CPEN 455: Deep Learning

Lecture 9: Large Language Models

Renjie Liao, Qi Yan

University of British Columbia

Winter, Term 2, 2024

Outline

- **Introduction & Background**
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Introduction & Background

Language Model (LM) learns a probability distribution over sequences of tokens.
For a vocabulary V of a set of tokens $\{x_1, x_2, \cdots, x_{|V|}\}$, the LM learns the joint probability for each sequence of tokens:

$$p(x_1, \ldots, x_L).$$

Introduction & Background

Language Model (LM) learns a probability distribution over sequences of tokens.
For a vocabulary V of a set of tokens $\{x_1, x_2, \cdots, x_{|V|}\}$, the LM learns the joint probability for each sequence of tokens:

$$p(x_1, \ldots, x_L).$$

Each token can represent a word. For example:

$$V = \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\}$$
$$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = 0.02,$$
$$p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) = 0.01,$$
$$p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) = 0.0001,$$

Introduction & Background

Language Model (LM) learns a probability distribution over sequences of tokens.
For a vocabulary V of a set of tokens $\{x_1, x_2, \cdots, x_{|V|}\}$, the LM learns the joint probability for each sequence of tokens:

$$p(x_1, \ldots, x_L).$$

Each token can represent a word. For example:

$$
\begin{aligned}
V &= \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\} \\
p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) &= 0.02, \\
p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) &= 0.01, \\
p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) &= 0.0001,
\end{aligned}
$$

The objective of language modeling is intuitively simple, but it becomes significantly complex as we scale up the size of the vocabulary and the sequence length.

Just imagine all the possible language and word combinations!

Introduction & Background

Language Model (LM) learns a probability distribution over sequences of tokens.
For a vocabulary V of a set of tokens $\{x_1, x_2, \cdots, x_{|V|}\}$, the LM learns the joint probability for each sequence of tokens:

$$p(x_1, \ldots, x_L).$$

Each token can represent a word. For example:

$$V = \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\}$$

$$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = 0.02,$$

$$p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) = 0.01,$$

$$p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) = 0.0001,$$

The assigned probability indicates two types of knowledge:

1) **Syntactic knowledge**, which involves reasoning over ungrammatical sequences.
2) **World knowledge**, which pertains to reasoning over semantic plausibility.

Introduction & Background

Modern Large Language Models (LLMs) are typically autoregressive models, which model the joint distribution $p(x_{1:L})$ using the chain rule of probability:

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1})$$

Introduction & Background

Modern Large Language Models (LLMs) are typically autoregressive models, which model the joint distribution $p(x_{1:L})$ using the chain rule of probability:

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1})$$

For example:

$$\begin{aligned} p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = & \, p(\text{the}) \\ & p(\text{mouse} \mid \text{the}) \\ & p(\text{ate} \mid \text{the}, \text{mouse}) \\ & p(\text{the} \mid \text{the}, \text{mouse}, \text{ate}) \\ & p(\text{cheese} \mid \text{the}, \text{mouse}, \text{ate}, \text{the}). \end{aligned}$$

Introduction & Background

Modern Large Language Models (LLMs) are typically autoregressive models, which model the joint distribution $p(x_{1:L})$ using the chain rule of probability:

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1})$$

For example:

$$\begin{aligned} p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = & \, p(\text{the}) \\ & p(\text{mouse} \mid \text{the}) \\ & p(\text{ate} \mid \text{the}, \text{mouse}) \\ & p(\text{the} \mid \text{the}, \text{mouse}, \text{ate}) \\ & p(\text{cheese} \mid \text{the}, \text{mouse}, \text{ate}, \text{the}). \end{aligned}$$

Particularly, we learn a conditional probability distribution for the next token:

$$p(x_i \mid x_{1:i-1})$$

We typically use a single feedforward neural network (such as transformers) to model such conditional distributions.

Introduction & Background

Modern LLMs size has increase more than **5000x** in last 4 years.

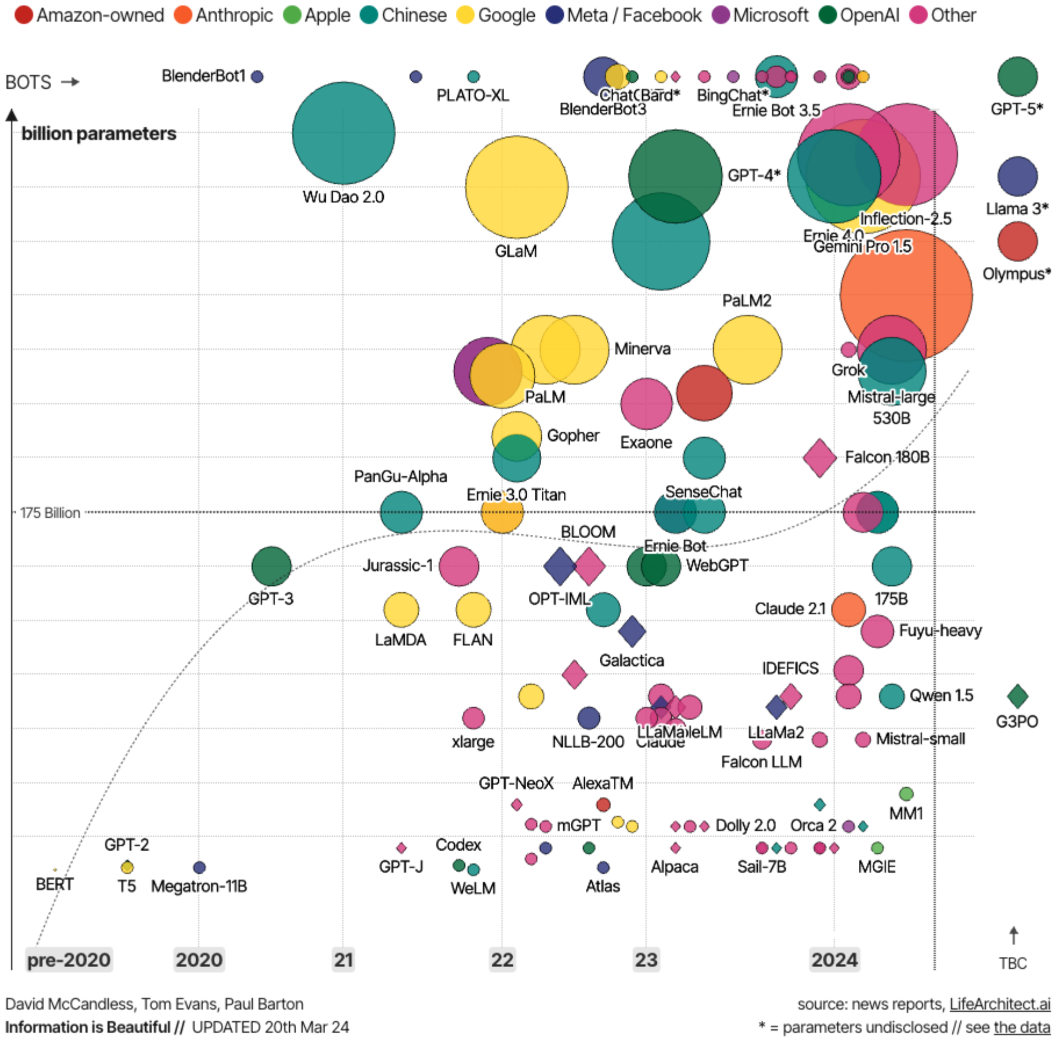


Image Credit: [1]

Introduction & Background

As LLMs get more powerful, will they lead to Artificial General Intelligence (AGI)?

Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck Varun Chandrasekaran Ronen Eldan Johannes Gehrke
Eric Horvitz Ece Kamar Peter Lee Yin Tat Lee Yuanzhi Li Scott Lundberg
Harsha Nori Hamid Palangi Marco Tulio Ribeiro Yi Zhang

Microsoft Research

OpenAI Research API ChatGPT Safety Company Search Log in Try ChatGPT

Blog

Planning for AGI and beyond

Our mission is to ensure that artificial general intelligence—AI systems that are generally smarter than humans—benefits all of humanity.

Article

Solving olympiad geometry without human demonstrations

https://doi.org/10.1038/s41586-023-06747-5

Received: 30 April 2023

Accepted: 13 October 2023

Published online: 17 January 2024

Open access

Check for updates

Trieu H. Trinh[1,2], Yuhuai Wu[1], Quoc V. Le[1], He He[2] & Thang Luong[1]

Proving mathematical theorems at the olympiad level represents a notable milestone in human-level automated reasoning[1–4], owing to their reputed difficulty among the world's best talents in pre-university mathematics. Current machine-learning approaches, however, are not applicable to most mathematical domains owing to the high cost of translating human proofs into machine-verifiable format. The problem is even worse for geometry because of its unique translation challenges[1,5], resulting in severe scarcity of training data. We propose AlphaGeometry, a theorem prover for Euclidean plane geometry that sidesteps the need for human demonstrations by synthesizing millions of theorems and proofs across different levels of complexity. AlphaGeometry is a neuro-symbolic system that uses a neural language model, trained from scratch on our large-scale synthetic data, to guide a symbolic deduction engine through infinite branching points in challenging problems. On a test set of 30 latest olympiad-level problems, AlphaGeometry solves 25, outperforming the previous best method that only solves ten problems and approaching the performance of an average International Mathematical Olympiad (IMO) gold medallist. Notably, AlphaGeometry produces human-readable proofs, solves all geometry problems in the IMO 2000 and 2015 under human expert evaluation and discovers a generalized version of a translated IMO theorem in 2004.

Image Credit: [2, 3, 4]

Outline

- Introduction & Background
- Models
  - **Tokenization**
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Tokenization

Recall the previous example on vocabulary:

$$V = \{\mathrm{ate}, \mathrm{ball}, \mathrm{cheese}, \mathrm{mouse}, \mathrm{the}\}$$

A tokenizer converts string (natural language representations) into machine-readable tokens:

$$\text{the mouse ate the cheese} \Rightarrow [\mathrm{the}, \mathrm{mouse}, \mathrm{ate}, \mathrm{the}, \mathrm{cheese}]$$

Tokenization

Recall the previous example on vocabulary:

$$V = \{\mathrm{ate}, \mathrm{ball}, \mathrm{cheese}, \mathrm{mouse}, \mathrm{the}\}$$

A tokenizer converts string (natural language representations) into machine-readable tokens:

$$\mathrm{the\ mouse\ ate\ the\ cheese} \Rightarrow [\mathrm{the}, \mathrm{mouse}, \mathrm{ate}, \mathrm{the}, \mathrm{cheese}]$$

Practical concerns: **split by spaces** don’t work in general.

1. Some languages don’t have spaces between words.

   English: What is machine learning? Chinese: 什么是机器学习？ Japanese: 機械学習とは何ですか？

2. Special cases like hyphenated words (e.g., *GPT-4*) or contractions (e.g., *don’t*).

Tokenization

Recall the previous example on vocabulary:

$$V = \{\mathrm{ate}, \mathrm{ball}, \mathrm{cheese}, \mathrm{mouse}, \mathrm{the}\}$$

A tokenizer converts string (natural language representations) into machine-readable tokens:

$$\text{the mouse ate the cheese} \Rightarrow [\mathrm{the}, \mathrm{mouse}, \mathrm{ate}, \mathrm{the}, \mathrm{cheese}]$$

Practical concerns: **split by spaces** don’t work in general.

1. Some languages don’t have spaces between words.
   English: What is machine learning? Chinese: 什么是机器学习？ Japanese: 機械学習とは何ですか？
2. Special cases like hyphenated words (e.g., *GPT-4*) or contractions (e.g., *don’t*).

We need a more principled approach to tokenization, ensuring that we have neither too many nor too few tokens, with each token representing a linguistically meaningful unit.

Tokenization

Here we introduce **byte pair encoding (BPE)** algorithm, which is one of the most popular tokenizers and has been used in OpenAI's products such as GPT-4.

Input: A training corpus composed of character sequences.
Initialization: Treat each character as an individual token. Establish initial vocabulary V as the set of distinct characters.
while V needs expansion **do**
    Identify the most frequently co-occurring pair of elements $x, x' \in V$.
    Replace every instance of x, x' with a new symbol xx'.
    Add the new symbol xx' to V.
end while

Tokenization

Example of BPE learning:
Step 1: [t, h, e, ␣, c, a, r], [t, h, e, ␣, c, a, t], [t, h, e, ␣, r, a, t]
Step 2: [th, e, ␣, c, a, r], [th, e, ␣, c, a, t], [th, e, ␣, r, a, t] (*th* occurs 3x)
Step 3: [the, ␣, c, a, r], [the, ␣, c, a, t], [the, ␣, r, a, t] (*the* occurs 3x)
Step 4: [the, ␣, ca, r], [the, ␣, ca, t], [the, ␣, r, a, t] (*ca* occurs 2x)
…

Tokenization

Example of BPE learning:
Step 1: [t, h, e, ␣, c, a, r], [t, h, e, ␣, c, a, t], [t, h, e, ␣, r, a, t]
Step 2: [th, e, ␣, c, a, r], [th, e, ␣, c, a, t], [th, e, ␣, r, a, t] (*th* occurs 3x)
Step 3: [the, ␣, c, a, r], [the, ␣, c, a, t], [the, ␣, r, a, t] (*the* occurs 3x)
Step 4: [the, ␣, ca, r], [the, ␣, ca, t], [the, ␣, r, a, t] (*ca* occurs 2x)
…

Results:

- Updated vocabulary: [a, c, e, h, t, r, ca, th, the]
- The merges that we made (important for applying the tokenizer):

  $t, h \Rightarrow th$
  $th, e \Rightarrow the$
  $c, a \Rightarrow ca$

In practice, we run BPE on the byte level encoding of all Unicode characters to handle multilingual tasks.
Example in Chinese:

今天 [gloss: today]
[x62, x11, 4e, ca]

Tokenization

Off-the-shelf BPE has a vocabulary size of 50K.

