实验任务书：大语言模型强化学习（PPO & GRPO）

本任务书面向**没有强化学习基础的本科生**，目标是帮助你在本项目中，循序渐进地：

1. 理解「大模型 + 奖励函数 + RL 更新」的整体流程；
2. 亲手补全并跑通 **PPO** 和 **GRPO** 两种经典算法；
3. 做几组简单的小实验，观察训练曲线和模型行为的变化。

> 建议：完成本任务书时，边做边记笔记，把「问题-思路-结论」写下来，会帮助你真正理解算法而不是只会抄代码。

公式速查（本实验会用到）

不用背公式，你只需要知道它们大概长什么样、每一项代表什么即可。实现代码时可以反复回来对照。

1. 策略梯度与 Advantage

强化学习目标（最大化期望奖励）

$$J(\theta) = \mathbb{E}[R_t]$$

最基础的策略梯度形式（只记住形状即可）

$$\nabla_\theta J(\theta) \approx \mathbb{E}\big[A_t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t)\big]$$

其中 A_t 就是「优势函数（advantage）」。

本项目中使用的最简单 advantage 定义

$$A_t = R_t - V_\phi(s_t)$$

- R_t：从环境（或奖励模型）得到的标量奖励；
- $V_\phi(s_t)$：价值网络估计的状态价值。

2. PPO 的 Clip 目标

PPO 的裁剪目标（你在 `ppo_clip_loss` 中要实现的）

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

其中：

$$r_t(\theta) = \exp\big(\log \pi_\theta(a_t \mid s_t) - \log \pi_{\theta_{\text{old}}}(a_t \mid s_t)\big)$$

3. GRPO 的组内相对优势

对每个 prompt（记为第 i 个），采样 K 个回复，其奖励记为：

- 第 i 个 prompt 的第 k 个回复奖励：$r_{i,k}$，其中 $k = 1, \ldots, K$

组内平均奖励（baseline）

$$\bar{r}_i = \frac{1}{K}\sum_{k=1}^{K} r_{i,k}$$

对应的组内优势

$$A_{i,k} = r_{i,k} - \bar{r}_i$$

在代码中，这就是 `compute_group_advantages` 要做的事情。

4. GRPO 的策略梯度损失

使用组内优势的 REINFORCE 形式：

$$L_{\text{GRPO}} = -\mathbb{E}\big[A_{i,k}\ \log \pi_\theta(a_{i,k} \mid s_i)\big]$$

在 `grpo_loss` 中实现成：

$$L_{\text{GRPO}} \approx -\frac{1}{N}\sum_{n=1}^{N} A_n\ \log \pi_\theta(a_n \mid s_n)$$

5. PPO 中的 KL 惩罚（可选）

为了防止新策略偏离旧策略太远，可以在 PPO 损失上加一个 KL 项：

$$L_{\text{total}} = L_{\text{PPO}} + \beta\, \text{KL}\big(\pi_{\theta_{\text{old}}}(\cdot \mid s)\ \|\ \pi_\theta(\cdot \mid s)\big)$$

在代码中，我们使用一个近似：

$$\mathrm{KL}\big(\pi_{\theta_{\text{old}}} \,\|\, \pi_\theta\big) \approx \mathbb{E}\big[\log \pi_{\theta_{\text{old}}}(a \mid s) - \log \pi_\theta(a \mid s)\big]$$

也就是训练脚本里记录的 `approx_kl` 。

0. 预备工作（环境与代码结构）

目标： 能够在你的机器上跑起来基础脚本，知道每个文件大概干什么。

1. 按 `README.md` 中的说明完成：
   - 创建虚拟环境并 `pip install -r requirements.txt`
   - 确认 `python train_ppo.py` 能够正常启动（即使很慢 / 很快 OOM 都算发现问题）
2. 通读以下文件，尝试用自己的话概括每个文件的功能（可以写在自己的笔记中）：
   - `rlhf_practice/config.py` ： 有哪些超参数？哪些会影响显存占用？
   - `rlhf_practice/data.py` ： 数据是从哪里来的？最终 `DataLoader` 里一条样本长什么样？
   - `rlhf_practice/modeling.py` ： `PolicyValueModel` 做了什么封装？ `evaluate_sequences` 返回了什么？
   - `rlhf_practice/reward.py` ： 奖励函数如何根据文本打分？
   - `train_ppo.py` 、 `train_grpo.py` ： 大致流程是什么？（可以粗略画一条“数据流动路线图”）

