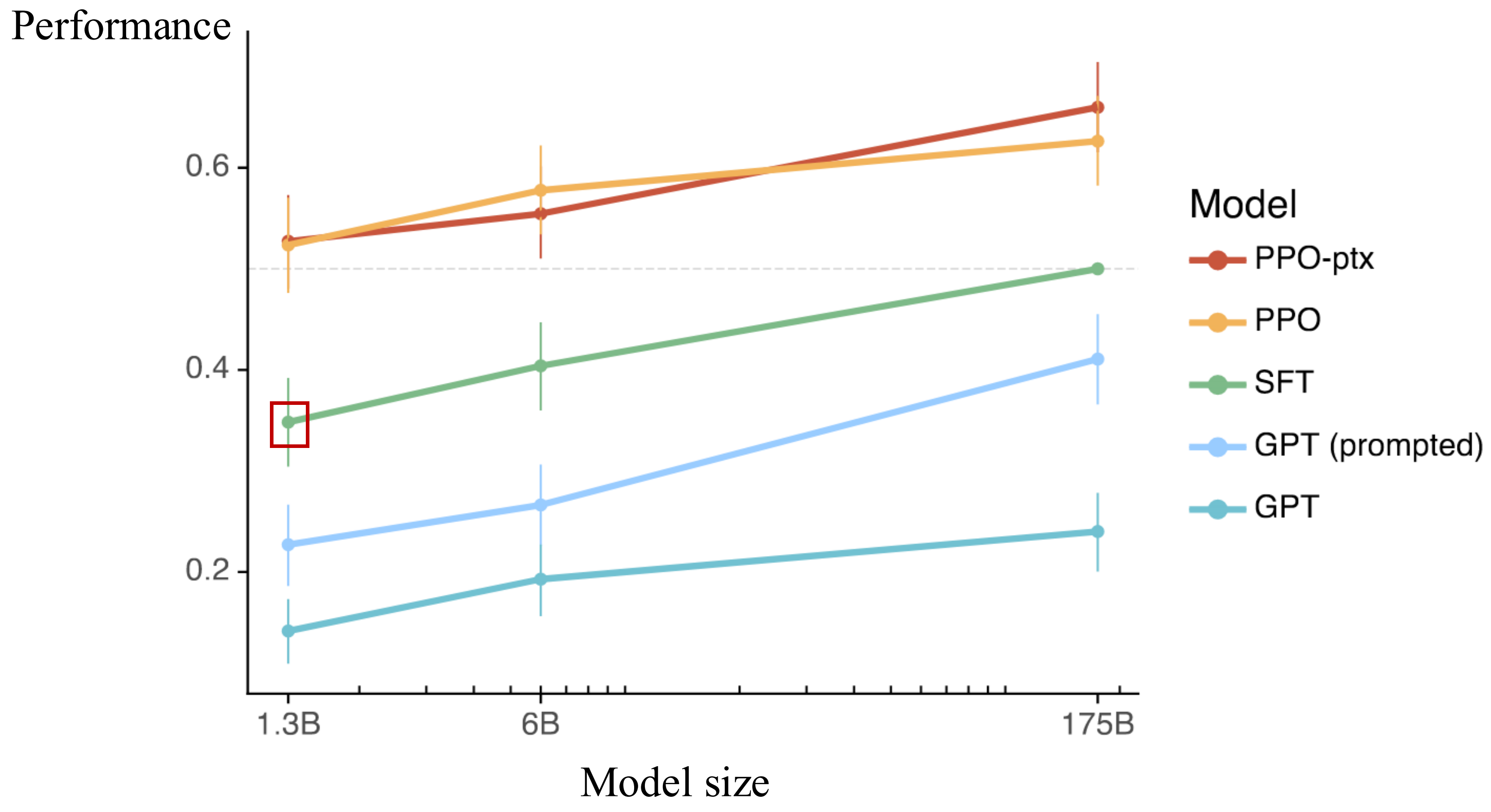
Understanding LLM from Methods to Practices

Scaling Effectiveness of LLM

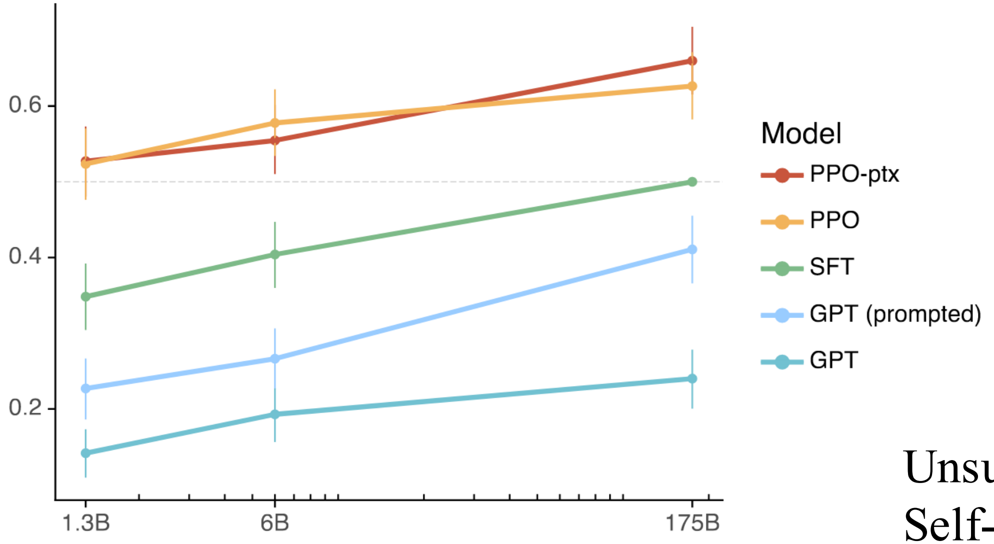


Three observations

- Scaling model size contributes to the performance
- Supervised FineTuning (SFT) and Proximal Policy Optimization (PPO) contribute to the performance
- 1.3B GPT-3 with SFT surpasses the vanilla 175B GPT-3

[*] Ouyang, Long, Jeffrey Wu, et al. "Training language models to follow instructions with human feedback." Advances in Neural Information Processing Systems 35 (2022): 27730-27744.

Scaling Effectiveness of LLM



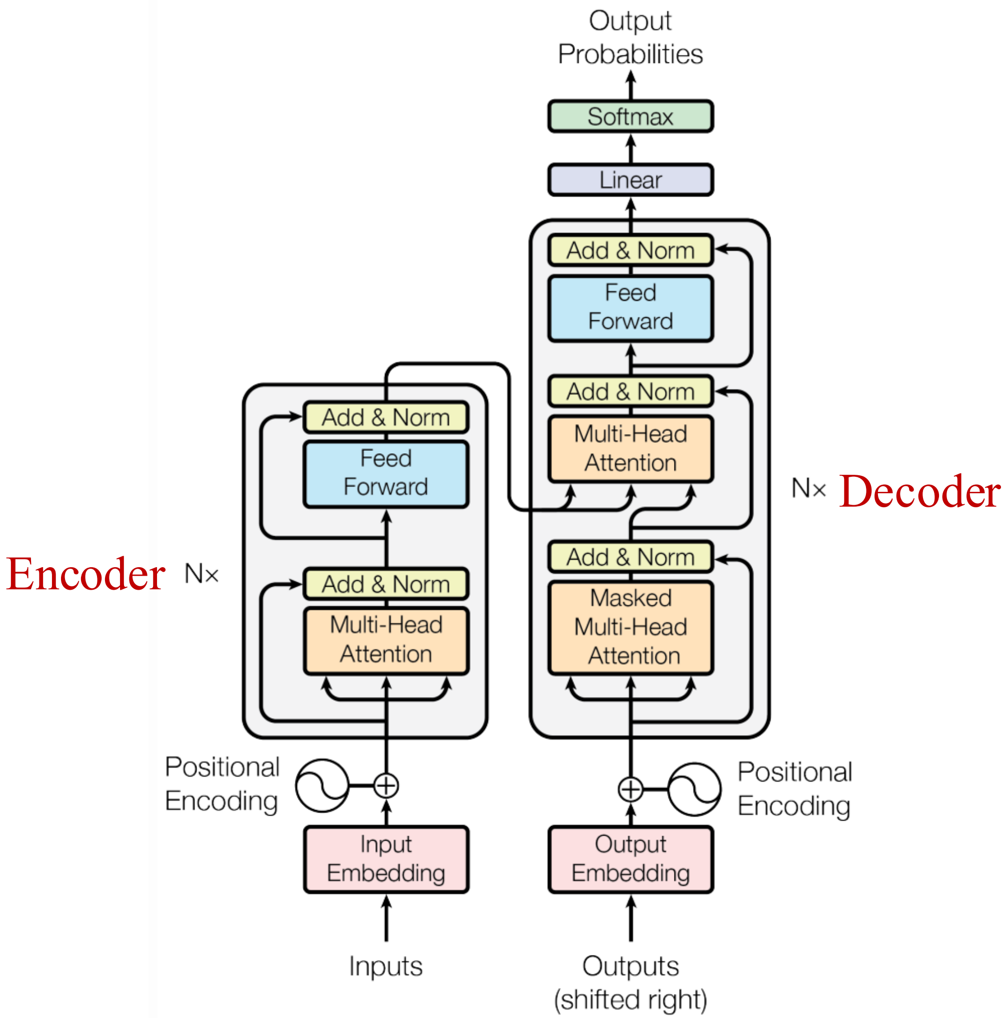
GPT-3 → SFT Method → PPO Method → InstructGPT

