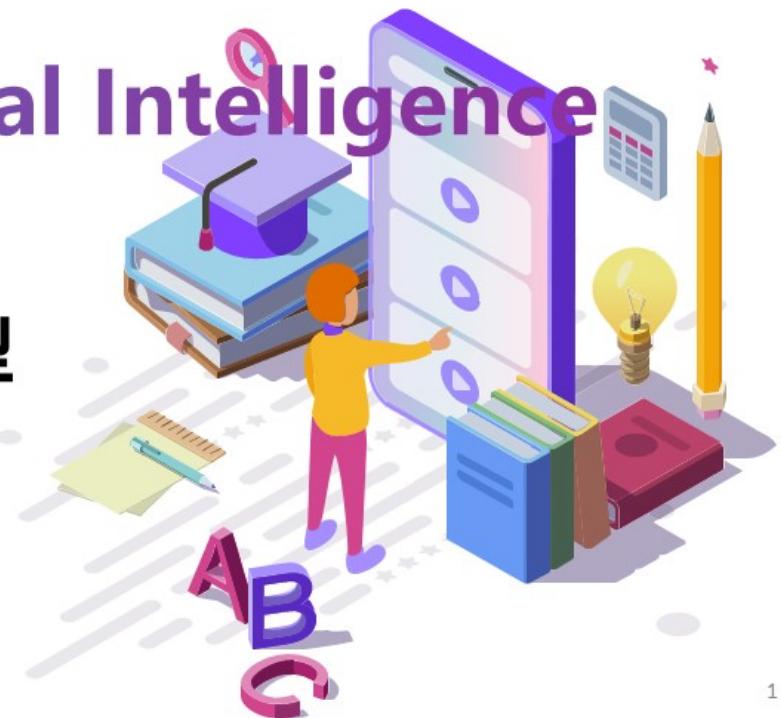


人工智能导论

Introduction to Artificial Intelligence

第六章 预训练模型与大语言模型



单选题 1分

预训练的核心目标是什么？

- A 在小规模标注数据上直接训练模型
- B 让模型在大规模无标注数据上学习通用规律
- C 只优化分类任务的交叉熵
- D 仅训练模型的最后一层

什么是预训练？

预训练（pre-training）是指在人工智能和机器学习中，先使用大规模、通用数据集训练模型，使其学习**通用的知识和特征**，然后再将其**迁移适配**到特定任务。

1. **发现规律，总结共性—预训练阶段**（Pre-training Phase）：模型在大规模的通用数据集上进行训练，以学习通用的知识和特征。这个阶段不针对任何特定的下游任务。
2. **迁移—微调阶段**（Fine-tuning Phase）：预训练的模型在特定任务的数据集上进行进一步训练，以适应该任务的特定需求。这个阶段的训练数据通常比预训练阶段的数据少得多。

单选题 1分

GPT 的预训练方式属于：

- A 自回归语言模型（左到右）
- B 双向语言模型（可见全上下文）
- C 编码器-解码器式填空模型
- D 对比学习模型

填空题 3分

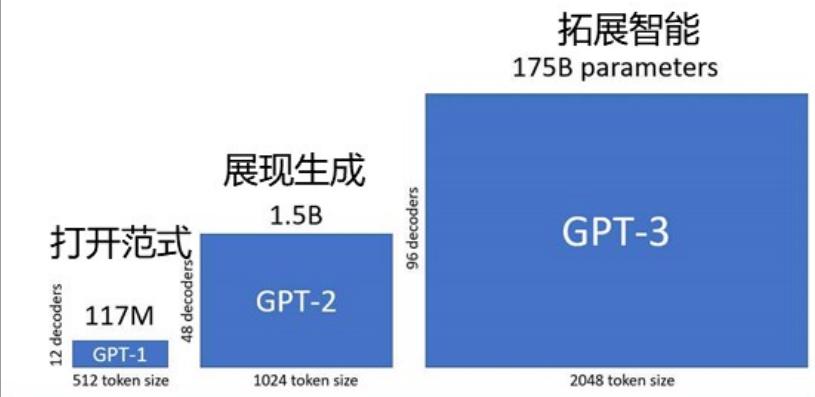
GPT 的网络结构是 [填空1]。GPT-2 和 GPT-3 的一个最重要突破是提出了 [填空2] 与 [填空3] 能力，使得模型无需微调即可根据提示完成任务。

预训练语言模型：GPTs

➤ GPTs：从学习语言到展现智能

规模扩展方向：**↑层数 ↑维度 ↑数据 ↑参数量**

当模型规模、数据量、计算量持续增长时，语言模型的性能会以可预测的幂次规律稳定提升。



随着**模型参数的增多**，研究者发现了自监督预训练的 GPT逐渐具有了处理需要有监督微调的下流任务的能力。

研究者尝试了以下方法来挖掘GPT的这种**涌现能力**：

- 在下游任务中嵌入**固定提示**的零样本微调；
- 嵌入**提示+演示**(称作上下文学习)的少样本微调。

这些能力在小规模模型中不存在，而在大規模模型中存在

单选题 1分

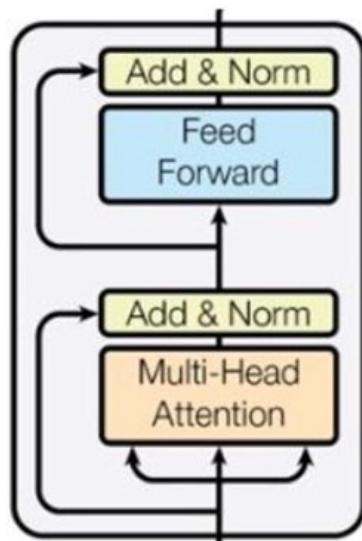
BERT 的预训练目标是 _____ 和 _____，它是基于 Transformer _____ 架构实现的。

- A MLM、NSP; Encoder-only
- B 自回归、NSP; Decoder-only
- C MLM、Span Masking; Encoder-Decoder
- D 对比学习、NSP; Decoder-only

预训练语言模型：BERT

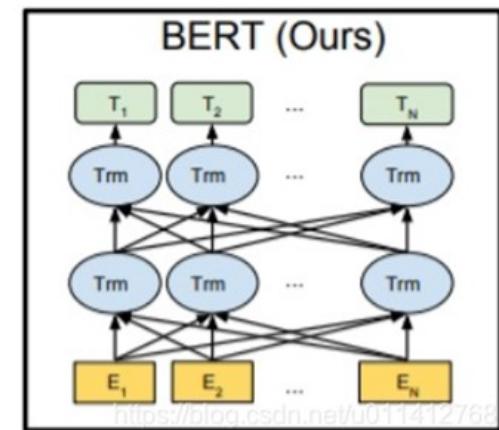
➤ BERT

BertLayer (Encoder Block)

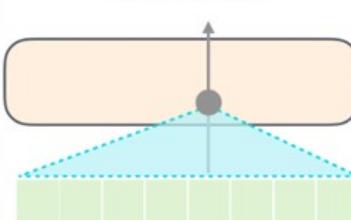


➤ 基于 **Transformer Encoder** (全双向自注意力) 结构。

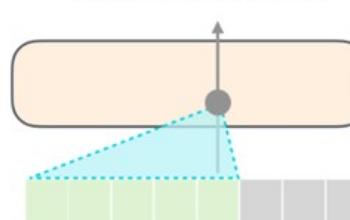
➤ BERT 是“双向”——每个词都能同时看到前后文



Self-Attention



Masked Self-Attention



单选题 1分

以下哪种位置编码能够根据(token 的)相对距离建模?

- A 原始 Transformer 正余弦位置编码
- B RoPE (旋转位置编码)
- C 固定位置 ID
- D 拼接绝对位置向量

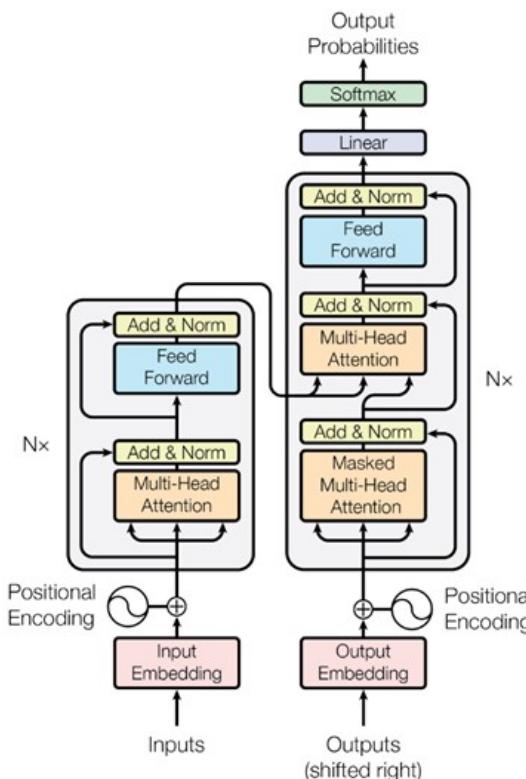
单选题 1分

T5 的结构属于哪一种 Transformer 架构?

- A Decoder-only
- B Encoder-only
- C Encoder–Decoder
- D 双向 Decoder

预训练语言模型：T5

➤ T5: Text-to-Text Transfer Transformer



- 基于 **Transformer Encoder–Decoder**
- Encoder: 读取输入语义
- Decoder: 按词生成输出文本

任务类型	输入	输出
翻译	“translate English to French: Hello”	“Bonjour”
摘要	“summarize: 今天我们学习了BERT模型...”	“BERT是一种理解式语言模型。”
分类	“classify sentiment: 这部电影太棒了！”	“positive”

填空题 2分

ViT 将图像处理成 token 的方法是：将图像固定切分为 [填空1]，
然后对每个 patch 通过 [填空2] 投影为序列 token。

预训练视觉模型

➤ Vision Transformer (ViT): “图像的BERT”

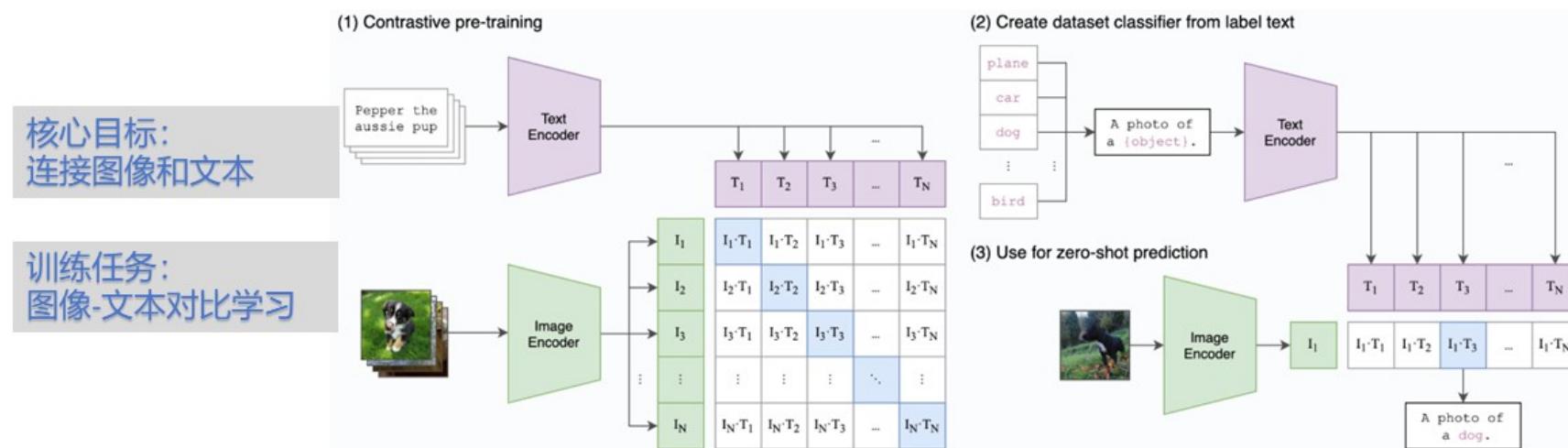
使用大型标注数据集（如ImageNet），输入图像，模型输出一个类别标签（如“猫”、“狗”）。通过标准的交叉熵损失进行训练，目标是让预测的标签与真实标签一致。

1. 将图片转换成patches序列
2. 将patches铺平
3. 添加position embedding
4. 添加class token
5. 输入Transformer Encoder
6. 分类

预训练多模态模型

➤ CLIP: 图文对比学习的里程碑——从单模态到多模态

CLIP (Contrastive Language-Image Pre-training) 是 OpenAI 于 2021 年提出的多模态模型，通过对比学习的方式建立图像和文本之间的关联。

