PR #51538 完整报告

vllm-project/vllm

[Bugfix] Make DSV4 sparse MLA work end-to-end for plain decode, MTP, and DSpark

合并时间：2026-08-15 22:01

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51538

执行摘要

- 一句话：修复 DSV4 稀疏 MLA 七处缺陷，打通普通解码、MTP、DSpark 端到端
- 推荐动作：值得精读。重点关注三处：一是 persistent_topk.cuh 死锁根因（uint32 溢出 + host/device 两侧判定不一致导致 inter-CTA barrier 悬挂）——这是一个教科书级的 CUDA 内核并发陷阱；二是 workspace lane 的 (ubatch, lane) 双维分配设计与 ContextVar 用法，以及 WoosukKwon 对必要性的质疑；三是 TheEpicDolphin 提出的 " captured 区域中间 tensor 来自共享池、replay 前会被覆盖 " 的分析方法，对理解 CUDA graph 与工作区生命周期很有价值。

功能与动机

PR body 明确说明：DeepSeek-V4-Flash-0731 could not run reliably through the SM120 sparse MLA backend，并列出阻塞的七项缺陷；其中 commits 6-7 修复的 MTP 挂起是 main 上已存在、与 DSpark 无关的缺陷，已单独归档为 issue #51593（DeepSeek-V4-Flash MTP hangs after the batch drains to 3 requests）。用户侧佐证来自 SamSaffron 的 H200 报告：We are seeing duplicate blocks of reasoning and corruption on a dual H200，移除 dspark 的 speculative-config 才能绕开，说明该缺陷在真实生产场景有直接影响。

实现拆解

按 PR body 的 7 个修复点组织实现拆解。

1. 分离逻辑窗口与填充宽度（SWA width 修复）：vllm/v1/attention/backends/mla/compressor_utils.py 新增 get_dspark_swa_index_width，把 window_size + num_speculative_tokens 按 64 的倍数向上取整（K=5 时 192 代替 256），匹配内核 64 条目 tile 并带来 ≥8 token 时 13-16% 性能提升。DeepseekSparseSWAMetadata 新增 decode_swa_width 字段（vllm/v1/attention/backends/mla/sparse_swa.py），vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py 的 _build_sparse_index_metadata 改为按该宽度 reshape（修复前按 window_size reshape 直接崩溃），vllm/models/deepseek_v4/amd/rocm.py 的 ragged 路径同步适配；vllm/models/deepseek_v4/common/ops/cache_utils.py 的 build_flashinfer_mixed_sparse_indices 从索引张量推导 SWA_INDEX_WIDTH，与 WINDOW_SIZE 解耦。这是数据契约层面最核心的改动，测试 test_flashinfer_sparse_index_preserves_logical_window 与 test_flashinfer_mixed_sparse_indices_separates_window_and_padded_width 精确锁定

