CS336 作业 5（对齐）：对齐与推理强化语言

版本 1.0.2

CS336 员工

2025 年春季

1 作业概述

在本作业中，你将获得一些关于训练语言模型以推理数学问题的实践经验。

您将实施的内容。

1. 竞争数学题 MATH 数据集的零样本提示基线，Hendrycks 等人。[2021].
2. 监督微调，基于更强推理模型的推理痕迹（DeepSeek R1，DeepSeekAI 等，2025）。
3. 专家迭代，用于提升推理表现，并获得经过验证的奖励。
4. 群体-相对策略优化（GRPO），用于提升推理性能并获得验证奖励。

对于感兴趣的人，我们将在接下来几天内发布一个完全可选的作业部分，内容是如何将语言模型与人类偏好对齐。

您将运行什么。

1. 测量 Qwen 2.5 数学 1.5B 零样本提示表现（我们的基线）。
2. 在 Qwen 2.5 Math 1.5B 上运行 SFT，并使用 R1 的推理痕迹。
3. 在 Qwen 2.5 Math 1.5B 上运行专家迭代，并获得验证奖励。
4. 在 Qwen 2.5 Math 1.5B 上运行 GRPO，并获得验证奖励。

代码是什么样的。所有赋值代码以及这篇文章都可以在 GitHub 上获得：

github.com/stanford-cs336/assignment5-alignment

请 git clone 存储库。如果有任何更新，我们会通知您，您可以 git pull 获取最新信息。

1. cs336_alignment/*：这里是你写作业 5 代码的地方。注意这里没有代码（除了一点入门代码），所以你应该可以从零开始做任何你想做的事。

2. cs336_alignment/提示/*：为了方便你，我们提供了带有提示的文本文件，以减少将 PDF 中的提示复制粘贴到你的代码中可能出现的错误。

3. 测试/*.py：这里包含了你必须通过的所有测试。你只需要通过考试/test_sft.py 和 test_grpo.py 考试——其他测试都是非强制部分的。这些测试调用测试/adapters.py 中定义的挂钩。你会实现适配器，把代码连接到测试。写更多测试和/或修改测试代码有助于调试代码，但你的实现应通过原始提供的测试套件。

4. README.md：这个文件包含一些关于设置环境的基本说明。

你能用什么。我们期望你从零开始构建大部分强化学习相关的组件。你可以使用像 vLLM 这样的工具从语言模型生成文本（§3.1）。此外，你可以使用 HuggingFace Transformers 加载 Qwen 2.5 Math 1.5B 模型和分词器并执行前向传递（§4.1），但不能使用任何训练工具（例如 Trainer 类）。

如何提交。您将向 Gradescope 提交以下文件：

- writeup.pdf：回答所有书面问题。请排版您的回复。
- code.zip：包含您编写的所有代码。

2 语言模型推理

2.1 赋予动机

语言模型的一个显著应用场景是构建能够处理各种自然语言处理任务的通用系统。在本作业中，我们将重点关注语言模型的一个发展用例：数学推理。它将作为我们设置评估、进行监督微调，并尝试通过强化学习（RL）教导 LM 推理的试验平台。

与我们以往的作业方式将有两个不同之处。

- 首先，我们不会使用之前的语言模型代码库和模型。理想情况下，我们希望使用从之前作业训练出来的基础语言模型，但微调这些模型无法获得令人满意的结果——这些模型太弱，无法展现非平凡的数学推理能力。因此，我们将转向一个现代且高性能的语言模型（Qwen 2.5 Math 1.5B Base），并基于该模型进行大部分工作。
- 其次，我们将引入一个新的基准来评估我们的语言模型。直到目前为止，我们一直接受交叉熵是许多下游任务的良好替代方法。然而，本次作业的目的是弥合基础模型与下游任务之间的差距，因此我们必须使用与交叉熵分开的评估方法。我们将使用 Hendrycks 等人[2021]的 MATH 12K 数据集，该数据集包含具有挑战性的高中竞赛数学问题。我们将通过与参考答案进行比较来评估语言模型的输出。

2.2 思维链推理与现实逻辑推理

语言模型中一个令人兴奋的趋势是利用思维链推理来提升各种任务的表现。思维链指的是逐步推理问题的过程，通过生成中间的推理步骤，最终得出最终答案。

关于大型语言模型的思维链推理。早期的思维链方法通过使用“草稿”将问题拆分为中间步骤，微调语言模型以解决简单的数学任务，如算术 [Nye 等，2021]。其他研究则促使一个强有力的模型“一步步思考”再回答，发现这显著提升了小学数学题等数学推理任务的表现[Wei 等，2023]。

学会用专家反复推理。自学推理者（STaR）[Zelikman 等，2022]将推理框架为一个自助循环：预训练模型首先采样多样的思维链（CoT），只保留那些能得出正确答案的，然后对这些“专家”痕迹进行微调。重复此循环可以提升 LM 的推理能力和求解率。STaR 证明了这种基于自动字符串匹配的专家迭代版本能够在无需人文推理痕迹的情况下启动推理技能。

用验证奖励、o1 和 R1 来推理强化学习。近期研究探讨使用更强大的强化学习算法，并配备经过验证的奖励，以提升推理表现。OpenAI 的 o1（及后续 o3/o4）[OpenAI 等，2024]，DeepSeek 的 R1[DeepSeek-AI 等，2025]和 Moonshot 的 kimi k1.5[Team 等，2025]使用策略梯度方法[Sutton 等，1999]训练通过字符串匹配或单元测试验证正确性的数学和代码任务，展示了竞赛数学和编码性能的显著提升。后续研究如 Open-R1 [Face， 2025]、SimpleRL-Zoo[Zeng 等， 2025]和 TinyZero [Pan 等， 2025]证实了纯强化学习与验证奖励——即使在仅 1.5 亿参数的模型上——也能提升推理性能。

我们的设置：模型和数据集。在接下来的章节中，我们将探讨越来越复杂的方法，用于训练基础语言模型，逐步推理以解决数学问题。本次作业将使用 Qwen 2.5 Math 1.5B 基础模型，该模型是基于 Qwen 2.5 1.5B 模型在高质量合成数学预训练数据上持续预训练而来的 [Yang 等， 2024]。MATH 数据集可在 Together 集群访问，地址为/data/a5-alignment/MATH。

给开源审计员的建议：替代数据集

遗憾的是，由于版权索赔，MATH 数据集尚未公开。如果你在家跟进，可以使用以下开源数学推理数据集之一：

- 倒计时 [Pan 等，2025]，可在此获取：基于英国电视节目《倒计时》的简单综合任务，作为小尺度强化逻辑的热门测试平台。
- GSM8K [Cobbe 等，2021a]，可在此获取：小学数学题，比数学简单，但应帮助你调试正确性并熟悉强化学习推理流程。
- Tulu 3 SFT Math [Lambert 等，2025]，见此处：使用 GPT-4o 和 Claude 3.5 Sonnet 生成的合成数学问题。因为这些答案是合成的，有些答案（甚至问题本身）可能并不完全正确。
- 其他数学 SFT 数据集链接在此。如果没有直接提供的简短基层真值标签（例如 1/2），可以用数学答案解析器如 Math-Verify 处理基准值列。

3 零测定数学分析性能

我们将从测量基础语言模型在 5KMATH 示例测试集上的表现开始。建立该基线有助于理解后续方法如何影响模型行为。

除非另有说明，关于 MATH 的实验将使用 DeepSeek R1-Zero 模型[DeepSeek-AI 等，2025]的提示。我们将此称为 r1_zero 提示：

用户与助手之间的对话。用户提出问题，助理
↪ 解决了问题。助手首先思考心中的推理过程，然后向用户提供答案。推理过程被包含在标签中，答案分别被包含在
↪ 标签中，即 推理过程在这里 回答。
↪
↪
用户：{question}
助理：

r1_zero 提示词位于文本文件 cs336_alignment/prompts/r1_zero.prompt 中。
在提示中，问题指的是我们插入的某个问题（例如，Natalia 卖给了 48 个
四月时她的朋友们，五月她卖出了一半的片段。娜塔莉亚拍了多少片段
四月和五月完全卖出？）。预期模型扮演助手角色，开始生成思考过程（因为我们已经包含左思考标签），闭合思考过程，然后在答案标签内生成最终符号答案，如4x + 10。让模型生成 类似标签的目的是让我们能轻松解析模型输出并与真实答案进行比较，同时当我们看到正确的答案标签时，可以停止响应生成。
关于提示词选择的说明。事实证明，r1_zero 提示词并不是最大化强化学习后下游性能的最佳选择，因为提示词与 Qwen 2.5 Math 1.5B 模型的预训练方式不匹配。Liu 等人[2025]发现，仅仅用问题提示模型（仅用其他方式）就能获得非常高的准确率，例如在进行 100+步强化学习后，匹配 r1_zero 提示。

他们的发现表明，Qwen 2.5 Math 1.5B 已经在这些问答对上进行了预训练。
尽管如此，我们选择了 r1_zero 题目，因为强化学习在短步骤内明显提升了准确性，使我们能够快速了解强化学习的机制和理智检查的正确性，即使最终表现不佳。作为现实检验，你将在作业后面直接与 question_only 题目进行比较。

3.1 使用 vLLM 进行离线语言模型推理

为了评估我们的语言模型，我们需要为各种提示生成续答（回复）。虽然当然可以实现自己的生成函数（例如，你在作业 1 中做的），但高效实现强化学习需要高性能推理技术，而这些推理技术的实现超出了本次作业的范围。因此，在本作业中，我们将推荐使用 vLLM 进行离线批次推断。vLLM 是一款高吞吐量和内存高效的语言模型推理引擎，融合了多种实用的效率技术（例如，优化的 CUDA 内核、用于高效注意力 KV 缓存的 PagedAttention [Kwon 等，2023]等）。使用 vLLM 生成提示列表的续写：

来自 vllm 导入 LLM，SamplingParams

示例提示。
提示 = [
“你好，我叫”，“美国总统是”，“法国首都是”，“人工
智能的未来是”，]

[1] 举例来自 https://github.com/vllm-project/vllm/blob/main/examples/offline_inference.py。

# 创建一个采样参数对象，在换行时停止生成。
sampling_params = 采样参数（
温度=1.0， top_p=1.0， max_tokens=1024， stop=[“\n”] ）

# 创建一个大型语言模型。
LLM = LLM（model=<路径到模型>）

从提示词生成文本。输出是包含提示词、生成文本及其他信息的 RequestOutput 对象 # 的列表。

输出 = LLM.生成（提示，sampling_params）

打印输出。
输出为：
prompt = output.prompt generated_text = output.outputs[0].text print（f“提示词： {prompt！r}， 生成文本： {generated_text！r}”）

在上述示例中，LLM 可以用 HuggingFace 模型的名称初始化（如果本地找不到该模型，该模型会自动下载并缓存），或者用 HuggingFace 模型的路径来初始化。由于下载可能需要较长时间（尤其是较大的模型，如 70B 参数），且为了节省集群磁盘空间（避免每个人都有独立的预训练模型副本），我们在 Together 集群的以下路径下载了以下预训练模型。请不要在 Together 集群中重新下载这些模型：

- Qwen 2.5 数学 1.5B 基础（用于推理实验）：/data/a5-alignment/models/Qwen2.5-Math-1.5B
- Llama 3.1 8B 基础（用于可选指令调优实验）：/data/a5-alignment/models/Llama-3.1-8B
- Llama 3.3 70B Instruct（用于可选指令调优实验）：/data/a5-alignment/models/Llama-3.3-70B-Instruct

3.2 零点 MATH 基线

提示设置。为了评估 MATH 测试集的零样本表现，我们只需加载示例，并提示语言模型使用上面 r1_zero 提示来回答问题。
评估指标。当我们评估一项选择题或二元回答任务时，评估指标很明确——我们测试模型是否准确地输出正确答案。
在数学问题中，我们假设存在已知的基层真值（例如 0.5），但我们不能简单地测试模型是否恰好输出 0.5——它也可以回答1/2。因此，在评估数学时，我们必须解决 LM 语义等价响应匹配的棘手问题。
为此，我们想设计一个答案解析函数，输入模型输出和已知的真实值，返回一个布尔值，指示模型输出是否正确。例如，一个奖励函数可以接收模型的字符串输出，结尾为她卖了 15 个剪辑。以及金色答案 72，如果模型输出正确则返回 True，否则返回 False（此例中应返回 False）。

