PR #35926 完整报告

sgl-project/sglang

Report per-token weight-version spans in generation meta info

合并时间：2026-08-24 20:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35926

执行摘要

- 一句话：生成 meta info 新增逐 token 权重版本跨度，支撑 RL 归因
- 推荐动作：值得精读。重点看三点：一是 compute_weight_version_spans() 的游程编码边界处理与零 token 特例；二是“参考模型 + 随机 / 穷举对照”的单测写法（_expected_spans 朴素实现与生产实现互证），这是算法类改动很值得借鉴的测试模式；三是提交序列本身展示了“先提取复用函数（collect_inflight_reqs、_make_abort_req），再修状态同步 bug，最后接新功能”的重构顺序，可作为多提交 PR 的结构范例。

功能与动机

RL 训练器需要区分同一次请求中由不同权重版本产出的 token：权重热更新（update_weight_version）后继续生成时，一条回复的前后 token 可能来自不同版本的权重。提交 a955c2ff 明确说明："Threads the weight version that produced each token through the scheduler, output streamer and tokenizer manager, so an RL trainer can tell which tokens came from which published weights." 此外提交 9a444be2 指出原有设计缺陷：POST /update_weight_version 只更新 tokenizer 侧副本，且权重更新路径从未写入调度器自身记录，导致任何按调度器读取版本号的逻辑读到启动时的陈旧值。

实现拆解

实现按 5 步拆解：

1. 新增共享模块 python/sglang/srt/utils/weight_versions.py：定义 WeightVersionSpan（msgspec.Struct，array_like=True，带 __get_pydantic_core_schema__ 钩子以兼容 Pydantic 序列化）、WeightVersionEvent（冻结 dataclass，记录 old_version 与变更时累计 token 数）、record_weight_version_events（仅对已有 output_ids 的请求打点）、truncate_weight_version_events（token 截断时同步压缩事件）、compute_weight_version_spans（将事件流游程编码为连续半开区间）、add_weight_versions_to_meta_info（写入 meta info）、build_endpoint_weight_version_metadata（OpenAI 兼容端点透出）。
2. 数据结构接线：schedule_batch.py 为 Req 增加 weight_version_events 字段；io_struct.py 为 AbortReq 增加 weight_versions 字段并默认 None，保持 IPC 兼容。
3. 调度器接线（scheduler.py）：先提取 collect_inflight_reqs() 复用 PP 分支下的请求遍历，再提取 _make_abort_req() 统一 8 处 AbortReq 构造点；

record_weight_version_change() 现在会收集全部“存活”请求来源（inflight、waiting queue、chunked req、hisparse ack_staging_queue）并逐请求记录事件；_make_abort_req() 在构造时通过 compute_weight_version_spans() 计算并携带版本区间。

4. 输出流式处理与 TokenizerManager 接线：output_streamer.py 的 _GenerationStreamAccumulator 新增 current_weight_version 与 weight_versions 字段，accept() 对已完成请求填充区间、对流式中的请求填 None，to_payload() 仅在存在有效区间时序列化；tokenizer_manager.py 调用 add_weight_versions_to_meta_info() 把区间合并进 meta info。
5. 测试配套：算法参考模型单测（test_weight_versions.py，含 _expected_spans 朴素实现对比）、调度器打点覆盖（test_scheduler_weight_version_tracking.py）、输出流负载断言（test_output_streamer_customized_info.py）、msgpack IPC 往返（test_msgpack_ipc_roundtrip.py）、多 tokenizer mixin 嵌套输出测试，以及 2 卡 nightly RL 集成测试（test_weight_version_spans.py，覆盖暂停 / 继续、EAGLE、DP attention 场景）。

