PR #35323 完整报告

sgl-project/sglang

fix(openai): avoid duplicate routed expert in response when `return_meta_info = True`

合并时间：2026-08-21 06:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35323

执行摘要

- 一句话：修复双标志下 routed_experts 在 chat 响应中重复返回
- 推荐动作：建议精读。虽然源码改动仅 5 行，但 review 中关于「return_meta_info 是否应成为全局 sglext 开关」的设计交锋很有价值，最终选择最小化抑制、保持扩展字段独立性的决策适用于同类 API 兼容层的字段治理；测试通过序列化断言（精确计数 "routed_experts" 出现次数）锁住了去重语义，这种字段级测试写法值得借鉴。

功能与动机

PR body 指出症状：The base64 routed_experts payload appeared in both choices[*].meta_info and response-level sglext when both return flags were enabled。根因是 _build_chat_response 把第一个 choice 的 routed_experts 投影进 SglExt，而 choice_meta_info 又为每个 choice 序列化同一份路由数据。对 RL 工作流影响明显，因为 routed_experts 是按 [num_tokens, num_layers, top_k] 展开的完整专家路由历史，重复回传既浪费带宽又可能让客户端拿到两份不一致数据。

实现拆解

1. 变更入口与根因定位：核心改动集中在 python/sglang/srt/entrypoints/openai/serving_chat.py 的 _build_chat_response。该函数先构造响应级 SglExt（从 first_ret 提取 routed experts、cached token 明细与 spec token 明细），再逐 choice 构建响应，而每个 choice 的 meta_info 又会携带 routed experts，这是重复数据的来源。
2. 核心逻辑改造：将 routed_experts = process_routed_experts_from_ret(first_ret, request) 改为三元表达式，request.return_meta_info 为真时直接置 None。由于 SglExt 的创建条件是“任一扩展字段非空”，当 routed_experts 是唯一请求的扩展字段时，响应级 sglext 整体消失；当同时请求 cached/spec 明细时，sglext 仍存在但只含后两者。这一设计刻意避开为“禁用 sglext”再引入一个新的布尔开关。
3. 测试配套：test/registered/unit/entrypoints/openai/test_serving_chat.py 新增 3 个高信号用例：test_non_streaming_extension_fields_emit_sglext_without_meta_info（return_meta_info=false 时扩展字段全部进 sglext，choice 无 meta_info）；test_non_streaming_meta_info_omits_response_level_routed_experts（双标志且 n=2 时 sglext 缺失，序列化后 "routed_experts" 恰好出现 2 次，防重复）；test_non_streaming_meta_info_preserves_cache_and_spec_in_sglext（组合标志下 sglext 保留 cache/spec，仅省略 routed experts，防未来误删）。

4. 文档配套：docs/docs/basic_usage/openai_api_completions.mdx 更新 return_routed_experts 一节，明确 return_meta_info=true 时导出数据只出现在 choice 的 meta_info，其余扩展字段仍留在 sglext。

关键文件：

- python/sglang/srt/entrypoints/openai/serving_chat.py（模块 聊天接口；类别 source；类型 core-logic；符号 _build_chat_response）：非流式 chat 响应构建的核心入口，_build_chat_response 在此将 routed_experts 提取改为受 return_meta_info 控制，是全部行为变更所在。
- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 聊天测试；类别 test；类型 test-coverage；符号 test_non_streaming_extension_fields_emit_sglext_without_meta_info, test_non_streaming_meta_info_omits_response_level_routed_experts, test_non_streaming_meta_info_preserves_cache_and_spec_in_sglext）：测试配套主体，新增 3 个字段级用例覆盖各标志组合，用序列化断言精确锁住去重语义，是评审讨论的焦点之一。
- docs/docs/basic_usage/openai_api_completions.mdx（模块 使用文档；类别 docs；类型 documentation）：同步更新 return_routed_experts 的接口说明，明确双标志下字段归属，避免调用方误解新契约。

关键符号：_build_chat_response, test_non_streaming_extension_fields_emit_sglext_without_meta_info, test_non_streaming_meta_info_omits_response_level_routed_experts, test_non_streaming_meta_info_preserves_cache_and_spec_in_sglext