Example of open-sourced BPE from OpenAI:

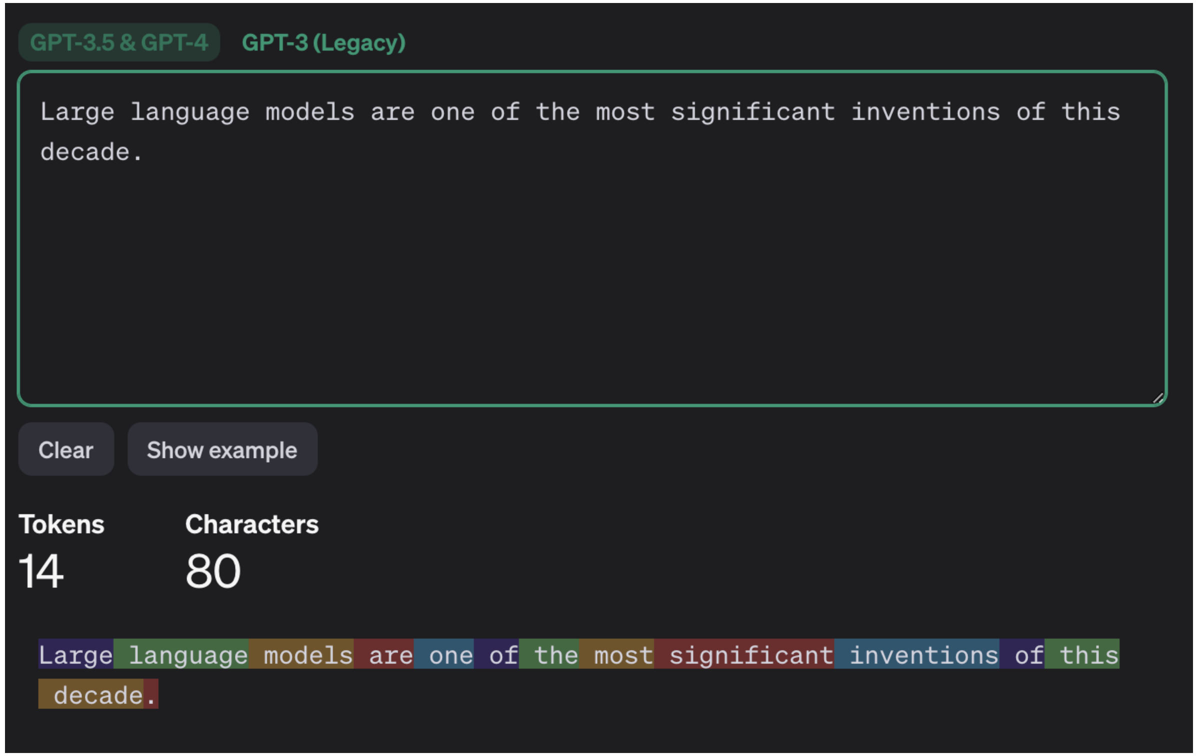


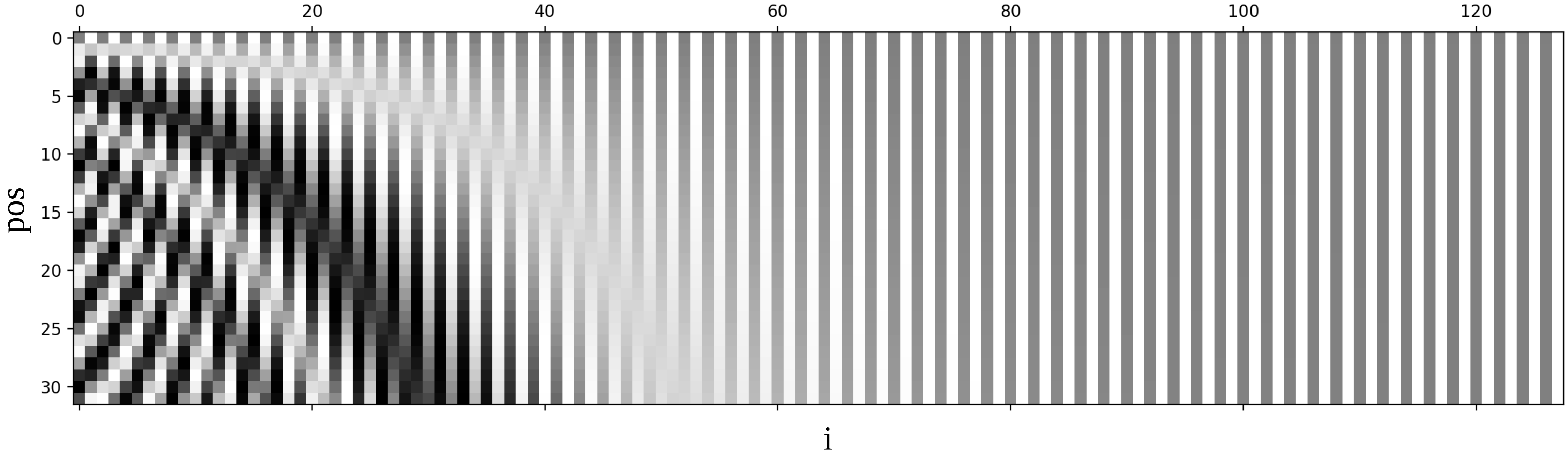
Image Credit: [5]

Outline

- Introduction & Background
- Models
  - Tokenization
  - **Rotary Positional Encoding**
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Rotary Positional Encoding

Recall the sinusoidal positional encoding for transformer:



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Image Credit: [6]

Rotary Positional Encoding

Problems with Positional Encoding:

- Fixed sinusoidal embeddings can theoretically handle sequences of arbitrary lengths. However, models often underperform when sequence lengths greatly differ from those in the training data.

- It only encodes the absolute position of a token within a sequence.

Rotary Positional Encoding

Problems with Positional Encoding:

- Fixed sinusoidal embeddings can theoretically handle sequences of arbitrary lengths. However, models often underperform when sequence lengths greatly differ from those in the training data.

- It only encodes the absolute position of a token within a sequence.

Rotary Positional Embeddings (RoPE) [24] are proposed to address such limitations:

- It encodes absolute position with a rotation matrix

- It encodes the explicit relative position dependency in self-attention

Rotary Positional Encoding

1. Encode absolute position with a rotation matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Rotary Positional Encoding

1. Encode absolute position with a rotation matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

2. Apply rotation to token embedding:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

Rotary Positional Encoding

1. Encode absolute position with a rotation matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

2. Apply rotation to token embedding:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

The inner product within the self-attention encodes the relative position:

$$\boldsymbol{q}_m^{\intercal} \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^{\intercal} (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^{\intercal} \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n$$

Rotary Positional Encoding

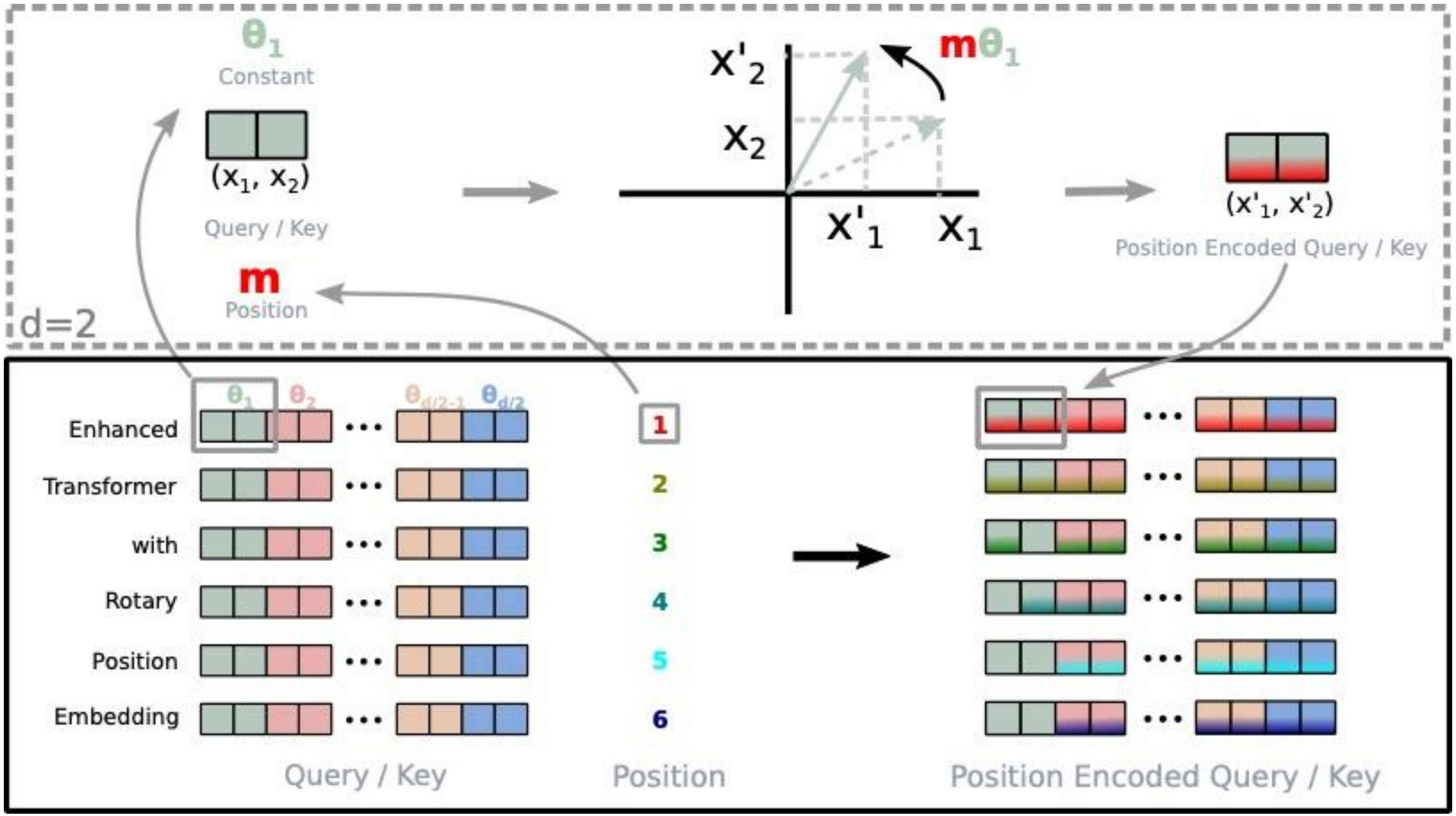


Image Credit: [24]

Rotary Positional Encoding

Code of RoPE:

```
import numpy as np
def rotary_positional_embedding(position, d_model):
    freqs = np.exp(np.linspace(0., -1., d_model // 2) * np.log(10000.))
    angles = position * freqs
    rotary_matrix = np.stack([np.sin(angles), np.cos(angles)], axis=-1)
    return rotary_matrix.reshape(-1, d_model)
```

RoPE rotates each token's embedding based on its position in the sequence.

Imagine the RoPE is like a clock with multiple hands. Each hand rotates at a different speed (different frequencies). Every token in your sequence corresponds to a specific clock hand.

Impact on dot-product in attention: closer positions -> closer angles -> higher dot product -> higher relevance.

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - **Architecture**
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Architecture

Modern LLMs architectures are based on transformers.

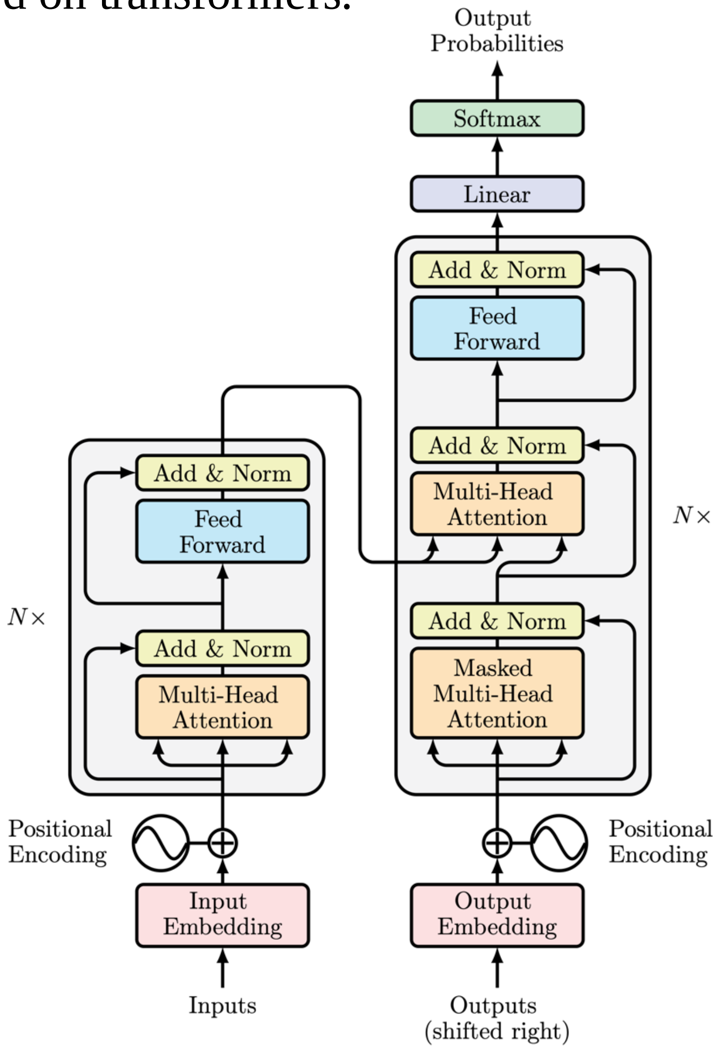


Image Credit: [7]

Architecture

Type 1: encoder-only.

These LMs generate contextual embeddings from given inputs.

$$x_{1:L} \Rightarrow \phi(x_{1:L}),$$

where $\phi : V^L \rightarrow \mathbb{R}^{d \times L}$ is the embedding function for input tokens.

Use of encoder-only LMs:

- Sentiment analysis

$$[[\mathrm{CLS}], \mathrm{the}, \mathrm{movie}, \mathrm{was}, \mathrm{great}] \Rightarrow \mathrm{positive}.$$

- Natural language inference

$$[[\mathrm{CLS}], \mathrm{all}, \mathrm{animals}, \mathrm{breathe}, [\mathrm{SEP}], \mathrm{cats}, \mathrm{breathe}] \Rightarrow \mathrm{entailment}.$$

Advantage: **bidirectional** context embeddings for each token in the input sequence.
Limit: cannot directly generate text and require specific training objectives.

Architecture

Type 1: encoder-only.