GPT-3	SFT Method	PPO Method
Pretraining (Memorize)	Finetuning (Imitate)	Reinforce (Reasoning)
Unsupervised Learning/ Self-supervised Learning	Supervised Learning	Reinforcement Learning

Outline

- Pretraining (Memorize)
- Finetuning (Imitate)
- Reinforce (Reasoning)
- Engeneering Practices

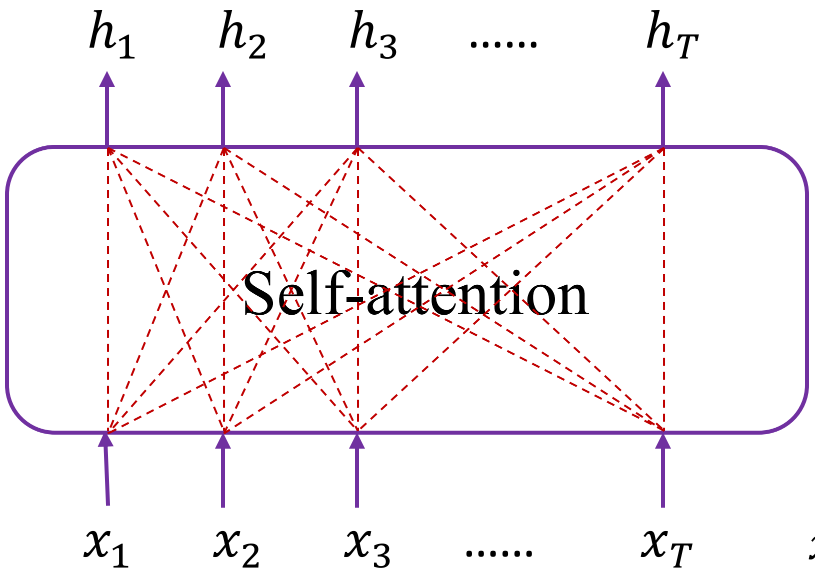
Encoder-decoder Architecture of Transformer



Goal of encoder: get contextualized representation for each token

Self-attention in encoder

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

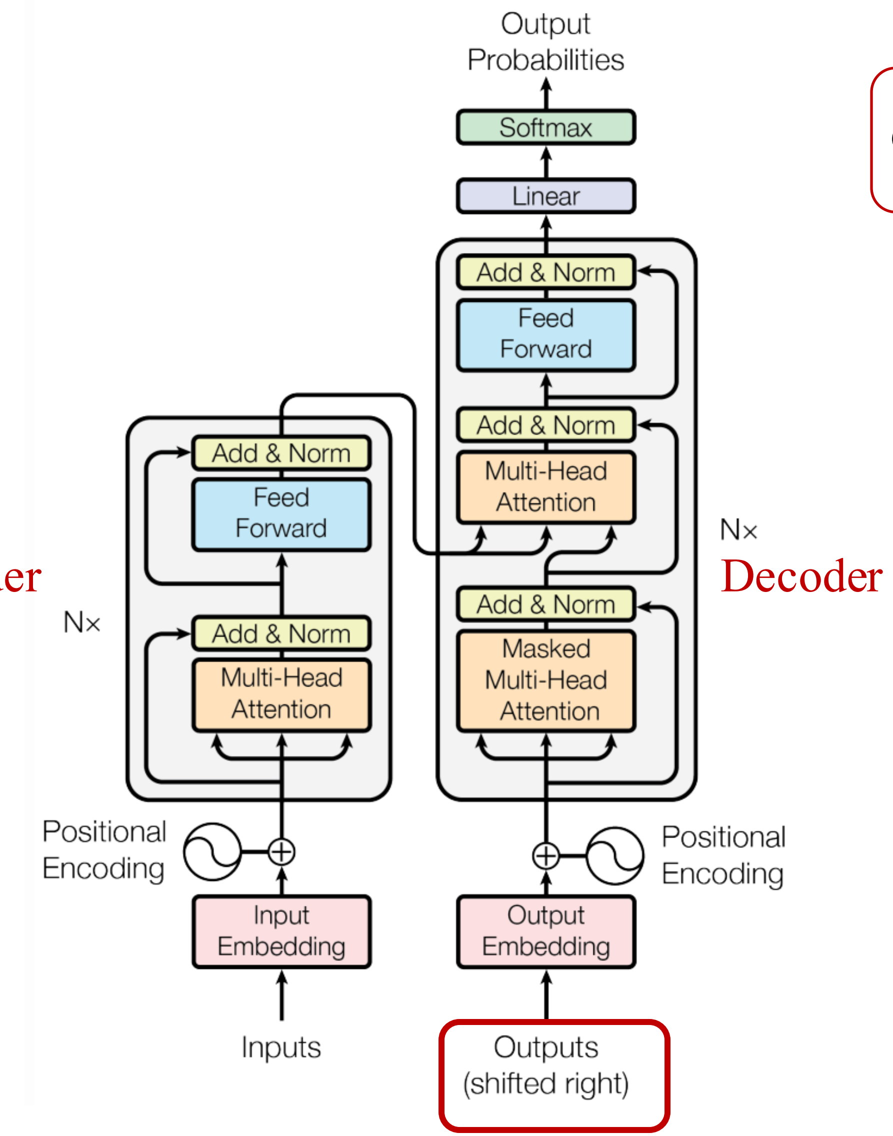


h_t: hidden representation of token t

h_t is related to every input token

x_t: embedding of token t

Encoder-decoder Architecture of Transformer



Goal of decoder: accomplish the next token prediction task autoregressively

Suppose the target sequence is [“A”, “B”, “C”]

input of decoder: [<SOS>, “A”, “B”, “C”]

label: [“A”, “B”, “C”, <EOS>]

Task of decoder:

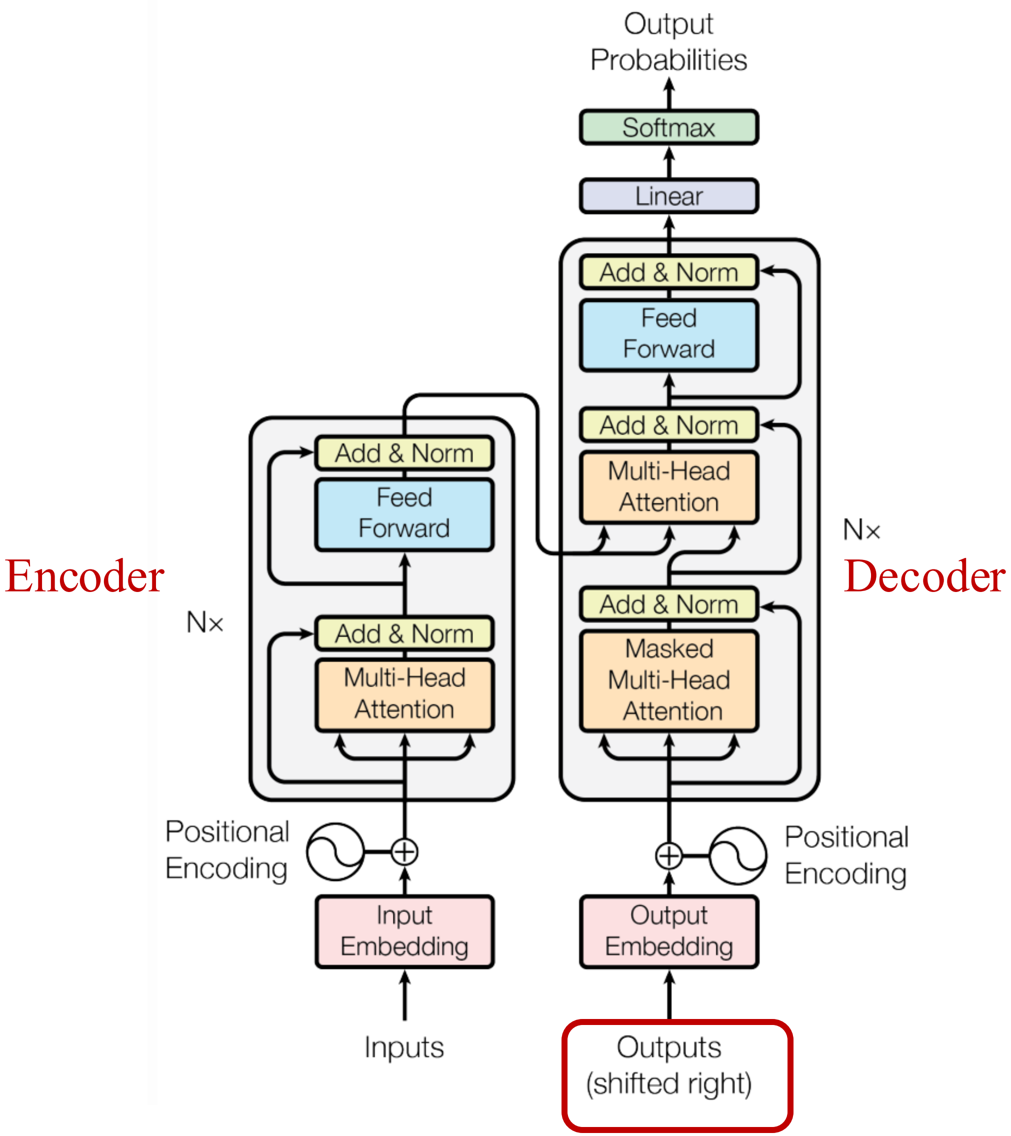
input [<SOS>], predict “A”

