PR #35371 完整报告

sgl-project/sglang

[Spec] DFlash2: local convolution + candidate selector

合并时间：2026-08-19 08:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35371

执行摘要

- 一句话：DFlash2 新增局部卷积与候选选择器，draft 接受率吞吐大幅提升
- 推荐动作：值得精读。三个设计决策尤其值得学习：(1) 无损失 verify 的结构性保证——把 q 定义为 ' 实际抽取到的分布 '，贪心即点质量；(2) 一个 CUDA graph 同时服务贪心与采样，通过 host 侧 stage_sampling_params 静态刷新温度与 greedy_mask；(3) Triton walk 以单 program + 寄存器驻留解 slot 依赖，把成本与 batch 解耦。建议后续关注：为新 Triton kernel 与 flashinfer 降级路径补充更深的集成测试，并对大 top_k 的寄存器压力做 profiling。

功能与动机

在不增加额外 backbone pass 的前提下提升 DFlash draft 的提案质量与验证接受长度。PR body 明确说明：卷积使 'a proposal position can see the ones before it without another backbone pass'；selector 用 'score adjacent transitions edge(p→c) = ⟨A[p] ⊙ project(h), B[c]⟩ + unary[c]' 替代 'an independent argmax per slot'，并强调无损失性必须是结构性的——贪心行输出 point mass、采样行返回真实抽取的 q，'so the verify sees the proposal distribution it actually drew from'。给出的基准显示 DFlash2 在 GSM8K 上 acceptance 5.46、单并发吞吐 3.43x，全面优于 MTP 与 DSpark。

实现拆解

1. 配置契约扩展（python/sglang/srt/speculative/dflash_utils.py） DFlashDraftConfig 新增 conv_kernel_size、conv_group_size、selector_rank、selector_top_k、output_multiplier、final_logit_softcapping 六个字段，默认值 0 / 1.0 即未启用；parse_dflash_draft_config 强制 conv 两字段、selector 两字段成对出现，缺一即抛 ValueError；final_logit_softcapping 与 output_multiplier 一起定义 selector unary logit 的变换。另把原本定义在 worker 内的 _is_dense_head_weight 上移为 utils 级 is_dense_head_weight，供模型层与 worker 共用。
2. 模型层新增模块（python/sglang/srt/models/dflash.py） 新增 DFlashGroupedConv、_grouped_conv（@torch.compile(dynamic=True)）、CandidateSelector、DFlash2DraftModel 与 _radix_topk。DFlashDecoderLayer 可选择性挂接 attention_conv / mlp_conv，在 attention 与 MLP 的输入输出两侧用同一次 kernel_projection 包裹；DFlashDraftModel.set_block_size 允许运行时 Block Size 覆盖 checkpoint 内建值，卷积的块内位置索引依赖该值。_radix_topk 优先使用 flashinfer 确定

性 top_k，缺失时回退 torch.topk。

3. Worker 接线（python/sglang/srt/speculative/dflash_worker_v2.py） 新增 _SelectorDraftSampler 把 selector decode 折叠进 draft CUDA graph，一个 graph 同时服务贪心与 T>0 采样：stage_sampling_params 在 graph 重放前刷新温度与 greedy_mask，__call__ 内完成 lattice 构建、graph 内 philox uniform 抽取与 walk；_selector_lattice 把 [batch, slots, hidden] 展平喂给 2D top-k 再还原；_propose_selector_block 与 _commit_accept 处理块提交与 bonus token 落位。构建 selector 前用 is_dense_head_weight 拒绝量化 target lm_head 并退化为 eager。
4. Triton walk kernel（python/sglang/kernels/ops/speculative/dflash.py） 新增 _selector_walk_kernel：每个请求一个 program，slot 的 K 个分数驻留寄存器，slot 间依赖用程序内循环递进，避免每 slot 启动一个 kernel；贪心行以 one-hot 概率写 q，采样行以逆 CDF 抽取并写 softmax q，两种路径在同一 kernel 内合并。
5. 配套修复与测试 python/sglang/srt/model_executor/model_runner_components/spec_aux_hidden_state.py 抽出 _map_muse_target_layer_ids，把原先 " 无条件 +1" 改为仅当目标模型为 muse_glimmer 且 draft 架构为 DFlash2DraftModel / MuseGlimmerAssistantModel 时才偏移层 id；新增 test/registered/unit/spec/test_dflash_logits.py 覆盖 unary 变换、混合 batch 贪心确定性、量化 head 拒绝、非 2 的幂 block size 卷积四类场景，并新增 test_spec_aux_hidden_state.py 参数化测试层 id 映射，两组测试合计 4 + 5 = 9 项通过。

