

PR #38630 完整报告

sgl-project/sglang

[profiler] Label draft-runner steps DRAFT and target verify VERIFY in step spans

合并时间: 2026-09-15 15:57

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38630>

执行摘要

- 一句话: 为 profiler step span 加 DRAFT/VERIFY 阶段标签
- 推荐动作: 值得精读, 但不必投入大量时间: 这是约 30 行的可观测性改进, 重点看两个设计选择——一是用「worker 身份」而非「forward mode」作为标签的第一判据, 二是用强制关键字参数 + 默认值实现向后兼容。若你维护 profiler、trace 解析脚本或性能报告, 请顺手把 TARGET_VERIFY 的匹配规则更新为 VERIFY 并补上 DRAFT, 同时评估是否要为 draft 的 EXTEND 形态单独保留一个更可读的 DRAFT_EXTEND 标签; 若你只是关注推理正确性, 可以跳过本 PR。

功能与动机

PR body 明确指出问题: `step[...]` span 只按 forward mode 命名, 而 draft model runner 可能运行在 `TARGET_VERIFY` 模式下, 导致 trace 里 draft 与 target 的 verify step 无法区分——「Reading a speculative-decoding trace then requires guessing which `step[TARGET_VERIFY ...]` belongs to which model」。作者因此引入 `is_draft_worker` 标志, 用「模型身份」而非「forward mode」来消歧。

实现拆解

1. 标签生成逻辑改造 (`python/sglang/srt/utils/profile_utils.py`): `build_step_span_name` 新增强制关键字参数 `is_draft_worker: bool = False`, 函数体内把原来直接使用 `mode.name` 的分支替换为三级 stage 判定: `is_draft_worker` 为真时 `stage = "DRAFT"`; 否则 `mode == ForwardMode.TARGET_VERIFY` 时 `stage = "VERIFY"`; 其余情况沿用 `mode.name`。EXTEND 分支的 `toks=` 字段与 `detailed_annotations` 后缀逻辑保持不变, `base` 字符串改为拼接 `stage`。默认值 `False` 保证所有既有调用方行为不变, * 强制关键字调用则避免后续按位置误传。
2. 调用点接线 (`python/sglang/srt/model_executor/model_runner.py`): `ModelRunner.forward` 中创建 step span 的一行改为 `build_step_span_name(forward_batch, is_draft_worker=self.is_draft_worker)`。同一方法内部原本就用 `not self.is_draft_worker` 控制 canary manager, 因此这个属性在 runner 上已是稳定可用的身份标记, 复用代价极低。
3. 测试配套 (`test/registered/unit/managers/test_detailed_annotations.py`): `test_target_verify_uses_draft_token_width` 与 `test_target_verify_without_cpu_mirror_falls_back_to_base` 的期望值由

step[TARGET_VERIFY ...] 改为 step[VERIFY ...]; 新增 test_draft_worker_prefixes_stage, 用 ForwardMode.TARGET_VERIFY + is_draft_worker=True 组合断言输出 step[DRAFT bs=2], 直接覆盖「draft 借用 target mode」这一核心场景。

4. 未覆盖的边界（作者在 body 中主动说明）：multi_layer_eagle_worker_v2 的 draft-extend span 仍以不带标志的方式调用该函数，因为其 mode 名 DRAFT_EXTEND_V2 已能表明 draft 身份；同理，任何未传 is_draft_worker 的调用点仍会退回旧命名。

关键文件：

- python/sglang/srt/utils/profile_utils.py（模块 性能分析；类别 source；类型 core-logic；符号 build_step_span_name）：核心逻辑所在：build_step_span_name 新增 is_draft_worker 关键字参数并引入 stage 三级判定，直接决定 trace 里可见的 span 名称。
- python/sglang/srt/model_executor/model_runner.py（模块 模型执行器；类别 source；类型 data-contract；符号 ModelRunner.forward）：唯一接线的调用点：ModelRunner.forward 把自身的 is_draft_worker 传入 step span 构造，使标签能反映真实运行者身份。
- test/registered/unit/managers/test_detailed_annotations.py（模块 标注测试；类别 test；类型 test-coverage；符号 test_draft_worker_prefixes_stage, test_target_verify_uses_draft_token_width, test_target_verify_without_cpu_mirror_falls_back_to_base）：覆盖新行为的单元测试：两条 target-verify 期望值改为 VERIFY，并新增 test_draft_worker_prefixes_stage 验证 draft 借 mode 的场景。