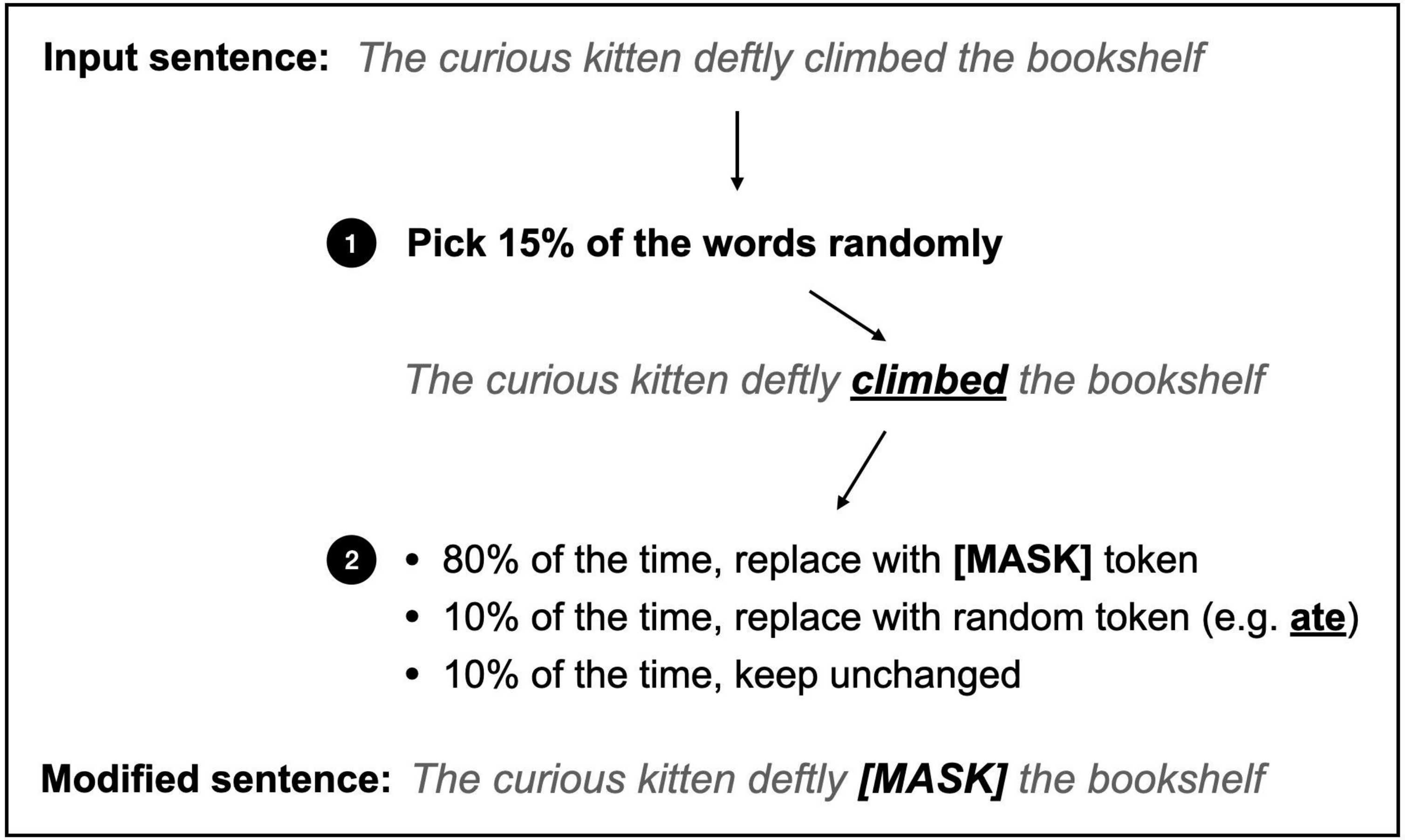


Image Credit: [8]

Architecture

Type 2: decoder-only.

They are standard autoregressive LMs that generate both contextual embedding and a conditional distribution for next token.

$$x_{1:i} \Rightarrow \phi(x_{1:i}), p(x_{i+1} \mid x_{1:i}).$$

Use of decoder-only LMs:

- Text autocomplete

$$[[\mathrm{CLS}], \mathrm{the}, \mathrm{movie}, \mathrm{was}] \Rightarrow \mathrm{great}$$

Advantage: natural text generation.
Limit: **unidirectional** context embedding depending on the left part $x_{1:i-1}$.

Architecture

Type 3: encoder-decoder.

They use bidirectional contextual embeddings and can naturally generate next token as output.

$$x_{1:L} \Rightarrow \phi(x_{1:L}), p(y_{1:L} \mid \phi(x_{1:L})).$$

Use of decoder-only LMs:

- Table-to-text generation

$$[\text{name}, :, \text{Clowns}, —, \text{eatType}, :, \text{coffee}, \text{shop}] \Rightarrow [\text{Clowns}, \text{is}, \text{a}, \text{coffee}, \text{shop}].$$

Advantage: **bidirectional** context embeddings; natural generation of text.
Limit: require specific training objectives.

Architecture

Type 3: encoder-decoder.

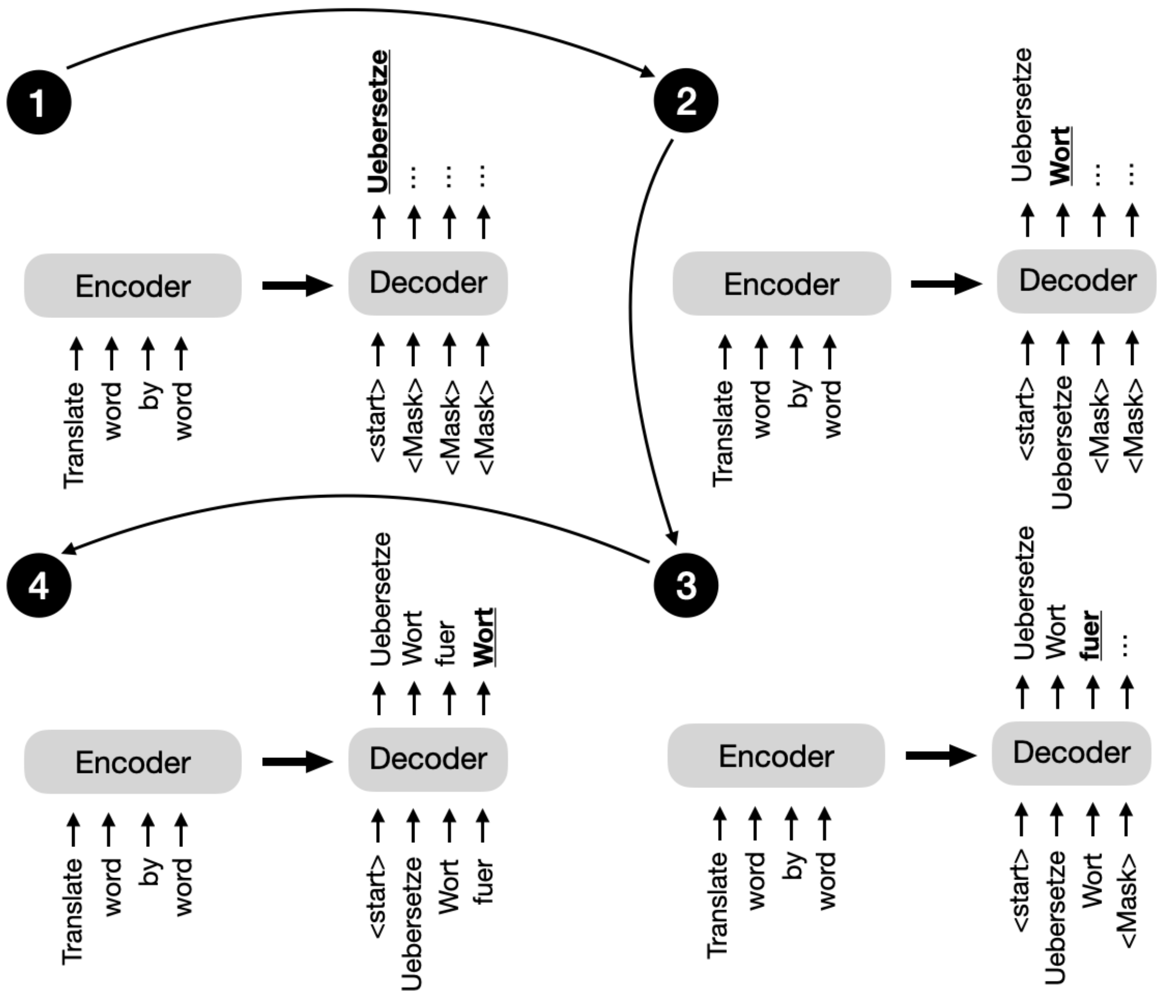
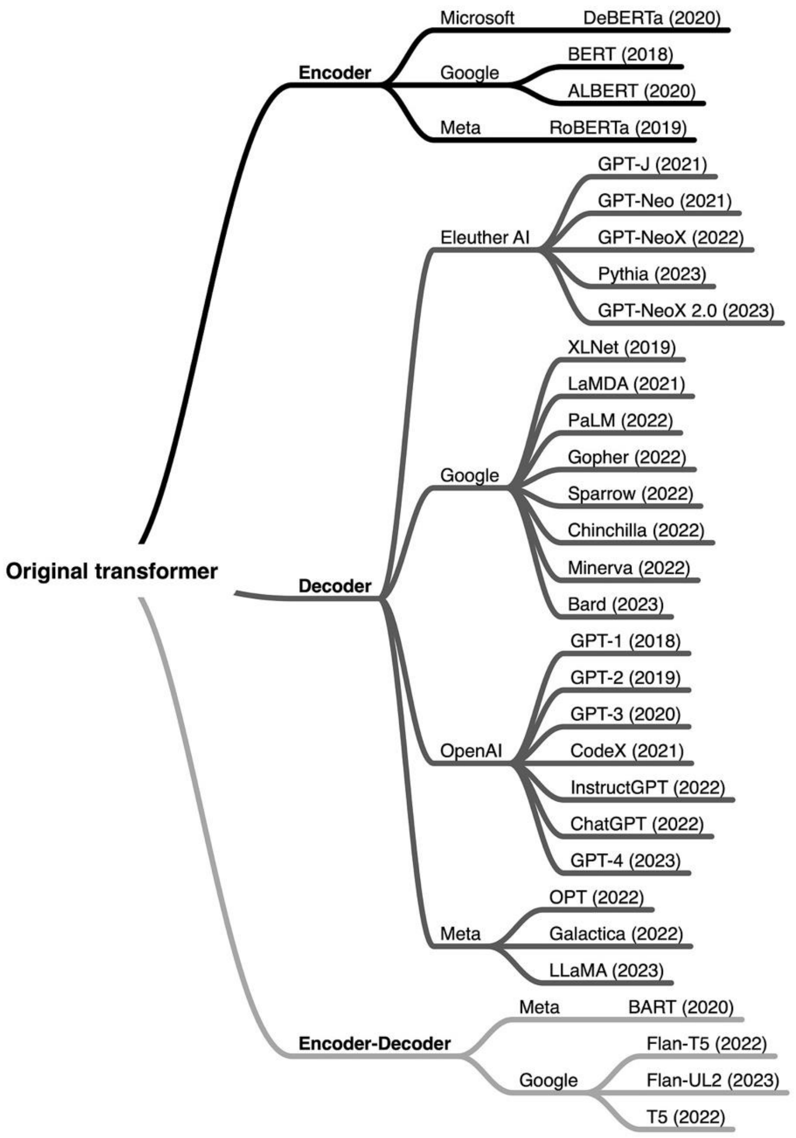


Image Credit: [8]

Architecture



Powerful conversional LLMs (e.g., ChatGPT, LLaMA) are mainly driven by decoder-only models.

Image Credit: [8]

Outline

- Introduction & Background
- Models
    - Tokenization
    - Rotary Positional Encoding
    - Architecture
- **Sampling**
- Training & Scaling Law
- Finetuning
    - Low Rank Adaptation (LoRA)
    - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
    - Zero/Few-shot Prompting
    - Chain of Thought (CoT) Prompting

Sampling

Suppose we train a decoder-only LLM like GPT-3, how can we generate next token one by one?

Sampling

Suppose we train a decoder-only LLM like GPT-3, how can we generate next token one by one?

- Greedy Sampling
- Beam Search
- Top-K
- Nucleus Sampling

……

Greedy Sampling

Denoting the model as $P_\theta(X_t|X_{<t})$, we "sample" the token with maximum conditional probability:

Algorithm 1 Greedy Sampling

1: Special start token x_0, vocabulary V, sequence length T
2: $S = [x_0]$
3: **for** $t \leftarrow 1$ **to** T **do**
4: $\quad x_t = \text{argmax}_{v \in V} \quad P_\theta\left(X_t = v \mid X_{<t} = S\right)$
5: $\quad S = [S, x_t]$ ▷ Concatenate the new token
6: **end for**
7: **return** S

Beam Search

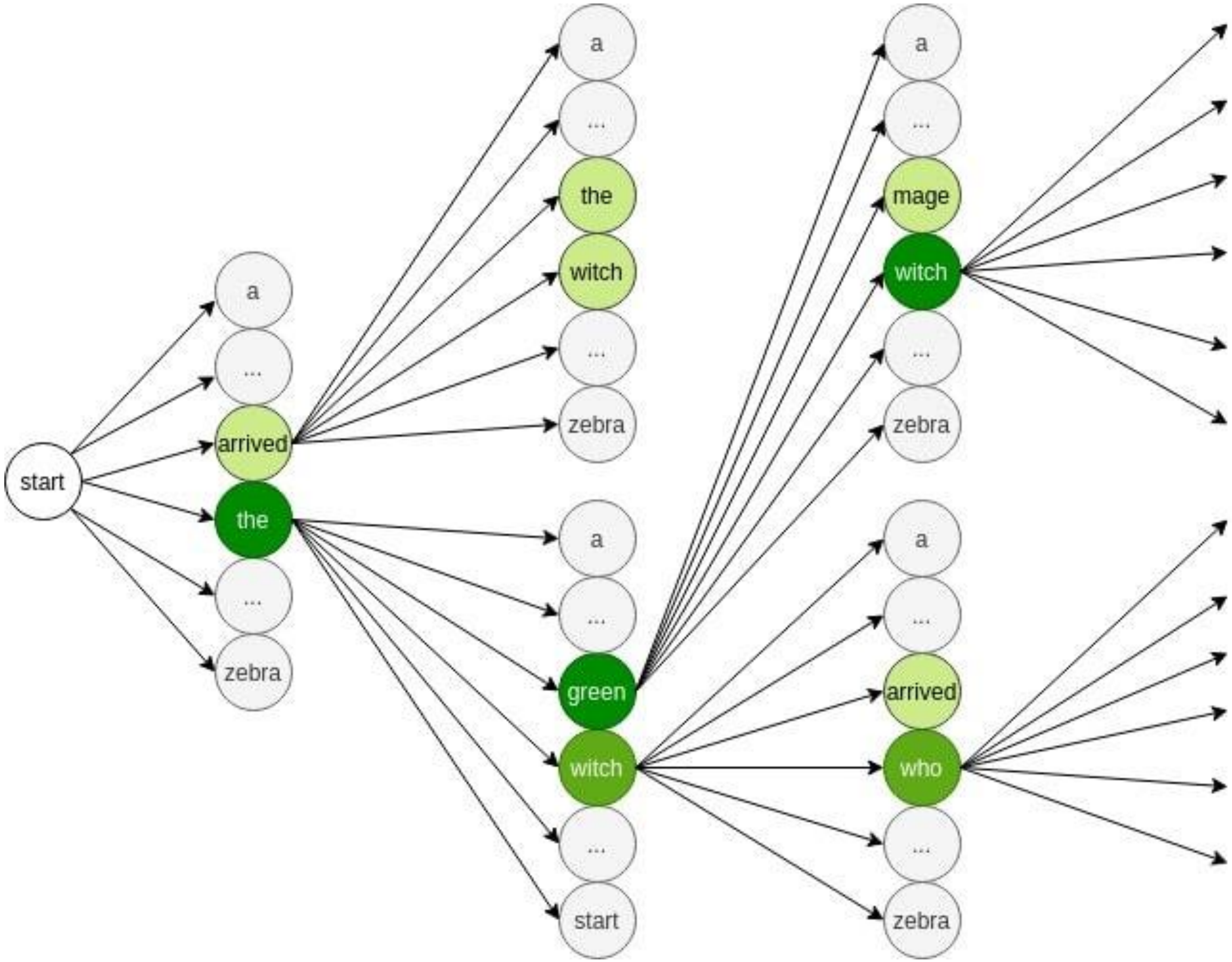


Image Credit: [9]