在我们的数学实验中，我们将使用一种快速且相当准确的答案解析器，该分析器在近期强化逻辑推理研究中使用 [Liu 等，2025]。这个奖励函数在 cs336_alignment.drgrpo_ grader.r1_zero_reward_fn 实现，除非另有说明，你应该用它来评估 MATH 的表现。

生成超参数。生成响应时，我们采样温度为 1.0，topp 为 1.0，最大生成长度为 1024。提示请求模型以字符串 结束其回答，因此当模型输出该字符串时，我们可以指示 vLLM 停止：

# 基于 GRPO 博士：模型完成答案后停止 # https://github.com/sail-sg/understand-r1-zero/blob/ # c18804602b85da9e88b4aeeb6c43e2f08c594fbc/train_zero_math.py#L167
sampling_params.stop = [“”]
sampling_params.include_stop_str_in_output = True

问题（math_baseline）：4 分

（a） 编写脚本评估 Qwen 2.5 Math 1.5B 在 MATH 上的零中场表现。该脚本应（1）加载来自 /data/a5-alignment/MATH/validation.jsonl 的 MATH 验证示例，（2） 将其格式化为字符串提示，使用 r1_zero 提示符向语言模型发送，（3） 为每个示例生成输出。该脚本还应（4）计算评估指标，并（5）将示例、模型生成及相应评分序列化到磁盘，以便后续问题分析。

你的实现中包含一个参数类似的方法 evaluate_vllm 可能会有帮助，因为你以后可以复用：

防御 evaluate_vllm（
vllm_model：LLM，reward_fn：可调用[[str， str]， dict[str, float]]，提示符：List[str]， eval_sampling_params：SamplingParams ） -> 无：

“”“ 在提示词列表上评估语言模型，计算评估指标，并将结果序列化到磁盘。”

交付物：评估零测距数学基础表现的脚本。

（b） 在 Qwen 2.5 Math 1.5B 上运行你的评估脚本。有多少个模型世代属于以下类别：（1） 格式和答案奖励 1 都正确，（2） 格式化奖励 1 和答案奖励 0，（3） 格式化奖励 0 和答案奖励 0？观察到至少 10 个格式奖励为 0 的情况，你认为问题出在基础模型的输出，还是解析器？

为什么？那在（至少 10 个）格式奖励是 1 但答案奖励是 0 的情况下呢？

交付物：对模型和奖励函数表现的评论，包括每个类别的示例。

（c） Qwen 2.5 Math 1.5B 零射击基线在 MATH 上的表现如何？

交付内容：1-2 句带有评估指标的句子。

4 数学监督精细调校

算法 1 监督微调（SFT）

输入初始政策模型π;SFT 数据集 D 1：策略模型π← π 2：
对于 step = 1，…，n_sft_steps do

3：　从 D 中抽取一批问答对 D。
4：　利用模型计算给定问题时回答的交叉熵损失π
通过对交叉熵损失进行梯度步更新模型参数θ，6：输出结束π

推理的监督微调。在本节中，我们将在 MATH 数据集（算法 1）上微调基础模型。我们的目标是提升模型的推理能力，而不是微调它以直接预测正确答案，而是先微调它生成一个思维链推理轨迹，然后给出答案。为此，我们提供了来自 DeepSeek R1 DeepSeek-AI 等[2025]的推理痕迹数据集，发表于

/data/a5-alignment/MATH/sft.jsonl

在实际训练推理模型时，SFT 常被用作第二个强化学习微调步骤的热启。这主要有两个原因。首先，SFT 需要高质量的注释数据（即带有预先推理痕迹），而强化学习只需正确答案作为反馈。其次，即使在注释数据丰富的环境中，强化学习仍可以通过找到比 SFT 数据更好的策略来释放性能提升。遗憾的是，我们使用的模型规模不足以展示 SFT 和 RL 组合时的效应，因此本次作业将分别处理这两个阶段。

4.1　使用 HuggingFace 模型

正在加载一个 HuggingFace 模型和分词器。要从本地 dir 加载 HuggingFace 模型和分名器（在 bfloat16 中，并用 FlashAttention-2 节省内存），可以使用以下入门代码：

from transformers import AutoModelForCausalLM，　AutoTokenizer

模型 = AutoModelForCausalLM.from_pretrained（
“/data/a5-alignment/models/Qwen2.5-Math-1.5B”，　torch_dtype=torch.bfloat16,
attn_implementation=“flash_attention_2”，　） tokenizer = AutoTokenizer.from_pretrained（“/data/a5-alignment/models/Qwen2.5-Math-1.5B”）

前传。加载模型后，我们可以对一批输入 ID 进行前向处理，获取输出的 logits（带有.logits）属性。然后，我们可以计算模型预测对数与实际标签之间的损失：

input_ids = train_batch[“input_ids”].to（device） labels =
train_batch[“labels”].to（device）

logits = model（input_ids）.logits loss
= F.cross_entropy）

保存训练好的模型。训练结束后要将模型保存到目录，可以使用 .save_pretrained（） 函数，将路径传递到目标输出目录。记得保存在 /data/yourusername 下，因为这些标签可能相当大。我们也建议保存标记器（即使你没有修改它），这样模型和标记器才能自包含，并且可以从同一个目录加载。

```
# 保留模型的重量
model.save_pretrained（save_directory=output_dir）
tokenizer.save_pretrained（save_directory=output_dir）
```

梯度积累。尽管在 bfloat16 中加载模型并使用 FlashAttention-2，即使是 80GB 的 GPU 也无法支持合理的批处理容量。要使用更大的批次，我们可以使用一种称为梯度积累的技术。梯度累积的基本思想是，我们不是每批后更新模型权重（即采取优化步骤），而是在多个批次中积累梯度，然后再进行梯度步骤。直观上，如果我们有更大的 GPU，我们应该同时计算 32 个样本的梯度，而不是把它们分成 16 批，每批 2 个样本，最后再平均，结果是一样的。

梯度累积在 PyTorch 中实现起来非常简单。回想每个权重张量都有一个属性 .grad，用于存储其梯度。在调用 loss.backward（之前），.grad 属性是 None。调用 loss.backward（） 后，.grad 属性包含梯度。通常，我们会先进行优化步骤，然后用 optimizer.zero_grad（） 将梯度归零，这样会重置权重张量的 .grad 场：

```
对于输入，标签在 data_loader：# 前向传递。
    logits = 模型（输入） 损耗 = loss_fn（logits， labels）

    # 后传。
    loss.backward（）

    # 更新权重。
    optimizer.step（）
    # 零渐变，为下一次迭代做准备。
    optimizer.zero_grad（）
```

实现梯度累积时，我们只需每 k 步调用 optimizer.step（） 和 optimizer.zero_grad（），其中 k 是梯度累积步数。我们在调用 loss.backward（）之前将损失除以 gradient_ accumulation_steps，使梯度在各梯度累积步骤中均值。

```
gradient_accumulation_steps = 4，idx（输入，标签）在枚举（data_loader）：# 前向传递。

    logits = 模型（输入） 损耗 = loss_fn（logits， labels） / gradient_accumulation_steps

    # 后传。
    loss.backward（）

    如果 （idx + 1） % gradient_accumulation_steps == 0：
        # 更新会加权每个“gradient_accumulation_steps”批次。
        optimizer.step（）
```

```
# 每批“零 gradient_accumulation_steps”梯度。
optimizer.zero_grad（）
```

因此，训练时的有效批次规模乘以 k，即梯度累积步数。

4.2 SFT 辅助方法

接下来，我们将实现一些辅助方法，这些方法将在 SFT 及后续强化学习实验中使用。关于命名法的简要说明：在接下来的章节中，我们将交替使用提示词时的模型完成度称为“输出”、“完成”或“响应”。

提示词和输出的标记化。对于每对问题和目标输出（q， o），我们将分别分词问题和输出并串联。然后，我们可以用 SFT 模型（或后面的强化学习策略）对输出的对数概率进行评分。此外，我们还需要构造一个 response_mask：一个布尔掩码，对所有反应中的标记为真，对所有问题和填充标记为假。我们将在训练循环中使用该掩码，确保只计算响应令牌的损失。

问题（tokenize_prompt_and_output）：提示与输出令牌化（2 分）

交付物：实现一种方法 tokenize_prompt_and_output，将问题和输出分别标记化，将它们串联起来，并构造一个 response_mask。推荐使用以下接口：

def tokenize_prompt_and_output（prompt_strs， output_strs， tokenizer）：分词化
提示和输出字符串，并构造一个遮罩，响应标记为 1，其他标记（提示或填充）为 0。

Args：

prompt_strs： 列表[str] 提示词字符串列表。

output_strs： list[str] 输出字符串列表。

分词器：PreTrainedTokenizer 用于分词工具。

返回：

DICT[力量，火炬。张量]。设 prompt_and_output_lens 是一个包含 长度的列表
标记化的提示词和输出字符串。那么返回的词典应包含以下键：

input_ids 火把。形状张量（batch_size，max（prompt_and_output_lens）- 1）：分词化的提示和输出字符串，最终的令牌被切掉。

标签火炬。形状张量（batch_size，max（prompt_and_output_lens） - 1）：移动输入 ID，即没有第一个标记的输入 ID。

response_mask 火把。形状张量（batch_size，max（prompt_and_output_lens） 1）：标签中响应标记上的掩码。

要测试你的代码，实现[adapters.run_tokenize_prompt_and_output]。然后，运行
用 UV 测试，运行 pytest -k test_tokenize_prompt_and_output，确保你的实现通过。

记录每个令牌熵。在进行强化学习时，跟踪每个代币的熵通常很有用，以判断模型的预测分布是否变得（过）自信。我们现在将实现这一点，并比较每种微调方法如何影响模型的预测熵。

支持为 X 的离散分布 p（x）的熵定义为

$$H(p) = -\sum_{x \in X} p(x) \log p(x)。 \tag{1}$$

给定我们的 SFT 或 RL 模型的 logit，我们将计算每个代币的熵，即每个下一代币预测的熵。

问题（compute_entropy）：每个代币熵（1 点）

交付物：实现一种方法 compute_entropy 计算下一代币预测的每个代币熵。推荐使用以下接口：

def compute_entropy（logits： torch.张量）->火炬。张肌：

获取下一个标记预测的熵（即词汇维度上的熵）。

Args：

洛吉茨：火炬。张量张量 形状为（batch_size、sequence_length、vocab_size），包含未归一化的 logit。

返回：

炬。张量形状（batch_size，sequence_length）。每个下一个代币预测的熵。

注意：你应使用数值稳定的方法（例如使用 logsumexp）以避免溢出。

要测试你的代码，实现[adapters.run_compute_entropy]。然后运行 uv run pytest -k test_compute_entropy，确保实现通过。

从模型中获取对数概率。从模型中获得对数概率是我们在 SFT 和 RL 中都需要的基础。
对于前缀 x、产生下一个令牌 logitf（x） ∈R 的 LM，以及标签 y∈V，y 的对数概率为

$$\log p(y \mid x) = \log [\text{softmax}(f(x))], \tag{2}$$

其中符号[x]表示向量 x 的第 y 个元素。
你需要使用数值稳定的方法来计算，也可以自由使用 torch.nn.functional 的方法。我们还建议包含一个参数，可选择性地计算和返回令牌熵。

问题（get_response_log_probs）：响应对数概率（及熵）（2 分）

交付物：实现一种方法 get_response_log_probs，从因果语言模型中获得每个词的条件对数概率（给定前述词），并可选地获取模型下一词牌分布的熵。推荐使用以下接口：

def get_response_log_probs（
模型：PreTrainedModel，input_ids：火炬。
张量，标签：火炬。张量，
return_token_entropy：bool = False，） ->
dict[str， torch.张量]：

