

总览

任务完成情况：

任务	完成情况	报告位置
收集主流模型在 Math-500 上的公开评测分数	已完成	见1. 数据收集与整理
在 EleutherAI LM Evaluation Harness 框架中，实现 Math-500 评测任务	已完成	见2. 评测代码实现
在至少 1 个主流模型上运行评测，确保实测分数与公开报告的差异尽可能小。并提供包含推理结果和评估结果的日志文件。	已完成，但是是基于opencompass评测框架完成的	见 3. 通过opencompass框架进行测评
记录开发过程中你遇到的问题、思考过程和解决方案	已完成	见2. 3. 1——2. 3. 3
分析实测分数与报告存在差异的可能原因	已完成	见 4. 和官方结果差距的分析
思考如何提升模型在 Math-500 上的表现	已完成	见5. 模型优化的方向

github链接：[lm-eval-math-500](#)

1. 数据收集与整理

模型	MATH 500 分数
OpenAI o1	94.8
OpenAI o1-mini	90
GPT-4o	74.6
DeepSeek-R1	97.3
DeepSeek-R1-32B	94.3
DeepSeek-R1-V3	90.2
DeepSeek-R1-Zero	95.9

模型	MATH 500 分数
deepseek-v2-lite-chat-hf	43.6
Kimi k1.5 long CoT	96.2
Kimi k1.5 short-CoT	94.6
Claude-3.5-Sonnet-1022	78.3
QVQ-72B-Preview	90.6

参考文档：

- [Kimi-k1.5 参考文档](#)
- [DeepSeek-R1 参考文档](#)
- [DeepSeek-V2 参考文档](#)

2. 评测代码实现

2.1 安装评测框架

```
git clone https://github.com/EleutherAI/lm-evaluation-harness
cd lm-evaluation-harness
pip install -e .
```

使用

```
lm-eval
```

命令，查看输入的格式。输出如下：

```
usage: lm-eval [-h] [--model MODEL] [--tasks task1,task2]
              [--model_args MODEL_ARGS] [--num_fewshot N]
              [--batch_size auto|auto:N|N] [--max_batch_size N]
              [--device DEVICE] [--output_path DIR|DIR/file.json]
              [--limit N|0<N<1] [--use_cache DIR]
              [--cache_requests {true,refresh,delete}]
              [--check_integrity] [--write_out] [--log_samples]
              [--system_instruction SYSTEM_INSTRUCTION]
              [--apply_chat_template [APPLY_CHAT_TEMPLATE]]
              [--fewshot_as_multiturn] [--show_config]
              [--include_path DIR] [--gen_kwargs GEN_KWARGS]
              [--verbosity CRITICAL|ERROR|WARNING|INFO|DEBUG]
              [--wandb_args WANDB_ARGS]
              [--hf_hub_log_args HF_HUB_LOG_ARGS] [--predict_only]
              [--seed SEED] [--trust_remote_code]
              [--confirm_run_unsafe_code]
```

2.2 建立评测任务

我打开了github上介绍这个框架能评测的任务的介绍：

https://github.com/EleutherAI/lm-evaluation-harness/tree/main/lm_eval/tasks

然后，我尝试寻找math500对应的评测任务，但是没有找到，只有找到gsm8k和mathqa的评测任务。这都不是math500。所以，我根据官方的指引：https://github.com/EleutherAI/lm-evaluation-harness/blob/main/docs/new_task_guide.md，自己建立了一个名为math500的评测任务，建立的过程如下：

2.2.1. 基本配置

分叉主存储库、克隆它、创建一个以您的任务名称命名的分支，然后在环境中安装项目要求：

```
# After forking...
git clone https://github.com/oddfnction0205/lm-evaluation-harness.git
cd lm-evaluation-harness
git checkout -b math500
pip install -e ".[dev]"
```

这几行命令的主要作用是将 Fork 后的项目克隆到本地，创建一个新的开发分支，并安装项目的开发依赖项，为后续的开发工作做好准备。

2.2.2. 创建 YAML 文件

要实现新的标准任务，我们需要编写一个 YAML 文件来配置我们的任务逻辑。我们首先创建一个新的空 YAML 文件。

```
mkdir -p lm_eval/tasks/math500
touch lm_eval/tasks/math500/math500.yaml
```