input [<SOS>, “A”], predict “B”

input [<SOS>, “A”, “B”], predict “C”

input [<SOS>, “A”, “B”, “C”], predict <EOS>

Encoder-decoder Architecture of Transformer



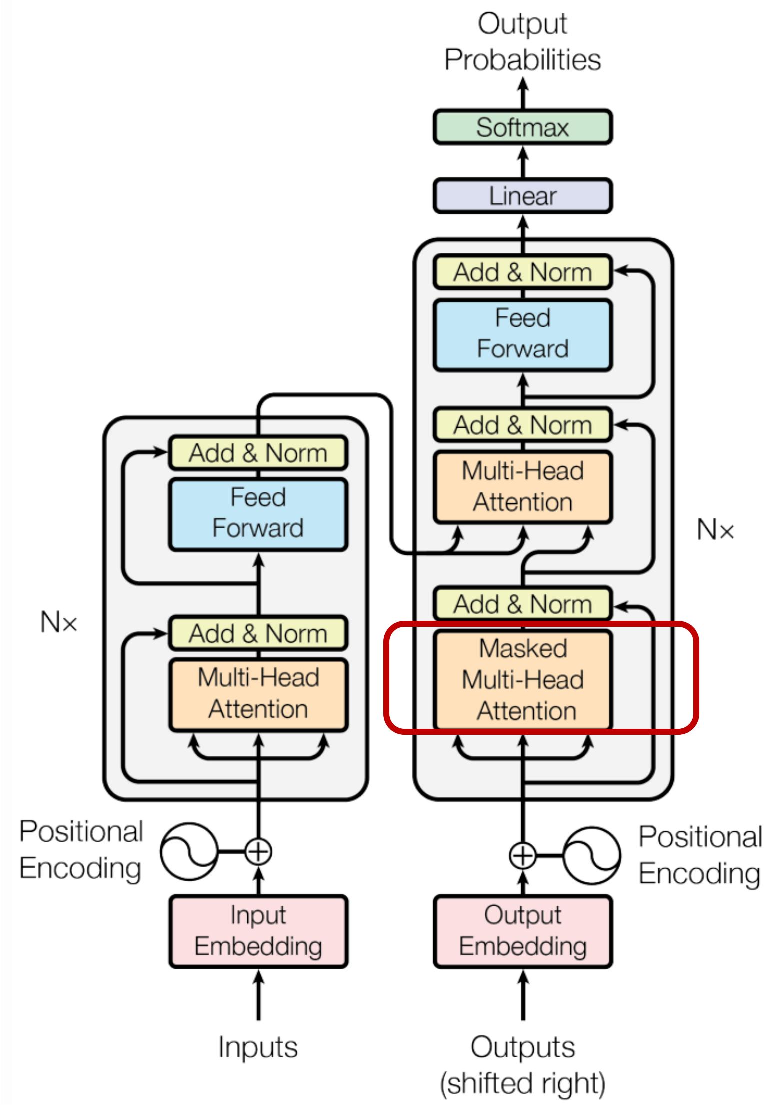
Next token prediction task in decoder part

input of decoder: $[< \mathrm{SOS} >, x_1, x_2, \ldots, x_T]$ SOS: start of sentence

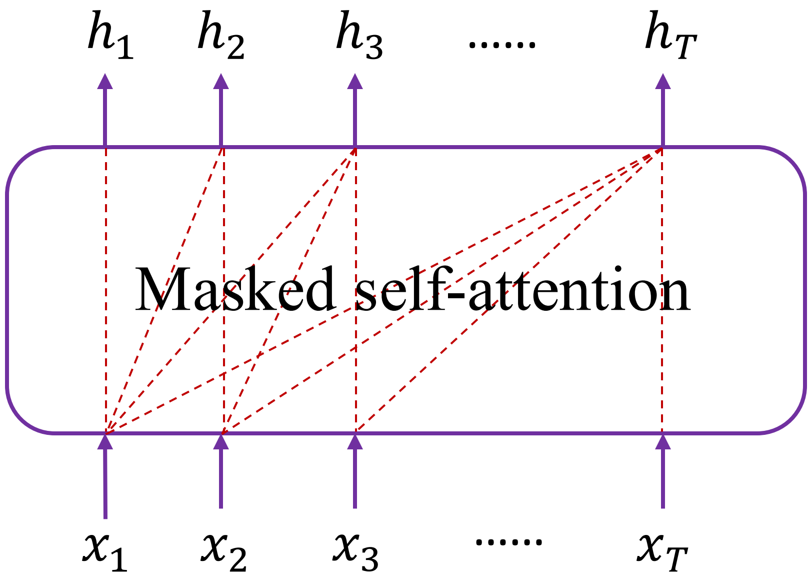
label: $[x_1, x_2, x_3, \ldots, x_T, < \mathrm{EOS} >]$ EOS: end of sentence

- Vanilla implementation: Given previous $t-1$ tokens, predict the t-th token
- Efficient implementation: Given all T tokens as inputs, predict all the outputs simultaneously using masked self-attention
    - Benefits: Fully exploits parallel computatioin capabilities

Encoder-decoder Architecture of Transformer

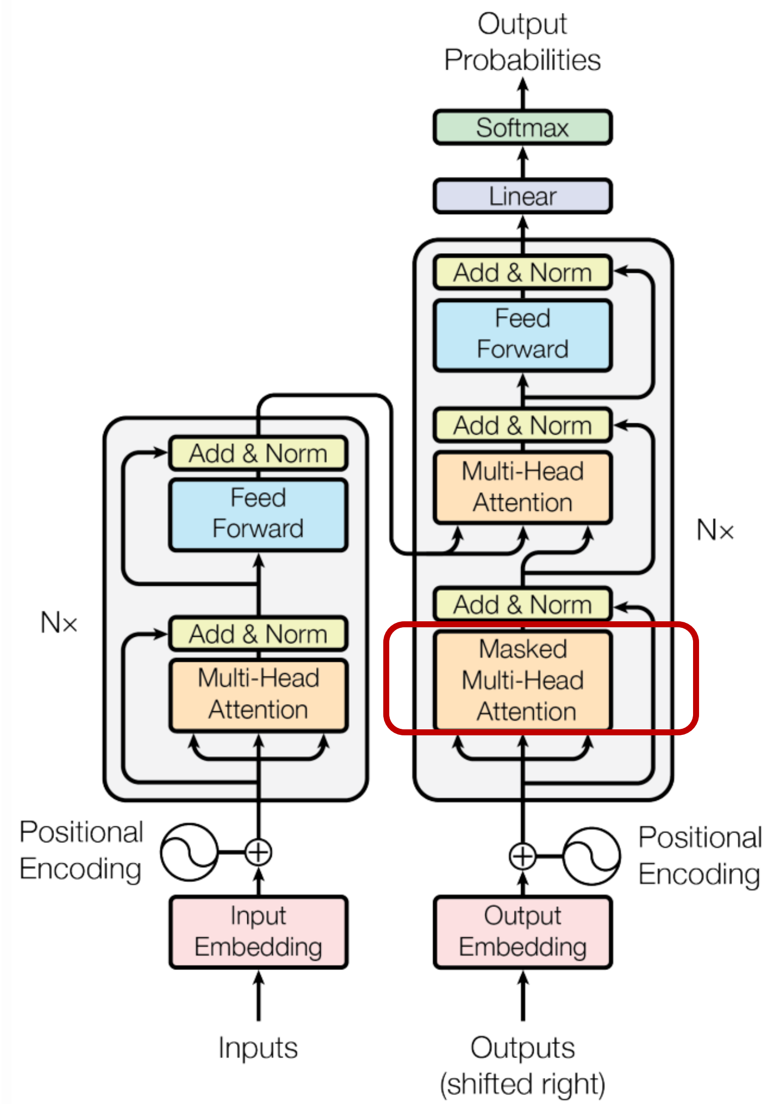


Masked self-attention in decoder



- h_t is only related to $[x_1, x_2, \dots, x_t]$

Encoder-decoder Architecture of Transformer



Efficient implementation of masked self-attention in decoder

$$\mathrm{MaskedAttention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

- M is causal mask with upper triangular part is $-\infty$, lower triangular part is 0, for example:

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Encoder-decoder Architecture of Transformer

$$\text{MaskedAttention}(Q,K,V) = \text{softmax}\left(\underbrace{\boxed{\frac{QK^T}{\sqrt{d_k}}}}_{S} + M\right)V \qquad M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

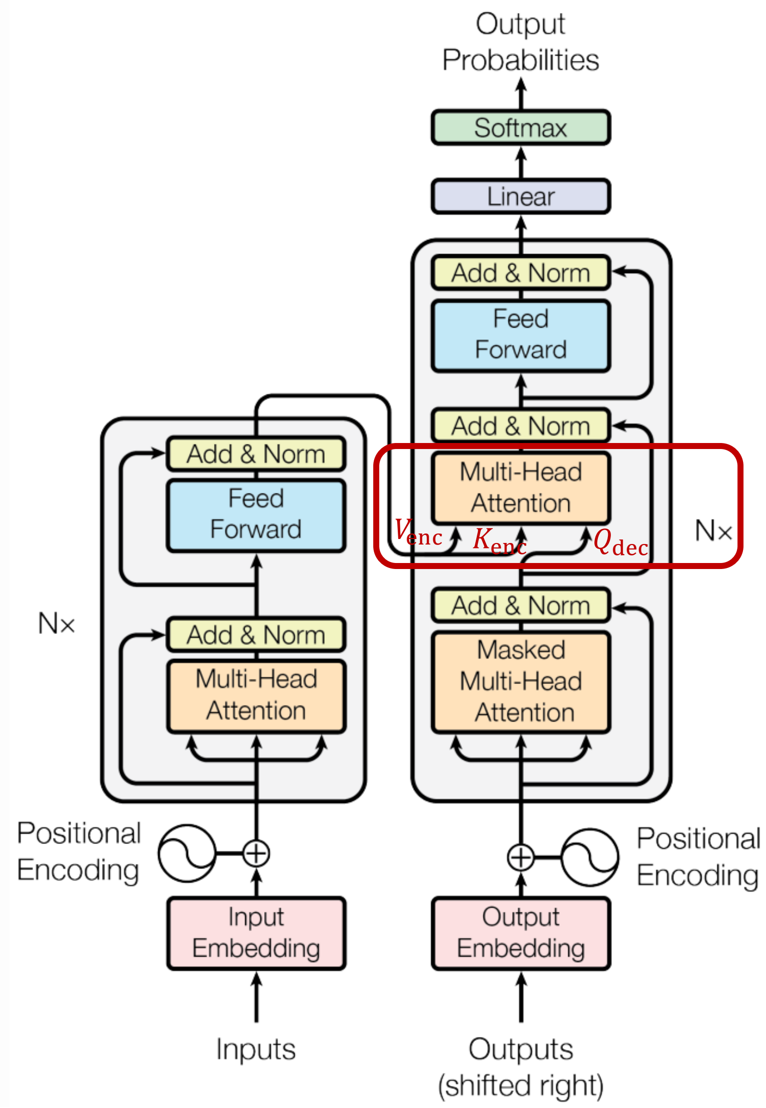
$$\boxed{\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad e^{-\infty} = 0}$$

$$S = \frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} \qquad S + M = \begin{bmatrix} s_{11} & -\infty & -\infty & -\infty \\ s_{21} & s_{22} & -\infty & -\infty \\ s_{31} & s_{32} & s_{33} & -\infty \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} \qquad \text{softmax}(S+M) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 \\ c_{31} & c_{32} & c_{33} & 0 \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

$$\text{softmax}(S + M)V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ c_{21} & c_{22} & 0 & 0 \\ c_{31} & c_{32} & c_{33} & 0 \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} v_1 \\ c_{21}v_1 + c_{22}v_2 \\ c_{31}v_1 + c_{32}v_2 + c_{33}v_3 \\ c_{41}v_1 + c_{42}v_2 + c_{43}v_3 + c_{44}v_4 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$

- v_t is row vector
- h_t is only related to tokens before t

Encoder-decoder Architecture of Transformer



Goal of cross attention: compute the relationship between h_t from decoder and contextualized representation from encoder

$$\mathrm{softmax}(S+M)V = \begin{bmatrix} v_1 \\ c_{21}v_1 + c_{22}v_2 \\ c_{31}v_1 + c_{32}v_2 + c_{33}v_3 \\ c_{41}v_1 + c_{42}v_2 + c_{43}v_3 + c_{44}v_4 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = H_{\mathrm{dec}}$$

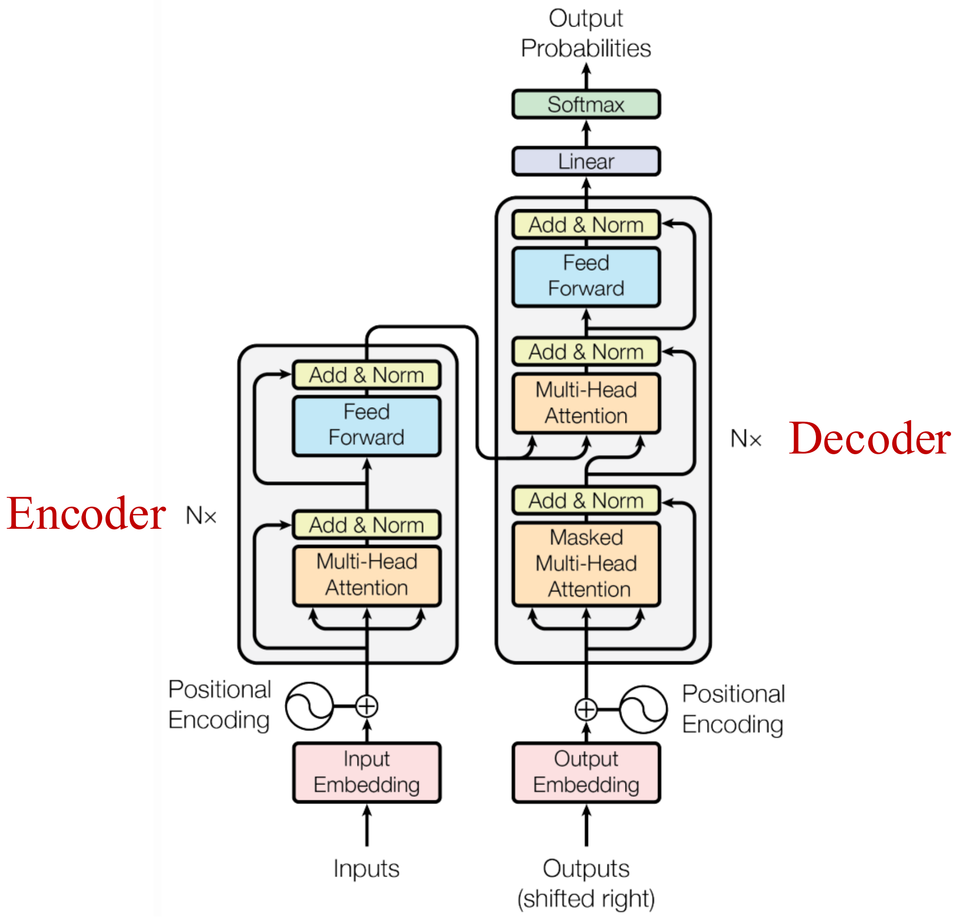
Query (from decoder): $Q_{\mathrm{dec}} = H_{\mathrm{dec}} W_Q$