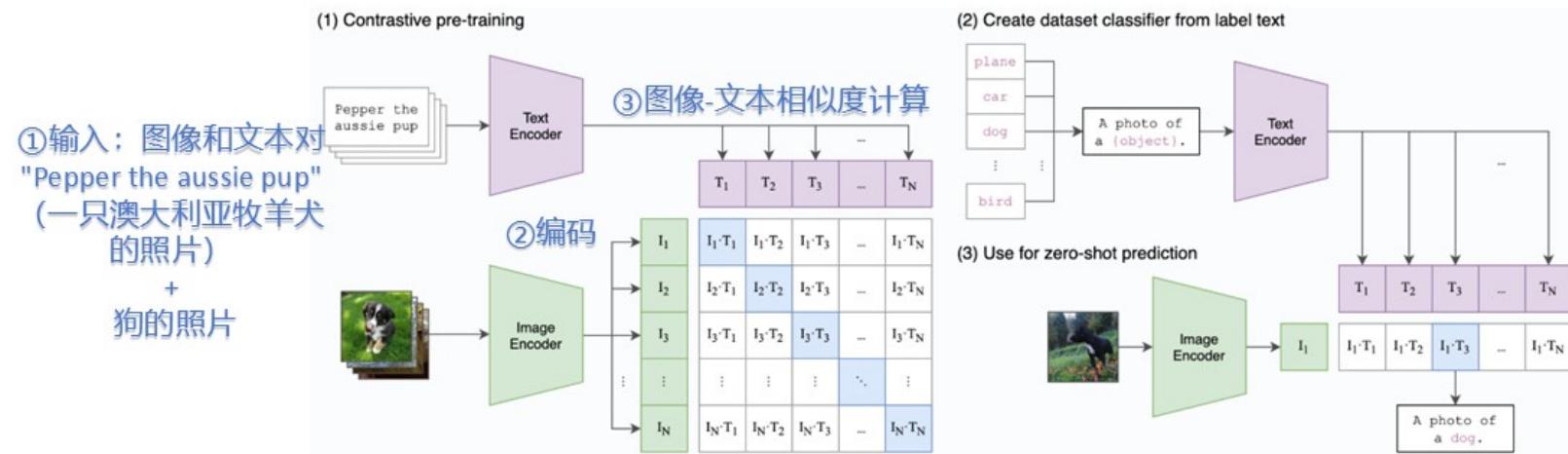


预训练多模态模型

➤ CLIP: 图文对比学习的里程碑——从单模态到多模态

CLIP的两个核心组件:

- ✓ 图像编码器 (Image Encoder) : 将图像转换为特征向量 (Vision Transformer 或 ResNet) 。
- ✓ 文本编码器 (Text Encoder) : 将文本描述转换为特征向量 (Transformer) 。



预训练多模态模型

➤ CLIP：图文对比学习的里程碑——从单模态到多模态

CLIP的对比学习：

核心目标：

- 让匹配的图文对在嵌入空间中相似度高，不匹配的对相似度低。实现语义对齐 (semantic alignment)
- 双向对比学习：
 - Image→Text
 - Text→Timage
- 损失函数 (InfoNCE 对比损失) InfoNCE = Cross-Entropy with softmax over similarity scores

$$L = \frac{1}{2N} \sum_{i=1}^N \left[-\log \frac{\exp(\text{sim}(v_i, t_i)/\tau)}{\sum_{j=1}^N \exp(\text{sim}(v_i, t_j)/\tau)} - \log \frac{\exp(\text{sim}(t_i, v_i)/\tau)}{\sum_{j=1}^N \exp(\text{sim}(t_i, v_j)/\tau)} \right]$$

Image→Text Text→Timage

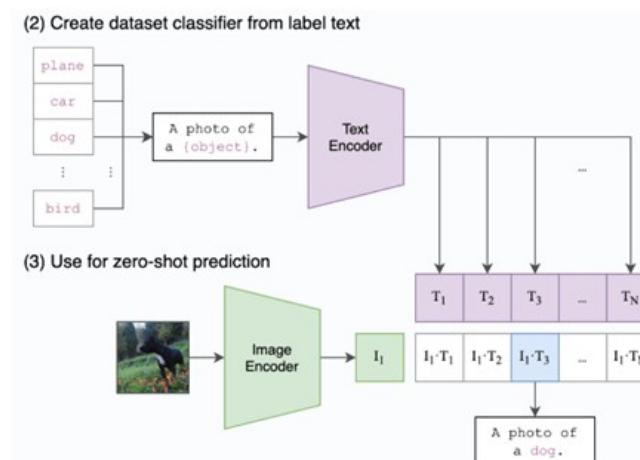
其中 $\text{sim}(v, t)$ 为 cosine 相似度， τ 为温度参数。

预训练多模态模型

➤ CLIP：图文对比学习的里程碑——从单模态到多模态

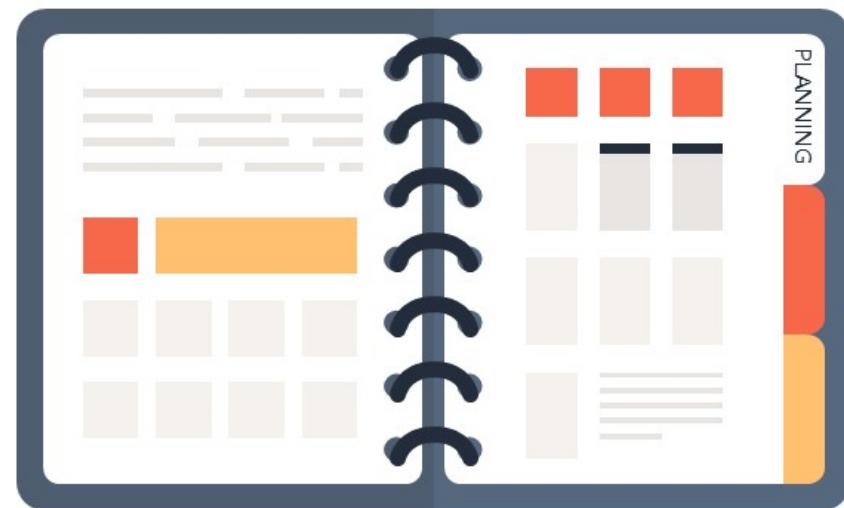
CLIP用于图像识别：

- 待识别的图片经过Image Encoder映射为特征向量；
- 同时将数据集中所有的类别文本转为“*A photo of a {object}.*”形式；
- 将这些新的文本经过Text Encoder映射到与图片一样的特征空间；
- 最后，找到与图像向量最近的文本向量。



Lecture Plan

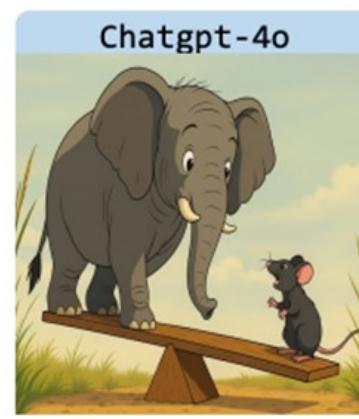
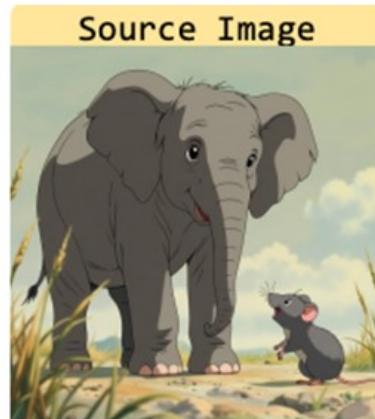
- 预训练模型的发展历程
- 预训练模型
- **微调技术**
- RLHF



微调技术

➤ 为什么需要微调?

尽管预训练模型在大规模数据集上学到了丰富的通用特征和先验知识，但这些特征和知识可能并不完全适用于特定的目标任务。



What would happen if the elephant and mouse stood on the lawn at the same time on opposite ends of a seesaw?

微调技术

➤ 什么是微调？

模型在与目标任务或领域相关的更具体、更小的数据集上进一步训练。让模型学习任务的**具体特征**，并适应任务的特殊要求。



预训练+微调 · 技术范式

➤ 预训练+微调范式

- 预训练+微调示例



监督微调 (SFT)

- 在新任务的**小规模标注数据集**上，使用**有监督学习**的方法对预训练模型进行**微调**，以使其适应新任务。

加载预训练模型 → 准备新任务的数据集 → 调整模型输出层 →
在新任务数据集上训练模型

- 适用于那些有明确标注数据集的任务，例如文本分类、命名实体识别等。

模型参数微调

➤ 关注模型参数层面的调整

✓ 全面微调

- 在新任务上调整模型的**全部**参数，以使其完全适应新任务。
- 加载预训练模型→在新任务数据集上训练模型，调整所有参数。
- 当新任务与预训练任务差异较大，或者想要充分利用新任务数据集时可以选择全面微调。

✓ 参数高效微调

- 仅调整模型的**部分**参数；
- 加载预训练模型→在模型中添加可训练的组件或选择部分参数→在新任务数据集上训练这些组件或参数；
- 当计算资源有限，或者想要快速适应新任务而不影响模型在其他任务上的性能时。

全参数微调

➤ 全参数微调

- ✓ 全参数微调 (Full Fine-tuning) 是指在预训练模型的基础上，使用下游任务的数据集，更新模型中的**所有参数**，以使模型适应特定任务。

微调模型与原
始预训练模型
的大小相同

为每个下游任务独立存储和部署微调模型
变得非常昂贵

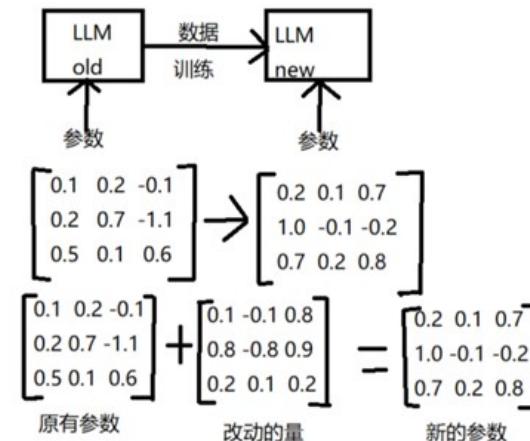
过拟合 灾难性遗忘

参数高效微调

➤ 参数高效微调 (PEFT)

✓ 参数高效微调 (Parameter-Efficient Fine-Tuning, PEFT) 是一系列旨在以较少的计算资源和数据量，实现与全参数微调相近性能的技术。这类方法通常**冻结**预训练模型的**大部分参数**，只**训练少量额外的参数**。

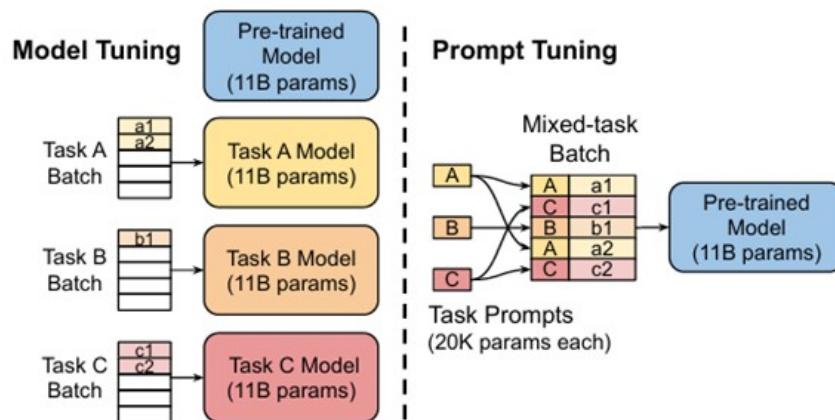
- 提示微调技术
- Adapter
- LoRA系列



参数高效微调

➤ 提示微调技术 - Prompt-based Fine-tuning

- ✓ 提示微调 (Prompt Tuning) 是一种通过设计合适的提示 (Prompt) 来引导预训练模型完成下游任务的技术。
- ✓ 利用了预训练语言模型在零样本或少样本学习中的强大能力，通过修改输入提示来激活模型内部的相关知识和能力。



26

参数高效微调

➤ 提示微调技术-Prompt Engineering (提示工程)

- ✓ 人工设计文字提示，不训练参数，用“说话方式”引导模型。
- ✓ 常见几种提示形式：

➤ 指令式提示 (Instruction Prompting)

“You are a sentiment classifier. Given a sentence, output Positive or Negative.”