关键文件：

- python/sglang/srt/utils/weight_versions.py（模块 权重版本；类别 source；类型 core-logic；符号 WeightVersionSpan, get_pydantic_core_schema, WeightVersionEvent, record_weight_version_events）：新增核心模块：定义版本区间 / 事件数据结构与 span 计算、记录、截断、meta info 组装等全部核心逻辑，是整条链路的中枢。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 collect_inflight_reqs, record_weight_version_change, _make_abort_req）：调度器接线主文件：提取 collect_inflight_reqs 与 _make_abort_req 两个复用点，在权重更新时给全部 live 请求打点，并在 abort 路径携带版本区间。
- python/sglang/srt/managers/scheduler_components/output_streamer.py（模块 输出流；类别 source；类型 dependency-wiring；符号 _GenerationStreamAccumulator, accept, to_payload）：输出流式处理接线：_GenerationStreamAccumulator 新增 current_weight_version 与 weight_versions 字段，完成请求填充区间、流式请求填 None，to_payload 仅在需要时序列化，避免负载膨胀。
- python/sglang/srt/managers/tokenizer_manager.py（模块 请求路由；类别 source；类型 dependency-wiring）：将版本区间写入最终生成 meta info 的入口，是用户可见输出的最后一站。
- python/sglang/srt/managers/schedule_batch.py（模块 数据模型；类别 source；类型 data-contract；符号 Req）：为 Req 增加 weight_version_events 字段，是每请求记录版本事件的数据基础。
- python/sglang/srt/managers/io_struct.py（模块 IPC 协议；类别 source；类型 data-contract；符号 AbortReq）：AbortReq 新增 weight_versions 字段，保证 IPC 往返可以携带版本区间并向后兼容。
- test/registered/unit/utils/test_weight_versions.py（模块 权重版本；类别 test；类型 test-coverage；符号 _ReqStub, _expected_spans, TestComputeWeightVersionSpans, test_no_events_returns_single_span）：核心算法测试：用朴素参考模型（逐 token 命名

后游程编码）与生产实现随机对照，并覆盖零 token、更新早于首个 token、多次更新等边界。

- test/registered/rl/test_weight_version_spans.py（模块 集成测试；类别 test；类型 test-coverage；符号 _assert_spans_contiguous, TestWeightVersionSpans, _generate, _pause）：2 卡 nightly RL 集成测试：真实拉起 EAGLE + DP attention 服务，覆盖暂停 / 继续、权重热更新、多请求并发等端到端场景，并断言区间连续无重叠。
- test/registered/unit/managers/test_output_streamer_customized_info.py（模块 输出流；类别 test；类型 test-coverage；符号 _accumulator, TestOutputStreamerWeightVersions, test_payload_carries_spans_for_finished_requests, test_payload_omits_spans_while_all_requests_stream）：验证输出流负载语义：已完成请求携带区间、流式中请求为 None，并适配 accumulator 新构造参数。
- test/registered/unit/managers/test_msgpack_ipc_roundtrip.py（模块 IPC 协议；类别 test；类型 test-coverage；符号 TestWeightVersionSpansRoundTrip, test_abort_req_carries_spans, test_abort_req_defaults_to_no_spans）：验证 AbortReq 携带区间经 msgpack IPC 往返不丢失，默认值为空时保持兼容。

关键符号：compute_weight_version_spans, record_weight_version_events, truncate_weight_version_events, add_weight_versions_to_meta_info, build_endpoint_weight_version_metadata, Scheduler.record_weight_version_change, Scheduler.collect_inflight_reqs, _make_abort_req, _GenerationStreamAccumulator.accept, _GenerationStreamAccumulator.to_payload

