EECE 571F: Advanced Topics in Deep Learning

Lecture 6: Large Language Models

Renjie Liao, Qi Yan

University of British Columbia

Winter, Term 2, 2025

Outline

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:

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

The 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.

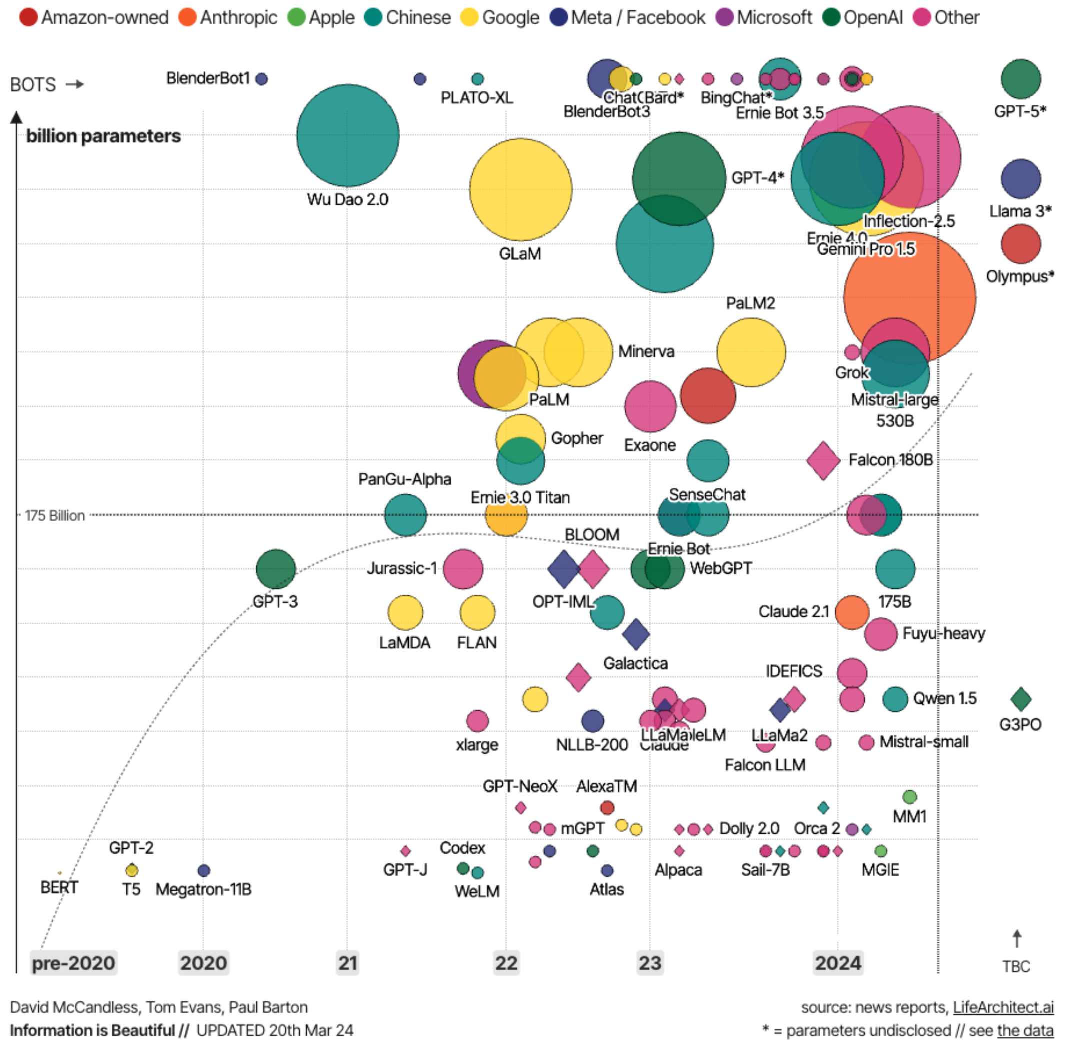
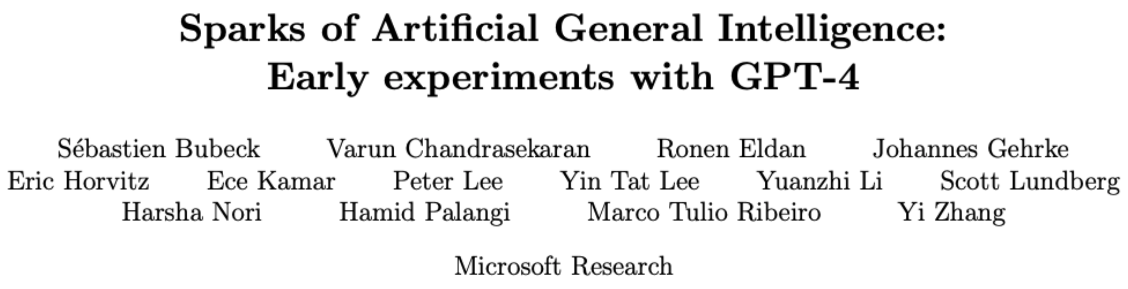


Image Credit: [1]

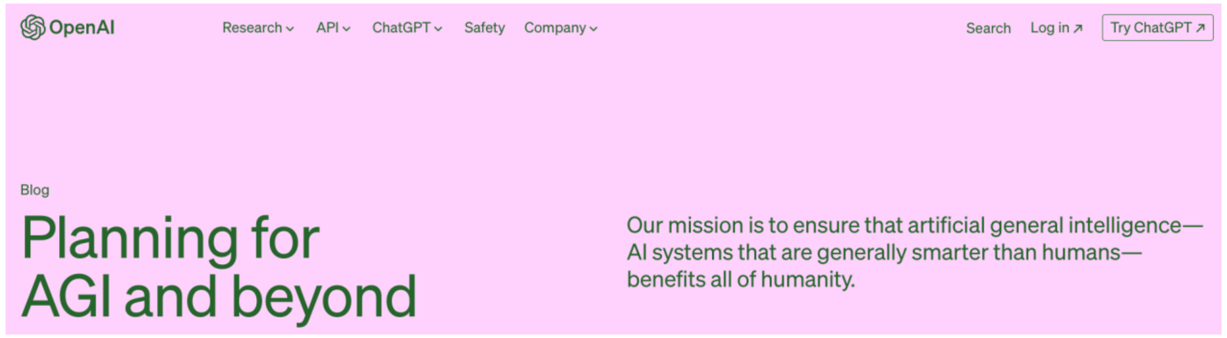
Introduction & Background

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

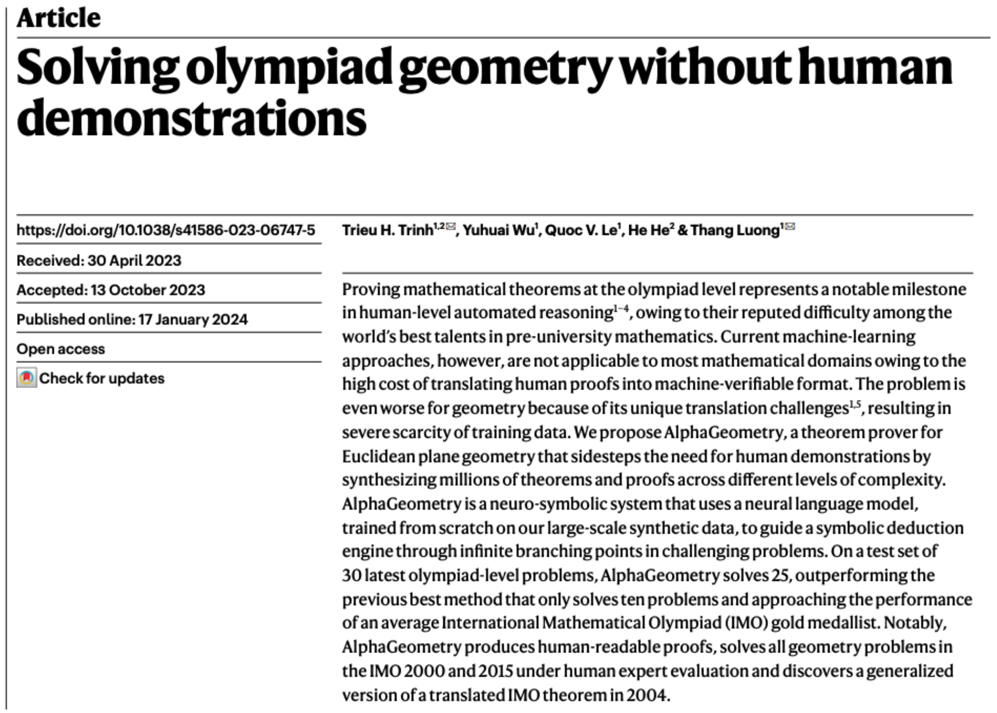
Microsoft Research



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

Tokenization

Recall the previous example on vocabulary:

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

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

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

Tokenization

Recall the previous example on vocabulary:

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

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

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

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:

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

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

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

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.

```
1: **Input:** A training corpus composed of character sequences.
2: **Initialization:** Treat each character as an individual token. Establish initial vocabulary $V$ as the set of distinct characters.
3: **while** $V$ needs expansion **do**
4:     Identify the most frequently co-occurring pair of elements $x, x' \in V$.
5:     Replace every instance of $x, x'$ with a new symbol $xx'$.
6:     Add the new symbol $xx'$ to $V$.
7: **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 ⇒ *th*
  th, e ⇒ *the*
  c, a ⇒ *ca*

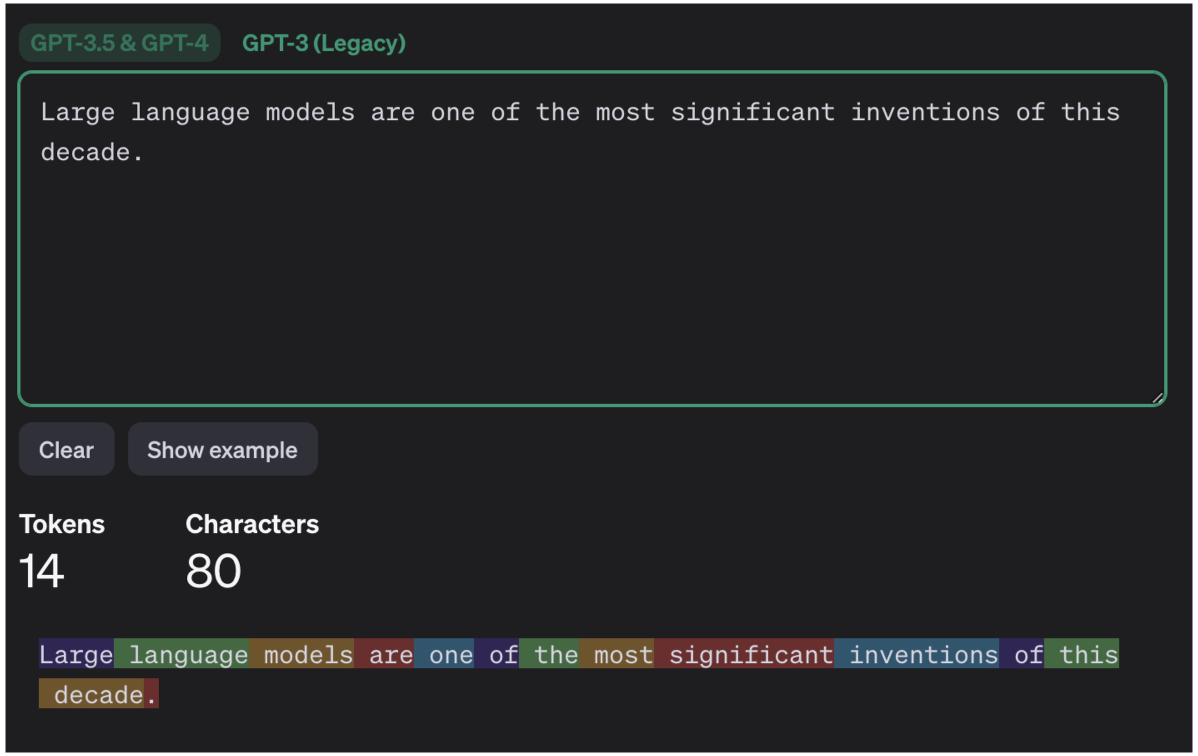
In practice, we run BPE on the byte level encoding of all Unicode characters to handle multilingual tasks.
Example in Chinese:

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:



```
Large language models are one of the most significant inventions of this
decade.
```

Tokens 14

Characters 80

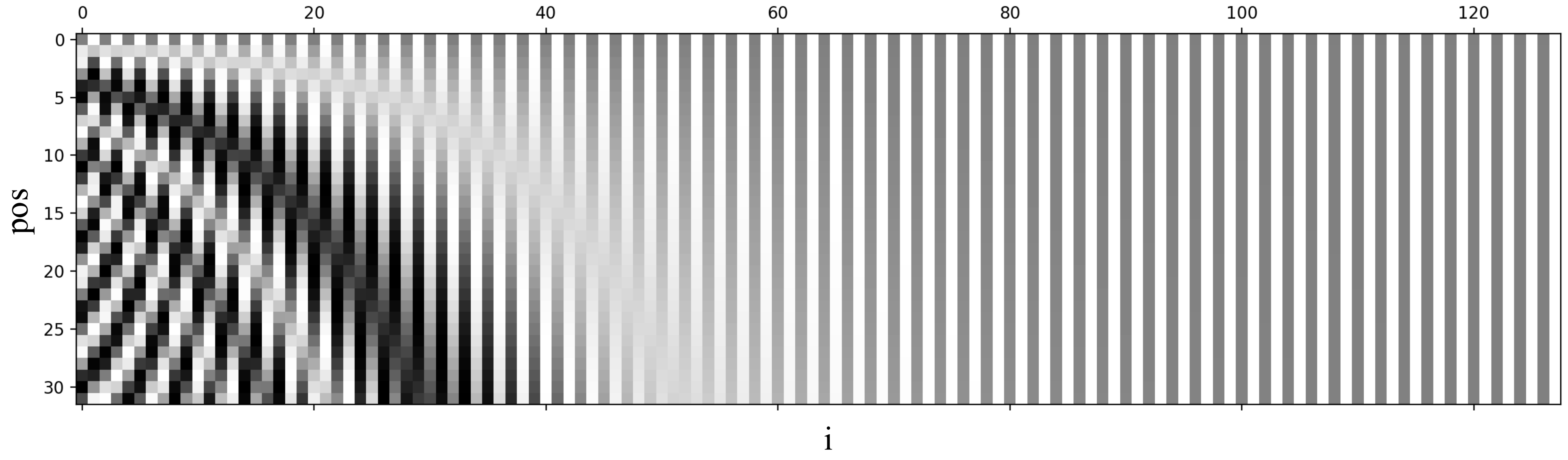
```
Large language models are one of the most significant inventions of this
decade.
```

Image Credit: [5]

Outline

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

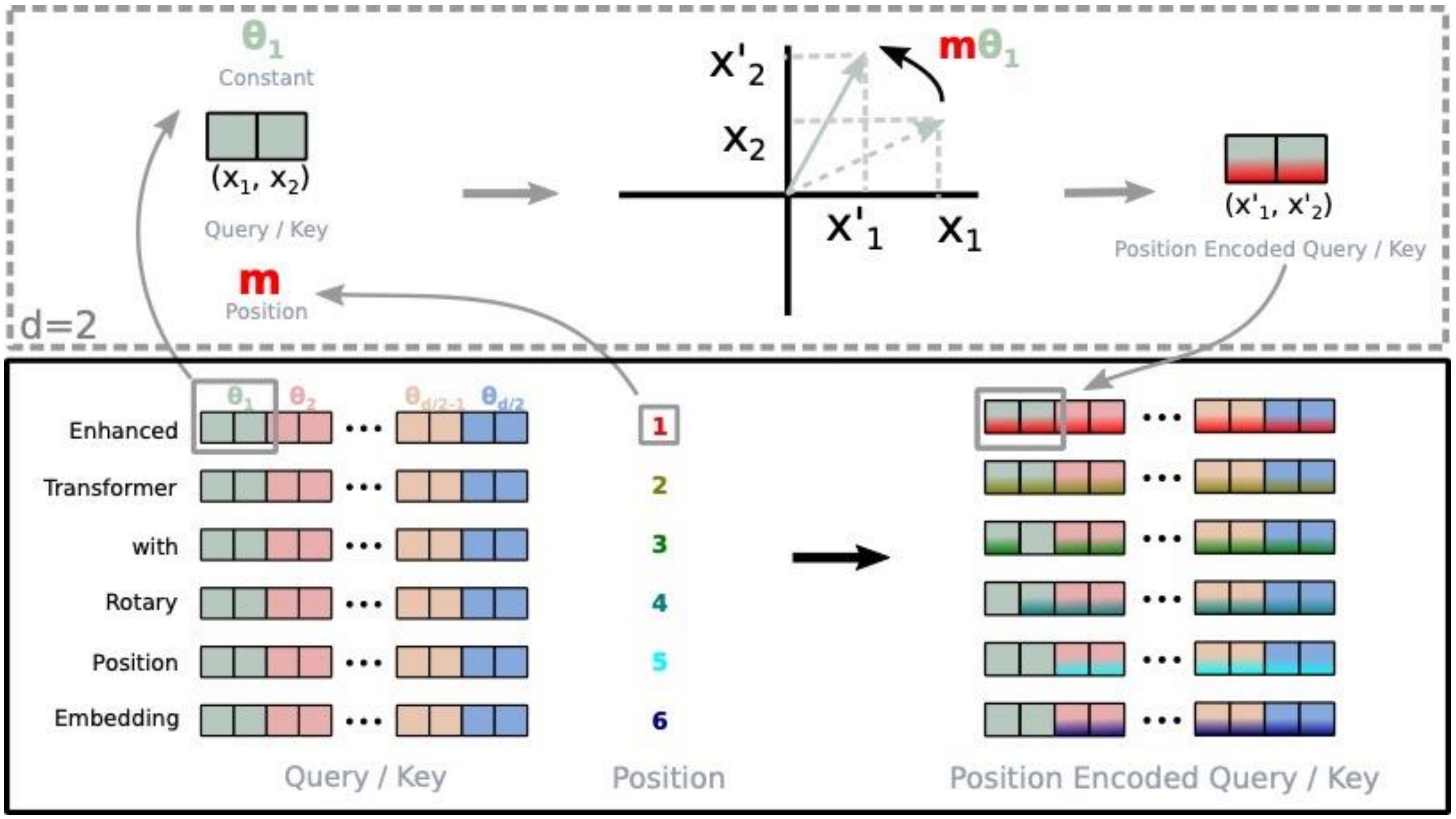


Image Credit: [24]

Rotary Positional Encoding

Code of RoPE:

```python
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

Architecture

Modern LLMs architectures are based on transformers.