Beam Search

Denoting the model as $P_\theta(X_t|X_{<t})$, we have

Algorithm 2 Beam Search

Special start token x_0, beam size B, vocabulary V, sequence length T

$S = \{\underbrace{[x_0], \ldots, [x_0]}_{B}\}$

for $t \leftarrow 1$ **to** T **do**

  $C = \{\}$

  for $i \leftarrow 1$ **to** B **do**

    $\mathcal{N} = \text{argsort}_{v \in V} \quad P_\theta\left(X_t = v \mid X_{<t} = S[i]\right)$ ▷ Descending order

    for $j \leftarrow 1$ **to** B **do** ▷ Take top B tokens

      $C = C \cup \{[S[i], V[\mathcal{N}[j]]]\}$ ▷ Concatenate with existing sequence

    end for

  end for

  $\mathcal{C} = \text{argsort}_{c \in C} \quad P_\theta(X_{\leq t} = c)$ ▷ Descending order

  $S = \{C[\mathcal{C}[j]] | j = 1, \ldots, B\}$ ▷ Take top B subsequences

end for

return S

Top-K Sampling

Denoting the model as $P_\theta(X_t|X_{<t})$, we restrict the support to top-K candidate tokens:

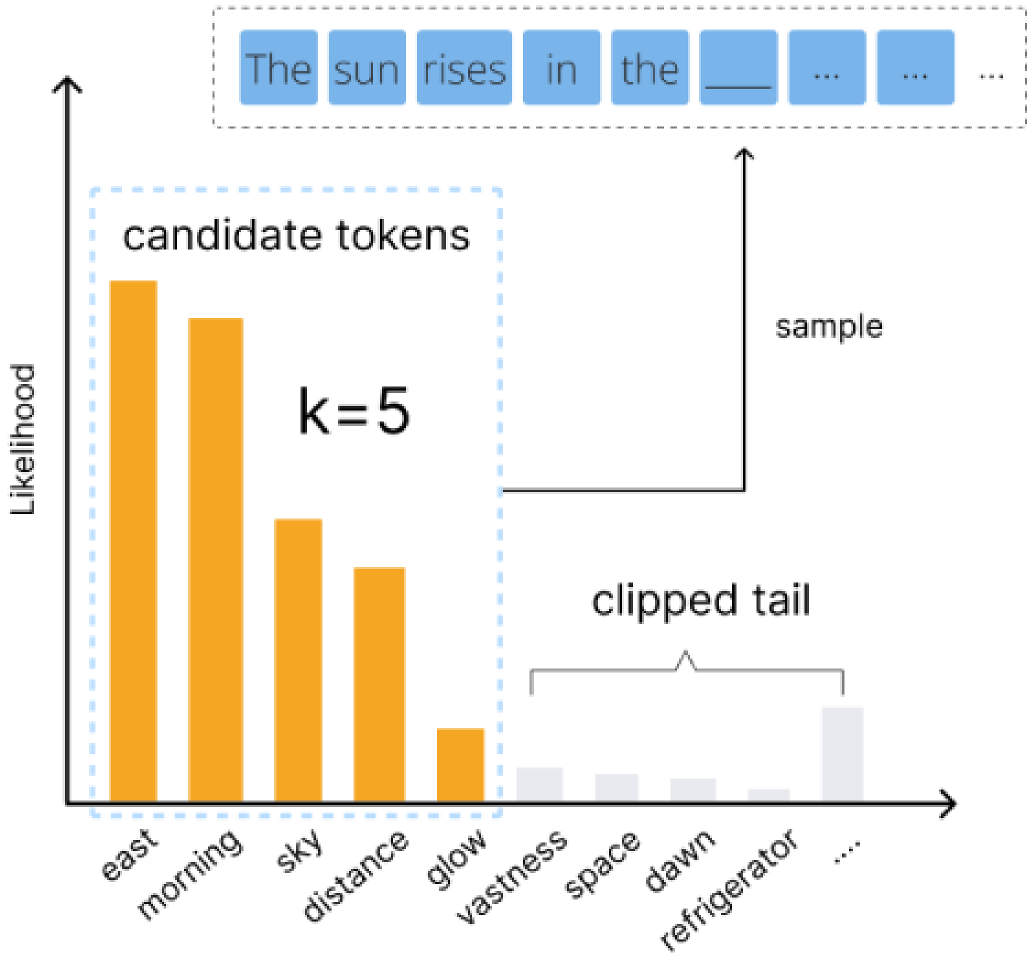


Image Credit: [10]

Top-K Sampling

Denoting the model as $P_\theta(X_t|X_{<t})$, we have

Algorithm 3 Top-K Sampling

Special start token x_0, vocabulary V, support size K, sequence length T
$S = [x_0]$
for $t \leftarrow 1$ **to** T **do**
  $\mathcal{N} = \text{argsort}_{v \in V} \quad P_\theta\left(X_t = v \mid X_{<t} = S\right)$ ▷ Descending order
  for $i \leftarrow 1$ **to** K **do**
    $\bar{P}\left(X_t = V[\mathcal{N}[i]] \mid X_{<t} = S\right) = \frac{P_\theta(X_t = V[\mathcal{N}[i]] | X_{<t} = S)}{\sum_{j=1}^{K} P_\theta(X_t = V[\mathcal{N}[j]] | X_{<t} = S)}$
  end for
  $x_t \sim \bar{P}$
  $S = [S, x_t]$ ▷ Concatenate the new token
end for
return S

Nucleus (Top-P) Sampling

Following top-K sampling, nucleus sampling [11] dynamically changes K so that their probabilities sum exceeds some threshold:

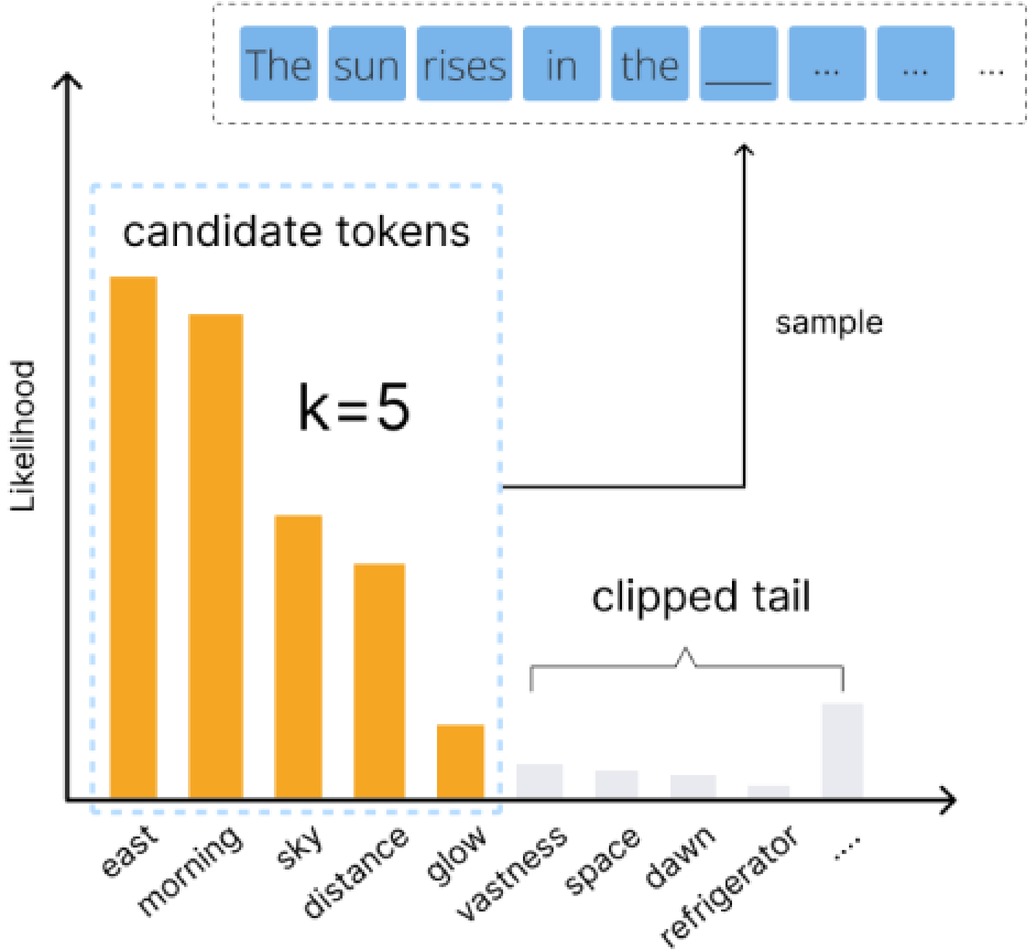


Image Credit: [10]

Nucleus (Top-P) Sampling

Denoting the model as $P_\theta(X_t|X_{<t})$, we have

Algorithm 4 Nucleus Sampling

1: Special start token x_0, vocabulary V, threshold $\rho \in (0,1)$, sequence length T
2: $S = [x_0]$
3: **for** $t \leftarrow 1$ **to** T **do**
4: $\quad \mathcal{N} = \mathrm{argsort}_{v \in V} \quad P_\theta\left(X_t = v \mid X_{<t} = S\right)$ ▷ Descending order
5: $\quad K = \min_k \quad \sum_{j=1}^{k} P_\theta\left(X_t = V[\mathcal{N}[j]] \mid X_{<t} = S\right) \geq \rho$
6: $\quad$ **for** $i \leftarrow 1$ **to** K **do**
7: $\quad\quad \bar{P}\left(X_t = V[\mathcal{N}[i]] \mid X_{<t} = S\right) = \frac{P_\theta(X_t = V[\mathcal{N}[i]] | X_{<t} = S)}{\sum_{j=1}^{K} P_\theta(X_t = V[\mathcal{N}[j]] | X_{<t} = S)}$
8: $\quad$ **end for**
9: $\quad x_t \sim \bar{P}$
10: $\quad S = [S, x_t]$ ▷ Concatenate the new token
11: **end for**
12: **return** S

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- **Training & Scaling Law**
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Loss Function

We train decoder-only LLMs (e.g., GPT3 [12]) to predict the next token by minimizing negative log likelihood:

$$L(\theta) = \frac{1}{B}\sum_{i=1}^{B} = -\log p_\theta(\boldsymbol{x}_T^i, \boldsymbol{x}_{T-1}^i, \ldots, \boldsymbol{x}_1^i) = \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{T} -\log p_\theta(\boldsymbol{x}_t^i | \boldsymbol{x}_{t-1}^i, \ldots, \boldsymbol{x}_1^i)$$

Loss Function

We train decoder-only LLMs (e.g., GPT3 [12]) to predict the next token by minimizing negative log likelihood:

$$L(\theta) = \frac{1}{B}\sum_{i=1}^{B} = -\log p_\theta(\boldsymbol{x}_T^i, \boldsymbol{x}_{T-1}^i, \ldots, \boldsymbol{x}_1^i) = \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{T} -\log p_\theta(\boldsymbol{x}_t^i|\boldsymbol{x}_{t-1}^i, \ldots, \boldsymbol{x}_1^i)$$

For encoder-only and encoder-decoder LLMs (e.g., BERT [13], BART [14], and T5 [15]), they do mostly masked language modeling, i.e., predicting the masked tokens:

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Image Credit: [15]

Scaling Law

Hyperparameter tuning for LLMs has a huge cost!

Scaling law [16, 17] allows fast prediction of model performances, e.g., validation loss L, from the dataset size D, computational cost C, and the number of parameters N.

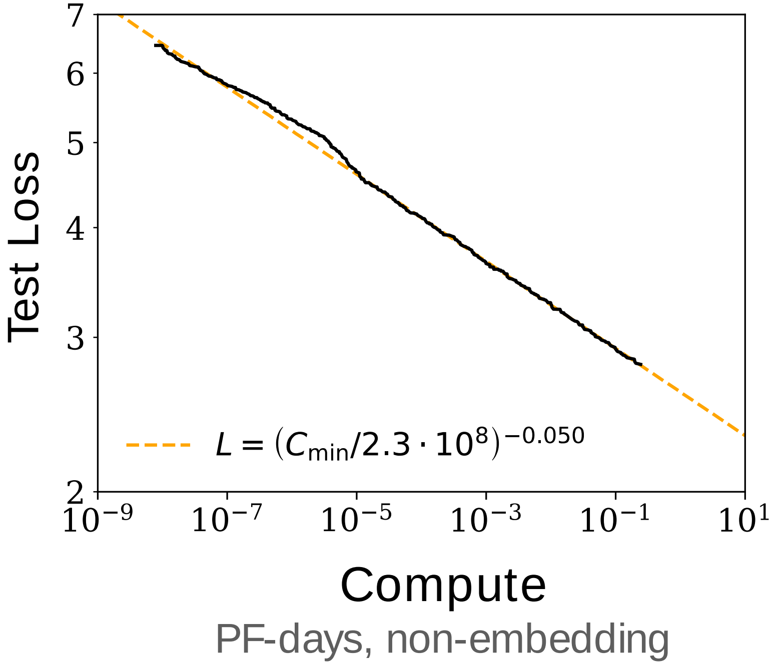


Image Credit: [17]

Scaling Law

Hyperparameter tuning for LLMs has a huge cost!

Scaling law [16, 17] allows fast prediction of model performances, e.g., validation loss L, from the dataset size D, computational cost C, and the number of parameters N.