Args：

模型：预训练模型 HuggingFace，用于评分（放置在正确的设备上，若不计算梯度，则处于推理模式）。

input_ids：火把。张量形状（batch_size，sequence_length），由你的分词方法生成的串接提示+响应令牌。

标签：Torch。张量形状（batch_size，sequence_length），由你的分词方法生成的标签。

return_token_entropy：布尔 如果为真，也通过调用 compute_entropy 返回每个代币熵。

返回：

DICT[力量，火炬。张量]。

“log_probs”形状（batch_size，sequence_length），条件对数概率 log p（x| x）。

“token_entropy”可选，形状（batch_size，sequence_length），每个位置的每个符号熵（仅在 return_token_entropy=真时出现）。

实施建议：

· 通过 model（input_ids）.logits 获取日志。

要测试你的代码，实现[adapters.run_get_response_log_probs]。然后运行 uv run pytest -k test_get_response_log_probs，确保测试通过。

推荐采用以下接口：SFT 微批量列车步骤。我们在 SFT 中最小化的损失是给定提示时目标输出的负对数似然。为了计算这种损失，我们需要计算给定提示时目标输出的对数概率，并对输出中所有标记进行加和，掩盖提示词中的标记和填充标记。

我们将为此实现一个辅助功能，之后在强化学习中也会加以利用。

问题（masked_normalize）：蒙面归一（1 分）

交付物：实现一种方法 masked_normalize，对张量元素求和并以常数归一化，同时尊重布尔掩模。
推荐使用以下接口：

def masked_normalize（
张量：火炬。张量，面罩：火炬。
张量，normalize_constant：
float，dim：int |无 = 无，） ->火炬。张肌：

对一个维度求和并以常数归一化，只考虑掩码 =1 的元素。

Args：

张量：火炬。张量 张量 用于求和归一化的张量。

面具：火炬。张量 与张量形状相同;1 的局面也包含在和中。

normalize_constant：浮点常数以除以进行归一化。

DIM：智力 |在归一化前，没有需要随之求和的维数。如果无，则对所有维度求和。

返回：

炬。张量为归一化和，其中蒙蔽元素（mask == 0）不对求和贡献。

要测试你的代码，实现[adapters.run_masked_normalize]。然后运行 uv run pytest -k test_masked_normalize，确保通过。

SFT 微批次列车步骤。我们现在准备为 SFT 实现单一微批次列车步骤（记住，对于火车小批次，我们会迭代多个微批次，如果 gradient_accumulation_steps >1）。

问题（sft_microbatch_train_step）：微批次列车步骤（3 点）

交付物：实现 SFT 的单次微批次更新，包括交叉熵损失、掩码求和和梯度缩放。推荐使用以下接口：

def sft_microbatch_train_step（
policy_log_probs：火把。张量，response_mask：火炬。
张量，元组，gradient_accumulation_steps：int，
normalize_constant：float = 1.0，） ->元组[火炬。张量，力
量，火炬。张量]]：

对微批次执行前后传递。

Args：

policy_log_probs（batch_size，sequence_length），来自训练中的 SFT 策略中的每个令牌日志概率。

response_mask（batch_size，sequence_length），1 用于响应标记，0 用于提示/填充。

gradient_accumulation_steps 每个优化步骤的微批次数量。

normalize_constant 除以和的常数。保持 1.0 版本没问题。

返回：

双重[火炬。张量，力量，火炬。张量]]。

标量张量损失。微批量损失，经过梯度积累调整。我们会返回这个数据以便记录。

包含底层损失调用的元数据，以及你可能想记录的其他统计数据。

实施建议：

· 你应该在这个函数中调用 loss.backward（）。一定要调整梯度积累。

要测试你的代码，实现[adapters.run_sft_microbatch_train_step]。然后运行 uv run pytest -k test_sft_microbatch_train_step，确认通过。

循环中记录世代。做一些包含从模型生成的环内日志总是很好的做法，推理 SFT/RL 也不例外。写一个函数 log_generations，提示你的模型对某些给定提示（例如从验证集中抽样的）生成响应。最好为每个例子至少记录以下内容：

1. 输入提示。
2. SFT/RL 模型产生的响应。
3. 真实答案。
4. 奖励信息，包括格式、答案和总奖励。
5. 响应的平均代币熵。
6. 平均响应长度，正确回答的平均响应长度，错误回答的平均响应长度。

问题（log_generations）：记录世代（1 分）

交付物：实现一个函数 log_generations，用于记录模型中的生成。

4.3 SFT 实验

利用上述部分，你现在将实现完整的 SFT 过程（算法 1），以微调 MATH 数据集上的 Qwen 2.5 Math 1.5B 基础模型。/data/a5-alignment/MATH/sft.jsonl 中的每个示例由一个格式化的提示词和一个目标响应组成，目标响应包括一个思路推理链和最终答案。特别地，每个示例都是类型为{“prompt”： str， “response”： str}的 JSON 元素。为了跟踪模型在培训过程中的进展，你应定期在 MATH 验证集上进行评估。你应该用两块 GPU 运行脚本，一块用于策略模型，另一块用于 vLLM 实例来评估策略。为了实现这一点，这里有一些初始化代码，用于初始化 vLLM，并在每个部署阶段前将策略权重加载到 vLLM 实例中：

从 vllm.model_executor 导入 set_random_seed 作为 vllm_set_random_seed

def init_vllm（model_id： str， device： str， seed： int， gpu_memory_utilization： float = 0.85）：
“”“ 开始推理过程，这里我们用 vLLM 在 GPU 上保存模型，与策略分开。

"""
vllm_set_random_seed（种子）

来自 TRL 的 Monkeypatch： # https://github.com/huggingface/trl/blob/ # 22759c820867c8659d00082ba8cf004e963873c1/trl/trainer/grpo_trainer.py # 打丁 vLLM 以确保我们能 # （1） 将 vLLM 模型放置在目标设备上（world_size_patch）和 # （2） 避免为我们设置（profiling_patch）设计的测试。

world_size_patch = patch（“torch.distributed.get_world_size”， return_value=1）
profiling_patch = patch（
“vllm.worker.worker.Worker._assert_memory_footprint_increased_during_profiling”，return_value=无），world_size_patch，profiling_patch：

返回 LLM（
model=model_id,
device=device， dtype=torch.bfloat16,
enable_prefix_caching=True,
gpu_memory_utilization=gpu_memory_utilization， ）

def load_policy_into_vllm_instance（策略：PreTrainedModel，LLM：LLM）：
“” 抄袭 https://github.com/huggingface/trl/blob/

22759c820867c8659d00082ba8cf004e963873c1/trl/trainer/grpo_trainer.py#L670。"""

state_dict = policy.state_dict（） llm_model =
llm.llm_engine.model_executor.driver_worker.model_runner.model
llm_model.load_weights（state_dict.items（））

你可能会发现记录训练和验证步骤的指标（这在后续强化学习实验中也会有用）。要在 wandb 中实现这一点，可以使用以下代码：

设置 wandb 指标
wandb.define_metric（“train_step”）# X 轴用于训练 wandb.define_metric
（“eval_step”）# X 轴用于评估
一切以火车开头的事物/都与 train_step 相连
wandb.define_metric（“列车/*”，step_metric=“train_step”）

所有以 eval 开头的事物/都与 eval_step 相连
wandb.define_metric（“eval/*”， step_metric=“eval_step”）

最后，我们建议你使用剪辑值 1.0 的渐变裁剪。

问题（sft_experiment）：在数学数据集上运行 SFT（2 点）（2 小时 100 小时）

---

1. 在推理 SFT 示例（提供于/data/a5-alignment/MATH/sft.jsonl 中）上运行 SFT，使用 Qwen 2.5 Math 1.5B 基础模型，变化 SFT 的唯一实例数量，使得

范围为{128， 256， 512， 1024}，同时使用完整数据集。调整学习率和批处理量，使使用完整数据集时至少达到 15%的验证准确率。

交付物：与不同数据集大小相关的验证准确率曲线。

2. 筛选推理 SFT 示例，只包含能产生正确答案的例子。对（完整）过滤后的数据集运行 SFT，报告过滤后的数据集大小和你实现的验证准确率。

交付物：报告数据集的规模和你实现的验证准确率曲线。
将你的发现与之前的 SFT 实验进行比较。

5 数学专家迭代

在前一节中，我们观察到通过过滤 SFT 数据中的不良样本，可以提升 SFT 模型的性能。在本节中，我们将更进一步：将该过滤程序应用于我们从基础模型生成的迹数推理。这一过程在文献中被称为专家迭代 [Anthony 等，2017]，在语言模型的语境下已有探讨，见 Cobbe 等人 [2021b]、Zelikman 等人 [2022]、Dohan 等人 [2022]、Gulcehre 等人 [2023]。

算法 2 专家迭代（EI）

输入初始政策模型π;奖励函数 R;任务问题 D

1：策略模型 π← π 2：对于 step = 1， ...， n_ei_steps 3：从 D 中抽样一批问题 4：设置旧的策略模型 π← π 5：样本 G 对每个问题 q 输出{o}~ π（· | q） ∈ D 6：计算每个抽样输出的奖励{r}，通过运行奖励函数 R（q， o） 7： 滤除错误输出（即 o，r= 0），以获得数据集 Dof 正确的问题-回答对 8： π← SFT（π， D）（算法 1） 9： 输出结束 π

接下来，我们将对 MATH 数据集进行专家迭代。
作为建议，你应该把一个 min_tokens 值传递给你的 vLLM SamplingParams，这样可以确保你不会生成空字符串（这可能会在下游造成 NaN，具体取决于你的实现）。这可以通过

```
sampling_min_tokens = 4
sampling_params = 采样参数（
    温度=sampling_temperature,
    max_tokens=sampling_max_tokens,
    min_tokens=sampling_min_tokens,
    n=G,
    seed=seed,
）
```

和 SFT 一样，你应该用剪辑值 1.0 的渐变裁剪。

问题（expert_iteration_experiment）：在 MATH 数据集上进行专家迭代（2 点）（6 小时 100 小时）

对 MATH 数据集进行专家迭代（提供地址为 /data/a5-alignment/MATH/train.jsonl）

采用 Qwen 2.5 Math 1.5B 基础模型，调整每题的展开次数 G 和 SFT 步骤中使用的历时数，并使用 n_ei_steps = 5。在{512， 1024， 2048}中，调整每个专家迭代步骤的批次大小（即 D 的大小）。（你不需要尝试所有可能的超参数组合。只要足够让大家对每个人做出结论就够了。）记录模型对训练反应的熵。确保 vLLM 在第二个答案标签处终止生成，就像 SFT 部分那样。
交付成果：与不同推广配置相关的验证准确率曲线。至少尝试两种不同的推广计数和跨度计数。

交付物：在 MAT 上验证准确率达到至少 15%的模型。
交付成果：简短两句话的讨论，比较你的 SFT 表现以及各阶段的绩效。

交付物：模型响应对训练的熵图。

6 政策梯度入门

jsonl）语言模型研究中的一个令人振奋的新发现是，针对具有强大基础模型的验证奖励进行强化学习，可以显著提升其推理能力和性能 [OpenAI 等，2024;DeepSeek-AI 等，2025]。最强的开放推理模型，如 DeepSeek R1 和 Kimi k1.5 [Team 等，2025]，是通过策略梯度训练的，这是一种强大的强化学习算法，可以优化任意奖励函数。
我们在下面简要介绍了语言模型上强化学习的政策梯度。我们的演讲紧密基于几本深入讲解这些概念的优秀资源：OpenAI 的《深度强化学习中的旋转》（Achiam，2018a）和 Nathan Lambert 的《来自人类反馈的强化学习（RLHF）书籍》（Lambert，2024）。

6.1 语言模型作为策略

参数为θ的因果语言模型（LM）定义了在当前文本前缀 s（状态/观察）下一个标记 a∈V 上的概率分布。在强化学习的语境中，我们认为下一个标记 a 是动作，当前文本前缀则是状态。因此，LM 是一种类别随机策略

$$a \sim \pi(\cdot \mid s), \qquad \pi(a \mid s) = \left[\operatorname{softmax}\left(f(s)\right)\right]_a . \tag{3}$$

在优化带有政策梯度的策略时，需要两种原始作：

1. 从策略中抽样：从上述类别分布中抽取一个动作;

2. 对动作的对数可能性进行评分：评估对数 π（a| s）。

通常，在使用大型语言模型进行强化学习时，sis 是目前产生的部分完成/解，每个 AI 是解的下一个令牌;当文本结束标记发出时，本集结束，如<|end_of_text|>，或者在我们的 r1_zero 提示中。

