PR #36933 完整报告

sgl-project/sglang

[2/N][Mixed] Mixed chunk prefill with spec enabled

合并时间：2026-09-01 01:48

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36933

执行摘要

- 一句话：打通 mixed chunk 与投机解码的组合降级路径
- 推荐动作：值得精读，尤其关注两条设计决策：一是把混合 step 的降级契约表达为 supports_mixed_chunk() 能力位而非算法字符串硬编码；二是 overlap 下“调度期占位 + forward 入口晚期绑定”的尾部重建模式（resolve_mixed_spec_tails），它绕开了 CPU 无法得知在飞 accept 数的根本矛盾。阅读建议：先看 mix_with_running 与 resolve_mixed_spec_tails，再读测试文件中的三个 cell，最后按 commit 顺序浏览 [Fix] 记录理解每个坑的成因。

功能与动机

PR body 基本是模板占位，动机主要体现在 commit 叙事与代码注释中。改动前 speculative_hook.py 对 dflash、dspark、eagle 三处硬编码 declare_resolution(..., enable_mixed_chunk=False)，因为 mixed step 与 spec 的 draft/verify 循环不兼容；本 PR 把该组合升级为一等公民。作者在 commit 中说明关键权衡：“Mixed chunking already saturates compute, so drafting adds latency for little gain there”，即混合步骤计算已饱和，draft 只增加延迟，因此降级 running 请求为纯 1-token extend 并无损失；同时降级契约是算法无关的，应表达为 worker 能力而非硬编码算法名。ngram 因 overlap relay 发布到 accept 缓冲区而非 output_tokens_buf 暂不支持，STANDALONE 因 draft-KV 恢复路径未验证保持关闭。

实现拆解

整个实现分为 5 步：

1. 能力门控（spec_info.py、spec_registry.py、speculative_hook.py、validation_hook.py） - SpeculativeAlgorithm.supports_mixed_chunk() 新增，EAGLE/EAGLE3/DFLASH/DSPARK 返回 True；spec_registry.py 插件基类默认 False，插件算法必须显式实现能力位。 - _handle_dflash、_handle_dspark、_handle_eagle_family 中删除三处硬编码禁用，改为按能力判断并输出 warning；check_server_args 断言同步放开，报错文案改为指明具体算法不支持。 - 影响：后续新增 spec 算法只需实现一个方法即可加入该组合，默认安全关闭。
2. 混合批次构造（schedule_batch.py 的 mix_with_running） - spec 分支不再直接拼接 running batch 的 out_cache_loc，而是按 tail_base = r.seqlen - 1 从 req_to_token_pool.req_to_token gather 尾部 bonus 槽。 - spec 的 seq_lens 停在提交

长度（bonus 未提交），本 step 提交该 token，尾部行必须 carry base + 1，否则 attention 元数据算出 qo_len = 0。 - 因 spec relay 调度期未解析，merge_batch 会置空 seq_lens_cpu，改为从请求状态重建 CPU 镜像；delta 在 spec 下恒为 -1（两种模式下尾部请求状态都无延迟）。 - 新增 mix_running_indices_cpu，供 overlap 尾部解析在 CPU 侧取 pinned 镜像。

3. 尾部 token relay 与 overlap 晚期绑定（scheduler.py、overlap_utils.py） - 非 overlap 路径：mix 时用新增的 FutureMap.stash_bonus_tokens() 把 output_ids[-1] 写入 running 请求的 pool 行，否则混合输入解析会 gather 到未初始化行。 - overlap 路径：resolve_forward_inputs 在 forward 入口调用新增的 FutureMap.resolve_mixed_spec_tails(batch)，在 publish 栅栏后从 new_seq_lens_buf 读取已提交长度，重建尾部 seq_lens（+1）、out_cache_loc（提交长度处槽位）以及 seq_lens_cpu / seq_lens_sum / prefix_lens；FutureMap 因此持有 req_to_token 引用。 - 设计动机：调度期 CPU 无法得知在飞 verify 的 accept 数，任何调度期 tail 值都必然过期，所以采用“调度期占位 + forward 入口晚期绑定”。
4. 结果处理与 worker 适配（batch_result_processor.py、dflash_worker_v2.py、dspark_worker_v2.py） - mixed spec tail 提交 bonus token 后 req.kv.kv_committed_len += 1，使下一轮 spec prepare 从正确 base 预留槽位；该逻辑依赖上游Req→req.kvbookkeeping移动（commit7d93b30曾因上游移动而崩溃后修复）。 - 两个 dflash 家族 worker 在 forward_batch_generation 中改用 new_seq_lens = batch.seq_lens 生成 next draft input，保证发布的提交长度一致。
5. 测试配套 - 新增 test/registered/spec/test_spec_mixed_chunk.py：每个算法一个 cell（TestEagle3MixedChunk / TestDFlashMixedChunk / TestDSparkMixedChunk），全部走 overlap、chunk 128，覆盖三个 bring-up 失败模式；注册 base-b 阶段、1-gpu-large、约 800s。 - 单元测试适配：test_schedule_batch_out_of_place.py 给 _FakeReq 补齐 seqlen 属性与 spec_algorithm 初始化；test_decode_bookkeeping_ownership.py 记录 mixed-tail 的 kv_committed_len owner。

