PR #32828 完整报告

sgl-project/sglang

[Kimi] Support DCP + DSpark (ported from kimi-k3 branch)

合并时间：2026-08-01 08:39

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32828

执行摘要

- 一句话：移植 Kimi Linear 的 DCP + DSPARK 支持并修复验证后状态漂移
- 推荐动作：值得精读，尤其关注三点：verify 后状态提交的时机与后端能力门控设计、DCP 下 workspace 尺寸推导（head count × dcp_size 与真实 draft 宽度）、以及 allocator page_size 对齐的 KV 释放修复。对实现 speculative decoding、混合注意力模型或数据并行 + 推测解码组合的工程师有直接借鉴意义。

功能与动机

PR body 明确指出 main 只有 DCP 一半（#32612）而没有 DSPARK 一半：混合线性注意力目标（Kimi Linear）的 KDA/mamba recurrent state 在 verify 后从未提交到 accepted length，状态每步漂移。提交 75669cac 进一步说明 server 保持健康但生成序列不正确；最早一个测试提交还指出 #32612 落地时测试被裁剪到 test_kimi_linear_dcp4.py，speculative decoding 路径完全没有测试覆盖，因此本 PR 需要同时补上实现与测试。

实现拆解

1. DSpark worker 提交验证后 mamba 状态（python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py）：新增 _need_mamba_verify_commit 标志，在 init_attention_backends() 中通过 mambaish_config(model_config) is not None and hasattr(attn_backend, "update_mamba_state_after_mtp_verify") 做能力门控，仅对 mamba 类目标且后端暴露钩子时启用。_forward_decode() 在 verify 前调用 prepare_mamba_track_for_verify(batch)（从 spec_utils 新导入），verify 结束后调用新增的 _commit_target_mamba_states_after_verify()，按 chain 布局（断言 speculative_eagle_topk in (None, 1)）以 commit_lens - 1 为最后接受步骤索引，将 KDA/mamba 状态写入持久缓存；若配置了 mamba_track_interval，还计算本次 verify 跨越的 track 点。这是本 PR 的核心正确性修复。
2. TokenSpeed workspace 尺寸修正（python/sglang/srt/layers/attention/tokenspeed_mla_backend.py）：_get_tokenspeed_workspace() 新增 max_q_len 参数，默认为 _TOKENSPEED_MAX_Q_LEN；DCP target verify 会把 Q 按全量 head count gather 后再启动 TokenSpeed，因此 num_heads *= get_parallel().attn_dcp_size，max_q_len 取 max_speculative_num_draft_tokens or 1，避免 workspace 不足导致越界或错误结果。
3. DCP 内存预算与 KV 释放对齐（mem_cache/kv_cache_configurator.py、model_executor/pool_configurator.py、mem_cache/common.py）：replicated draft

pool 分别按 DCP 虚拟 loc 空间与 dcp_size x 内存预算放大；_release_overallocated_kv_indices() 改为从 allocator 读取物理 page_size 而非全局 schedule page_size（DCP 下逻辑页为 1 而物理页可能加宽到 4），避免与 cache_finished_req 的尾部释放重复释放同一物理页，同时清理不再使用的 get_schedule/ get_server_args 导入。这些改动在 dcp_size == 1 时均为 no-op。

4. 测试配套：新增 test/registered/dcp/test_kimi_linear_dcp_dspark4.py（4×Blackwell GSM8K 验收，含 / 不含 --dcp-replicate-q-proj，动态写入 Qwen3DSpark 代理 draft 模型配置）、test/registered/dcp/test_tokenspeed_mla_dcp_metadata.py（验证 TARGET_VERIFY 下 global vs per-rank KV 长度拆分、DECODE 不追加 draft token），并在 test/registered/unit/mem_cache/test_paged_free_segment.py 中新增 test_overallocated_tail_uses_allocator_page_size_under_dcp 单元测试，覆盖 DCP 下 allocator 页加宽的场景。