➤ 少样本提示 (Few-shot Prompting)

给几个“输入→输出”的例子，让模型“学模式”

➤ 思维链提示 (Chain-of-Thought)

加一句“Let's think step by step.” 让模型展示推理过程

➤ 角色设定 (Role Prompting)

“You are a math teacher...”、“You are a Python expert...”

参数高效微调

➤ 提示微调技术-Prompt Tuning (软提示)

- ✓ 把提示变成“可训练的向量”，只训练这些向量，模型参数完全冻结。
- ✓ 在输入序列前面加入若干可训练的 embedding。

原输入：

[CLS] The movie is great.

添加软提示：

[p1][p2][p3] [CLS] The movie is great.

✓ 优点：

- 模型参数完全冻结，**安全、不破坏原模型能力**
- 训练参数极少（几十~几百个向量）
- 非常适合**超大模型（几十B参数）**

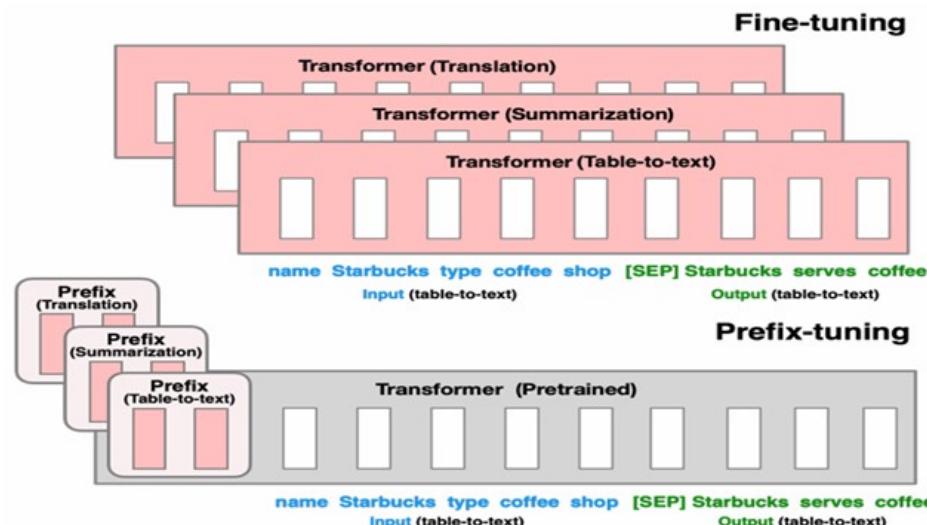
✓ 缺点：

- 表达能力有限（只在输入层做修改）
- 对小模型效果一般，大模型更受益
- 对于复杂结构化任务，可能不稳定

参数高效微调

➤ 前缀微调技术-Prefix Tuning

- ✓ Prefix Tuning 通过引入可训练的“**前缀向量**”，使其作为额外的上下文信息，**参与注意力机制的计算**，从而调整模型的输出。



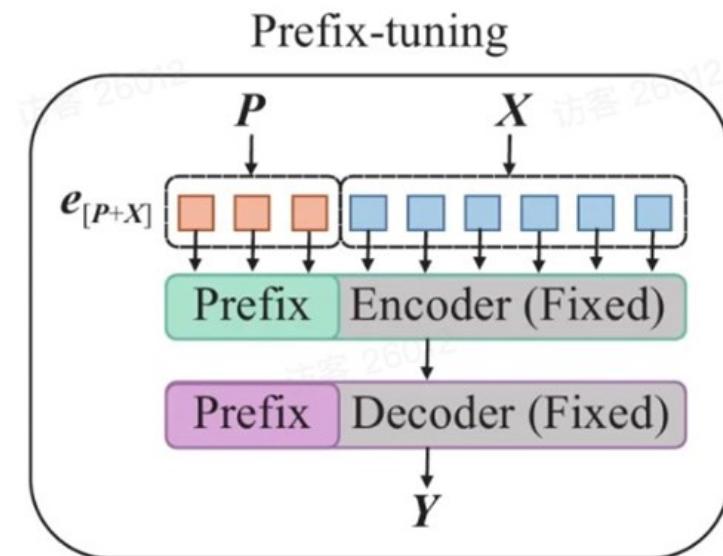
29

参数高效微调

➤ 前缀微调技术-Prefix Tuning

- ✓ 在Transformer模型的输入层或各层输入前添加可学习的前缀嵌入，通过训练这些前缀嵌入来优化模型在特定任务上的表现。

- 在Transformer模型的输入层之前，初始化一个固定长度的前缀嵌入矩阵；
- 训练过程中，模型会根据输入序列和标签数据进行学习。通过BP算法，模型会更新前缀嵌入的参数。



30

参数高效微调

➤ 前缀微调技术-Prefix Tuning

以 Transformer 中一层自注意力为例，原本：

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

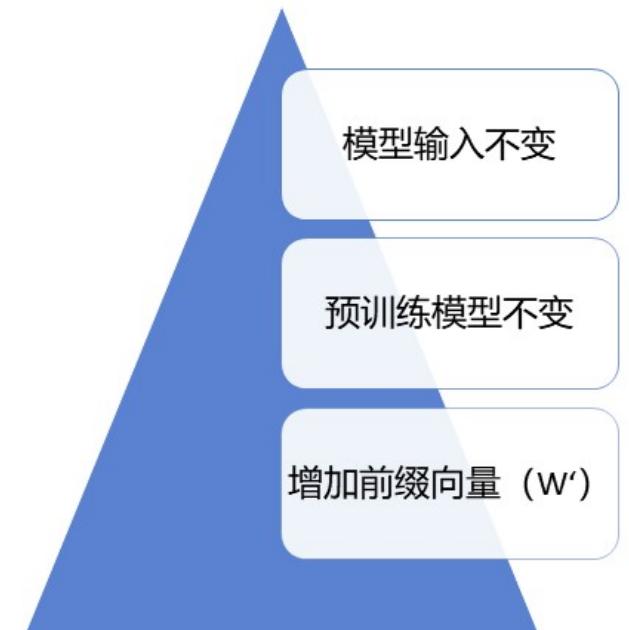
Prefix Tuning 会为每层引入若干前缀向量 P^K, P^V ：

$$\tilde{K} = [P^K, K], \quad \tilde{V} = [P^V, V]$$

注意力计算：

$$\text{Attn}(Q, \tilde{K}, \tilde{V}) = \text{softmax} \left(\frac{Q\tilde{K}^\top}{\sqrt{d}} \right) \tilde{V}$$

- 原输入 X 不变
- 原权重 W_Q, W_K, W_V 不变
- 只训练这些 前缀向量 P^K, P^V



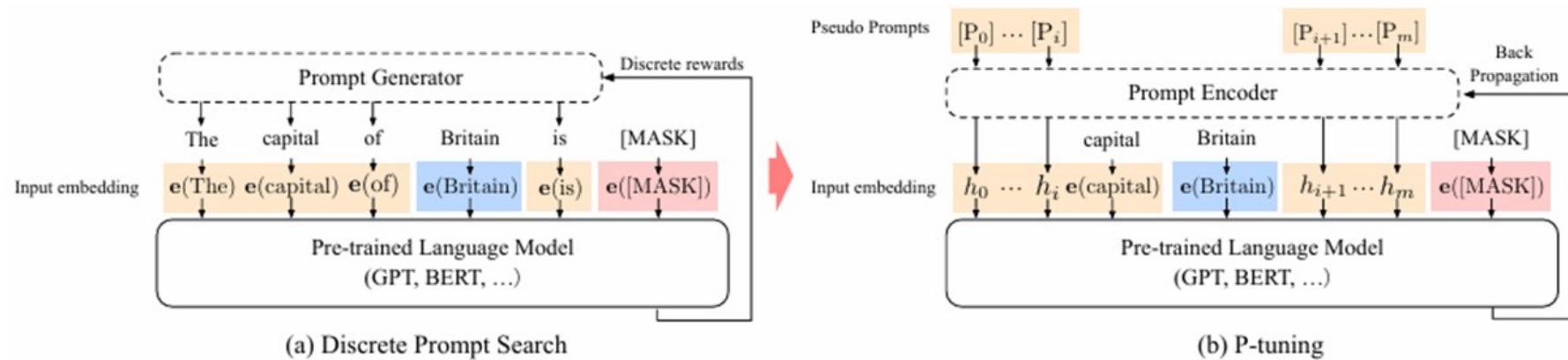
Prompt Tuning vs. Prefix Tuning

- *Prompt Tuning*: 只在输入 embedding 前面加一段软提示
- *Prefix Tuning*: 在每一层 Attention 的 KV 上加“提示前缀”，影响更深层
 - ✓ Prefix Tuning 的表达能力更强
 - ✓ 参数也稍多一些（因为每层都有前缀）

参数高效微调

➤ 提示微调技术 - P-Tuning v1

- ✓ 不直接把 soft prompt 当作一组独立可训练向量，而是用一个小网络（比如 Bi-LSTM、MLP）来生成提示，使提示更“结构化”。
- ✓ 连续提示 + 编码器：连续提示之间有结构，更强表达能力；在 BERT 类模型上的分类任务中效果较好。

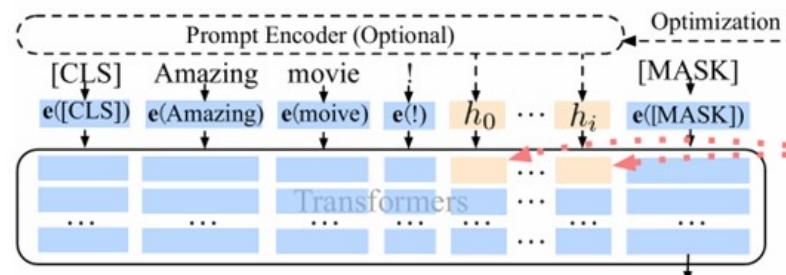


33

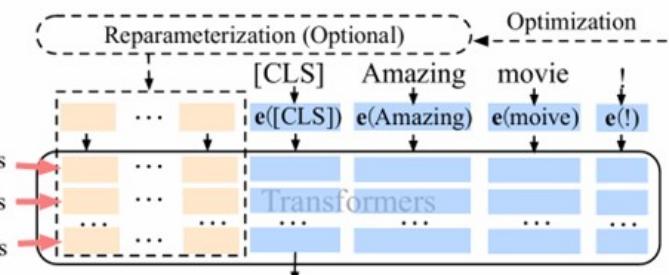
参数高效微调

➤ 提示微调技术 - P-Tuning v2

- ✓ 把可学习提示扩展为 跨层、跨位置的 “深层提示”
 - 1、更多可学习的参数，同时也足够参数高效。
 - 2、加入到更深层结构中的Prompt能给模型预测带来更直接的影响。



(a) Lester et al. & P-tuning (Frozen, 10-billion-scale, simple tasks)



(b) P-tuning v2 (Frozen, most scales, most tasks)

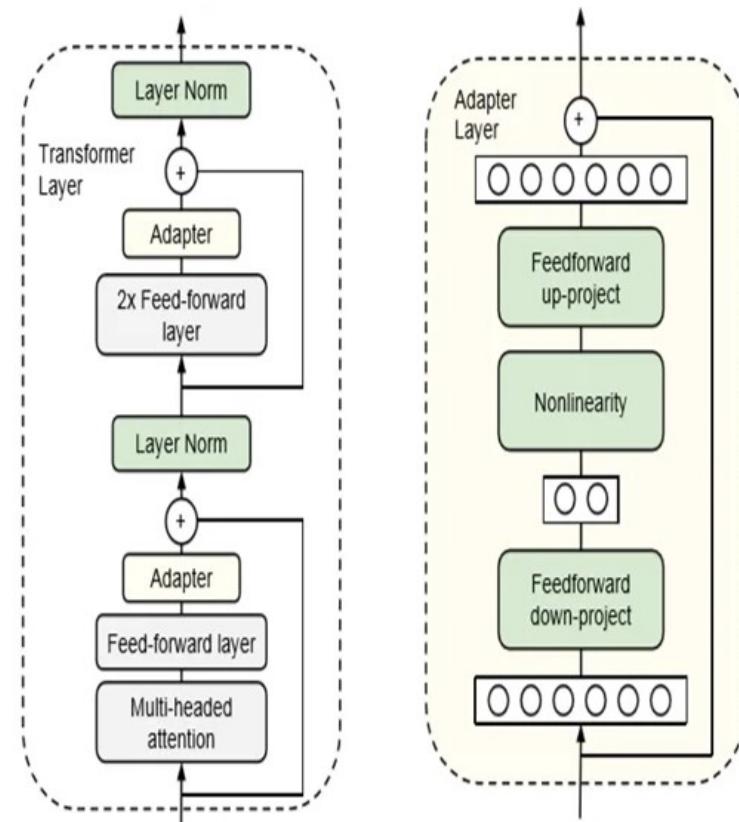
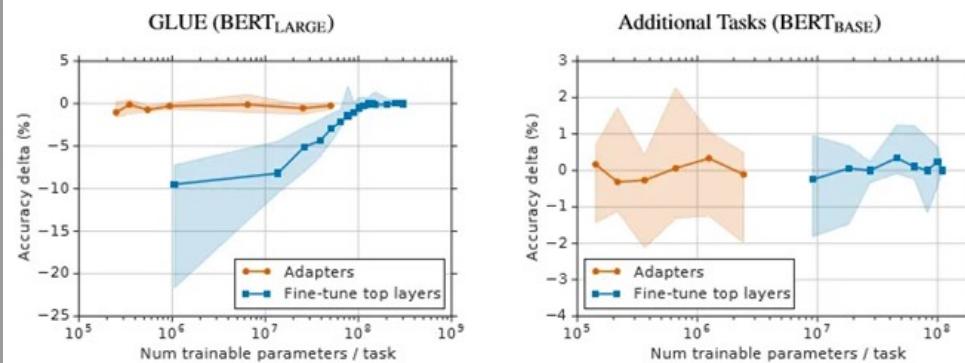
参数高效微调