关键文件：

- python/sglang/srt/managers/overlap_utils.py（模块 重叠调度；类别 source；类型 core-logic；符号 stash_bonus_tokens, resolve_mixed_spec_tails）：核心实现：新增 stash_bonus_tokens 与 resolve_mixed_spec_tails，后者是 overlap 下尾部晚期绑定的关键路径，在 publish 栅栏后重建 seq_lens 与 out_cache_loc。
- python/sglang/srt/managers/schedule_batch.py（模块 批次管理；类别 source；类型 core-logic；符号 mix_with_running）：mix_with_running 改为 spec 感知：按提交长度 gather 尾部 out_cache_loc、重建 seq_lens_cpu、seq_lens 尾部 +1，并新增 mix_running_indices_cpu 字段。
- python/sglang/srt/speculative/spec_info.py（模块 推测解码；类别 source；类型 core-logic；符号 supports_mixed_chunk）：新增 supports_mixed_chunk() 能力位，作为整个降级契约的门控入口，EAGLE/EAGLE3/DFLASH/DSPARK 返回 True。
- python/sglang/srt/arg_groups/speculative_hook.py（模块 参数解析；类别 source；类型 dependency-wiring；符号 _handle_dflash, _handle_dspark, _handle_eagle_family）：删除 dflash/dspark/eagle 三处硬编码禁用 mixed chunk 的逻辑，改为按

supports_mixed_chunk() 能力判断，是本次功能开启的入口。

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _get_new_batch_prefill_raw）：_get_new_batch_prefill_raw 在非 overlap + spec 路径下调用 stash_bonus_tokens 写入尾部 pending token，保证混合输入解析不读垃圾行。
- python/sglang/srt/managers/scheduler_components/batch_result_processor.py（模块 结果处理；类别 source；类型 core-logic；符号 process_batch_result_prefill）：process_batch_result_prefill 为 mixed spec tail 递增 req.kv.kv_committed_len，使下一轮 spec prepare 从正确 base 预留槽位。
- python/sglang/srt/arg_groups/validation_hook.py（模块 参数校验；类别 source；类型 dependency-wiring；符号 check_server_args）：check_server_args 放开 mixed chunk + spec 的硬断言，改为按能力位校验，报错信息更精准。
- python/sglang/srt/speculative/spec_registry.py（模块 插件注册；类别 source；类型 core-logic；符号 supports_mixed_chunk）：插件算法基类新增默认返回 False 的 supports_mixed_chunk，保证第三方算法默认安全关闭。
- python/sglang/srt/speculative/dflash_worker_v2.py（模块 推测解码；类别 source；类型 core-logic；符号 forward_batch_generation）：forward_batch_generation 改用 new_seq_lens 生成 next draft input，保证发布到 relay 的提交长度与 draft 输入一致。
- python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py（模块 推测解码；类别 source；类型 core-logic；符号 forward_batch_generation）：与 dflash worker 同样的 new_seq_lens 修正，保持 DSPARK 与 DFLASH 家族行为一致。
- test/registered/spec/test_spec_mixed_chunk.py（模块 推测解码；类别 test；类型 test-coverage；符号 TestEagle3MixedChunk, TestDFlashMixedChunk, TestDSparkMixedChunk, setUpClass）：新增三算法回归测试矩阵，覆盖 tail 行丢失、relay 未写、overlap 过期状态三个 bring-up 失败模式，是正确性的主要保障。
- test/registered/unit/managers/test_schedule_batch_out_of_place.py（模块 批次管理；类别 test；类型 test-coverage；符号 seqlen）：适配 spec 感知的 mix_with_running，给 _FakeReq 补齐 seqlen 属性与 spec_algorithm 初始化，保证单元测试不因新分支挂掉。
- test/registered/unit/spec/test_decode_bookkeeping_ownership.py（模块 推测解码；类别 test；类型 test-coverage）：记录 mixed-tail 的 kv_committed_len owner，守护 bookkeeping 归属不因新路径漂移。