关键源码片段

python/sglang/srt/utils/weight_versions.py

新增核心模块：定义版本区间 / 事件数据结构与 span 计算、记录、截断、meta info 组装等全部核心逻辑，是整条链路的中枢。

```
# python/sglang/srt/utils/weight_versions.py（整理后的核心实现）

class WeightVersionSpan(msgspec.Struct, kw_only=True, array_like=True):
    # array_like = True 让 msgspec 按紧凑数组布局编解码，跨进程传输时字段顺序稳定；
    # __get_pydantic_core_schema__ 让 Pydantic 也能直接使用该 Struct（OpenAI
    风格端点会经过此路径）。
    version: str
    start: int
    end: int


WeightVersionSpans = List[WeightVersionSpan]


@dataclasses.dataclass(frozen=True, slots=True)
class WeightVersionEvent:
    old_version: str
    num_output_tokens: int
```

```
def record_weight_version_events(reqs: Iterable[Req], old_version: str) -> int:
    # 权重变更发生在某轮调度前，此刻请求已有的 output_ids 全部属于旧版本；
    # 记录“变更发生时的累计 token 数”，后续据此在 token 序列上切出版本边界。
    num_recorded = 0
    for req in reqs:
        if req.output_ids:
            req.weight_version_events.append(
                WeightVersionEvent(
                    old_version=old_version,
                    num_output_tokens=len(req.output_ids),
                )
            )
            num_recorded += 1
    return num_recorded


def compute_weight_version_spans(
    events: List[WeightVersionEvent],
    current_version: str,
    num_output_tokens: int,
) -> WeightVersionSpans:
    # 把事件流转换为 [start, end) 半开区间，最终覆盖 0..num_output_tokens：
    # 每个事件声明“到该位置为止的 token 由旧版本采样”，末尾追加当前版本兜底；
    # 同时保证区间单调整齐、相邻同版本合并，零 token 时产出空区间但保留版本语义。
    changes = [(event.old_version, event.num_output_tokens) for event in events]
    changes.append((current_version, num_output_tokens))

    spans: WeightVersionSpans = []
    for version, end in changes:
        end = min(end, num_output_tokens)
        if spans and end <= spans[-1].end:
            continue
        if spans and version == spans[-1].version:
            spans[-1].end = end
            continue
        start = spans[-1].end if spans else 0
        spans.append(WeightVersionSpan(version=version, start=start, end=end))
    return spans
```

python/sglang/srt/managers/scheduler.py

调度器接线主文件：提取 collect_inflight_reqs 与 _make_abort_req 两个复用点，在权重更新时给全部 live 请求打点，并在 abort 路径携带版本区间。

```
# python/sglang/srt/managers/scheduler.py（整理后的关键逻辑）

def record_weight_version_change(self, new_version: Optional[str]) -> None:
    old_version = get_serving().weight_version
    get_context().override("scheduler.weight_version", weight_version=new_version)
```

```
    # 收集所有“存活”请求来源：正在 prefill/decode 的 inflight、等待队列、
    # 当前 chunked 请求，以及 hisparse 协调器暂未 ack 的请求。
    # 只有全部打上时间戳，后续才可能把整条输出序列无遗漏地切分成版本区间。
    live_reqs = {
        *self.collect_inflight_reqs(),
        *self.waiting_queue,
        *([self.chunked_req] if self.chunked_req is not None else []),
    }
    if self.hisparse_coordinator is not None:
        live_reqs.update(act.req for act in self.hisparse_coordinator.ack_staging_queue)

    num_recorded = record_weight_version_events(live_reqs, old_version=old_version)
    logger.info(f"Weight version changed. {old_version=} {new_version=} {num_recorded=}")


def _make_abort_req(
    req: Req, finished_reason: Optional[FinishReasonDict] = None
) -> AbortReq:
    # abort 时一并携带算好的版本区间，tokenizer 侧无需再回看调度器状态；
    # 即使请求被提前中断，RL 训练器也能拿到逐段权重归因。
    return AbortReq(
        rid=req.rid,
        finished_reason=finished_reason,
        weight_versions=compute_weight_version_spans(
            req.weight_version_events,
            current_version=get_serving().weight_version,
            num_output_tokens=len(req.output_ids),
        ),
    )
```

test/registered/unit/utils/test_weight_versions.py

核心算法测试：用朴素参考模型（逐 token 命名后游程编码）与生产实现随机对照，并覆盖零 token、更新早于首个 token、多次更新等边界。

```
# test/registered/unit/utils/test_weight_versions.py（参考模型片段）

def _expected_spans(events, current_version, num_output_tokens):
    # 参考模型：先给每个 token 命名所属版本，再做游程编码；
    # 相比生产实现完全直白、无边界优化，专门用来对照验证。
    per_token = []
    for index in range(num_output_tokens):
        owner = next(
            (event.old_version for event in events if event.num_output_tokens > index),
            current_version,
        )
        per_token.append(owner)

    # 零 token 请求仍然产出一个空区间，确保“当前版本”语义可见。
    if not per_token:
```

```
        first_event_end_at_zero = next(
            (event for event in events if event.num_output_tokens >= 0), None
        )
        version = (
            first_event_end_at_zero.old_version
            if first_event_end_at_zero is not None
            else current_version
        )
        return [WeightVersionSpan(version=version, start=0, end=0)]

    spans = []
    for index, version in enumerate(per_token):
        if spans and spans[-1].version == version:
            spans[-1].end = index + 1
        else:
            spans.append(WeightVersionSpan(version=version, start=index, end=index + 1))
    return spans
```