关键文件：

- python/sglang/srt/models/dflash.py（模块 草稿模型；类别 source；类型 core-logic；符号 _radix_topk, _grouped_conv, DFlashGroupedConv, DFlash2DraftModel）：核心模型实现：新增 DFlashGroupedConv、_grouped_conv、CandidateSelector、DFlash2DraftModel 与 _radix_topk，并把卷积包裹进每个 DecoderLayer 的 attention 与 MLP 两侧，是本 PR 的算法主体。
- python/sglang/srt/speculative/dflash_worker_v2.py（模块 投机解码；类别 source；类型 core-logic；符号 _SelectorDraftSampler, _selector_lattice, _propose_selector_block, _commit_accept）：Worker 侧集成：_SelectorDraftSampler 把 selector decode 折叠进 draft CUDA graph，统一贪心与 T>0 采样路径；新增 _propose_selector_block、_commit_accept，并对量化 target lm_head 做拒绝退化。
- test/registered/unit/spec/test_dflash_logits.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_dflash_unary_logit_transform, test_selector_greedy_row_walk_is_deterministic_in_a_mixed_batch, test_selector_rejects_a_quantized_target_lm_head, test_grouped_conv_supports_runtime_block_sizes）：新增测试覆盖 selector 端到端采样路径、混合贪心 / 采样 batch 的逐行确定性、量化 target lm_head 拒绝，以及非 2 的幂 block size 下卷积的数值正确性。
- python/sglang/srt/speculative/dflash_utils.py（模块 配置解析；类别 source；类型 configuration；符号 is_dense_head_weight, parse_dflash_draft_config, DFlashDraftConfig）：DFlashDraftConfig 扩展 6 个新字段并做成对校验，是整个特性开关的契约来源；is_dense_head_weight 上移为公共工具。

- python/sglang/srt/model_executor/model_runner_components/spec_aux_hidden_state.py（模块 隐状态映射；类别 source；类型 data-contract；符号 _map_muse_target_layer_ids）：修复 Muse 目标层 id 映射：+1 偏移只应作用于 muse_glimmer target 且 draft 输出层 id 的架构（含 DFlash2DraftModel），避免无条件 +1 引入错位。
- test/registered/unit/model_executor/model_runner_components/test_spec_aux_hidden_state.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_muse_target_layer_id_mapping）：参数化测试覆盖 muse_glimmer / qwen3 与三种 draft 架构的 5 种组合，锁定层 id 映射修复不回归。
- python/sglang/kernels/ops/speculative/dflash.py（模块 算子内核；类别 infra；类型 jit-kernel；符号 _selector_walk_kernel, selector_walk_triton）：新增 _selector_walk_kernel Triton kernel：每请求一个 program，slot 依赖用程序内循环解，贪心与采样共用 kernel，是 selector 性能的关键载体。

关键符号：DFlash2DraftModel.compute_candidates, DFlash2DraftModel._transform_unary_logits, CandidateSelector.build_lattice, CandidateSelector.sample_path, DFlashGroupedConv.prepare, DFlashGroupedConv.finish, _grouped_conv, _radix_topk, _SelectorDraftSampler.stage_sampling_params, _SelectorDraftSampler.call, _selector_lattice, _propose_selector_block, _commit_accept, selector_walk_triton, _selector_walk_kernel, is_dense_head_weight, _map_muse_target_layer_ids, set_block_size

