PR #31847 完整报告

sgl-project/sglang

[spec decoding] support inkling dspark

合并时间：2026-08-11 04:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31847

执行摘要

- 一句话：为 Inkling 打通 DSPARK 投机解码并新增融合 Triton 内核
- 推荐动作：值得精读，尤其是三个设计决策：(1) fast path + 严格 eligibility 检查并静默回退的模式（dspark.py 的 _build_fused_kv_write_bundle）；(2) 在 kernel 内用 HAS_COMMIT_LENS 处理截断写入，替代 host 端 mask 尾部列的做法；(3) muP folded head 的除法位置选择——在 base logits 处除一次而不是在加载权重时改写权重。建议阅读顺序：dspark.py 的 bundle 构建 → fused_kv_write.py 的 kernel → mamba_state_scatter_triton.py 的 meta 表设计。

功能与动机

PR body 仅使用模板，动机需从源码注释推断：让 DSPARK 投机解码支持 Inkling 这类 muP 训练的模型。源码给出三条关键线索：(1) Inkling 真 vocab 200058、padding 后 201024、mask token 200064，mask token 需要 embedding 行而不是 tokenizer 条目；(2) muP 目标训练时使用折叠 head（权重预除以 logits_mup_width_multiplier），serving 挂载未折叠 head，因此 compute_base_logits 必须恰好除一次以匹配 markov bias / confidence head 训练的尺度；(3) 融合 kernel 的动机是减少 decode 热路径的 kernel launch 与中间张量开销。

实现拆解

实现按以下 5 步展开：

1. Inkling / muP 适配（dspark_worker_v2.py + dspark.py）：DSparkWorkerV2.__init__ 中把 mask token 边界从真实 vocab 改为 embedding 行数（取 padded_vocab_size 优先），并向 draft_model 注入 logits_mup_width_multiplier（来自 target hf config，非 muP 目标为 None）；DSparkDraftMixin.compute_base_logits 在乘 lm_head 前先除该 multiplier，确保 base logits 与 markov bias / confidence head 训练尺度一致。同时缓存 _linear_accept_index_cache 并为 mambaish 目标设置 _target_is_mambaish。
2. DFlash 热路径 kernel 化（dflash_utils.py + dflash.py）：compute_dflash_correct_drafts_and_bonus 在 CUDA 上改用新增的 _fused_correct_drafts_and_bonus_kernel 单 launch 计算接受长度与 bonus token（tl.min(tl.where(eq, BLOCK, offs)) 求首个失配位置），CPU 分支保留原 cumprod 实现；新增 table_qk_norm_rope_ 及 _table_qk_norm_rope_kernel，原地对融合 QKV 张量做 QK RMSNorm + 查表 neox RoPE，DFlashAttention.forward 通过 use_table_qk_norm_rope 门控（非 NPU、bf16、neox 全维度旋转）接入该路径。

3. DSPARK 跨层 fused KV 写入（dspark.py + 新文件 fused_kv_write.py）：DSparkDraftMixin 新增 _fused_kv_write_bundle / _build_fused_kv_write_bundle，以严格 eligibility 检查（每层无 bias、无 k/v scale、bf16 连续 stride、共享 rotary 表与 eps）为前提，把所有层的 KV 投影权重拼接为一次 F.linear；随后新增 fused_kv_norm_rope_write 在单 kernel 内完成逐层 K 的 RMSNorm、查表 neox RoPE 与 K/V 写池，并通过 HAS_COMMIT_LENS 在 kernel 内按 commit_lens 截断写入，替代 host 端把尾部列 mask 成 -1 的做法。
4. Mamba 状态散射融合（mamba_state_scatter_triton.py）：新增 _fused_conv_window_scatter_multi_kernel 与 fused_conv_window_scatter_multi，用 meta 表（每类型记录 src/dst 指针、各维 stride、block 区间）单 launch 处理最多 8 组 (dst, src) 卷积窗口对与两套请求索引（accept commit + interval track）；新增 fused_commit_track_indices 在单 kernel 内同时算出 last-correct 索引与跨 interval 的 track 步数。scatter_mamba_states_after_mtp_verify 优先走融合路径，条件不满足时回退到逐对 fused_conv_window_scatter_with_mask。
5. 测试与 CI 配套：本 PR 未附带直接单元测试，依赖现有 CI（PR Test 与 PR Test Extra 均通过）与 run-ci / run-ci-extra 标签；_commit_target_mamba_states_after_verify 同步多传 req_pool_indices 以配合上游接口变化。

