

PR #55737 完整报告

vllm-project/vllm

[Perf][GLM-5.3-Flash] Use FlashKDA for KDA chunked prefill (1.7-3.8x faster than the Triton chunk path)

合并时间: 2026-09-14 16:36

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55737>

执行摘要

- 一句话: KDA 预填切换 FlashKDA, 单层提速 1.7-3.8 倍
- 推荐动作: 值得精读。三个可复用的设计点: 一是「能力门控 + 逃生舱 + 显式指定时 fail-fast」的后端自动选择模板 (`_resolve_kda_prefill_backend`), 比直接改默认分支更安全; 二是 spec / 非 spec 混合步用 workspace 缓冲 + `non_spec_token_indx` 散射的方案, 是在不牺牲投机解码语义 (`num_accepted_tokens` 回滚仍走 recurrent kernel) 前提下复用 chunk kernel 的通用手法; 三是 `ns_out` 直写层输出与合并路径的耦合, 读代码时要注意「合并逻辑被简化」的前提条件。若要移植到其他模型, 重点关注 workspace 静态上界是否被断言保护、以及是否有对应的自动化测试; 对运维同学, 如需复现旧数值可用 `additional_config.kda_prefill_backend=triton` 一键回退后做 A/B。

功能与动机

PR body 开篇说明这是「One of three independent GLM-5.3-Flash perf PRs」: GLM-5.3-Flash 的分块预填此前走 `chunk_kda_with_fused_gate`, 单层要启动约 15 个 Triton kernel, 是 prefill 延迟的主要来源。FlashKDA 已经作为 `vllm._flashkda_C` 为 Kimi-K3 构建并投入使用, 且实现的是同一套有界门 KDA 递推 (`lower_bound * sigmoid(exp(A_log) * (g + dt_bias))`)、kernel 内 q/k l2norm、raw beta logits), 因此对 GLM-5.3-Flash 是 drop-in 替换。PR 给出的动机量化目标是层内 183us→49us (8x2048)、1371us→787us (1x16384), 并强调「The output is written straight into the layer buffer (no merge copy) and the pre-sigmoid beta cast is not needed on this path」。

实现拆解

变更入口是唯一的源文件 `vllm/models/glm5next/nvidia/kda.py` (+144/-28), 核心是把 KDA 分块预填的执行路径从 Triton 切到 FlashKDA, 同时保留 Triton 作为回退。

1. 后端解析与能力门控 (新增 `_resolve_kda_prefill_backend`): 从 `additional_config.kda_prefill_backend` 读取 `auto` / `triton` / `flashkda`, 取值非法抛 `ValueError`; 能力判断为 `current_platform.is_cuda()`、`get_device_capability().major in (9, 10, 12)`、`head_dim == 128`、`dtype == torch.bfloat16`、`lower_bound is not None` (有界门)。显式指定 `flashkda` 但条件不满足时抛 `RuntimeError` 拒绝启动, 而不是静默退回慢路径; `auto` 满足即用 FlashKDA, `triton` 永远保留旧路径作为逃生舱。这一步直接改变了默认行为: 不改配置的 GLM-5.3-Flash 部署会换内核。

2. 构造期缓冲区规划 (`Glm5NextLinearAttention.__init__`) : 解析结果存为 `self.kda_prefill_backend`; 当选到 `flashkda` 时提前 `import vllm._flashkda_C` 以快速失败, 并用 `torch.ops._flashkda_C.get_workspace_size(max_num_batched_tokens, local_num_heads, max_num_seqs)` 生成 `_flashkda_buffer_specs` 三段规格——`(max_seqs, local_num_heads, head_dim, head_dim)` 的 `final_state` (`dtype` 取 `get_state_dtype()[1]`)、`(workspace_size,)` 的 `uint8` 工作区、以及 `(1, max_tokens, local_num_heads, head_dim)` 的输出缓冲 (专门服务带投机 token 的混合步)。把规格挂在实例上、交给 `current_workspace_manager().get_simultaneous` 复用, 避免每步重新分配显存。
3. FlashKDA 调用封装 (新增 `_flashkda_prefill`) : 从 `workspace` 取三段 `buffer`, `final_state` 按 `initial_state.shape[0]` 裁到本次实际序列数, `out is None` 时回落到 `workspace_out[:, :q.shape[1]]`; 随后调用 `torch.ops._flashkda_C.fwd(...)`, 传入 `q/k/v/g` 的 `.contiguous()` 副本 (FlashKDA 硬编码 dense stride)、`self.head_dim**-0.5` 缩放、`A_log.view(-1)`、`dt_bias.view(-1, head_dim)`、`self.kda_lower_bound`、`initial_state` 与 `cu_seqlens` 的连续副本。该 kernel 直接消费原始 gate logits `g` 与原始 beta logits, 在 kernel 内完成 l2norm 与有界门计算, 语义对齐 `chunk_kda_with_fused_gate(..., safe_gate=True)`。
4. forward 双路分派与合并路径精简: `attn_metadata.narrowed.num_prefills > 0` 时按 `self.kda_prefill_backend` 分派——`flashkda` 分支设 `ns_out = None if use_spec else core_attn_out[:, :num_actual_tokens]`, 纯预填步直接写层输出缓冲, 混合步写 `workspace`; `triton` 分支保留 `beta=_cast_sigmoid(beta_ns.squeeze(0)).unsqueeze(0)` 与 `chunk_kda_with_fused_gate`。因为混合步不再把结果写进 `core_attn_out`, 合并逻辑从「临时 merged 张量 + 两次 `index_copy_` + 整体赋值」简化为直接对 `core_attn_out` 做两次 `index_copy_`, 省掉一次 `(1, num_actual_tokens, H, D)` 的分配与拷贝。投机 token 本身仍走 `fused_recurrent_kda` (`num_accepted_tokens` 回滚语义), 不受影响。
5. 测试与配置配套: 仓库内没有新增或修改任何测试文件; 验证依赖 `bench/kda_numerics.py` (对 naive 参考实现 `mean|Δ| 2e-5`、`max 4.8e-4`)、`bench/flashkda_vs_chunk.py` 微基准、`pytest tests/models/kimi_k3/test_kda.py -k flashkda` (扩展 sanity), 以及 4x GB300 TP4 上的 serving 基准与 `gsm8k / prompt-logprob / 36-needle` 精度对比。外部配置入口只有 `additional_config.kda_prefill_backend`; ROCm 分支 (`vllm/models/glm5next/amd/...`) 完全未改动。