配置文件便创建完成了。

2.2.3. 选择并配置数据集

所有数据下载和管理均通过 HuggingFace (HF) datasetsAPI 处理。经过检查, MATH 500已经在huggingface里面了<https://huggingface.co/datasets/HuggingFaceH4/MATH-500>。所以可以顺利继续。

2.2.4. 实现配置文件

根据新任务创建的提示: https://github.com/EleutherAI/lm-evaluation-harness/blob/main/docs/new_task_guide.md, 我实现了math500.yaml文件。文件内容如下:

```
# 任务名称
task: math500
# 任务标签, 可根据实际情况添加
tag:
  - math
  - generation

# 数据集配置
dataset_path: HuggingFaceH4/MATH-500
dataset_name: null
dataset_kwargs: null
training_split: null
validation_split: null
test_split: null
fewshot_split: null

# 数据预处理函数
process_docs: !function utils.process_docs

# 提示模板
doc_to_text: "{{problem}}"
doc_to_target: "{{solution}}"

# 指标设置
metric_list:
  - metric: exact_match
    aggregation: mean
    higher_is_better: true

# 任务元数据, 包含版本信息
metadata:
  version: 0

# 任务别名, 用于美化展示
task_alias: MATH-500
```

该文件放置于lm_eval\tasks\math500文件夹下。

2.2.5. 实现数据集处理函数

根据提示, 我实现了数据集预处理函数 `utils.py`。函数如下:

```
import datasets

def process_docs(dataset: datasets.Dataset) -> datasets.Dataset:
    def _process_doc(doc):
        question = doc["problem"].strip()
        answer = doc["solution"].strip()
        out_doc = {
            "question": question,
            "answer": answer
        }
        return out_doc

    return dataset.map(_process_doc)
```

该文件放置于lm_eval\tasks\math500文件夹下。

2.3 调用OpenAI的API进行评测

根据lm_eval框架官网上的格式，我写出了评测的命令。在终端中执行以下命令：

```
export OPENAI_API_KEY=(替换成你的API)
```

```
lm_eval --model openai-completions \
  --model_args model=gpt-4o-mini-2024-07-18 \
  --tasks math500
```

2.3.1 解决无法直接从huggingface上下载数据的问题（已解决）

在测评过程中，我发现由于我的服务器没有挂梯子，导致无法从huggingface上下载math500进行测评。于是我在服务器上搭建了梯子。

参考教程：https://www.bilibili.com/video/BV1mJ1sYKEoK/?spm_id_from=333.337.search-card.all.click&vd_source=0cf0e7a265a93dd5f6a617cc12842466

开启代理的步骤：

```
cd clash
chmod +x clash
./clash -d .
```

出现类似的信息，说明梯子启动正常：

```
(base) root@autodl-container-e3534a912f-635cb8ff:/lm-evaluation-harness/clash# ./clash -d .
INFO[0000] Start initial compatible provider 故障转移
INFO[0000] Start initial compatible provider 自动选择
INFO[0000] Start initial compatible provider SDK DNS
INFO[0000] RESTful API listening at: 127.0.0.1:9090
INFO[0000] Mixed(http+socks) proxy listening at: [::]:7890
```

然后保持这个终端不要关闭，再开一个新的终端，执行以下命令：

```
export http_proxy="http://127.0.0.1:7890"  
export https_proxy="http://127.0.0.1:7890"
```

于是网络连接的问题就被顺利解决了。

2.3.2 解决配置文件test_split字段的参数问题（已解决）

接下来测评时，遇到了问题：

```
ValueError: Task dataset (path=HuggingFaceH4/MATH-500, name=None) must have valid or test docs!
```