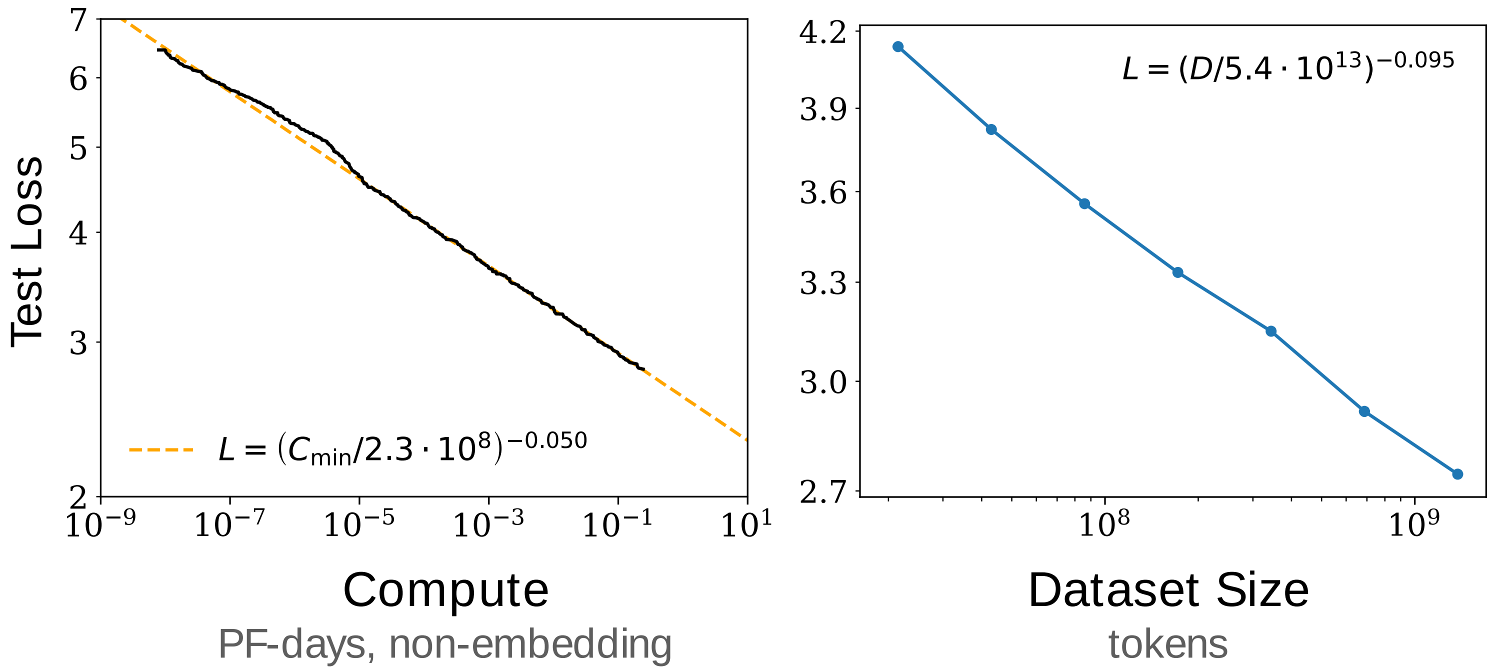


Image Credit: [17]

Scaling Law

Hyperparameter tuning for LLMs has a huge cost!

Scaling law [16, 17] allows fast prediction of model performances, e.g., validation loss L, from the dataset size D, computational cost C, and the number of parameters N.

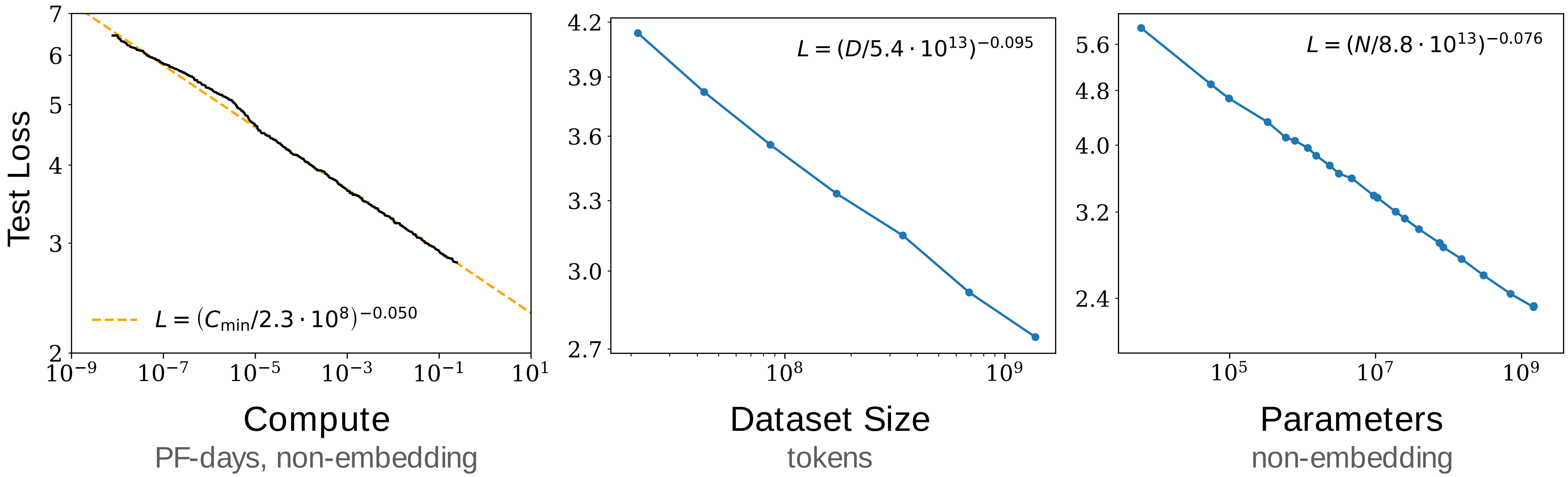
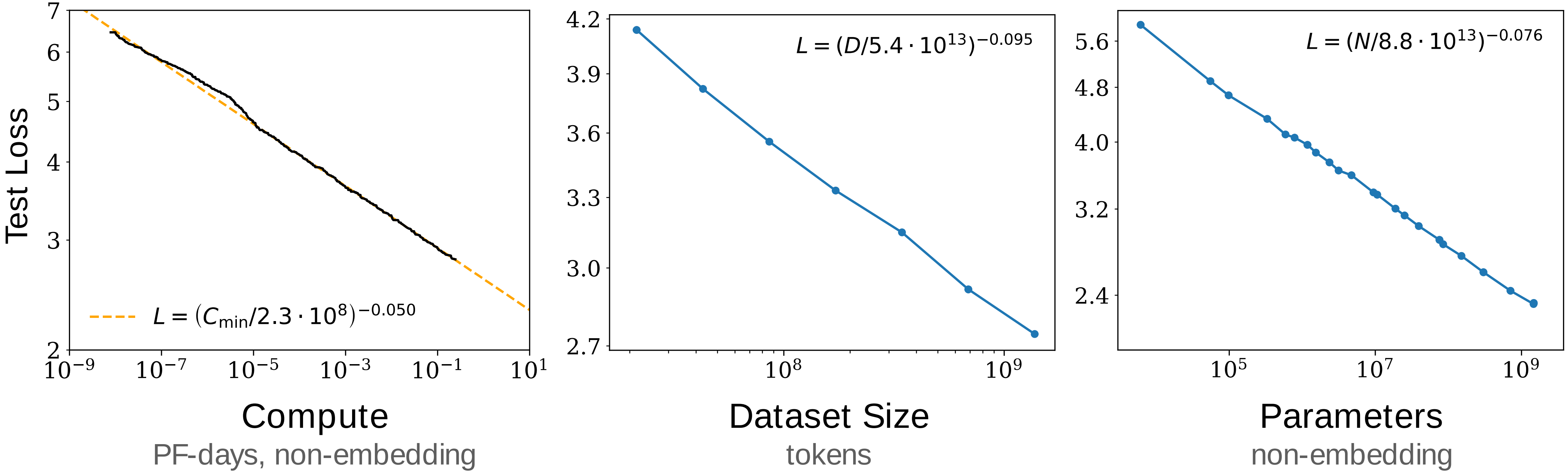


Image Credit: [17]

Scaling Law

Hyperparameter tuning for LLMs has a huge cost!

Scaling law [16, 17] allows fast prediction of model performances, e.g., validation loss L, from the dataset size D, computational cost C, and the number of parameters N.

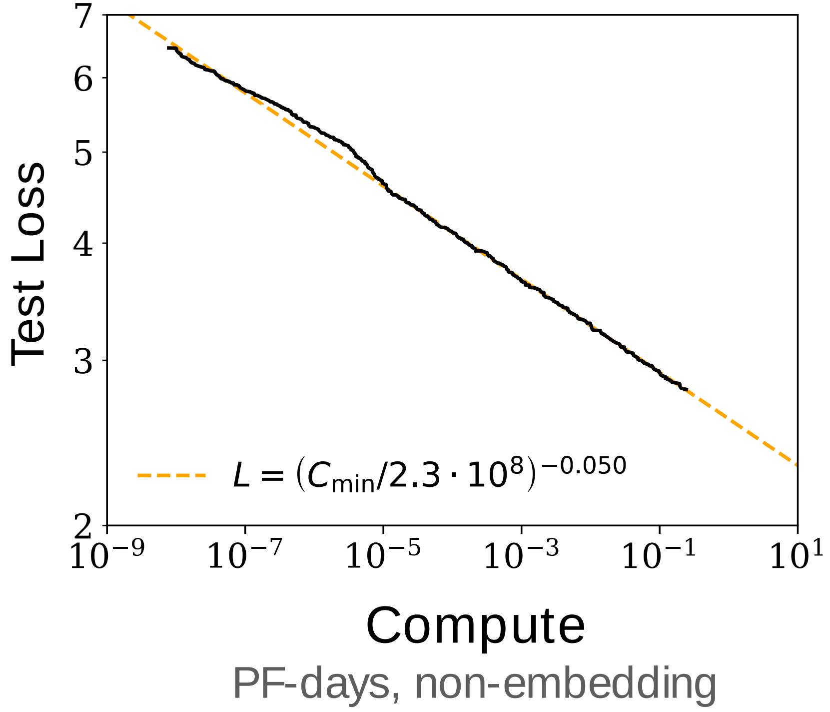


Power law $y = ax^k$ appears as straight lines in log-log plot!

Image Credit: [17]

Scaling Law

Many factors, e.g., the architecture, could affect the scaling law.



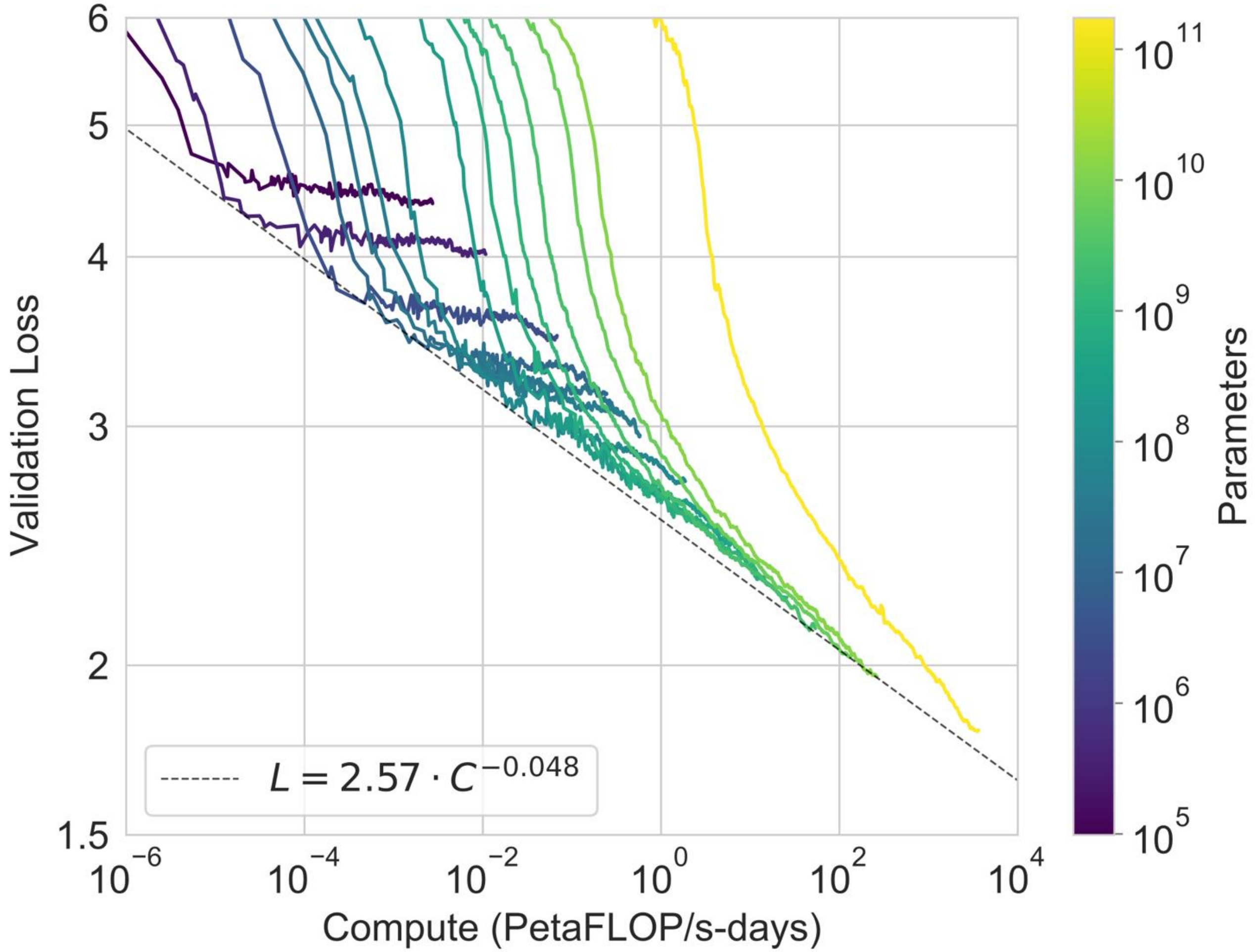
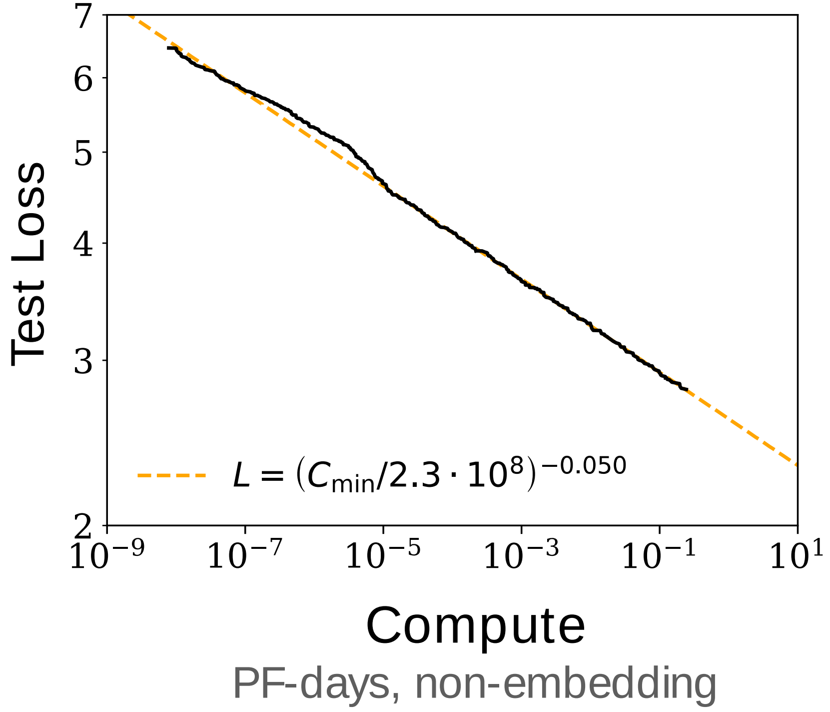


Image Credit: [17]

Scaling Law

Many factors, e.g., the architecture, could affect the scaling law.