关键符号：build_step_span_name, ModelRunner.forward, test_draft_worker_prefixes_stage

关键源码片段

python/sglang/srt/utils/profile_utils.py

核心逻辑所在：build_step_span_name 新增 is_draft_worker 关键字参数并引入 stage 三级判定，直接决定 trace 里可见的 span 名称。

```
# python/sglang/srt/utils/profile_utils.py
def build_step_span_name(
    forward_batch: ForwardBatch,
    detailed_annotations: bool | None = None,
    *, # 强制关键字传参：避免既有调用方按位置误传，也为后续扩展留余地
    is_draft_worker: bool = False,
) -> str:
    """Build the profile-trace span name for one forward step.

    The target-verify step is labeled ``VERIFY``; every step a draft model
    runner emits is labeled ``DRAFT`` (some draft paths borrow the TARGET_VERIFY
    mode, so the mode name alone cannot tell the two models apart).
    """
    if detailed_annotations is None:
```

```

detailed_annotations = detailed_annotations_enabled()

mode = forward_batch.forward_mode
bs = forward_batch.batch_size

# 关键点：标签优先级是「worker 身份 > forward mode」。
# draft runner 可能借用 TARGET_VERIFY 模式，只看 mode 会和 target 撞名，
# 所以 is_draft_worker 先判，保证 draft 的每一步都读作 DRAFT。
if is_draft_worker:
    stage = "DRAFT"
elif mode == ForwardMode.TARGET_VERIFY:
    stage = "VERIFY" # target 侧 verify 简写，避免长 mode 名挤占 trace 视图
else:
    stage = mode.name # 其余模式保持原名，旧标签语义不变

# 注意：这里判的是 mode 而不是 stage。
# 因此 draft worker 以 EXTEND 形态跑时输出 step[DRAFT bs=.. toks=..],
# 扩展形态只能从 toks= 字段推断，字面量 EXTEND 不再出现。
if mode == ForwardMode.EXTEND:
    ext_toks = forward_batch.extend_num_tokens or 0
    base = f"step[{stage} bs={bs} toks={ext_toks}"
else:
    base = f"step[{stage} bs={bs}"

# detailed annotations 后缀逻辑未变，保证与既有标注能力正交
if detailed_annotations:
    suffix = build_detailed_annotation_suffix(forward_batch)
    if suffix:
        base = f"{base} {suffix}"
return f"{base}"

```

python/sglang/srt/model_executor/model_runner.py

唯一接线的调用点：`ModelRunner.forward` 把自身的 `is_draft_worker` 传入 `step span` 构造，使标签能反映真实运行者身份。

```

# python/sglang/srt/model_executor/model_runner.py, ModelRunner.forward 内的 step span
创建段
self.forward_pass_id += 1

```

```

# ... msprobe debugger 分支略 ...

```

```

# Step span: 把 runner 自身的 draft 身份交给标签构造函数。
# self.is_draft_worker 在本方法后续的 canary_ctx 判定里已经被使用，
# 说明它是 runner 级稳定属性，直接复用无需新增状态。
step_span_ctx = profile_range(
    build_step_span_name(forward_batch, is_draft_worker=self.is_draft_worker)
)

```

同一个身份标记也用于决定是否启用 canary 管理，二者语义一致：

```

# draft worker 不参与 target 侧的 canary 流程。
canary_ctx = (
    context_tuple(
        c.with_ops_outside_graph(
            single_forward_indices=[0],
            maybe_inaccurate_forward_batch=forward_batch,
        ),
        c.with_active_single_forward_manager(0),
    )
    if not self.is_draft_worker and ((c := self.canary_manager) is not None)
    else contextlib.nullcontext()
)

with (
    canary_ctx,
    step_span_ctx, # span 名字此时已带上 DRAFT / VERIFY 阶段前缀
    get_global_expert_distribution_recorder().with_forward_pass(
        self.forward_pass_id,
        forward_batch,
    ) as recorder_outputs,
):
    output = self._forward_raw(
        forward_batch,
        pp_proxy_tensors,
        reinit_attn_backend,
        split_forward_count,
    )

```

test/registered/unit/managers/test_detailed_annotations.py

覆盖新行为的单元测试：两条 target-verify 期望值改为 **VERIFY**，并新增 **test_draft_worker_prefixes_stage** 验证 draft 借 mode 的场景。

```

# test/registered/unit/managers/test_detailed_annotations.py

# 既有用例：target-verify 的标签由 TARGET_VERIFY 改为 VERIFY,
# 详细标注后缀 (g_sq / g_sqsq / g_sqsk / g_sk) 完全不变
self.assertEqual(
    self._name(fb),
    "step[VERIFY bs=2 g_sq=6 g_sqsq=18 g_sqsk=90 g_sk=30]",
)

# 既有用例：缺少 CPU mirror 时退回基础标签，同样改用 VERIFY
self.assertEqual(self._name(fb), "step[VERIFY bs=2]")

# 新增用例：draft runner 借用 TARGET_VERIFY 模式时，span 必须读作 draft 自己，
# 而不是被误认为 target 的 verify step —— 这正是本 PR 要修的歧义场景
# A draft runner can run under TARGET_VERIFY; its span must read as
# the draft's, not the target's.
fb = _fb(ForwardMode.TARGET_VERIFY, batch_size=2)