这是因为配置要求test doc文件，而我之前配置文件test_split字段写为null。查询huggingface数据集后，我用test来填充test_split字段。修改配置文件如下：

```

# 任务名称
task: math500
# 任务标签，可根据实际情况添加
tag:
  - math
  - generation

# 数据集配置
# 保持原来的数据集名称
dataset_path: HuggingFaceH4/MATH-500
# dataset_name保持为null
dataset_name: null
# 指定本地数据集目录的路径
dataset_kwargs:
  data_dir: null
training_split: null
validation_split: null
test_split: test
fewshot_split: null # 如果不需要fewshot数据，也可以设置为null

# 数据预处理函数
process_docs: !function utils.process_docs

# 提示模板
doc_to_text: "{{problem}}"
doc_to_target: "{{solution}}"

# 指标设置
metric_list:
  - metric: exact_match
    aggregation: mean
    higher_is_better: true

# 任务元数据，包含版本信息
metadata:
  version: 0

# 任务别名，用于美化展示
task_alias: MATH-500

```

修改之后，评测便可以顺利进行。

2.3.3 API超过限额的问题（待解决）

但是在测评的时候，我遇到了API超过限额的问题：


```

^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/__main__.py", line 388, in cli_evaluate
    results = evaluator.simple_evaluate(
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/utils.py", line 422, in _wrapper
    return fn(*args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/evaluator.py", line 308, in simple_evaluate
    results = evaluate(
    ^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/utils.py", line 422, in _wrapper
    return fn(*args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/evaluator.py", line 528, in evaluate
    resps = getattr(lm, reqtype)(cloned_reqs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/models/api_models.py", line 623, in generate_until
    outputs = retry(
    ^^^^^
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 336, in wrapped_f
    return copy(f, *args, **kw)
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 475, in __call__
    do = self.iter(retry_state=retry_state)
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 376, in iter
    result = action(retry_state)
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 418, in exc_check
    raise retry_exc.reraise()
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 185, in reraise
    raise self.last_attempt.result()
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/concurrent/futures/_base.py", line 449, in result
    return self.__get_result()
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/root/miniconda3/lib/python3.12/concurrent/futures/_base.py", line 401, in __get_result
    raise self._exception
File "/root/miniconda3/lib/python3.12/site-packages/tenacity/__init__.py", line 478, in __call__
    result = fn(*args, **kwargs)
    ^^^^^^^^^^^^^^^^^^^^^^^^^
File "/lm-evaluation-harness/lm_eval/models/api_models.py", line 380, in model_call
    response.raise_for_status()
File "/root/miniconda3/lib/python3.12/site-packages/requests/models.py", line 1024, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 429 Client Error: Too Many Requests for url: https://api.openai.com/v1/completions
Requesting API: 0%|
(base) root@autodl-container-e3534a912f-635cb8ff:/lm-evaluation-harness#

```

报错显示"You exceeded your current quota, please check your plan and billing details."

这表示API已经超出了当前账户的配额，无法继续使用 OpenAI API。错误代码为 `insufficient_quota` 和 `Too Many Requests`，这可能有以下几个原因：

问题一. API 配额不足

我已经超出了免费的额度或者当前的 API 使用限制。

因为使用的是免费账户，可能会遇到配额限制，可以考虑升级到付费账户来获得更多的 API 请求次数和更高的配额。

问题二. 调用频率限制

错误信息中的 `Too Many Requests` 表示你在短时间内发送了太多的请求，超过了 API 的调用频率限制。这通常是因为每分钟、每小时或每天的调用次数有限制。

所以目前就进展到这一步。

2.3.3 可能的解决方法

对于API配额不足问题，可以考虑升级到付费账户来获得更多的 API 请求次数。

对于调用频率限制问题，我调研了openAI的文档，发现可以考虑采用指数回退重试的调用策略。

官方文档链接：

<https://openai.xiniushu.com/docs/guides/rate-limits>

3. 通过opencompass框架进行测评

因为OpenAI API限额和账户付费的问题，所以最终没有在GPT o1 mini上完成。为了完成“在至少 1 个主流模型上运行评测，确保实测分数与公开报告的差异尽可能小。并提供包含推理结果和评估结果的日志文件。”这一任务，我进行了替代方案的查找。