6.2 轨迹

（有限视界）轨迹是代理人经历的状态和动作的交错序列：

$$t = (s, a, s, a, \ldots, s, a), \tag{4}$$

其中 T 是轨迹长度，即 ais 是文本结尾的标记，或者我们已达到标记的最大生成预算。
初始态取自起始分布 s~ ρ（s）;在带有大型语言模型的强化学习中，ρ（s）是格式化提示词上的分布。在一般环境中，状态转移遵循某种环境

状态转变遵循某些环境动力学 s~ P（· | s， a）。在带有 LLM 的强化学习中，环境是确定性的：下一个状态是旧前缀与发射令牌串接的 s= s ∥ a。轨迹也称为剧集或展开;我们将交替使用这两个术语。

6.3 奖励与回报

标量奖励 r= R（s， a）判断状态 s 下所采取动作的即时质量。对于已验证域的强化学习，标准做法是为中间步骤分配零奖励，为终端动作分配已验证奖励

$$r = R(s, a) := \begin{cases} 1 & \text{如果轨迹 } S \parallel A \text{ 根据我们的奖励函数与地面真实值匹配} \\ 0 & \text{否则。} \end{cases}$$

回报 R（τ）汇总了沿轨迹的奖励。两种常见选择是有限视野无贴现收益

$$R(t) := \sum_{t=0}^{T} r, \tag{5}$$

以及无限视野贴现收益

$$R(t) := \sum_{t=0}^{\infty} gr, \qquad 0 < c < 1。 \tag{6}$$

在我们的情况下，我们将使用无折扣表述，因为剧集有一个自然的终止点（文本结束或最大生成长度）。

代理人的目标是最大化预期收益

$$J(\theta) = E[R(\tau)], \tag{7}$$

导致优化问题

$$\theta = \text{最大值}_{\theta} \; J(\theta)。 \tag{8}$$

6.4 原版政策梯度

接下来，我们尝试以梯度上升的方式学习策略参数θ，以期望收益为基础：

$$\theta = \theta + \alpha \nabla J(\theta)。 \tag{9}$$

我们将用来实现这一目标的核心身份是 REINFORCE 政策梯度，如下所示。

$$\nabla J(\pi) = E\left[\sum_{t=0} \nabla \log \pi(a|s) R(\tau)\right]. \tag{10}$$

推导政策梯度。我们是怎么得到这个公式的？为了完整性，我们将在下面给出该恒式的推导。我们将使用几个身份。

1. 轨迹的概率由 P（τ | θ） = ρ（s） 给出

$$\prod_{t=0}^{T} P(s| s, a) \pi(a| s)。 \tag{11}$$

因此，轨迹的对数概率为：

$$\log P(t | \theta) = \log \rho(s) + \sum_{t=0}^{T} \left[\log P(s| s, a) + \log \pi(a| s)\right]. \tag{12}$$

2. 对数导数技巧：

$$\nabla P = P\ \nabla \text{对数}\ P。 \tag{13}$$

3. 环境项在 θ 中为常数。ρ， P （· | ·） 和 R（τ） 不依赖于策略参数，因此

$$\nabla \rho = \nabla P = \nabla R（t） = 0。 \tag{14}$$

结合上述事实：

$$\nabla J（\theta） = \nabla E[R（\tau)] \tag{15}$$

$$= \nabla \sum_{\tau} P（t |\theta）R（t） \tag{16}$$

$$= \sum_{\tau} \nabla P（t |\theta）R（t） \tag{17}$$

$$= \sum_{\tau} P（t |\theta）\nabla \log P（t |\theta）R（t） \quad （对数导数技巧） \tag{18}$$

$$= E[\nabla \log P（\tau |\theta）R（\tau）], \tag{19}$$

因此，代入轨迹的对数概率并利用环境项在θ中恒定的事实，我们得到原版或 REINFORCE 策略梯度：

$$\nabla J（\pi） = E\left[\sum_{t=0} \nabla \log \pi（a|s）R（\tau） \right]. \tag{20}$$

直观上，这个梯度会增加轨迹中高回报动作的对数概率，否则会降低。
梯度样本估计。给定一批 N 个滚动输出 D = {τ }通过采样起始态 s~ ρ（s）收集，然后在环境中运行策略π，我们可以构建梯度的无偏估计量为

$$\hat{g} = \frac{1}{N} \sum_{我=1}^{N} \sum_{t=0}^{T} \nabla \log \pi（a| s）R（\tau）。 \tag{21}$$

该向量用于梯度上升更新 $\theta \leftarrow \theta + \alpha \hat{g}$。

6.5 政策梯度基线

普通政策梯度的主要问题是梯度估计的方差很大。一种常见的缓解方法是从奖励中减去一个仅依赖于状态的基线函数 b。这是一种控制变量[Ross， 2022]：其目的是通过减去与估计量相关的项来降低估计量的方差，而不引入偏倚。

我们将基线政策梯度定义为：

$$B = E\left[\sum_{t=0} \nabla 日志\ \pi（a|s）\big(R（\tau） - b（s）\big) \right]. \tag{22}$$

例如，一个合理的基准是策略值函数 V （s） = E[R（τ ）|s= s]，即从 s=s 开始并从策略π下计算时的期望收益。那么，量 （R（τ） − V （s）） 直观上表示实现轨迹比预期好多少。

只要基线仅依赖于各州，基线政策梯度是无偏的。我们可以通过重写基线政策梯度为

$$B = E\left[\sum_{t=0} \nabla \log \pi(a|s)\, R(\tau)\right] - E\left[\sum_{t=0} \nabla \log \pi(a|s)\, b(s)\right]. \tag{23}$$

聚焦基线项，我们看到

$$E\left[\sum_{t=0} \nabla \log \pi(a|s)\, b(s)\right] = \sum_{t=0}^{T} E\left[b(s)\, E[\nabla \log \pi(a|s)]\right]. \tag{24}$$

一般来说，得分函数的期望值为零：$E[\nabla \log P(x)] = 0$。因此，式 24 中的表达式为零，且

$$B = E\left[\sum_{t=0} \nabla \log \pi(a|s)\, R(\tau)\right] - 0 = \nabla J(\pi), \tag{25}$$

因此，我们得出基线政策梯度是无偏的。我们稍后将进行实验，看看基线化是否能提升下游性能。

关于保单梯度“损失”的说明。当我们在像 PyTorch 这样的框架中实现策略梯度方法时，会定义所谓的策略梯度丢失 pg_loss，使得调用 pg_loss.backward（）时，模型参数的梯度缓冲区会被我们的近似策略梯度 g 填充。在数学中，它可以表示为

$$\text{pg\_loss} = \frac{1}{N} \sum_{我=1}^{N} \sum_{t=0}^{T} \log \pi(a|s)\,(R(\tau) - b(s))。 \tag{26}$$

pg_loss 并不是典型意义上的损失——将列车或验证集的 pg_loss 报告为评估指标并不重要，而良好的验证 pg_loss 也不意味着我们的模型具有良好的泛化能力。pg_loss 实际上只是一个标量，使得当我们调用 pg_loss.backward（），通过反向推进得到的梯度就是近似的政策梯度 g。

做强化学习时，你应该始终记录并报告训练和验证奖励。这些是“有意义”的评估指标，也是我们试图通过政策梯度方法优化的指标。

6.6 非政策梯度

REINFORCE 是一个基于策略的算法：训练数据由我们正在优化的同一策略收集。为了理解这一点，我们写出 REINFORCE 算法：

1. 从当前政策π中抽取一批 {τ } 的部署。
2. 将政策梯度近似为 $\nabla J(\pi) \approx g = \sum_{我=1} \sum_{t=0} \nabla_\theta \log \pi(a|s)\, R(\tau)$。
3. 使用计算出的梯度更新策略参数：$\theta \leftarrow \theta + \alpha g$。我们需要做大量推断来采样一批新推出的方案，只用一步梯度。

LM 的行为通常不会在单一步内发生显著变化，因此这种策略内方法效率极低。

政策层次偏离政策。在非策略学习中，我们会从优化的政策以外的其他政策中抽样推广。流行策略梯度算法如 PPO 和 GRPO 的非策略变体利用前一版本的政策π展开来优化当前策略π。非政策梯度估计为

$$\hat{}\text{高夫政策} = \frac{1}{N}\sum_{\text{我}=1}^{N}\sum_{t=0}^{T}\frac{\pi(a|s)}{\pi(a|s)}\nabla\log\pi(a|s)\,R(\tau). \tag{27}$$

这看起来像是原版政策梯度的重要性抽样版本，并加了词的重权 $\frac{\pi(\text{st})}{\pi(\text{在}|\text{st})}$。
事实上，方程 27 可以通过重要性抽样和应用合理的近似推导，只要π和π差异不大：详见 Degris 等人[2013]。

7 群体相对策略优化

接下来，我们将介绍群体相对策略优化（Group Relative Policy Optimization，简称 GRPO），这是一种策略梯度的变体，您将在解决数学问题时实现并进行实验。

7.1 GRPO 算法

优势估计。GRPO 的核心思想是从策略π中抽取每个问题的多个输出，并用它们计算基线。这很方便，因为我们避免了学习神经价值函数 V（s）的需求，这在系统角度可能难以训练且繁琐。对于问题 q，组输出{o}~ π（·|q），设 r= R（q，o）为第 i 个输出的奖励。DeepSeekMath [Shao 等，2024] 和 DeepSeek R1 [DeepSeek-AI 等，2025] 计算第 i 个输出的组归一化奖励为

$$A = \frac{r - \text{平均值}(r,\ r,\ \ldots,\ r)\ \text{std}}{(r,\ r,\ \ldots,\ r) + \text{advantage\_eps}}, \tag{28}$$

其中 advantage_eps 是一个小常数，以防止被零除。注意，这个优势 A 对响应中的每个标记都是相同的，即 A= A，∀t 1 ∈，……，|o|，因此我们在下文中省略了 t 下标。

高级算法。在深入 GRPO 目标之前，先通过写出 Shao 等人[2024]在算法 3 中的算法来了解列车环路。

GRPO 目标。GRPO 的目标结合了三个理念：

1. 政策层次，如方程 27 所示。
2. 计算优势 A，如方程 28 所示的群归一化。
3. 一种裁剪机制，如近端策略优化（PPO， Schulman 等 [2017]）。

裁剪的目的是在单批滚动中进行多次梯度步时保持稳定性。
它通过保持政策π不偏离旧策略太远来发挥作用。

[3] 这是 DeepSeekMath 的 GRPO 的一个特殊案例，带有经过验证的奖励函数，没有 KL 项，也没有对参考和奖励模型进行迭代更新。

算法 3 组相对策略优化（GRPO）

输入初始政策模型π;奖励函数 R;任务问题 D 1：政策模型π← π

2：对于 step = 1，…，n_grpo_steps do
3：　一组题目示例 Dfrom D
4：　建立旧的政策模式，π← π
5：　样本 G 对每个问题 q 输出{o}~ ∈D 的每个问题 {o}~ π（· | q）
6：　计算每个采样输出 oby 运行奖励函数 R（q，o） 的奖励 {r}
7：　计算 A，并进行群归一化（方程 28）
8：　当列车步 = 1，…，n_train_steps_per_rollout_batch do
9：　　通过最大化 GRPO-Clip 目标来更新策略模型π（待讨论，方程 29）
10：　结束
11：结束

输出π

先写出完整的 GRPO-Clip 目标，然后我们可以直观地理解裁剪的作用：

$$J(\theta) = E\left[\frac{1}{G}\sum_{\text{我}=1}^{G}\frac{1}{|o|}\sum_{t=1}^{|o|}\underbrace{\text{明}\left(\frac{\pi(o|q,o)}{\pi(o|q,o)}\text{—A，卡}\left(\frac{\pi(o|q,o)}{\pi(o|q,o)}\text{—}, 1-\epsilon, 1+\epsilon\right)\text{A}\right)}_{\text{每个代币的目标}}\right]. \tag{29}$$

