PR #23954 完整报告

sgl-project/sglang

[Bench] Fix bench_serving missing reasoning_content stream chunks

合并时间：2026-05-01 06:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23954

执行摘要

- 一句话：修复 bench_serving 忽略 reasoning_content 导致指标为零
- 推荐动作：此 PR 值得合并，它解决了 bench_serving 在处理推理模型时的关键 bug。对于开发者，可以借鉴其处理 API 扩展字段的代码风格：通过 or "" 安全处理 null，以及保留顺序的拼接方式。新增的单元测试设计模式（本地 SSE 服务器 + 多场景覆盖）也值得在其他类似工具测试中复用。

功能与动机

关联 Issue #23949 指出 bench_serving 对推理模型报告零值的 TTFT/ITL 和 retokenized tokens，即使后端完成了实际工作。根本原因是流式响应通过 delta.reasoning_content 传递内容，而解析逻辑只关注 delta.content。

实现拆解

分为三个步骤：

1. 调整 usage 解析顺序：在 python/sglang/bench_serving.py 的流式循环中，将 output_len 的提取移至循环顶部，以处理可能出现的 usage-only chunks（choices: []），随后跳过无 choices 的 chunk。
2. 融合 reasoning_content 与 content：将 delta 的获取改为 choices[0].get("delta") or {}，然后构造 content = (delta.get("reasoning_content") or "") + (delta.get("content") or "")，保留 reasoning 在前面 content 在后面的顺序，并用 or "" 安全处理 null 值。之后 metrics 计算（TTFT、ITL、text_chunks）统一基于此 content。
3. 新增单元测试：在 test/registered/bench_fn/test_bench_serving_reasoning_stream.py 中创建本地 SSE 服务器，模拟推理模型流式输出。覆盖场景包括：纯 reasoning_content 流、reasoning 后接 content 流、同个 delta 中同时包含两者、仅 usage 的最终 chunk、纯 content 流（回归测试）、以及 reasoning_content: null 的显式 null 值。测试验证 generated_text、TTFT、ITL、text_chunks、output_len 等指标正确。

关键文件：

- python/sglang/bench_serving.py（模块 基准测试；类别 source；类型 core-logic；符号 async_request_openai_chat_completions）：核心修改文件，修复了流式解析中遗漏 reasoning_content 的问题

- test/registered/bench_fn/test_bench_serving_reasoning_stream.py（模块 单元测试；类别 test；类型 test-coverage；符号 _free_port, _SSEHandler, _make_chunk, TestBenchServingReasoningStream）：新增全面单元测试，覆盖各种流式场景，确保修复正确性

关键符号：async_request_openai_chat_completions, _make_chunk, _run, TestBenchServingReasoningStream.setUpClass

关键源码片段

python/sglang/bench_serving.py

核心修改文件，修复了流式解析中遗漏 reasoning_content 的问题

```
async for chunk_bytes in response.content:
    chunk_bytes = chunk_bytes.strip()
    if not chunk_bytes:
        continue

    chunk = remove_prefix(chunk_bytes.decode('utf-8'), 'data: ')
    latency = time.perf_counter() - st
    if chunk == '[DONE]':
        pass
    else:
        data = json.loads(chunk)

        # 先提取 completion_tokens，因为部分后端会发送 usage-only 的 chunk（choices 为空）
        output_len = (data.get('usage') or {}).get(
            'completion_tokens', output_len
        )

        choices = data.get('choices') or []
        if not choices:
            # usage-only chunk，跳过
            continue

        # 安全获取 delta，并拼接 reasoning_content 与 content （保留顺序）
        delta = choices[0].get('delta') or {}
        # 使用 or '' 安全处理 None
        content = (delta.get('reasoning_content') or '') + (
            delta.get('content') or ''
        )

        if content:
            timestamp = time.perf_counter()
            if ttft == 0.0:
                # 第一个 token 到达时间 = TTFT
                ttft = timestamp - st
                output.ttft = ttft
            else:
```

```
                # 后续每个 token 记录 ITL
                output.text_chunks.append(content)
                output.itl.append(timestamp - most_recent_timestamp)

            most_recent_timestamp = timestamp
            generated_text += content

    output.generated_text = generated_text
    output.success = True
    output.latency = latency
    output.output_len = output_len
```

test/registered/bench_fn/test_bench_serving_reasoning_stream.py

新增全面单元测试，覆盖各种流式场景，确保修复正确性

```
    def _run(self, chunks):
        port = _free_port()

        class Handler(_SSEHandler):
            pass

        Handler.chunks = list(chunks)
        server = HTTPServer(('127.0.0.1', port), Handler)
        thread = threading.Thread(target=server.serve_forever, daemon=True)
        thread.start()
        try:
            req = RequestFuncInput(
                prompt='hello',
                api_url=f'http://127.0.0.1:{port}/v1/chat/completions',
                prompt_len=1,
                output_len=64,
                model='dummy-model',
                lora_name='',
                image_data=None,
                extra_request_body={},
            )
            return asyncio.run(async_request_openai_chat_completions(req))
        finally:
            server.shutdown()
            server.server_close()


    def test_reasoning_only_stream_populates_metrics(self):
        # 模拟推理模型流：纯 reasoning_content ，然后只含 usage 的 chunk
        chunks = [
            _make_chunk(reasoning_content='Let '),
            _make_chunk(reasoning_content='me '),
            _make_chunk(reasoning_content='think.'),
            _make_chunk(completion_tokens=3),
```

```
        ]
        out = self._run(chunks)

        self.assertTrue(out.success)
        # 验证 generated_text 由 reasoning_content 拼接而成
        self.assertEqual(out.generated_text, 'Let me think.')
        self.assertGreater(out.ttft, 0.0)
        self.assertEqual(len(out.itl), 2)
        for v in out.itl:
            self.assertGreater(v, 0.0)
        self.assertEqual(out.text_chunks, ['me ', 'think.'])
        self.assertEqual(out.output_len, 3)
```

评论区精华

在 Review 中，AgainstEntropy 评论建议保持 reasoning_content 在 content 之前的顺序，并指出即使在实际中不太可能出现同一 delta 包含两者的场景，但保留顺序更安全。作者采纳该建议，最终实现为 (delta.get("reasoning_content") or "") + (delta.get("content") or "")。此外，gemini-code-assist 给出了无反馈的评论。整体讨论简洁，无争议。

- 保持 reasoning_content 与 content 的顺序 (design): 采纳建议，实现中已按此顺序拼接。

风险与影响

- 风险：主要风险在于该函数被 sglang-oai-chat、vllm-chat、lmdeploy-chat 三个后端共享，改动可能影响所有后端的指标计算。但改动仅扩展了解析范围，未改变计算结果的结构，风险较低。对于非推理模型，若后端返回 reasoning_content: null，or "" 确保安全。另需注意，若某后端在同一 delta 中同时返回 reasoning_content 和 content，顺序拼接可能影响 token 计数，但测试已覆盖此场景。整体风险可控。
- 影响：对用户：修复了 bench_serving 对推理模型（DeepSeek-R1、MiMo-V2.5、Qwen3 reasoning、Kimi-K2 等）的基准测试指标，从此不再显示误导性的零值。对系统：仅修改 bench_serving 工具，不涉及推理引擎核心路径。对团队：提高了基准测试工具的准确性和可靠性，便于后续性能分析和文档引用。影响范围为所有使用 --backend sglang-oai-chat、vllm-chat、lmdeploy-chat 进行推理模型基准测试的用户。
- 风险标记：影响多个 OpenAI 兼容后端，流式解析逻辑变更

关联脉络

- 暂无明显关联 PR