在调研中，我发现用opencompass框架来进行测评会方便很多。

框架链接：

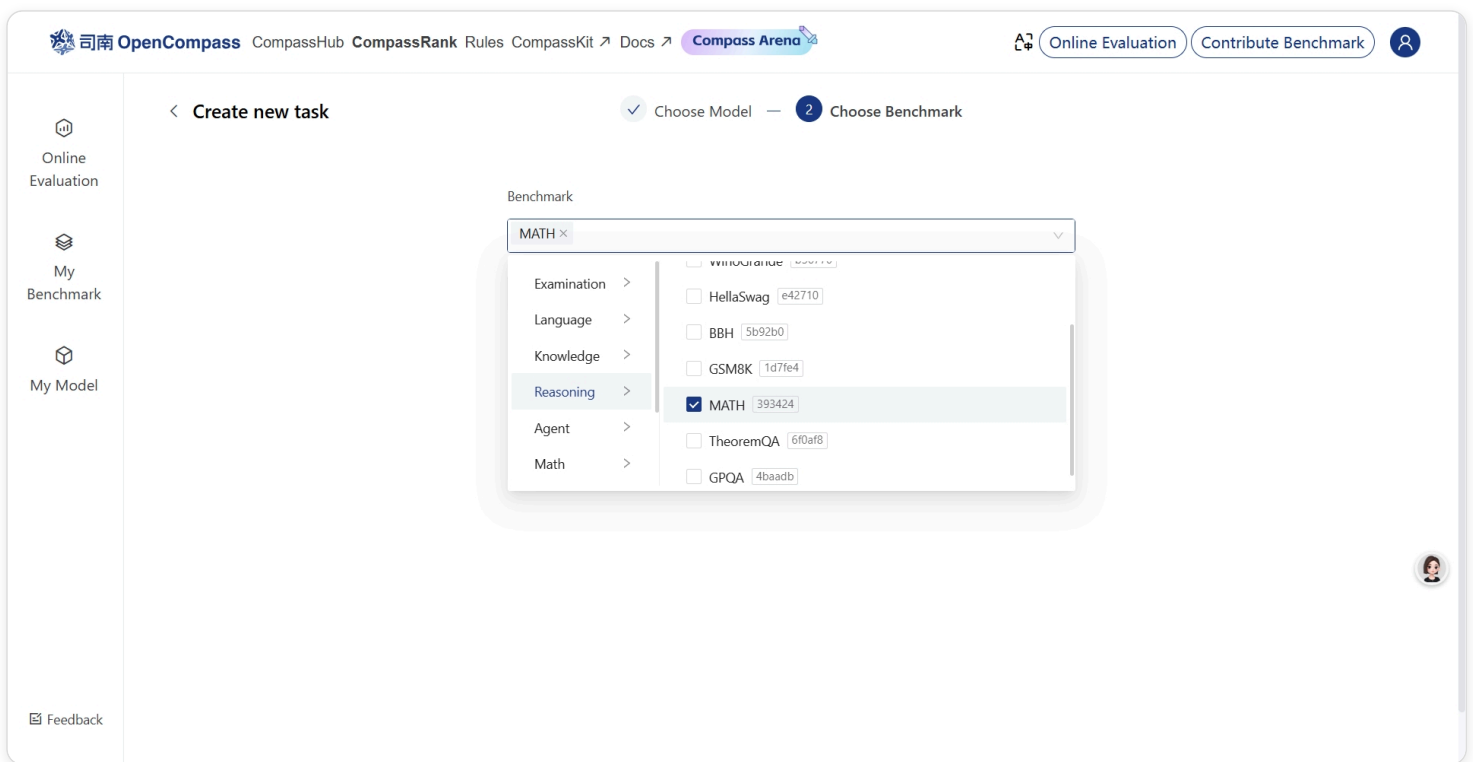