Key (from encoder): $K_{\mathrm{enc}} = H_{\mathrm{enc}} W_K$

Value (from encoder): $V_{\mathrm{enc}} = H_{\mathrm{enc}} W_V$

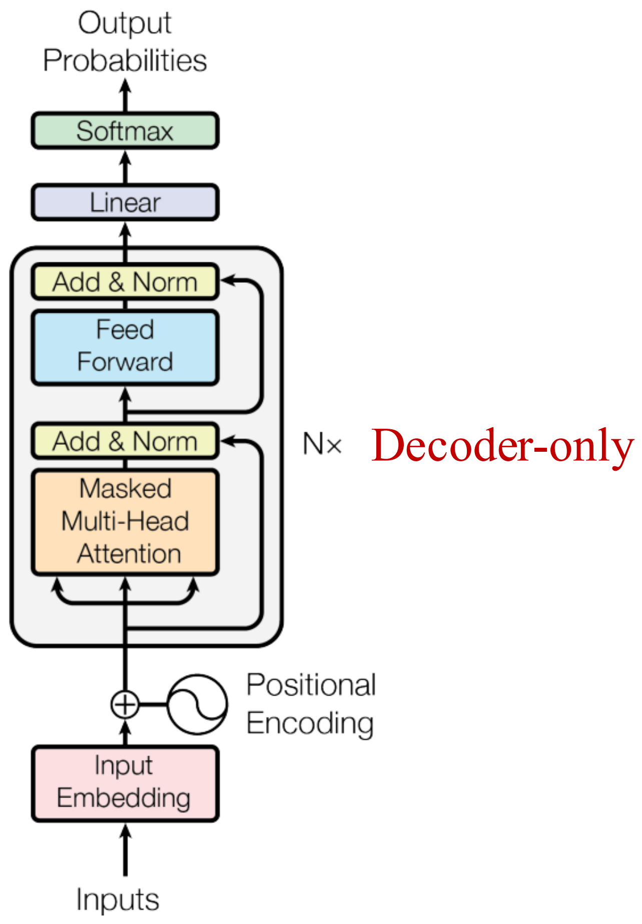
$$\mathrm{CrossAttention}(Q_{\mathrm{dec}}, K_{\mathrm{enc}}, V_{\mathrm{enc}}) = \mathrm{softmax}\left(\frac{Q_{\mathrm{dec}} K_{\mathrm{enc}}^T}{\sqrt{d_k}}\right) V_{\mathrm{enc}}$$

Decoder-only Architecture of Transformer



Encoder-decoder architecture

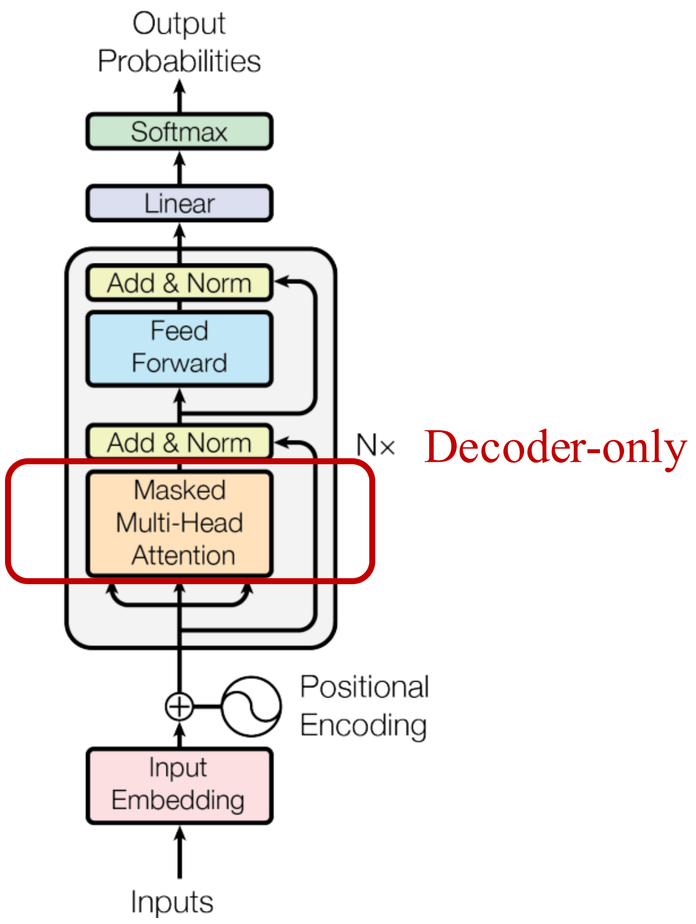
Example: vanilla Transformer



Decoder-only architecture

Example: GPT, LLaMA

Decoder-only Architecture of Transformer



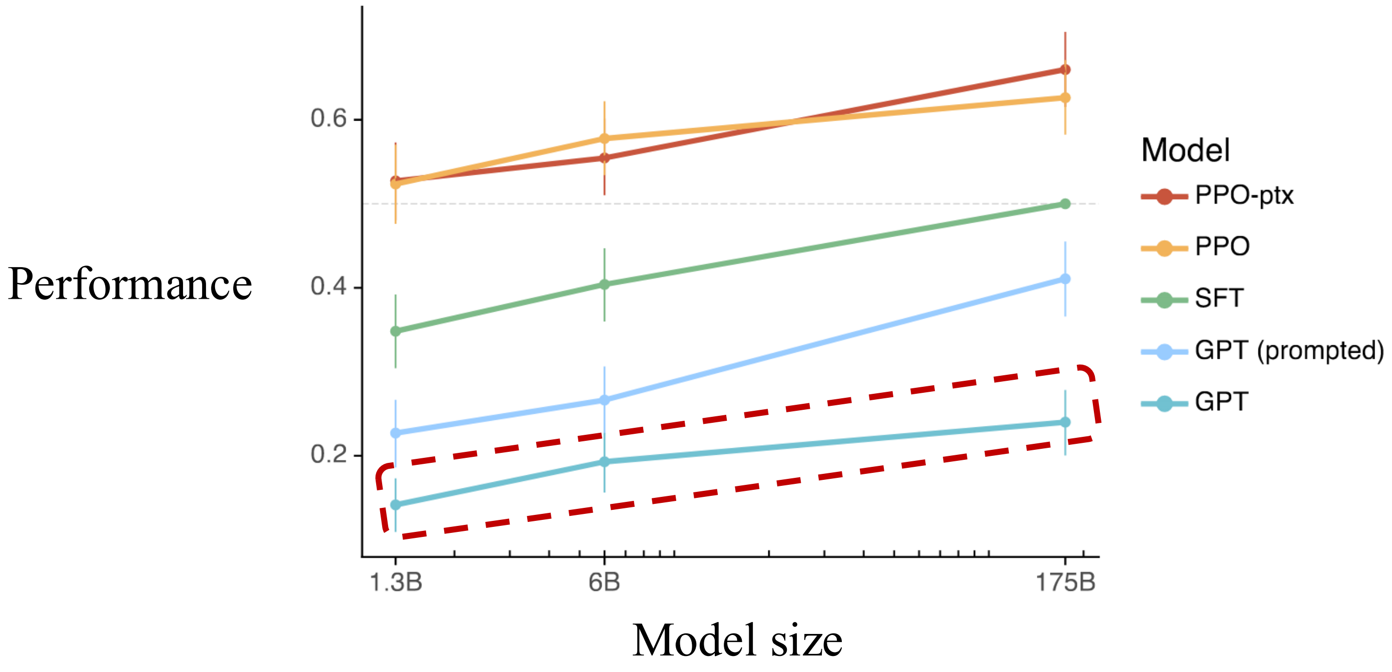
$$\text{MaskedAttention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Advantages over encoder-decoder architecture

- Simplified architecture
- Appropriate for the generative tasks

Performance of Decoder-only Transformer

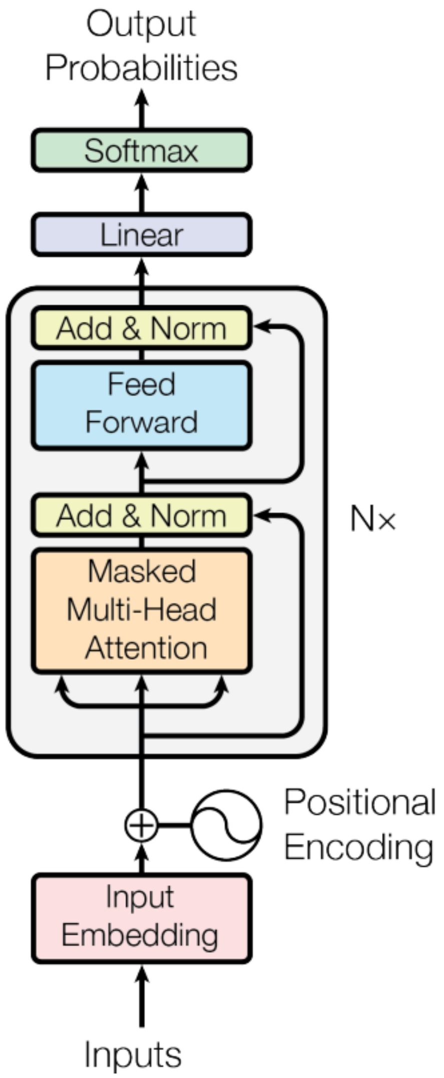


- Performance increases with the model size, but the increment is limited

Outline

- Pretraining (Memorize)
- Finetuning (Imitate)
- Reinforce (Reasoning)
- Engeneering Practices