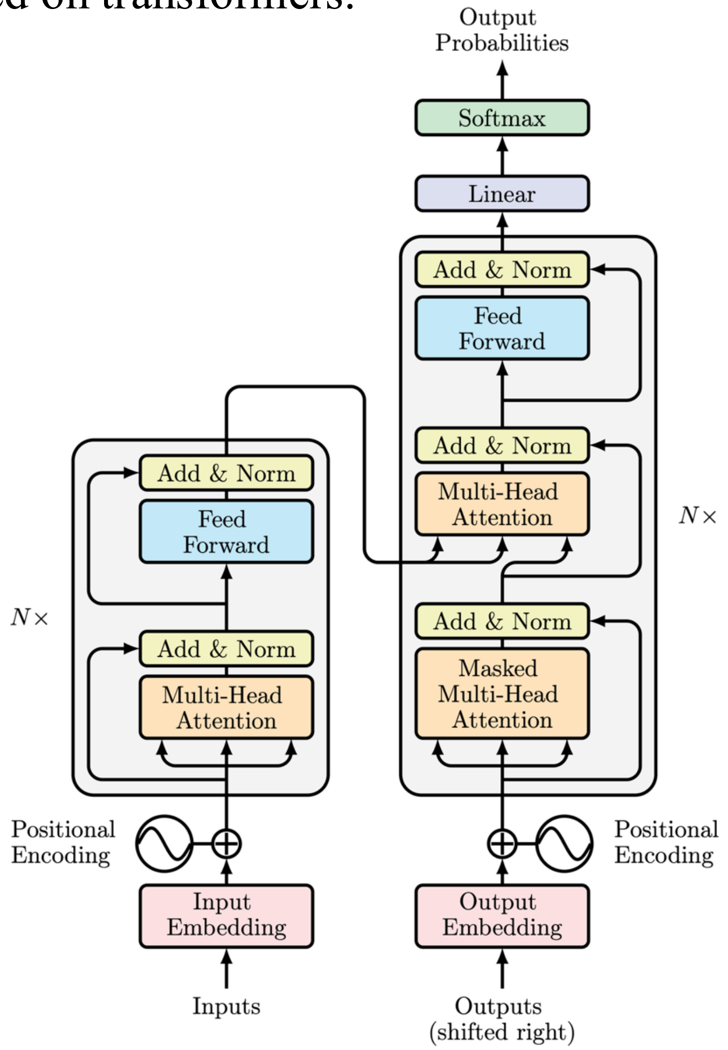


Image Credit: [7]

Architecture

Type 1: encoder-only.

These LMs generate contextual embeddings from given inputs.

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

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

Use of encoder-only LMs:

- Sentiment analysis

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

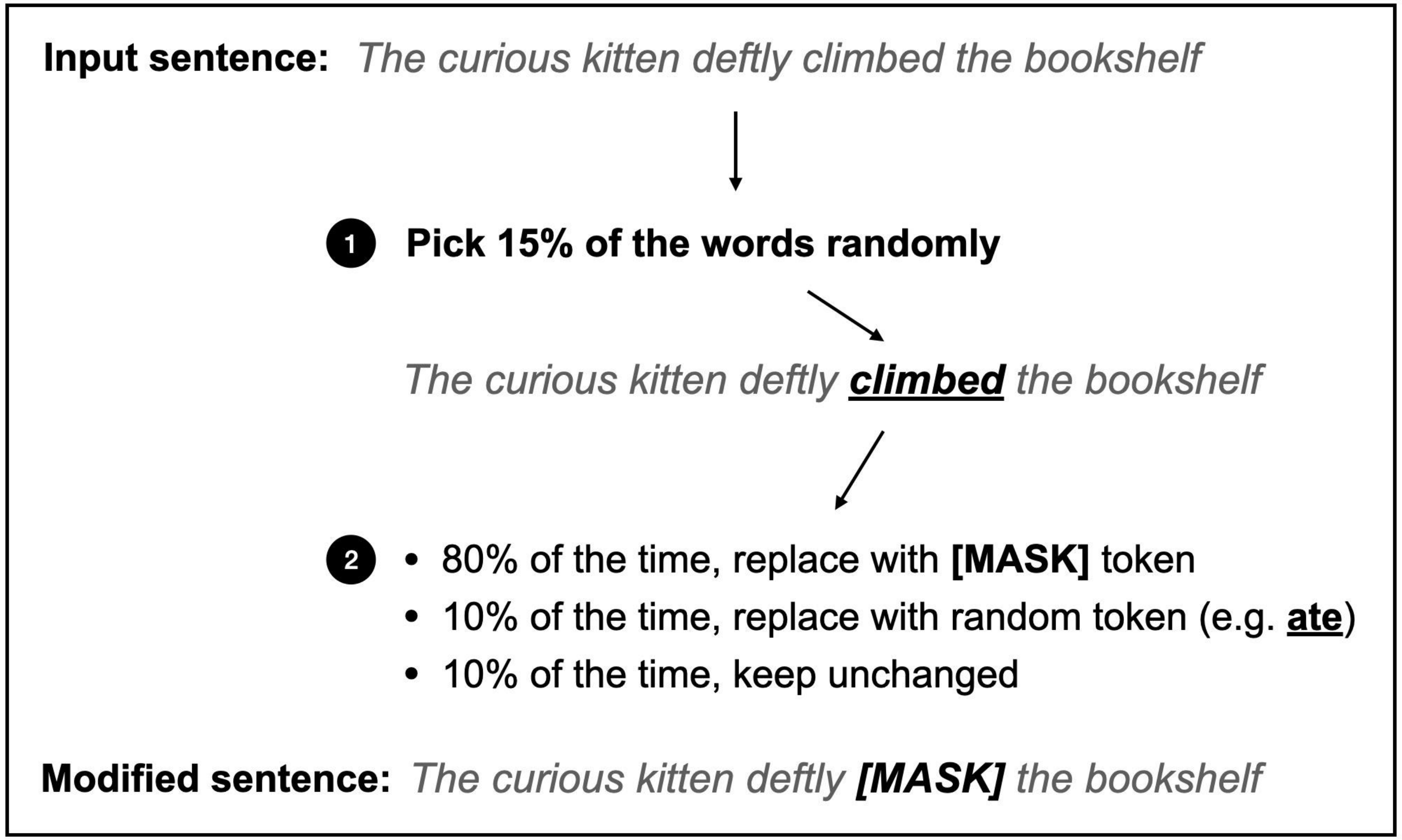
- Natural language inference

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

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.



Input sentence: *The curious kitten deftly climbed the bookshelf*
1 **Pick 15% of the words randomly**
The curious kitten deftly ***climbed*** *the bookshelf*
2 • 80% of the time, replace with **[MASK]** token
• 10% of the time, replace with random token (e.g. **ate**)
• 10% of the time, keep unchanged
Modified sentence: *The curious kitten deftly* ***[MASK]*** *the bookshelf*

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

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

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

Architecture

Type 3: encoder-decoder.

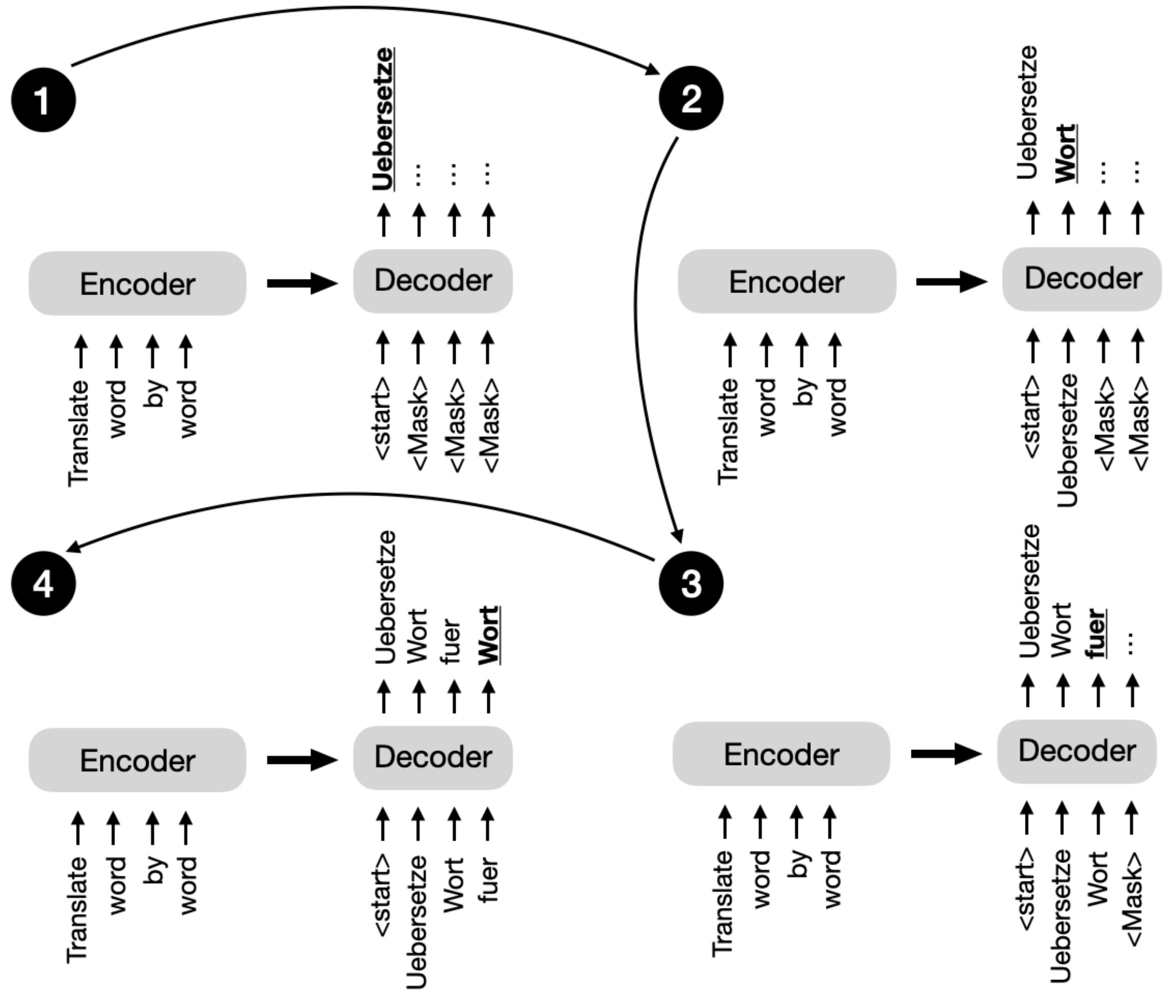
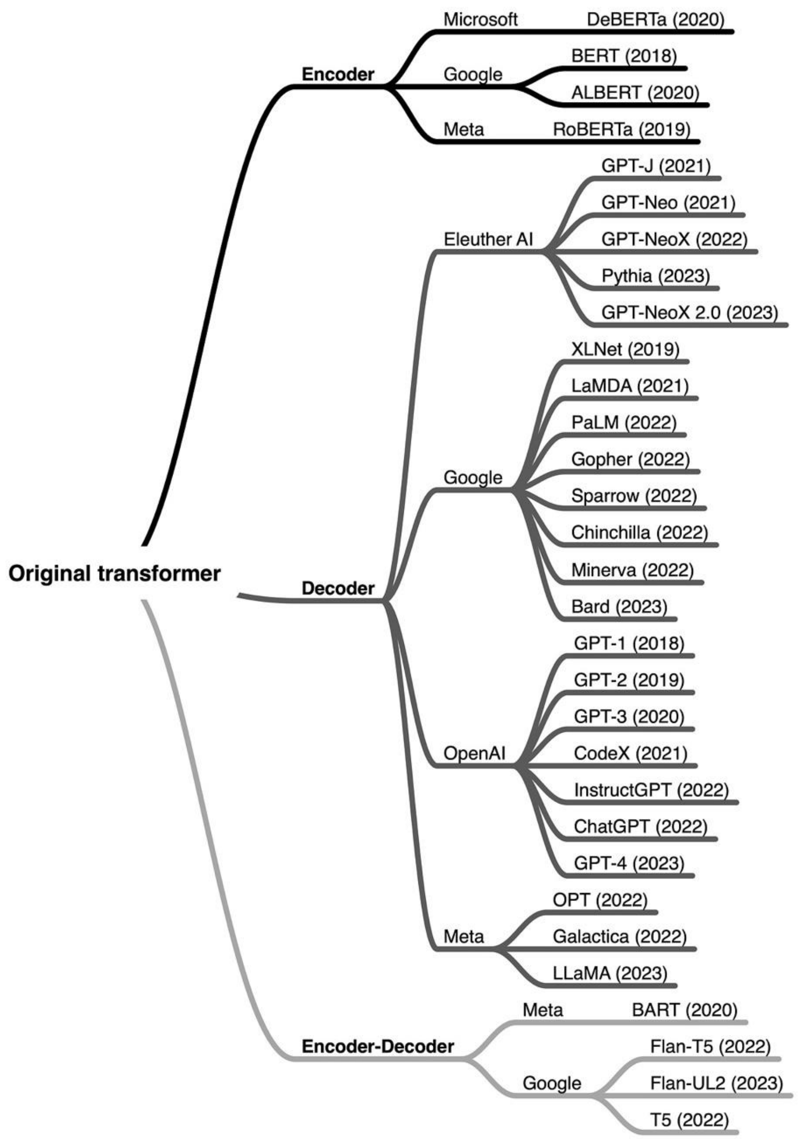


Image Credit: [8]