关键文件：

- python/sglang/srt/speculative/dflash_utils.py（模块 投机解码；类别 source；类型 core-logic；符号 compute_dflash_correct_drafts_and_bonus, fused_correct_drafts_and_bonus_kernel, _table_qk_norm_rope_kernel, table_qk_norm_rope）：DFlash 投机解码工具的核心改造：verify 阶段 correct/bonus 计算在 CUDA 上改为单 launch Triton kernel，并新增融合 QK RMSNorm + 查表 neox RoPE 的 table_qk_norm_rope_，供 DFlash 与 DSPARK 模型复用。
- python/sglang/srt/models/dspark.py（模块 草稿模型；类别 source；类型 core-logic；符号 compute_base_logits, _fused_kv_write_bundle, _build_fused_kv_write_bundle, write_target_hidden_kv）：DSPARK 草稿模型的核心改造：新增 logits_mup_width_multiplier 适配 Inkling 的 muP 折叠 lm_head，并新增跨层 fused KV write bundle，把逐层 norm + rope + 写池合并为一次 GEMM + 一次 kernel launch。
- python/sglang/kernels/ops/mamba/mamba_state_scatter_triton.py（模块 散射内核；类别 infra；类型 core-logic；符号 fused_conv_window_scatter_multi, _conv_multi_build_meta, _conv_multi_eligible, _fused_conv_window_scatter_multi_kernel）：Mamba 状态散射基础设施升级：新增单 launch 的多类型卷积窗口散射 fused_conv_window_scatter_multi 与合并 commit / track 索引计算的 fused_commit_track_indices，减少 decode 路径 kernel launch 次数。
- python/sglang/srt/models/dflash.py（模块 模型层；类别 source；类型 core-logic；符号 use_table_qk_norm_rope, forward）：DFlash 模型 forward 接入融合 QK norm + RoPE 路径，以 use_table_qk_norm_rope 门控（非 NPU、bf16、neox 全维度旋转），其余情况保持原路径。
- python/sglang/kernels/ops/speculative/dspark/fused_kv_write.py（模块 KV 写入；类别 infra；类型 core-logic；符号 _fused_kv_norm_rope_write_kernel,

fused_kv_norm_rope_write）：新增 DSPARK fused KV 写入 kernel 模块：fused_kv_norm_rope_write 在一个 kernel 内完成逐层 K 的 RMSNorm、查表 neox RoPE 和 K / V 写池，且支持 in-kernel 的 commit_lens 截断（替代 host 端把尾部列 mask 成 -1）。

- python/sglang/srt/speculative/dspark_components/dspark_worker_v2.py（模块 执行器；类别 source；类型 core-logic；符号 logits_mup_width_multiplier, _target_is_mambaish, _commit_target_mamba_states_after_verify）：DSPARK worker 初始化适配 Inkling：mask token 边界改用 padded embedding 行数，向 draft model 注入 target 的 logits_mup_width_multiplier，并补充 mambaish 目标识别与 req_pool_indices 传递。

关键符号：compute_dflash_correct_drafts_and_bonus, fused_correct_drafts_and_bonus_kernel, table_qk_norm_rope, _table_qk_norm_rope_kernel, compute_base_logits, _fused_kv_write_bundle, _build_fused_kv_write_bundle, write_target_hidden_kv, fused_kv_norm_rope_write, _fused_kv_norm_rope_write_kernel, fused_conv_window_scatter_multi, fused_commit_track_indices, scatter_mamba_states_after_mtp_verify