超参数ε >0 控制策略可更改的幅度。为此，我们可以按照 Achiam [2018a，b]以更直观的方式重写每个代币目标。定义函数

$$g(\varepsilon, A) = \begin{cases} (1+\epsilon)\text{艾夫 A}\geqslant 0 \\ (1-\epsilon)\text{A}<0\text{。} \end{cases} \tag{30}$$

我们可以将每个标记的目标重写为

$$\text{每个代币目标} = \text{最小值}\left(\frac{\pi(o|q,o)}{\pi(o|q,o)}\text{—A}, g(\varepsilon, A)\right)$$

我们现在可以根据案例进行推理。当优势 Ais 为正时，每个代币的目标简化为

$$\text{每个代币目标} = \text{最小值}\left(\frac{\pi(o|q,o)}{\pi(o|q,o)}\text{—}, 1+\epsilon\right)\text{一个。}$$

由于 A> 0，当作用 o 在 π 下变得更可能，即 π（o| q，o） 增加时，目标函数上升。最小值的裁剪限制了目标的增加幅度：一旦π（o| q，o） > （1 + ε）π（o| q，o），该每个标记目标达到最大值 （1 + ε）A。因此，该政策并不会被激励远离旧政策π。

类似地，当优势 Ais 为负时，模型试图向下推 π（o| q，o），但没有动力将其降至 （1 − ε）π（o| q，o）（完整论证参见 Achiam [2018b]）。

7.2 实现

现在我们已经对 GRPO 训练循环和目标有了高层次的理解，将开始实施其中的部分内容。SFT 和 EI 部分中实施的许多部件也将被 GRPO 重复使用。

计算优势（群体归一化奖励）。首先，我们将实现逻辑，计算每个示例在推广批次中的优势，即群体归一化奖励。我们将考虑两种可能获得群体归一化奖励的方法：上方 28 中提出的方法，以及一种近期简化的方法。

GRPO 博士[Liu 等，2025]强调，按标准化标准化会奖励答题正确度低的批次问题，这可能并不理想。他们建议简单地去除归一化步骤，计算

$$A = r - \text{均值}(r, r, \ldots, r)。 \tag{31}$$

我们将在作业后面实施这两种变体，并比较它们的表现。

问题（compute_group_normalized_rewards）：群归一化（2 分）

交付物：实现一种方法 compute_group_normalized_rewards 计算每个部署响应的原始奖励，在它们的组内进行归一化，并返回归一化和原始奖励，以及你认为有用的元数据。推荐使用以下接口：

```
def compute_group_normalized_rewards（
    reward_fn,
rollout_responses,
repeated_ground_truths,
group_size,
advantage_eps,
normalize_by_std, ）:
```

计算每个推广响应组的奖励，并按组规模进行归一化。

Args：

reward_fn：可调用[[str， str]， dict[str， float]] 对部署响应进行评分
这些是实地真相，产生带有“奖励”、“format_reward”和“answer_reward”键的口述。

rollout_responses：政策中的列表[str] 推广。该列表的长度为 rollout_batch_size = n_prompts_per_rollout_batch * group_size。

repeated_ground_truths：列表[str] 示例的真实情况。这段时间
列表 rollout_batch_size，因为每个例子的真实情况会重复 group_size 次。

group_size：int 每题回答数（组）。
advantage_eps：浮点小常数以避免归一化时除以零。
normalize_by_std：布尔 如果为真，除以每组标准差;否则只减去群均值。

返回：

双重[火炬。坦索，火把。张量，字数[str，float]]。
优势形状（rollout_batch_size, ）。每次推广响应都给予群体标准化奖励。

raw_rewards 形状（rollout_batch_size, ）。每次推广响应都有非规范化奖励。

元数据，你可以选择其他统计数据记录（例如平均值、标准日数、最大/最小奖励）。

要测试你的代码，实现[adapters.run_compute_group_normalized_rewards]。然后用 uv run pytest -k test_compute_group_normalized_rewards 运行测试，确保你的实现通过。

天真的政策梯度损失。接下来，我们将实现一些计算“损失”的方法。
提醒/免责声明，这些并非真正的标准损失，不应作为评估指标报告。在强化学习中，你应当跟踪列车和验证回报，以及其他指标（参见第 6.5 节讨论）。
我们将从朴素的政策梯度损失开始，它只是将优势乘以行动的对数概率（并抵消）。问题 q、响应 o 和响应令牌 o，简单每令牌的策略梯度损失为

$$-A \cdot \log p(o|q, o)。 \tag{32}$$

问题（compute_naive_policy_gradient_loss）：政策天真坡度（1 分）

交付成果：实现一种方法 compute_naive_policy_gradient_loss，利用原始奖励或预先计算的优势计算每个代币的策略梯度损失。推荐使用以下接口：

```
反 compute_naive_policy_gradient_loss（
raw_rewards_or_advantages：火把。张量，
policy_log_probs：火炬。张量，）->火炬。张肌：
```

计算每个代币的策略梯度损失，其中 raw_rewards_or_advantages 是原始奖励或已归一化的优势。

Args：

raw_rewards_or_advantages：火把。张量形状（batch_size，1），每个滚动响应的标量奖励/优势。

policy_log_probs：火把。张量形状（batch_size，sequence_length），每个标记的对数概率。

返回：

炬。张量形状（batch_size，sequence_length），每个令牌的策略梯度损失（将在训练循环中跨批处理和序列维度汇总）。

实施建议：

- 通过 sequence_length 维度广播 raw_rewards_or_advantages。

要测试你的代码，实现[adapters.run_compute_naive_policy_gradient_loss]。然后运行 uv run pytest -k test_compute_naive_policy_gradient_loss，确保测试通过。

GRPO-弹夹丢失。接下来，我们将实现更有趣的 GRPO-Clip 损失。

每个令牌的 GRPO-Clip 损失为

$$-\text{分钟}\left(\frac{\pi(o|q,\ o)}{\pi(o|q,\ o)}-A,\ \text{卡}\left(\frac{\pi(o|q,\ o)}{\pi(o|q,\ o)}-,\ 1-\epsilon,\ 1+\epsilon\right)\ A\right)\ . \tag{33}$$

问题（compute_grpo_clip_loss）：GRPO 扣子损失（2 分）

交付物：实现一种方法 compute_grpo_clip_loss 计算每个代币的 GRPO-Clip 损耗。推荐使用以下接口：

```
def compute_grpo_clip_loss（
优点：火炬。张量，policy_log_probs：火炬。张量，
old_log_probs：火炬。张量，剪裁范围：float,） -> 元组[火
炬。张量，力量，火炬。张量]]：
```

Args：

优点：火炬。张量形状（batch_size，1），示例优势 A。

policy_log_probs：火把。张量形状（batch_size，sequence_length），每个令牌的对数很可能来自被训练策略。

old_log_probs：火把。张量形状（batch_size，sequence_length），每个令牌的日志可能来自旧策略。

cliprange：float clip 参数ε（例如 0.2）。

返回：

双重[火炬。张量，力量，火炬。张量]]。

失去了火炬。形状张量（batch_size，sequence_length），每个代币截获的损失。

包含你想记录的内容的元数据 dict。我们建议记录每个代币是否被裁剪，即最小值右侧的截图策略梯度损失是否低于 LHS。

实施建议：

· 广播优势相较 sequence_length。

要测试你的代码，实现[adapters.run_compute_grpo_clip_loss]。然后运行 uv run pytest -k test_compute_grpo_clip_loss，确保测试通过。

保单梯度损失包装器。我们将进行三种不同版本的政策梯度进行减损：

（a） no_baseline：没有基线的天真政策梯度损失，即优势仅为原始奖励 A = R（q， o）。

（b） reinforce_with_baseline：天真的政策梯度损失，但利用我们的群体归一化奖励作为优势。如果 $\bar{r}$ 是来自 compute_group_normalized_rewards 的群归一化奖励（可能被群标准差归一化也可能不），则 A = $\bar{r}$。

（c） grpo_clip：GRPO 弹夹丢失。

为了方便，我们将实现一个包装器，方便在这三种保单梯度损失之间切换。

问题（compute_policy_gradient_loss）：策略梯度包装器（1 点）

交付物：实现 compute_policy_gradient_loss，一种便利包装程序，将分派到正确的损失例程（no_baseline、reinforce_with_baseline 或 grpo_clip），并返回每个代币的损失和任何辅助统计数据。推荐使用以下接口：

非议 compute_policy_gradient_loss（
policy_log_probs：火把。张量，loss_type：字面意思[“no_baseline”，
“reinforce_with_baseline”， “grpo_clip”]，raw_rewards：火炬。张量 |无 = 无，优势：火炬。张量 |无 = 无，old_log_probs：火炬。张量 |None = 无，cliprange： float |None = 无，） ->
元组[火炬。张量，力量，火炬。张量]]：

选择并计算所需的策略梯度损失。

Args：

policy_log_probs（batch_size，sequence_length），来自被训练策略的每个令牌日志概率。

loss_type “no_baseline”、“reinforce_with_baseline”或“grpo_clip”中的一个。
raw_rewards 如果 loss_type 则是必需的 == “no_baseline”;形状（batch_size，1）。
“reinforce_with_baseline”和“grpo_clip”所需的优势;形状（batch_size，1）。

old_log_probs “grpo_clip”的必修词句;形状（batch_size，sequence_length）。
“grpo_clip”需要的 cliprange;标量ε用于削波。

返回：

双重[火炬。张量，力量，火炬。张量]]。

损失（batch_size，sequence_length），每个代币的损失。
元数据 dict，底层例程的统计数据（例如，GRPO-Clip 的剪辑比例）。

实施建议：

- 委托给 compute_naive_policy_gradient_loss 或 compute_grpo_clip_loss。
- 执行参数检查（见上文断言模式）。
- 将返回的元数据汇总为单一语汇。

要测试你的代码，实现[adapters.run_compute_policy_gradient_loss]。然后运行 uv run pytest -k test_compute_policy_gradient_loss，验证是否通过。

戴着面具的凶狠。到目前为止，我们已经有了计算优势、对数概率、每个代币损失以及诸如每个代币熵和剪辑分数等有用统计数据的逻辑。为了将形状（batch_size， sequence_length）的每个标记损耗张量简化为一个损失向量（每个实例一个标量），我们

我们将计算序列维度上的损失平均值，但仅计算对应响应的指标（即掩蔽[i， j]==1 的标记位置）。
通过序列长度归一化在大多数大型语言模型的强化语言代码库中已是规范做法，但这并非正确做法——你可能会注意到，查看我们在（21）中关于策略梯度估计的陈述，没有归一化因子。我们将从这个标准原语开始，通常称为 masked_mean，但稍后会测试使用我们在 SFT 中实现的 masked_normalize 方法。
我们允许指定计算均值的维数，如果 dim 为无，则计算所有蒙蔽元素的平均值。这有助于获得响应标记、剪辑分数等的每个代币的平均熵。

问题（masked_mean）：蒙面弱势（1 分）

交付成果：实现一种方法 masked_mean，在尊重布尔掩码的同时平均张量元素。推荐使用以下接口：

def masked_mean（
张量：火炬。张量，面罩：火
炬。张量，调暗：int |无 =
无,） ->火炬。张肌：

计算给定维度上的张量均值，只考虑掩码 =1 的元素。

Args：

张量：火炬。张量：需要平均的数据。

面具：火炬。张量 与张量形状相同;均值为 1 的位置包含在均值中。
DIM：智力 |没有任何维度可以进行平均。如果无，则计算所有蒙蔽元素的平均值。

返回：

炬。坦索尔：蒙面意味着;形状与张量匹配。均值（微分）语义。

要测试你的代码，实现[adapters.run_masked_mean]。然后运行 uv run pytest -k test_masked_mean，确保通过。

GRPO 微批次列车步骤。现在我们准备为 GRPO 实现单一微批次列车步骤（回想一下，对于火车小批，我们会在 gradient_accumulation_steps >1 时遍历多个微批次）。
具体来说，鉴于原始奖励或优势和对数概率，我们将计算每个代币的损失，利用 masked_mean 为每个实例汇总到标量损失，对批次维度进行平均，调整梯度累积，然后进行反向传播。