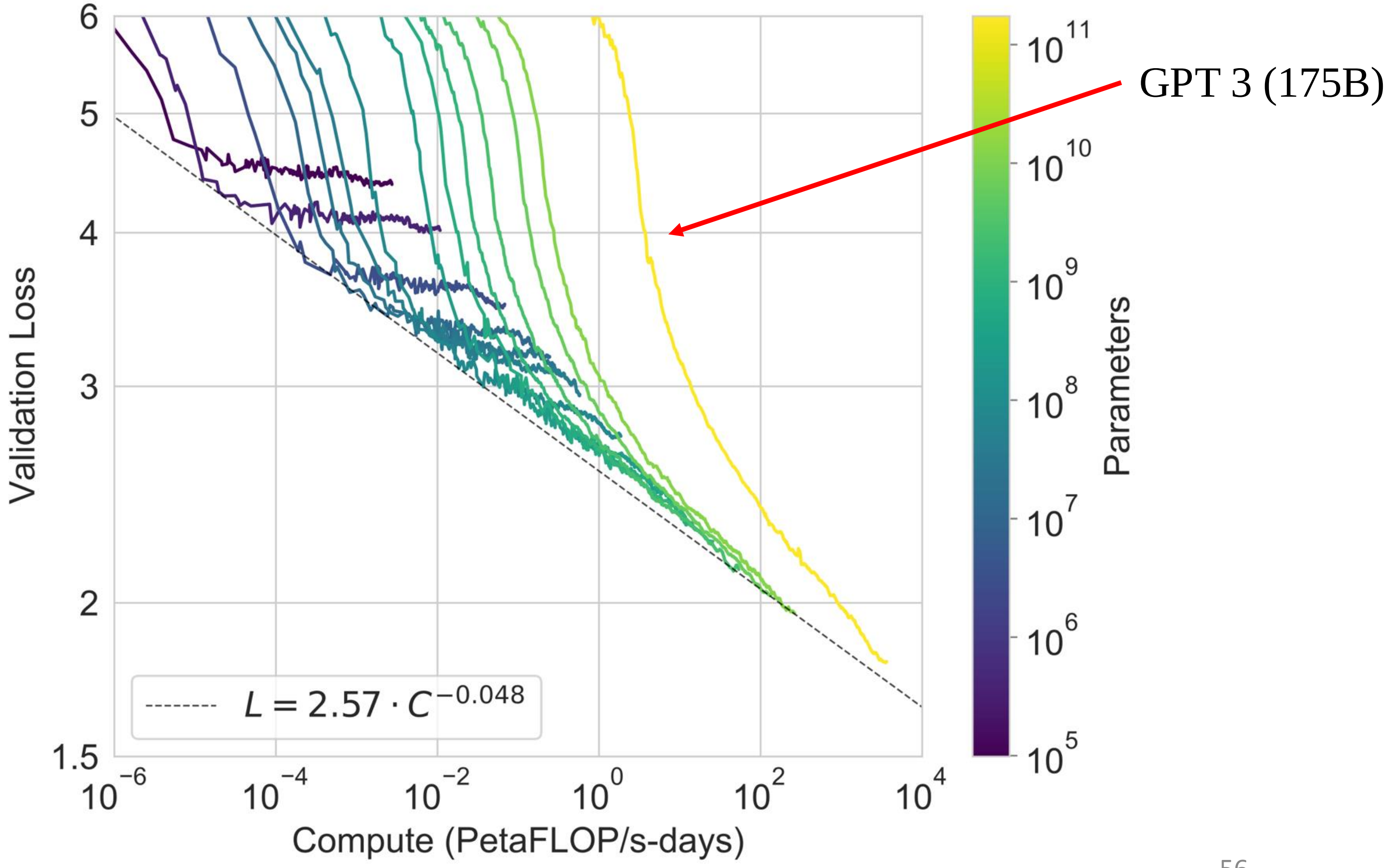
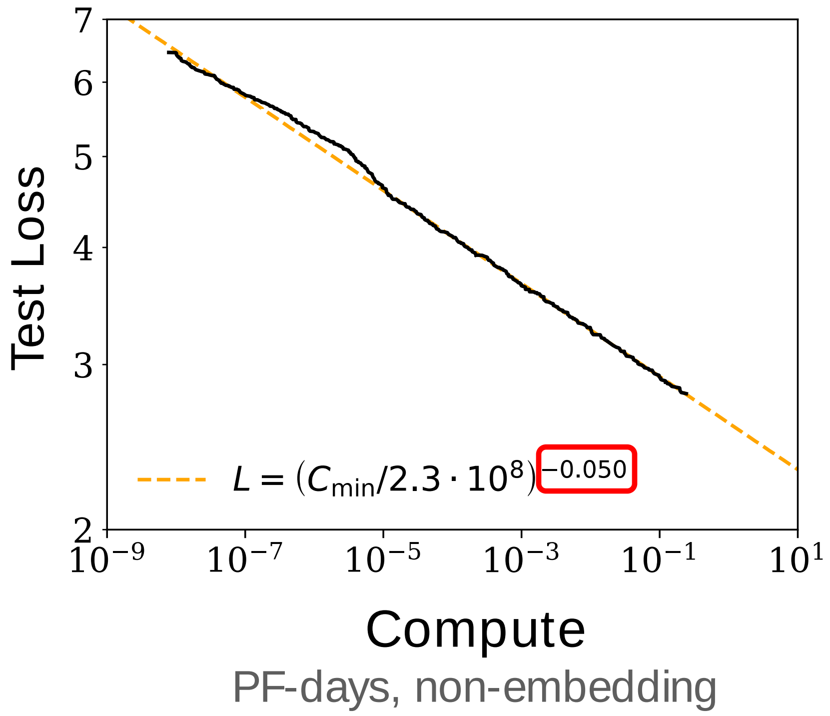


Image Credit: [17]

Scaling Law

Many factors, e.g., the architecture, could affect the scaling law. But the exponent seems quite stable!



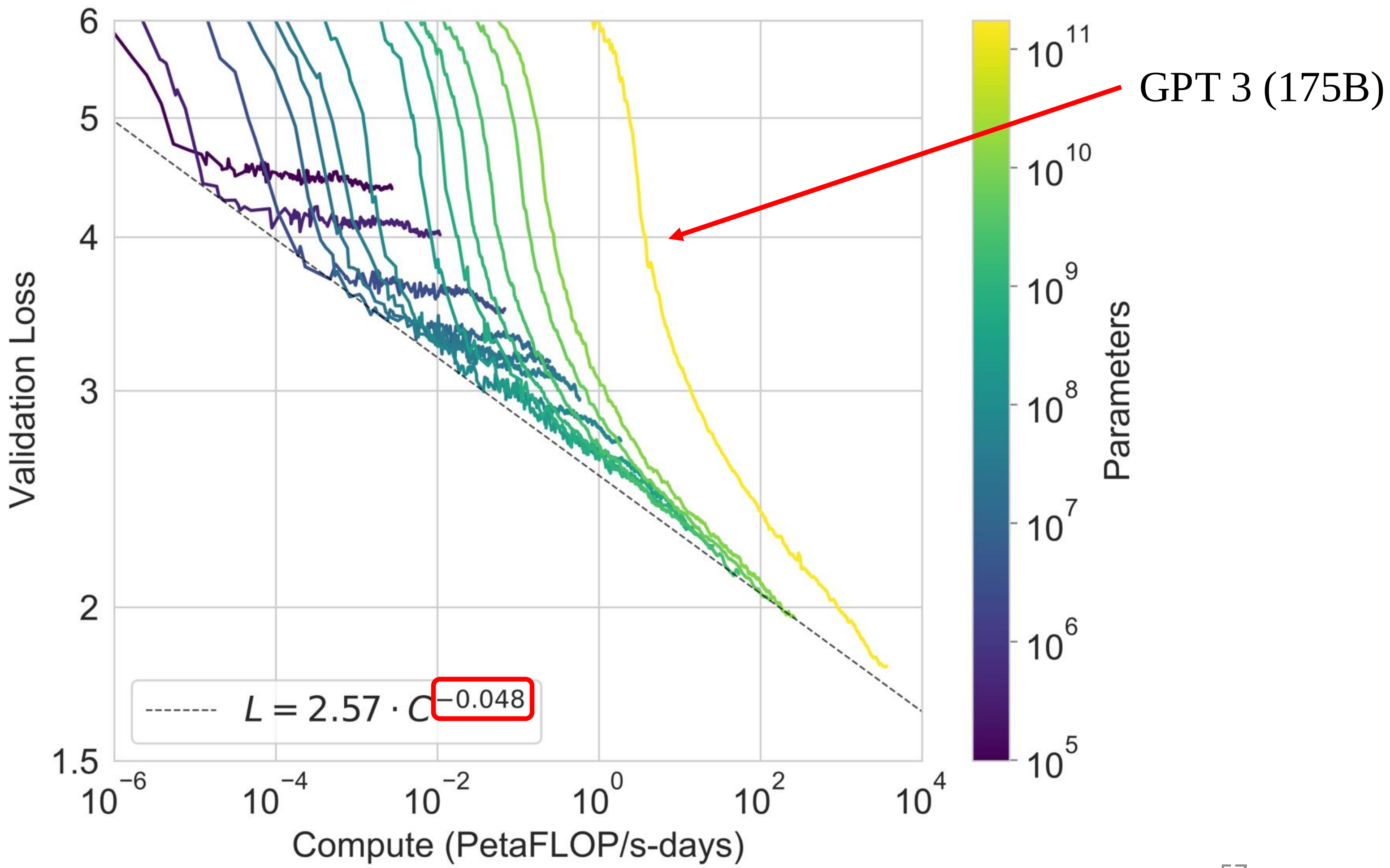


Image Credit: [17]

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - **Low Rank Adaptation (LoRA)**
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - Chain of Thought (CoT) Prompting

Low Rank Adaptation (LoRA)

Fine-tuning LLMs is computationally expensive!

When adapting LLMs to a specific task, pre-trained LLMs have a low ``intrinsic dimension" [18]

Low Rank Adaptation (LoRA)

Fine-tuning LLMs is computationally expensive!

When adapting LLMs to a specific task, pre-trained LLMs have a low ``intrinsic dimension” [18]

LoRA [19] thus learns a low-rank weight update:

$$\begin{aligned}\hat{W}x &= Wx + \Delta W x \\ &= Wx + BAx\end{aligned}$$

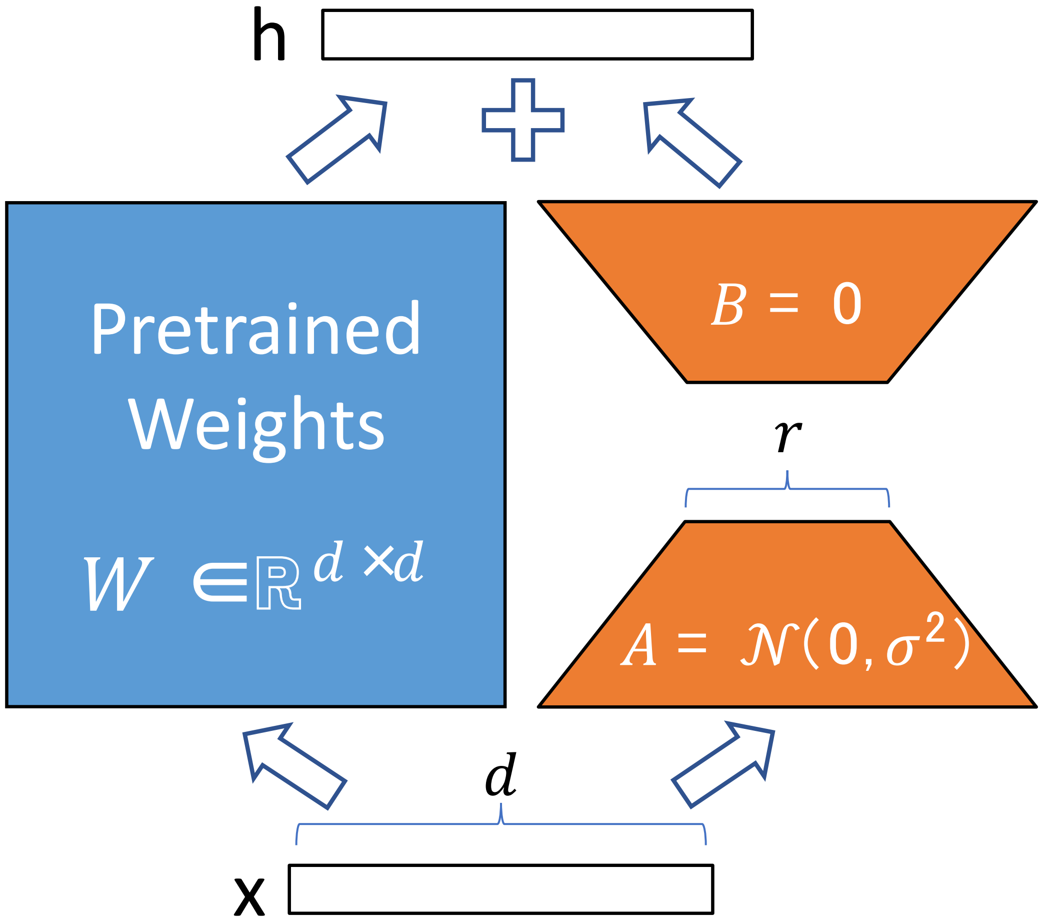


Image Credit: [19]

Low Rank Adaptation (LoRA)

Fine-tuning LLMs is computationally expensive!