该契约。

2. DSpark 独立 workspace lane：vllm/v1/worker/workspace.py 用 ContextVar 实现 use_workspace_lane 上下文管理器，WorkspaceManager 从每 ubatch 单 buffer 扩展为 (ubatch, lane) 双维 _current_workspaces，resize 只作用于当前槽位；vllm/v1/worker/gpu_worker.py 新增 _num_workspace_lanes，仅 V2 Model Runner + DSpark 时返回 2，其余配置保持 1 个 lane；vllm/v1/worker/gpu/model_runner.py 在 draft 模型 load_model、_dummy_run 的 propose、capture_model、sample_tokens 四处调用点包裹 use_workspace_lane(self._draft_workspace_lane)。原因是 DSpark 的 target 与 draft CUDA graph 并发持有 workspace 视图，单 buffer 下某一侧 resize 会让另一侧的 live tensor 成为孤儿。新增 tests/v1/worker/test_workspace.py 4 个用例锁定 lane 计数、隔离、与 ubatch 组合行为及非法 lane 校验。
3. 图回放与 draft KV 安全：vllm/v1/worker/gpu/spec_decode/dflash/speculator.py 的 prepare_dflash_inputs 让 rejected context suffix 落到 PAD_SLOT_ID，replacement query 从最后接受位置前进，杜绝 draft KV 写入物理块 0（null block）；sample_idx_mapping 以 -1 标记惰性行，避免 capture 期间 padding 行 scatter 进 request slot 0（main 已独立采纳 fill_(-1)，本 PR 保留 torch.full(..., -1) 初始化与 PAD_SLOT_ID 路由）。新增 tests/v1/spec_decode/test_dflash_prepare_inputs.py 的两个用例分别固定 rejected suffix 排除与 null block 保护行为。
4. MXFP4 SwiGLU 参数与 SM120 gate：vllm/model_executor/layers/fused_moe/experts/flashinfer_cutlass_moe.py 删除对 mxfp4 权重无条件注入 GPT-OSS 激活常量（gemm1_alpha=1.702、gemm1_beta=1.0、gemm1_clamp_limit=7.0）的逻辑，改从 quant_config 取参；vllm/utils/flashinfer.py 新增 has_flashinfer_sparse_mla_sm120_config，读取 FlashInfer 的 _DECODE_DSV4_DISPATCH 表；flashinfer_sparse.py::__init__ 在初始化期按 (padded_heads, required_topk) 精确校验并给出包含 flashinfer-ai/flashinfer#4380 提示的可操作 RuntimeError，把不透明的首轮 decode 启动失败前置。test_ocp_mx_moe.py 新增两个用例分别固定 GPT-OSS 常量仍由 quant config 提供、FlashInferExperts 参数缺省时为 None。
5. MTP 负 context 长度与内核夹紧：vllm/v1/attention/backends/mla/indexer.py 在 uniform 与 native 两条 spec-decode 路径对 per-token context 长度 clamp 下限 0（next_n > 1 时 padded slot 计算得 0 - 2 + 0 + 1 = -1）；csrc/libtorch_stable/persistent_topk.cuh 与 cooperative_topk.cuh 把行长度在比较 RADIX_THRESHOLD 前先按 min(stride, max_seq_len) 夹紧，杜绝负数经 uint32 读成 ~4.29e9 后误入 multi-CTA radix 路径的 inter-CTA barrier 悬挂，同时消除越过本行宽度的越界读。PR 自带 drain 形态复现器：未修复构建 30-180 秒内 4 卡 100% 利用率挂死，commit 6 单独、commit 7 单独（commit 6 回退）均通过，证明内核守卫并非死代码。
6. 测试与验证配套：除上述新增测试外，还扩展了 tests/kernels/attention/test_flashmla_sparse.py（逻辑窗口与填充宽度分离）、tests/v1/attention/test_flashinfer_sparse_mla_sm120_api.py（DSV4 能力检查与 required_topk 跟踪）、tests/kernels/test_compressor_kv_cache.py（get_dspark_swa_index_width）。PR 测试计划总计 71 passed / 141 skipped，并在 Blackwell（sm103）上跑通全部 test_ocp_mx_moe.py。

关键文件：

- vllm/v1/worker/workspace.py（模块 工作区管理；类别 source；类型 core-logic；符号 use_workspace_lane, WorkspaceManager, _ensure_workspace_size, init_workspace_manager）：引入 (ubatch, lane) 双维工作区分配与 ContextVar 的 use_workspace_lane，是 DSpark target/draft 两份 CUDA graph 并发持有 workspace 视图而不互相 orphan 的关键机制；resize 逻辑精确到单槽位，避免兄弟 lane 拿到已释放的孤儿 tensor。
- vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py（模块 稀疏注意力；类别 source；类型 data-contract；符号 _required_sm120_sparse_topk, init, _build_sparse_index_metadata）：修复 decode_swa_indices 按 window_size 而非实际宽度 reshape 导致的崩溃，并新增 _required_sm120_sparse_topk 在模型初始化时用精确 (num_q_heads, top_k) 做 SM120 DSV4 特化能力 gate，是数据契约与启动路径的双重修复。
- vllm/v1/worker/gpu/model_runner.py（模块 模型执行器；类别 source；类型 core-logic；符号 GPUModelRunner, _draft_workspace_lane）：workspace lane 机制真正接入 MRV2 执行路径的位置：draft 模型 load、propose（dummy 与真实）、capture 四处调用点包裹 use_workspace_lane，且 _draft_workspace_lane 仅 DSpark 时为 1。
- csrc/libtorch_stable/persistent_topk.cuh（模块 TopK 内核；类别 source；类型 core-logic；符号 persistent_topk_kernel）：MTP drain 死锁的根因修复：lengths 先转 uint32 再比 RADIX_THRESHOLD，负数变 ~4.29e9 误入 cooperative 路径，组长 CTA 在 inter-CTA barrier 永久等待（cuda-gdb 确认 wait_ge 空转）。加 min(stride, max_seq_len) 夹紧后同时消除越界读。
- vllm/v1/worker/gpu_worker.py（模块 GPU 工作器；类别 source；类型 core-logic；符号 _num_workspace_lanes, init_device）：新增 _num_workspace_lanes 决定 lane 数量：仅 V2 Model Runner 且 DSpark 时返回 2，其余配置保持单 lane，确保非 DSpark 路径行为完全不变。
- vllm/model_executor/layers/fused_moe/experts/flashinfer_cutlass_moe.py（模块 MoE 专家；类别 source；类型 data-contract；符号 FlashInferExperts）：移除对所有 mxfp4 权重无差别注入 GPT-OSS SwiGLU 常量的逻辑，改从 quant_config 取参；DeepSeek V4 在 --moe-backend flashinfer_cutlass 下曾因错误常量导致生成崩塌，这是易被忽视的隐性回归。
- vllm/v1/attention/backends/mla/indexer.py（模块 索引器；类别 source；类型 core-logic）：MTP 场景 padded decode slot（seq_len=0, next_n>1）会产生负数 per-token context 长度，两条 spec-decode 路径均 clamp 到 0，避免 -1 被 uint32 读成 ~4.29e9 触发后续内核异常。
- vllm/utils/flashinfer.py（模块 FlashInfer 工具；类别 source；类型 core-logic；符号 has_flashinfer_sparse_mla_sm120_config）：新增 has_flashinfer_sparse_mla_sm120_config，通过检查 FlashInfer 内部 _DECODE_DSV4_DISPATCH 表是否含精确 (num_q_heads, top_k)，把不透明的首轮 decode 启动失败提前到初始化报错。