关键源码片段

python/sglang/srt/models/dflash.py

核心模型实现：新增 DFlashGroupedConv、_grouped_conv、CandidateSelector、DFlash2DraftModel 与 _radix_topk，并把卷积包裹进每个 DecoderLayer 的 attention 与 MLP 两侧，是本 PR 的算法主体。

```
# DFlash2 的块内分组动态深度卷积：当前 token 的输出由自身及块内更早的 tap 个 token
# 共同决定，系数由输入侧一次投影得到，避免额外的 backbone pass；tap 跨块边界归零。
@torch.compile(dynamic=True, backend=get_compiler_backend(), disable=_is_npu)
def _grouped_conv(hidden_states, delta, base, block_size, num_groups, group_size, taps):
    # 把 hidden 最后一维切分为 (num_groups, group_size) 分组。
    blocks = hidden_states.unflatten(-1, (num_groups, group_size))
    # 系数形状 [1, taps, num_groups, group_size]：base 是 checkpoint 静态核，
    # delta 提供按 token 的组级动态偏置，即 out[i,c] = Σ_t (base + delta) · x[i−t,c]。
    coefficients = base.view(1, taps, num_groups, group_size) + delta.unsqueeze(-1)
    out = coefficients[:, 0] * blocks
    # 位置索引只在块内有效，块边界外的 tap 必须强制归零。
    position = torch.arange(hidden_states.shape[0], device=hidden_states.device)
    # 2 的幂 block size 用位与取模，避免整数除法；worker 解析的 block size 可能非 2 的幂。
    if block_size & (block_size - 1) == 0:
        position = position & (block_size - 1)
    else:
```

```
        position = position % block_size
    for tap in range(1, taps):
        # 前向移位实现因果卷积：位置 i 只能看到 i-tap 及更早。
        shifted = F.pad(blocks[:-tap], (0, 0, 0, 0, tap, 0))
        out = out + coefficients[:, tap] * shifted * (position >= tap).view(-1, 1, 1)
    return out.flatten(-2)


class DFlashGroupedConv(nn.Module):
    """跨一个 DFlash 块的分组动态深度卷积。

    每个子层被包裹两次：prepare 卷积输入并返回输出侧系数，finish 再用同一份投影
    系数卷积输出，输入与输出共用一个 kernel projection。
    """

    def __init__(self, hidden_size, block_size, taps, group_size):
        super().__init__()
        if hidden_size % group_size:
            raise ValueError(
                f'DFLASH conv_group_size={group_size} 必须整除 hidden_size={hidden_size}。'
            )
        hidden_size = int(hidden_size)
        self.block_size = int(block_size)
        self.taps = int(taps)
        self.group_size = int(group_size)
        self.num_groups = hidden_size // self.group_size
        # 布局 [input/output, tap, channel]，与训练导出的权重一致。
        base_kernel = torch.zeros(2, self.taps, hidden_size)
        base_kernel[:, 0] = 1.0
        self.base_kernel = nn.Parameter(base_kernel)
        # 从输入的单一投影同时产出输入侧与输出侧两组动态系数。
        self.kernel_projection = nn.Linear(
            hidden_size, 2 * self.taps * self.num_groups, bias=False
        )

    def _convolve(self, hidden_states, delta, side):
        # 在这里标记 static 而非在编译函数内部：编译期 trace 时维度已符号化，
        # 若在内部做 group 索引会额外花费整数除法与取模。
        torch._dynamo.mark_static(hidden_states, 1)
        torch._dynamo.mark_static(delta, 1)
        torch._dynamo.mark_static(delta, 2)
        return _grouped_conv(
            hidden_states, delta, self.base_kernel[side], self.block_size,
            self.num_groups, self.group_size, self.taps,
        )

    def prepare(self, hidden_states):
        # 一次投影拆出输入侧卷积结果与输出侧系数。
        coefficients = self.kernel_projection(hidden_states).reshape(
```

```
            *hidden_states.shape[:-1], 2, self.taps, self.num_groups
        )
        return (
            self._convolve(hidden_states, coefficients[..., 0, :, :], side=0),
            coefficients[..., 1, :, :],
        )

    def finish(self, hidden_states, coefficients):
        return self._convolve(hidden_states, coefficients, side=1)
```