关键源码片段

python/sglang/srt/speculative/dflash_utils.py

DFlash 投机解码工具的核心改造：verify 阶段 correct/bonus 计算在 CUDA 上改为单 launch Triton kernel，并新增融合 QK RMSNorm + 查表 neox RoPE 的 table_qk_norm_rope_，供 DFlash 与 DSPARK 模型复用。

```
# 融合 QK RMSNorm + 查表 neox RoPE 的 Triton kernel（dflash_utils.py 新增）
# 关键设计：cos / sin 从与未融合路径相同的 rotary 表中读取，
# 避免 theta 重算 kernel 在大位置上的角度漂移；V 列完全不动。
@triton.jit
def _table_qk_norm_rope_kernel(
    qkv_ptr,
    q_weight_ptr,
    k_weight_ptr,
    cos_sin_ptr,
    pos_ptr,
    row_stride,
    q_size,
    NHQ: tl.constexpr, # Q 头数
    D: tl.constexpr, # head_dim
    EPS: tl.constexpr,
):
    t = tl.program_id(0).to(tl.int64)
    h = tl.program_id(1)
    pos = tl.load(pos_ptr + t).to(tl.int64)

    HALF: tl.constexpr = D // 2
    half_ar = tl.arange(0, HALF)
```

```
    d_ar = tl.arange(0, D)
    # 查表取 cos / sin，与 eager 路径使用同一 rotary 表
    cos = tl.load(cos_sin_ptr + pos * D + half_ar).to(tl.float32)
    sin = tl.load(cos_sin_ptr + pos * D + HALF + half_ar).to(tl.float32)

    # h < NHQ 是 Q 头，否则是 K 头；Q / K 各自有独立的 RMSNorm 权重
    is_q = h < NHQ
    col0 = tl.where(is_q, h * D, q_size + (h - NHQ) * D).to(tl.int64)
    w_ptr = tl.where(is_q, q_weight_ptr.to(tl.int64), k_weight_ptr.to(tl.int64)).to(
        tl.pointer_type(tl.bfloat16)
    )

    row = qkv_ptr + t * row_stride + col0
    x = tl.load(row + d_ar).to(tl.float32)
    # RMSNorm：均值平方 + eps 后取倒数
    ms = tl.sum(x * x, 0) / D
    inv = 1.0 / tl.sqrt(ms + EPS)
    # bf16 权重提升到 fp32 再参与归一化
    w1 = tl.load(w_ptr + half_ar).to(tl.float32)
    w2 = tl.load(w_ptr + HALF + half_ar).to(tl.float32)
    x1 = tl.load(row + half_ar).to(tl.float32) * inv * w1
    x2 = tl.load(row + HALF + half_ar).to(tl.float32) * inv * w2
    # 中间结果先落回 bf16，与 eager 路径的 round 行为保持一致
    x1 = x1.to(tl.bfloat16).to(tl.float32)
    x2 = x2.to(tl.bfloat16).to(tl.float32)
    # neox 风格的 RoPE 旋转
    o1 = x1 * cos - x2 * sin
    o2 = x2 * cos + x1 * sin
    tl.store(row + half_ar, o1.to(tl.bfloat16))
    tl.store(row + HALF + half_ar, o2.to(tl.bfloat16))


def table_qk_norm_rope_(
    qkv: torch.Tensor,
    positions: torch.Tensor,
    q_weight: torch.Tensor,
    k_weight: torch.Tensor,
    cos_sin_cache: torch.Tensor,
    num_q_heads: int,
    num_k_heads: int,
    head_dim: int,
    eps: float,
) -> None:
    """原地对融合 QKV 张量执行 QK RMSNorm + 查表 neox RoPE。"""
    T = qkv.shape[0]
    if T == 0:
        return
    # grid 第一维是 token，第二维是 Q 头 + K 头
    grid = (T, num_q_heads + num_k_heads)
```

```
    _table_qk_norm_rope_kernel[grid](
        qkv,
        q_weight,
        k_weight,
        cos_sin_cache,
        positions,
        qkv.stride(0),
        num_q_heads * head_dim,
        NHQ=num_q_heads,
        D=head_dim,
        EPS=eps,
    )
```