关键源码片段

python/sglang/srt/entrypoints/openai/serving_chat.py

非流式 chat 响应构建的核心入口，_build_chat_response 在此将 routed_experts 提取改为受 return_meta_info 控制，是全部行为变更所在。

```
def _build_chat_response(
    self,
    request: ChatCompletionRequest,
    ret: List[Dict[str, Any]],
    created: int,
) -> Union[ChatCompletionResponse, ORJSONResponse]:
    """根据生成结果构建非流式 chat 补全响应。"""

    # kimi_k3 编码规格需要重写 prompt_tokens 的上报口径，其余模型直接透传
    if self.chat_encoding_spec == "kimi_k3":
        ret = [
            {
                **item,
                "meta_info": {
                    **item["meta_info"],
                    "prompt_tokens": self._reported_prompt_tokens(item["meta_info"]),
                },
            }
```

```
            for item in ret
        ]

    choices = []

    # 响应级 sglext 从第一个 ret_item 提取请求级扩展字段；
    # routed_experts 是逐 token 的专家路由数据，体积大，当 return_meta_info 开启时
    # 由每个 choice 的 meta_info 承担，避免同一份 base64 数据在响应中出现两次
    first_ret = ret[0]
    routed_experts = (
        None
        if request.return_meta_info
        else process_routed_experts_from_ret(first_ret, request)
    )
    cached_tokens_details = process_cached_tokens_details_from_ret(
        first_ret, request
    )
    spec_details = [
        detail
        for detail in (
            process_spec_tokens_details_from_ret(item, request) for item in ret
        )
        if detail is not None
    ]
    # n > 1 时 spec 详情按 choice 顺序平铺，n == 1 时退化为单值或 None
    spec_tokens_details = (
        spec_details
        if request.n > 1
        else (spec_details[0] if spec_details else None)
    )
    response_sglext = None
    if routed_experts or cached_tokens_details or spec_tokens_details:
        # 任一扩展字段存在才创建 SglExt；routed_experts 被抑制时
        # 只要 cached/spec 仍在，sglext 就继续存在，字段彼此独立
        response_sglext = SglExt(
            routed_experts=routed_experts,
            cached_tokens_details=cached_tokens_details,
            spec_tokens_details=spec_tokens_details,
        )
    # 后续循环为每个 ret_item 构建 choice，meta_info 中保留 routed_experts 原值
```

test/registered/unit/entrypoints/openai/test_serving_chat.py

测试配套主体，新增 3 个字段级用例覆盖各标志组合，用序列化断言精确锁住去重语义，是评审讨论的焦点之一。

```
  def test_non_streaming_meta_info_omits_response_level_routed_experts(self):
    # n=2 且双标志开启：每个 choice 需要各自携带路由数据
    req = ChatCompletionRequest(
        model="x",
```

```
        messages=[{"role": "user", "content": "Hi?"}],
        max_tokens=100,
        n=2,
        return_meta_info=True,
        return_routed_experts=True,
    )
    ret = [
        {
            "text": f"Response {index}",
            "meta_info": {
                "id": "chatcmpl-meta-test",
                "prompt_tokens": 10,
                "completion_tokens": 2,
                "cached_tokens": index,
                "routed_experts": routed_experts,
                "finish_reason": {"type": "stop", "matched": None},
                "weight_version": "default",
            },
        }
        for index, routed_experts in enumerate(["cm91dGUtYQ==", "cm91dGUtYg=="])
    ]

    response = self.chat._build_chat_response(req, ret, 1234567890)

    # routed_experts 是唯一请求的扩展字段时，响应级 sglext 整体缺失
    self.assertIsNone(
        response.sglext,
        "sglext is absent only when routed_experts is the sole extension",
    )
    # 每个 choice 的 meta_info 保持权威数据，sglext 不再重复投影
    self.assertEqual(
        [choice.meta_info for choice in response.choices],
        [ret_item["meta_info"] for ret_item in ret],
    )
    dumped_response = response.model_dump()
    self.assertNotIn("sglext", dumped_response)
    # 序列化后 routed_experts 只出现 2 次（两个 choice 的 meta_info），无重复
    serialized_response = json.dumps(dumped_response)
    self.assertEqual(serialized_response.count('"routed_experts"'), 2)
```

评论区精华

> nanjiangwill（design 质疑）：Do we want return_meta_info to disable the entire sglext object? It removes the duplicated routed_experts and cached_tokens_details, but it would also remove unrelated extension fields. main just added spec_tokens_details in #33518, so after rebasing, would combining these flags also drop sglext.spec_tokens_details? Would an explicit return_sglext=false be safer for callers like miles?