```

```

self.assertEqual(
    build_step_span_name(fb, detailed_annotations=False, is_draft_worker=True),
    "step[DRAFT bs=2]",
)

# 未被本 PR 覆盖的路径: draft-extend 仍走自己的 mode 名,
# 其 span 保持 step[DRAFT_EXTEND_V2 ...], 调用时并未传 is_draft_worker
self.assertEqual(
    self._name(fb),
    "step[DRAFT_EXTEND_V2 bs=2 c_sq=5 c_sqsq=13 c_sqsk=93 c_sk=35]",
)

```

评论区精华

该 PR 没有 inline review 评论，唯一的讨论痕迹是作者在 Issue 评论区发出的 [/tag-and-rerun-ci](#)，说明主要精力花在 CI 信号的反复确认上，而非代码设计交锋。第二个 commit ([d924ce6](#)，由 merrymercy 提交) 的 message 记录了另一条流程性讨论：「Merge main to satisfy required CI base — Integrate main at 6410800... for the MIN_BASE_SHA gate in issue #21065」，即仓库要求 PR 先合入最新 main 才能跑通必需 CI，这也解释了 PR 状态里 Base/Extra/AMD ROCm 三个测试槽位出现失败标记的背景。整体上这是一个低争议、快速合入的改动。

- 为满足 MIN_BASE_SHA 门槛合并 main 并重跑 CI (testing): 通过把 main 合入 head 满足必需 CI 基础版本要求后再跑流水线，代码改动本身未作调整（「The profiling change remains unchanged」）。
- draft-extend 路径是否也应带 DRAFT 标志 (design): 作者主动保留现状并给出理由，未产生新的讨论或返工，但「draft 步骤一律带 DRAFT 前缀」的不变式在代码上并不成立。

风险与影响

- 风险：
 1. trace 标签契约变化: TARGET_VERIFY 字面量从 span 名中消失，任何按 step[TARGET_VERIFY 做字符串匹配的外部脚本、trace 看板或文档示例都会失配；PR 只更新了单元测试，未在本 diff 中体现对文档 / 工具链的同步排查。
 2. 标签信息量在特定组合下退化: stage 三元判定与 EXTEND 判定使用的是不同条件（前者 stage，后者 mode == ForwardMode.EXTEND），因此一个 draft worker 若以 EXTEND 形态运行，会输出 step[DRAFT bs=.. toks=..]，字面上不再出现“EXTEND”，下游只能靠 toks= 字段反推扩展形态。
 3. 消歧不彻底: 只有 ModelRunner.forward 一个调用点接上了 is_draft_worker; multi_layer_eagle_worker_v2 等路径按作者说明刻意保留原样，因此「draft 步骤一律带 DRAFT 前缀」这一不变式在代码层面并不成立，依赖 span 名的分析代码需要按调用点分别处理。
 4. 测试脆弱性（可接受）: 新增与修改的断言都是完整字符串比较，未来任何标签措辞调整都会直接失败；考虑到这正是本次要锁定的行为，属于有意的回归护栏。

5. 性能与正确性：每步只做一次字符串拼接，无额外同步、无额外集合操作；
detailed_annotations 与 TARGET_VERIFY 的 attention 语义均未触碰，不存在精度或性能回归风险。 - 影响：影响面集中在「用 profiler 看 speculative decoding 的人」：开启 profiling / detailed annotations 后，EAGLE、MTP 等场景的 trace 里不再需要靠猜来区分 draft 与 target 的 verify step，排障效率提升。对普通推理请求路径零影响——标签字符串不参与调度、不进入 KV 缓存或结果返回。对下游生态属于轻量 breaking change：以 span 名做正则或前缀匹配的 trace 解析脚本、Perfetto 预设查询、内部 profiling 报告需要把 TARGET_VERIFY 替换为 VERIFY，并新增 DRAFT 分支。对团队而言，这是 profiler 可观测性持续打磨的一个小步，落地成本低、收益直接。 - 风险标记：trace 标签契约变更，下游按旧标签匹配需同步，draft+EXTEND 丢失 EXTEND 字面量，仅一个调用点接线

关联脉络

- PR #36821 [KDA] Support ReplaySSM ring-write in the fused chain-verify kernel: 同属 speculative decoding 的 verify 路径优化，本 PR 恰好改善了该类 verify 步骤在 profiler 中的可辨识度，二者在排障分工上互补（一个改内核、一个改可观测性）。
- PR #39366 fix: stop shadowing the DSpark shared-experts fusion guard: 同为 speculative-decoding 相关的模型侧改动，涉及 draft/target 双模型协作路径；本 PR 解决的正是同一协作在 trace 维度上的 draft/target 混淆问题。
- PR #38687 [Kernel] Add OOT dispatch for clamp position: 该 PR 修改了 forward_batch_info.py 中的 ForwardBatch 消费方式，而本 PR 的 build_step_span_name 同样以 ForwardBatch (forward_mode、batch_size、extend_num_tokens) 为输入契约，可作为 ForwardBatch 字段语义演进脉络的参考。