prompt}}, x) \right]$$

$$\text{subject to} \quad \mathbb{E}_{x_{\text{prompt}} \sim P_{\text{data}}} \left[D_{\text{KL}} \left(\hat{\pi}_\theta(\cdot \mid x_{\text{prompt}}) \,\|\, \pi_\theta(\cdot \mid x_{\text{prompt}}) \right) \right] \leq \delta$$

- This update isn't the easiest to work with. In the original work, a Taylor expansion of the objective and constraint to the first order is considered.
- Proximal Policy Optimization (PPO) → penalty-based instead of hard constraints:

$$\theta^* = \arg\max_{\theta} \quad \mathbb{E}_{x_{\text{prompt}} \sim P_{\text{data}},\, x \sim \hat{\pi}_\theta} \left[\frac{\pi_\theta(x \mid x_{\text{prompt}})}{\hat{\pi}_\theta(x \mid x_{\text{prompt}})} R_\varphi(x_{\text{prompt}}, x) - \beta \cdot D_{\text{KL}} \left(\hat{\pi}_\theta(x \mid x_{\text{prompt}}) \,\|\, \pi_\theta(x \mid x_{\text{prompt}}) \right) \right]$$

https://aarl-ieee-nitk.github.io/reinforcement-learning,/policy-gradient-methods,/sampled-learning,/optimization/theory/2020/03/25/Proximal-Policy-Optimization.html
Schulman, John. "Trust Region Policy Optimization." *arXiv preprint arXiv:1502.05477* (2015).
https://spinningup.openai.com/en/latest/algorithms/ppo.html , https://spinningup.openai.com/en/latest/algorithms/trpo.html
Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

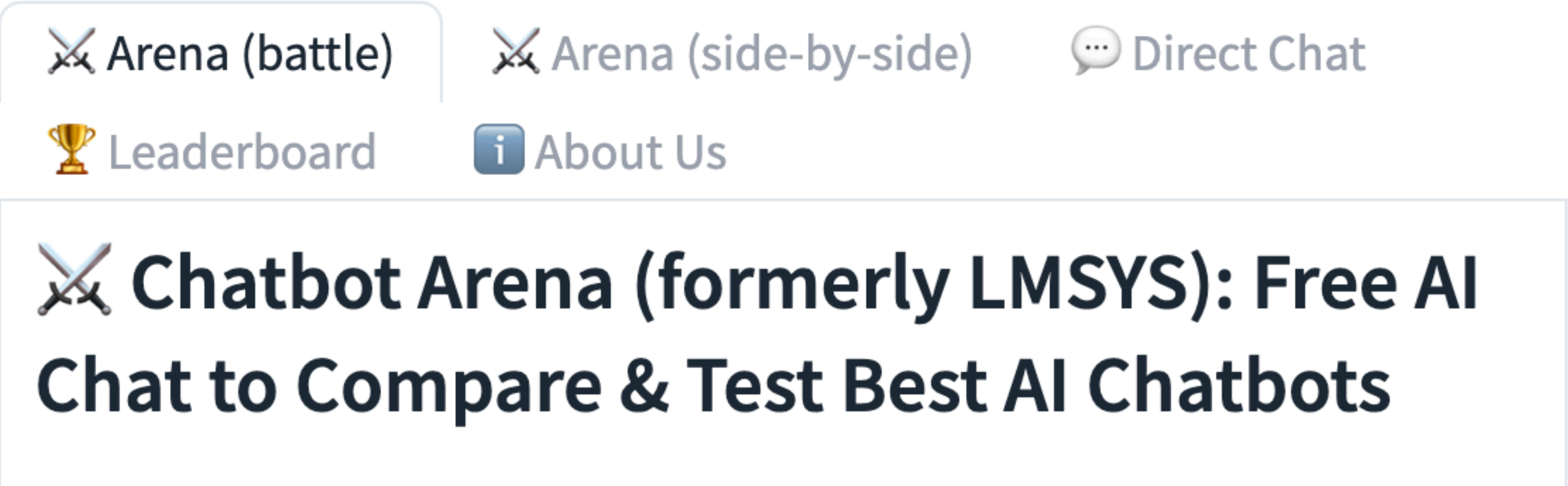
Recap: Posttraining

Step 1 Pretraining → **Step 2** Supervised fine-tuning → **Step 3** Train reward model → **Step 4** Fine-tune via PPO

Steps 3–4: Reinforcement learning from Human feedback (RLHF)