- vllm/v1/worker/gpu/spec_decode/dflash/speculator.py（模块 推测解码；类别 source；类型 core-logic；符号 prepare_dflash_inputs）：修复 draft KV 写入物理块 0（null block）与 rejected suffix 污染 draft KV 的问题，sample_idx_mapping 用 -1 标记惰性行；配合新增测试锁定 prepare_dflash_inputs 的行为。
- vllm/v1/attention/backends/mla/compressor_utils.py（模块 注意力后端；类别 source；类型 core-logic；符号 get_dspark_swa_index_width）：新增 get_dspark_swa_index_width，把 window_size + num_speculative_tokens 向上取整到 64 的倍数；这是 DSpark 宽度从 256 降到 192 的出处，性能收益 +13-16%。
- tests/v1/worker/test_workspace.py（模块 工作区管理；类别 test；类型 test-coverage；符号 test_workspace_lane_count_is_dspark_only, test_workspace_lanes_do_not_alias_and_restore_context, test_workspace_lanes_compose_with_ubatches, test_workspace_lane_validation）：新增 4 个用例精确锁定 lane 数量规则（仅 V2+DSpark 为 2）、lane 间不 alias、与 ubatch 组合、非法 lane 校验，是 workspace lane 机制的回归防线。
- tests/kernels/moe/test_ocp_mx_moe.py（模块 MoE 测试；类别 test；类型 test-coverage；符号 test_gpt_oss_quant_config_supplies_clamped_swiglu_params, test_flashinfer_experts_swiglu_params_follow_quant_config）：两个新测试分别固定 GPT-OSS quant config 仍提供 1.702/1.0/7.0 常量、FlashInferExperts 的 SwiGLU 参数跟随 quant config，防止 MXFP4 参数修复被回归。
- vllm/v1/attention/backends/mla/sparse_swa.py（模块 稀疏注意力；类别 source；类型 data-contract；符号 DeepseekSparseSWAMetadata）：DeepseekSparseSWAMetadata 新增 decode_swa_width 字段，是 SWA 宽度数据契约的载体，FlashInfer 与 ROCm 两条路径都依赖它。
- vllm/models/deepseek_v4/common/ops/cache_utils.py（模块 KV 缓存工具；类别 source；类型 data-contract；符号 build_flashinfer_mixed_sparse_indices）：build_flashinfer_mixed_sparse_indices 从索引张量推导 padded width 并作为 SWA_INDEX_WIDTH 传给内核，与 WINDOW_SIZE 解耦，是数据契约层的核心改动。

关键符号：use_workspace_lane, _num_workspace_lanes, _required_sm120_sparse_topk, has_flashinfer_sparse_mla_sm120_config, get_dspark_swa_index_width, prepare_dflash_inputs, persistent_topk_kernel, _build_sparse_index_metadata

