

PR #38554 完整报告

sgl-project/sglang

[Spec] Allow speculative workers to stage prefill shared reads

合并时间: 2026-09-15 05:03

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38554>

执行摘要

- 一句话: 为投机解码 worker 新增 prefill 共享读 staging 回调, 默认关闭不改变上游行为
- 推荐动作: 值得精读, 尤其是负责投机解码、prefill 调度与 KV 生命周期的人。关注三点设计决策: (1) 用「worker 自证返回 bool 的 opt-in 回调」而不是框架内启发式判断来放开 EAGLE/MTP, 把风险显式外移到实现方, 这是本 PR 最核心的取舍; (2) 把 shared_read_ends 后端声明检查提到 spec 检查之前, 保证 staging 只发生在后端已声明 PRE_REPLAY 的路径上, 是正确的最小权限做法; (3) 单测用 calls.mock_calls == [call.stage(batch)] + ([call.Event()] if staged else []) 锁定调用顺序, 而不是只断言最终状态, 这种「顺序即契约」的测试写法值得复用。阅读时应同步确认: 任何后续启用该 hook 的 worker PR 必须带上精度与性能验证, 并在 code review 中明确 stage() 返回 True 的语义证明。

功能与动机

PR body 明确写出痛点: "Speculative workers with later draft-prefill readers cannot publish the early prefill read-done event, even when those reads can be staged before target replay." 原实现中 maybe_publish_prefill_shared_read_done 对非 NONE 且非 DFLASH 家族的算法 (EAGLE/MTP 等) 直接 return, 并留有 TODO: "TODO(Jialin): Relax for EAGLE/MTP after validating the later draft-extend reader's WAR boundary."。这意味着这些算法只能退回 shared_read_done_event = None 的粗粒度整轮 forward fence, 牺牲了 prefill 与后续写侧的 overlap 机会。该 PR 提供 opt-in 机制: worker 自己保证共享读已在 target replay 前完成, 才允许提前发布读完成事件, 并声明 "Model-specific staging and performance validation remain with the worker implementation"。

实现拆解

1. 扩展点声明 (python/sglang/srt/model_executor/model_runner.py): 在 ModelRunner.__init__ 中紧接 shared_read_done_event 之后新增 self.prefill_shared_read_stager: Optional[Callable[[ForwardBatch], bool]] = None, 并把 typing 导入从 Optional, Union 扩到 Callable, Optional, Union。字段语义是「由某个投机 worker 作用域内的回调」, 默认 None 表示不启用; 由于没有上游 worker 安装它, 默认路径与改动前完全一致。
2. 门控顺序调整 (python/sglang/srt/model_executor/runner_utils/shared_read_event.py): shared_read_ends(...) is not SharedReadEnds.PRE_REPLAY 的检查被提到 spec 算法

检查之前。这样 staging 回调只会在后端已显式声明「共享读在 replay 前完成」时才被调用，避免对未声明合规的后端做无意义的提前读；同时删除了旧的 EAGLE/MTP TODO 注释。

3. spec 分支从「直接 return」改为「尝试 staging」：非 NONE 且非 DFLASH 家族时，用 `getattr(model_runner, "prefill_shared_read_stager", None)` 取回调；stage is None or not stage(forward_batch) 就 return，保持「不发事件、调度器退回粗粒度 fence」的旧 fallback；只有回调返回 True 才继续 `device_module.Event() + record()`，并写回 `shared_read_done_event`。getattr 兜底同时兼容缺少该属性的轻量 runner 与单测里的 `SimpleNamespace`。
4. DFLASH/DSPARK 与其它门控保持不变：SpeculativeAlgorithm.DFLASH/DSPARK 仍走直接发布分支，不调用 stager；SGLANG_ENABLE_PREFILL_WAR_READ_DONE 总开关、ForwardMode.EXTEND 检查语义均未变。
5. 测试配套 (`test/registered/unit/model_executor/runner/test_prefill_shared_read_done.py`, +44/-1)：_model_runner 假对象补 `prefill_shared_read_stager=None`；新增 `test_speculative_prefill_publishes_only_after_staging`，参数化 staged 并用 `calls.mock_calls == [call.stage(batch)] + ([call.Event()]) if staged else []` 锁定「staging 必须先于 event 构造」的时序契约；新增 `test_prefill_gates_skip_staging`，用 5 组 (enabled, mode, compliant) 参数断言门控不通过时 stager `assert_not_called`；DFLASH 用例补 `assert_not_called()` 明确 DFLASH 家族不走 staging。测试注册为 `register_cpu_ci(est_time=11, suite="base-a-test-cpu")`，属 CPU 快速套件。
6. 无配置 /schema/ 部署改动，也无 benchmark：PR body 明确「No benchmark: this PR adds the hook without enabling it in an upstream worker」，性能收益留给下游 worker 实现去验证。