评论区精华

本 PR 没有人工 review 评论，全部讨论来自 CI 自动化与作者操作：

- 机器人 Codex 指出 rebased-head CPU shard 3 的失败来自 frozen-base 不兼容：test_full_builder_sizes_sidecar_for_anchor_logical_space 调用 get_memory().hicache_mem_layout，但该测试未发布 memory 配置包；后来上游提交 362c2ee849 已修复，不属本 PR 回归。
- 作者通过 /rerun-test 显式跑全部新增 / 修改测试，并让 Codex 逐文件手动校验，包括普通按提交调度可能遗漏的 2 卡 nightly 测试，最终通过。
- CI 失败归因：冻结基线不兼容而非本 PR 回归 (other): 判定为 frozen-base 不兼容，非 op1-3c 回归；作者重跑相关测试后通过。
- 无人工 review，自动化逐文件测试校验 (other): 所有新增 / 修改测试最终通过。

风险与影响

- 风险：
  1. IPC 数据契约变更：AbortReq 与输出 payload 新增 weight_versions 字段，WeightVersionSpan 使用 array_like=True；若调度器与 tokenizer/worker 版本不一致（灰度发布），字段布局差异可能引发反序列化失败。字段默认值降低了风险，但未做跨版本兼容测试。
  2. 调度器主路径开销：record_weight_version_change() 每次权重更新都要遍历全部 live 请求（inflight、waiting queue、chunked、hisparse ack 队列）并创建事件对象；权重更新属低频管理操作，但超大 batch 下遍历成本需关注。
  3. 每请求新增内存：每个 Req 常驻 weight_version_events 列表；长期运行请求若经历多次版本更新，事件数等于更新次数（通常很小），内存可控，但缺少显式上限。

4. 截断正确性：truncate_weight_version_events() 通过 min 钳制 token 数，事件边界在 retract 场景下依赖该函数与 compute_weight_version_spans() 的单调性保证，若未来引入新截断路径容易漏接。
5. 测试体量大：新增约 1300 行测试、20 个文件，但由作者自动合并，无人工 review 兜底，存在覆盖盲区未被发现的可能。 - 影响：对用户：/generate 等生成端点的 meta info 新增 weight_versions（版本区间数组）与 weight_version（当前版本）字段，已完成请求会携带逐 token 归因信息；流式请求在未完成时该字段为 None，避免负载膨胀，属于向后兼容的增量字段。对系统：改动贯穿调度器、输出流式处理、TokenizerManager 与 OpenAI 端点，核心路径每请求增加一次区间计算（事件数通常为个位数，开销可忽略）。对团队：为 RL 训练按 token 归因权重版本提供标准数据通道，后续可在此基础上实现“丢弃过期 token”“按版本加权”等策略；同时该 PR 顺带修复了调度器版本号滞后 bug，降低了 /get_model_info、abort 等路径读到陈旧版本的风险。 - 风险标记：核心路径变更，IPC 数据契约变更，每请求新增内存开销，测试为主且无人工审查

关联脉络

- PR #35928 Expose the declared sglang env vars of a scheduler in its internal state: 同作者同批次针对调度器内部状态与可观测性的系列改动，扩展 Scheduler 状态表达，与本 PR 的权重版本状态追踪同属调度器观测能力建设。
- PR #35929 Report the whole server's world size in the scheduler's internal state: 同为调度器内部状态扩展（新增整机 world_size 字段），与本 PR 在调度器状态与观测信息上报方向上相互补充。