python/sglang/srt/speculative/dflash_worker_v2.py

Worker 侧集成：_SelectorDraftSampler 把 selector decode 折叠进 draft CUDA graph，统一贪心与 T>0 采样路径；新增 _propose_selector_block、_commit_accept，并对量化 target lm_head 做拒绝退化。

```
def _selector_lattice(draft_model, pred_hidden, anchor_token_ids):
    # radix top-k kernel 是 2D 的，先展平 [batch, slots, hidden] 再按 K 复原。
    bs, num_pred = pred_hidden.shape[0], pred_hidden.shape[1]
    candidate_ids, unary_logits = draft_model.compute_candidates(
        pred_hidden.reshape(-1, pred_hidden.shape[-1])
    )
    candidate_ids = candidate_ids.view(bs, num_pred, -1)
    # build_lattice 用 unary logit、hidden 状态与已验证的 anchor token 构建
    # slot 间转移边，供后续 walk 使用。
    return candidate_ids, draft_model.candidate_selector.build_lattice(
        candidate_ids=candidate_ids,
        unary_logits=unary_logits.view(bs, num_pred, -1),
        hidden_states=pred_hidden,
        anchor_token_ids=anchor_token_ids,
    )


class _SelectorDraftSampler:
    """折叠进 draft CUDA graph 的 selector decode，贪心与 T>0 采样共用一条路径。

    一个捕获的 graph 服务两种模式：始终走采样路径，静态 greedy_mask 为每个请求行
    选出 argmax。静态 buffer 地址在捕获时被烘焙进 graph，重放前由宿主侧刷新参数。
    """

    def __init__(self, *, draft_model, block_size, max_bs, device):
        self.draft_model = draft_model
        self.selector = draft_model.candidate_selector
        self.block_size = int(block_size)
        max_bs, gamma, top_k = int(max_bs), self.block_size - 1, self.selector.top_k
        self.out = torch.empty((max_bs * gamma,), dtype=torch.int64, device=device)
        # 这些 buffer 的地址被捕获进 graph，重放前必须由 stage_sampling_params 刷新。
        self.temperatures = torch.ones((max_bs,), dtype=torch.float32, device=device)
        self.greedy_mask = torch.ones((max_bs,), dtype=torch.bool, device=device)
        self.uniforms = torch.empty((max_bs, gamma), dtype=torch.float32, device=device)
```

```
        self.candidate_out = torch.empty(
            (max_bs, gamma, top_k), dtype=torch.int64, device=device
        )
        self.q_out = torch.empty(
            (max_bs, gamma, top_k), dtype=torch.float32, device=device
        )

    def stage_sampling_params(self, *, bs, sampling_info):
        # 必须在 draft graph 重放前调用；贪心请求把温度置 1.0 并把
        # greedy_mask 置 True，使采样路径退化为确定性 argmax。
        if sampling_info is None:
            self.temperatures[:bs].fill_(1.0)
            self.greedy_mask[:bs].fill_(True)
            return
        torch.clamp(
            sampling_info.temperatures.view(-1)[:bs].to(torch.float32),
            min=1e-5,
            out=self.temperatures[:bs],
        )
        self.greedy_mask[:bs].copy_(
            resolve_greedy_mask(
                bs=bs, sampling_info=sampling_info, device=self.greedy_mask.device
            )
        )

    def __call__(self, hidden_states, input_ids):
        bs = hidden_states.shape[0] // self.block_size
        block_ids = input_ids.view(bs, self.block_size)
        # 块内位置 0 是已验证的 anchor token，proposal 从位置 1 开始。
        hs = hidden_states.view(bs, self.block_size, -1)[:, 1:, :]
        candidate_ids, scores = _selector_lattice(self.draft_model, hs, block_ids[:, 0])
        # graph 内 philox 采样：每次重放推进生成器并重新抽取 uniform。
        tokens, q_rows = self.selector.sample_path(
            candidate_ids=candidate_ids,
            scores=scores,
            uniforms=self.uniforms[:bs].uniform_(),
            temperatures=self.temperatures[:bs],
            greedy_mask=self.greedy_mask[:bs],
        )
        # 输出写给 worker 在重放后读取；候选与 q 一并落盘供 T>0 无损失 verify。
        self.out[: tokens.numel()].copy_(tokens.reshape(-1))
        self.candidate_out[:bs].copy_(candidate_ids)
        self.q_out[:bs].copy_(q_rows)
```