> guapisolo（回应）：Updated... return_meta_info now suppresses only response-level sglext.routed_experts. Independently requested cached_tokens_details and spec_tokens_details remain in sglext, so this avoids a broader return_sglext switch。非 routed expert 字段体积不大，重复开销可接受。
>
> nanjiangwill（测试覆盖）：should we also cover return_spec_tokens_details=true here after rebasing? I'm wondering whether this assertion makes the test assume that return_meta_info should always remove every current and future sglext field。
>
> guapisolo（答复）：Addressed in 584a0880b... 组合测试覆盖 return_meta_info + routed + cached + spec + n=2，显式断言 serialized sglext.spec_tokens_details 仍保留，且 sglext 缺失仅发生在 routed_experts 为唯一扩展字段时。

评审结论：两位 reviewer nanjiangwill 与 hnyls2002 均批准；讨论确认了“字段独立性优先、不做全局 sglext 开关”的取舍，并用组合测试锁住未来回归。

- return_meta_info 是否应整体禁用 sglext / 是否引入 return_sglext 开关 (design)：guapisolo 改为仅抑制 sglext.routed_experts，cached_tokens_details 与 spec_tokens_details 保持独立，不引入新开关。
- 组合标志测试是否应覆盖 spec_tokens_details 防止未来回归 (testing)：已解决；测试明确字段级行为，并注明 sglext 缺失仅发生在 routed_experts 为唯一扩展字段时。

风险与影响

- 风险：
  1. 公共契约变更：return_meta_info=true + return_routed_experts=true 时响应级 sglext.routed_experts 消失，依赖该位置的客户端（如 miles 类 RL 调用方）需迁移到 choices[*].meta_info；属于轻微 breaking change，文档已同步但缺少显式的迁移提示。
  2. 隐式耦合：抑制完全依赖 return_meta_info 与「meta_info 内含 routed_experts」的假设。若后续某个模型 / 编码规格（如 kimi_k3 分支）在 meta_info 中改变该字段的填充方式，路由数据会被静默丢弃；当前仅靠单测锁住这一假设。
  3. 行为修正的负载影响：修复后 n>1 时各 choice 的 meta_info 各自携带独立路由数据，而此前所有 choice 共享第一份；整体响应体积在多数场景下降，但并行采样下 per-choice 数据量会随 n 线性增长，大 n 场景需关注序列化时长。
  4. 回归面：改动集中在非流式 chat 单分支，流式与 completions 端点不受影响，CI 中相关 130 个单测与 55 个子测试全部通过。 - 影响：对用户与调用方而言，双标志请求的响应格式发生变化（routed_experts 位置从响应级 sglext 移到每个 choice 的 meta_info），重复数据被消除，客户端不再需要处理两份可能不一致的副本；对系统而言减少了同一份大体积 base64 路由数据的序列化与网络传输，性能只增不减；对团队而言，该修复明确了扩展字段的归属原则——per-choice 数据进 meta_info、per-request 数据进 sglext，为后续新增扩展字段（如 spec_tokens_details 这类）提供了规范参考。影响范围限定在非流式 /v1/chat/completions 的特定标志组合，整体可控。 - 风险标记：公共 API 响应契约变更，字段来源隐式耦合，非流式 chat 路径行为修正

关联脉络

- PR #33518 Add spec_tokens_details to chat response extension: nanjiangwill 在 review 中明确引用 #33518（main 刚合并的 spec_tokens_details 功能），本 PR 的目标之一就是确保 return_meta_info 不会连带删除该字段；该 title 依据 review 内容推断。
- PR #26510 Fix _GenerationStreamAccumulator logprob_end off-by-one under retract: 同为 OpenAI 输出数据正确性修复，与 logprob 游标错位问题一样，属于输出契约逐步精确化的脉络。