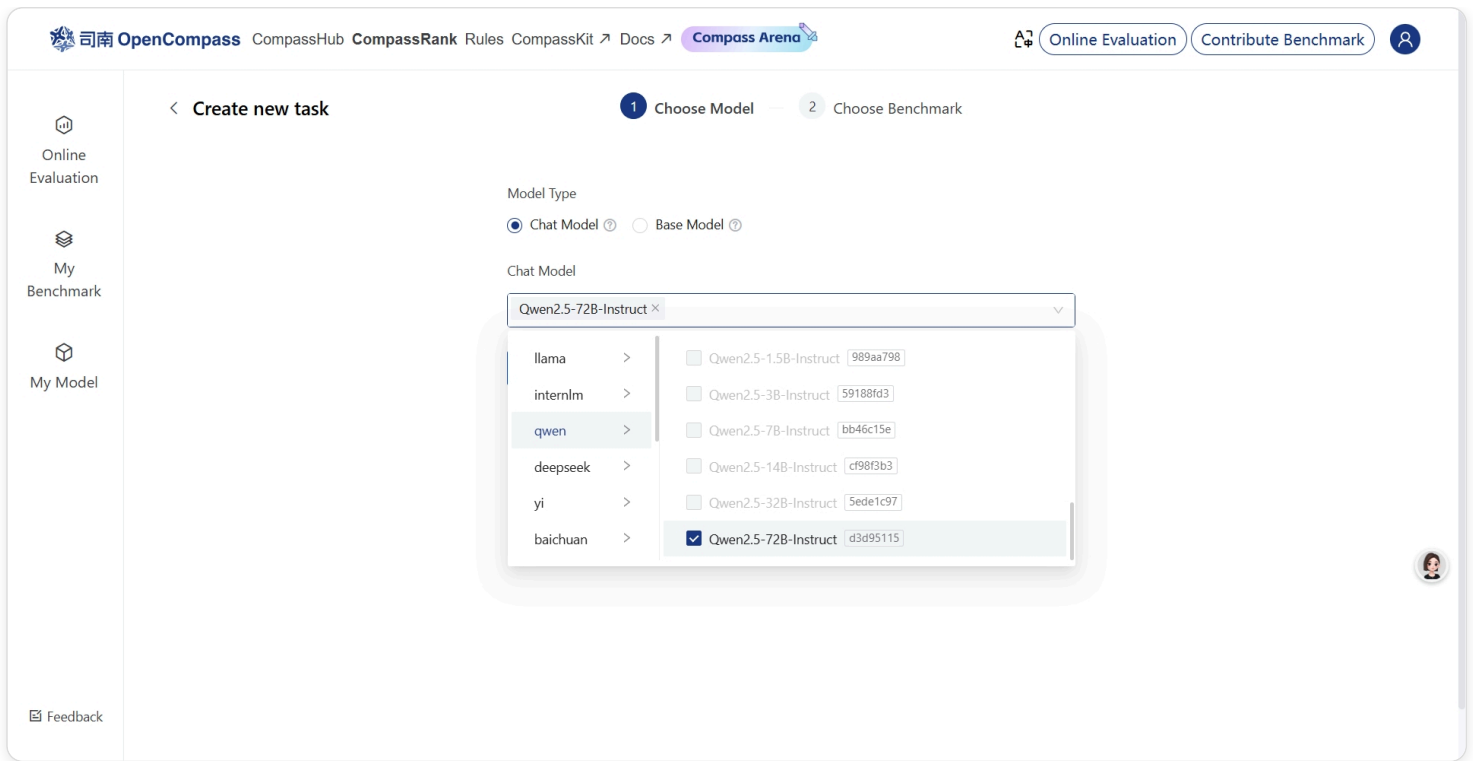
<https://github.com/open-compass/opencompass?tab=readme-ov-file>