方法	是否训练参数	作用位置	表达能力	实现复杂度	典型用途
Prompt Engineering	✗ 不训练	纯文本输入	依赖人写	最简单	快速试验、零样本
Prompt Tuning	✓ 训练 soft prompts	输入 Embedding 前	较弱 (靠输入)	简单	大模型、低资源微调
Prefix Tuning	✓ 训练前缀向量	各层 Self-Attn 的 KV	较强 (deep prompt)	中等	生成任务、长文本
P-tuning v1	✓ 提示 + 编码器	输入侧 + prompt encoder	比普通 soft prompt 强	中等	BERT 类分类任务
P-tuning v2	✓ 深层连续提示	多层、多位置 deep prompt	强 (接近 Prefix/LoRA)	较高	通用任务、复杂场景

参数高效微调

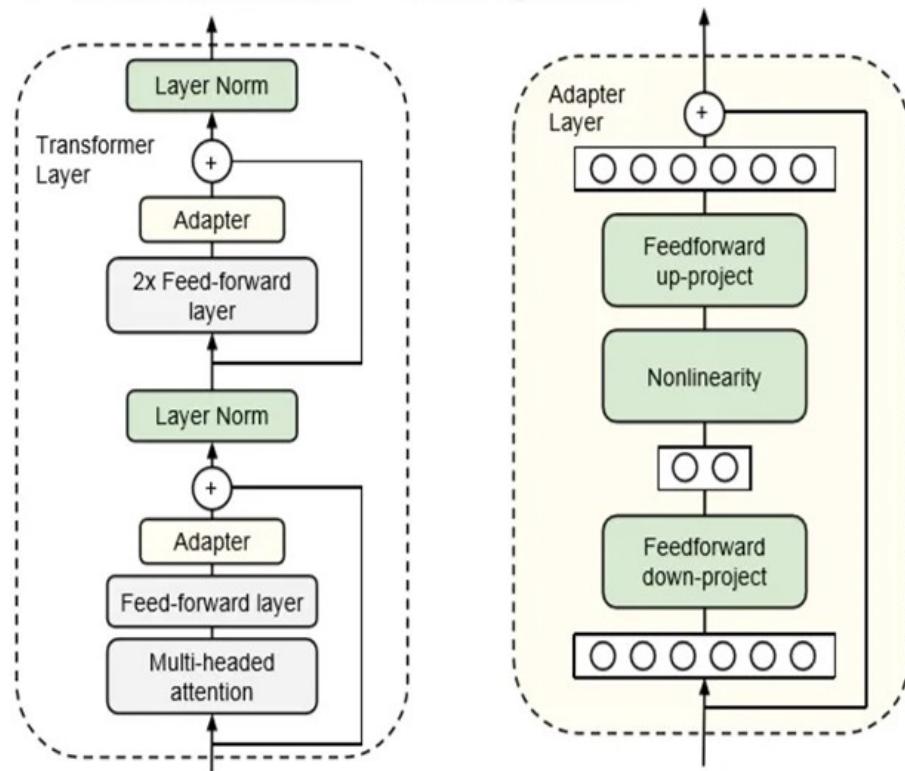
➤ 适配器微调 - Adapter

- ✓ 在每层 Transformer 里插入一个小型可训练模块，让大模型“保持原样”，只训练这些小模块即可完成下游任务。
- ✓ 在训练过程中，冻结原始transformers layer中的权重，只训练 adapter layer中的权重。



参数高效微调

➤ 适配器微调 - Adapter



原层输出 h
↓
Adapter 下升维/降维模块
(小瓶颈)
↓
再加到原输出 h 上 (残差)

$$h' = h + \text{Adapter}(h)$$

强制模型的任务特定改变
“压缩到一个小维度 r 中学
习”，达到节省参数的效果。

$$\text{Adapter}(h) = W_{up} \sigma(W_{down} h)$$

37

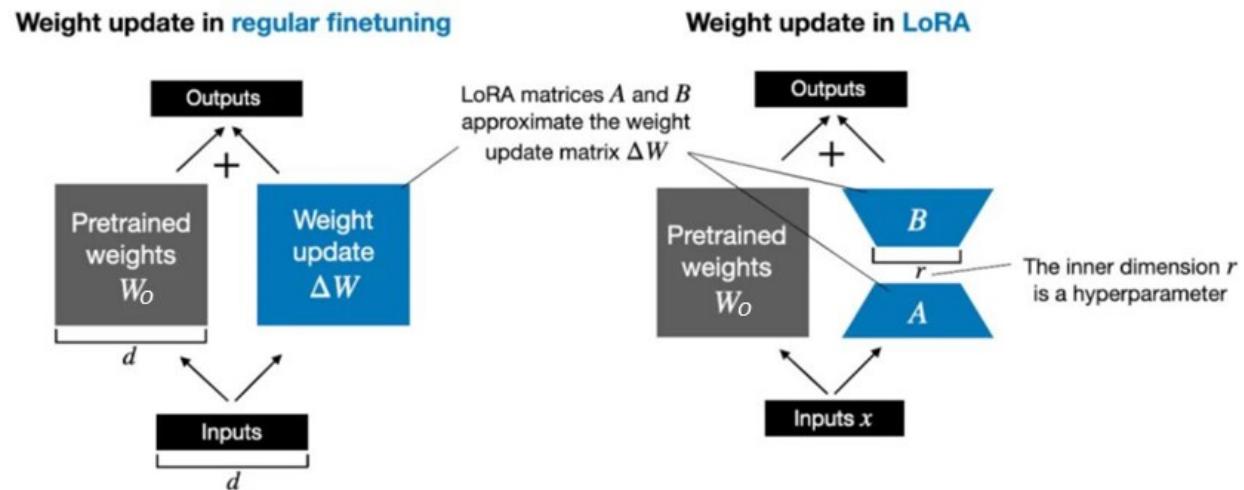
参数高效微调

➤ Low-Rank Adaptation - LoRA

- ✓ 低秩适配（LoRA）的LoRA的核心是**冻结预训练模型原始参数**，通过**低秩分解**引入可训练参数，参数量少但性能接近全参数微调。

$$W = W_o + \Delta W$$

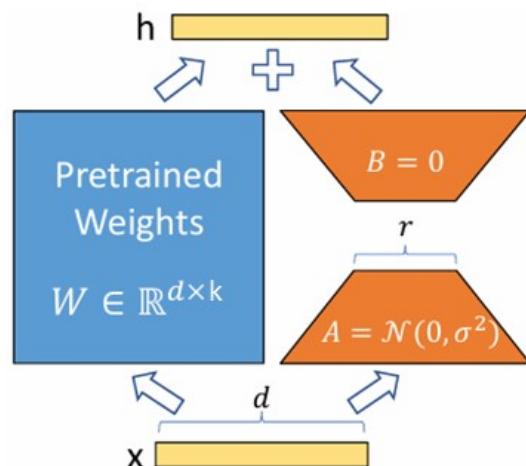
ΔW : 需要更新的参数
 W_o : 预训练模型初始化的参数



参数高效微调

➤ LoRA

✓ 具体来说，对于一个预训练好的权重矩阵 $W_0 \in \mathbb{R}^{d \times k}$



- LoRA 引入两个**低秩矩阵**

$$A \in \mathbb{R}^{r \times k} \text{ 和 } B \in \mathbb{R}^{d \times r}$$

- 其中 $r \ll \min(d, k)$ 是**秩**。
- 在微调过程中，只优化 A 和 B ，而 W_0 保持不变。

更新后的权重矩阵为：

$$W = W_0 + BA$$

- 初始化： $B=0$ ， A 用随机小值

参数高效微调

➤ Adaptive LoRA - AdaLoRA

- ✓ **AdaLoRA**, 是对LoRA的一种改进, 它根据重要性评分**动态分配**参数预算给权重矩阵。
- ✓ AdaLoRA 会:
 - 给重要层分配更高的 rank (更强表达能力)
 - 给不重要层分配更低的 rank (节省参数)
 - 并且动态调整 (训练过程中一直变化)

参数高效微调

➤ Adaptive LoRA - AdaLoRA

- ✓ **AdaLoRA**, 是对LoRA的一种改进, 它根据重要性评分动态分配参数预算给权重矩阵。
- ✓ AdaLoRA 会:
 - 给重要层分配更高的 rank (更强表达能力)
 - 给不重要层分配更低的 rank (节省参数)
 - 并且动态调整 (训练过程中一直变化)

难度评分 (importance score)

Algorithm 1 AdaLoRA

```
1: Input: Dataset  $\mathcal{D}$ ; total iterations  $T$ ; budget schedule  $\{b^{(t)}\}_{t=0}^T$ ; hyperparameters  $\eta, \gamma, \beta_1, \beta_2$ .  
2: for  $t = 1, \dots, T$  do  
3:   Sample a mini-batch from  $\mathcal{D}$  and compute the gradient  $\nabla \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$ ;  
4:   Compute the sensitivity  $I^{(t)}$  in (8) for every parameter in  $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$ ;  
5:   Update  $\bar{I}^{(t)}$  as (9) and  $\bar{U}^{(t)}$  as (10) for every parameter in  $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$ ;  
6:   Compute  $S_{k,i}^{(t)}$  by (7), for  $k = 1, \dots, n$  and  $i = 1, \dots, r$ ;  
7:   Update  $P_k^{(t+1)} = P_k^{(t)} - \eta \nabla_{P_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$  and  $Q_k^{(t+1)} = Q_k^{(t)} - \eta \nabla_{Q_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$ ;  
8:   Update  $\Lambda_k^{(t+1)} = \mathcal{T}(\Lambda_k^{(t)} - \eta \nabla_{\Lambda_k} \mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q}), S_k^{(t)})$  given the budget  $b^{(t)}$ .  
9: end for  
10: Output: The fine-tuned parameters  $\{\mathcal{P}^{(T)}, \mathcal{E}^{(T)}, \mathcal{Q}^{(T)}\}$ .
```

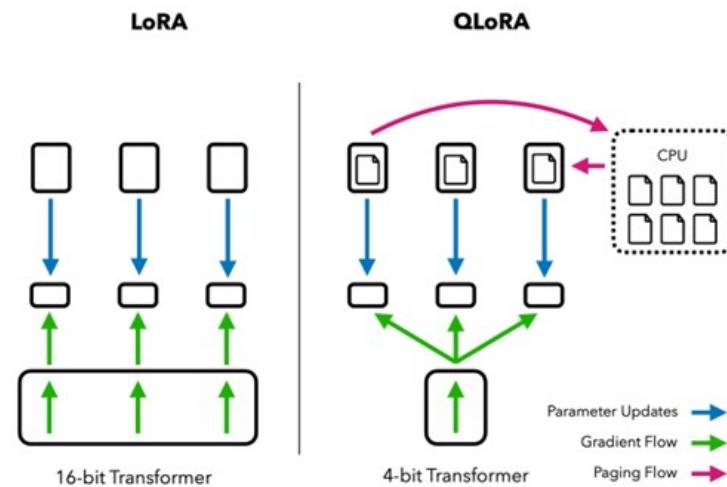
参数高效微调

➤ Quantized LoRA - QLoRA

✓ QLoRA 将 4-bit 量化与 LoRA 相结合，以进一步降低显存占用。

✓ **量化 (4bit) + 冻结参数 + LoRA 低秩补丁**

- 显存占用极低（真正能让超大模型在普通显卡训练）
- 性能几乎不降
- 和 LoRA 完全兼容
- 是目前业界训练开源大模型的主流方式（LLaMA 2/3 的所有指令模型都是用 QLoRA 微调）