关键文件：

- `python/sglang/srt/model_executor/runner_utils/shared_read_event.py`（模块 读完成门控；类别 source；类型 core-logic；符号 `maybe_publish_prefill_shared_read_done`）：核心逻辑所在：`maybe_publish_prefill_shared_read_done` 的门控被重排，spec 分支由「直接 return」改为「尝试调用 stager，失败即退回旧 fallback」，并删除了 EAGLE/MTP 的旧 TODO，是本 PR 的语义中心。
- `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.init`）：扩展点的宿主：`ModelRunner.__init__` 新增 `prefill_shared_read_stager` 字段与 `Callable` 导入，决定该 hook 默认 None、按 runner 实例作用域安装，是「不影响上游默认行为」的关键。
- `test/registered/unit/model_executor/runner/test_prefill_shared_read_done.py`（模块 门控单测；类别 test；类型 test-coverage；符号 `test_speculative_prefill_publishes_only_after_staging`, `test_prefill_gates_skip_staging`）：配套单测把回调契约固化为可执行断言：新增 staged 参数化用例锁定「staging 先于 event 构造」的调用顺序，新增门控用例证明未 opt-in 时退回旧 fallback，并给 DFLASH 用例补 `assert_not_called`。

关键符号：`maybe_publish_prefill_shared_read_done`, `ModelRunner.init`,

`test_speculative_prefill_publishes_only_after_staging`, `test_prefill_gates_skip_staging`

关键源码片段

python/sglang/srt/model_executor/runner_utils/shared_read_event.py

核心逻辑所在： `maybe_publish_prefill_shared_read_done` 的门控被重排， `spec` 分支由「直接 return」改为「尝试调用 stager，失败即退回旧 fallback」，并删除了 EAGLE/MTP 的旧 TODO，是本 PR 的语义中心。

```
# python/sglang/srt/model_executor/runner_utils/shared_read_event.py
def maybe_publish_prefill_shared_read_done(
    model_runner, forward_batch, device_module
) -> None:
    """Publish prefill read-done after compliant metadata initialization."""
    # 总开关默认关闭：上游未启用时整条 fastpath 不生效，行为与旧版一致
    if not envs.SGLANG_ENABLE_PREFILL_WAR_READ_DONE.get():
        return
    # 只有 EXTEND (prefill) 在这里发布；TARGET_VERIFY / MIXED / DECODE
    # 由 decode graph runner 那条路径负责，本次改动不影响它们
    if forward_batch.forward_mode != ForwardMode.EXTEND:
        return
    # 注意：这个检查被「提前」了（原来在 spec 检查之后）。
    # 只有后端显式声明共享读在 replay 前结束 (PRE_REPLAY)，才允许提前发事件；
    # 先做这个判断可以避免对未声明合规的后端白跑一次 staging
    declared = model_runner.attn_backend.shared_read_ends(forward_batch.forward_mode)
    if declared is not SharedReadEnds.PRE_REPLAY:
        return
    # 非 NONE 且非 DFLASH 家族 (EAGLE / MTP 等)：draft 阶段可能存在更晚的 prefill 读者。
    # 旧逻辑在这里直接 return，退回「整轮 forward」的粗粒度 fence；
    # 现在改成先给 worker 一次把共享读搬走的机会
    if (
        not model_runner.spec_algorithm.is_none()
        and not model_runner.spec_algorithm.is_dflash_family()
    ):
        # 用 getattr 而不是直接属性访问：兼容缺少该字段的轻量 runner、
        # 老版本对象以及单测里的 SimpleNamespace 假对象
        stage = getattr(model_runner, "prefill_shared_read_stager", None)
        # 回调缺失 (未 opt-in) 或自评失败 (共享读尚未搬完) 都退回旧行为：
        # 不 record、不写 mailbox，调度器继续用粗粒度 fence，只是慢不会错
        if stage is None or not stage(forward_batch):
            return
    logger.info_once(
        "Prefill shared-read-done fastpath active (%s)",
        type(model_runner.attn_backend).__name__,
    )
    # 走到这里意味着：要么算法是 NONE / DFLASH 家族，要么 worker 已自证共享读完成。
    # 事件必须在此刻 record，之后写侧才被允许覆盖这些共享 KV 页 (WAR 边界)
    read_done = device_module.Event()
    read_done.record()
    model_runner.shared_read_done_event = read_done
```

