

PR #39328 完整报告

sgl-project/sglang

Fix first-token metadata and reused attention-layer indexing

合并时间: 2026-09-16 06:22

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39328>

执行摘要

- 一句话: 修复首 token 延迟元数据与复用层 attention 索引
- 推荐动作: 值得快速精读。两个修复都很小但边界处理值得借鉴:
index_attention_layers_by_global_id 用对象引用相等区分“复用”与“重复”, 并保留位置索引表的语义; convert_to_output_meta_info 用 first_token_time > created_time 的守卫防止脏延迟输出。关注点: 复用层返回原始表后, 调用方对表长度的假设是否仍然成立。

功能与动机

PR body 指出两点: 客户端需要引擎测得的首 token 延迟 (“Clients need the engine's first-token latency”); 复用层模型 (同一层占据多个执行位置) 在初始化前会预展开 attention 查找表, 这些表必须被保留 (“pre-expanded attention tables must survive initialization when the same layer occupies multiple execution positions”)。原实现一方面没有在响应元数据中暴露 first_token_latency, 另一方面 index_attention_layers_by_global_id 会把任何重复 layer_id 一律判为非法, 导致复用层模型在 CUDA graph 初始化阶段被误拒绝。

实现拆解

1. 响应元数据暴露首 token 延迟: 在 convert_to_output_meta_info (python/sglang/srt/observability/req_time_stats.py) 中新增 created_time > 0.0 and first_token_time > created_time 守卫, 满足时才写入 first_token_latency。该条件同时排除未初始化的 0 哨兵与相等 / 回退时间戳, 避免输出无效或负数延迟。
2. 复用层索引放行: 在 index_attention_layers_by_global_id (python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py) 中把原“重复即抛错”拆分为: layer_id < 0 仍报错; 目标位置已有节点时, 仅当现有节点与当前节点是同一对象 (is 比较、companion 相同) 才视为复用并跳过, 否则抛 ValueError。任一复用出现时, 函数直接返回调用方传入的原始 attention_layers 与 mha_companion_layers, 因为预展开表按执行位置索引, 压缩会丢失位置语义。
3. 测试配套: test_req_time_stats.py 新增 TestOutputMetaInfo, 用 subTest 覆盖 5 组时间戳边界 (正常、created 为 0、first 为 0、两者相等、first 小于 created); test_cuda_graph_setup.py 新增 test_reuse_tables_pass_through_but_distinct_duplicates_raise, 覆盖复用表 is 原样返回与不同对象重复 layer_id 抛错两条路径。两个用例均注册 CPU CI。

4. 无其他配套：无配置、部署或 schema 改动；第二个 commit 为 merge main 以满足 CI 最低基线要求。

关键文件：

- python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py (模块 图捕获；类别 source；类型 data-contract；符号 index_attention_layers_by_global_id)：核心修复所在：index_attention_layers_by_global_id 从“只要 layer_id 重复就抛错”改成“同一对象复用放行并返回原始预展开表，不同对象重复仍拒绝”，避免复用层模型在 CUDA graph 初始化时被误拒绝。
- python/sglang/srt/observability/req_time_stats.py (模块 可观测性；类别 source；类型 core-logic；符号 convert_to_output_meta_info)：在响应元数据中暴露 first_token_latency，是客户端可观测性需求的核心落点；仅在时间戳有效且有序时输出，避免脏数据。
- test/registered/unit/observability/test_req_time_stats.py (模块 测试；类别 test；类型 test-coverage；符号 TestOutputMetaInfo, test_first_token_latency_requires_valid_time_stamps)：新增 TestOutputMetaInfo 覆盖 5 组时间戳边界，保证 first_token_latency 只在有效且有序时出现在 meta_info 中，防止回归。
- test/registered/unit/model_executor/model_runner_components/test_cuda_graph_setup.py (模块 测试；类别 test；类型 test-coverage；符号 test_reuse_tables_pass_through_but_distinct_duplicates_raise)：新增回归用例验证复用层表原样返回与不同对象重复 layer_id 抛错，锁定新校验语义。

关键符号：index_attention_layers_by_global_id, convert_to_output_meta_info

关键源码片段

python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py

核心修复所在：index_attention_layers_by_global_id 从“只要 layer_id 重复就抛错”改成“同一对象复用放行并返回原始预展开表，不同对象重复仍拒绝”，避免复用层模型在 CUDA graph 初始化时被误拒绝。

```
# 来自 python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py
```

```
def index_attention_layers_by_global_id(
    attention_layers: list[Any],
    mha_companion_layers: list[Any],
    layer_model=None,
) -> tuple[list[Any], list[Any]]:
    '''把 PP 局部的 attention 元数据按全局 layer_id 对齐。
```

```
    复用层模型（同一 layer 对象出现在多个执行位置）会预先展开查找表；这种表必须原样返回，不能按 layer_id 压缩，否则会丢失执行位置语义。
```

```
'''
```

```

if len(attention_layers) != len(mha_companion_layers):
    raise ValueError('attention and MHA companion metadata must be parallel')

populated = [layer for layer in attention_layers if layer is not None]
if not populated or any(not hasattr(layer, 'layer_id') for layer in populated):
    if layer_model is not None:
        return (
            _align_pipeline_layers(attention_layers, layer_model),
            _align_pipeline_layers(mha_companion_layers, layer_model),
        )
    return attention_layers, mha_companion_layers

max_layer_id = max(int(layer.layer_id) for layer in populated)
indexed_attention = [None] * (max_layer_id + 1)
indexed_companions = [None] * (max_layer_id + 1)
has_reused_layers = False # 标记是否出现同一层对象的复用

for attention, companion in zip(attention_layers, mha_companion_layers):
    if attention is None:
        if companion is not None:
            raise ValueError('MHA companion has no primary attention layer')
        continue

    layer_id = int(attention.layer_id)
    if layer_id < 0:
        raise ValueError(f'invalid or duplicate attention layer_id: {layer_id}')

    if indexed_attention[layer_id] is not None:
        # 已存在同 ID 节点：只有同一对象复用才放行，不同对象重复仍报错
        if (
            indexed_attention[layer_id] is not attention
            or indexed_companions[layer_id] is not companion
        ):
            raise ValueError(f'invalid or duplicate attention layer_id: {layer_id}')
        has_reused_layers = True
        continue

    indexed_attention[layer_id] = attention
    indexed_companions[layer_id] = companion

# 复用层场景下，预展开表以执行位置为索引，长度与 layer_id 无关，
# 压缩会丢失位置语义，因此原样返回调用方传入的列表
if has_reused_layers:
    return attention_layers, mha_companion_layers
return indexed_attention, indexed_companions