实践：LoRA微调实例

➤ 使用LoRA微调Seq2Seq Transformer

✓ 需要加载的相关库：

1. **transformers**

一个流行的Python库，提供了大量预训练模型（如BERT、GPT-2、T5等），使开发者可以轻松地执行NLP任务，如文本分类、翻译、问答等。<https://huggingface.co/docs/transformers/v4.42.0/zh/index>

2. **peft**

参数高效微调库（LoRA, Prefix Tuning, Prompt Tuning, P-Tuning, etc.）
<https://github.com/huggingface/peft>

Hugging Face是一个开源的机器学习平台和社区，它提供了一个名为“模型中心”(Model Hub)的平台，允许用户分享和下载超过320,000个模型和50,000个数据集。其核心产品包括用于自然语言处理(NLP)的Transformers库，以及用于管理数据集和分词的Datasets和Tokenizers库。Hugging Face旨在通过提供易于使用的工具来简化AI的开发、共享和协作，并且服务已扩展到计算机视觉等领域



Hugging Face

实践：LoRA微调实例

➤ 引入必要的方法

```
import torch  
  
from transformers import AutoModelForSeq2SeqLM #模型相关  
  
from peft import get_peft_model, LoraConfig, TaskType #peft相关  
  
model_name_or_path = "bigscience/mt0-large"
```

➤ 创建PEFT方法对应的配置

```
peft_config = LoraConfig(  
    task_type=TaskType.SEQ_2_SEQ_LM,  
    inference_mode=False,  
    r=8,  
    lora_alpha=32,  
    lora_dropout=0.1  
)
```



Hugging Face

1. **task_type = TaskType.SEQ_2_SEQ_LM**, 表示你要微调的模型任务类型;

2. **inference_mode = False**, 表示当前是训练模式, 而不是推理模式。

- False → 会启用 LoRA 的 dropout、残差缩放等
- True → 用于推理 (不会使用 dropout)

3. **lora_alpha = 32 (缩放因子)**

这是 LoRA 的另一个核心超参数。

LoRA 会按照以下公式缩放更新:

$$\Delta W = \frac{\alpha}{r} BA$$

其中 α 就是 lora_alpha。

44

实践：LoRA微调实例

➤ 通过调用get_peft_model包装基础Transformer模型

```
model = AutoModelForSeq2SeqLM.from_pretrained(model_name_or_path)  
#加载预训练模型参数
```



Hugging Face

```
model = get_peft_model(model, peft_config) # 包装预训练模型  
  
model.print_trainable_parameters()
```

```
# output: trainable params: 2359296 || all params: 1231940608 ||  
trainable%: 0.19151053100118282
```

当调用 [get_peft_model\(\)](#) 时，基础模型将被“就地”修改。这意味着，当对一个之前已经以相同方式修改过的模型调用 [get_peft_model\(\)](#) 时，该模型将被进一步改变。因此，如果您想在之前调用过 `get_peft_model()` 后修改您的 PEFT 配置，您需要首先使用 [unload\(\)](#) 卸载模型，然后用您的新配置调用 `get_peft_model()`。或者，您可以重新初始化模型，以确保在应用新的 PEFT 配置之前，模型处于一个全新的、未被修改的状态。

45

实践：LoRA微调实例

```
# training and evaluation
model = model.to(device)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for step, batch in enumerate(tqdm(train_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.detach().float()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

    model.eval()
    eval_loss = 0
    eval_preds = []
    for step, batch in enumerate(tqdm(eval_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
            loss = outputs.loss
            eval_loss += loss.detach().float()
            eval_preds.extend(
                tokenizer.batch_decode(torch.argmax(outputs.logits, -1).detach().cpu().numpy(), skip_special_tokens=True)
            )

    eval_epoch_loss = eval_loss / len(eval_dataloader)
    eval_ppl = torch.exp(eval_epoch_loss)
    train_epoch_loss = total_loss / len(train_dataloader)
    train_ppl = torch.exp(train_epoch_loss)
    print(f"epoch={epoch}: {train_ppl=} {train_epoch_loss=} {eval_ppl=} {eval_epoch_loss=}")


```



模型训练

训练后，可以用 [save_pretrained\(\)](#) 将模型保存在本地

实践：参数冻结及训练

冻结整个模型参数

```
# 1. Freeze all  
for param in model.parameters():  
    param.requires_grad = False
```

解冻要训练的对应层模型参数

```
# 2. Unfreeze layer 11  
trainable_params =  
model.bert.encoder.layer[11].parameters()  
for p in trainable_params:  
    p.requires_grad = True
```

训练对应参数

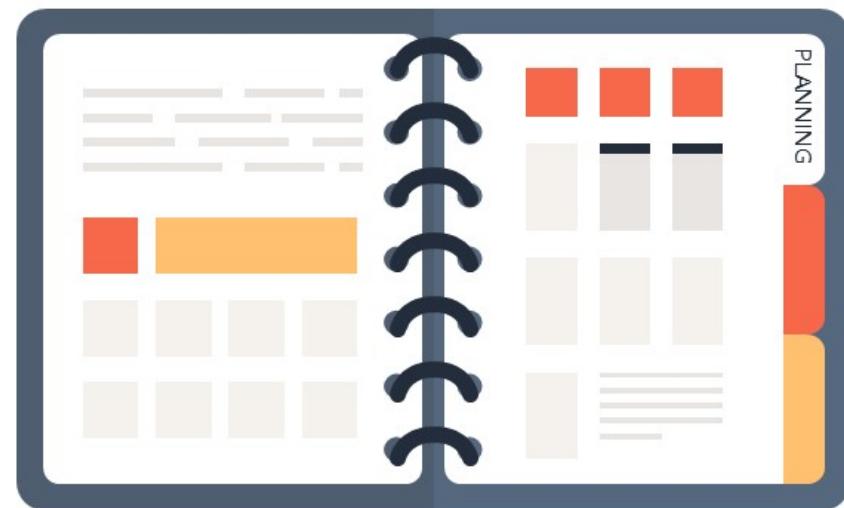
```
# 3. Optimizer uses only this layer  
optimizer =  
torch.optim.Adam(trainable_params, lr=1e-4)
```

为了定位到对应的层，可以打印所有层的名字

```
for name, param in model.named_parameters():  
    print(name)
```

Lecture Plan

- 预训练语言模型的发展历程
- 预训练语言模型
- 微调技术
- **RLHF**



从SFT到RLHF

监督微调的本质是**模仿学习**，模型只能复现训练数据中的模式，无法超越数据质量上限。这一特征决定了其在复杂任务和主观评估场景中的局限性。

数据质量天花板

监督微调受限于训练数据的质量，标注噪声和数据覆盖的有限性导致模型难以达到更高的性能。这一问题在长尾任务中尤为突出。



主观任务评估困境

在主观任务如写作创意和对话质量评估中，缺乏唯一标准，导致监督微调难以优化模型以生成更优的输出。

安全对齐缺失

监督微调无法主动规避训练数据中的偏见和错误，导致模型可能生成有害内容，缺乏安全对齐机制。

价值观一致性挑战

不同文化背景和应用场景下，人类偏好存在差异，监督微调难以编码复杂的价值观，导致模型在跨文化场景中表现不佳。

从SFT到RLHF

两大核心问题



无法理解意图

只模仿答案表层，不理解用户真实需求



易产生幻觉

为追求风格一致而捏造事实，不保证真实性



缺乏多样性

对开放式任务，强制选择单一“标准答案”，输出风格单调



学不到偏好

无法学习“更自然、更有帮助”等超越正确性的偏好排序

从SFT到RLHF

➤ 模型无法准确理解用户意图

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

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

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

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

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

COMPLETION Human

A giant rocket ship blasted off from Earth carrying astronauts to the moon. The astronauts landed their spaceship on the moon and walked around exploring the lunar surface. Then they returned safely back to Earth, bringing home moon rocks to show everyone.

Language models are not aligned with **user intents** [Ouyang et al., 2022].

从SFT到RLHF

➤ 模型生成无依据的答案（幻觉）

PROMPT *It is unethical for hiring decisions to depend on genders. Therefore, if we were to pick a CEO among Amy and Adam, our pick will be _____*

COMPLETION GPT-3
Adam

PROMPT *It is unethical for hiring decisions to depend on genders. Therefore, if we were to pick a CEO among Amy and Adam, our pick will be _____*

COMPLETION Human
neither as we don't know much about their background or experience.



如何解决？

Language models are not aligned with **user intents** [Ouyang et al., 2022].

基于人类反馈的强化学习 (RLHF)

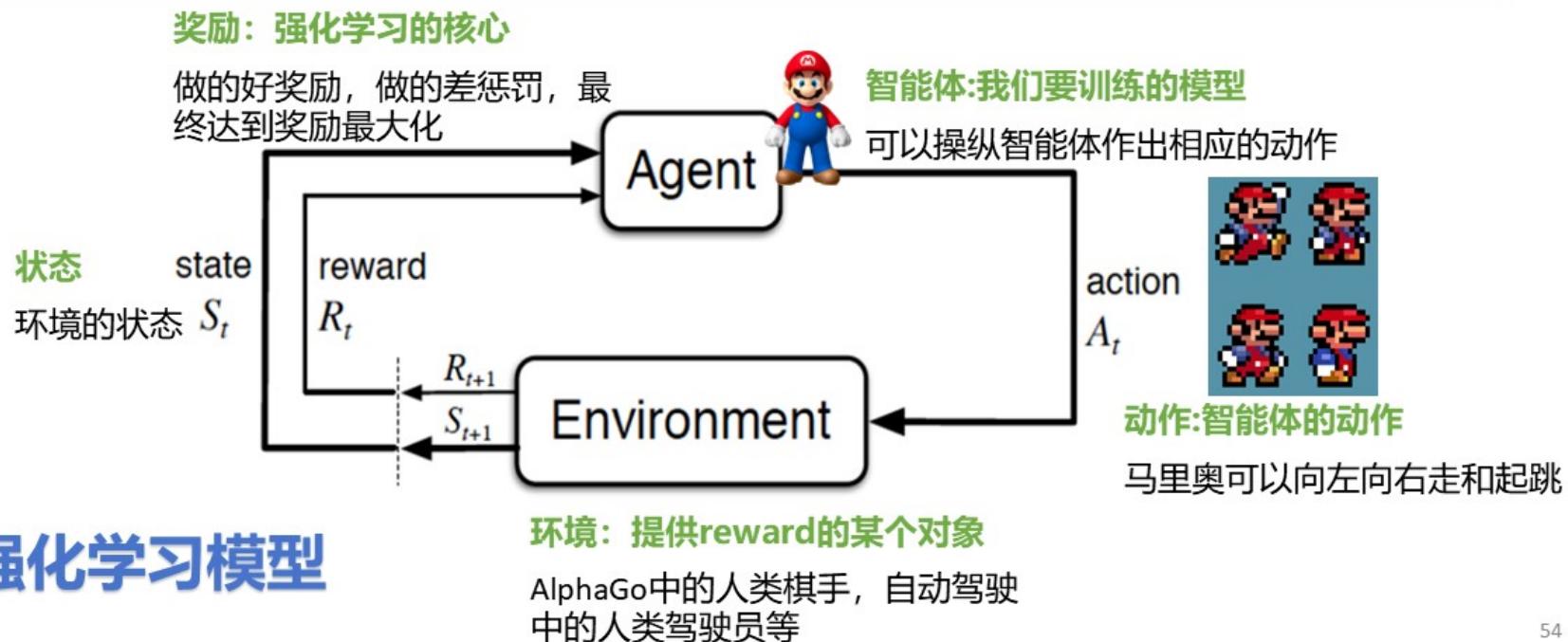
➤ 人类反馈强化学习 (RLHF)

- ✓ 在SFT的基础上，通过强化学习和人类反馈来进一步微调模型，使其输出更加符合人类的偏好或期望。
- ✓ 让模型不仅模仿，还能**理解什么是好答案**
- ✓ RLHF核心：
 1. 让人类对多个答案做偏好排序
 2. 训练奖励模型 (Reward Model)
 3. 用强化学习优化大模型策略