python/sglang/srt/model_executor/model_runner.py

扩展点的宿主: `ModelRunner.__init__` 新增 `prefill_shared_read_stager` 字段与 `Callable` 导入, 决定该 hook 默认 `None`、按 runner 实例作用域安装, 是「不影响上游默认行为」的关键。

```
# python/sglang/srt/model_executor/model_runner.py
from typing import Callable, Optional, Union # 新增 Callable, 用于回调类型标注

# Init forward stream for overlap schedule
self.forward_stream = torch.get_device_module(self.device).Stream()

# Read-done mailbox: the scheduler's WAR barrier reads it from the runner
# its worker names, and treats None as the coarse whole-forward fence.
# None 代表「没有提前发布的读完成事件」, 调度器只能等整轮 forward 结束
self.shared_read_done_event: Optional[torch.cuda.Event] = None
# Scoped by a speculative worker to stage its shared reads before
# the target prefill graph publishes the read-done event.
# 语义: 入参是本次 prefill 的 ForwardBatch, 返回 True 才表示
# worker 已经把共享读提前完成 (自证), 框架据此才敢发布 read-done。
# 默认 None -> 没有任何 worker opt-in, 上游行为与改动前完全一致
self.prefill_shared_read_stager: Optional[Callable[[ForwardBatch], bool]] = None
```

test/registered/unit/model_executor/runner/test_prefill_shared_read_done.py

配套单测把回调契约固化为可执行断言: 新增 `staged` 参数化用例锁定「staging 先于 event 构造」的调用顺序, 新增门控用例证明未 `opt-in` 时退回旧 fallback, 并给 DFLASH 用例补 `assert_not_called`。

```
# test/registered/unit/model_executor/runner/test_prefill_shared_read_done.py
@pytest.mark.parametrize("staged", [False, True])
def test_speculative_prefill_publishes_only_after_staging(staged):
    # EAGLE 属于「可能存在更晚 draft-prefill 读者」的算法,
    # 在旧实现里必然直接 return, 本用例是这次改动的核心回归保护
    runner, batch = _model_runner(spec_algorithm=SpeculativeAlgorithm.EAGLE), _batch()
    calls = Mock()
    runner.prefill_shared_read_stager = calls.stage
    calls.stage.return_value = staged
    # Event 用同一个 Mock 家族替身, 这样 stage 与 Event 会落在同一条 mock_calls 上
    calls.Event.side_effect = _Event
    with envs.SGLANG_ENABLE_PREFILL_WAR_READ_DONE.override(True):
        maybe_publish_prefill_shared_read_done(
            runner, batch, SimpleNamespace(Event=calls.Event)
        )
    # 关键断言: 顺序即契约 —— 必须先 staging, 且只有 staged=True 才构造 Event;
    # 这直接对应「读完才能发事件」的 WAR 语义
    assert calls.mock_calls == [call.stage(batch)] + ([call.Event()] if staged else [])
    published = runner.shared_read_done_event
    if staged:
        assert isinstance(published, _Event) and published.recorded
```

else:

```
# staging 失败时保持 mailbox 为 None，调度器退回粗粒度整轮 fence
assert published is None
```

评论区精华

本次 PR 的 review 讨论几乎为空：review_comments_count = 0, comments_count = 0，只有 Qiaolin-Yu 一条空 body 的 APPROVED 评审，外加一条由 Qiaolin-Yu 提交的 Merge branch 'main' into codex/spec-prefill-shared-read-stager 合并提交。因此有价值的信息只能从 PR body 与代码本身提炼：(1) 作者主动声明 hook 未在上游 worker 启用，把 model-specific staging 与性能验证的责任显式划给后续 worker 实现；(2) PR body 的 CI 状态显示 PR Test 通过，但 "PR Test (Extra)" 与 "AMD ROCm 10" 两个 run 失败 (:x:)，且没有任何评论解释或归类这些失败；(3) 变更本身把 EAGLE/MTP 的 TODO 从「等验证 WAR 边界后放开」改成了「提供 staging 契约后再放开」，设计意图未被 reviewer 追问。

- staging 回调的「自证」语义是否足够安全 (design): 框架只提供契约，不承担正确性证明；正确性风险被显式外移到 opt-in 的 worker 实现。
- 门控顺序重排与 fallback 保留是否覆盖所有旧路径 (testing): 旧 fallback 路径被显式测试覆盖，DFLASH/DSPARK 与门控不通过场景均不会调用 stager，行为保持。
- CI 两个 run 失败未澄清 (other): 无法从现有材料判定失败是否与本改动相关；流程上属于未闭环项。