这个框架测评十分方便。几乎跟着github操作一遍就可以成功。

此外，这个框架还支持在线测评。在线网址：

<https://rank.opencompass.org.cn/create/quick>

在这个网站里，可以直接选择模型和数据集进行评测。以下是我测试qwen2.5-72b-instruct-hf在MATH数据集上表现的过程。



通过Opencompass测试平台，qwen2.5-72b-instruct-hf在MATH数据集上的得分为：75.16
以下是测评的详细信息：

司南 OpenCompass

CompassHub

CompassRank

Rules

CompassKit

Docs

Compass Arena

Online Evaluation

Contribute Benchmark

评测任务-2025-02-26 23:40:10

2025-02-26 23:40:10

Model

Qwen2.5-72B-Instruct

d3d95115

Benchmark

Reasoning

MATH

Online Evaluation

My Benchmark

My Model

Feedback

Evaluation Result

Download Result

Download Wrong Answer

Result Overview

math

Dataset	Setting	Metric	Mode	qwen2.5-72b-instruct-hf
math 393424	4-shot	accuracy	gen	75.16

Dataset	Version	Metric	Mode	Model	Accuracy
math	393424	accuracy	gen	qwen2.5-72b-instruct-hf	75.16

测评错误题目见附件：评测错误题目-20250110_150616.zip

4. 和官方结果差距的分析

对结果进行分析：qwen2.5-72b-instruct-hf 在MATH数据集上的评测指标为75.16，而官方文档中给出的结果为90.6，两者之间存在较大的差距。为了更好地理解和分析这一差距，我们需要从多个角度来探讨可能的原因。

为什么复现不了官方模型的结果？

在机器学习和大规模语言模型（LLM）的评测过程中，复现官方结果的难度较大，特别是在一些具有高复杂度的任务上。导致差距的原因有很多，以下是几个可能的因素：

1. LLM评估的复杂性

- LLM模型的评估通常是一个非常复杂的过程，涉及多个参数的设置，而这些参数对最终的评测结果会产生深远影响。以下是一些可能的影响因素：
- Filter设置：不同的模型评测环境可能采用了不同的filter设置，导致评测过程中不符合条件的样本被过滤掉。即使是微小的差异，也可能影响到整体评测结果。

提示词（Prompt）设计：不同的提示词会影响模型的生成结果。官方和我们在复现过程中，使用的提示词可能有所不同，这可能导致模型的推理方式和最终输出有所差异。

是否使用CoT (Chain of Thought) : CoT是一种常用于增强推理过程的技巧，在某些任务中，CoT能够显著提升模型的表现。如果官方实现使用了CoT而我们没有使用，可能导致复现结果的差距。

Zero-shot或N-shot设置: Zero-shot和N-shot学习方式对模型的性能有很大影响。如果官方实现使用了更多的N-shot样本，而我们采用了Zero-shot设置，也可能导致评测结果的差异。

Batch-size: Batch-size的大小直接影响模型在推理过程中处理的样本数量，进而影响计算效率和结果的稳定性。在不同的batch-size设置下，模型的表现可能会有所不同。

2. 如何发现差异

参考官方实现: 在复现模型时，首先需要尽可能参考官方实现的方式，尽量使得我们的设置与官方一致。即使无法做到完全一致，接近官方设置也是一种常见的做法。

与官方结果的比较: 我们可以通过逐步调整上述参数，来比较不同设置下的评测结果，以找出造成差距的具体原因。

3. 推荐做法

重新计算基线 (Baseline) 结果: 在进行模型微调之前，建议基于自己设置的环境和参数重新计算baseline的结果。这样可以确保自己在评测过程中对模型的理解和设置有清晰的掌握，而不是直接将官方的评测结果作为对照。通过这种方式，我们可以更好地理解模型在不同环境下的表现差异。

调试和优化过程: 在重新计算基线后，调试和优化过程中要保持灵活性，特别是在调整各种超参数（如学习率、batch-size、CoT设置等）时，要注意观察每个变化对最终结果的影响。可能会发现一些细节上的差异是导致性能差距的关键。

5. 模型优化方向

5.1 强化学习

5.1.1 WizardMath (2023-08)

这篇论文从Evol-Instruct反馈中学习的强化学习 (Reinforcement Learning from Evol-Instruct Feedback, RLEIF) 进行微调，以增强其在数学问题解决方面的表现。

在方法论上，RLEIF方法包含三个关键步骤：首先是监督式微调 (Supervised Fine-Tuning)，利用少量样本重新生成GSM8k和MATH数据集中的答案，并与原始数学指导数据混合，以提高模型对指令的遵循能力；其次是训练指令奖励模型 (Instruction Reward Model, IRM) 和过程监督奖励模型 (Process-supervised Reward Model, PRM)，IRM用于评估进化指令的质量，而PRM则为解决方案的每一步提供反馈；最后是主动Evol-Instruct和近端策略优化 (Proximal Policy Optimization, PPO) 训练，通过增加数据量和利用奖励模型生成最终的指令奖励和答案奖励。