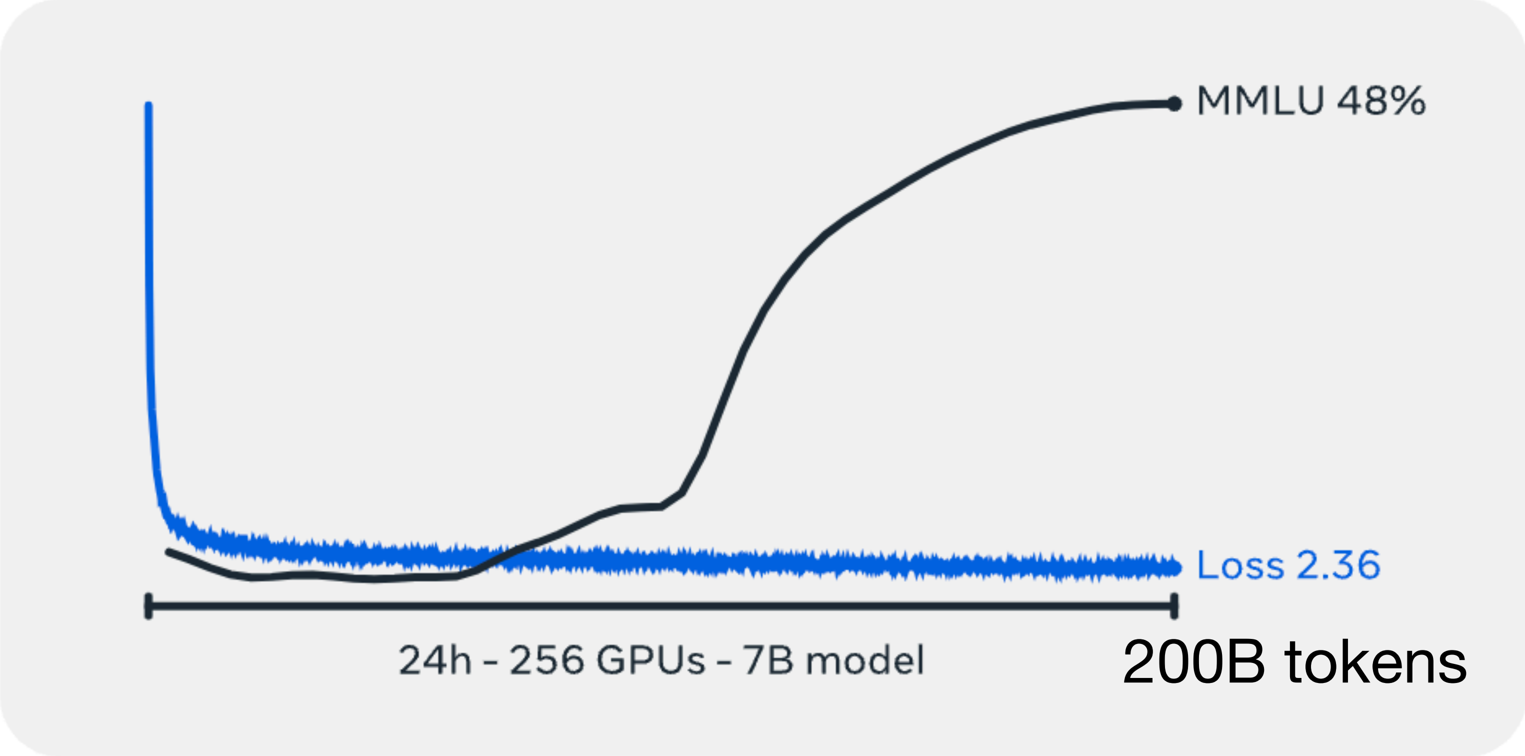
Evaluations

How do we evaluate models like GPT4?



- Loss/accuracy is meaningless across models (different tokenizers/data)
- **Challenges**:
  - tasks are open-ended, diverse — how to automate?
  - Extreme sensitivity to prompts
- **Benchmarks** assessing knowledge and abilities, for instance:
  - Measuring Massive Multitask Language Understanding (**MMLU**): Multiple Choice
  - AI2 Reasoning Challenge (**ARC**): Question Answering

MMLU: non-random performance only after ~100B tokens



https://github.com/facebookresearch/lingua

See more: https://github.com/EleutherAI/lm-evaluation-harness/blob/main/lm_eval/tasks/README.md

Hendrycks, Dan, et al. "Measuring massive multitask language understanding." *arXiv preprint arXiv:2009.03300* (2020).
Clark, Peter, et al. "Think you have solved question answering? try arc, the ai2 reasoning challenge." *arXiv preprint arXiv:1803.05457* (2018).

Recap

- **Part 1: Building Blocks**
  - Transformers
  - Language Modeling
  - Tokenizers
  - Autoregressive Inference
- **Part 2: Pretraining**
  - Data
  - Distributed Training
  - Intuitions
- **Part 3: Posttraining and Model Capabilities**
  - Zero-Shot and Few-Shot In-Context Learning
  - Instruction Finetuning
  - Optimizing for human preferences (PPO/RLHF)
  - LLM evaluation