Supervised FineTuning (SFT)



Prompt and response of Supervised FineTuning (SFT)

Prompt: $[x_1, x_2, \dots, x_T]$

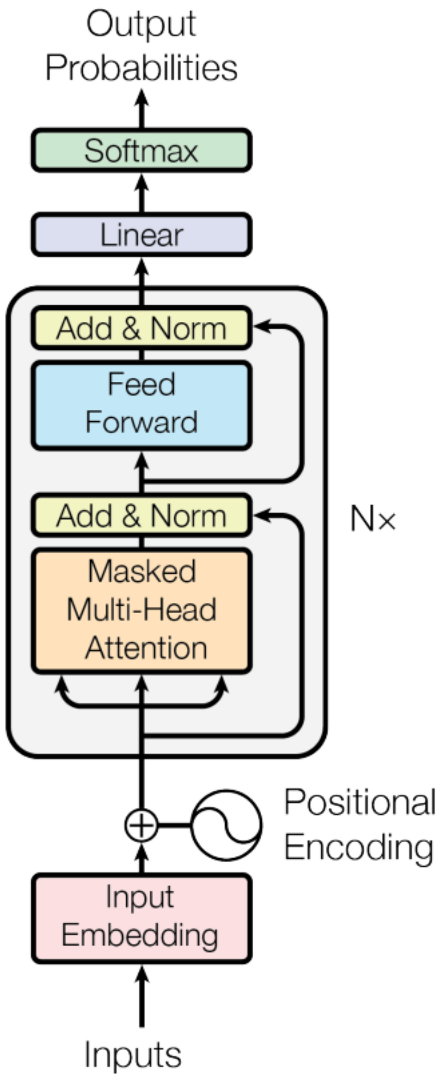
Response: $[y_1, y_2, \dots, y_K]$, Provided by annotator

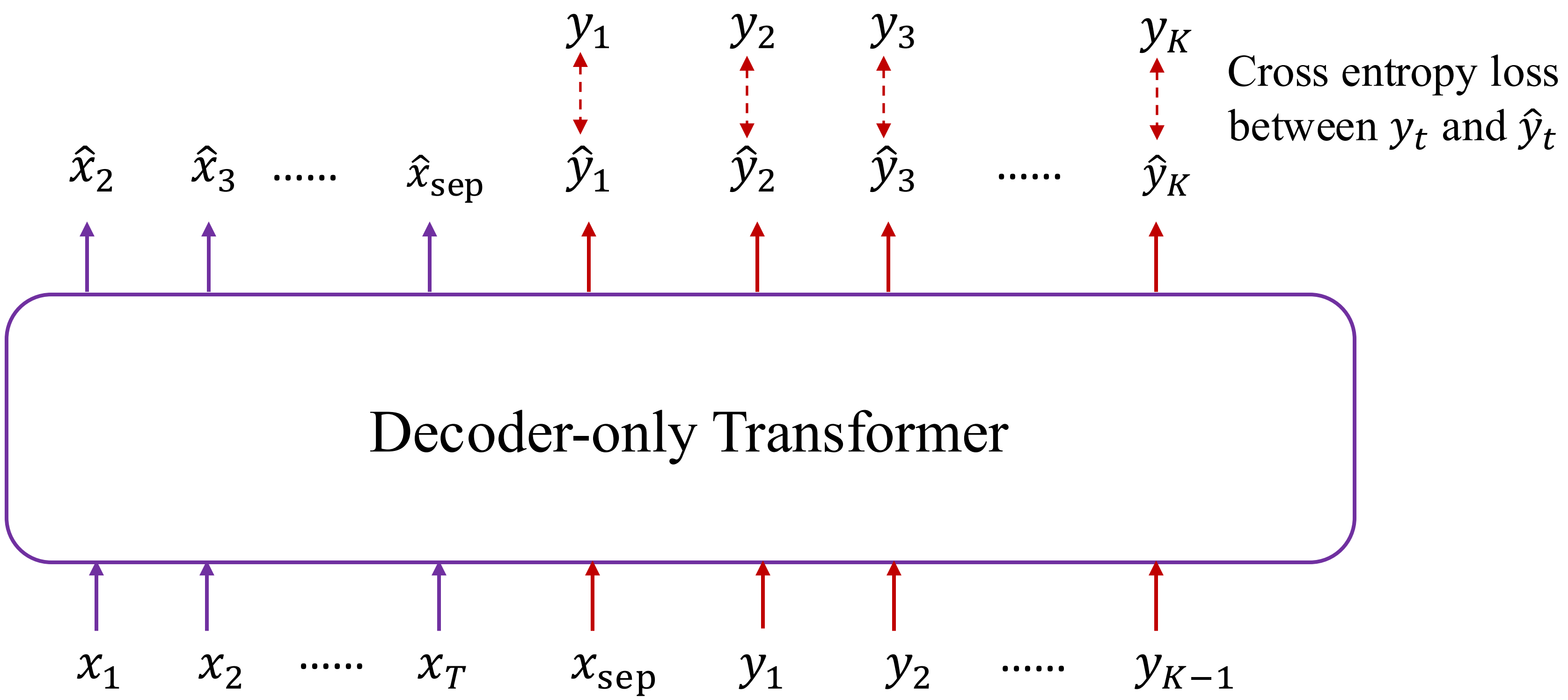
Training data of STF

$$[x_1, x_2, \dots, x_T, < \text{sep} >, y_1, y_2, \dots, y_K]$$

- Concatenate prompt and response to form training sample
- SFT is also an autoregressive generative task with the input consists of concatenated prompt and response

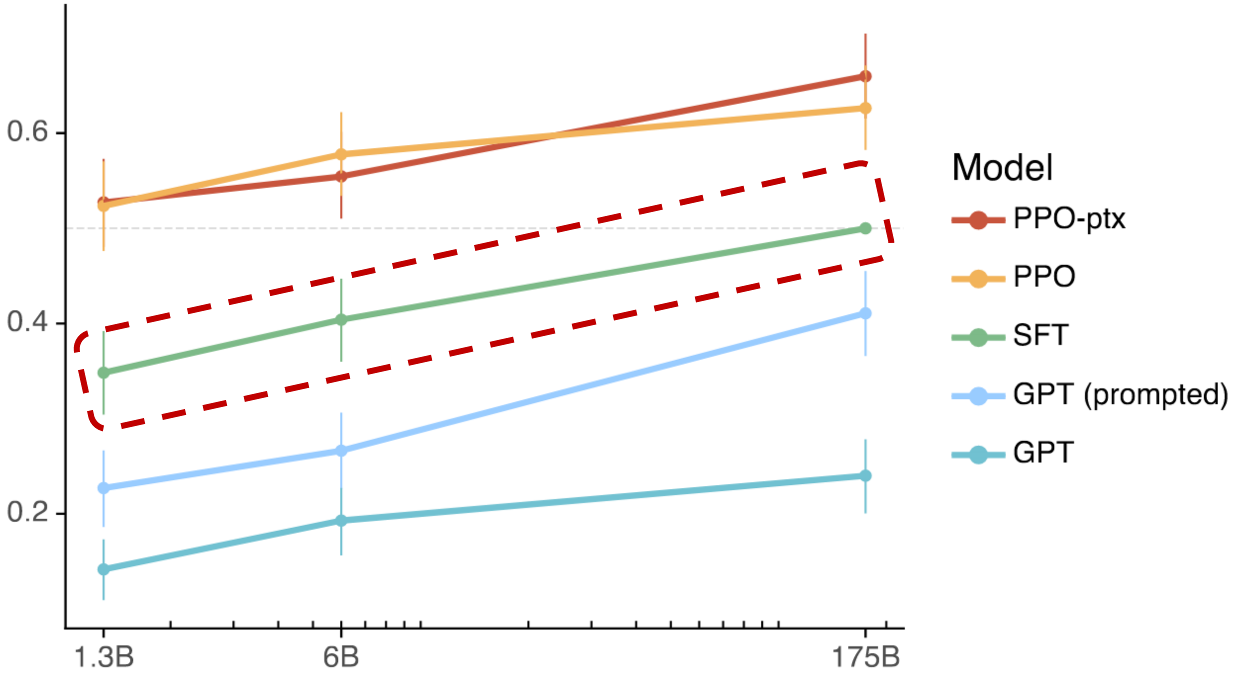
Supervised FineTuning (SFT)





- Generate prompt and response as an autoregressive task, but calculate cross entropy loss only on the generated response
- Teacher forcing: feed input into the model despite the correctness of output

Supervised FineTuning (SFT)



- ✓ Advantage: SFT can significantly increasing the performance
- x Disadvantage: requires annotations (responses) from human

- Unsupervised (self-supervised) pretraining: large scale unannotated dataset, but limited performance
- Supervised finetuning: significant improvement in performance, but dataset is quite expensive

How to further improve the performance of LLM with manageable cost?