论文: [WizardMath论文链接](#)

5.1.2 RFT (Rejection Sampling Fine-Tuning) :

RFT 是一种更为复杂的微调方法，主要步骤如下：

- 数据生成:** 首先使用预训练模型生成大量的候选输出。
- 筛选过程:** 通过某种筛选机制（如人工评审或自动评分系统）从这些候选输出中挑选出高质量的样本。

3. **模型训练**：使用筛选后的高质量样本对模型进行微调。

RFT的关键：

- 筛选过程可以显著提高数据质量，从而提升模型的性能
- 筛选机制可以是人工的，也可以是基于某种自动化评分系统的

如何将RFT用于数学推理任务？

RFT 的核心思想是利用已有的监督模型来生成新的数据样本，如果将其用于数学推理任务，可以通过选择正确的推理路径来增强模型的训练数据集。

步骤：

1. **生成候选推理路径**：使用一个已经通过监督微调（SFT）训练好的模型来生成针对训练集中每个问题的多个候选推理路径。这些路径包括一系列计算步骤，旨在解决问题。
2. **筛选正确路径**：从生成的候选路径中筛选出那些能够正确推导出问题答案的推理路径。
3. **去重和多样化**：进一步从筛选出的正确路径中选择具有不同计算过程或表达方式的路径，以增加数据集的多样性。这有助于模型学习不同的解决问题的方法。
4. **微调**：使用这些经过筛选和去重的推理路径作为新的训练数据，对原始的监督模型进行进一步的微调。
5. **提高泛化能力**：通过引入多样化的推理路径，RFT 旨在提高模型在未见过的问题上的泛化能力。

优势：

- 通过引入模型生成的数据来增强其推理能力
- 避免了昂贵的人工标注成本
- 特别适用于那些难以通过增加监督数据量来提升性能的场景，因为它允许模型从未充分利用的训练数据中学习新的推理策略

5.2 依靠CoT解决数学问题

采用**Chain-of-Thought (CoT)** 方法可以显著提升大语言模型在数学推理方面的能力。CoT的核心思想是通过在推理过程中显式地展示推理步骤，帮助模型更好地理解问题的结构，进而提高其推理和解题能力。

CoT的基本概念

Chain-of-Thought (CoT) 指的是一种推理策略，要求模型在给出最终答案之前，逐步展示其思考过程。通过分步推理，模型能够在处理复杂问题时更好地模拟人类的思考方式。这个方法尤其适用于需要多步骤推理的任务，例如数学题、逻辑推理等。

如何用CoT提升大模型的数学推理能力？

1. 训练模型进行多步推理：

在训练过程中，使用带有明确推理步骤的数学问题来训练大模型，而不仅仅是直接给出答案。这样，模型可以学会如何分解问题并逐步思考。具体来说，可以通过以下方式实现：

提供明确的推理步骤：例如，在数学题中，不仅给出最终答案，还需要让模型列出每一步的计算过程或推理路径。

CoT训练数据：使用具有推理步骤的数学问题作为训练数据集。这些数据集可以是人类手动标注的，或者通过模拟生成。

2. 引导模型进行推理：

通过设计合适的提示词或prompt来引导模型逐步推理。给模型提供一个框架，让它知道在回答时需要分步骤地思考。比如：

引导推理模板：给模型一个明确的框架来展示它的思考过程。示例模板如下：

题目：计算 $24 \div 8$

思考过程：

- 第一步：首先要知道， $24 \div 8$ 就是问 8 可以放进 24 几次。
- 第二步：24 除以 8 得到 3。
- 答案：3

利用自然语言提示：用自然语言描述推理过程，使模型能够更好地理解如何分解问题。例如，提示模型：“请逐步思考，首先算出XX，然后进行YY，最后得出答案。”

