代码实现难点与解决方案

难点一：如何创建数据集的 yaml 文件

即使说明文档内有教程，但是依然需要通过测试才能得出对于新数据集，每个关键词后到底应该填写什么。再结合仓库内的 gsm8k 的 yaml 文件，我仿写出了可用于 MATH-500 的 yaml 文件。此时的文件虽然可用，但不够好。

难点二：如何创建加载数据集的脚本文件

官方的教程文档内，有一行字写着：This will assume that there is a loading script with the same name as the directory. 这意味着除了 yaml 文件，我们还需要写一个脚本文件来加载数据。这里我选择先运行实例中的 hellaswag，再提取 hellaswag 的脚本文件。根据提取的脚本文件，我仿写了一个可用于 MATH-500 的脚本文件。

难点三：模型的输出不完整

以上问题解决后，我们可以在本地运行 HugginFace 模型来测试 MATH-500。然而，DeepSeek-R1-Distill-Qwen-1.5B 的回答会中断。即，还没完成全部输出就已结束。经过一番测试，发现 HuggingFace 模型默认的输出长度被限制在 256 个 token。我通过更改 yaml 文件来延长至 8192 个 token。

难点四：LaTeX 等价判断

在测试 MATH-500 前几题时发现，模型输出的"(3, \frac{\pi}{2})"与答案的"\left(3, \frac{\pi}{2} \right)"实际等价，都为$(3,\frac{\pi}{2})$。但判断规则为字符串的全匹配，所以模型在这题被误判。为解决此问题，我参考了 MATH 论文的 GitHub 仓库，发现作者提供了一个 is_equiv 方法来解决 LaTeX 的等价问题。这里我在 lm-eval 的仓库中搜索同样的方法名，发现 hendrycks_math 任务使用了该方法且已被注册。借用该任务的 yaml 文件和 util.py 文件，我进一步完善了 MATH-500 任务的实现。

难点五：缺少算力

在 MATH-500 实现的测试通过后，我发现了算力缺失问题。由于本机的 GPU 只有 4GB 内存，所以最多只能运行不超过 2B 的 16-bit 模型或者 1B 的 32-bit 模型。这也是为什么最初选择半精度的 DeepSeek-R1-Distill-Qwen-1.5B 模型。但是，即使运行通过，运行时长也是问题。根据估计，需要约 16 小时才能跑完所有 500 题。考虑到课内学业也需要用到电脑，此方案被迫放弃。

如果使用在线算力，例如 Google Colab Pro，依然有需要掉线的风险。如果选择分段测试，保险方法是 10 题一次，一共需要 50 次。在有课内学业的情况下，这种方案的实际可行性也并不高。而实验室的算力也很紧张，近期申请不到算力。

综上，为了测试 MATH-500 我们只能调用 API 接口。

难点六：国内 API 不稳定性

在实现 DeepSeek 的接口后，发现每次运行会随机有一部分 query 丢失，即，模型返回为空。这可能是我在国外所以导致使用国内大模型接口会有网络上的一些问题。这里我换成 OpenAI API 测试。由于 gpt-4o-mini-2024-07-18 运行地快，且官方测试过该模型在 MATH-500 上的表现，我选择用该模型测试。发现没有 query 丢失，确认了是网络而不是代码实现问题。最终结果为 71.4%，和官方的 70.2%接近。

如何提升模型在 Math-500 上的表现

MATH-500 是从 MATH 数据集提取而成的测试集，想要提升模型在 MATH-500 上的表现，就约等于想办法提升模型在 MATH 上的表现。以下是我从数据和训练两方面的调研和思考。

数据增强(Data Augmentation):

MATH 作为一个相对早期的数据集，所以已经被很多模型用来训练过。因此，想要提升模型在 MATH 上的表现，直接将 MATH 用于训练很可能会因为过拟合(over-fitting)而适得其反。

如果要利用 MATH 数据集的题目，我们需要用数据增强的方法来基于 MATH 生成类似但不同的数据。相关研究包括：MATH-Perturb: Benchmarking LLMs' Math Reasoning Abilities against Hard Perturbations 提出了 simple 和 hard perturbation 方法来从旧题创建类似新题。经测试，主流模型在新题上的表现有所降低。我认为此方法不仅可以用来 benchmarking，也可以用来数据增强。如果模型能在 perturbation 之后的数据上的表现不减，这就意味着模型学会了解相关题目的方法，或者记住了所有 perturbation 后的题目。实际中更可能是两者皆有。但是如果增强的数据足够多，控制好训练的 epoch 数，模型会更倾向于学会解法而不是记住答案。

大模型训练方法：强化学习

对于大模型，去解决数学之类的推理问题，主流的 inference 方法是 scale up 测试时计算量(test-time compute)，并且先让模型先思考(输出 Chain-of-Thoughts，即 CoT)，再回答。为了支持这个 inference 范式，当下最前沿的训练方法是 DeepSeek-R1 使用的以强化学习为主，加以监督微调(SFT)的训练方法。首先，用 CoT 数据进行 Cold Start。然后用 GRPO 训练模型，奖励模型(reward model)使用基于规则的奖励，而且只在回答完毕后给予非零奖励，这也被称为 sparse-reward，和 OpenAI 先前提出的过程奖励模型(PRM)不同。

DeepSeek-R1 在强化学习之后加上了额外的一步：在 general-purpose 的数据集上的 SFT。如果只考虑模型在 MATH-500 上的表现，或者训练一个数学特化模型，最后的 SFT 可以省略。

小模型训练方法：蒸馏

大模型的强化学习方法未必能在小模型上同样适用。当下更可行的做法是蒸馏(distillation)，即让小模型来模仿大模型的输出。对于数学任务，直接用 SFT 蒸馏会有些局限性。因为 SFT 的 loss 使用了 log-likelihood，更侧重 token 间是否连贯、通顺。然而这在数学方面并不是最重要的。在数学和代码中，几个 token 对整体正确性有很大的影响。另外，只模仿输出往往不足以让小模型学会大模型的推理(reasoning)能力。

因此，前沿的数学任务蒸馏通常会选择多任务训练(multi-objective training)。除了大模型的输出本身被用来训练小模型，大模型的 CoT 等思考过程和最终答案的正确性也可以考虑用来训练。

使用 CoT 等思考过程参考：Distilling Mathematical Reasoning Capabilities into Small Language Models，Mixed Distillation Helps Smaller Language Model Better Reasoning

使用最终答案正确性参考：Improving Large Language Model Fine-tuning for Solving Math Problems