关键符号：supports_mixed_chunk, stash_bonus_tokens, resolve_mixed_spec_tails, mix_with_running, process_batch_result_prefill, forward_batch_generation, _get_new_batch_prefill_raw, _handle_eagle_family, check_server_args

关键源码片段

python/sglang/srt/managers/overlap_utils.py

核心实现：新增 stash_bonus_tokens 与 resolve_mixed_spec_tails，后者是 overlap 下尾部晚期绑定的关键路径，在 publish 栅栏后重建 seq_lens 与 out_cache_loc。

```
def stash_bonus_tokens(self, indices: torch.Tensor, bonus_tokens: torch.Tensor) -> None:
```

```
        """仅写 output_tokens_buf 行；用于不携带 draft 额外信息的 relay。
        普通 stash() 会按 payload 惰性初始化 spec 缓冲区，这里不需要。"""
        self.output_tokens_buf[indices] = bonus_tokens.to(self.output_tokens_buf.dtype)

    def resolve_mixed_spec_tails(self, batch: ScheduleBatch) -> None:
        """在 overlap 下晚期绑定 spec 混合 batch 的 decode 尾部：调度期的长度
        落后于在飞 step 的 accept 数，因此要在 publish 栅栏之后，从已发布
        的提交长度重建尾部行。"""
        idx = batch.mix_running_indices
        n = int(idx.shape[0])
        if n == 0:
            return
        # 等待最近一次 forward publish 完成，保证 new_seq_lens_buf 已写入
        if self.publish_ready is not None:
            if _is_hip:
                # AMD MI355 上 Event.wait() 拖慢 TPOT，暂时改用同步
                self.publish_ready.synchronize()
            else:
                self.publish_ready.wait()
        fresh = self.new_seq_lens_buf[idx]
        # 尾部 seq_lens 取提交长度 + 1（本 step 提交了 pending 的 bonus
        # token），否则 attention 元数据算出 qo_len = 0，flashinfer 硬崩溃
        seq_lens = batch.seq_lens.clone()
        seq_lens[-n:] = fresh + 1
        batch.seq_lens = seq_lens
        # out_cache_loc 重建：尾部 KV 槽位于提交长度处（预留槽），
        # 按提交长度从 req_to_token gather，而非使用调度期的过期值
        out_cache_loc = batch.out_cache_loc.clone()
        out_cache_loc[-n:] = self.req_to_token[idx.long(), fresh.long()].to(
            out_cache_loc.dtype
        )
        batch.out_cache_loc = out_cache_loc

        # CPU 镜像走独立 D2H 流，避免阻塞调度流；非 CUDA 平台退回 .cpu()
        if self.fwd_prepare_d2h_stream is None or self.publish_ready is None:
            fresh_cpu = fresh.cpu() # bootstrap / non-CUDA
        else:
            self.fwd_prepare_d2h_stream.wait_event(self.publish_ready)
            with torch.get_device_module(self.device).stream(
                self.fwd_prepare_d2h_stream
            ):
                self.new_seq_lens_cpu_pinned.copy_(
                    self.new_seq_lens_buf, non_blocking=True
                )
            self.fwd_prepare_d2h_stream.synchronize()
            fresh_cpu = self.new_seq_lens_cpu_pinned[batch.mix_running_indices_cpu]
        # seq_lens_cpu / seq_lens_sum / prefix_lens 同步 +1，供 extend 元数据使用
        if batch.seq_lens_cpu is not None:
            seq_lens_cpu = batch.seq_lens_cpu.clone()
```

```
            seq_lens_cpu[-n:] = fresh_cpu + 1
            batch.seq_lens_cpu = seq_lens_cpu
            batch.seq_lens_sum = int(batch.seq_lens_cpu.sum())
        batch.prefix_lens = batch.prefix_lens[:-n] + [
            int(x) for x in fresh_cpu.tolist()
        ]
```