问题（grpo_microbatch_train_step）：微批次列车步骤（3 点）

交付物：实现 GRPO 的单次微批量更新，包括策略梯度损失、掩码平均和梯度缩放。

推荐使用以下接口：

反 grpo_microbatch_train_step（
policy_log_probs：火把。张量，response_mask：火炬。张量，
gradient_accumulation_steps：int，loss_type：字面意义[“no_baseline”，
“reinforce_with_baseline”，“grpo_clip”]，raw_rewards：火炬。张量 |无 = 无，优势：火炬。张量 |无 = 无，old_log_probs：火炬。张量 |None = 无，cliprange： float |None = 无，） ->
元组[火炬。张量，力量，火炬。张量]]：

对微批次执行前后传递。

Args：

policy_log_probs（batch_size，sequence_length），来自被训练策略的每个令牌日志概率。

response_mask（batch_size，sequence_length），1 用于响应标记，0 用于提示/填充。

gradient_accumulation_steps 每个优化步骤的微批次数量。

loss_type “no_baseline”、“reinforce_with_baseline”、“grpo_clip”之一。

raw_rewards 需要 loss_type == “no_baseline”;形状（batch_size，1）。

当 loss_type ！ = “no_baseline”时需要优势;形状（batch_size，1）。
old_log_probs GRPO 夹必需;形状（batch_size，sequence_length）。

cliprange 剪辑参数ε用于 GRPO-Clip。

返回：

双重[火炬。张量，力量，火炬。张量]]。

标量张量损失。微批量损失，经过梯度积累调整。我们会返回这个数据以便记录。

包含底层损失调用的元数据，以及你可能想记录的其他统计数据。

实施建议：

· 你应该在这个函数中调用 loss.backward（）。一定要调整梯度积累。

要测试你的代码，实现[adapters.run_grpo_microbatch_train_step]。然后运行 uv run pytest -k test_grpo_microbatch_train_step，确认通过。

综合来看：GRPO 火车环线。现在我们将为 GRPO 组装一个完整的火车环线。你应参考第 7.1 节中的算法，了解整体结构，并在适当情况下使用我们实现的方法。
下面我们提供一些入门超参数。如果你的实现正确，应该能看到合理的效果。

n_grpo_steps：int = 200
learning_rate：float = 1e-5

advantage_eps：float = 1e-6
rollout_batch_size：int = 256
group_size：int = 8
sampling_temperature：float = 1.0
sampling_min_tokens：int = 4 # 与 Expiter 一样，禁止空字符串响应
sampling_max_tokens： int = 1024 epochs_per_rollout_batch： int = 1 # 策略启动
train_batch_size： int = 256 # 策略启动 gradient_accumulation_steps： int = 128 # 微批次大小为 2，可安装在 H100 上 gpu_memory_utilization： float = 0.85 loss_type： literal[

“no_baseline”,
“reinforce_with_baseline”,
“grpo_clip”,] =
“reinforce_with_baseline”
use_std_normalization： bool = True
Optimizer = torch.optim.AdamW（
policy.parameters（），
lr=learning_rate,
weight_decay=0.0,
betas=（0.9， 0.95），）

这些默认超参数会让你进入策略设置——每批推出时，我们采取一个梯度步骤。从超参数角度看，这意味着 train_batch_size 等于 rollout_ batch_size，epochs_per_rollout_batch 等于 1。
以下是一些合理性检查的断言和常数，应该能消除一些边缘情况，并为你指明正确的方向：

断言 train_batch_size % gradient_accumulation_steps == 0， （
“train_batch_size 必须被 gradient_accumulation_steps 整除”） micro_train_batch_size = train_batch_size // gradient_accumulation_steps 断言 rollout_batch_size% group_size == 0， （

“rollout_batch_size 必须被 group_size 整除”）
n_prompts_per_rollout_batch = rollout_batch_size // group_size 断言
train_batch_size >= group_size， （

“train_batch_size 必须大于或等于 group_size”）n_microbatches_per_rollout_batch = rollout_batch_size // micro_train_batch_size

这里还有一些额外建议： · 记得使用 r1_zero 提示，并指示 vLLM 在第二个答案标签处停止生成，就像之前的实验一样。

· 我们建议使用 typer 进行参数解析。
· 使用剪辑值 1.0 的渐变裁剪。
· 你应定期记录验证奖励（例如每 5 或 10 步记录一次）。你至少应该评估 1024 个验证示例用于比较超参数，因为 CoT/RL 评估可能存在噪声。
· 在我们实现损失时，GRPO-Clip 应仅在非策略时使用（因为它需要旧的对数概率）。 · 在非策略设置下，每个部署批次有多个梯度更新，重新计算每个时代的旧日志概率是浪费的。相反，我们可以计算旧的对数概率

每次都用一次，然后为每个时代重复使用。
· 你不应对旧的对数概率进行区分。 · 每次优化器更新时，您应记录以下部分或全部内容：– 损失。

——梯度范数。
——象征熵。
——剪辑比例，如果不合规。
– 训练奖励（总奖励、格式和答案）。
——还有其他你觉得对调试有用的东西吗?

问题（grpo_train_loop）：GRPO 火车环线（5 分）

交付物：实现 GRPO 的完整列车循环。开始培训 MATH 政策，确认你看到验证奖励在改善，同时随着时间推移，合理推出。提供一个图表，包含关于步骤验证奖励的情形，以及一些随时间推展的示例。

8 GRPO 实验

我们可以计算旧的对数概率，现在我们可以开始实验我们的 GRPO 列车循环，尝试不同的超参数和算法调整。每个实验需要两块 GPU，一块用于 vLLM 实例，一块用于策略。
注意提前停止跑步。如果你在 200 个 GRPO 步前发现超参数差异显著（例如配置偏离或明显次优），当然可以提前停止实验，节省时间和计算，方便后续运行。下面提到的 GPU 使用小时数只是粗略估计。

问题（grpo_learning_rate）：调整学习速度（2 分）（6 小时 100 小时）

从上述建议的超参数开始，对学习率进行扫描，并报告最终验证答案的奖励（如果优化器出现偏差，则报告出道）。
交付成果：与多重学习率相关的验证奖励曲线。
交付物：在 MAT 上验证准确率达到至少 25%的模型。
成果：简短两句话讨论你注意到的其他记录指标趋势。

对于其他实验，你可以使用上面扫描中表现最好的学习率。

基线的影响。继续上述超参数（除了你调优的学习率），我们现在将探讨基线化的影响。我们处于保单内的环境，因此将比较损失类型：

· no_baseline
· reinforce_with_baseline

注意，use_std_normalization 在默认超参数中为真。

问题（grpo_baselines）：基线调整的影响（2 分）（2 小时 100 小时）

与 reinforce_with_baseline 和 no_baseline 一起培训保单。
交付物：验证每种损失类型的奖励曲线。
成果：简短两句话讨论你注意到的其他记录指标趋势。

接下来的几个实验中，你应该使用上述实验中找到的最佳损失类型。

长度归一化。正如我们在实施 masked_mean 时所暗示的，在序列长度内平均损失既非必要也不正确。如何对损失进行加和是一个重要的超参数，这会导致政策行动中不同类型的信用归因。
让我们用 Lambert [2024]的例子来说明这一点。检查 GRPO 列车步骤，我们首先获取每个代币的政策梯度损失（暂且忽略裁剪）：

优势 #（batch_size，1）per_token_probability_ratios #（batch_size，
sequence_length）per_token_loss = -优势 * per_token_probability_ratios

我们已广播了序列长度的优势。让我们比较两种汇总每个代币损失的方法： · 我们实现的 masked_mean，对每个序列中未掩蔽的代币进行平均。 · 对每个序列中未掩蔽的标记求和，并除以常数标量（我们的 masked_normalize 方法支持 constant_normalizer ！ = 1.0）[Liu 等，2025，Yu 等，2025]。
我们考虑一个批次大小为 2 的例子，第一个响应有 4 个 token，第二个响应有 7 个 token。然后，我们可以看到这些归一化方法如何影响梯度。

your_utils 进口 masked_mean，masked_normalize

比值 = 火炬张量（[
[1， 1， 1， 1， 1， 1， 1，]，
[1， 1， 1， 1， 1， 1， 1，]，
]， requires_grad=真）

advs = 火炬张量（[
[2, 2, 2, 2, 2, 2, 2,], [2, 2, 2, 2,
2, 2, 2,],])

masks = torch.tensor（[
# 第一代：4 个代币
[1, 1, 1, 1, 0, 0, 0,],
第二代：7 个代币
[1, 1, 1, 1, 1, 1, 1,],])

# 每次接近都让自己变得正常
max_gen_len = 7 masked_mean_result = masked_mean（比率 * advs，掩模，dim=1） masked_normalize_result = masked_normalize（

比率 * ADVS，掩膜，DIM=1，constant_normalizer=max_gen_len）

print（“masked_mean”，masked_mean_result） print（“masked_normalize”，masked_normalize_result）

masked_mean 张量（[2.， 2.]， grad_fn=） # masked_normalize 张量（[1.1429, 2.0000]， grad_fn=）

masked_mean_result.mean（）.backward（） print（“ratio.grad”， ratio.grad）
ratio.grad：

度数：# 张量（[[0.2500， 0.2500， 0.2500， 0.2500， 0.0000， 0.0000， 0.0000]， #
[0.1429， 0.1429， 0.1429， 0.1429， 0.1429， 0.1429]]
ratio.grad.zero_（）

masked_normalize_result.mean（）.backward
（） print（“ratio.grad”， ratio.grad）
ratio.grad： # 张量（[[0.1429， 0.1429， 0.1429， 0.1429， 0.0000， 0.0000，
0.0000]， #
[0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429]])

问题（think_about_length_normalization）：考虑长度归一化（1 点）

交付成果：比较两种方法（尚未进行实验）。每种方法的优缺点是什么？有没有什么具体的场景或例子，哪种方法看起来更好？

现在，让我们从实证上比较 masked_mean 与 masked_normalize。

问题（grpo_length_normalization）：长度归一化的影响（2 分）（2 小时 100 小时）

交付物：将归一化与 masked_mean 和 masked_normalize 进行端到端 GRPO 训练进行比较。报告验证答案的奖励曲线。请对这些发现发表评论，包括任何有明显趋势的其他指标。

提示：考虑与稳定性相关的度量，如梯度范数。

以下实验采用性能更好的长度归一化方法。

带组标准差的归一化。回想一下我们标准的 compute_实现
group_normalized_rewards（基于 Shao 等人[2024]，DeepSeek-AI 等[2025]），我们以组标准差进行归一化。Liu 等人[2025]指出，除以组标准差可能会给训练过程引入不良偏差：标准差较低的问题（例如，所有奖励几乎全为 1 或 0 的过简单或过难题目）在训练中权重会更高。
Liu 等人 [2025] 建议去除标准差归一化，我们已在 compute_group_normalized_rewards 实现并进行测试。

问题（grpo_group_standard_deviation）：标准差归一化的影响（2 分）（2 小时 100 小时）

交付物：比较 use_std_normalization 的表现 == 真与 use_std_
归一化 == 错误。报告验证答案的奖励曲线。请对这些发现发表评论，包括任何有明显趋势的其他指标。

提示：考虑与稳定性相关的度量，如梯度范数。

以下实验采用性能更好的组归一化方法。

非政策与非政策。我们迄今为止实验的超参数都是策略上的：我们每个部署批次只采取一个梯度步骤，因此几乎完全采用了“原则性”

将政策梯度近似为 g（除上述长度和优势归一化选择外）。
虽然这种方法理论上合理且稳定，但效率低下。推广需要从保单中缓慢生成，因此是 GRPO 的主要成本;每批推广只采取单一步梯度似乎是浪费的，这可能不足以实质性改变政策的行为。
我们现在将尝试非策略训练，即每个部署批次采用多重梯度步骤（甚至多个阶段）。

问题（grpo_off_policy）：实施非政策的 GRPO

交付成果：实施非政策的 GRPO 培训。
根据你对上述完整 GRPO 列车环线的实现，你可能已经拥有实现这一目标的基础设施。如果不行，你需要实现以下功能： · 你应该能够为每个推出批次进行多个梯度步骤，其中每个推广批次的迭代和优化器更新数量由 rollout_batch_size、epochs_ per_rollout_batch 和 train_batch_size 控制。

· 编辑你的主训练循环，在每个部署批次生成阶段后、梯度步骤内循环之前，从策略中获取响应 logprobs——这些将是 old_log_probs。
我们建议使用 torch.inference_mode（）。
· 你应该使用“GRPO-Clip”损失类型。