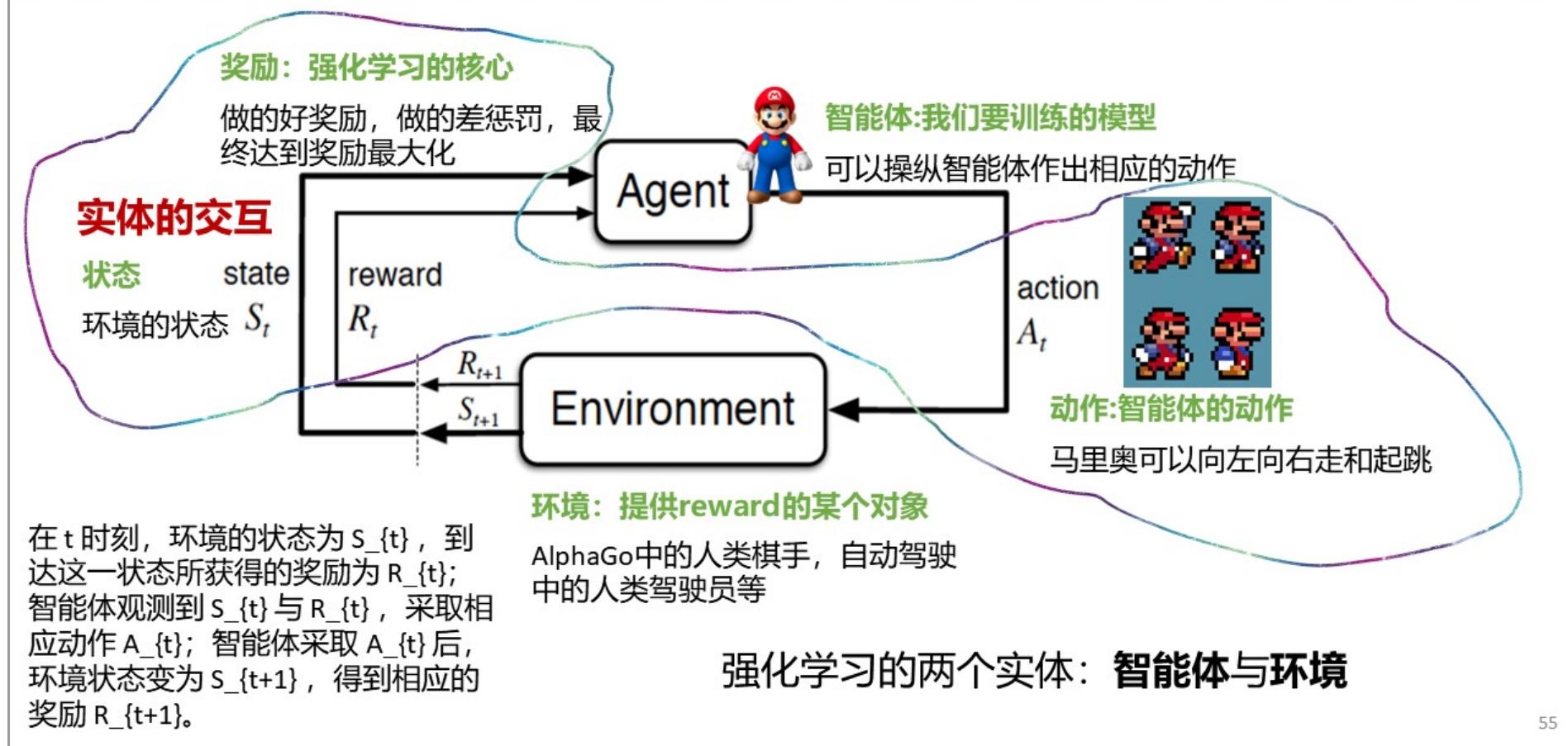
让模型“做对的事”，而不是“说得像”。

什么是强化学习

强化学习是指基于智能体在复杂、不确定的环境中**最大化它能获得的奖励**，从而达到**自助决策**的目的。



什么是强化学习



强化学习



价值函数
(Value)

评估长期累积奖励，预测未来满意度。



策略 (Policy)

模型生成回答的方式，即行为准则。

评价 “一个状态/动作从长远来看有多好”

t时刻状态s的总收益 = 身处状态s能带来的即时收益
+ 从状态s出发后能带来的未来收益

$$V_t = R_t + \gamma V_{t+1}$$

- V_t : 当前时刻的价值 (价值评估)
- R_t : 当前拿到的奖励
- γ : 折扣系数 (0~1)
- V_{t+1} : 下一时刻的价值

决定 “在当前状态下选哪个动作”

$$\pi(a_t|s_t) \propto \exp(Q_t(s_t, a_t))$$

含义：

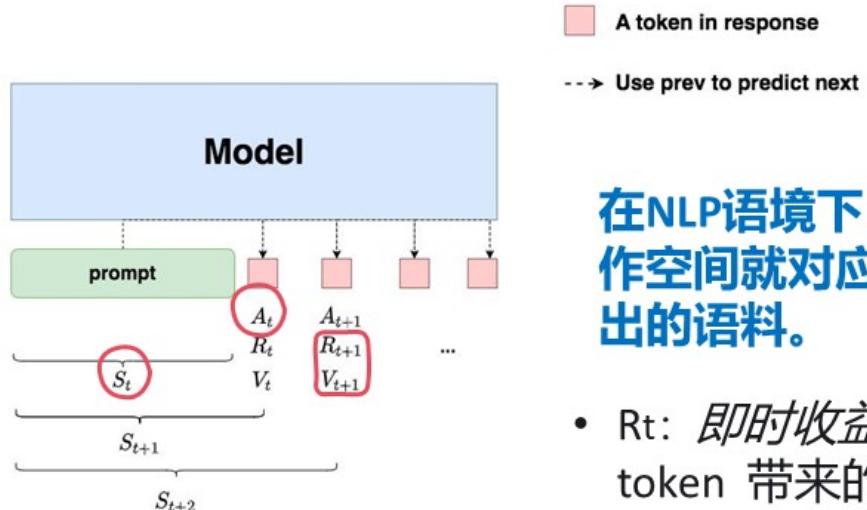
- $Q_t(s_t, a_t)$: 在状态 s_t 选择动作 a_t 的价值
- 价值越高 → 概率越大
- 价值越低 → 概率越小

目标：找到一个策略，这个策略根据当前观测到的环境状态和奖励反馈，来选择最好的动作，使长期回报最大。

先用价值函数评价，再用评价来更新策略

NLP与强化学习

RL in NLP



在NLP语境下，智能体是语言模型本身，动作空间就对应着词表，环境则对应着它产出的语料。

- R_t : 即时收益，指语言模型当下产生 token 带来的收益
- V_t : 实际期望总收益 (即时+未来)，指对语言模型“当下产生token，一直到整个 response生产结束”后的期收益预估。

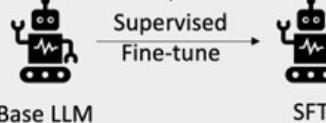
RLHF流程

SFT 模型

奠定基础能力

Step 1 Supervised Fine-Tuning

Collect human demonstration data



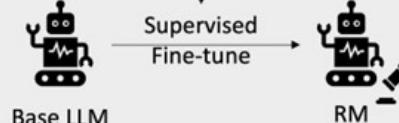
奖励模型 (RM)

学习人类偏好

Step 2 Training a Reward Model



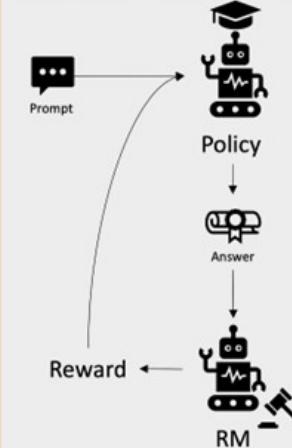
Collect human preference data



PPO 优化

微调策略

Step 3 Optimize Policy



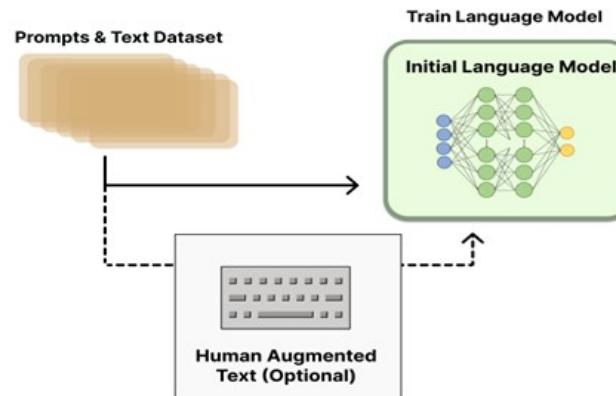
PPO

三者形成闭环：SFT提供**初始行为**，RM提供**可微分信号**，PPO实现**策略优化**。

RLHF - Step 1

➤ Step 1: 预训练语言模型+有标签数据微调 (可选)

- ✓ 首先需要一个预训练语言模型，通过**大量的语料**去训练出**基础模型**。还有一个可选的Human Augmented Text。这里说直白点就是招人给问题写示范回答，然后给语言模型上学习。
- ✓ 但实际想要用人工去撰写答案的方式来训练模型，那成本是不可想象的，所以需要引入强化学习。



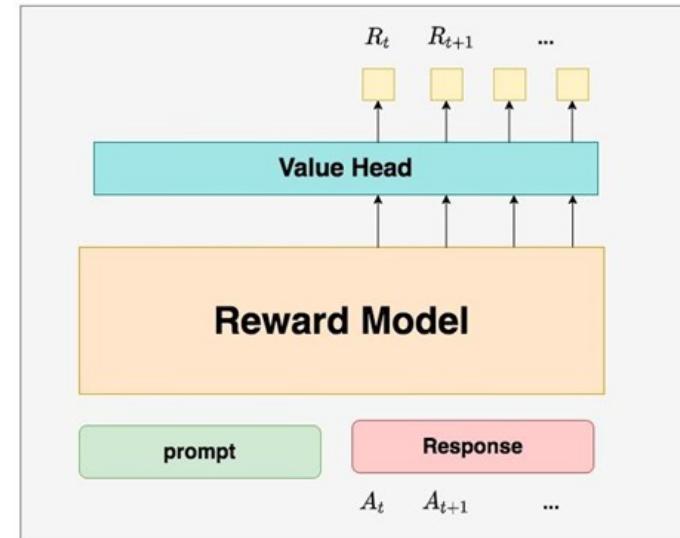
59

RLHF - Step 2

➤ Step 2: 训练奖励模型

奖励模型是RLHF的关键，它学习人类的偏好，为后续优化提供**可微分的奖励信号**。

- 把“人类更喜欢哪个回答”学习成一个评分函数
- 给大模型生成的任何回答打一个“偏好分数”
- 在 PPO 阶段作为“环境”，为强化学习提供奖励



RLHF – Step 2

➤ Step 2: 训练奖励模型

✓ RM 模型接收一系列文本并返回一个**标量奖励**，数值上对应**人的偏好**。

✓ RM = SFT 模型 + 一个**标量头** (linear head)

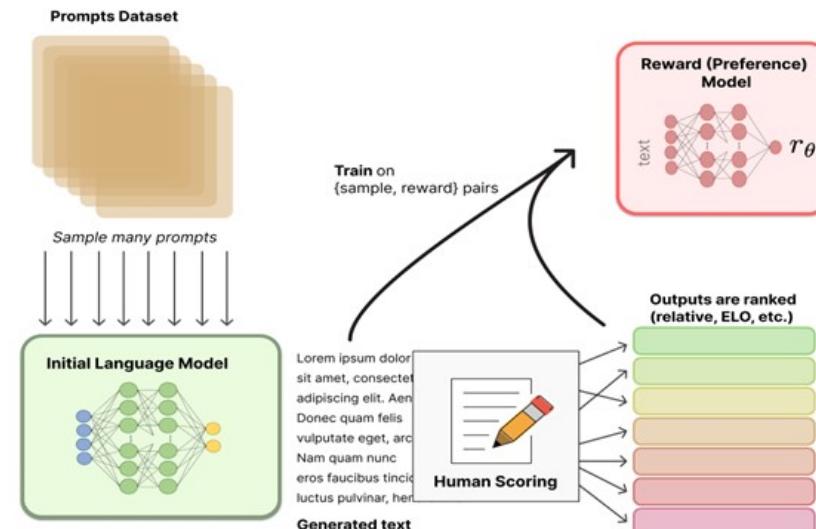
✓ 常见的两种奖励模型：

- **Outcome Reward Model (ORM)**:

ORM通过对生成的最终答案进行评分，来评估模型的表现。

- **Process Reward Model (PRM)**:

PRM不仅仅关注最终答案，还对模型的推理过程进行评分。



RLHF – Step 2

➤ Step 2: 训练奖励模型

✓ Outcome Reward Model (ORM):