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**
- Pre-Training & Scaling Law
- Post-Training
    - Low Rank Adaptation (LoRA)
    - Reinforcement Learning: Basics, Policy Gradients, Actor-Critic, and PPO
    - Reinforcement Learning from Human Feedback (RLHF)
- 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:     $x_t = \text{argmax}_{v \in V} \quad P_\theta \left( X_t = v \mid X_{<t} = S \right)$
5:     $S = [S, x_t]$                                  ▷ Concatenate the new token
6: end for
7: return $S$
```

Beam Search

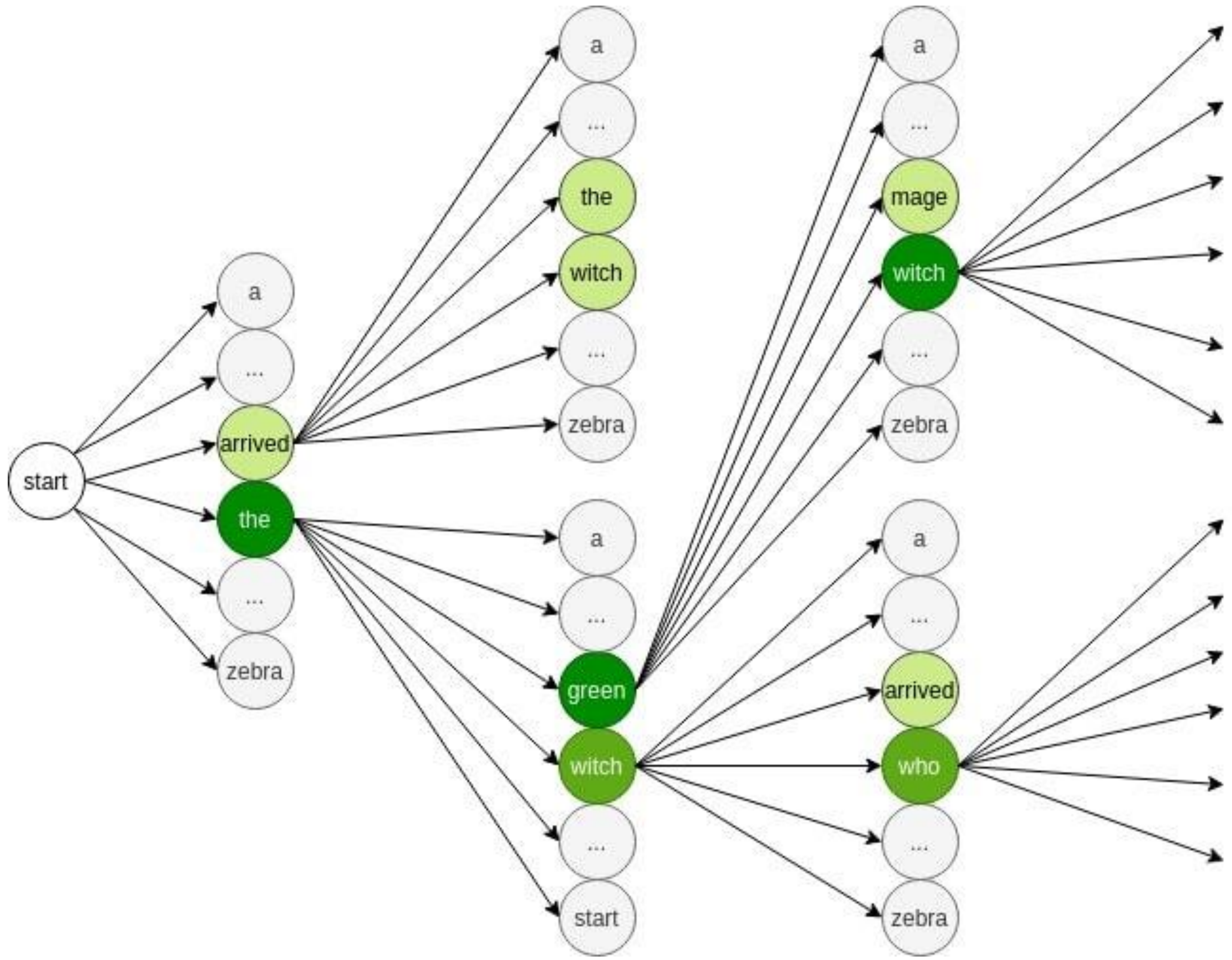


Image Credit: [9]

Beam Search

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

Algorithm 2 Beam Search

```
1:  Special start token $x_0$, beam size $B$, vocabulary $V$, sequence length $T$
2:  $S = \{\underbrace{[x_0], \ldots, [x_0]}_{B}\}$
3:  for $t \leftarrow 1$ to $T$ do
4:      $C = \{\}$
5:      for $i \leftarrow 1$ to $B$ do
6:          $\mathcal{N} = \text{argsort}_{v \in V} \quad P_\theta\left(X_t = v \mid X_{<t} = S[i]\right)$    ▷ Descending order
7:          for $j \leftarrow 1$ to B do    ▷ Take top B tokens
8:              $C = C \cup \{[S[i], V[\mathcal{N}[j]]]\}$    ▷ Concatenate with existing sequence
9:          end for
10:     end for
11:     $\mathcal{C} = \text{argsort}_{c \in C} \quad P_\theta(X_{\leq t} = c)$    ▷ Descending order
12:     $S = \{C[\mathcal{C}[j]] | j = 1, \ldots, B\}$    ▷ Take top B subsequences
13: end for
14: return $S$
```

Top-K Sampling

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

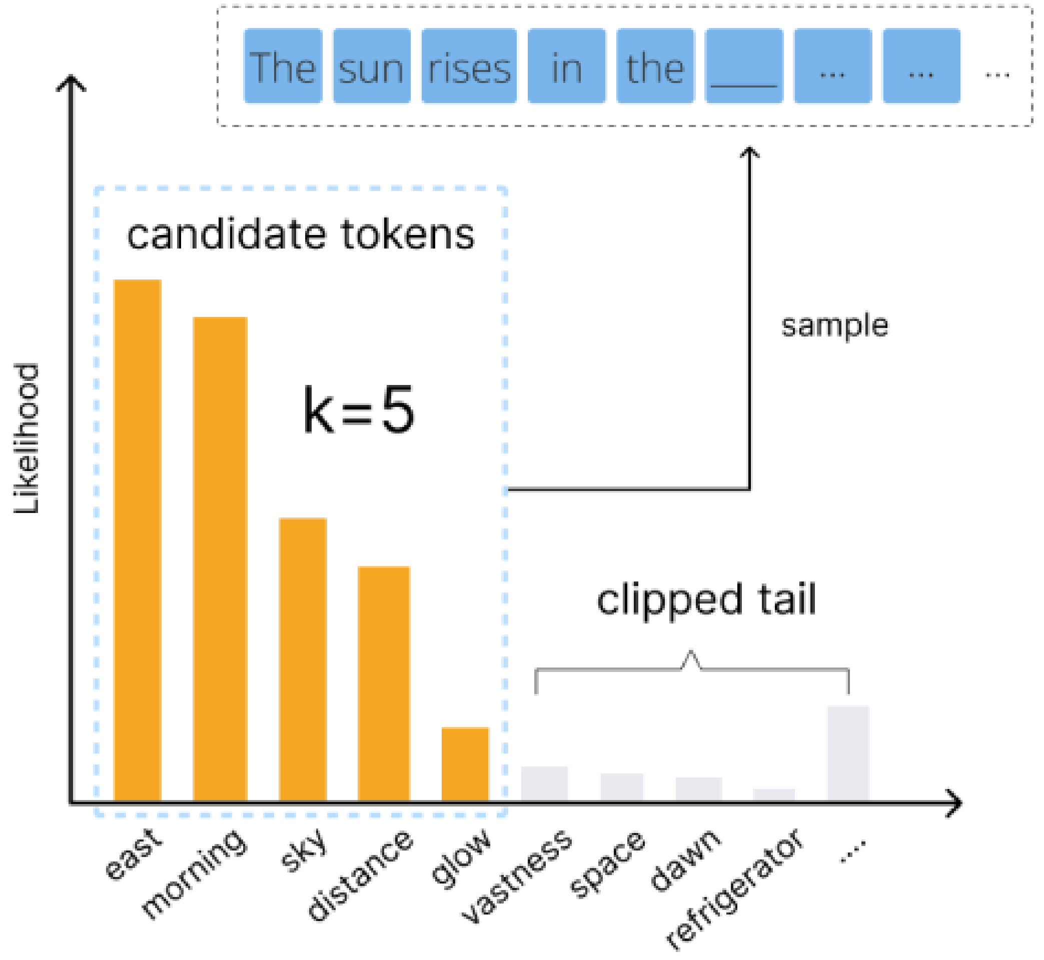


Image Credit: [10]

Top-K Sampling

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

Algorithm 3 Top-K Sampling

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

Nucleus (Top-P) Sampling

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

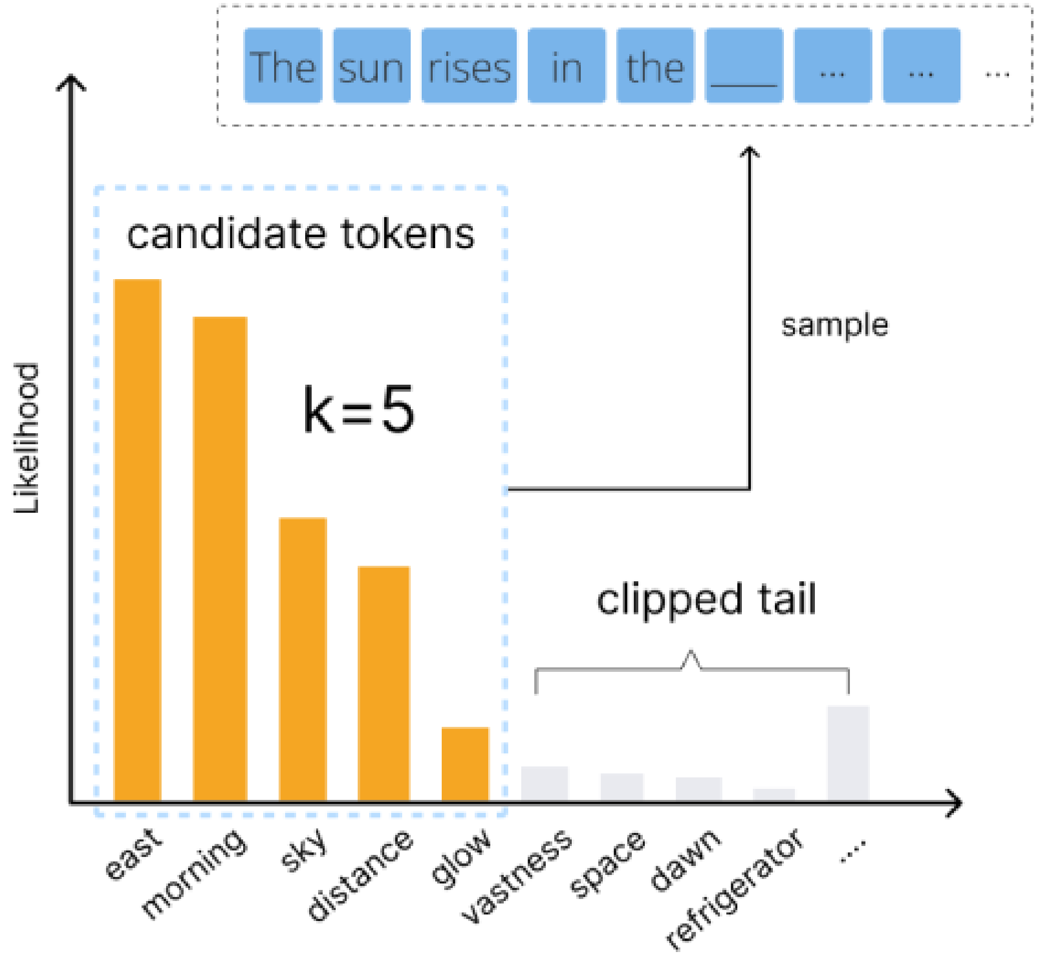


Image Credit: [10]

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:     $\mathcal{N} = \text{argsort}_{v \in V} \quad P_\theta(X_t = v \mid X_{<t} = S)$        ▷ Descending order
5:     $K = \min_k \quad \sum_{j=1}^{k} P_\theta(X_t = V[\mathcal{N}[j]] \mid X_{<t} = S) \geq \rho$
6:     for $i \leftarrow 1$ to $K$ do
7:         $\bar{P}(X_t = V[\mathcal{N}[i]] \mid X_{<t} = S) = \frac{P_\theta(X_t = V[\mathcal{N}[i]] | X_{<t} = S)}{\sum_{j=1}^{K} P_\theta(X_t = V[\mathcal{N}[j]] | X_{<t} = S)}$
8:     end for
9:     $x_t \sim \bar{P}$
10:    $S = [S, x_t]$        ▷ Concatenate the new token
11: end for
12: return $S$
```

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- **Pre-Training & Scaling Law**
- Post-Training
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning: Basics, Policy Gradients, Actor-Critic, and PPO
  - Reinforcement Learning from Human Feedback (RLHF)
- Prompting

Loss Function

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

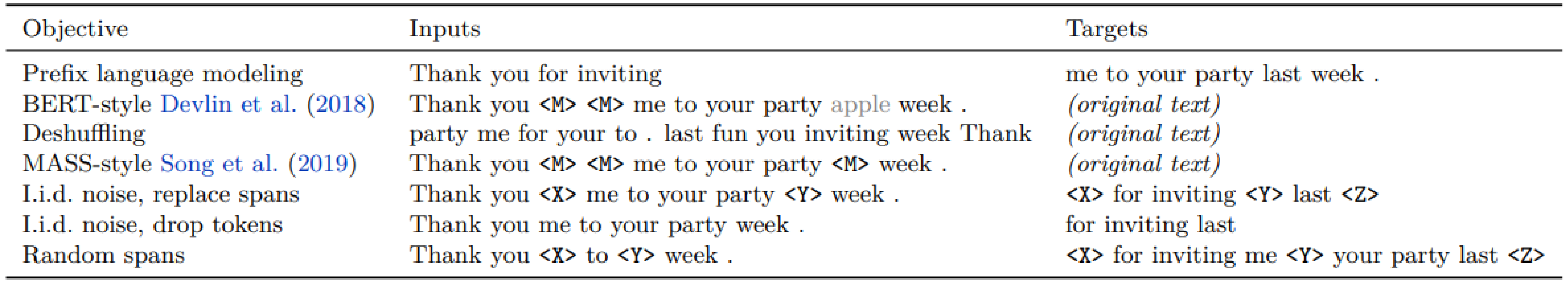
$$L(\theta) = \frac{1}{B}\sum_{i=1}^{B} = -\log p_\theta(\boldsymbol{x}_T^i, \boldsymbol{x}_{T-1}^i, \ldots, \boldsymbol{x}_1^i) = \frac{1}{B}\sum_{i=1}^{B}\sum_{t=1}^{T} -\log p_\theta(\boldsymbol{x}_t^i | \boldsymbol{x}_{t-1}^i, \ldots, \boldsymbol{x}_1^i)$$

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.

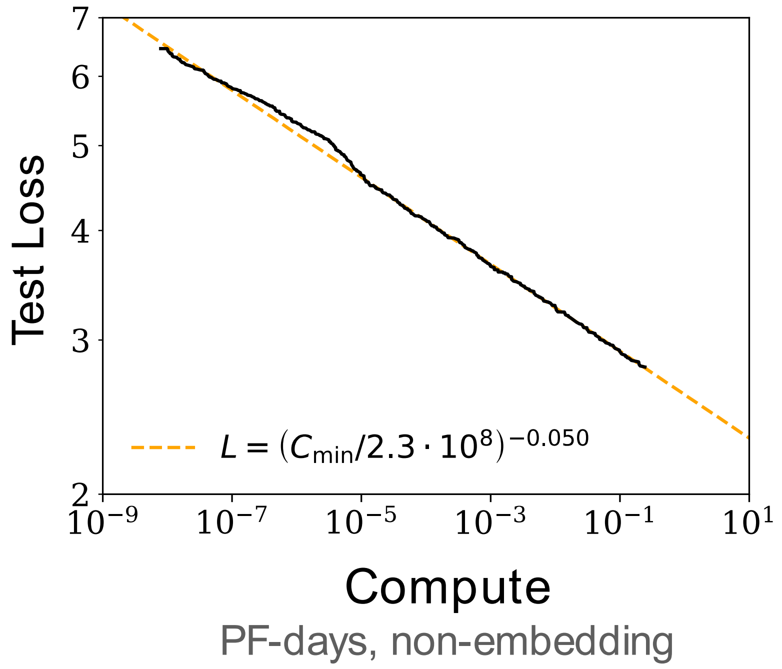
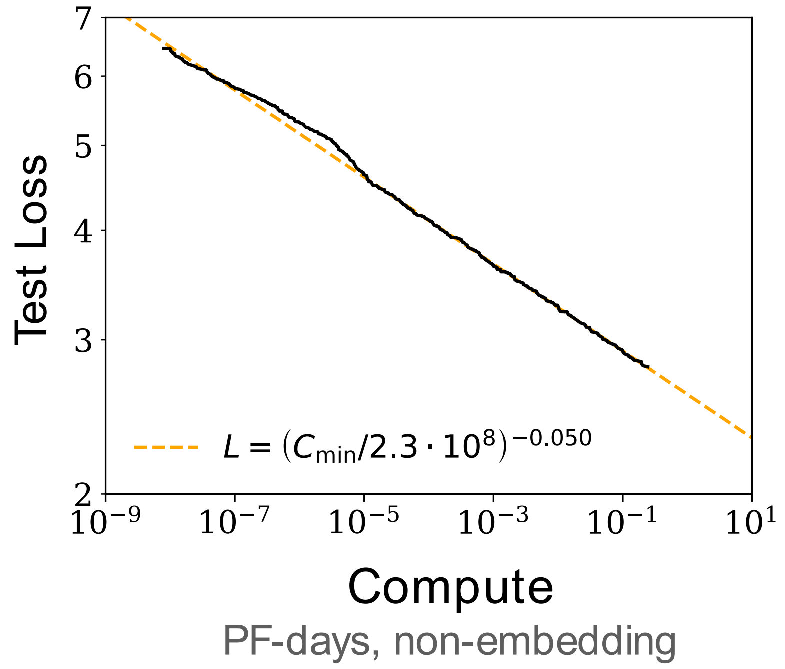


Image Credit: [17]

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.