python/sglang/kernels/ops/speculative/dflash.py

新增 _selector_walk_kernel Triton kernel：每请求一个 program，slot 依赖用程序内循环解，贪心与采样共用 kernel，是 selector 性能的关键载体。

```
@triton.jit
```

```
def _selector_walk_kernel(
    scores_ptr, candidate_ptr, uniforms_ptr, temperatures_ptr,
    greedy_ptr, tokens_ptr, q_ptr,
    slots: tl.constexpr, top_k: tl.constexpr,
):
    # 每个 program 负责一个请求：一个 slot 的 K 个分数留在寄存器中，
    # slot 与 slot 的依赖是程序内循环，而不是每个 slot 启动一个 kernel。
    row = tl.program_id(0)
    offsets = tl.arange(0, top_k)
    temperature = tl.load(temperatures_ptr + row)
    greedy = tl.load(greedy_ptr + row) != 0
    previous = 0 # 从已验证的 anchor 出发，previous 记录上一 slot 选中的候选索引。
    for slot in range(slots):
        base = (row * slots + slot) * top_k
        # 取上一 slot 候选 c 对应的转移分数行：即 edge(p -> c) 的分数切片。
        scores = tl.load(scores_ptr + (base + previous) * top_k + offsets).to(tl.float32)
        if greedy:
            # 贪心行取 argmax，并以 one-hot 概率写出 q：verify 看到的就是
            # 确定性请求实际抽取的点质量分布。
            best = tl.max(scores, axis=0)
            index = tl.min(tl.where(scores == best, offsets, top_k), axis=0)
            probabilities = tl.where(offsets == index, 1.0, 0.0)
        else:
            # 采样行按逆 CDF 用 uniform 抽取候选，概率即 softmax 后的分布。
            scaled = scores / temperature
            exponentials = tl.exp(scaled - tl.max(scaled, axis=0))
            probabilities = exponentials / tl.sum(exponentials, axis=0)
            uniform = tl.load(uniforms_ptr + row * slots + slot)
            index = tl.sum(
                tl.where(uniform >= tl.cumsum(probabilities, axis=0), 1, 0), axis=0
            )
            index = tl.minimum(index, top_k - 1)
        # q 是 " 实际抽取到的分布 "：贪心为点质量、采样为软分布，
        # verify 据此做无损失拒绝采样。
        tl.store(q_ptr + base + offsets, probabilities)
        tl.store(tokens_ptr + row * slots + slot, tl.load(candidate_ptr + base + index))
        previous = index


def selector_walk_triton(*, candidate_ids, scores, uniforms, temperatures, greedy_mask):
    batch, slots, top_k = candidate_ids.shape
    tokens = torch.empty((batch, slots), dtype=torch.int64, device=scores.device)
    q_rows = torch.empty(
        (batch, slots, top_k), dtype=torch.float32, device=scores.device
    )
    _selector_walk_kernel[(batch,)]( # 一次启动 batch 个 program。
        scores.contiguous(), candidate_ids.contiguous(),
        uniforms.contiguous(), temperatures.contiguous(),
        greedy_mask.contiguous(), tokens, q_rows,
```

```
        slots=slots, top_k=top_k, num_warps=1,
    )
    return tokens, q_rows
```

评论区精华

本 PR 的 review 未产生实质性技术交锋：合并者 hnyls2002 仅给出 APPROVED，正文为 'LGTM'，随后以 3 个补充 commit 处理了 flashinfer fallback 告警、pytest main 入口与 TP size 引用。核心设计论证集中在 PR body 中：'Lossless decoding is the property that matters here and it is structural'——贪心行输出点质量、采样行返回真实抽取的 q，verify 看到的 proposal 分布与实际抽取一致，从结构上避免采样路径引入偏差；代价表显示 conv + selector 的额外开销为 1.5%-2.7%，且随 batch 增大而摊薄。这些属于设计说明而非 review 讨论，评审深度有限。