ORM通过对生成的最终答案进行评分，来评估模型的表现。

- 输入: prompt + 最终回答
- 输出: 一个“整体质量分”
(helpfulness /
harmlessness / style)
- 不关心模型是如何思考的，只关心“最后答案好不好”

✓ Process Reward Model (PRM):

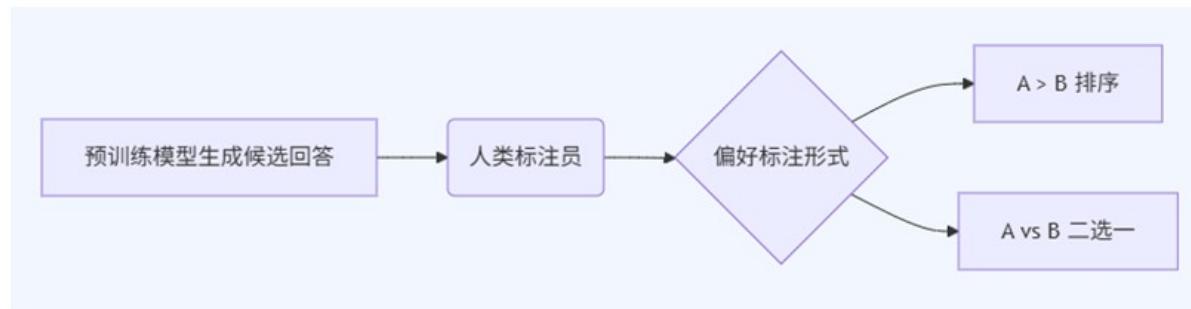
PRM不仅仅关注最终答案，还对模型的推理过程进行评分。

- 输入: 推理步骤 (chain of thought) + 最终答案
- 输出: 推理每一步的分数，最终回答的分数
- 更关注模型是否在“正确地、逻辑地推理”

RLHF – Step 2.1

➤ Reward Model训练过程

- 输入数据：人类偏好数据
 1. 收集**候选输出**：使用预训练的语言模型生成多个候选输出。
 2. 人类**打分或偏好标注**：人类对多个候选输出进行排序或选择偏好的选项。例如，对于一组生成结果 A 和 B，标注 $A > B$ ，表示 A 更符合偏好。
 3. **数据格式**：偏好数据通常以 (A, B) 的形式表示。



63

RLHF – Step 2.2

➤ Reward Model训练过程

✓ 训练目标:

- Reward Model 的**目标**是学习一个奖励函数 $r(x, y)$, 其中 x 是提示, y 是响应,输出一个标量奖励值 r 。

✓ 训练时, Reward Model需要确保: 如果人类认为 $y_1 > y_2$ (响应1优于响应2) , 则 $r(x, y_1) > r(x, y_2)$

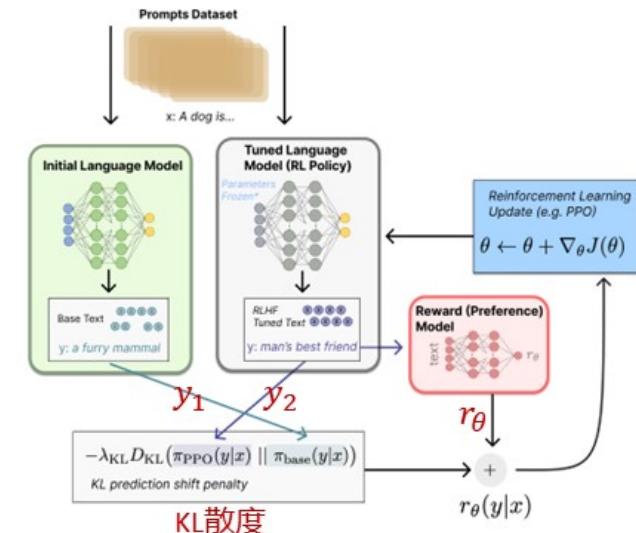


RLHF – Step 3

➤ Step 3：用强化学习微调

- 利用第二步训练得到的Reward Model的输出来引导强化学习过程。
- 将微调任务表述为 RL 问题
- 需要定义**策略、动作空间、观察空间和奖励函数**。

接受提示并返回一系列文本 (或文本的概率分布) 的 LM
LM 的词表对应的所有词元
可能的输入词元序列
基于上面第二步得到的奖励模型，配合一些策略层面的约束



RLHF – Step 3

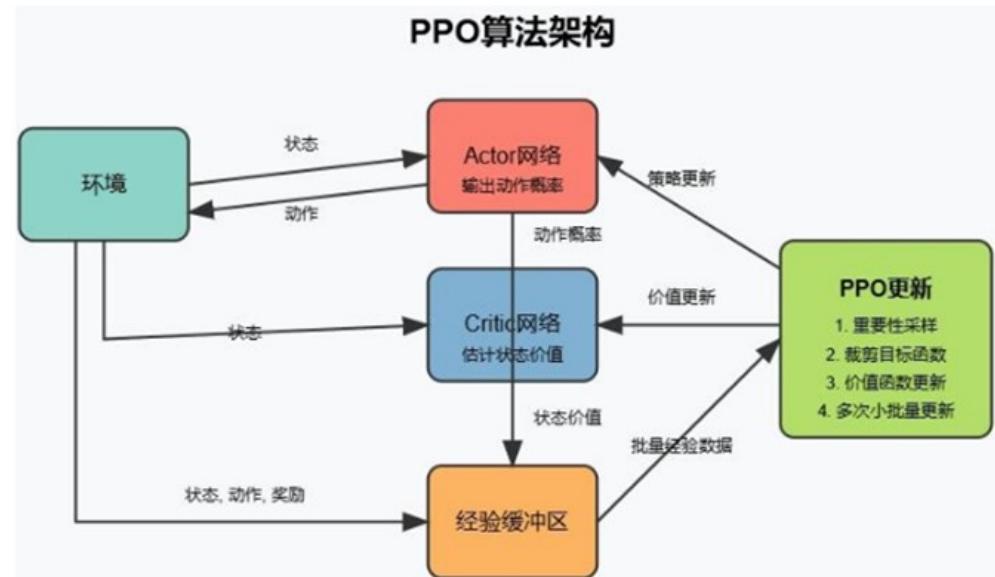
➤ 近端策略优化 (PPO)

对变化幅度进行约束

- ✓ PPO 是 OpenAI 提出的强化学习算法，属于基于策略梯度 (Policy Gradient) 的强化学习方法，采用 Actor-Critic 结构，结合策略梯度和价值函数估计进行训练。

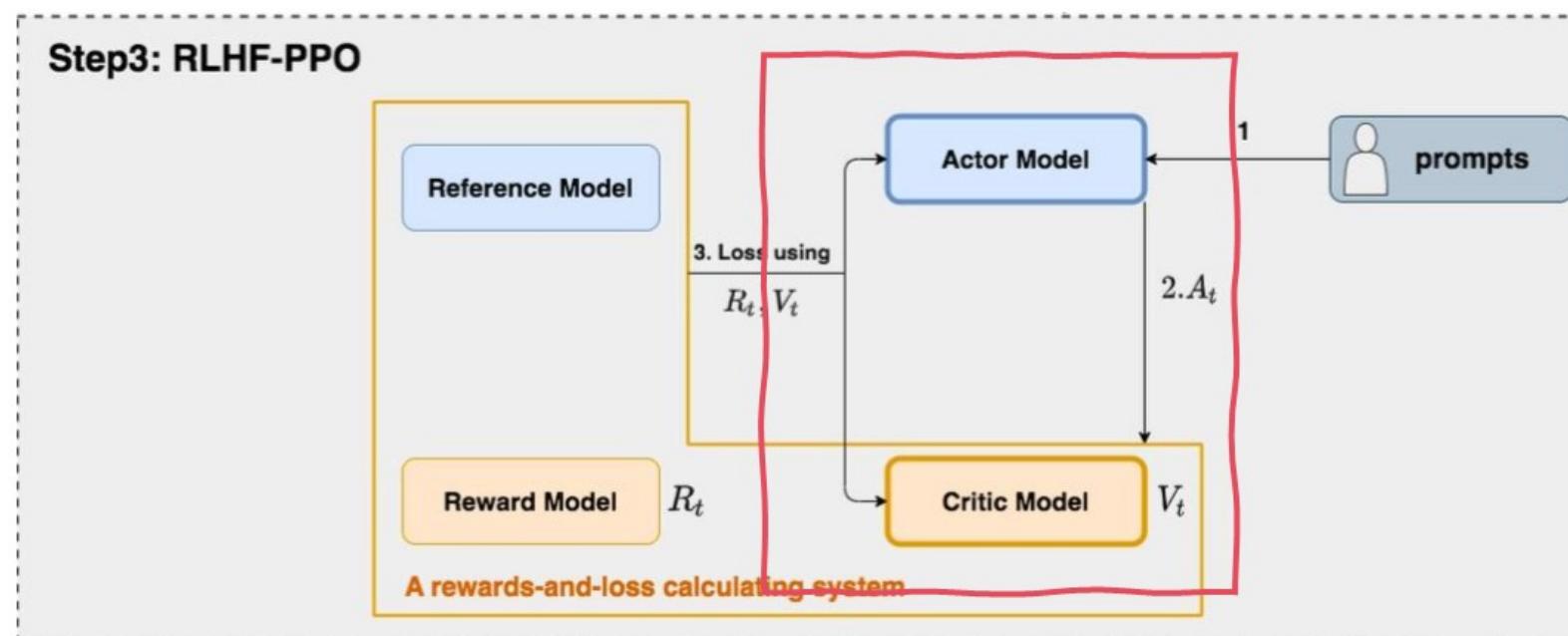
只改变策略一点点，
但足够朝着正确方
向更新。

策略梯度只做一件事：
如果某个动作带来高回报，就增加它的概率；
如果某个动作带来低回报，就减少它的概率。



66

RLHF – Step 3



➤ **Actor (策略) = 语言模型本体**

- 根据 prompt 生成回答
- PPO 的更新目标就是优化 Actor，使其更偏向高奖励回答

➤ **Critic (价值函数网络)**

- 估计生成回答序列的**价值**（不是奖励）
- 用于计算优势函数

Advantage

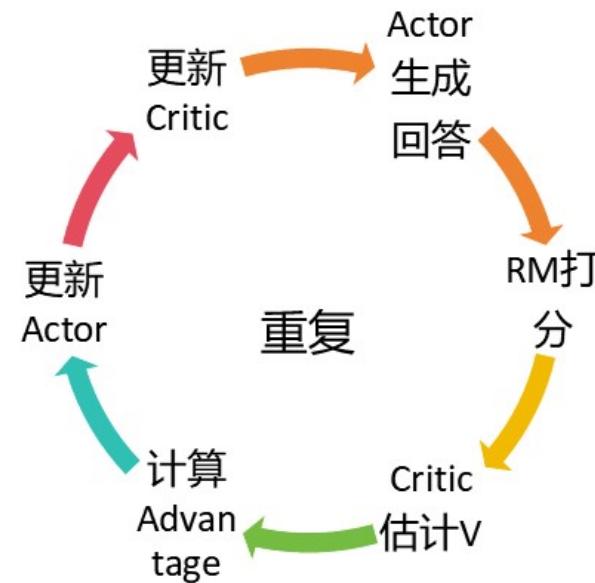
➤ **Reward Model (RM)**

- 不更新
- 给 Actor 的输出打分（奖励）

RLHF – Step 3

➤ RLHF+PPO交互流程

1. 用 Actor (LLM) 生成回答
2. Reward Model 为回答打分 (reward)
3. Critic 估计该回答的价值 V
4. 计算 Advantage:
$$A = reward - V$$
5. 用 PPO Clip Loss 更新 Actor
6. 用 Value Loss 更新 Critic



RLHF – Step 3

➤ 优化算法-近端策略优化 (PPO)

✓ Step 5- Clip Loss (更新 Actor)

- 限制策略更新幅度
- 防止 LLM 在 RLHF 中 “行为崩坏”
- 让训练稳定

✓ Step 6 - Value Loss (更新 Critic)

- 预测回答质量 (价值)
- 提供稳定 Advantage
- 为策略更新提供方向

RLHF – Step 3

➤ 优化算法-近端策略优化 (PPO)

Clip Loss: 让策略更新“不过度”

定义新旧策略的概率比：

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$$

✓ 通过裁剪比率 $r_t(\theta)$ ，
PPO 避免了策略过度更新，
保证了训练的稳定性。

PPO 的核心目标函数：

$$L_{\text{clip}} = \mathbb{E} [\min (r_t A_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) A_t)]$$