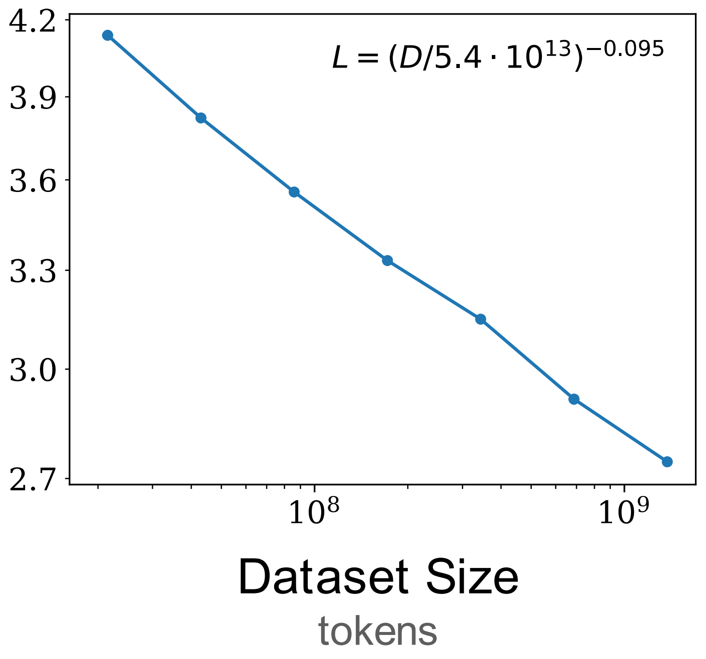


Image Credit: [17]

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.

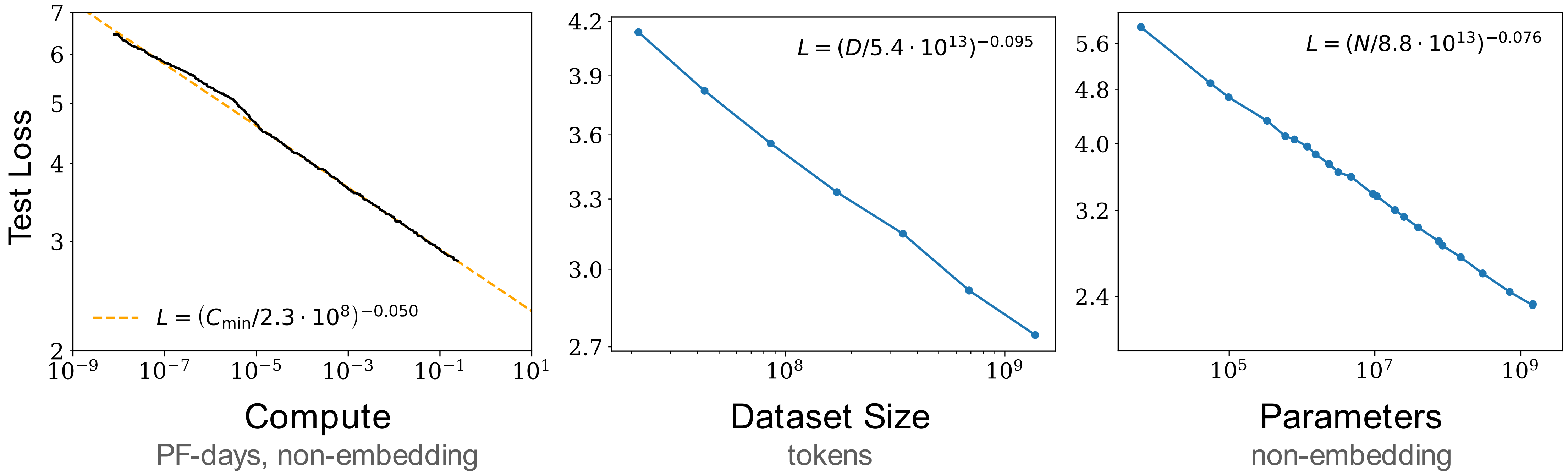
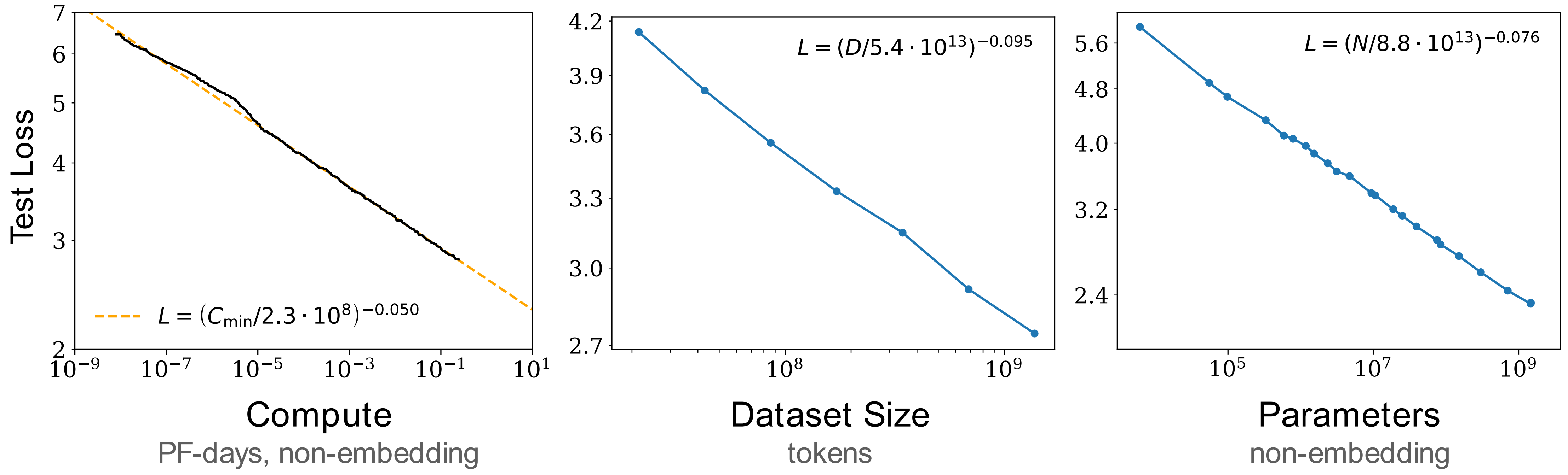


Image Credit: [17]

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.

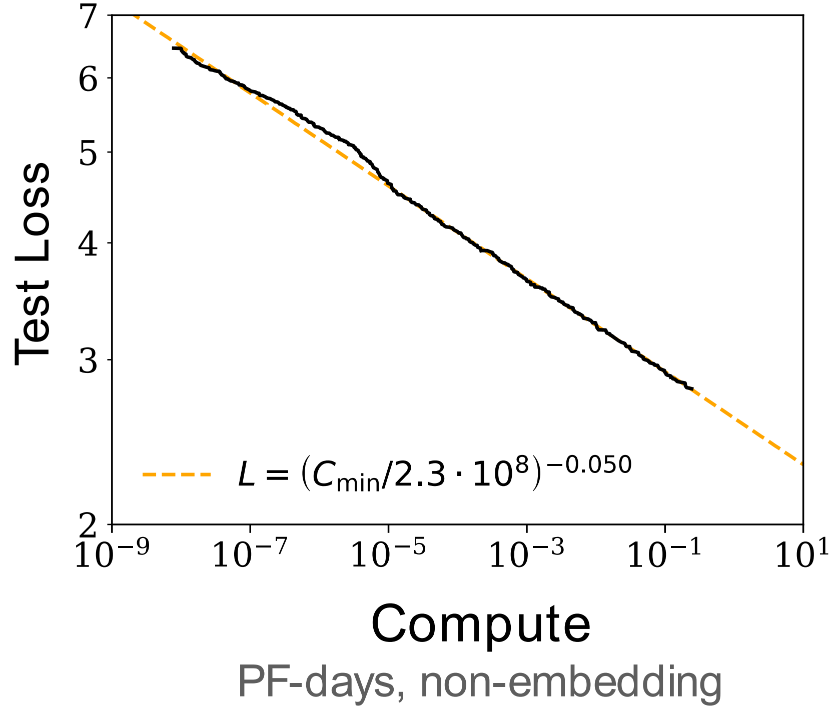


Power law $y = ax^k$ appears as straight lines in log-log plot!

Image Credit: [17]

Scaling Law

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



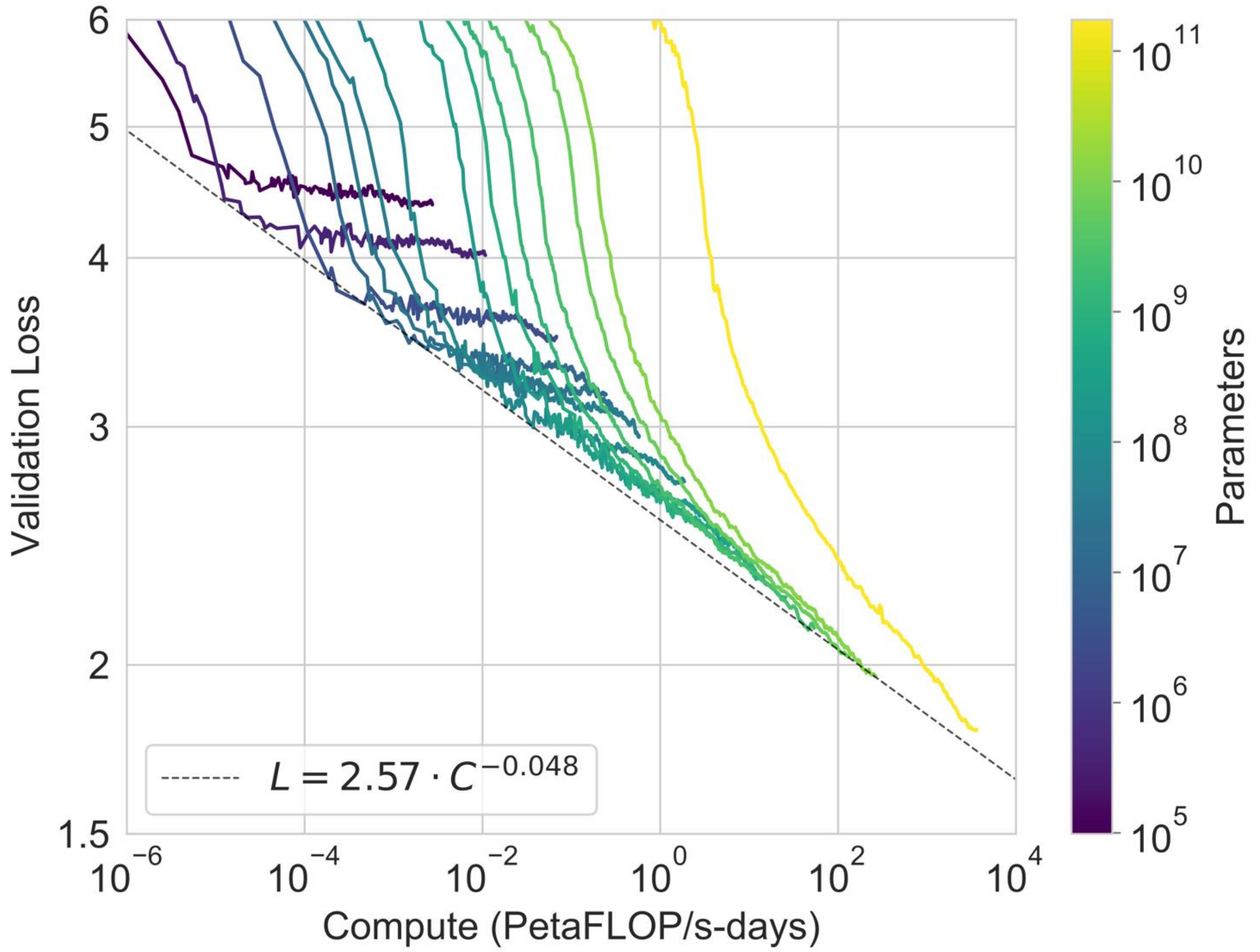
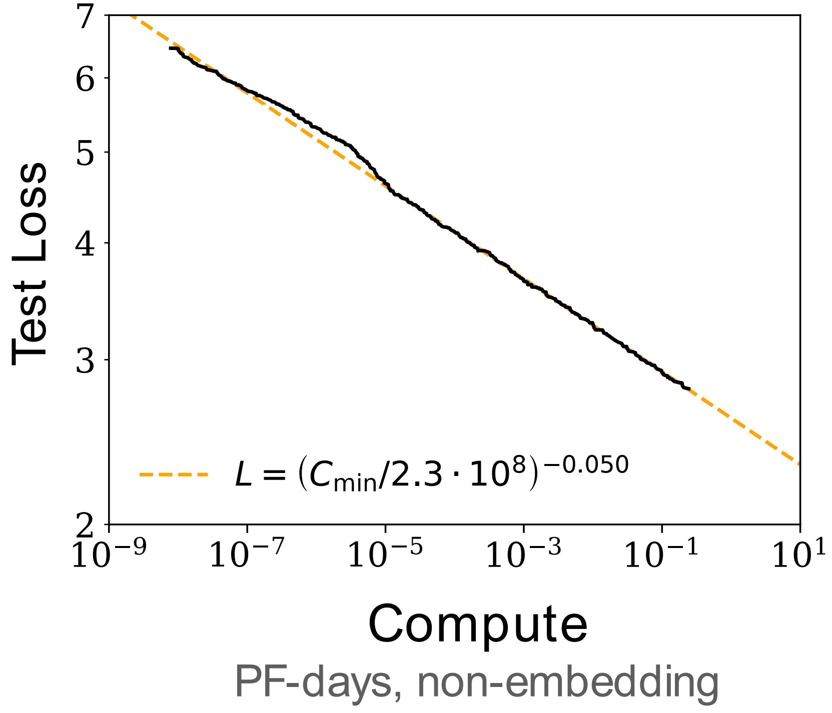


Image Credit: [17]

