PR #23864 完整报告

sgl-project/sglang

[Bench] fix MMMU answer-extraction regex dropping multi-line responses

合并时间：2026-04-29 14:48

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23864

执行摘要

- 一句话：修复 MMMU 评估正则截断多行答案
- 推荐动作：该 PR 值得精读，特别是 _parse_explicit_multi_choice_answer 中正则优先级的设计和测试中模块桩的隔离技巧。建议在后续类似评估脚本中参考此模式。

功能与动机

MMMU 评估中，模型返回的多行响应因默认正则 (.*) 不匹配换行而被截断，导致 process_result 收到的文本不完整，评估分数被低估（PR body 指出修复后正确率从 0.5933 提升至 0.7433）。同时，旧解析逻辑在模型先列出 (A)/(B)/(C)/(D) 选项再给出最终答案时容易命中错误选项。

实现拆解

1. 修改默认正则表达式：在 benchmark/mmmu/eval_utils.py 的 EvalArgs 类中，将 response_answer_regex 从 "(.*)" 改为 "(?s)(.*)"，启用 DOTALL 标志，使正则跨行捕获完整响应。
2. 新增显式答案解析函数：在 eval_utils.py 中添加 _parse_explicit_multi_choice_answer(response, all_choices)，优先通过 answer: 标记或行尾独立字母模式提取明确选项，返回 None 时再走旧逻辑。
3. 集成到解析入口：在 parse_multi_choice_response 函数开头调用新函数，若得到非 None 结果则提前返回，否则继续执行原候选扫描逻辑。
4. 添加 CPU 单元测试：新建 test/registered/unit/bench/test_mmmu_eval_utils.py，通过模块桩隔离依赖，覆盖多行捕获、显式答案优先、旧逻辑回退等场景，并注册为 stage-a-test-cpu CI 任务。

关键文件：

- benchmark/mmmu/eval_utils.py（模块 MMMU 评估；类别 source；类型 core-logic；符号 _parse_explicit_multi_choice_answer, parse_multi_choice_response, EvalArgs.response_answer_regex）：核心逻辑变更：修正默认正则并新增显式答案解析函数，直接影响 MMMU 评估结果。
- test/registered/unit/bench/test_mmmu_eval_utils.py（模块 MMMU 测试；类别 test；类型 test-coverage；符号 TestMMMUEvalUtils, _load_mmmu_eval_utils, _build_data_utils_stub, _build_datasets_stub）：新增完整的 CPU 单元测试，隔离依赖验

证解析逻辑，确保回归覆盖。

关键符号：_parse_explicit_multi_choice_answer, parse_multi_choice_response, _load_mmmu_eval_utils

关键源码片段

benchmark/mmmu/eval_utils.py

核心逻辑变更：修正默认正则并新增显式答案解析函数，直接影响 MMMU 评估结果。

```
# benchmark/mmmu/eval_utils.py

@dataclasses.dataclass
class EvalArgs:
    # ... 其他字段
    response_answer_regex: str = "(?s)(.*)" # 启用 DOTALL 模式，捕获多行响应
    # ...


def _parse_explicit_multi_choice_answer(response, all_choices):
    """优先从 response 中提取显式答案标记（如 'Answer: B'、'**B**'、'**(B)**'）

    返回匹配的最靠后的选项字母，或 None。
    """
    choice_map = {choice.upper(): choice for choice in all_choices}
    matches = []

    # 模式 1: 'answer:' 后的可选括号和星号
    answer_pattern = r"\banswer\s*:\s*\*{0,2}\s*\(?([A-Z])\)?\s*\*{0,2}(?![A-Za-z])"
    for match in re.finditer(answer_pattern, response, flags=re.IGNORECASE):
        candidate = match.group(1).upper()
        if candidate in choice_map:
            matches.append((match.start(1), choice_map[candidate]))

    # 模式 2: 行首 / 行尾独立的字母（可能带括号、星号、句点）
    final_letter_pattern = r"(?:^|\n)\s*\*{0,2}\s*\(?([A-Z])\)?\s*\*{0,2}\s*\.?\s*$"
    for match in re.finditer(final_letter_pattern, response, flags=re.IGNORECASE):
        candidate = match.group(1).upper()
        if candidate in choice_map:
            matches.append((match.start(1), choice_map[candidate]))

    # 返回匹配位置最靠后的（通常是最晚出现的答案）
    return max(matches)[1] if matches else None


def parse_multi_choice_response(response, all_choices, index2ans):
    # 优先使用显式解析
    explicit_answer = _parse_explicit_multi_choice_answer(response, all_choices)
    if explicit_answer is not None:
```

```
            return explicit_answer
        # 原逻辑：扫描 (A)/(B) 等，回退到内容匹配或随机选择
        # ...
```