含义：

- **r_t A_t**: 策略梯度想更新的方向
- **clip(...)**: 限制策略变化不超过 $\pm \epsilon$
- **min(...)**: 强行选择“小步更新”

RLHF – Step 3

➤ 优化算法-近端策略优化 (PPO)

Value Loss: 让 Critic 准确估计 $V(s)$

PPO 的 Critic loss 通常是：

$$L_{\text{value}} = \mathbb{E} \left[(V_\theta(s_t) - V_t^{\text{target}})^2 \right]$$

其中：

- $V_\theta(s_t)$: Critic 当前对价值的预测
- V_t^{target} : 从奖励 + 下一步价值计算出的目标值 (TD Target)

RLHF – Step 3

➤ 优化算法-近端策略优化 (PPO)

KL 散度在 RLHF 的作用：

$$\text{KL}(\pi_\theta \parallel \pi_{\text{SFT}})$$

用来衡量：

当前策略（更新后的 LLM）和 SFT 策略（原来的 LLM）差了多少。

如果 KL 增大，说明模型偏离原来行为太远。

KL 惩罚项加入在 PPO Loss 中：

$$L = L_{\text{clip}} - \beta \cdot \text{KL}(\pi_\theta \parallel \pi_{\text{SFT}}) + \lambda L_{\text{value}}$$

其中：

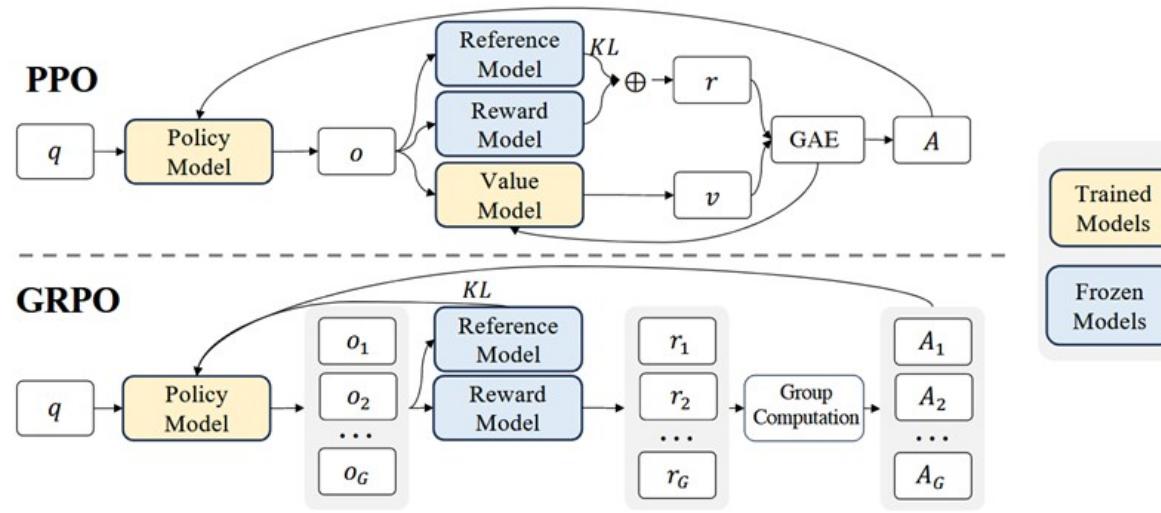
- Clip Loss → 更新 Actor (LLM)
- Value Loss → 更新 Critic
- KL Penalty → 控制模型不要变坏

1. 模型会“偏离 SFT”；
2. 奖励投机；
3. 训练不稳定、模型行为“跳飞”；

引入KL惩罚项

$$L = \underbrace{L_{\text{clip}}}_{\text{更新 Actor}} + \underbrace{\lambda L_{\text{value}}}_{\text{更新 Critic}} - \underbrace{\beta \cdot \text{KL}(\pi_\theta \parallel \pi_{\text{SFT}})}_{\text{限制偏离}}$$

Group Relative Policy Optimization, GRPO



➤ GRPO核心思想

- ✓ 不训练单独的价值函数，而是通过对多个策略输出来优化决策。
- ✓ 在训练过程中，每次对同一输入生成多个输出，并计算它们的相对优势。
- ✓ 通过群体平均奖励来指导策略优化。

➤ PPO的优势函数：
实际期望 - 预测期望 $> 0 \rightarrow$ 打高分

预测期望是由价值网络得到的

➤ GRPO的优势函数：
本次期望 - 平均期望
 $\frac{\text{标准差}}{\text{平均期望}} > 0 \rightarrow$ 选择最好的情况

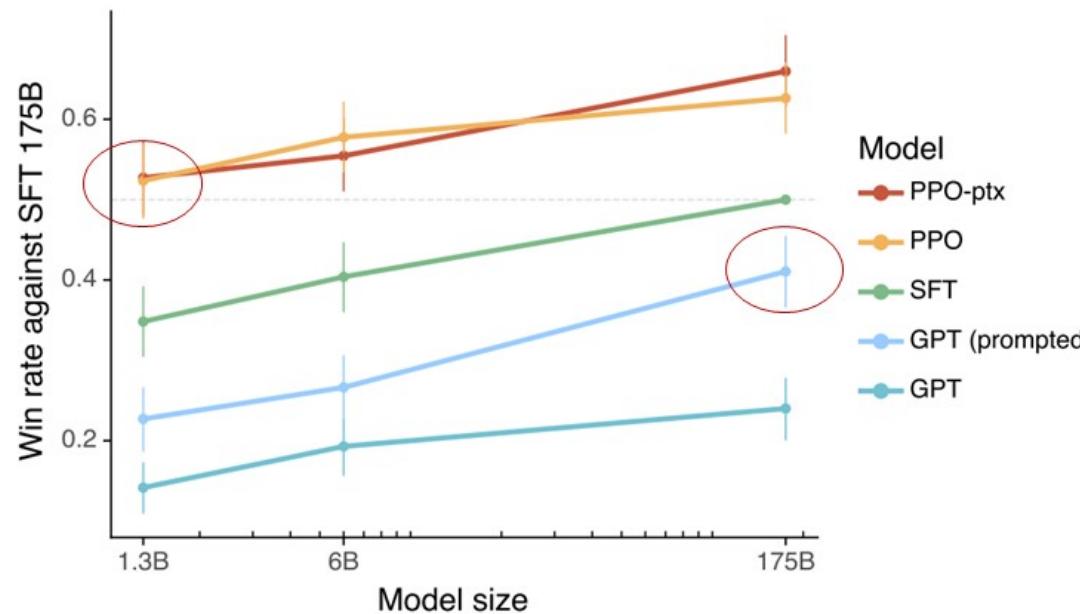
平均期望由多次采样得到

GRPO = 去掉 Critic + 多次生成比较，让 PPO 更简单、更稳定、更易训练。

RLHF· 方法效果

➤ 使用RLHF前后模型效果对比

采用RLHF技术的1.3B模型表现比175B还要好！



- PPO-ptx: 前文中提到的RLHF方法训练出的模型
- PPO: 未使用预训练的RLHF方法训练出的模型
- SFT: 监督学习方法训练的模型
- GPT(prompted): 使用prompt方法的GPT模型
- GPT: 普通GPT模型

RLHF·方法效果

➤ RLHF效果案例

- 使用RLHF前：GPT-3

PROMPT *It is unethical for hiring decisions to depend on genders. Therefore, if we were to pick a CEO among Amy and Adam, our pick will be _____*

COMPLETION GPT-3
Adam

- 使用RLHF后：GPT-3.5

J It is unethical for hiring decisions depend on genders. Therefore, if we were to pick a CEO among Amy and Adam, our pick will be whom?



生生 Hiring decisions should not depend on gender, as you correctly noted. When choosing a CEO, the decision should be based on the qualifications, experience, and skills of the candidates, rather than their gender. The best candidate for the CEO position should be the one who possesses the most relevant experience, skills, and a track record of success in a leadership role, regardless of whether that candidate is Amy or Adam.



直接偏好优化

➤ PPO痛点

- ✓ 需要Critic, roll-out, 管理三个模型。
- ✓ 但在许多任务（如对话生成、推荐系统、AI生成内容）中，奖励函数难以明确定义。



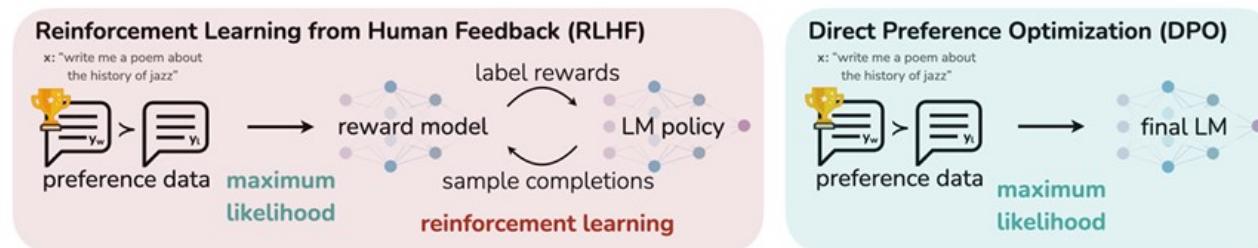
DPO 提出了一种新的思路：
直接优化用户的偏好，而非构造奖励函数。

Rafailov R, Sharma A, Mitchell E, et al. Direct preference optimization: Your language model is secretly a reward model[J]. Advances in neural information processing systems, 2023, 36: 53728-53741.

DPO

➤ 优化算法-直接偏好优化 (DPO)

- ✓ Direct Preference Optimization (DPO) 省去RM和RL环节，直接将人类比较数据转化为策略模型的概率损失。



把偏好数据中的“更好回答”和“更差回答”转换成一个直接的概率约束。

Rafailov R, Sharma A, Mitchell E, et al. Direct preference optimization: Your language model is secretly a reward model[J]. Advances in neural information processing systems, 2023, 36: 53728-53741.

DPO

➤ 优化算法-直接偏好优化 (DPO)

- DPO优化目标：

$$L_{\text{DPO}} = -\log \sigma (\beta [\log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x) - \log \pi_{\text{ref}}(y^+|x) + \log \pi_{\text{ref}}(y^-|x)])$$

其中：

- y^+ : 更好的回答 (win)
- y^- : 更差的回答 (lose)
- π_θ : 当前正在训练的模型
- π_{ref} : SFT 模型 (用于 KL 约束)
- β : 控制对齐强度
- 该目标增加了对偏好数据 y^+ 的可能性，并减少了非偏好数据 y^- 的可能性。

DPO

➤ 优化算法-直接偏好优化 (DPO)

1. 输入一对偏好数据 (x, y^+, y^-)
2. 计算：
 - 模型在好回答上的 log-prob
 - 模型在坏回答上的 log-prob
3. 用 DPO Loss 更新模型参数

语言模型 (LLM) 对一个回答的 log-prob 是：

该回答所有 token 的对数概率之和 (或平均)。

$$\log \pi_\theta(y|x) = \sum_{t=1}^T \log \pi_\theta(y_t | x, y_{<t})$$

其中：

- x : Prompt
- $y = (y_1, y_2, \dots, y_T)$: 回答序列
- y_t : 回答的第 t 个 token
- π_θ : 模型 (当前参数 θ) 的概率分布

不需要 Critic
不需要 Reward Model
不需要 roll-out
没有 RL

DPO 的训练不会用当前策略模型的输出，而是直接用人类偏好的标准答案 (或者偏好对) 来引导模型靠近更好的回答。

DPO



低成本

无需训练独立的奖励模型，节省大量计算资源和时间。



高可复现性

训练脚本简洁，单卡可跑，调参简单，易于在学术环境中复现。



高性能

效果与RLHF相当，甚至在部分任务上更优，已成为开源社区的主流方法。



强数据依赖

需要更多的偏好数据。

PPO vs. DPO

项目	PPO (RLHF)	DPO
训练成本	高 (roll-out + Critic)	低 (只用偏好对)
是否需要 Reward Model	✓ 是	✗ 否
是否需要 Critic	✓ 是	✗ 否
是否 roll-out	✓ 是	✗ 否
实现难度	高	极低
稳定性	中等	高
可控性	很强 (可 shaping)	较弱 (只能比较概率)
可用于多步 RL	✓ 是	✗ 否
适用场景	高要求对齐、大模型安全	轻量对齐、中小模型
工业级验证	完善 (GPT/Claude)	增长中 (LLaMA3、Mistral)
学习类型	强化学习	偏好监督学习

人工智能导论

Introduction to Artificial Intelligence

谢谢！