python/sglang/srt/models/dspark.py

DSPARK 草稿模型的核心改造：新增 logits_mup_width_multiplier 适配 Inkling 的 muP 折叠 lm_head，并新增跨层 fused KV write bundle，把逐层 norm + rope + 写池合并为一次 GEMM + 一次 kernel launch。

```
# 构建跨层 fused KV 写入 bundle 的核心逻辑（dspark.py 新增）
# 这是典型的 fast path + 严格 eligibility 检查模式：
# 只有全部层满足条件时才启用融合路径，否则返回 None 走逐层回退。
def _build_fused_kv_write_bundle(self, pool):
    layers = list(self.layers)
    if not layers:
        return None
    # 池接口不满足时直接回退
    if not (hasattr(pool, "get_key_buffer") and hasattr(pool, "get_value_buffer")):
        return None
    attn0 = layers[0].self_attn
    head_dim = attn0.head_dim
    kv_size = attn0.kv_size
    rotary = attn0.rotary_emb
    # 只支持标准 RotaryEmbedding + neox 风格 + 全维度旋转（查表路径前提）
    if type(rotary).__name__ != "RotaryEmbedding":
        return None
    if not getattr(rotary, "is_neox_style", False):
        return None
    if getattr(rotary, "rotary_dim", None) != head_dim:
        return None
    eps = attn0.k_norm.variance_epsilon
    weights, knws, meta_rows = [], [], []
    for layer in layers:
        attn = layer.self_attn
        ok, _ = can_dflash_slice_qkv_weight(attn.qkv_proj)
        if not ok:
            return None
        if attn.qkv_proj.bias is not None:
            return None # fused kernel 不处理 bias
        if attn.attn.k_scale is not None or attn.attn.v_scale is not None:
```

```
            return None # fp8 等量化 scale 不支持
        if attn.head_dim != head_dim or attn.kv_size != kv_size:
            return None
        if attn.rotary_emb is not rotary and not torch.equal(
            attn.rotary_emb.cos_sin_cache, rotary.cos_sin_cache
        ):
            return None # 各层必须共享同一张 rotary 表
        if attn.k_norm.variance_epsilon != eps:
            return None
        k_buf = pool.get_key_buffer(attn.attn.layer_id)
        v_buf = pool.get_value_buffer(attn.attn.layer_id)
        nh = kv_size // head_dim
        for buf in (k_buf, v_buf):
            if buf.dtype != torch.bfloat16:
                return None
            if buf.shape[1:] != (nh, head_dim):
                return None
            # 要求 h-dim 连续布局，kernel 才能直接按 loc * stride 寻址
            if buf.stride(1) != head_dim or buf.stride(2) != 1:
                return None
        kv_slice = slice(attn.q_size, attn.q_size + 2 * attn.kv_size)
        w = attn.qkv_proj.weight[kv_slice]
        if w.dtype != torch.bfloat16:
            return None
        weights.append(w)
        knws.append(attn.k_norm.weight.data)
        # meta 行：k_buf / v_buf 指针 + 各自的行 stride，供 kernel 单 launch 写全部层
        meta_rows.append(
            [k_buf.data_ptr(), v_buf.data_ptr(), k_buf.stride(0), v_buf.stride(0)]
        )
    device = weights[0].device
    w_all = torch.cat(weights, dim=0).contiguous() # 一次 GEMM 出所有层 KV
    knw = torch.stack(knws).to(device)
    meta = torch.tensor(meta_rows, dtype=torch.int64, device=device)
    cos_sin = rotary.cos_sin_cache.to(device)
    return (w_all, meta, knw, cos_sin, eps, len(layers), kv_size, head_dim)
```

python/sglang/kernels/ops/speculative/dspark/fused_kv_write.py

新增 DSPARK fused KV 写入 kernel 模块：fused_kv_norm_rope_write 在一个 kernel 内完成逐层 K 的 RMSNorm、查表 neox RoPE 和 K / V 写池，且支持 in-kernel 的 commit_lens 截断（替代 host 端把尾部列 mask 成 -1）。

```
# DSPARK fused KV 写入入口（新增文件 fused_kv_write.py）
# 核心思路：把逐层的 K RMSNorm + neox RoPE + 写池合并进一个 kernel，
# commit_lens 由 kernel 内部处理，host 端不再把尾部列 mask 成 -1。
def fused_kv_norm_rope_write(
    kv: torch.Tensor,
    meta: torch.Tensor,
    k_norm_weights: torch.Tensor,
```

```
    cos_sin_cache: torch.Tensor,
    positions: torch.Tensor,
    locs: torch.Tensor,
    num_layers: int,
    kv_size: int,
    head_dim: int,
    eps: float,
    commit_lens: Optional[torch.Tensor] = None,
    locs_row_width: Optional[int] = None,
) -> None:
    """写入逐层 normed + roped K 与原始 V 行到 KV pool。

    行为约定：loc < 0 的行直接跳过；当给定 commit_lens 时，locs 是展平后的
    [bs, locs_row_width] verify 窗口，只有每行前 commit_lens[b] 列被写入。
    """
    T = kv.shape[0]
    if T == 0:
        return
    has_commit_lens = commit_lens is not None
    # 两个参数必须成对出现，避免调用方漏传导致静默错误
    if has_commit_lens != (locs_row_width is not None):
        raise ValueError(
            "commit_lens and locs_row_width must be passed together, got "
            f"commit_lens={'set' if has_commit_lens else None}, "
            f"locs_row_width={locs_row_width}."
        )
    if has_commit_lens:
        # 形状校验：commit_lens 数乘窗口宽度必须等于 locs 元素数
        if commit_lens.numel() * locs_row_width != locs.numel():
            raise ValueError(
                f"locs must be a flattened [{commit_lens.numel()}, "
                f"{locs_row_width}] window, got numel={locs.numel()}."
            )
        commit_lens_arg = commit_lens.contiguous()
    else:
        locs_row_width = 1
        commit_lens_arg = locs
    # grid：一维是所有待写 token，二维是层
    grid = (T, num_layers)
    _fused_kv_norm_rope_write_kernel[grid](
        kv,
        meta,
        k_norm_weights,
        cos_sin_cache,
        positions.to(torch.int64).contiguous(),
        locs.to(torch.int64).contiguous(),
        commit_lens_arg,
        locs_row_width,
        KV=kv_size,
```

```
        D=head_dim,
        NH=kv_size // head_dim,
        L=num_layers,
        EPS=eps,
        HAS_COMMIT_LENS=has_commit_lens,
    )
```