Scaling Law

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



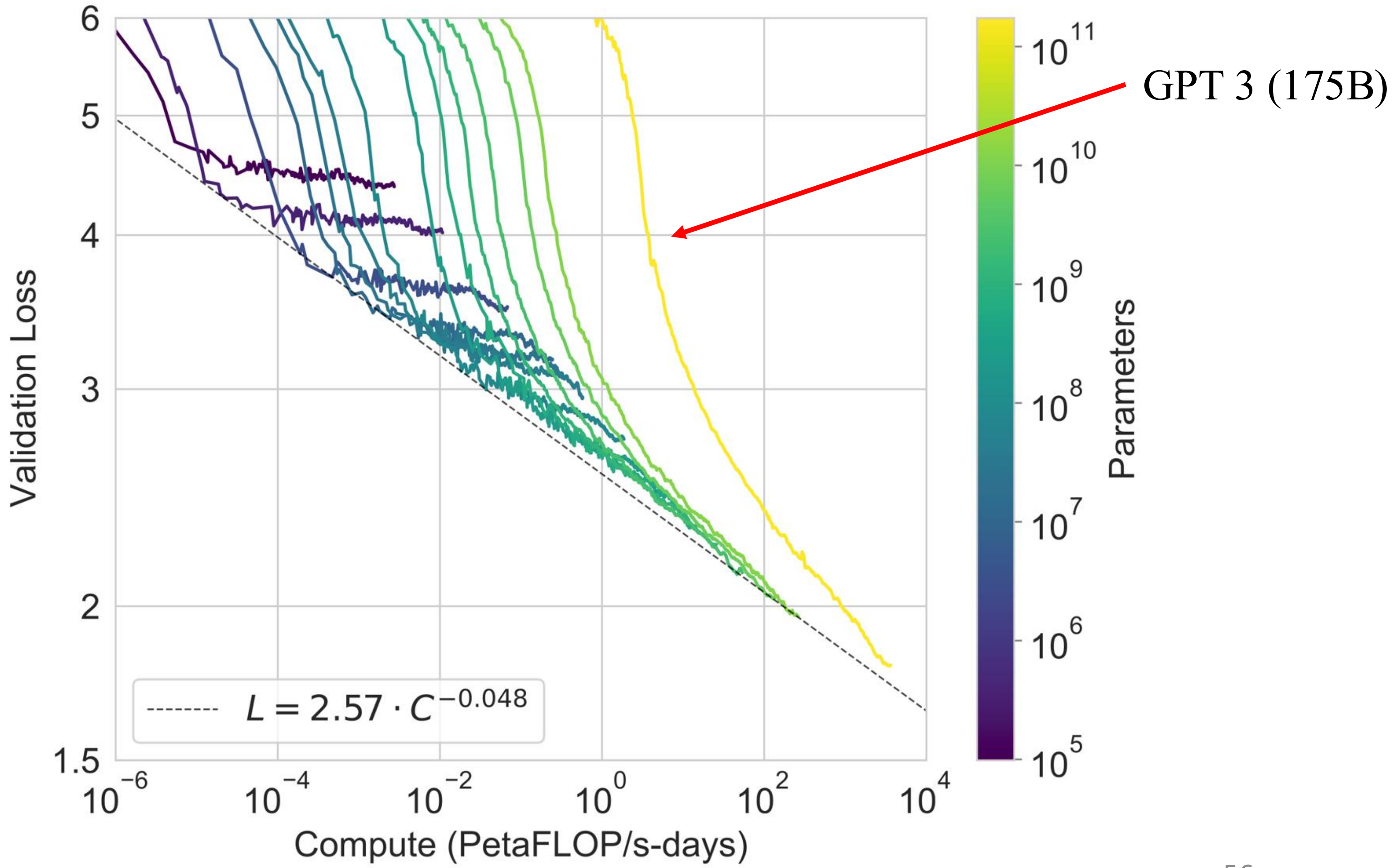
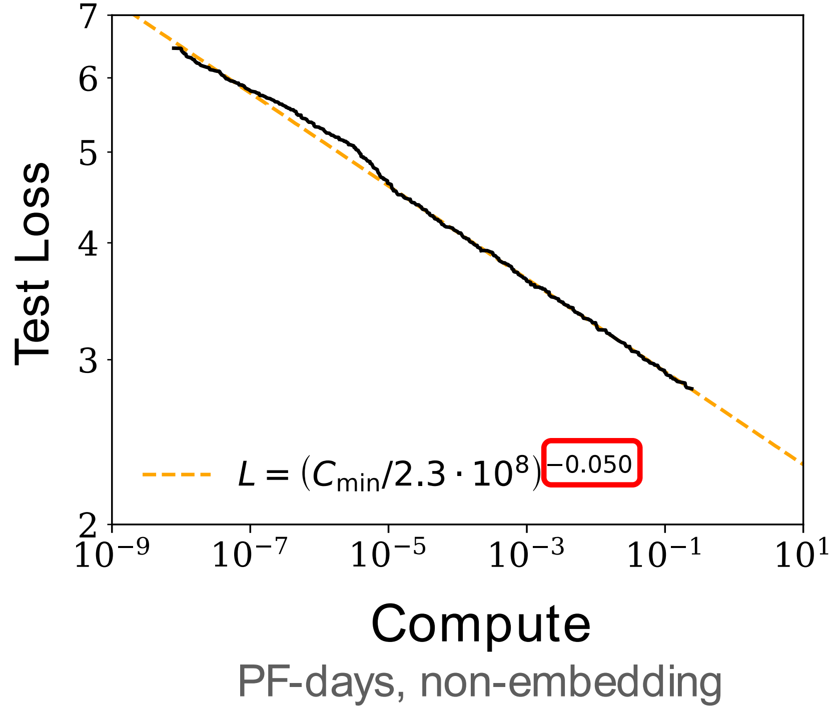


Image Credit: [17]

Scaling Law

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



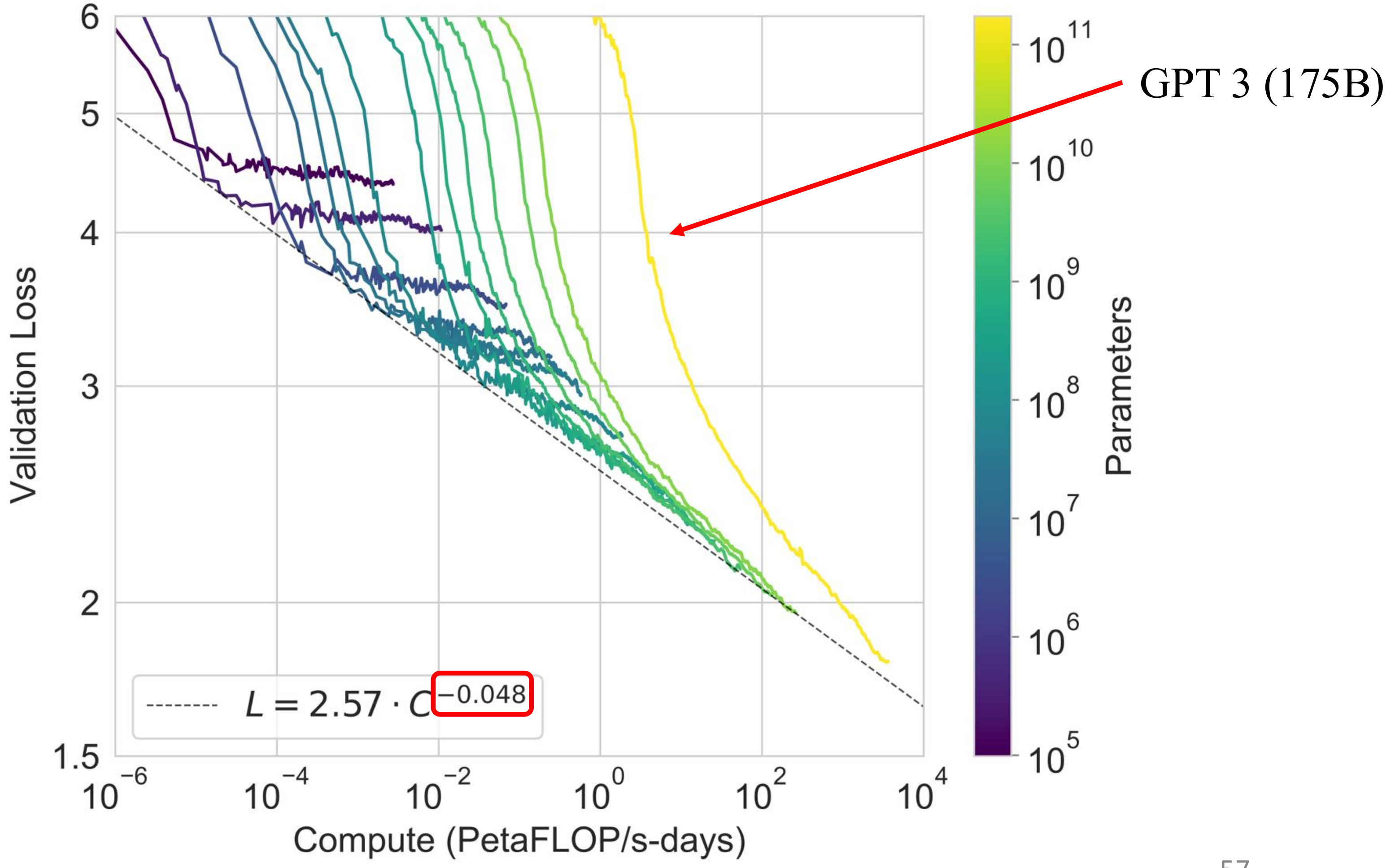


Image Credit: [17]

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Pre-Training & Scaling Law
- Post-Training
  - **Low Rank Adaptation (LoRA)**
  - Reinforcement Learning: Basics, Policy Gradients, Actor-Critic, and PPO
  - Reinforcement Learning from Human Feedback (RLHF)
- 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]

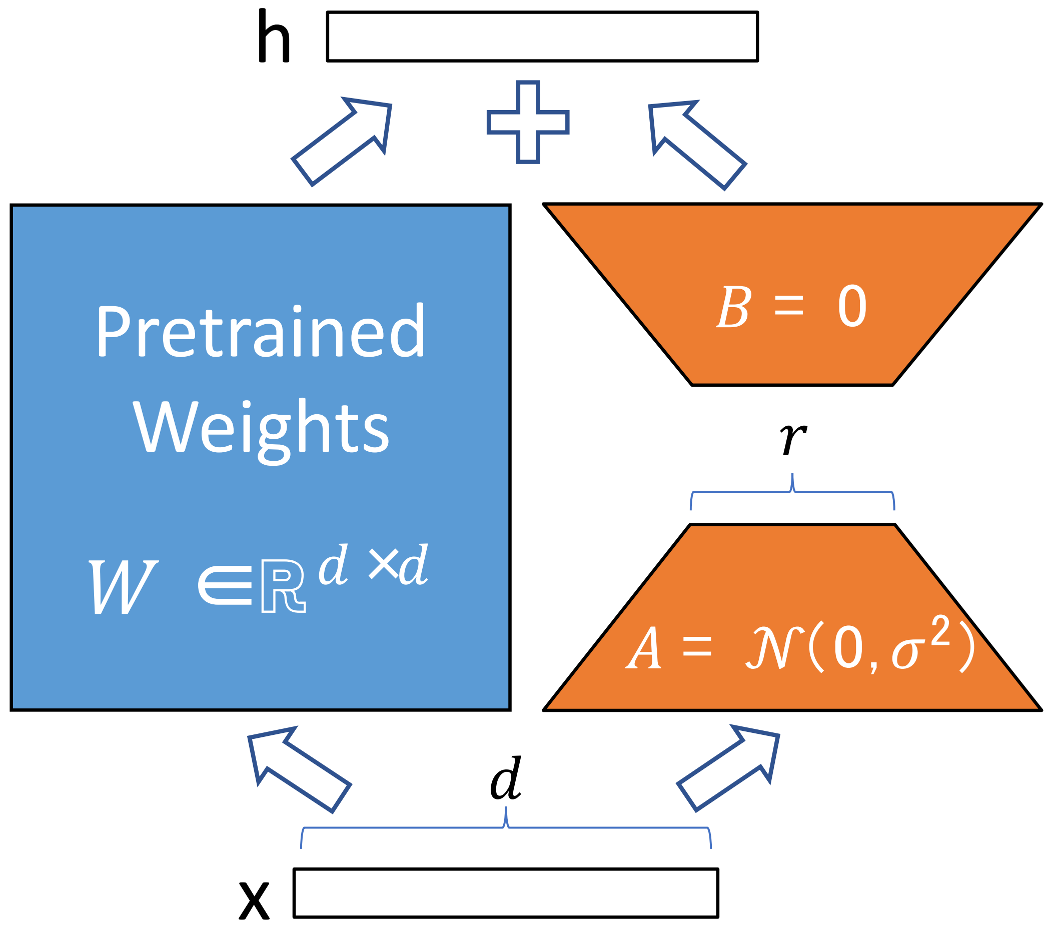
Low Rank Adaptation (LoRA)

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:

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



h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Image Credit: [19]

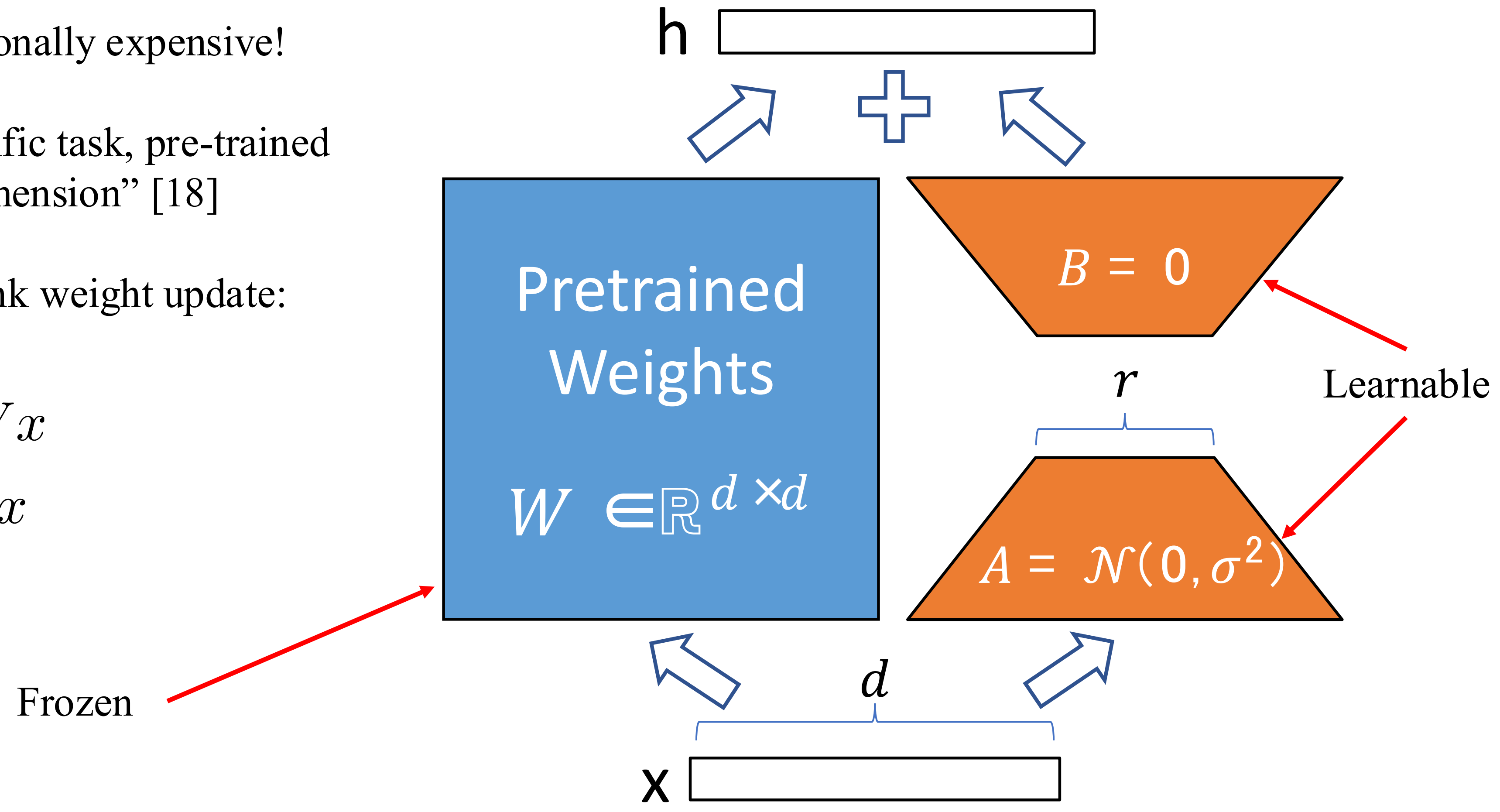
Low Rank Adaptation (LoRA)

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:

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



h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
Learnable
$A = \mathcal{N}(0, \sigma^2)$
Frozen
d
x

Image Credit: [19]

Outline

- Introduction & Background
- Models
    - Tokenization
    - Rotary Positional Encoding
    - Architecture
- Sampling
- Pre-Training & Scaling Law
- Post-Training
    - Low Rank Adaptation (LoRA)
    - **Reinforcement Learning: Basics, Policy Gradients, Actor-Critic, and PPO**
    - Reinforcement Learning from Human Feedback (RLHF)
- Prompting

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$
- $\gamma \in [0, 1]$ is a discount factor

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Markov Property: The future is independent of the past given the present!

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\quad \mathcal{P}_{ss'}^{a} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function $\quad \mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]$
- $\gamma \in [0, 1]$ is a discount factor

Markov Decision Process

Almost all RL problems can be formalized as Markov decision processes (MDPs).

A Markov decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

Markov Property: The future is independent of the past given the present!

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix $\mathcal{P}^a_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma \in [0, 1]$ is a discount factor

MDP describes an environment where all states are Markov and can be extended to:

- countably infinite states and or action spaces
- continuous state and or action spaces
- continuous time (requires partial differentiable equations)
- partially observable (POMDPs)

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi\left[G_t | S_t = s\right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Optimal value function $$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking an action a, and then following policy π

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking an action a, and then following policy π

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

Optimal Q function

$$Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking an action a, and then following policy π

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

Optimal Q function

$$Q_*(s, a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

What is the relationship between the value function and Q function?