Reinforcement Learning

Outline

- Pretraining (Memorize)
- Finetuning (Imitate)
- Reinforce (Reasoning)
- Engeneering Practices

Policy Gradient in RL

Policy gradient method

s (state) → Policy network with parameter θ (actor) → $\pi(a|s,\theta)$ or a

probability of action a or just a

Example of Atari game

image
state → Policy network (e.g., CNN or ViT)

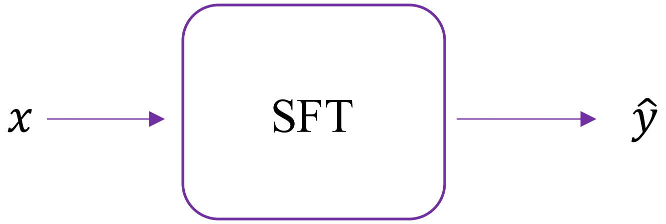
$\pi(a_{\text{left}}|s,\theta)$
e.g., 0.8

$\pi(a_{\text{right}}|s,\theta)$
e.g., 0.2

Example of LLM

$\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_T \end{bmatrix}$ prompt
state → Policy network (SFT) → response
action $\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \cdots \\ \hat{y}_K \end{bmatrix}$

Reward Model in RLHF



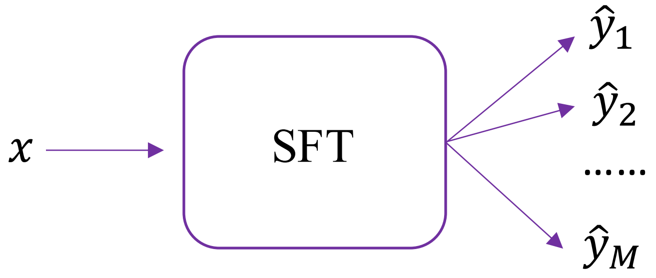
- x and $\hat{y}$ are abbreviated as prompt and generated response, respectively

Two questions

- How to measure the quality of $\hat{y}$ → Reward model
- How to improve the quality of $\hat{y}$ → Policy gradient

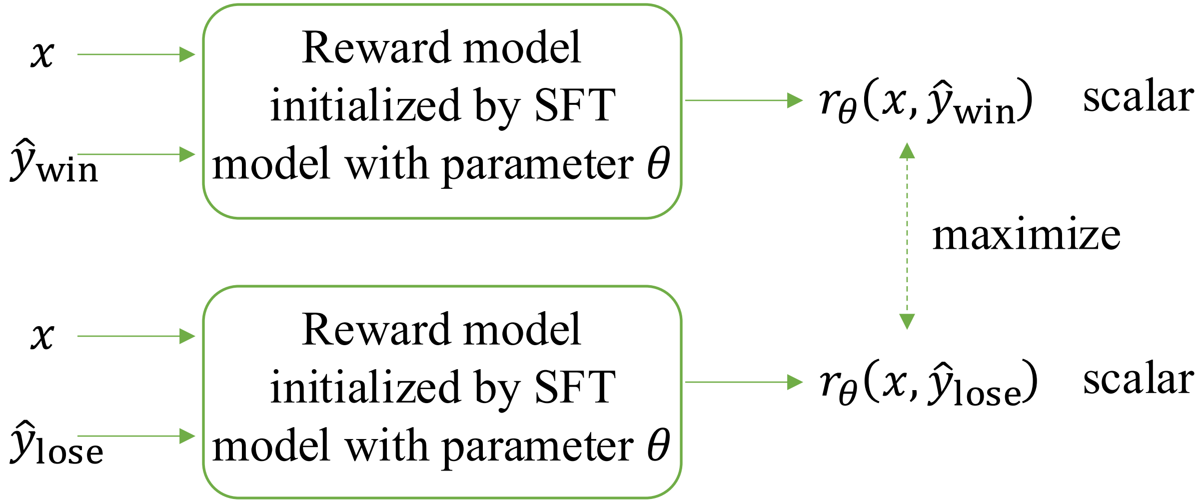
Reward Model in RLHF

- How to measure the quality of $\hat{y}$
- How to improve the quality of $\hat{y}$



- Generate M (e.g., 9) possible responses given the same prompt x
- Rank the quality of M responeses by annotators

comparison between each two responses ($\hat{y}_{\text{win}}$ and $\hat{y}_{\text{lose}}$) is used as the signal to train a reward model



$$\text{maximize} \quad r_\theta(x, \hat{y}_{\text{win}}) - r_\theta(x, \hat{y}_{\text{lose}})$$

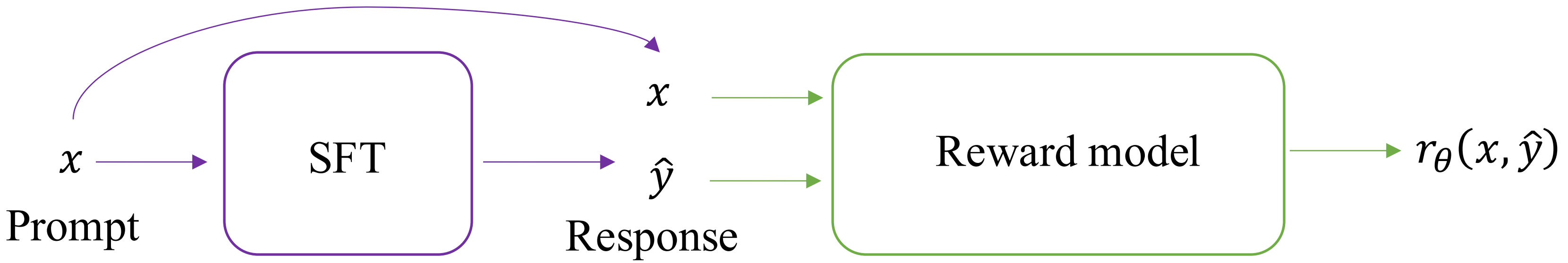
σ: Sigmoid function

$$\text{maximize} \quad \log\left(\sigma\left(r_\theta(x, \hat{y}_{\text{win}}) - r_\theta(x, \hat{y}_{\text{lose}})\right)\right)$$

$$\text{minimize} \quad -\log\left(\sigma\left(r_\theta(x, \hat{y}_{\text{win}}) - r_\theta(x, \hat{y}_{\text{lose}})\right)\right)$$

This is binary classification task

Reward Model in RLHF

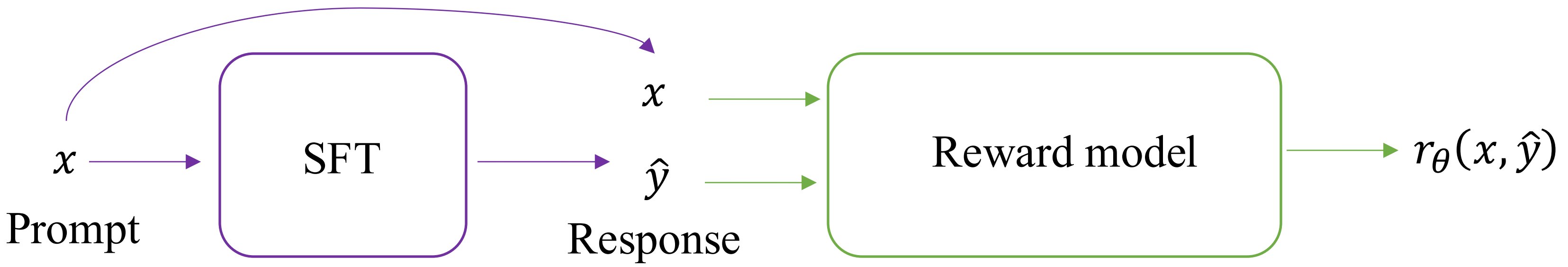


- Larger $r_\theta(x, \hat{y})$ indicates $\hat{y}$ is preferred
- Otherwise, $\hat{y}$ is unpreferred

How to further optimize the SFT model to obtain high $r_\theta(x, \hat{y})$ with powerful generative capability?