```

python/sglang/srt/observability/req_time_stats.py

在响应元数据中暴露 `first_token_latency`，是客户端可观测性需求的核心落点；仅在时间戳有效且有序时输出，避免脏数据。

```
# 来自 python/sglang/srt/observability/req_time_stats.py

def convert_to_output_meta_info(
    self, scheduler_time_stats=None, completion_tokens=0
):
    meta_info = {}
    if self.created_time > 0.0:
        meta_info['request_received_ts'] = convert_time_to_realtime(
            self.created_time
        )

    # 仅当 created_time 与 first_token_time 均有效 (> 0) 且有序
    # (first_token_time > created_time) 时才输出，避免 0 哨兵和
    # 乱序时间戳产生负数或错误的延迟值
    if self.created_time > 0.0 and self.first_token_time > self.created_time:
        meta_info['first_token_latency'] = self.get_first_token_latency()

    if self.api_server_dispatch_finish_time > 0.0:
        meta_info['api_server_dispatch_finish_ts'] = convert_time_to_realtime(
            self.api_server_dispatch_finish_time
        )
    if self.response_sent_to_client_time > 0.0:
        meta_info['response_sent_to_client_ts'] = convert_time_to_realtime(
            self.response_sent_to_client_time
        )
    if self.finished_time > 0.0:
        meta_info['request_finished_ts'] = convert_time_to_realtime(
            self.finished_time
        )

    decode_latency = self.get_decode_latency()
    if decode_latency > 0.0 and completion_tokens > 1:
        meta_info['decode_throughput'] = (completion_tokens - 1) / decode_latency
    return meta_info
```

评论区精华

该 PR 没有任何 review 评论，唯一的评论是作者合并前发出的 `/tag-and-rerun-ci` 维护者指令（用于重跑 CI，对应 Run #34944830577 与 Run #34944830585）。因此本次改动的设计权衡未经过公开评审：复用层的放行条件（同一对象 `is` 判定 + 原始表原样返回）由实现和测试直接固化。分支名 `lianmin/request-timing-layer-reuse-sync-20260913` 也表明这是一个同步分支 PR，低 review 密度符合预期。

- 合并前重新触发 CI (other): 无实质讨论；PR 由作者直接合并，设计决策固化在实现与测试中。分支名提示这是一条同步分支 PR。

风险与影响

- 风险:

1. 核心路径变更: `cuda_graph_setup.py` 位于 CUDA graph 捕获初始化路径, 行为从“重复 ID 一律拒绝”放宽为“同一对象复用放行并返回原始表”。若调用方假定返回表长度恒等于 `max_layer_id + 1`, 复用层场景下会不一致, 需确认所有调用点只按 `layer_id` 寻址而不依赖压缩后的表。
2. 校验语义放宽: 错误消息 `invalid or duplicate attention layer_id` 在新逻辑下对 `layer_id < 0` 场景略不精确, 但不影响行为正确性。
3. 契约新增字段: `meta_info` 新增 `first_token_latency`, 属于向后兼容增量, 但依赖精确键集合做快照比对的客户端会观察到新字段。
4. 验证缺口: 作者声明未运行 GPU 推理, 复用层模型路径 (共享层 / 循环层) 未经真实初始化验证; CI Extra 运行失败 (Run #34944830585) 的根因未在 PR 中说明。- 影响: 对用户 / API: 响应 `meta_info` 新增可选 `first_token_latency` 字段, 客户端可直接读取引擎测得的首 token 延迟; 在非法时间戳场景该字段保持缺席, 避免脏数据。对系统: 修复了复用层模型 (同一层对象出现在多个执行位置) 在 CUDA graph 初始化阶段被误拒绝的问题, 属于初始化路径的正确性恢复。对团队: 改动范围小、无 breaking change, 但需要在真实 GPU 上补跑复用层模型初始化验证, 并跟进 CI Extra 失败。- 风险标记: 核心路径变更, 校验语义放宽, 缺少 GPU 验证, CI Extra 失败

关联脉络

- PR #39164 Generalize auxiliary outputs: 同一时期重构输出元数据契约 (`auxiliary_output.py`、`batch_result_processor.py`), 与本 PR 的 `convert_to_output_meta_info` 变更同属 `meta_info` 契约演进方向。
- PR #38630 [profiler] Label draft-runner steps DRAFT and target verify VERIFY in step spans: 同属 observability 改善线, 为可观测性链路补充更细粒度的时序信息; 与本 PR 的 `first_token_latency` 暴露互补。