现在我们可以利用每个部署批次的纪元数和优化器更新数量来控制我们是否偏离政策。

问题（grpo_off_policy_sweep）：偏离策略的 GRPO 超参数扫描（4 分）（12 小时 100 小时）

交付物：固定 rollout_batch_size=256，选择 epochs_per_rollout_范围内的区间
批次和 train_batch_size 要扫过。首先对有限的 GRPO 步数做广泛扫描（<50 步），以了解性能状况，然后对更多 GRPO 步数做更集中的扫视（200 步）。提供一份简短的实验日志，说明你选择的范围。
与你在策略内运行时 epochs_per_rollout_batch = 1 和 train_batch_size = 256 进行比较，报告图时基于验证步骤数以及墙时钟时间。
报告验证答案的奖励曲线。对研究结果发表评论，包括任何有明显趋势的指标，如熵和响应长度。将模型在训练后的反应熵与你在 EI 实验中观察到的情况进行比较。

提示：你需要更改 gradient_accumulation_steps 以保持内存使用恒定。

在非策略设置中消除削波。请记住，GRPO-Clip 中裁剪的目的是防止策略在单次推广批次中采取多级梯度步骤时，与旧策略过远。接下来，我们将在非策略设置中削减削波，以测试其实际必要程度。换句话说，我们将使用每个代币损失

$$-\frac{\pi(o|q,\ o)}{\pi(o|q,\ o)}\text{——一个。} \tag{34}$$

问题（grpo_off_policy_clip_ablation）：违规的 GRPO-夹子消融（2 分）（2 小时 100 小时）

交付物：将未剪辑的每个代币损失实现为一种新的损失类型“GRPO-No-Clip”。取前一问题中表现最佳的非策略超参数，运行未削减的损失版本。报告验证答案的奖励曲线。请评论与你 GRPO-Clip 运行的发现，包括任何有明显趋势的指标，如熵、响应长度和梯度范数。

提示效果。最后，我们将探讨一个令人惊讶的现象：强化学习中使用的提示词会对模型的性能产生显著影响，这取决于模型的预训练方式。

我们不再在 cs336_alignment/prompts/r1_zero.prompt 使用 R1-Zero 提示，而是在 cs336_alignment/prompts/question_only.prompt 使用一个非常简单的提示：

{问题}

你将用这个提示进行训练和验证，并将奖励函数（训练和验证中都用到）更改为 cs336_alignment/drgrpo_ grader.py 中的 question_only_reward_fn。

问题（grpo_prompt_ablation）：快速消融（2 分）（2 小时 100 小时）

交付物：报告 R1-Zero 提示和仅问题提示的验证答案奖励曲线。指标如何比较，包括其他有明显趋势的指标，如熵、响应长度和梯度规范？试着解释你的发现。

9 排行榜：GRPO 在数学上的排名

作为（必修）作业的最后部分，你将尝试在 2 个 H100 GPU 上训练 4 小时内获得最高验证奖励的方法。

型。我们将继续使用 Qwen 2.5 数学 1.5B 基础模型。

数据。我们将继续使用集群中 /data/a5-alignment/MATH/train.jsonl 和 /data/a5-alignment/MATH/validation.jsonl 上可用的 MATH 训练和验证数据集。你不允许使用其他数据，也不能对更强模型的推理链做 SFT 等。你必须使用上述抽样超参数（温度 1.0，最大 tokens 1024）报告整个验证集（全部 5K 示例）的验证准确率。你可以根据自己的意愿过滤列车集，或基于数据设计课程。你必须使用 R1-Zero 提示进行验证，验证过程中必须使用入门代码中提供的 r1_zero_reward_fn 奖励函数（如果你愿意，可以开发另一个奖励函数用于训练）。

算法。你可以自由调整超参数或完全更改训练算法，只要不使用多余数据或其他模型（如果愿意，可以使用更多模型副本）。

系统优化。你可能会注意到，在我们上面简单的 GRPO 实现中，至少有一块 GPU 始终处于空闲状态。通过改进我们管道的系统特性，您很可能会发现显著的改进。例如，你可以考虑用于部署或培训时使用较低精度，torch.compile 以及其他系统优化。你绝对不被限制只能把 vLLM 放在一个设备上，把列车策略放在另一个设备上，我们鼓励你思考更好的并行化方法。

想法。关于可能改进的建议，请参见以下仓库：

- veRL
- trl
- 火炬调
- 燕麦

关于吉隆坡分歧。我们还注意到，在上述实验中，我们没有包含针对某个参考模型的 KL 散度项（通常是冻结的 SFT 或预训练检查点）。在我们的实验以及文献中的其他研究[Liu 等，2025，NTT123,2025]中，我们发现省略 KL 项对性能没有影响，同时节省了 GPU 内存（无需存储参考模型）。不过，许多 GRPO 仓库默认包含它，鼓励你尝试 KL 或其他正则化形式，只要你使用 Qwen 2.5 Math 1.5B Base 或通过算法获得的模型。

问题（排行榜）：排行榜（16 分）（16 小时 100 小时）

交付物：报告在训练后 4 小时内对 2 块 H100 显卡获得的验证准确率，并截图显示你在墙时钟时间内的验证准确性，其中 x 轴在 4 小时结束≤。提醒一下，我们对您的评估设置了以下限制：

1. 你的验证准确率应为整个 MATH 验证集（全部为 5K 样本）的平均准确率。
2. 你必须在验证时使用 R1-Zero 提示。
3. 你必须使用温度 1.0 和 max tokens 1024 配合 vLLM 进行评估。
4. 你必须通过平均起始代码中提供的 r1_ zero_reward_fn 奖励函数产生的答案奖励来计算验证准确率。

10 结语

恭喜你完成了本课的最后一项作业！你应该为自己的努力感到自豪。希望你通过从零构建其主要组件，学习现代语言模型的基础，过程愉快。

引用

丹·亨德里克斯、科林·伯恩斯、索拉夫·卡达瓦斯、阿库尔·阿罗拉、史蒂文·巴萨特、唐恩、道恩·宋，以及雅各布·斯坦哈特。2021 年数学数据集测量数学问题解决能力。网址 https： //arxiv.org/abs/2103.03874。

深艾、郭大雅、杨德建、张浩伟、宋俊孝、张若玉、徐仁欣、朱启豪、马世荣、王佩义、肖碧、张晓康、余星凯、余武、吴志斌、邵志洪、李卓书、高子怡、刘爱欣、雪冰、王炳轩、吴伯补、北峰、陆承大、赵成刚、邓承起、张辰宇，重阮、戴大麦、陈德丽、季东杰、李二恒、林方云、戴福聪、罗富利、郝光波、陈关廷、李国伟、张浩、韩宝、徐汉玮、王浩成、丁洪辉、新华建、高华佐、曲慧、李慧、李建中、李嘉世、王嘉伟、陈景昌、袁景阳、邱俊杰、李俊龙、蔡志一， 倪嘉祺、建亮、陈金、开东、胡开歌、高凯歌、康官、黄克新、奎宇、王连、张乐聪、梁赵、王立通、张璃月、雷旭、夏蕾、张明华、唐明辉、李孟、王苗君、李明明、天宁天、黄潘潘、张鹏、王千成、陈秦宇、杜秋时， 葛瑞琦、张瑞松、潘瑞哲、王润吉、R. J.

陈、金 R.L.、陈如懿、陆上浩、周商言、陈山煌、叶盛峰、王世玉、于水平、周顺峰、潘舒廷、李 S.S.、周双、吴少庆、叶盛峰、陶云、田培、孙天宇、王鼎鼎曾、赵万家、刘温、梁文峰、高文军、于文琴、张文涛、W.L. Xiao， 魏安、刘晓东、王晓涵、陈晓康、聂晓涛、心成、刘欣、谢、刘星超、杨欣宇、李欣远、苏雪成、林旭恒、李晓月、陈晓沙、孙晓文、王晓祥、宋欣南、周周欣一、王先祖、单欣夏、李英俊、王英启、魏欣欣， 杨张、徐彦宏、姚莉、姚赵、孙耀峰、王耀辉、易宇、张一超、石一范、熊一梁、何颖、皮氏一石、王一松、谭一轩、马一阳、刘一元、郭永强、袁欧、王玉端、岳公、宇恒佐、何于佳、雄云凡、罗玉祥、刘玉轩、周宇阳、朱以熙， 徐彦洪、黄延平、李耀辉、一正、朱宇晨、马云仙、唐颖、扎玉坤、燕玉廷、任子志、任泽辉、沙张礼、傅哲、徐振、谢振达、张正延、郝哲文、马志成、颜志刚、吴志宇、顾子辉、朱子家、刘子君、李子林、谢子薇、宋紫阳、潘子正、黄振， 徐志鹏、张忠玉和张振。Deepseek-r1：通过强化学习激励 LLM 中的推理能力，2025 年。网址 https://arxiv.org/abs/2501.12948。

麦克斯韦·奈伊、安德斯·约翰·安德烈森、盖伊·古尔-阿里、亨里克·米哈莱夫斯基、雅各布·奥斯汀、大卫·比伯、大卫·多汉、艾托尔·莱夫科维奇、马尔滕·博斯马、大卫·卢安、查尔斯·萨顿和奥古斯都·奥德纳。

展示你的工作：2021 年《中级计算用语言模型的草稿板》。网址 https： //arxiv.org/abs/2112.00114。

杰森·魏、王学之、戴尔·舒尔曼斯、马尔滕·博斯玛、布莱恩·伊克特、菲夏、艾德·池、国乐和周周。2023 年，大型语言模型中的思维链引导引发推理。网址 https://arxiv.org/abs/2201.11903。

埃里克·泽利克曼、吴宇怀、穆杰西和诺亚·D·古德曼。《星：用推理自力更生》，2022 年。网址 https://arxiv.org/abs/2203.14465。

托马斯·安东尼、郑天和大卫·巴伯。深度学习与树状搜索的快速与慢思考，2017 年。网址 https://arxiv.org/abs/1705.08439。

OpenAI, :， 亚伦·贾赫、亚当·卡莱、亚当·勒勒、亚当·理查森、艾哈迈德·埃尔-基什基、艾登·洛、亚历克·赫利亚尔、亚历山大·马德里、亚历克斯·博伊特尔、亚历克斯·卡尼、亚历克斯·伊夫蒂米、亚历克斯·卡尔彭科、亚历克斯·塔查德·帕索斯、亚历山大·奈茨、亚历山大·普罗科菲耶夫、亚历山大·魏、艾莉森·谭、艾莉·贝内特、阿南娅·库马尔、安德烈·萨拉伊瓦、安德烈亚·瓦隆、安德鲁·杜伯斯坦、安德鲁·康德里奇、安德烈·米申科、安迪·阿普尔鲍姆、安吉拉·江、阿什文·奈尔， 巴雷特·佐夫、贝赫鲁兹·戈尔巴尼、本·罗森、本杰明·索科洛夫斯基、博阿兹·巴拉克、鲍勃·麦格鲁、鲍里斯·米奈耶夫、鲍陶·郝、鲍恩·贝克、布兰登·霍顿、布兰登·麦金齐、布赖登·伊斯特曼、卡米洛·卢加雷西、凯里·巴辛、凯里·哈德森、查克·明利、查尔斯·德·鲍西、切尔西·沃斯、陈申、张钟、克里斯·科赫、克里斯·奥辛格、克里斯托弗·赫斯、克劳迪娅·费舍尔、克莱夫·陈、丹·罗伯茨、丹尼尔·卡普勒， 丹尼尔·莱维、丹尼尔·塞尔萨姆、大卫·多汉、大卫·法尔希、大卫·梅利、大卫·罗宾逊、迪米特里斯·齐普拉斯、道格·李、德拉戈斯·奥普里卡、埃本·弗里曼、张艾迪、埃德蒙·黄，