Policy gradient method

Policy Gradient in RLHF



Object 1

Maximize $r_\theta(x, \hat{y})$, where x and $\hat{y}$ are drawn from the updated policy network (π^{RL})

Maximize $\mathbb{E}_{(x,\hat{y})\sim D_{\pi^{RL}}}[r_\theta(x, \hat{y})\ ]$

Object 2

Control the magnitude of policy network updates, avoid unstable training

Maximize $\mathbb{E}_{(x,\hat{y})\sim D_{\pi^{RL}}}\left[-\beta \log\left(\frac{\pi^{RL}(\hat{y}|x)}{\pi^{SFT}(\hat{y}|x)}\right)\right]$

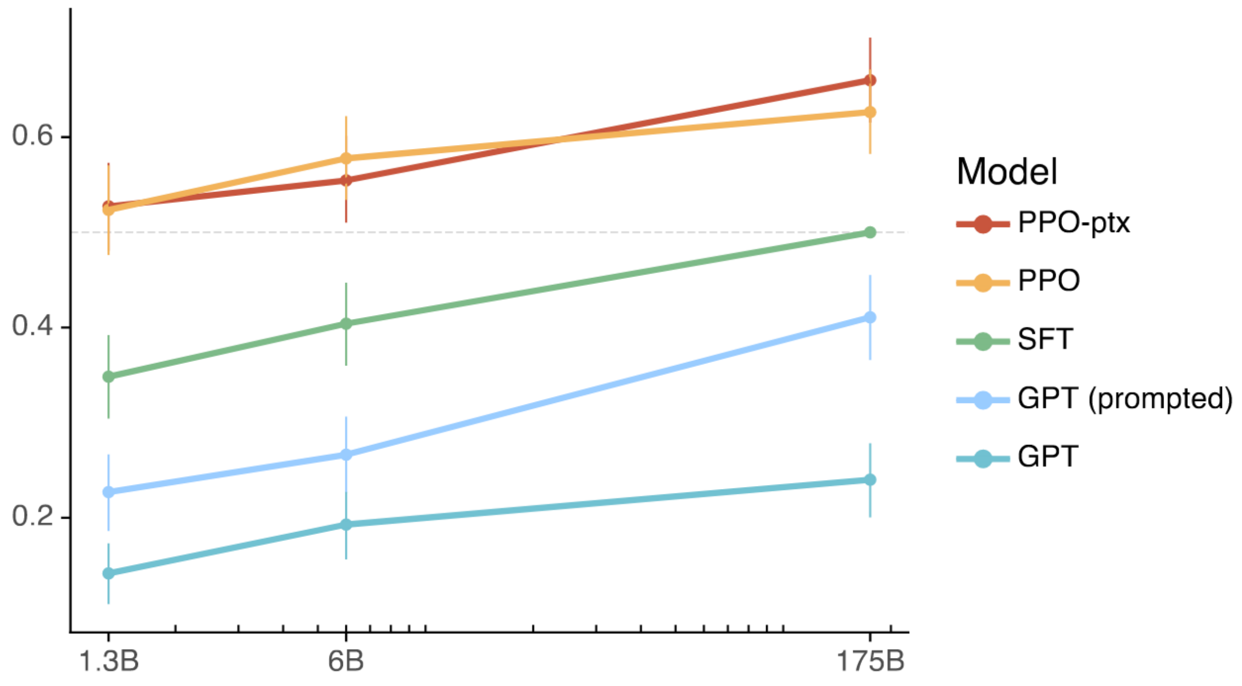
π^{RL}: updated policy network
π^{SFT}: frozen policy network (or reference model) initialized by SFT

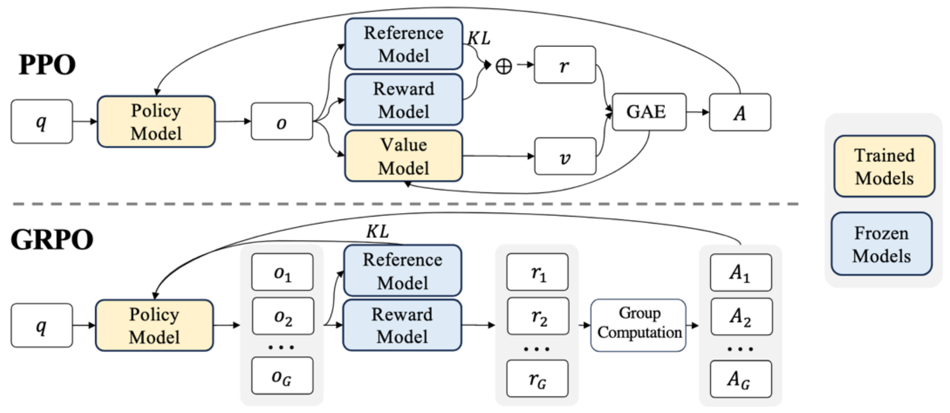
Object 3 (optional)

Updated policy network (π^{RL}) can still accomplish auto-regressive generative task, resulting PPO-ptx

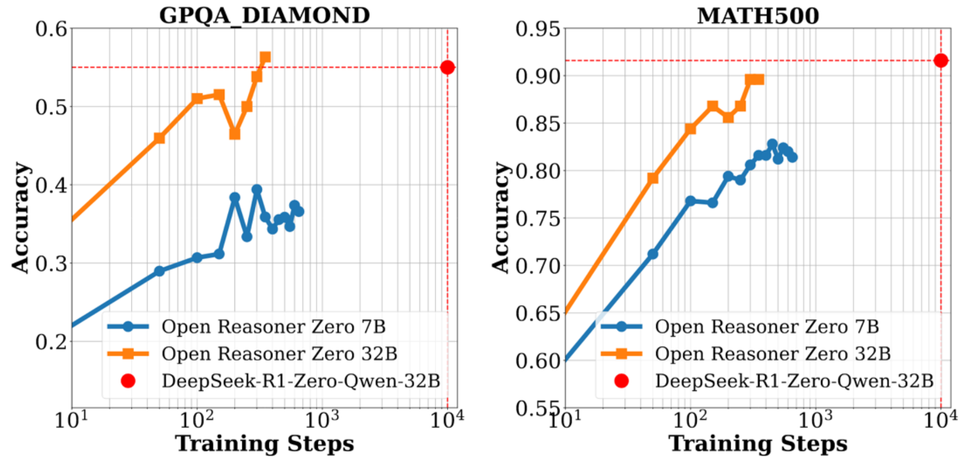
Maximize $\mathbb{E}_{x\sim D_{\mathrm{pretrain}}}[\gamma \log(\pi^{RL}(x)\ )]$

RLHF





- Variant of PPO (GRPO) proposed by DeepSeek



- Policy gradient significantly increasing the performance
- Potential of RL in LLM has not been fully explored
- Vanilla PPO is enough (Open Reasoner Zero)

Outline

- Pretraining (Memorize)
- Finetuning (Imitate)
- Reinforce (Reasoning)
- Engeneering Practices

Important Library

Transformers

Transformers provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.

Datasets

Datasets provides dataloaders for numerous public datasets and efficient preprocessing methods

TRL

TRL is a full stack library that provides a set of tools to train transformer language models with Reinforcement Learning, from the Supervised Fine-tuning (SFT), Reward Modeling (RM) to the Proximal Policy Optimization (PPO)

Unsloth

Unsloth provides a collection of commonly used LLMs and accelerates the finetuning process by 2x, reduces memory usage by 70%, all while maintaining the same level of accuracy

Prepare Finetuning Dataset

“Question”: “一个1岁的孩子在夏季头皮出现多处小结节，长期不愈合，xxx"

“CoT”: “用中医的角度来看，出现小结节、再加上长期不愈合，xxx"

Chain of Thought (CoT), Optional

“Response”: “这是一种因湿热导致的疾病，xxx"

```
from datasets import load_dataset

dataset = load_dataset("json",  data_files=”./data.json)
```

Prepare Model

- load model and tokenizer

```
from unsloth import FastLanguageModel

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "./DeepSeek-R1-Distill-Qwen-32B",
    local_files_only=True,
    max_seq_length = 4096,
    load_in_4bit = True)
```

- configure pretraining model

```
FastLanguageModel.for_training(model)

model = FastLanguageModel.get_peft_model(
  model,
  target_modules=["q_proj","k_proj","v_proj","o_proj"]
  r=32, # LoRA related parameter
  lora_alpha=16,  # LoRA related parameter
  use_gradient_checkpointing="unsloth")
```

Prepare Trainer

- load trainer

```
from trl import SFTTrainer
from transformers import TrainingArguments

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset,
    dataset_text_field = "text",  # 数据集字段名称
    max_seq_length = 4096,
    dataset_num_proc = 2,  # 处理数据集进程数目
    args = TrainingArguments(
        per_device_train_batch_size = 2,  # 每个GPU训练batch
        learning_rate = 2e-4,  # 学习率
        optim = "adamw_8bit",  # 使用8位AdamW优化器节省显存
        weight_decay = 0.01,   # 正则化强度
        output_dir = "outputs",  # 模型输出目录
        run_name = "medical-o1-sft-experiment",  # 实验名称
    ),
)
```

```
trainer.train()
```

Take a break and let the LLM do the magic

Thanks!