关键源码片段

vllm/v1/worker/workspace.py

引入 (ubatch, lane) 双维工作区分配与 ContextVar 的 use_workspace_lane，是 DSpark target/draft 两份 CUDA graph 并发持有 workspace 视图而不互相 orphan 的关键机制；resize 逻辑精确到单槽位，避免兄弟 lane 拿到已释放的孤儿 tensor。

```
# _workspace_lane 用 ContextVar 记录当前执行上下文的工作区 lane。
# 默认 0 是 target 模型；DSpark 的 draft 模型通过 use_workspace_lane(1)
# 切到独立 buffer，避免 target/draft 两份 CUDA graph 并发持有视图时，
# 某一侧 resize 把另一侧仍被引用的旧 tensor 提前释放。
_workspace_lane: ContextVar[int] = ContextVar("vllm_workspace_lane", default=0)
```

```python
@contextmanager
def use_workspace_lane(lane: int) -> Iterator[None]:
    """在指定 lane 内进行后续 workspace 分配，退出时恢复上一 lane。"""
    if lane < 0:
        raise ValueError(f"Workspace lane must be non-negative, got {lane}.")
    token = _workspace_lane.set(lane)
    try:
        yield
    finally:
        _workspace_lane.reset(token)


class WorkspaceManager:
    def __init__(self, device, num_ubatches=None, num_lanes=1):
        # ... 构造时完成 lane 数校验
        if num_lanes < 1:
            raise ValueError(f"num_lanes must be at least one, got {num_lanes}.")
        self._num_lanes = num_lanes
        # 每个 (ubatch, lane) 组合一个独立 buffer；DSpark 下 target 与
        # draft 各占一个 slot，互不挤占。
        self._current_workspaces: list[torch.Tensor | None] = [None] * (
            self._num_ubatches * self._num_lanes
        )
        # ...

    def _ensure_workspace_size(self, required_bytes: int) -> torch.Tensor:
        # ...
        ubatch_id = dbo_current_ubatch_id()
        lane = _workspace_lane.get()
        if lane >= self._num_lanes:
            raise RuntimeError(
                f"Workspace lane {lane} is not configured; manager has "
                f"{self._num_lanes} lane(s)."
            )
        workspace_id = ubatch_id * self._num_lanes + lane
        current_workspace = self._current_workspaces[workspace_id]
        current_size = self._workspace_size_bytes(current_workspace)

        if current_size < required_bytes:
            # 只 resize 当前 (ubatch, lane) 槽位，其他槽位延迟到下次
            # get_simultaneous 再分配。如果一次性 resize 所有槽位，
            # 仍持有旧视图的兄弟 lane 会拿到已释放的孤儿 tensor。
            self._current_workspaces[workspace_id] = None
            del current_workspace
            torch.accelerator.empty_cache() # 让 caching allocator 复用释放段
            self._current_workspaces[workspace_id] = torch.empty(
                (required_bytes,), dtype=torch.uint8, device=self._device
            )
            # ...
```

```
        return current_workspace
```

vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py

修复 decode_swa_indices 按 window_size 而非实际宽度 reshape 导致的崩溃，并新增 _required_sm120_sparse_topk 在模型初始化时用精确 (num_q_heads, top_k) 做 SM120 DSV4 特化能力 gate，是数据契约与启动路径的双重修复。

```
def _required_sm120_sparse_topk(vllm_config: VllmConfig, window_size: int) -> int:
    """返回该模型在 SM120 上需要的 DSV4 SWA 特化 top_k。

    非因果（DSpark 草稿）场景下，索引宽度要把 window_size 加上
    草稿 token 数，并按内核 64 条目 tile 向上取整；普通因果解码
    直接使用 window_size。
    """
    if not vllm_config.attention_config.use_non_causal:
        return window_size
    speculative_config = vllm_config.speculative_config
    if speculative_config is None:
        return window_size
    return get_dspark_swa_index_width(
        window_size,
        speculative_config.num_speculative_tokens,
    )


class DeepseekV4FlashInferMLASparseBackend(DeepseekV4SparseMLABackend):
    def _build_sparse_index_metadata(self, ...):
        # 修复前这里用 self.window_size 做 reshape，非因果草稿批次
        # 分配的是 192 宽的索引，按 128 窗口 reshape 直接崩溃；
        # 现在以 metadata 上携带的稠密宽度 decode_swa_width 为准。
        decode_swa_indices = swa_metadata.decode_swa_indices.reshape(
            num_decode_tokens, swa_metadata.decode_swa_width
        )
        # ...

    def __init__(self, vllm_config: VllmConfig, *args, **kwargs) -> None:
        # ...
        required_topk = _required_sm120_sparse_topk(vllm_config, self.window_size)
        # 老版本 FlashInfer 可能暴露 decode API 却不含本配置需要的 DSV4
        # 特化形状；在初始化期用精确 (num_q_heads, top_k) 检查，把
        # 首个 decode 的不透明启动失败换成可操作的报错。
        if not has_flashinfer_sparse_mla_sm120_config(self.padded_heads, required_topk):
            raise RuntimeError(
                "FLASHINFER_MLA_SPARSE_DSV4 on SM120 requires a FlashInfer "
                "DSV4 sparse MLA decode specialization for "
                f"(num_q_heads={self.padded_heads}, top_k={required_topk}). "
                "Install a FlashInfer build containing "
                "flashinfer-ai/flashinfer#4380."
            )
```

```
        # ...
```