伊丽莎白·普罗尔、张以诺、埃里克·米切尔、埃里克·华莱士、埃里克·里特、埃文·梅斯、王范、费利佩·彼得罗斯基·苏奇、菲利波·拉索、弗洛伦西亚·莱奥尼、福伊沃斯·齐姆普拉斯、弗朗西斯·宋、弗雷德·冯·洛曼、弗雷迪·苏利特、杰夫·萨尔蒙、贾姆巴蒂斯塔·帕拉斯坎多洛、吉尔达斯·沙博、格蕾丝·赵、格雷格·布罗克曼、纪尧姆·勒克莱尔、哈迪·萨尔曼、鲍海明、郝生、哈特·安德林、赫萨姆·巴格里内扎德、洪宇任、亨特·莱特曼， 钟炯、伊恩·基夫利昌、伊恩·奥康奈尔、伊恩·奥斯班德、伊格纳西·克拉韦拉·吉拉贝尔特、伊尔格·阿卡娅、伊利亚·科斯特里科夫、伊利亚·苏茨凯弗、伊琳娜·科夫曼、雅库布·帕乔茨基、詹姆斯·列侬、杰森·韦、让·哈布、杰瑞·特沃雷、冯家成、于嘉辉、翁佳义、唐杰、余杰奇、华金·基诺内罗·坎德拉、乔·帕勒莫、乔尔·帕里什、约翰内斯·海德克、约翰·霍尔曼、约翰·里佐、乔纳森·戈登、乔纳森·上里， 乔纳森·沃德、乔斯特·惠津加、朱莉·王、陈凯、凯·萧、卡兰·辛格尔、阮嘉琳、卡尔·科布、凯蒂·施、凯拉·伍德、肯德拉·林巴赫、顾伦·莱姆伯格、刘凯文、卢凯文、凯文·斯通、余凯文、拉玛·艾哈迈德、杨劳伦、刘利奥、莱昂·马克辛、何雷顿、利亚姆·费杜斯、翁莉莲、李登·李、林赛·麦考勒姆、林赛·赫尔德、洛伦茨·库恩、卢卡斯·康德拉丘克， 卢卡什·凯撒、卢克·梅茨、玛德琳·博伊德、玛雅·特雷巴茨、马纳斯·约格莱卡尔、马克·陈、马克·廷托尔、梅森·迈耶、马特·琼斯、马特·考弗、马克斯·施瓦泽、梅根·沙阿、穆罕默德·亚特巴兹、梅洛迪·Y。 关、许梦远、严蒙远、米娅·格雷斯、陈美安娜、迈克尔·兰佩、迈克尔·马莱克、王米歇尔、米歇尔·弗拉丁、迈克·麦克莱、米哈伊尔·帕夫洛夫、王迈尔斯、王明轩、米拉·穆拉蒂、莫·巴伐利亚、莫·罗哈尼内贾德、纳特·麦卡利斯、尼尔·乔杜里、尼尔·乔杜里、尼古拉斯·特扎克、诺姆·布朗、奥菲尔·纳胡姆、奥列格·博伊科、奥列格·穆克、奥利维亚·沃特金斯、帕特里克·赵、保罗·阿什伯恩、帕维尔·伊兹迈洛夫， 彼得·佐霍夫、瑞秋·迪亚斯、拉胡尔·阿罗拉、林兰德尔、拉法·贡蒂霍·洛佩斯、拉兹·高恩、宫良瑞亚、雷马尔·莱克、黄仁尼、节奏·加格、罗宾·布朗、罗山·詹姆斯、瑞伊·舒、瑞安·周、瑞安·格林、萨奇·贾因、萨姆·奥特曼、萨姆·托伊尔、萨姆·托耶、塞缪尔·米塞伦迪诺、桑迪尼·阿加瓦尔、圣地亚哥·埃尔南德斯、萨沙·贝克、斯科特·麦金尼、斯科蒂·颜、赵盛佳、胡盛利、希巴尼·桑图尔卡尔， 施拉曼·雷·乔杜里、张舒媛、傅思远、斯宾塞·帕佩、斯蒂芬·林、苏奇尔·巴拉吉、苏万什·桑吉夫、希蒙·西多尔、塔尔·布罗达、艾丹·克拉克、陶·王、泰勒·戈登、特德·桑德斯、特贾尔·帕特瓦尔丹、蒂博·索蒂奥、托马斯·德格里、托马斯·迪姆森、天浩·郑、蒂穆尔·加里波夫、汤姆·斯塔西、特拉皮特·班萨尔、特雷弗·克里奇、特洛伊·彼得森、蒂娜·埃伦杜、瓦莱丽·齐、维尼特·科萨拉朱、文尼·摩纳哥、维奇尔·庞， 弗拉德·福门科、郑伟一、文达·周、韦斯·麦凯布、沃伊切赫·扎伦巴、杨·杜波依斯、陆英海、陈毅宁、车永、余白、何宇晨、张宇晨、王云云、郑绍和李卓涵。Openai o1 系统卡，2024 年。网址 https://arxiv.org/abs/2412.16720。

金米队、安刚杜、高伯飞、博韦星、长九江、程辰、程立、晓晨军、杜辰庄、廖崇华、唐春宁、王从公、张德浩、袁恩明、陆恩哲、唐凤祥、洪松、光大魏、国昆、海青郭、韩朱、浩定、胡、浩阳、浩天姚、浩天赵、浩天赵、陆浩宇，李浩泽、于浩振、高洪成、郑华斌、欢远、贾陈、郭建行、苏建林、王建舟、赵杰、张金、刘景元、颜俊杰、吴君言、石立东、凌业、余龙辉、孟南东、张新、马宁晨、潘启伟、屈诚公、刘少微、马胜荣、魏舒鹏、曹思涵、黄思英、陶江， 高伟浩、熊卫民、何卫然、黄卫孝、吴文浩、何文阳、魏向辉、贾贤青、吴行哲、徐新然、祖新兴、周欣宇、潘学海、查理、李李、胡阳阳、刘杨阳、陈延如、王叶杰、刘一博、秦一道、刘一峰、杨颖、包益平、都宇伦、吴宇欣、王宇志、载达周， 王昭基、李赵伟、朱振、张正、王哲旭、杨志林、黄志奇、黄子浩、徐子耀和杨宗涵。Kimi k1.5：利用大型语言模型进行扩展强化学习，2025 年。网址 https://arxiv.org/abs/2501.12599。

理查德·S·萨顿、大卫·麦卡莱斯特、萨廷德·辛格和伊沙伊·曼苏尔。基于功能近似的策略梯度方法进行强化学习。载于 S. Solla、T. Leen 和 K. Müller 主编，《神经信息处理系统进展》，第 12 卷。麻省理工学院出版社，1999 年。网址 https://proceedings。

neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf。

抱着脸。Open r1：deepseek-r1 的完全开放复制品，2025 年 1 月。网址 https://github。com/huggingface/open-r1。

曾伟浩、黄宇真、刘钱、刘伟、何刻晴、马泽君和何君贤。简单动物园：

2025 年，《野外开放基模型的零强化学习研究与驯服》。网址 https： //arxiv.org/abs/2503.18892。

简单动物园：潘佳怡、张俊杰、王星耀、袁丽凡、庞浩、苏尔。 Tinyzero。
https://github.com/Jiayi-Pan/TinyZero，2025 年。访问时间：2025-01-24。

安杨、张贝琛、辉斌元、高伯飞、余博文、李承鹏、刘大一衡、屠建洪、周景仁、林君阳、陆克明、薛明凤、林润基、刘天宇、任行章和张振如。Qwen2.5-数学技术报告：通过自我提升迈向数学专家模型，2024 年。网址 https://arxiv.org/abs/2409.12122。

卡尔·科布、维尼特·科萨拉朱、穆罕默德·巴伐利亚、马克·陈、俊熙宇、卢卡什·凯撒、马蒂亚斯·普拉佩特、杰瑞·特沃雷克、雅各布·希尔顿、中野礼一郎、克里斯托弗·赫斯和约翰·舒尔曼。训练验证器以解决数学应用题，2021a。网址 https://arxiv.org/abs/2110.14168。

内森·兰伯特、雅各布·莫里森、瓦伦蒂娜·皮亚特金、黄胜义、哈米什·伊维森、法泽·布拉曼、莱斯特·詹姆斯·V·米兰达、刘艾丽莎、努哈·季里、谢恩·柳、顾玉玲、索米娅·马利克、维多利亚·格拉夫、杰娜·D。
黄、杨江江、罗南·勒布拉斯、奥伊文德·塔菲约德、克里斯·威廉姆、卢卡·索尔达尼、诺亚·A·史密斯、王毅中、普拉迪普·达西吉和汉娜内·哈吉希尔齐。图卢语 3：开拓开放语言模型培训后前沿，2025 年。网址 https://arxiv.org/abs/2411.15124。

刘子晨、陈长宇、李文军、齐鹏辉、庞天宇、超杜、李和林敏。
理解类 r1-zero 训练：批判视角，2025 年。网址 https://arxiv.org/abs/2503。
20783.

权宇锡、李卓涵、庄思远、盛颖、郑连民、宇浩、约瑟夫·E·冈萨雷斯、张浩和 Ion Stoica。与 pagedattention 合作的大型语言模型服务高效内存管理，2023 年。arXiv：2309.06180。

卡尔·科布、维尼特·科萨拉朱、穆罕默德·巴伐利亚、马克·陈、俊熙宇、卢卡什·凯撒、马蒂亚斯·普拉佩特、杰瑞·特沃雷克、雅各布·希尔顿、中野礼一郎、克里斯托弗·赫斯和约翰·舒尔曼。训练验证器解决数学应用题，2021b。arXiv：2110.14168。

大卫·多汉、徐温妮、艾托尔·列科维奇、雅各布·奥斯汀、大卫·比伯、拉斐尔·贡蒂霍·洛佩斯、吴宇怀、亨利克·米哈莱夫斯基、里夫·A·索鲁斯、雅沙·索尔-迪克斯坦、凯文·墨菲和查尔斯·萨顿。

语言模型级联，2022 年。网址 https://arxiv.org/abs/2207.10342。

卡格拉尔·古尔切赫雷、汤姆·勒潘恩、斯里瓦特桑·斯里尼瓦桑、克谢尼娅·科纽什科娃、洛特·韦尔茨、阿比谢克·夏尔马、阿迪提亚·西丹特、亚历克斯·艾亨、王苗森、顾晨杰、沃尔夫冈·马谢雷、阿尔诺·杜塞、奥尔汉
菲拉特和南多·德弗雷塔斯。强化自训（休息）用于语言建模，2023 年。网址 https： //arxiv.org/abs/2308.08998。

约书亚·阿基亚姆。深度强化学习中的加速。2018a。

内森·兰伯特。来自人类反馈的强化学习，2024 年。网址 https://rlhfbook.com。

谢尔顿·M·罗斯。模拟。学术出版社，2022 年。

托马斯·德格里斯、玛莎·怀特和理查德·S·萨顿。非政策演员评论家，2013 年。网址 https://arxiv。
组织/ABS/1205.4839。

邵志宏、王佩怡、朱启浩、徐润欣、宋俊晓、肖碧、张浩伟、张明川、李英凯、吴耀和郭大雅。Deepseekmath：推动开放语言模型中数学推理的极限，2024 年。网址 https://arxiv.org/abs/2402.03300。

约翰·舒尔曼、菲利普·沃尔斯基、普拉富拉·达里瓦尔、亚历克·拉德福德和奥列格·克利莫夫。近端策略优化算法，2017 年。网址 https://arxiv.org/abs/1707.06347。

约书亚·阿基亚姆。简化的 ppo-clip 目标，2018b。网址 1PDzn9RPvaXjJFZkGeapMHbHGiWWW20Eyhttps://drive.google.com/file/d/ /view。

于启英、郑张、朱若飞、袁宇风、晓辰左、于月、范田田、刘高宏、刘灵俊、刘欣、林海斌、林志奇、马博乐、光明盛、宇轩通、张赤、张莫凡、王章、韩竹、朱金华、陈嘉泽、陈江杰、王承义、于洪礼、戴维南、宋玉轩、魏祥鹏、浩周， 刘晶晶、马卫英、张雅琴、林燕、穆乔、吴永辉和王明轩。Dapo：一个大规模开源的大型语言模型强化学习系统，2025 年。网址 https://arxiv.org/abs/2503.14476。

NTT123。格拉-零。https://github.com/policy-gradient/GRPO-Zero，2025 年。访问时间：2025-05-22。