评论区精华

该 PR 全程由作者单人提交（8 个 commit 多为 'upd'），无 reviewer 评论。Issue 区仅有一条 gemini-code-assist[bot] 的每日配额提示和 ispobock 的 /rerun-failed-ci 指令，说明 CI 曾失败后重跑；最终 PR Test（Run #31350816508）与 PR Test Extra（Run #31350816469）均通过。技术讨论缺失，设计权衡只能从代码本身还原。

- 暂无高价值评论线程

风险与影响

- 风险：风险点集中在以下四处：

1. 数值一致性风险（dflash.py / dflash_utils.py）：table_qk_norm_rope_ 路径与旧 apply_qk_norm + rotary_emb 路径存在两套实现，注释声称共用同一张 cos/sin 表避免大位置角度漂移，但中间 bf16 round 行为是否与 eager 位精确一致没有测试兜底；_fused_correct_drafts_and_bonus_kernel 与 cumprod 路径也缺少等价性测试。
2. 回退路径依赖（dspark.py）：_build_fused_kv_write_bundle 对 bf16、无 bias、无 scale、连续 stride、共享 rotary 表等条件极其敏感，任一条件不满足会静默回退到逐层 set_kv_buffer；fused 与回退两条路径的输出一致性未被测试覆盖，未来若出现 fp8 或 bias 配置可能长期走回退而无人察觉。
3. 全局缓存抖动（mamba_state_scatter_triton.py）：_conv_multi_meta_cache 是模块级 dict，每次未命中都先 clear() 再写入，多池并发或 data_ptr 频繁变化时会反复重建 meta 表，带来性能抖动而非正确性问题。
4. 配置耦合（dspark_worker_v2.py）：logits_mup_width_multiplier 从 target hf config 读取并注入 draft model，若 checkpoint 训练语义与 config 不一致会静默产生 logits 尺度错误；该分支对非 muP 目标为 None，安全性依赖 config 正确性。 - 影响：影响范围与程度如下：
5. 模型支持：Inkling 系列模型首次获得 DSPARK 投机解码支持，mask token 使用 padded vocab 行（200064），muP 折叠 head 在 base logits 处完成尺度还原。
6. 性能路径：DFlash verify 的 correct/bonus 计算从逐 token torch 路径变为单 launch Triton；QK norm + RoPE 融合为原地 kernel；DSPARK KV 写入从逐层 2 次 GEMM 降为 1 次跨层 GEMM + 1 个 kernel；Mamba 多类型散射从多 launch 合并为单 launch——全部落在 decode 热路径上，对吞吐有直接正向影响。
7. 共享代码面：dflash_utils.py 与 mamba_state_scatter_triton.py 是 DFlash / Mamba 共享模块，改动通过门控与回退保护了旧路径，但任何回归都会波及 DFlash 与 Mamba 相关特性。

8. 团队协作：PR 无 review 讨论即合入，且提交信息全部为 'upd'，可追溯性较弱；后续 PR #33974 在统一内存池上继续演进 DSPARK，说明该功能线仍在快速迭代。 - 风险标记：核心路径变更，缺少测试覆盖，平台分支差异，全局缓存抖动

关联脉络

- PR #33974 [unified memory] Support DSPARK speculative decoding + fix two NaN root causes (page hand-out zeroing, CuTe int32 slot-stride wrap): 同一功能线在统一内存池上的延伸，共用 dspark_worker_v2、mamba_state_scatter_triton 等基础设施，并修复了本 PR 引入路径上的 NaN 根因。
- PR #34234 [Spec] Budget the DFLASH draft KV pool from its own attention geometry: 与本次同改 dflash_utils.py 及 DFLASH KV 材料化路径，修正 draft KV 池按层数估算的预算偏差。
- PR #34250 Update dspark draft path in Inkling small cookbook: 面向同一模型 + 算法组合（Inkling + DSPARK）的部署文档，说明本 PR 功能已进入正式使用。
- PR #33146 Support thinking budget for Inkling: 同属 Inkling 模型适配线，处理 sampling 侧 thinking budget 问题，与本 PR 的 muP 适配互为补充。