- 评审与合入过程 (other): 技术上未发生公开讨论；设计论证集中在 PR body 的成本表与无损失 verify 的结构性保证中。

风险与影响

- 风险：
  1. 对 target lm_head 的直接读取：DFlash2DraftModel.compute_candidates 把 target lm_head weight 当稠密矩阵做 matmul，量化 head（FP8/INT）会被当成激活值读取造成错误结果。新增 is_dense_head_weight 检查与 _eager 退化路径，并有单元测试兜底，但真实量化部署下的 e2e 行为仍需验证。
  2. 新 Triton kernel 的覆盖与算力：_selector_walk_kernel 中 top_k 与 slots 为 tl.constexpr，大 top_k 下寄存器占用线性增长；相关测试主要走 CandidateSelector.sample_path，对 selector_walk_triton 的间接覆盖有限。
  3. CUDA graph 内 RNG 重放：self.uniforms[:bs].uniform_() 在 captured graph 内执行，依赖 philox 状态在每次重放时推进；若捕获 / 重放顺序变化导致 uniform 序列复用，会破坏采样路径的随机性。
  4. flashinfer 可选依赖降级：_radix_topk 在缺少 flashinfer 时回退 torch.topk，两者对并列值的排序语义需保持一致才能保证采样结果 bit 级可复现，部署时需确认 flashinfer 版本。
  5. Muse 层 id 映射修复的影响面：_map_muse_target_layer_ids 把原先无条件 +1 改为按 target 与 draft 架构组合判断；DFlash2DraftModel 属新架构不受影响，但现有 Muse 部署若依赖旧行为则结果会变化，测试已覆盖 5 种组合。 - 影响：对用户：接入 DFlash2 checkpoint（如 z-lab/Qwen3.8-27B-DFlash2、meta-models/Muse-Glimmer-30B）后，同一部署无需修改任何服务参数即可获得更高 acceptance 与吞吐（GSM8K 单并发 3.43x vs DSpark 2.69x）。对系统：draft step 增加卷积与 selector 计算，额外开销 1.5%-2.7% 并随 batch 摊薄；selector decode 折叠进 CUDA graph 后不增加额外 kernel 启动。对团队：speculative 路径新增一条独立分支（selector 路径与原 argmax 路径并存），后续维护与回归成本上升；新测试挂在 CPU CI suite（base-a-test-cpu），由 run-ci-extra 覆盖。 - 风险标记：投机解码核心路径变更，新 Triton kernel 深度覆盖不足，CUDA graph 内 RNG 重放敏感，

flashinfer 可选依赖降级

关联脉络

- PR #35382 [Refactor] Share the page-aligned decode alloc lens between EAGLE and DFLASH: 同属 DFLASH spec 路径的配套重构，直接涉及 dflash_worker_v2.py 与 dflash_utils.py 的公共逻辑抽取，与本 PR 的 worker 改动同文件演进。
- PR #35265 [Spec] Page-align the DFLASH decode KV reservation: 修复 dflash_info_v2.py / dflash_worker_v2.py 的解码 KV 预留记账，与本 PR 在同一 DFLASH v2 解码路径上叠加，构成 DFLASH v2 的连续演进线。
- PR #35375 [Memory] Borrow CUDA graph pool storage for EAGLE sampling: 与本 PR 的 _SelectorDraftSampler 设计同源：都在 CUDA graph 捕获 / 重放边界上管理静态 buffer 与 RNG 状态，是投机解码 sampling 侧的共同优化方向。
- PR #35339 [diffusion] Per-request lossy accelerations: Cache-DiT, CFG gating, attention backend override: 同为 checkpoint/ 配置驱动的可选加速开关且默认保持旧路径，设计哲学一致，体现该阶段 SGLang 在加速能力上的可插拔式演进模式。