test/registered/unit/bench/test_mmmu_eval_utils.py

新增完整的 CPU 单元测试，隔离依赖验证解析逻辑，确保回归覆盖。

```
# test/registered/unit/bench/test_mmmu_eval_utils.py
import importlib.util
import re
import sys
import types
import unittest
from pathlib import Path

try:
    from sglang.test.ci.ci_register import register_cpu_ci
    from sglang.test.test_utils import CustomTestCase
except ModuleNotFoundError:
    CustomTestCase = unittest.TestCase
    def register_cpu_ci(*args, **kwargs):
        pass

register_cpu_ci(est_time=5, suite="stage-a-test-cpu")


def _load_mmmu_eval_utils():
    """加载 benchmark/mmmu/eval_utils.py，并为其依赖注入空桩，避免 GPU 依赖。"""
    repo_root = Path(__file__).resolve().parents[4]
    module_path = repo_root / "benchmark" / "mmmu" / "eval_utils.py"
    module_name = "_test_mmmu_eval_utils"

    stub_modules = {
        "data_utils": _build_data_utils_stub(),
        "datasets": _build_datasets_stub(),
        "numpy": _build_numpy_stub(),
        "torch": types.ModuleType("torch"),
        "tqdm": _build_tqdm_stub(),
    }
    previous_modules = {name: sys.modules.get(name) for name in stub_modules}
    sys.modules.update(stub_modules)

    spec = importlib.util.spec_from_file_location(module_name, module_path)
    module = importlib.util.module_from_spec(spec)
    sys.modules[module_name] = module
    try:
        spec.loader.exec_module(module)
    finally:
        # 还原 sys.modules，避免污染
```

```
        for name, previous_module in previous_modules.items():
            if previous_module is None:
                sys.modules.pop(name, None)
            else:
                sys.modules[name] = previous_module
    return module


class TestMMMUEvalUtils(CustomTestCase):
    @classmethod
    def setUpClass(cls):
        cls.eval_utils = _load_mmmu_eval_utils()

    def test_default_response_answer_regex_captures_multiline_response(self):
        # 验证 DOTALL 正则能捕获多行文本
        response = "Based on the diagram, compare the labeled points.\nAnswer: B"
        answer = re.search(self.eval_utils.EvalArgs.response_answer_regex, response)
        self.assertIsNotNone(answer)
        self.assertEqual(answer.group(1), response)

    def test_parse_multi_choice_prefers_explicit_answer_marker_after_copied_options(self):
        # 验证显式答案标记优先于旧选项列表扫描
        response = (
            "The options are:\n"
            "(A) red\n"
            "(B) blue\n"
            "(C) green\n"
            "(D) yellow\n"
            "Answer: B"
        )
        pred_ans = self.eval_utils.parse_multi_choice_response(
            response, ["A", "B", "C", "D"], self._index_to_answer()
        )
        self.assertEqual(pred_ans, "B")

    # 更多测试方法 ...
```

评论区精华

- 暂无高价值评论线程

风险与影响

- 风险：主要风险在解析逻辑变更可能影响原本正常单行答案的提取。但测试用例覆盖了显式答案标记和旧路径，且修复前后的评估对比显示正确率大幅提升（0.5933→0.7433），没有引入明显回归。新增测试在 CPU CI 中运行，不会影响 GPU 资源。
- 影响：直接影响 MMMU 基准测试的结果准确性，使用者无需修改配置即可自动受益。新增单元测试在 CI 中执行，增加约 5 秒耗时。对 SGLang 核心服务无影响。

- 风险标记：正则标志变更，解析优先级调整，测试覆盖主要场景

关联脉络

- 暂无明显关联 PR