Markov Decision Process

Return: the total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discount? Mathematically convenient, avoid infinite returns, uncertainty about the future……

Policy: the distribution over actions given states

$$\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$$

We assume *stationary* policies

Value (a.k.a., State-Value) function: the expected return starting from state s and then following policy π

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Optimal value function

$$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Q (a.k.a. Action-Value) function: the expected return starting from state s, taking an action a, and then following policy π

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$$

Optimal Q function

$$Q_*(s, a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$$

What is the relationship between the value function and Q function?

$$V_\pi(s) = \sum_a Q_\pi(s, a) \pi(a|s)$$

Optimal Policy

Optimal value function $V_*(s) = \max_{\pi} \mathbb{E}_{\pi}\left[G_t | S_t = s\right]$

Optimal Q function $Q_*(s,a) = \max_{\pi} \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right]$

One can define a partial ordering over policies $\pi \geq \pi' \iff V_{\pi}(s) \geq V_{\pi'}(s), \forall s$

Theorem (Optimal Policies)

For any Markov decision process

- *There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$*
  (There can be more than one such optimal policy.)
- *All optimal policies achieve the optimal value function, $v^{\pi^*}(s) = v^*(s)$*
- *All optimal policies achieve the optimal action-value function, $q^{\pi^*}(s,a) = q^*(s,a)$*

Image Credit: [12]

Optimal Policy

Optimal value function $$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Optimal Q function $$Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

If we know $Q_*(s,a)$, an optimal policy can be found:

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \underset{a}{\operatorname{argmax}} \; Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Optimal Policy

Optimal value function $$V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$$

Optimal Q function $$Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$$

If we know $Q_*(s,a)$, an optimal policy can be found:

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \underset{a}{\operatorname{argmax}} \; Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Why?

Optimal Policy

Optimal value function $V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$

Optimal Q function $Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$

If we know $Q_*(s,a)$, an optimal policy can be found:

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \underset{a}{\text{argmax}} \; Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

Why?

Because $V_\pi(s) = \sum_a Q_\pi(s,a)\pi(a|s)$ is a convex combination of $Q_\pi(s,a)$, we have

$$V_\pi(s) \le Q_\pi(s,a_*), \qquad a_* = \underset{a}{\text{argmax}} \; Q_*(s,a)$$

and the equality holds only if the policy is optimal.

Optimal Policy

Optimal value function $V_*(s) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s\right]$

Optimal Q function $Q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right]$

If we know $Q_*(s,a)$, an optimal policy can be found:

$$\pi_*(a|s) = \begin{cases} 1 & a = a_*(s) = \underset{a}{\operatorname{argmax}} \; Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP.
- There can be multiple actions that maximize $Q_*(s,a)$, we can just pick any of these.

RL Taxonomy

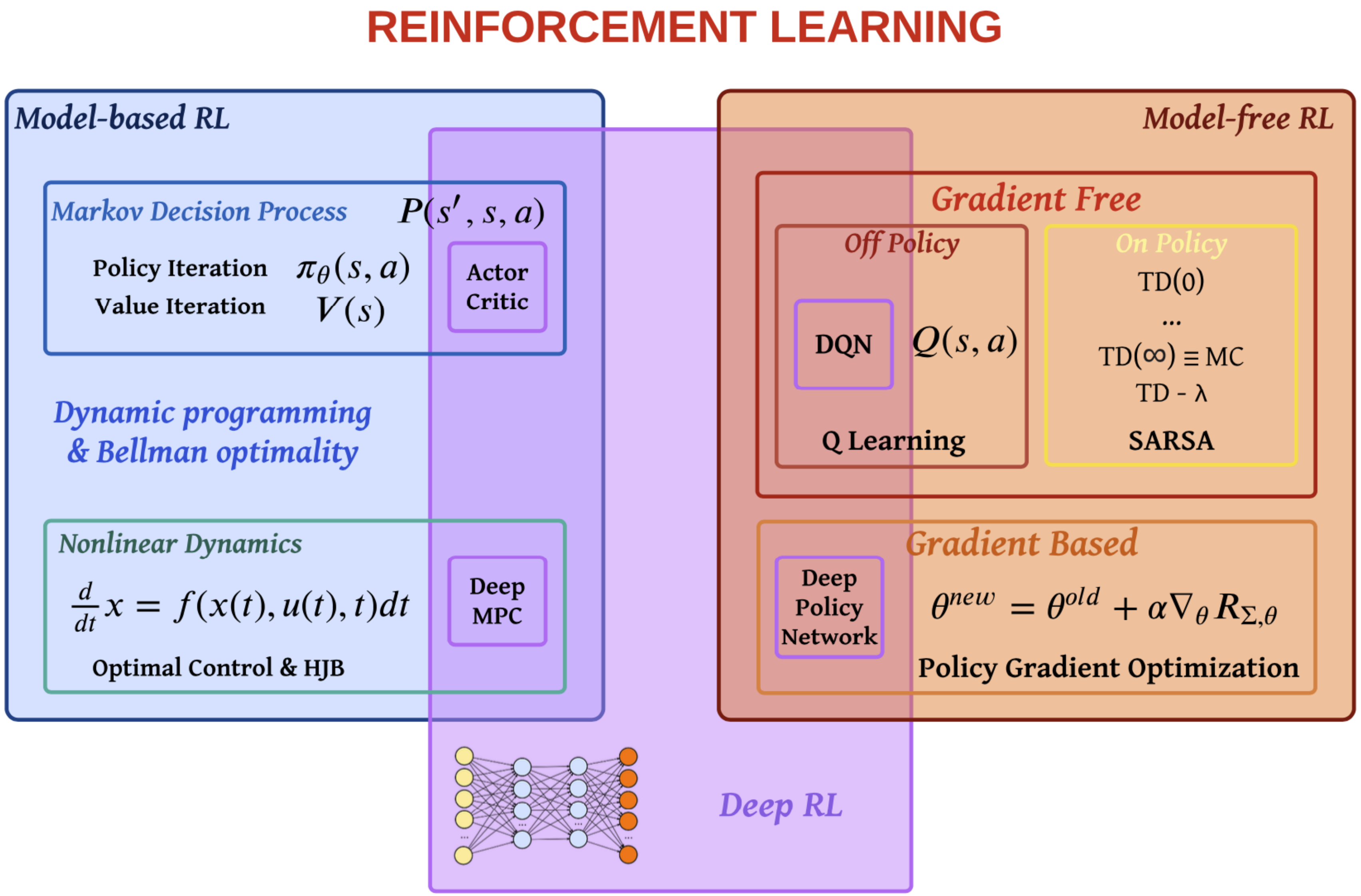


Image Credit: Murphy, K.P., 2023. Probabilistic machine learning: Advanced topics. MIT Press.

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Often converge to local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right]$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau} \\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right] \end{aligned}$$

Log derivative trick:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau} \\ &= \int \nabla_\theta \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau} \\ &= \int \mathbb{P}_\theta(\boldsymbol{\tau})\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau} \\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})\right] \end{aligned}$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have
$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau})\right]\\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T}R_t(A_t,S_t)\right)\right]\end{aligned}$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S) \prod_{t=1}^{T} \pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t, S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S) \prod_{t=1}^{T} \pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T} \log \pi_\theta(A_t|S_t) + \sum_{t=1}^{T} \log \mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right]
\end{aligned}$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t, S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log \pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log \mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}$$

We use the Monte Carlo Approximation. This is known as REINFORCE algorithm [25]!

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S) \prod_{t=1}^{T} \pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t, S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S) \prod_{t=1}^{T} \pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T} \log \pi_\theta(A_t|S_t) + \sum_{t=1}^{T} \log \mathbb{P}(S_{t+1}|S_t, A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t, S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\boxed{\left(\sum_{t=1}^{T} r_t^{(i)}\right)}
\end{aligned}$$

Full Reward

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$$

Reward-to-go: *my current policy can not change past rewards (causality)*!

Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right)} \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right)} \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right)} \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t \right) \right)$

Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t\right)\right)$

This works since

$$\begin{aligned}
\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(A_t|S_t)b_t\right] &= \iint_{A_t,S_t} p(S_t)\pi_\theta(A_t|S_t)\nabla_\theta \log \pi_\theta(A_t|S_t)b_t \mathrm{d}A_t\mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\int_{A_t}\nabla_\theta \pi_\theta(A_t|S_t)b_t\mathrm{d}A_t\mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\nabla_\theta\int_{A_t}\pi_\theta(A_t|S_t)\mathrm{d}A_t b_t\mathrm{d}S_t \\
&= \int_{S_t} p(S_t)(\nabla_\theta 1)b_t\mathrm{d}S_t = 0
\end{aligned}$$

Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t\right)\right)$

This works since

$$\begin{aligned}\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(A_t|S_t) b_t\right] &= \iint_{A_t,S_t} p(S_t)\pi_\theta(A_t|S_t)\nabla_\theta \log \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\ &= \int_{S_t} p(S_t)\int_{A_t}\nabla_\theta \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\ &= \int_{S_t} p(S_t)\nabla_\theta\int_{A_t}\pi_\theta(A_t|S_t)\mathrm{d}A_t b_t \mathrm{d}S_t \\ &= \int_{S_t} p(S_t)(\nabla_\theta 1) b_t \mathrm{d}S_t = 0\end{aligned}$$

Since the baseline does not depend on the action, subtracting the baseline still leads to an unbiased estimator. By carefully choosing the baseline, we can reduce the variance!

Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Actor-Critic Methods

Recall policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

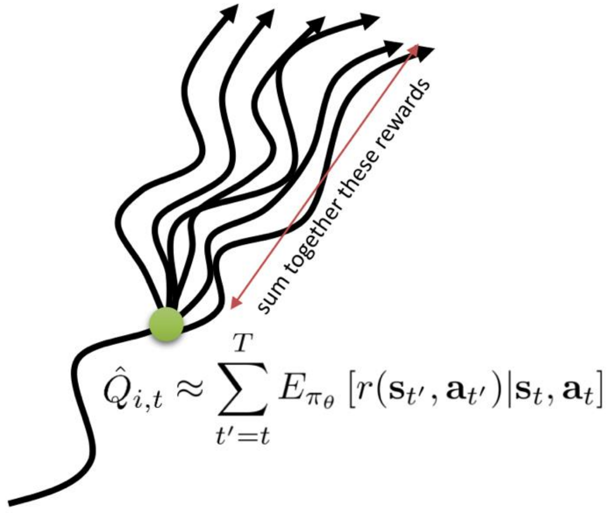

sum together these rewards
$\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$

Image Credit: https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/slides/lec-6.pdf

Actor-Critic Methods

Recall policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

Following the idea of reducing variance via baselines, we introduce the **advantage**:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

where the value does not depend on the action, thus serving as the baseline.

We can estimate the value based on: $V_\pi(s) = \sum_a Q_\pi(s,a)\pi(a|s)$

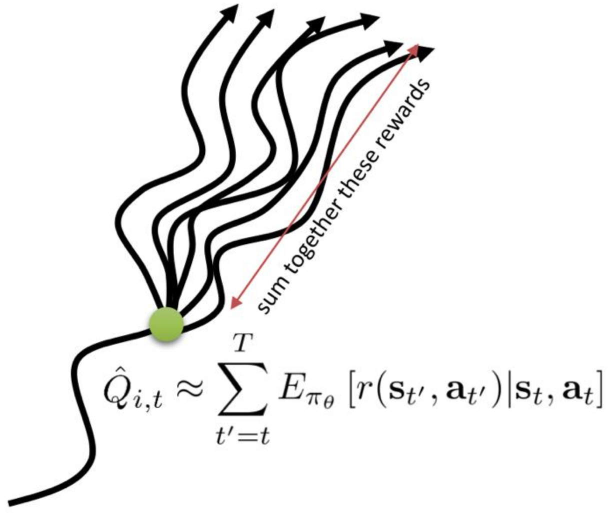

sum together these rewards
$\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right]$

Image Credit: https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/slides/lec-6.pdf

Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\boxed{\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)}\right)$$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

Following the idea of reducing variance via baselines, we introduce the **advantage**:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

where the value does not depend on the action, thus serving as the baseline.

We can estimate the value based on: $$V_\pi(s) = \sum_a Q_\pi(s,a)\pi(a|s)$$

Since subtracting a baseline still gives us an unbiased estimator, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)A_{\pi_\theta}(s_t,a_t)\right]$$

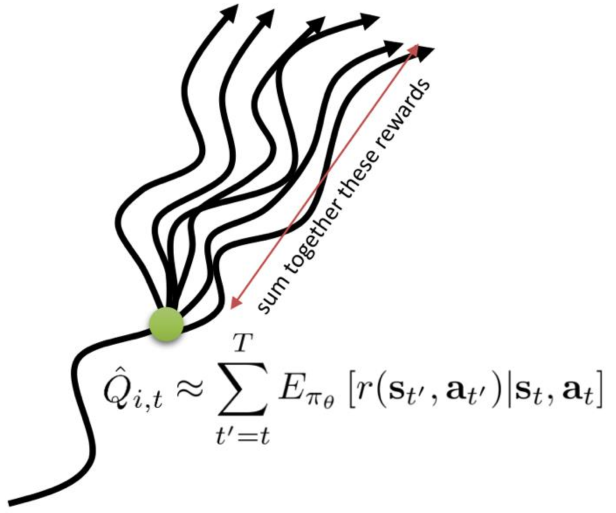

sum together these rewards
$\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right]$

Image Credit: https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/slides/lec-6.pdf

Actor-Critic Methods

How to estimate the advantage function?

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

Actor-Critic Methods

How to estimate the advantage function?

$$\begin{aligned} A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\ &= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\ &\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)} \end{aligned}$$

Temporal Difference (TD) error!

Based on the TD error (1-step advantage), we can reduce it to estimating the value function!

Actor-Critic Methods

How to estimate the advantage function?

$$\begin{aligned} A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\ &= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\ &\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)} \end{aligned}$$

Temporal Difference (TD) error!

Based on the TD error (1-step advantage), we can reduce it to estimating the value function!

How to estimate the value? We can learn a value network to fit the (Monte Carlo) Policy Evaluation!

Actor-Critic Methods

How to estimate the advantage function?

$$\begin{aligned} A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\ &= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\ &\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)} \end{aligned}$$

Temporal Difference (TD) error!

Based on the TD error (1-step advantage), we can reduce it to estimating the value function!

How to estimate the value? We can learn a value network to fit the (Monte Carlo) Policy Evaluation!

The target can be estimated from rollout:

- Vanilla $\bar{V}(s_t^{(i)}) \approx \sum_{t'=t}^{T} r_{t'}^{(i)}$

- Bootstrap $\bar{V}(s_t^{(i)}) \approx r_t^{(i)} + \gamma V_\phi(s_{t+1}^{(i)})$

Actor-Critic Methods

How to estimate the advantage function?

$$\begin{aligned} A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\ &= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\ &\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)} \end{aligned}$$

Temporal Difference (TD) error!

Based on the TD error (1-step advantage), we can reduce it to estimating the value function!

How to estimate the value? We can learn a value network to fit the (Monte Carlo) Policy Evaluation!

The target can be estimated from rollout:

- Vanilla $\bar{V}(s_t^{(i)}) \approx \sum_{t'=t}^{T} r_{t'}^{(i)}$
- Bootstrap $\bar{V}(s_t^{(i)}) \approx r_t^{(i)} + \gamma \boxed{V_\phi(s_{t+1}^{(i)})}$

current value network

Actor-Critic Methods

How to estimate the advantage function?

$$\hat{A}_t^{(1)} = \delta_t = \mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

$$\begin{aligned}\hat{A}_t^{(2)} &= \mathcal{R}_{s_t}^{a_t} + \gamma \mathcal{R}_{s_{t+1}}^{a_{t+1}} + \gamma^2 V_\pi(s_{t+2}) - V_\pi(s_t) \\ &= \delta_t + \gamma \delta_{t+1}\end{aligned}$$

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}$$

Actor-Critic Methods

How to estimate the advantage function?

$$\hat{A}_t^{(1)} = \delta_t = \mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

$$\begin{aligned}\hat{A}_t^{(2)} &= \mathcal{R}_{s_t}^{a_t} + \gamma \mathcal{R}_{s_{t+1}}^{a_{t+1}} + \gamma^2 V_\pi(s_{t+2}) - V_\pi(s_t) \\ &= \delta_t + \gamma \delta_{t+1}\end{aligned}$$

$$\boxed{\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}}$$

K-step discounted TD Error $\delta_{t+l} = \mathcal{R}_{s_{t+l}}^{a_{t+l}} + \gamma V_\pi(s_{t+l+1}) - V_\pi(s_{t+l})$

Actor-Critic Methods

How to estimate the advantage function?

$$\hat{A}_t^{(1)} = \delta_t = \mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$$

$$\begin{aligned}\hat{A}_t^{(2)} &= \mathcal{R}_{s_t}^{a_t} + \gamma \mathcal{R}_{s_{t+1}}^{a_{t+1}} + \gamma^2 V_\pi(s_{t+2}) - V_\pi(s_t) \\ &= \delta_t + \gamma \delta_{t+1}\end{aligned}$$

$$\boxed{\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}}$$

K-step discounted TD Error $\delta_{t+l} = \mathcal{R}_{s_{t+l}}^{a_{t+l}} + \gamma V_\pi(s_{t+l+1}) - V_\pi(s_{t+l})$

Generalized Advantage Estimation (**GAE**): It takes an exponentially weighted average of these estimated advantages, controlled by the decay hyperparameter $\lambda \in [0, 1]$

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\sum_{k=1}^{\infty} \lambda^{k-1} \hat{A}_t^{(k)} = (1-\lambda)\sum_{k=1}^{\infty} \lambda^{k-1} \left(\sum_{l=0}^{k-1} \gamma^l \delta_{t+l}\right) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

Actor-Critic Methods

To compute the target for Generalized Advantage Estimation (**GAE**):

1. Collect rewards and compute values in parallel for the entire trajectory using the current value network

$$[\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_{T-1}] \qquad [V_0, V_1, \ldots, V_T] \quad \text{where } V_t = V_{\pi_{\text{old}}}(s_t)$$

2. $$\hat{A}_t^{\text{GAE}} = \delta_t + \gamma\lambda \hat{A}_{t+1}^{\text{GAE}} \quad \text{where } \delta_t = \mathcal{R}_{s_t}^{a_t} + \gamma V_{t+1} - V_t, \hat{A}_T = 0$$

3. $$\hat{R}_t^{\text{GAE}} = \hat{A}_t^{\text{GAE}} + V_t$$

The final loss is $$\mathcal{L}_V(\phi) = \frac{1}{2}\left(V_\phi(s_t) - \hat{R}_t^{\text{GAE}}\right)^2$$

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \text{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \text{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

- Natural Policy Gradient [26]: Preconditioning with Fisher-Information Matrix

$$\mathbf{F} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right] \qquad D_{\text{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^\top \mathbf{F}(\theta' - \theta)$$

$$\theta' \leftarrow \text{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } D_{\text{KL}}(\pi_{\theta'} \| \pi_\theta) \leq \epsilon$$

$$\theta' \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \mathrm{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

- Natural Policy Gradient [26]: Preconditioning with Fisher-Information Matrix

  $\mathbf{F} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right]$ $\qquad$ $D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^\top \mathbf{F}(\theta' - \theta)$

$$\theta' \leftarrow \mathrm{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \leq \epsilon$$

$$\theta' \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

- Trust Region Policy Optimization (TRPO) [27]: select ϵ and solve for the optimal α while solving $\mathbf{F}^{-1}\nabla_\theta J(\theta)$ using conjugate gradient

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [28]: turn the constrained optimization in TRPO into an unconstrained one (with clipped loss) and use modern 1st-order optimizers like Adam.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [28]: turn the constrained optimization in TRPO into an unconstrained one (with clipped loss) and use modern 1st-order optimizers like Adam.