3. 使用CoT技巧解决多步骤数学问题：

数学问题，尤其是复杂的推理题目，往往需要多步骤的计算。在这种情况下，CoT可以通过以下方式帮助：

逐步分解：对于多步骤的问题，CoT允许模型逐步分解每一步的计算过程，减少一开始就做出错误推断的风险。

明确的中间状态：CoT方法有助于模型保持在每一步的正确路径上，减少错误的累积，因为每一步都可以有明确的验证和修改。

4. 增强模型的自我校验能力：

CoT方法通过分步推理使得模型的每个推理步骤都能够自我校验。当模型在某一步出现错误时，可以在下一步发现并纠正错误。这样，模型不仅仅是直接给出答案，而是能够通过逐步推理不断接近正确答案。

5. 提高模型的推理透明度：

通过CoT，模型的推理过程变得更加透明，用户可以看到每一步的推理路径，帮助理解模型如何得出结论。这对于一些数学问题（尤其是复杂的推理和计算）尤其重要，能帮助用户分析模型的潜在错误并进行修正。

6. 训练包含反向推理的任务：

除了正向推理（从题目到答案），反向推理（从答案推导问题）也可以通过CoT方法来提升模型的数学能力。例如，在给定一个数学结果时，模型需要推导出正确的问题及其解决步骤。这种能力对于数学推理和复杂题目解答尤为关键。

CoT对数学推理的优势

1. 帮助模型避免推理错误：

在每一步推理中，模型有机会检查之前的思考，减少错误的发生。特别是在解决复杂的数学问题时，CoT可以有效避免过早的错误影响整个推理过程。

2. 提升模型的解释能力：

通过显示每个推理步骤，CoT方法能让模型的推理过程更加透明，用户可以清楚地看到模型是如何得出结论的。

3. 适应更多类型的数学题目：

许多数学问题需要模型进行多步推理。通过CoT，模型能够逐步进行运算或推理，从而适应更复杂的题目。

总结

采用Chain-of-Thought (CoT) 方法提升大模型的数学推理能力，关键在于让模型明确地展示其推理过程，逐步推进问题的求解，而不仅仅给出最终结果。通过训练模型使用分步推理，结合正确的提示和推理框架，可以有效地提升模型在数学推理、逻辑推理等任务中的表现。

5.3 结合外部工具

5.3.1 Tool-integrated Reasoning Agents (2023-09) [4]

本文介绍了ToRA (Tool-Integrated Reasoning Agents)，这是一系列专为解决复杂数学问题而设计的微调大型语言模型。ToRA的核心思想是将自然语言推理与外部工具（如计算库和符号求解器）的使用无缝整合，以此结合语言的分析能力和工具的计算效率。通过这种集成方法，ToRA在多个数学推理数据集上取得了显著的性能提升。

论文: [Tool-integrated Reasoning Agents](#)

5.3.2 GPT-4 Code Interpreter (2023-07) [6]

文章指出现有的GPT-4 Code Interpreter在处理数学问题时表现出色，这得益于其生成和执行代码的能力，以及在收到不合理输出时调整解决方案的能力。基于此，研究者提出了一种名为显式基于代码的自我验证 (CSV) 的新颖方法，该方法通过零样本提示鼓励模型使用代码自我验证答案。当验证状态为“False”时，模型会自动修正其解决方案，类似于数学考试中纠正错误的过程。

5.4 扩大微调数据集

5.4.1 OpenMathInstruct (2024-02)

OpenMathInstruct-1是一个由NVIDIA团队开发的数学指令微调数据集，它包含了1.8百万个数学问题及其解决方案对，目的是提升开源大型语言模型 (LLMs) 在数学推理方面的能力。该数据集利用了最新发布的、具有宽松许可协议的Mixtral模型，该模型在数学推理任务上表现出色。

项目团队通过创新的提示方法和扩展策略，合成了GSM8K和MATH两个数学推理基准测试的解决方案，以此来构建数据集。他们开发的最佳模型OpenMath-CodeLlama-70B在GSM8K和MATH基准测试上取得了与顶尖gpt-distilled模型相媲美的成绩。

在方法论上，团队采用了few-shot prompting技术来合成解决方案，并通过多种提示策略来提高训练集的覆盖率。特别是，他们发现使用掩码文本解决方案的提示策略能显著提升MATH基准测试的覆盖率。

论文: [OpenMathInstruct](#)