关键文件：

- python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py（模块 推测引擎；类别 source；类型 core-logic；符号 _commit_target_mamba_states_after_verify, init_attention_backends, _forward_decode）：核心正确性修复：在 DSPARK verify 后提交 KDA/mamba recurrent state，解决状态漂移；同时引入能力门控与 chain 布局断言。
- python/sglang/srt/layers/attention/tokenspeed_mla_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _get_tokenspeed_workspace）：修正 DCP target verify 下的 TokenSpeed workspace 尺寸，按 gather 后的全量 head count 与真实 draft 宽度分配，避免越界或错误结果。
- python/sglang/srt/mem_cache/common.py（模块 缓存释放；类别 source；类型 core-logic；符号 _release_overallocated_kv_indices）：修复 DCP 下 over-allocation 释放按 allocator 物理页对齐，避免与 cache_finished_req 重复释放同一物理页。
- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 缓存配置；类别 source；类型 core-logic）：DCP 下 replicated draft pool 需要按虚拟 loc 空间放大，否则多卡并行时 draft KV 容量不足。
- python/sglang/srt/model_executor/pool_configurator.py（模块 内存池；类别 source；类型 data-contract）：内存求解时按 dcp_size 倍数预算 replicated draft pool，避免 DCP 下内存不足。
- test/registered/dcp/test_kimi_linear_dcp_dspark4.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestKimiLinearDCPDSpark4, _run_static, test_static_verify_cuda_graph）：4×Blackwell GSM8K 端到端验收测试，覆盖带与不带 --dcp-replicate-q-proj 两种配置，是本特性的主要回归防线。
- test/registered/dcp/test_tokenspeed_mla_dcp_metadata.py（模块 元数据测试；类别 test；类型 test-coverage；符号 TestTokenspeedMLADCPMetadata, test_target_verify_splits_global_and_local_lengths, test_decode_does_not_add_draft_tokens）：针对 TokenSpeed CUDA-graph metadata 中 global 与 per-rank KV 长度的拆分行为做单元测试，防止 DCP 下 metadata 错误。
- test/registered/unit/mem_cache/test_paged_free_segment.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_overallocated_tail_uses_allocator_page_size_und

er_dcp）：覆盖 DCP 下 allocator 页加宽后 overallocated tail 释放的单元测试，防止重复释放物理页。

关键符号：_commit_target_mamba_states_after_verify, init_attention_backends, _forward_decode, _get_tokenspeed_workspace, _release_overallocated_kv_indices, test_static_verify_cuda_graph, test_target_verify_splits_global_and_local_lengths, test_decode_does_not_add_draft_tokens

关键源码片段

python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py

核心正确性修复：在 DSPARK verify 后提交 KDA/mamba recurrent state，解决状态漂移；同时引入能力门控与 chain 布局断言。

```
# python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py
# 初始化 Attention 后端后，探测当前目标模型是否需要在 DSPARK verify 后提交 mamba 状态：
# 仅当目标是 mamba 类模型（mambaish_config 非空）且后端实现了
# update_mamba_state_after_mtp_verify 钩子时启用，普通注意力模型完全不受影响。
self._need_mamba_verify_commit = mambaish_config(
    self.model_runner.model_config
) is not None and hasattr(
    self.model_runner.attn_backend,
    "update_mamba_state_after_mtp_verify",
)

# _forward_decode 的热点路径：verify 前准备 mamba track，verify 后提交最后接受步骤的状态
def _forward_decode(self, ...):
    ...
    # 若启用了 mamba_track_interval，先把 batch 的 track 信息准备好，供后续提交时定位
    prepare_mamba_track_for_verify(batch)

    with self._observers.segment(InfoSegment.TARGET_VERIFY):
        # run_compact / run_ragged 执行目标模型 verify，产出 accept 结果
        ...

    # verify 结束后，把最后一次被接受步骤的 KDA/mamba 状态提交到持久缓存，
    # 否则状态会停在验证窗口末尾，导致每步与已接受 token 序列漂移。
    self._commit_target_mamba_states_after_verify(
        batch=batch,
        seq_lens_pre_verify=prefix_lens,
        seq_lens_post_verify=accept.new_seq_lens,
        commit_lens=accept.commit_lens,
    )
    ...


def _commit_target_mamba_states_after_verify(
    self,
    *,
```

```
    batch: ScheduleBatch,
    seq_lens_pre_verify: torch.Tensor,
    seq_lens_post_verify: torch.Tensor,
    commit_lens: torch.Tensor,
) -> None:
    """提交 verify 中最后接受的 KDA/mamba 状态（chain 布局：步骤索引 = commit_lens -
    1）到持久缓存。"""
    if not self._need_mamba_verify_commit:
        return
    # 仅支持 chain 布局（topk <= 1）：tree 布局需要 spec_utils 中按 accept
    索引映射的提交辅助函数，
    # 此处直接断言，防止误用产生错误状态。
    assert self.server_args.speculative_eagle_topk in (None, 1)
    attn_backend = self.target_worker.model_runner.attn_backend

    last_correct_step_indices = commit_lens.to(torch.int64) - 1
    mamba_steps_to_track = None

    if batch.mamba_track_indices is not None:
        # 若开启 mamba track 间隔，则找到本次 verify 跨越的 track 点，
        # 并算出该点在 draft 序列中的第几步（to_track_ith）。
        mamba_track_interval = self.server_args.mamba_track_interval
        to_track_mask = (
            seq_lens_pre_verify // mamba_track_interval
            != seq_lens_post_verify // mamba_track_interval
        )
        tracking_point = (
            seq_lens_post_verify // mamba_track_interval * mamba_track_interval
        )
        to_track_ith = torch.clamp(tracking_point - seq_lens_pre_verify - 1, min=0)
        ...
    # 最终调用后端钩子 update_mamba_state_after_mtp_verify 完成状态写入
```