python/sglang/srt/managers/schedule_batch.py

mix_with_running 改为 spec 感知：按提交长度 gather 尾部 out_cache_loc、重建 seq_lens_cpu、seq_lens 尾部 +1，并新增 mix_running_indices_cpu 字段。

```
    def mix_with_running(self, running_batch: ScheduleBatch):
        self.forward_mode = ForwardMode.MIXED
        running_bs = running_batch.batch_size()

        for req in running_batch.reqs:
            req._refresh_fill_ids()
            full_len = len(req.full_untruncated_fill_ids)
            req.set_extend_range(full_len - 1, full_len)

        # running 部分的 decode token 存放在 future_map.output_tokens_buf 中
        self.input_ids = None
        self.mix_running_indices = running_batch.req_pool_indices
        self.mix_running_indices_cpu = running_batch.req_pool_indices_cpu
        if not self.spec_algorithm.is_none():
            # spec 下 running batch 不保留每步 out_cache_loc；按提交长度
            # gather 尾部 bonus 槽（overlap 下 forward 入口还会再晚期绑定）
            tail_base = torch.tensor(
                [r.seqlen - 1 for r in running_batch.reqs],
                dtype=torch.int64,
                device=self.seq_lens.device,
            )
            running_out_cache_loc = self.req_to_token_pool.req_to_token[
                running_batch.req_pool_indices.long(),
                tail_base,
            ].to(self.out_cache_loc.dtype)
            # spec relay 调度期未解析，merge_batch 会置空 seq_lens_cpu；
            # 改从请求状态重建 CPU 镜像，供 extend 元数据路径使用
            running_seq_lens_cpu = torch.tensor(
                [int(r.seqlen) for r in running_batch.reqs], dtype=torch.int64
            )
            if self.seq_lens_cpu is None:
                merged_seq_lens_cpu = running_seq_lens_cpu
            else:
                merged_seq_lens_cpu = torch.cat(
                    [self.seq_lens_cpu, running_seq_lens_cpu]
                )
        else:
            # 非 spec：running batch 自带准备好的 seq_lens_cpu，直接拼接；
```

```
            # 若被 spec 分支误覆盖，会让尾部 CPU 长度短一步（qo_len = 0）
            tail_base = None
            running_out_cache_loc = running_batch.out_cache_loc
            merged_seq_lens_cpu = None
        out_cache_loc = torch.cat([self.out_cache_loc, running_out_cache_loc])

        self.merge_batch(running_batch)
        self.out_cache_loc = out_cache_loc
        if merged_seq_lens_cpu is not None:
            self.seq_lens_cpu = merged_seq_lens_cpu
        if tail_base is not None:
            # spec seq_lens 停在提交长度（bonus token 未提交）；本 step 提交它，
            # 尾部必须 carry base + 1，否则 attention 会丢弃这一行
            merged = self.seq_lens.clone()
            merged[-running_bs:] = tail_base + 1
            self.seq_lens = merged

        # overlap 调度下 output_ids 延迟一步；spec 尾部请求状态两种模式都不延迟
        if self.spec_algorithm.is_none():
            delta = 0 if self.enable_overlap else -1
        else:
            delta = -1

        # NOTE: prefix_indices 表示已缓存内容，但 decode 步不做缓存
        self.prefix_lens = self.prefix_lens + [
            len(r.origin_input_ids) + len(r.output_ids) + delta
            for r in running_batch.reqs
        ]
        self.extend_lens = self.extend_lens + [1] * running_bs
        self.extend_num_tokens = self.extend_num_tokens + running_bs
        self.extend_logprob_start_lens = (
            self.extend_logprob_start_lens + [0] * running_bs
        )
        self.is_prefill_only = False
```

评论区精华

PR 没有 review 评论（review_comments_count = 0），issue 评论仅为两条 CI 链接（base pass / extra pass）。最有价值的技术交锋内嵌在 28 个 commit 的 [Fix] 叙事里，相当于作者自审的 review 记录：

- “the spec seq_lens convention zeroed the tail's qo len - a hard crash on flashinfer, silent kv-span truncation elsewhere” —— 同一 bug 在不同 attention 后端的两种表现，flashinfer 直接崩溃、无校验后端静默截断 KV 跨度。
- “stale schedule-time tail state under overlap (0.970 -> 0.390 gsm8k before late binding)” —— 过期 tail 状态的精度灾难，是晚期绑定方案的直接动因。
- “pre-fix 0.655 + pool-leak aborts, post-fix 0.975 clean” —— 非 spec 路径被误伤的量化证据，说明 spec 分支的 seq_lens_cpu 重建必须严格 scope。