> 检查点：你应该能大致回答——「给一条影评文本，程序是如何把它变成奖励、再用奖励更新模型的？」

1. 任务一：理解并实现 PPO 的 Advantage

目标： 理解「优势函数 advantage」的含义，并在代码中实现最基础的 `A = R - V(s)` 。

1.1 阅读 PPO 相关代码

1. 打开 `rlhf_practice/rl/ppo.py` ，重点阅读：
   - `compute_advantages`
   - `ppo_clip_loss`
   - `ppo_step`
2. 打开 `answer/ppo_answer.py` 但**先不要**细看，实现之前只看函数签名和注释，确认：
   - 输入 / 输出张量的形状；

- 它们在 `train_ppo.py` 中是如何被调用的。

1.2 实现 `compute_advantages`

在 `rlhf_practice/rl/ppo.py` 中：

1. 根据注释补全 `compute_advantages`：
   - 从计算图中分离 `values`（使用 `.detach()`）；
   - 计算 `advantages = rewards - values_detached`；
   - 若 `normalize=True`，做标准化：
     `advantages = (advantages - mean) / (std + 1e-8)`。
2. 临时在函数末尾加上简单的 `print` 或 `assert`（完成后可以删掉），例如：
   - `assert rewards.shape == values.shape`
   - `assert advantages.shape == rewards.shape`

1.3 验证实现是否工作

1. 运行：

```
python train_ppo.py
```

2. 观察是否有报错（形状对不上 / `NotImplementedError` 等）。
3. 若能跑通几个 step，注意：

- 日志中的 `reward` 是否为一个合理的数（通常在一个较小区间内波动）；
- `policy_loss`、`value_loss` 是否为有限数值而不是 NaN。

> 检查点：你应能解释为什么要 `values.detach()`，以及为什么标准化 advantage 有助于训练稳定。

2. 任务二：实现 PPO 的 Clip Loss

目标： 理解 PPO 相比普通策略梯度的「裁剪」思想，并在代码中实现 clip loss。

2.1 推导与理解（建议纸上写一遍）

1. 记 PPO 的核心目标为：

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

2. 其中：

-
- 是上一节算出的 advantage。

3. 思考：为什么要 `min`？为什么要把 `r_t` 限制在 `[1-ε, 1+ε]` 范围内？

2.2 在代码中实现

在 `rlhf_practice/rl/ppo.py` 中补全 `ppo_clip_loss`：

1. 计算 `ratio = torch.exp(new_logprobs - old_logprobs)`；
2. 计算 `unclipped = ratio * advantages`；
3. 计算 `clipped_ratio = ratio.clamp(1 - clip_range, 1 + clip_range)`；
4. 计算 `clipped = clipped_ratio * advantages`；
5. 取 element-wise `torch.min(unclipped, clipped)`，再取负平均作为损失。

> 小练习：在实现后，可以写一个小的单元测试脚本（自己写在另一个 `.py`）随机生成 logprobs 和 advantages，打印比较 `unclipped` 与 `clipped` 的分布。

2.3 重新运行 PPO 训练

1. 再次运行：

```
python train_ppo.py
```

2. 观察：

- 训练初期 `policy_loss` 约为多少量级？
- 调大 / 调小 `ppo_clip_range`（例如 0.1, 0.3）时，loss 曲线有没有什么变化？

3. 记录至少一组对比结果（可以截图 loss 曲线或手动抄下几个 step 的日志）。

> 检查点：你应能描述「若不裁剪（即不取 min、不 clamp），模型可能出现什么问题？」。

3. 任务三：理解并实现 GRPO

目标： 理解 GRPO 的「组内相对优势」思想，并在代码中实现对应的损失。

3.1 理解 GRPO 的数据组织方式

1. 打开 `train_grpo.py`，找到如下逻辑：

- 使用 `repeat_interleave(group_size)` 把每个 prompt 复制 `group_size` 次；
- 对每个 prompt 生成 `group_size` 个回复；
- 因此最后的 `rewards` 和 `logprobs` 的长度都是 `batch_size * group_size`。

2. 在纸上画出一个小例子：

- 假设 `batch_size = 2`， `group_size = 3`；
- 写出 `rewards` 如何 reshape 成 `[2, 3]`，每行代表一个 prompt 的 3 个候选回复。