python/sglang/srt/layers/attention/tokenspeed_mla_backend.py

修正 DCP target verify 下的 TokenSpeed workspace 尺寸，按 gather 后的全量 head count 与真实 draft 宽度分配，避免越界或错误结果。

```
# python/sglang/srt/layers/attention/tokenspeed_mla_backend.py
def _get_tokenspeed_workspace(
    device: torch.device,
    num_heads: int,
    kv_lora_rank: int,
    max_q_len: int = _TOKENSPEED_MAX_Q_LEN,
) -> torch.Tensor:
    from sglang.srt.runtime_context import get_resources

    # DCP target verify 会把 Q 按全量 head count gather 后交给 TokenSpeed，
    # 因此 workspace 必须按 gather 后的 head 数分配，而不是 rank 本地的 head 数。
    num_heads *= get_parallel().attn_dcp_size
```

```
    max_q_len = max(max_q_len, _TOKENSPEED_MAX_Q_LEN)

    needed = (
        tokenspeed_mla.get_num_sm(device)
        * num_heads
        * max_q_len
        * (kv_lora_rank + 1)
        * 4
    )
    ...

# 构造后端时显式传入真实 speculative 宽度（draft 数）作为最大 Q 长度；
# 若未配置则退化为 1，保持原有行为。
self._tokenspeed_workspace = _get_tokenspeed_workspace(
    self.device,
    self.num_q_heads,
    self.kv_lora_rank,
    max_q_len=(model_runner.server_args.max_speculative_num_draft_tokens or 1),
)
```

python/sglang/srt/mem_cache/common.py

修复 DCP 下 over-allocation 释放按 allocator 物理页对齐，避免与 cache_finished_req 重复释放同一物理页。

```
# python/sglang/srt/mem_cache/common.py
def _release_overallocated_kv_indices(
    req: Req, start_p: int, end_p: int, tree_cache: BasePrefixCache
) -> None:
    # DCP 场景下逻辑页 size 为 1，但物理 allocator 页可能被加宽（如 4），
    # 必须用 allocator 的物理页尺寸做对齐，否则可能与 cache_finished_req
    # 的尾部释放重复释放同一物理页，导致页计数错乱。
    allocator = tree_cache.token_to_kv_pool_allocator
    page_size = allocator.page_size
    spec_algo = get_spec().speculative_algorithm

    # strip_thinking_cache 会把输出 token 视为 over-allocation 走释放路径（#22373）
    if spec_algo is None and not get_serving().strip_thinking_cache:
        assert (
            start_p == end_p
        ), f"Unexpected overallocated KV cache, {req.kv_committed_len=}, {req.kv.kv_allocated_len=
        }"

    if page_size > 1:
        start_p = ceil_align(start_p, page_size)

    if start_p < end_p:
        indices_to_free = tree_cache.req_to_token_pool.req_to_token[req.req_pool_idx][
            start_p:end_p
        ]
```

```
# start_p 对齐到 allocator 物理页后，永不与同一 group 的尾部释放共享页面
allocator.free_segment(indices_to_free, start_pos=start_p)
```