csrc/libtorch_stable/persistent_topk.cuh

MTP drain 死锁的根因修复：lengths 先转 uint32 再比 RADIX_THRESHOLD，负数变 ~4.29e9 误入 cooperative 路径，组长 CTA 在 inter-CTA barrier 永久等待（cuda-gdb 确认 wait_ge 空转）。加 min(stride, max_seq_len) 夹紧后同时消除越界读。

```
// 修复前：先做 uint32 转换再比较阈值，负长度（如 -1）变成 ~4.29e9，
// 把本行错误地推上 multi-CTA radix 路径；而 cta_in_group != 0 的
// 提前退出由 host 侧 max_seq_len 决定，与设备侧 per-row 分支不一致，
// 组长 CTA 在 inter-CTA barrier 上等一个早已退出的同伴，内核永不退休，
// 异步输出复制事件不触发，引擎整体挂起。
// 修复后：任何分支决策前先把行长度夹紧到 min(stride, max_seq_len)，
// 顺带消除了越过本行宽度读取下一行的越界读。
// 以下为按 PR 描述整理的关键判定逻辑示意，非原始逐行源码。
uint32_t clamped = min(static_cast<uint32_t>(row_len),
                       min(stride, static_cast<uint32_t>(max_seq_len)));
if (clamped > RADIX_THRESHOLD) {
    // 进入 multi-CTA cooperative radix 路径（仅在真实长行时触发）
}
```

评论区精华

本次 review 的争论焦点集中在三处。

draft_gumbel_pos 的拆分之争：TheEpicDolphin 的 CHANGES_REQUESTED 核心是要求移除 DSpark 专用的 draft_gumbel_pos 改动：The other speculators are still using the old approach, so now there's inconsistency between DSpark vs MTP, EAGLE, etc. I think this gumbel draft position change deserves it's own, separate PR which migrates all the speculators together. lucifer1004 照做并揭露了更深层问题：the rejection sampler keys both its acceptance uniform and its recovery Gumbel noise by token position, so draft proposals and verification currently draw from the same Philox counter range and the sampler isn't unbiased. That applies to MTP and EAGLE as well, not just DSpark——即所有 speculator 的采样都不是无偏的，需要单独 PR 统一迁移。

captured backbone output 的保留价值：TheEpicDolphin 认为 captured 区域内的中间 tensor 来自共享池，replay 时会被 forward 输出覆盖，保留无意义；lucifer1004 确认移除并补充发现：该 list 从不清理，DFlash 是唯一实际 append 的路径，每个捕获图形形状会 pin 一份 backbone activation，阻止 allocator 复用这些块，所以移除反而是一个 DFlash 小内存收益而非中性回退。

workspace lane 必要性的质疑：合并者 WoosukKwon 在 issue 评论中表示 I feel the workspace lane change in MRV2 might not be necessary 🤔，但该质疑在已提供的材料中未见后续回应或结论，最终 PR 合入说明该设计被接受但质疑未完全闭环。

- draft_gumbel_pos 改动要求拆分，统一所有 speculator (design): lucifer1004 移除该改动，并揭露更深层问题：rejection sampler 用 token position 同时作为 acceptance uniform 与 recovery Gumbel noise 的 key，所有 speculator 的 draft 与验证都从同一