3.2 实现 `compute_group_advantages`

在 `rlhf_practice/rl/grpo.py` 中：

1. 根据注释补全 `compute_group_advantages`：

- 假设 `rewards` 形状为 `[B * K]`；
- reshape 为 `[B, K]`，对每行求均值 baseline；
- `advantages_group = rewards_group - baseline`；
- 展平回 `[B * K]`；
- 若 `normalize=True`，做标准化。

2. 思考：相比 PPO 中使用 `V(s)` 作为基线，这里使用「组内平均奖励」作为基线，有什么优缺点？

3.3 实现 `grpo_loss`

在同一文件中，补全 `grpo_loss`：

1. 确保 advantages 不反向传播：`advantages_detached = advantages.detach()`；
2. 实现带权 REINFORCE 损失：

```
loss = -(advantages_detached * logprobs).mean()
```

3.4 跑通 GRPO 训练

1. 运行：

```
python train_grpo.py
```

2. 观察日志中的：

- `reward`：平均奖励；
- `loss`：GRPO 策略梯度损失；
- `adv_mean`：组内优势的平均值（正常情况下会非常接近 0）。

> 检查点：你应能解释为什么组内优势的均值应该接近 0，以及这对训练有什么好处。

4. 任务四：对比 PPO 与 GRPO

目标： 从实验角度比较两种算法的表现和特点。

建议在相同的设置下进行对比实验：

- 相同的模型（如 `gpt2`）；
- 相同的数据（IMDB）；
- 相近的训练步数 / 时间。

4.1 设计对比实验

1. 选定一组基础配置（在 `config.py` 中）：

- `train_batch_size`、`max_prompt_length`、`max_new_tokens`；
- `ppo_clip_range`、`grpo_group_size`。

2. 对 PPO 和 GRPO 各自跑一段时间（例如各训练 200~500 个 step，视显卡情况而定）。

4.2 记录与分析

至少记录以下信息：

- 训练过程中的平均奖励变化趋势（大致是上升、下降还是波动）；
- 损失曲线大致趋势（PPO 可以关注 `loss/total_no_kl` 与 `loss/total_with_kl`）；

- 训练稳定性：是否容易出现 NaN、梯度爆炸、奖励突然坍塌等现象。

给出你个人的结论，例如：

- 「在这个玩具 reward 下，PPO 上升更快 / 更慢」；
- 「GRPO 对超参数更敏感 / 更鲁棒」；
- 「我更喜欢哪一个，为什么」。

5. 任务五：小扩展（选做）

如果你已经能够顺利跑通 PPO 和 GRPO，可以尝试以下扩展任务：

1. **改造奖励函数**（ `rlhf_practice/reward.py` ）：

- 加入长度惩罚：太长 / 太短的回复扣分；
- 奖励句子以「I think」开头等；
- 奖励包含某些你感兴趣的词汇。

2. **更换模型**（ `ModelConfig.model_name` ）：

- 尝试 `sshleifer/tiny-gpt2` （更小、更快）；
- 或支持中文的模型（需要注意 tokenizer 和显存）。

3. **更换数据集**（ `DataConfig.dataset_name` ）：

- 使用中文评论类数据集；
- 自己准备一个简单的文本文件数据集并用 `datasets` 或自定义 `Dataset` 封装。

4. **日志与可视化**：

- 本项目已集成 TensorBoard 日志，你可以在训练时打开：

```
tensorboard --logdir runs
```

- 观察 PPO 与 GRPO 在 `reward` 、 `loss` 等指标上的曲线差异；以及 PPO 中不同 `kl_coef` 设置下 `kl/approx` 的变化；
- 若愿意，可以再额外导出 CSV，自行用 Jupyter / matplotlib 画更个性化的图。

6. 总结与反思

完成上述任务后，建议你写一份不超过 2 页 A4 的总结，至少包含：

1. 你对 PPO 和 GRPO 的**直觉理解**（用自己的话，而不是公式）；
2. 本项目中，**从 prompt 到奖励再到更新**的全链路描述；
3. 实验中遇到的主要问题（如显存、数值不稳定等）以及你如何解决；
4. 你觉得 RLHF 中最有趣或最难的部分是什么。

如果你希望进一步深入（例如加入 KL 惩罚、学习一个奖励模型，而不是手写规则奖励），可以把你的想法整理出来，我可以在此基础上帮你一起设计下一阶段的实验计划。