风险与影响

- 风险：
 1. 自证式契约的正确性风险（核心）：stage(forward_batch) 返回 True 完全依赖 worker 自证「共享读已完成」。shared_read_done_event 是调度器 WAR 屏障读取的 mailbox，一旦 worker 在真实共享读结束前返回 True，事件会在 replay 之后立即 record，写侧可能覆盖仍被读取的共享 KV 页，属于静默数据损坏而非崩溃。PR 明确将该验证留给 worker 实现，本 PR 不承担正确性保证。
 2. 门控顺序变化的副作用：model_runner.attn_backend.shared_read_ends(forward_batch.forward_mode) 现在在 spec 分支之前调用，虽只是纯查询，但对还没实现 / 未注册该接口的后端，异常会出现在更早的位置，堆栈与旧版不同，可能干扰既有排障。
 3. 静默降级难以发现：getattr(model_runner, "prefill_shared_read_stager", None) 的兜底使属性名写错、worker 未安装、或跑在旧版 ModelRunner 上都表现为「退回粗粒度 fence」，只损失性能不报错，缺少日志或断言提示（仅成功路径有 logger.info_once）。
 4. 测试覆盖边界：单测全部基于 SimpleNameSpace + create_autospec(AttentionBackend) + Mock，验证的是门控与调用顺序的契约；没有真实 ModelRunner、没有真实 worker 安装回调、没有端到端精度或 overlap 验证。test_prefill_gates_skip_staging 也只断言 stager 未被调用，不覆盖 SGLANG_ENABLE_PREFILL_WAR_READ_DONE=True + EXTEND + PRE_REPLAY + stager 缺失这一最关键的「静默退回」组合之外的真实运行态。
 5. CI 状态未澄清：PR 带 bypass-fastfail 标签合并，且 "PR Test (Extra)" 与 "AMD ROCm 10" 失败没有任何评论说明是否与本改动相关，无法从材料判断是既有 flaky 还是

新引入，属于流程性风险。

6. 性能收益未量化：前移 read-done 的实际 overlap 收益完全未测；若 staging 本身开销大于节省的等待，可能出现负收益，需要下游 worker 侧基准验证。 - 影响：对上游用户与系统：默认零行为变化——没有任何上游 worker 安装 `prefill_shared_read_stager`, `SGLANG_ENABLE_PREFILL_WAR_READ_DONE` 仍是总开关，非合规后端与 DFLASH/DSPARK 路径完全不变，因此不存在用户可见的精度 / 性能回归面。对下游 / 厂商 worker（如自研 EAGLE/MTP 投机实现）：这是决定性的能力开放——此前这类算法只能吃粗粒度 fence，现在可以获得 prefill 读完成即释放的 overlap 收益，但代价是必须自行证明 staging 的正确性。对团队：该 PR 把 EAGLE/MTP 的 WAR 边界问题从「框架内 TODO」转化为「worker 侧契约」，后续任何启用必须绑定精度回归与基准数据；同时 `getattr` 兜底使该契约是隐式的，缺少文档与显式接口声明，长期看需要补齐（例如在后端抽象或 runner 基类里正式定义该 hook）。 - 风险标记：自证式回调契约，静默降级难以发现，缺少端到端验证，CI 失败未澄清，收益未量化，默认行为不变

关联脉络

- PR #39318 Scope prefetch cache state to the request attempt: 同属 prefill/ 共享读的 WAR 与状态归属问题域：该 PR 把 prefetch 状态按请求尝试隔离以修复 requeue 串扰，涉及 `scheduler.py` 与 `mem_cache` 状态机；本 PR 则处理 read-done 事件的发布时机与共享读的提前完成，二者都在界定「何时可以认为共享读已经安全结束」。
- PR #39232 [Perf] trtllm_mla: reuse the fused fp8 KV/Q prepare on target verify: 同为投机解码路径的 overlap/ 复用优化：该 PR 让 target verify 复用融合的 fp8 KV/Q 准备工作，把每层 preparation 从 5 次 launch 降到 1 次；本 PR 让投机 worker 在 replay 前把共享读搬走，思路一致——把 replay 之前的准备工作前置，缩短关键路径上的等待。
- PR #38941 [Fix] Merge adjacent KV-row frees so a mid-page split under DCP cannot double-free: 同属 KV 生命周期安全边界：该 PR 处理 KV 行释放区间的合并以避免双重释放，本 PR 处理共享读完成后才允许写入的 WAR 边界，二者都直接关系到「KV 页在何种条件下可被安全释放 / 覆盖」这一正确性前提。