Replace reward-to-go with the advantage:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\hat{A}_t\right]\\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}\hat{A}_t\right]\end{aligned}$$

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [28]: turn the constrained optimization in TRPO into an unconstrained one (with clipped loss) and use modern 1st-order optimizers like Adam.

Replace reward-to-go with the advantage:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\hat{A}_t\right]$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}\hat{A}_t\right]$$

The gradient corresponds to the Conservative Policy Iteration (CPI) objective:

$$L^{\text{CPI}}(\theta) = \mathbb{E}_t\left[r_t(\theta)\hat{A}_t\right] \quad \text{where } r_t(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}$$

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [28]: turn the constrained optimization in TRPO into an unconstrained one (with clipped loss) and use modern 1st-order optimizers like Adam.

Replace reward-to-go with the advantage:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)\hat{A}_t\right]$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} \frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}\hat{A}_t\right]$$

The gradient corresponds to the Conservative Policy Iteration (CPI) objective:

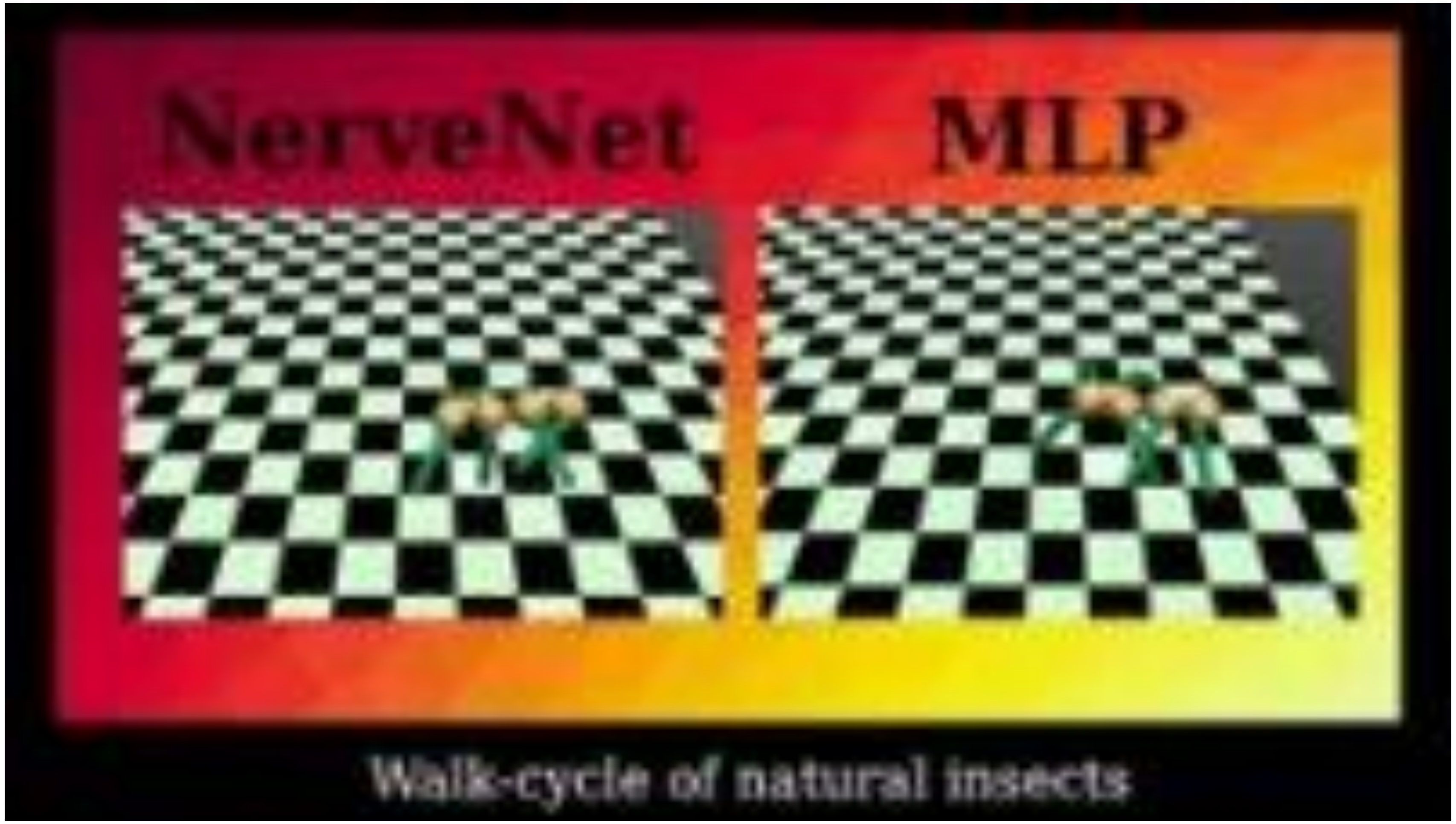
$$L^{\text{CPI}}(\theta) = \mathbb{E}_t\left[r_t(\theta)\hat{A}_t\right] \qquad \text{where } r_t(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}$$

Following TRPO, we constrain the change of distribution by clipping the ratio:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

Demo

Simulated Continuous Control in Mujoco using PPO and Graph Neural Networks:



Outline

Reinforcement Learning from Human Feedback (RLHF)

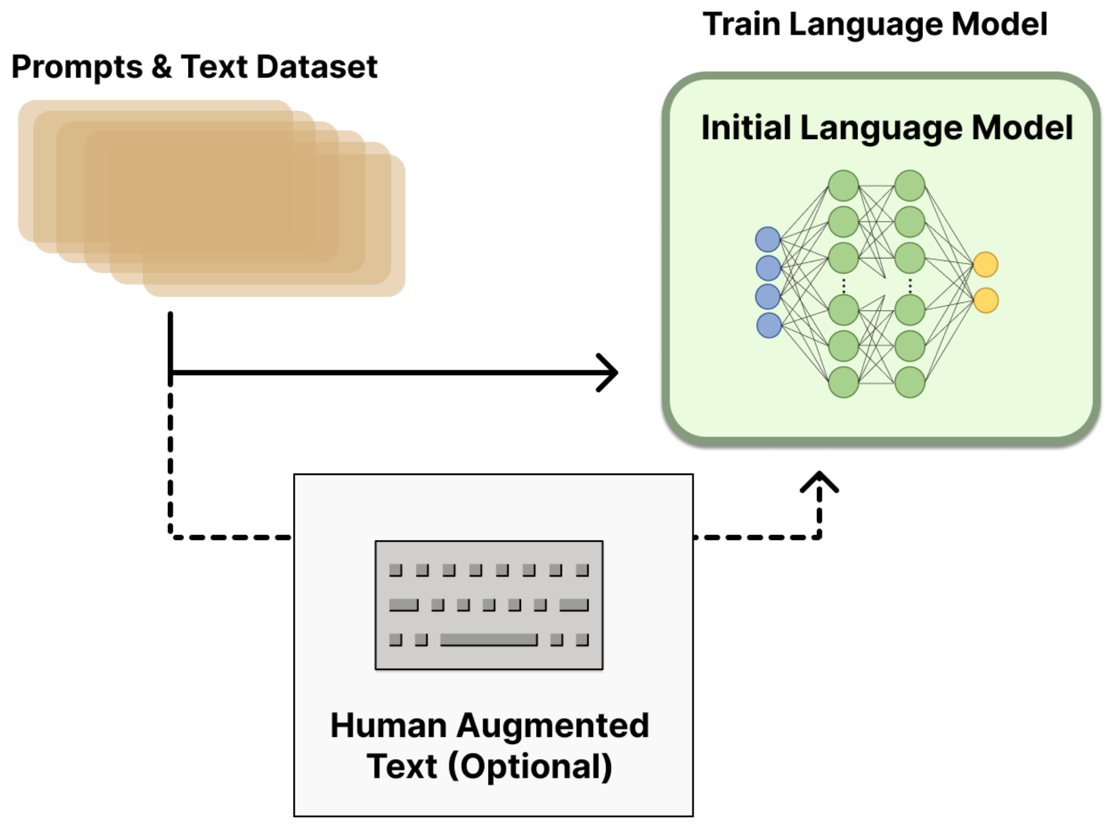
Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

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

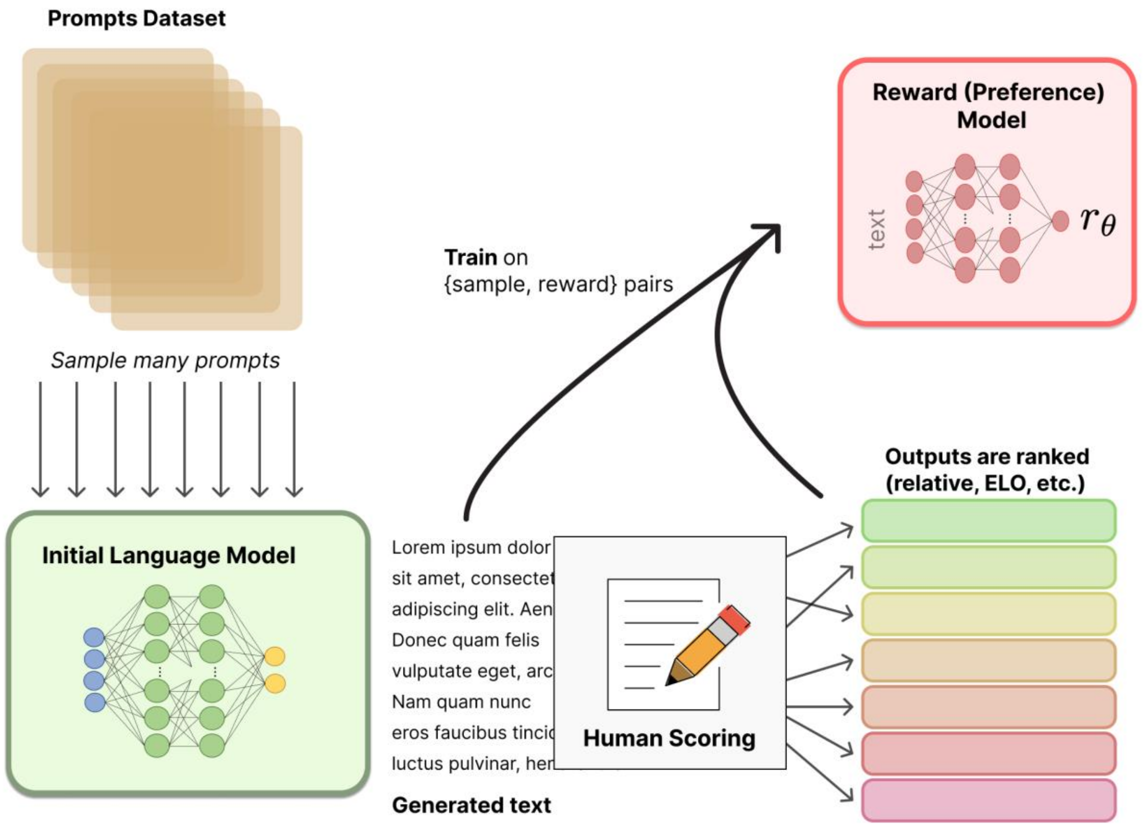

Prompts Dataset
Sample many prompts
Initial Language Model
Lorem ipsum dolor sit amet, consectet adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tincid luctus pulvinar, her
Generated text
Human Scoring
Outputs are ranked (relative, ELO, etc.)
Train on {sample, reward} pairs
Reward (Preference) Model
text
r_θ

Image Credit: [21]

Reinforcement Learning from Human Feedback (RLHF)

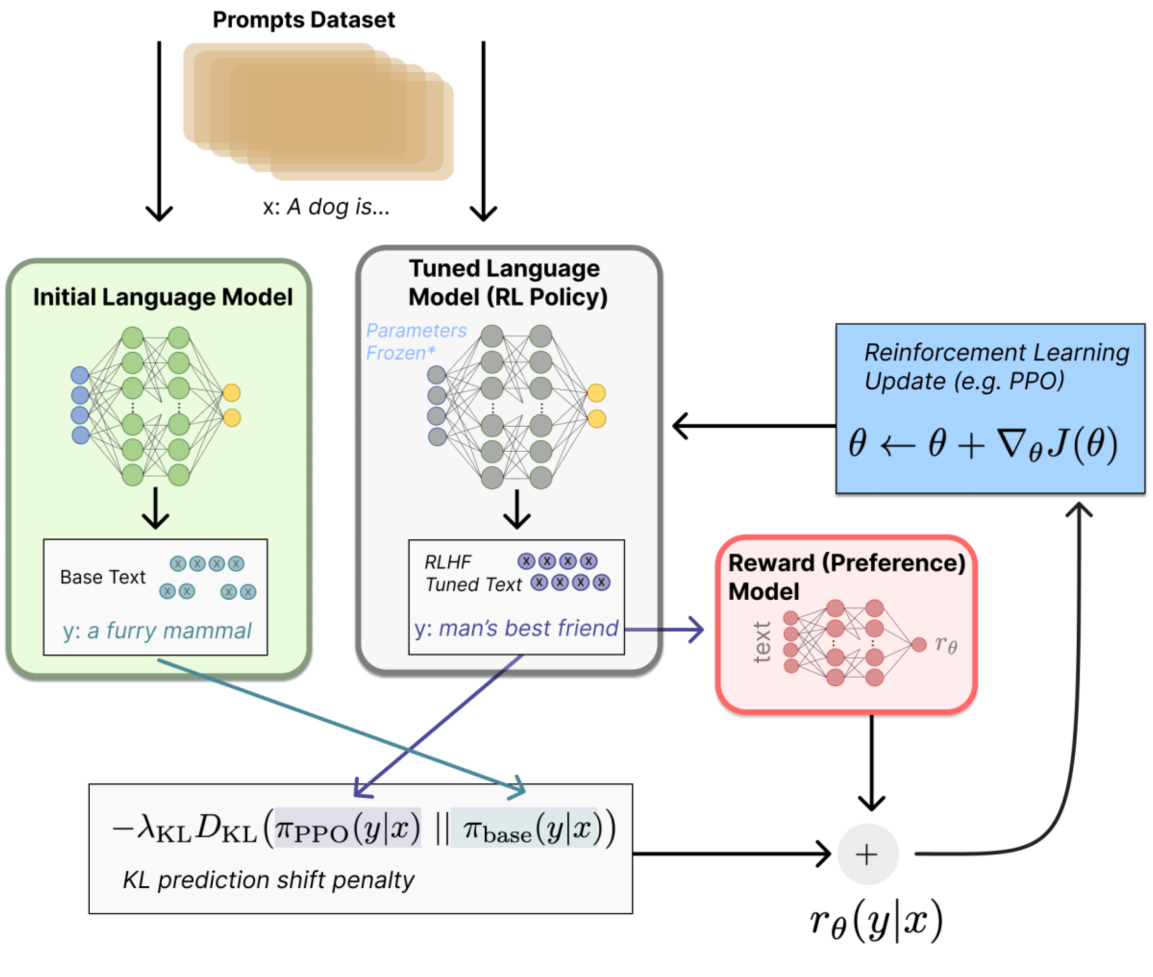
Fine-tuning LLMs with RLHF [e.g., 20] can align them with human values!