- “garbage token id -> embedding OOB -> cublas failure” —— 非 overlap relay 未写 output_tokens_buf 时的完整故障链。
- spec seq_lens 停在提交长度导致 tail qo_len = 0 (correctness): mix 时尾部 seq_lens 统一 carry base + 1（mix_with_running 中 merged[-running_bs:] = tail_base + 1），overlap 下由 resolve_mixed_spec_tails 在 forward 入口按发布长度重建。
- 非 overlap spec 的 relay 未写 output_tokens_buf，混合输入解析读到垃圾 token (correctness): 调度期在 mix 时用 stash_bonus_tokens 把 output_ids[-1] 写入尾部 pool 行；overlap 模式保持发布值不动。
- Overlap 下调度期 tail 状态必然过期，settled-tails 门在负载下永不生效 (design): 采用晚期绑定方案，FutureMap 持有 req_to_token 引用以支持 reserved-slot gather。
- 能力表达方式：硬编码字符串 vs supports_mixed_chunk 能力位 (design): 以能力位 + 默认 False 收敛，speculative_hook 与 validation_hook 统一按能力判断并输出 warning。

风险与影响

- 风险： 1) 核心调度路径变更：mix_with_running 与 process_batch_result_prefill 是每个 prefill/decode 周期必经路径，spec 分支新增 tensor 构造、gather 与 clone；overlap 入口新增一次栅栏等待与 D2H 拷贝，对 TPOT 有额外开销（AMD 路径用 synchronize 规避 MI355 上 Event.wait() 的退化）。 2) seq_lens 约定敏感：spec seq_lens 停在提交长度、mixed step 尾部必须 +1 的约定横跨所有 attention 后端，后端对 qo_len = 0 的容忍度不一，后续后端扩展需回归该组合。 3) 上游 bookkeeping 耦合：kv_committed_len 已迁至 req.kv，本 PR 的尾部 +1 依赖该结构，上游进一步重构会直接击穿（commit 7d93b30 即此类事故的现场修复）。 4) 非 spec 回归风险：seq_lens_cpu 重建与 delta 计算必须严格 scope 到 spec 分支，历史上曾导致非 spec mixed chunk 0.655 + pool-leak。 5) 支持矩阵有限：NGRAM / STANDALONE 与未实现能力位的插件算法会被降级（有 warning），与旧的“直接禁用”语义不同，需要用户注意行为差异。
  - 影响：对用户：开启 --enable-mixed-chunk 后不再需要关闭投机解码，混合负载（长 prefill 与 decode 并存）下可同时获得 chunked prefill 的调度收益与 spec 的 decode 加速；不支持的算法组合会得到明确的 warning 或参数校验报错。对系统：影响调度器批次构造、overlap 前向解析、批次结果处理三条核心路径，以及 EAGLE3 / DFLASH / DSPARK 三个 worker 的输入构建。对团队：CI 新增约 800s 的 base-b 用例（1-gpu-large），并确立“算法能力位”这一可扩展门控模式，后续插件算法只需实现 supports_mixed_chunk() 即可接入。
  - 风险标记：核心调度路径变更，seq_lens 约定敏感，上游 bookkeeping 耦合，多 attention 后端兼容风险

关联脉络

- PR #36897 Decouple speculative draft capacity from runtime state: 同属 spec 子系统，改动 spec_registry.py、runtime_context 等与本 PR 重叠的区域，本 PR 在其上扩展能力位门控。
- PR #37164 [mem_cache] Move mamba state and retraction_backup into ReqKvInfo: mem_cache bookkeeping 系列把状态收拢到 req.kv / ReqKvInfo；本 PR commit

7d93b30 明确记载“Upstream moved kv_committed_len onto req.kv”导致 mixed-tail +1 曾崩溃，两者强耦合。

- PR #35588 [Bugfix] Fix full prefill CUDA graph padding and EAGLE capture: 与 mixed/extend 家族的 CUDA graph 回放同源；本分支早期曾有 MixedCudaGraphRunner 重构（后放弃），与 prefill runner 的 MIXED→EXTEND 归一化思路相关。