评论区精华

本 PR 没有 formal review 代码评论，9 条 issue 评论集中在 CI 失败归因。作者两次 /rerun-test 后 test_kimi_linear_dcp_dspark4.py 在 4-gpu-b200 上仍失败，最终作者定位到 extra CI 的失败将由 #33127 修复，与本 PR 无关，并声明其余测试全部通过、可以安全合并。这反映出端到端 DSPARK 测试在多卡环境下极易受环境干扰，失败的归因需要单独隔离确认。

- extra CI 失败是否由本 PR 引入 (question): 作者确认失败为环境相关问题、不属于本 PR 引入，PR 以安全合并方式合入。

风险与影响

- 风险：
  1. 核心 verify 路径变更：dspark_worker_v2.py 的 _forward_decode() 在每个 decode 步新增 prepare + commit 两阶段调用，影响所有 DSPARK decode 路径，虽以 _need_mamba_verify_commit 门控，仍属于热点路径改动。
  2. 仅支持 chain 布局：_commit_target_mamba_states_after_verify() 断言 speculative_eagle_topk in (None, 1)，未来若启用 tree 布局会直接抛 AssertionError，需要按 accept 索引映射扩展提交逻辑。
  3. 依赖后端能力门控：通过 hasattr(attn_backend, "update_mamba_state_after_mtp_verify") 探测钩子，若后端方法改名或删除，Kimi Linear + DSPARK 会静默丢失状态提交，重新引入漂移 bug。
  4. 显存预算随 DCP 放大：TokenSpeed workspace 按 attn_dcp_size 倍数缩放、replicated draft pool 按 dcp_size x 预算，dcp_size > 1 时显存占用显著上升，小显存配置有 OOM 风险。
  5. KV 释放语义变化：mem_cache/common.py 从 schedule page_size 改为 allocator 物理 page_size，影响所有 speculative over-allocation 释放路径；虽已覆盖 DCP 场景，但 SWA 等其他 allocator 组合未全覆盖。
  6. 端到端测试依赖 4×Blackwell 硬件：常规 CI 无法覆盖，只能靠 extra CI，回归发现成本高。 - 影响：影响范围集中在 Kimi Linear + TokenSpeed + DSPARK + DCP 的组合部署；默认配置 dcp_size == 1 时行为完全不变，现有用户无感。启用后，用户可以得到无状态漂移的 Kimi Linear 推测解码，消除每步状态错位导致的生成退化。对团队而言，该 PR 合入 kimi-k3 分支的关键特性，降低双分支维护成本，并为 DSPARK 支持其他混合线性注意力模型建立了可复用的模式（能力门控 + chain 布局状态提交）。性能上，每个 decode 步多一次 mamba 状态提交与 track 计算，TokenSpeed workspace 更大，通常可忽略。 - 风险标记：核心 verify 路径变更，仅支持 chain 布局，依赖后端能力门控，显存预算随 DCP 放大，端到端测试依赖 4×Blackwell

关联脉络

- PR #32612 [Kimi] DCP support for Kimi Linear: PR body 明确说明 main 已有 DCP 一半（#32612），本 PR 补上缺失的 DSPARK 一半；且 #32612 落地时裁剪了 speculative 测试，本 PR 补回。
- PR #33127 Fix for the failing extra CI run: 作者在最后一条评论中指出 extra CI 的失败将由 #33127 修复，与本 PR 无关，用于隔离失败归因。
- PR #35957 Fix recurrent state loss on decode retraction: 同为 recurrent state 提交 / 恢复主题：该 PR 修复 decode retraction 时 recurrent state 静默丢失，与本 PR 的 verify 后状态提交互补，共同保证 mamba 类目标状态一致性。
- PR #35840 Add PD test for inkling with mxfp8 KV: 同样涉及 schedule_batch 与 mem_cache 的 retraction 状态备份，同属 mamba 状态生命周期与缓存一致性维护路线。