When adapting LLMs to a specific task, pre-trained LLMs have a low ``intrinsic dimension" [18]

LoRA [19] thus learns a low-rank weight update:

$$\begin{aligned}\hat{W}x &= Wx + \Delta W x \\ &= Wx + BAx\end{aligned}$$

Frozen

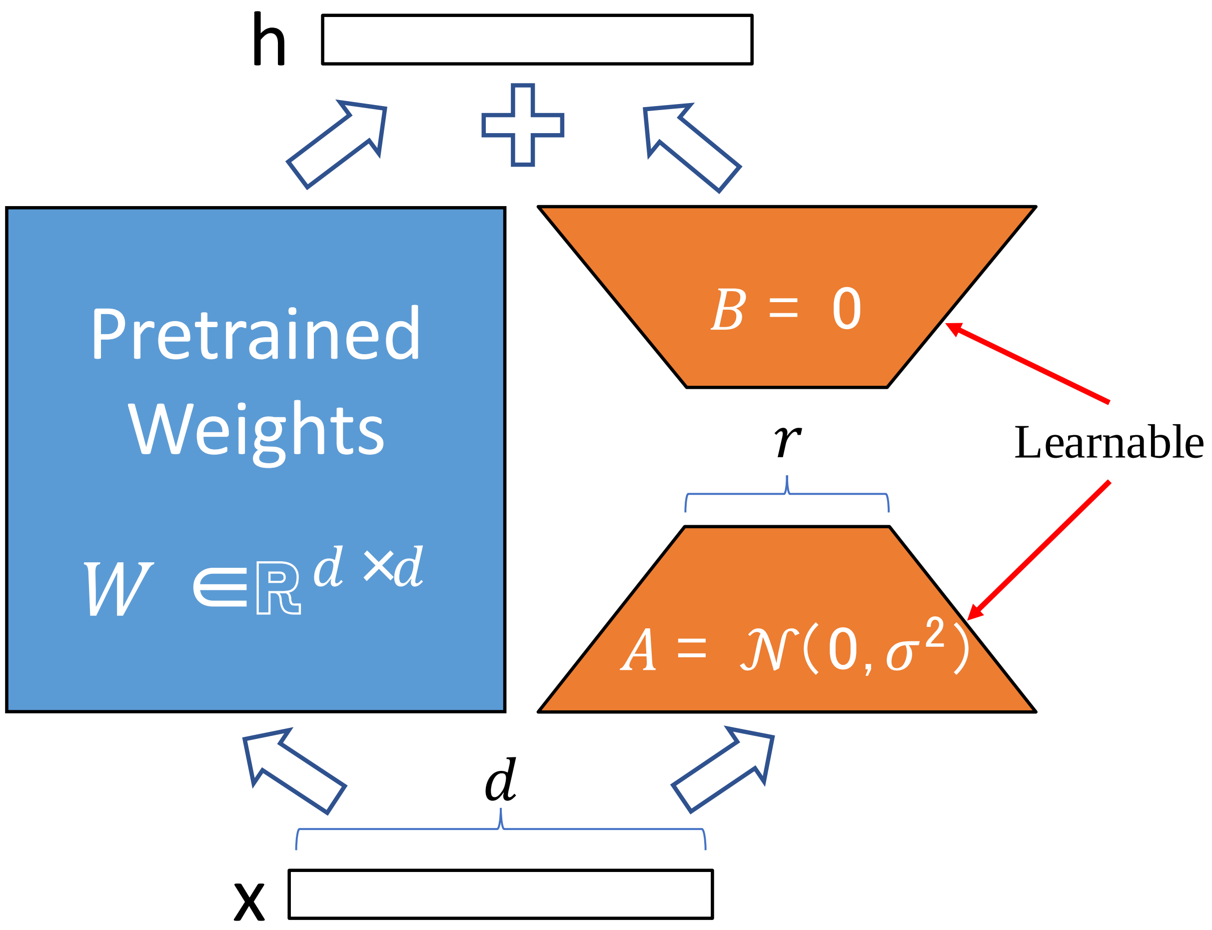


Image Credit: [19]

Outline

- Introduction & Background
- Models
    - Tokenization
    - Rotary Positional Encoding
    - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
    - Low Rank Adaptation (LoRA)
    - **Reinforcement Learning from Human Feedback (RLHF)**
- Prompting
    - Zero/Few-shot Prompting
    - Chain of Thought (CoT) Prompting

Reinforcement Learning from Human Feedback (RLHF)

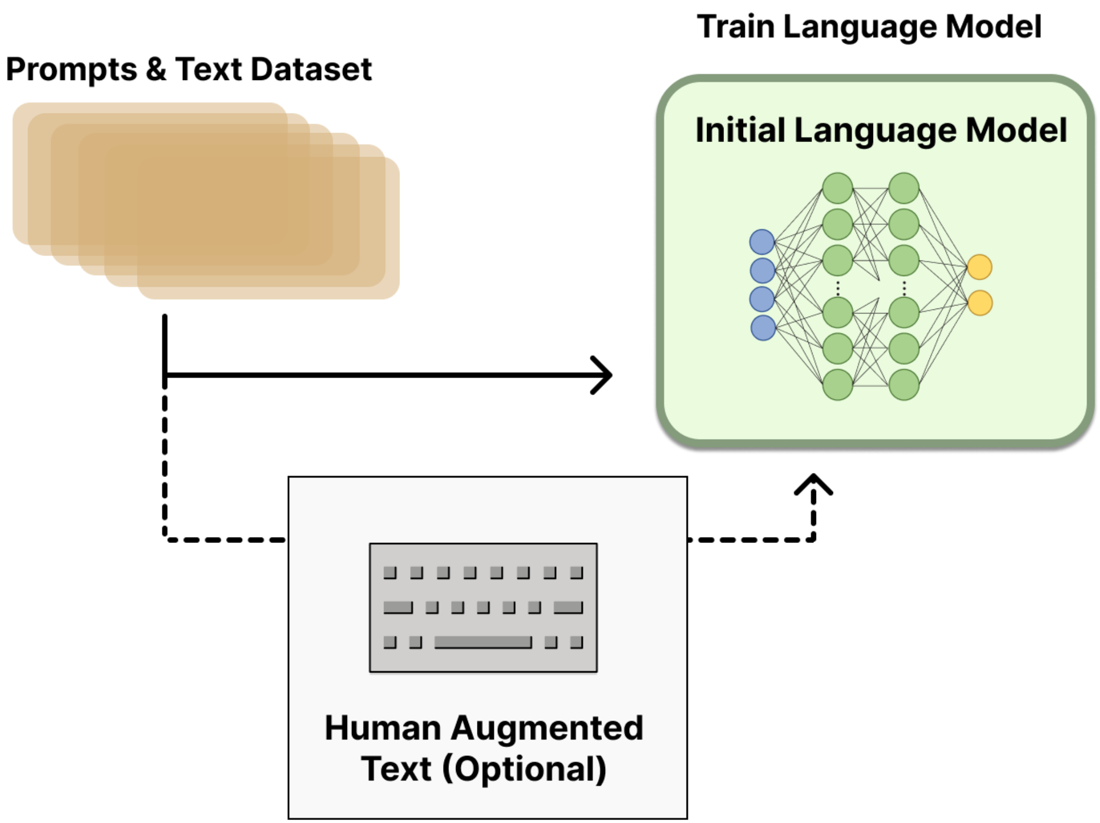
Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

Reinforcement Learning from Human Feedback (RLHF)

Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

It involves three steps:

- **Pretraining a LLM**



e.g., one curate a preferable text dataset

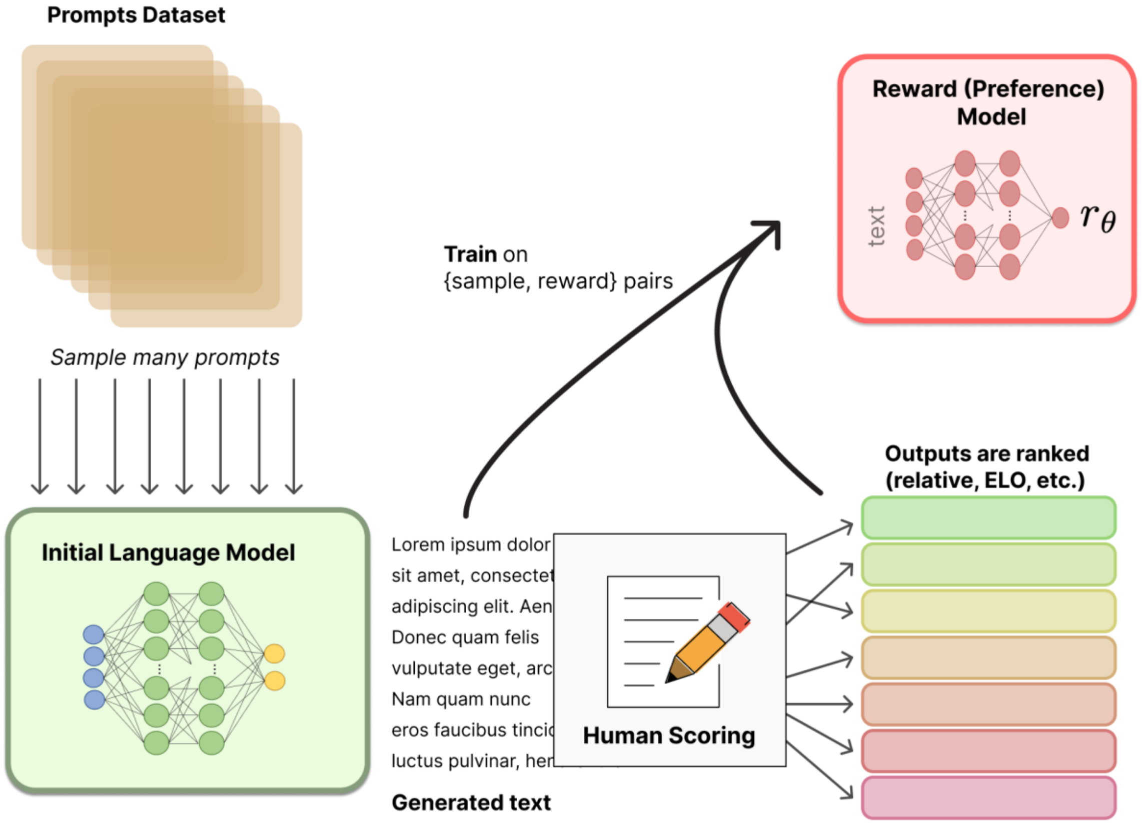
Image Credit: [21]

Reinforcement Learning from Human Feedback (RLHF)

Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

It involves three steps:

- Pretraining a LLM
- **Training a reward model**



- OpenAI uses 175B LM and 6B reward model
- Anthropic used LM and reward models from 10B to 52B
- DeepMind uses 70B Chinchilla models for both LM and reward

Image Credit: [21]

Reinforcement Learning from Human Feedback (RLHF)

Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

It involves three steps:

- Pretraining a LLM
- Training a reward model
- **Fine-tuning LLM with RL**

$$r = r_\theta - \lambda_{\mathrm{KL}} D_{\mathrm{KL}}$$

RL policy generates text, and that text is fed into the initial model to produce its relative probabilities for the KL penalty

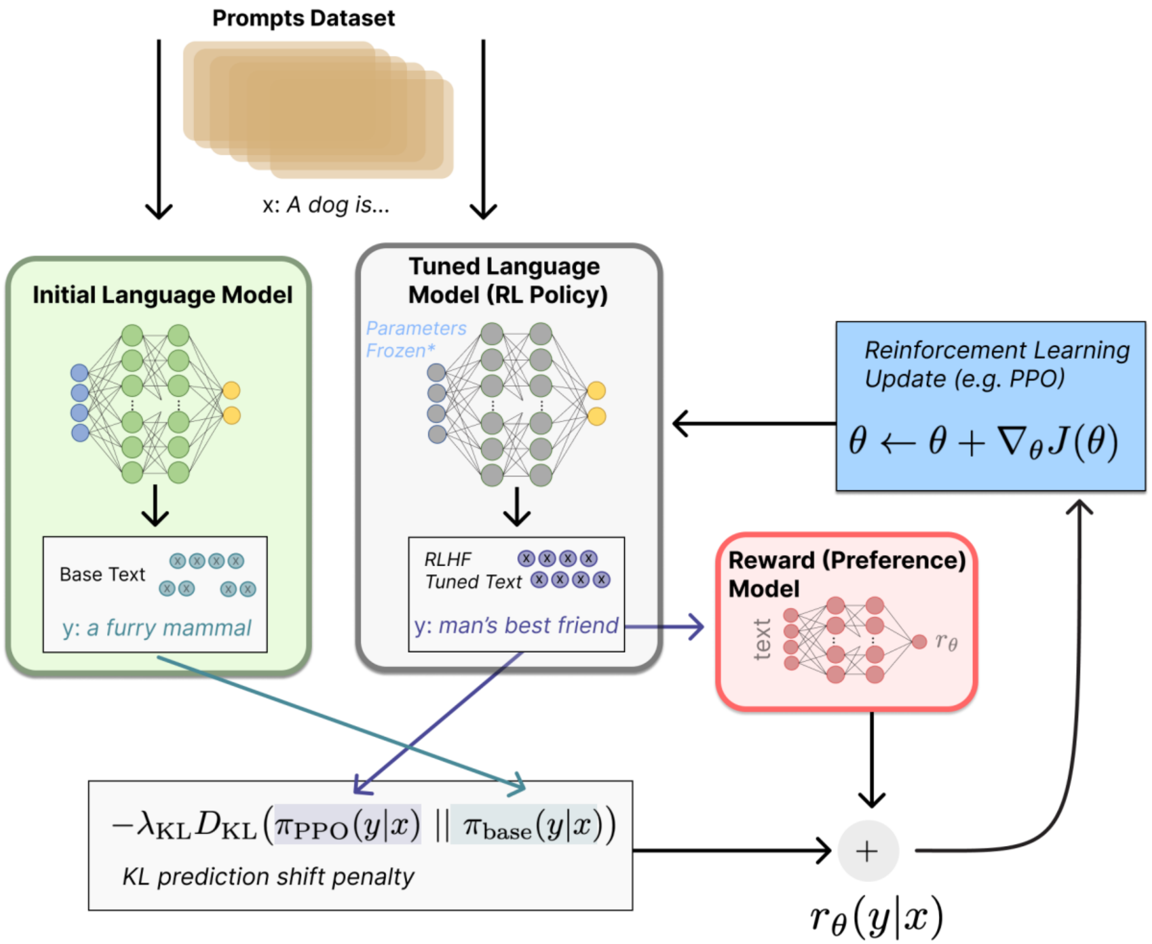


Image Credit: [21]

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - **Zero/Few-shot Prompting**
  - Chain of Thought (CoT) Prompting

Prompting

Prompts are the input to LLMs, of which the quality significantly affects the output of LLMs.

Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot Prompting

Prompts are the input to LLMs, of which the quality significantly affects the output of LLMs.

Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot prompting:

	Zero-shot, standard
In	Q: While shopping for music online, Janet bought 6 country albums and 2 pop albums. Each album came with a lyric sheet and had 9 songs. How many songs did Janet buy total? A:
Out	The answer is xxx

Ask it directly!

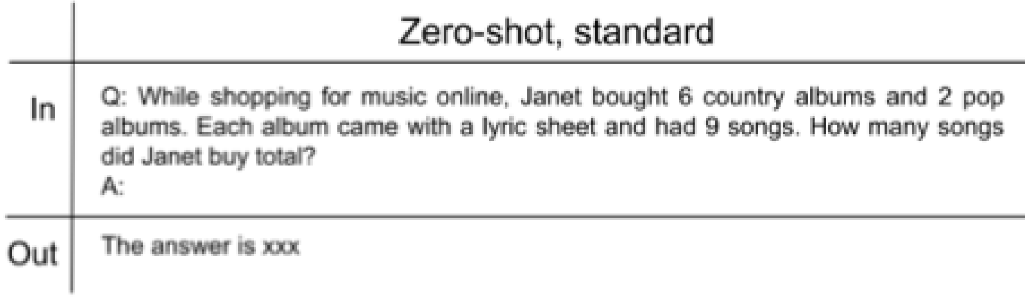
Image Credit: [22]

Zero/Few-shot Prompting

Prompts are the input to LLMs, of which the quality significantly affects the output of LLMs.

Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot prompting:



Ask it directly!

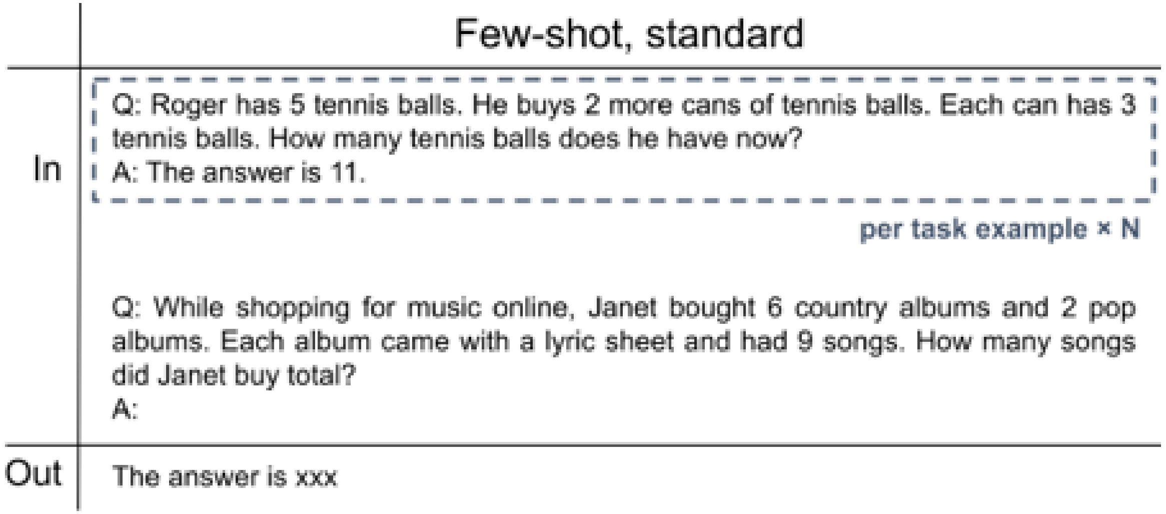


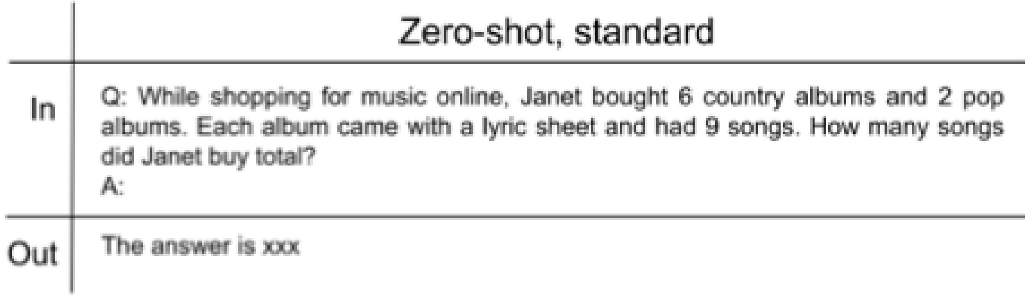
Image Credit: [22]

Zero/Few-shot Prompting

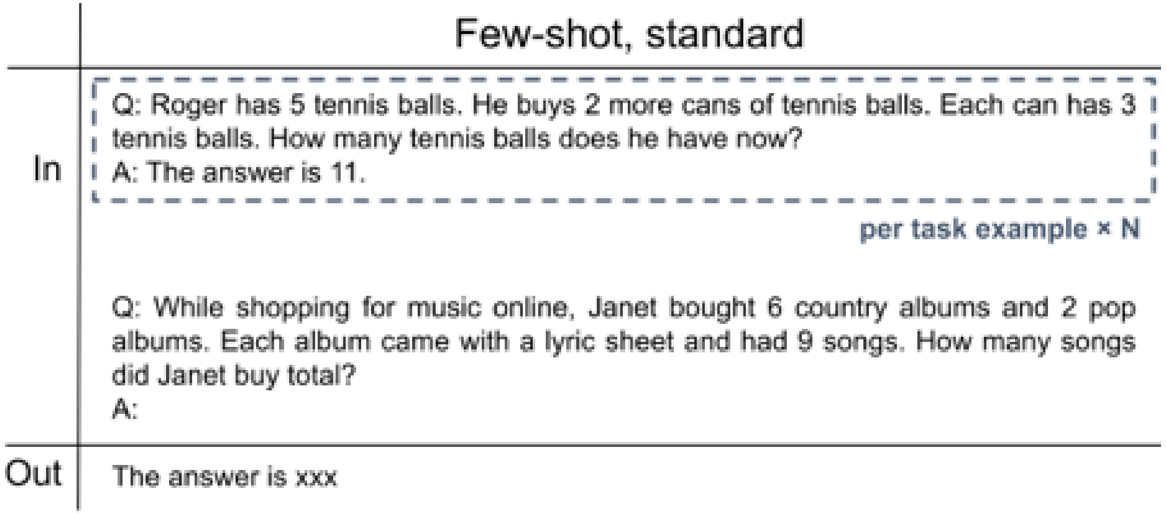
Prompts are the input to LLMs, of which the quality significantly affects the output of LLMs.

Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot prompting:



Ask it directly!



Ask with some guiding examples!

Image Credit: [22]

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Training & Scaling Law
- Finetuning
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting
  - Zero/Few-shot Prompting
  - **Chain of Thought (CoT) Prompting**

Chain-of-Thought (CoT) Prompting

CoT prompting [23] enables complex reasoning capabilities through intermediate reasoning steps:

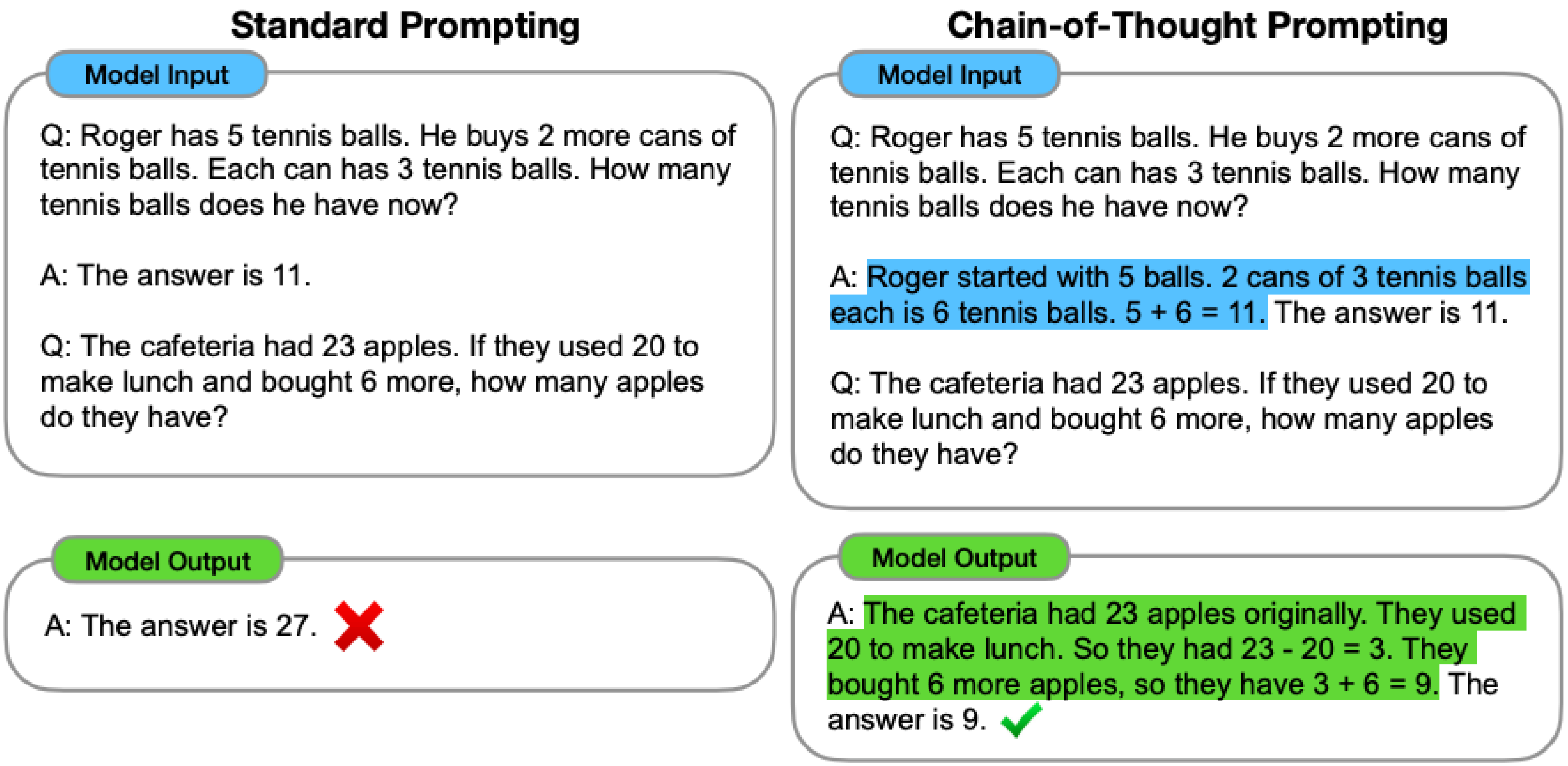


Image Credit: [23]

Chain-of-Thought (CoT) Prompting

CoT prompting [23] enables complex reasoning capabilities through intermediate reasoning steps:

Zero-shot CoT

Q: While shopping for music online, Janet bought 6 country albums and 2 pop albums. Each album came with a lyric sheet and had 9 songs. How many songs did Janet buy total?
A: Let's think step by step.

Janet bought 6 country albums and 2 pop albums. That is a total of 8 albums. Each album has 9 songs. So 8 * 9 = 72. The answer is 72

Few-shot CoT

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.

per task chain of thought example × N

Q: While shopping for music online, Janet bought 6 country albums and 2 pop albums. Each album came with a lyric sheet and had 9 songs. How many songs did Janet buy total?
A:

Janet bought 6 country albums and 2 pop albums. That is a total of 8 albums. Each album has 9 songs. So 8 * 9 = 72. The answer is 72

Image Credit: [22]

References

[1] https://informationisbeautiful.net/visualizations/the-rise-of-generative-ai-large-language-models-llms-like-chatgpt/

[2] Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S. and Nori, H., 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712.

[3] Trinh, T.H., Wu, Y., Le, Q.V., He, H. and Luong, T., 2024. Solving olympiad geometry without human demonstrations. Nature, 625(7995), pp.476-482.

[4] https://openai.com/blog/planning-for-agi-and-beyond

[5] https://platform.openai.com/tokenizer

[6] https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[8] https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder

[9] https://www.baeldung.com/cs/beam-search

[10] https://www.megaputer.com/mastering-language-models-a-deep-dive-into-input-parameters/

[11] Holtzman, A., Buys, J., Du, L., Forbes, M. and Choi, Y., 2019. The curious case of neural text degeneration. arXiv preprint arXiv:1904.09751.

References

[12] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. and Agarwal, S., 2020. Language models are few-shot learners. Advances in neural information processing systems, 33, pp.1877-1901.

[13] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[14] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V. and Zettlemoyer, L., 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461.

[15] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. and Liu, P.J., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research, 21(140), pp.1-67.

[16] Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M.M.A., Yang, Y. and Zhou, Y., 2017. Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409.

[17] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D., 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.

[18] Aghajanyan, A., Zettlemoyer, L. and Gupta, S., 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. arXiv preprint arXiv:2012.13255.

[19] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W., 2021. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685.

[20] https://huggingface.co/blog/rlhf

References

[21] Ziegler, D.M., Stiennon, N., Wu, J., Brown, T.B., Radford, A., Amodei, D., Christiano, P. and Irving, G., 1909. Fine-tuning language models from human preferences. arXiv 2019. arXiv preprint arXiv:1909.08593.

[22] Kojima, T., Gu, S.S., Reid, M., Matsuo, Y. and Iwasawa, Y., 2022. Large language models are zero-shot reasoners. Advances in neural information processing systems, 35, pp.22199-22213.

[23] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V. and Zhou, D., 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35, pp.24824-24837.

[24] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W. and Liu, Y., 2024. Roformer: Enhanced transformer with rotary position embedding. Neurocomputing, 568, p.127063.

Questions?