It involves three steps:

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

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

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



Prompts Dataset
x: A dog is...
Initial Language Model
Base Text
y: a furry mammal
Tuned Language Model (RL Policy)
Parameters Frozen*
RLHF Tuned Text
y: man's best friend
Reinforcement Learning Update (e.g. PPO)
$\theta \leftarrow \theta + \nabla_\theta J(\theta)$
Reward (Preference) Model
text
r_θ
$-\lambda_{\text{KL}} D_{\text{KL}}(\pi_{\text{PPO}}(y|x) \,||\, \pi_{\text{base}}(y|x))$
KL prediction shift penalty
$+$
$r_\theta(y|x)$

Image Credit: [21]

Outline

- Introduction & Background
- Models
  - Tokenization
  - Rotary Positional Encoding
  - Architecture
- Sampling
- Pre-Training & Scaling Law
- Post-Training
  - Low Rank Adaptation (LoRA)
  - Reinforcement Learning: Basics, Policy Gradients, Actor-Critic, and PPO
  - Reinforcement Learning from Human Feedback (RLHF)
- **Prompting**

Prompting

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

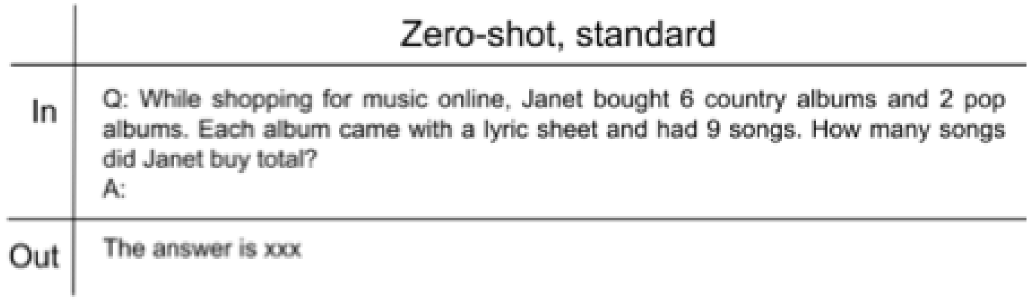
Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

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!

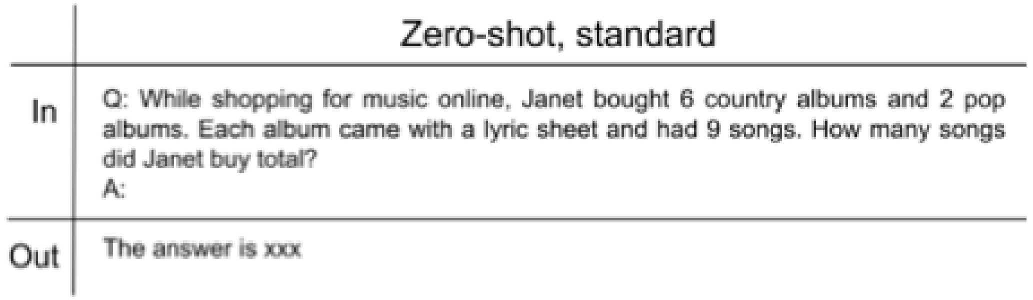
Image Credit: [22]

Zero/Few-shot Prompting

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

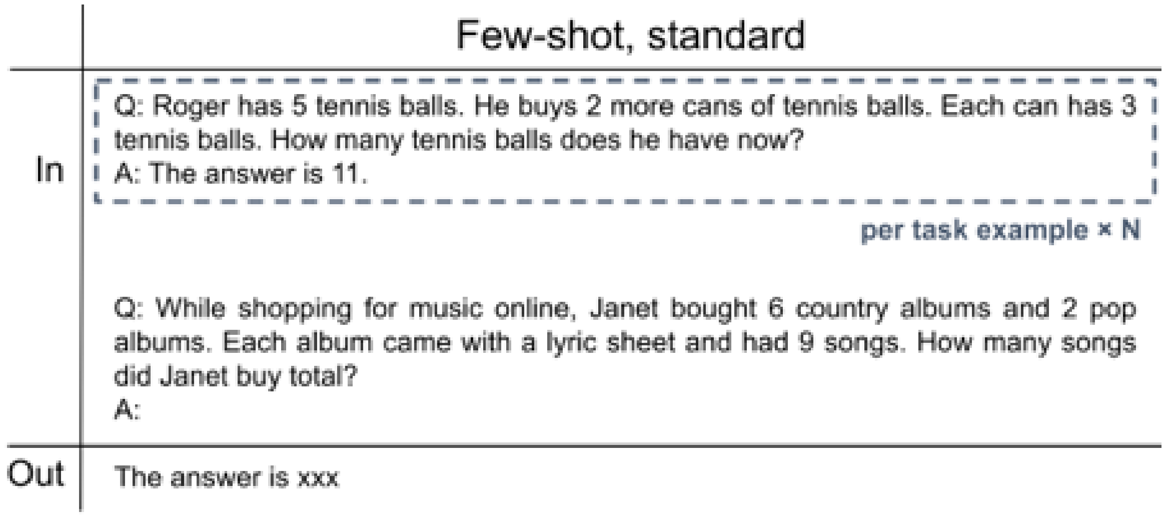
Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot prompting:



	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!



	Few-shot, standard
In	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. per task 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:
Out	The answer is xxx

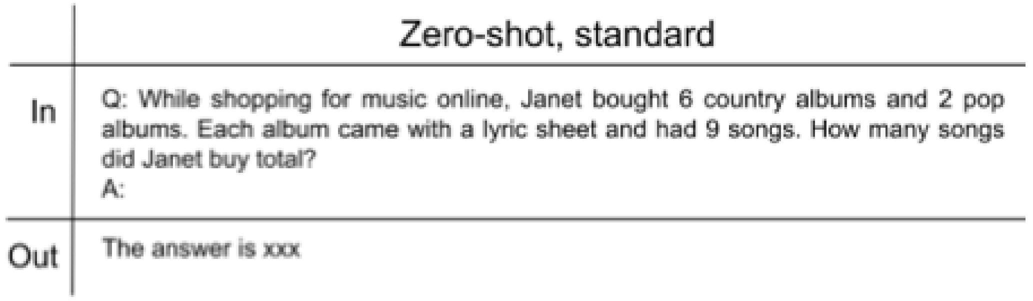
Image Credit: [22]

Zero/Few-shot Prompting

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

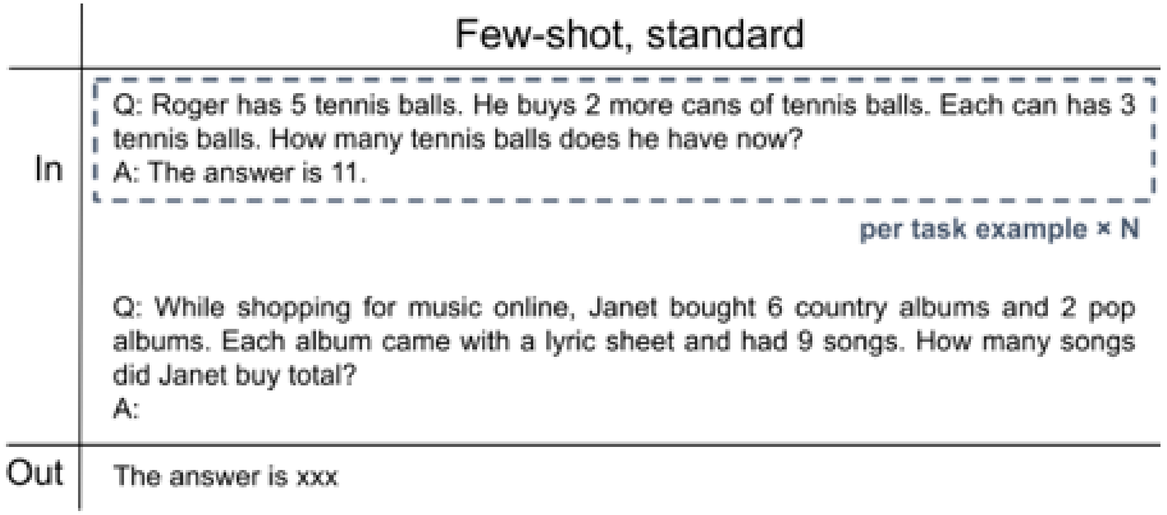
Designing effective prompts to instruct LLMs to perform a desired task is often referred to as *prompt engineering*.

Zero/Few-shot prompting:



	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!



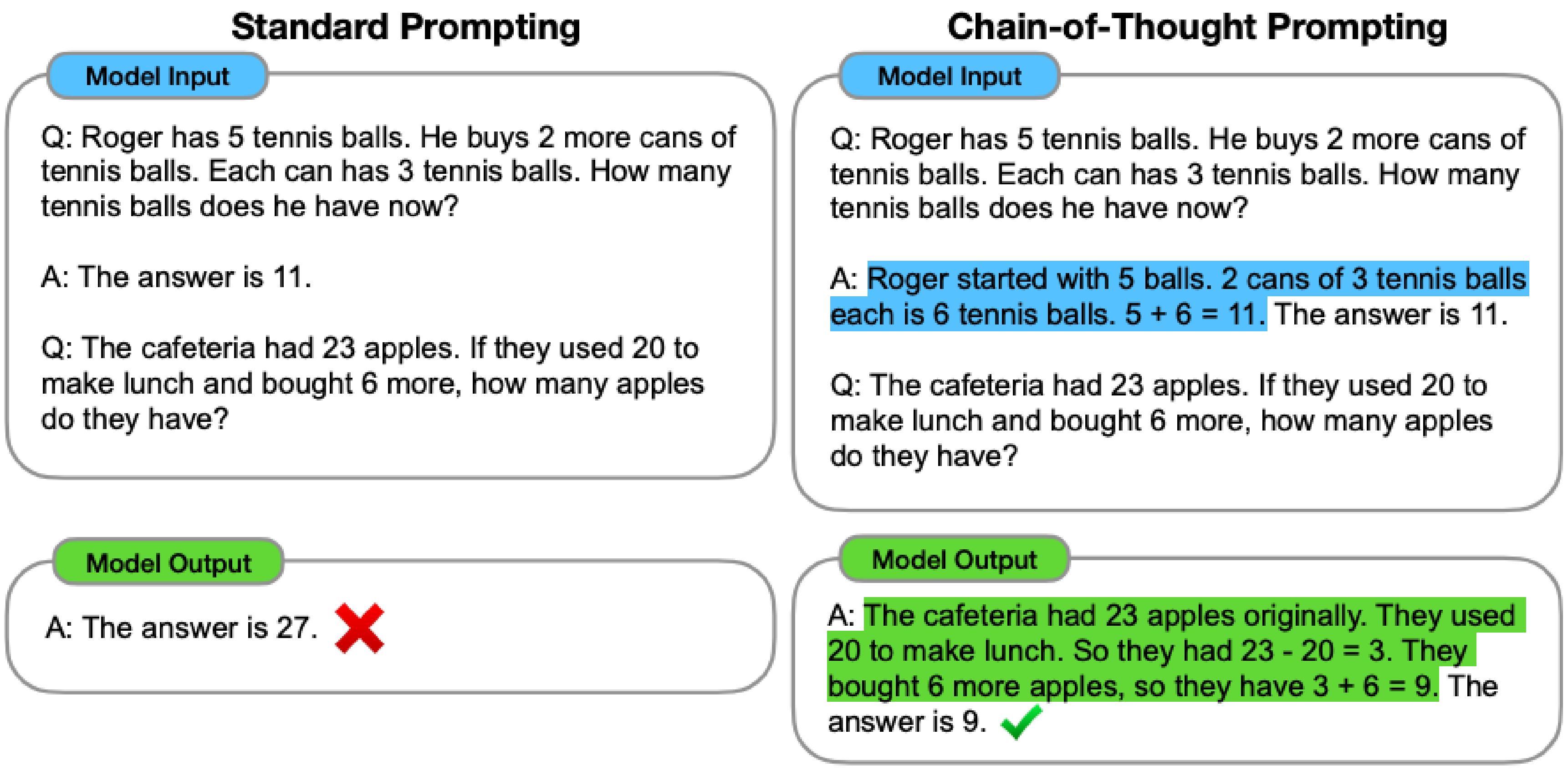
	Few-shot, standard
In	Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now? A: The answer is 11. per task example × N Q: While shopping for music online, Janet bought 6 country albums and 2 pop albums. Each album came with a lyric sheet and had 9 songs. How many songs did Janet buy total? A:
Out	The answer is xxx

Ask with some guiding examples!

Image Credit: [22]

Chain-of-Thought (CoT) Prompting

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



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?

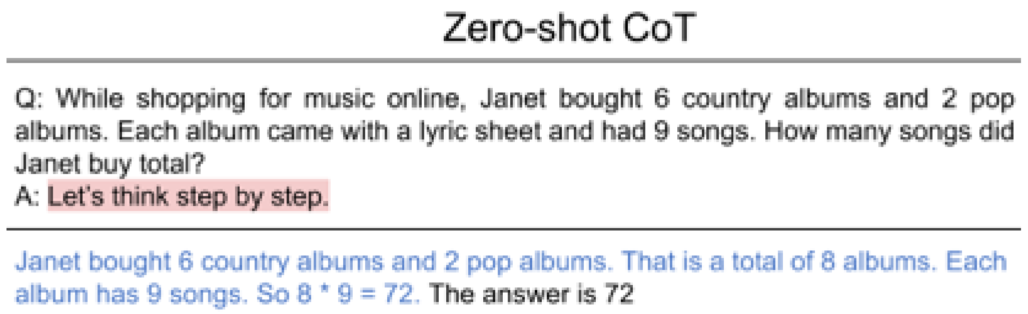
Model Output

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. ✓

Image Credit: [23]

Chain-of-Thought (CoT) Prompting

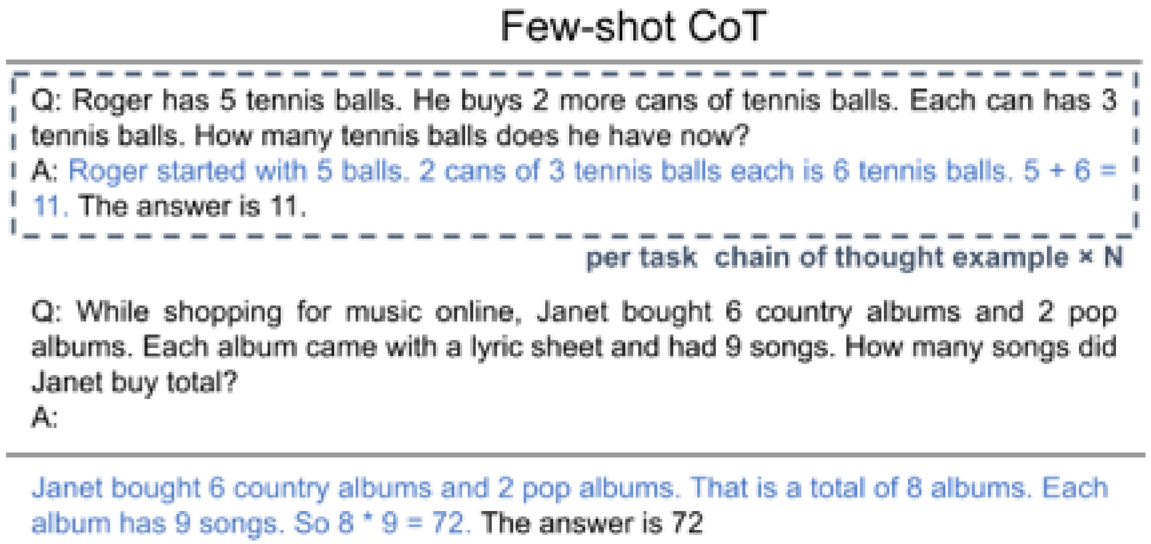
CoT prompting [23] enables complex reasoning capabilities through intermediate reasoning steps:



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.

[25] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement learning, pp.5-32.

[26] Peters, J. and Schaal, S., 2008. Reinforcement learning of motor skills with policy gradients. Neural networks, 21(4), pp.682-697.

[27] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

[28] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Questions?