Philox counter 范围取样，sampler 并非无偏，将另开 PR 统一迁移。

- _captured_backbone_outputs 保留是否必要 (question): lucifer1004 确认移除，并发现该 list 从不清理，DFlash 每个捕获形状会 pin 一份 backbone activation，移除反而是 DFlash 的内存收益；同时说明 DSpark 路径实际上覆盖了 _generate_draft，该守卫从未保护目标路径。
- workspace lane 改动的必要性被合并者质疑 (design): 已提供的材料中没有后续回应或明确结论，PR 最终合入说明该改动被接受，但质疑未完全闭环。
- SM121 双节点 TP=2 社区验证 (testing): 提供额外平台验证数据点，覆盖了 PR body 未明确提及的 (32, 192) 形状。
- H200 用户侧 corruption 报告 (other): 用户侧佐证了本 PR 修复缺陷的真实影响；PR 合入后需在 H200 场景跟进验证。

风险与影响

- 风险：
  1. MRV2 核心路径变更：vllm/v1/worker/workspace.py 与 vllm/v1/worker/gpu/model_runner.py 是 V1 引擎的分配与执行热路径，workspace lane 改动影响所有 V2 配置（虽然非 DSpark 仍为单 lane），WoosukKwon 曾质疑其必要性；若 DSpark 图捕获顺序改变，双 lane 的额外常驻显存成本可能被放大。
  2. 内核共享面回归风险：csrc/libtorch_stable/persistent_topk.cuh 是多个 top-k 消费者共享的内核，#49139 修复的是同一文件的 radix histogram 复用缺陷，两处改动交互可能覆盖不同的行长分界，clamp 到 min(stride, max_seq_len) 后需确认不改变长行（越过 RADIX_THRESHOLD）的既有行为。
  3. MXFP4 参数回归：flashinfer_cutlass_moe.py 从无条件注入变成跟随 quant_config，GPT-OSS 的常量现在依赖其 quant config 正确提供，两个新增测试覆盖了该契约，但 mxfp4+mxfp8 组合的 fake_input_scale 条件变化仍需留意。
  4. FlashInfer 版本耦合：SM120 gate 读取 FlashInfer 内部 _DECODE_DSV4_DISPATCH 符号，FlashInfer 版本变化可能导致探测失效或误报，且要求用户安装含 flashinfer#4380 的构建。
  5. 测试环境局限：lucifer1004 在 sm103 上验证 OCP MX MoE 测试，而 CI 任务跑在 sm100 上；SM121 的验证来自社区用户 Mirrdhyn，非官方 CI 覆盖。 - 影响：对用户：DeepSeek-V4-Flash-0731 在 SM120/SM121（Blackwell）上三种 decode 模式可端到端使用，MTP 服务不再在 batch drain 时挂死（issue #51593）；H200 上用户报告的 DSpark 重复推理块与 corruption 问题有望随之解决。对系统：workspace lane 增加少量显存常驻，192 宽度对齐换取 13-16%（≥8 token）吞吐收益；persistent_topk 的夹紧同时去除了潜在的越界读。对团队：确立了 " 能力 gate 前置到模型初始化 " 的 FlashInfer 集成模式，并在 review 中推动将 draft_gumbel_pos 拆为统一迁移所有 speculator 的独立 PR，避免 DSpark 单独偏离既有约定。 - 风险标记：核心路径变更，内核死锁修复，平台相关，版本耦合，隐性数据契约，回归覆盖风险

关联脉络

- PR #51395 [Bugfix][SM120][MLA] Disable dense prefill for FlashInfer sparse MLA: 同一 SM120 稀疏 MLA 后端功能线，同样修改 flashinfer_mla_sparse_sm120 相关注意力后端与测试，与本 PR 的 SM120 gate 和 DSV4 特化形状修复紧密相关。
- PR #52188 [Spec decode] Support Kimi-K3 DCP with DSpark: 同一 DSpark 推测解码功能线，修改 flashinfer_mla.py、mla_attention.py 等注意力后端与 speculative 配置，与本 PR 的 DSpark 解码路径修复互补。
- PR #52197 Support DSpark configs with architectures=DSparkDraftModel + model_type=qwen3: 同一 DSpark 配置处理线，修改 vllm/config/speculative.py，本 PR 的 _required_sm120_sparse_topk 同样依赖 speculative_config 的 num_speculative_tokens，两者在 DSpark 配置口径上相关。
- PR #51855 [K3] support recoverssm for K3: Kimi-K3 推测解码恢复路径，与本 PR 同属 speculative-decoding + mrv2 功能线，且都涉及模型状态与注意力元数据的数据契约扩展。