关键文件:

- `vllm/models/glm5next/nvidia/kda.py` (模块 线性注意力; 类别 source; 类型 core-logic; 符号 `_resolve_kda_prefill_backend`, `_flashkda_prefill`, `Glm5NextLinearAttention.init`, `Glm5NextLinearAttention.forward`) : 唯一改动文件, 承载全部逻辑: 新增 KDA 预填后端解析与 FlashKDA 调用封装, 改写 `prefill` 分支分派、`workspace` 规划与 `spec/ 非 spec` 输出合并路径。

关键符号: `_resolve_kda_prefill_backend`, `_flashkda_prefill`,
`Glm5NextLinearAttention.init`, `Glm5NextLinearAttention.forward`

关键源码片段

vllm/models/glm5next/nvidia/kda.py

唯一改动文件，承载全部逻辑：新增 KDA 预填后端解析与 FlashKDA 调用封装，改写 prefill 分支分派、workspace 规划与 spec/ 非 spec 输出合并路径。

后端选择与能力门控

```
def _resolve_kda_prefill_backend(
    backend: str, head_dim: int, dtype: torch.dtype, lower_bound: float | None
) -> str:
    """Pick the chunked-prefill kernel: FlashKDA (fused CUDA, ~2-4x faster on
    SM90/SM10x/SM12x for bf16, head_dim 128 and a bounded gate) or the Triton
    ``chunk_kda_with_fused_gate`` path. ``backend`` comes from
    ``additional_config.kda_prefill_backend`` (auto / triton / flashkda)."""
    # 取值拼写错误直接失败，避免配置写错后静默回落到 Triton 而无人察觉
    if backend not in ("auto", "triton", "flashkda"):
        raise ValueError(f"Unsupported KDA prefill backend: {backend}")
    capability = current_platform.get_device_capability()
    # FlashKDA 目前是 CUDA 专属：SM90 / SM10x / SM12x、bf16、head_dim == 128,
    # 且模型必须使用有界门（lower_bound 非 None，GLM-5.3-Flash 满足）
    supported = (
        current_platform.is_cuda()
        and capability is not None
        and capability.major in (9, 10, 12)
        and head_dim == 128
        and dtype == torch.bfloat16
        and lower_bound is not None
    )
    # 显式指定 flashkda 但硬件 / 精度不满足时拒绝启动，而不是悄悄退回慢路径
    if backend == "flashkda" and not supported:
        raise RuntimeError(
            "FlashKDA requires CUDA SM90/SM10x/SM12x, bfloat16, head_dim=128 "
            "and a bounded KDA gate."
        )
    # auto: 满足条件就用 FlashKDA; triton 永远保留旧路径作为逃生舱
    return "flashkda" if supported and backend != "triton" else "triton"
```

FlashKDA 预填调用封装

```
def _flashkda_prefill(
    self,
    q: torch.Tensor,
    k: torch.Tensor,
    v: torch.Tensor,
    g: torch.Tensor,
    beta: torch.Tensor,
    initial_state: torch.Tensor,
    cu_seqlens: torch.Tensor,
    out: torch.Tensor | None,
) -> tuple[torch.Tensor, torch.Tensor]:
    """Fused KDA chunked prefill (FlashKDA): 直接吃原始 gate logits ``g`` 和
```

```

原始 ``beta`` logits, 在 kernel 内做 q/k l2norm 并施加有界门
``lower_bound * sigmoid(exp(A_log) * (g + dt_bias))``, 与
``chunk_kda_with_fused_gate(..., safe_gate=True)`` 语义一致。"""
assert self._flashkda_buffer_specs is not None
# 三块 buffer 由构造期规划、workspace manager 统一复用:
# final_state (KDA 状态) / uint8 工作区 / 混合步输出缓冲
final_state, workspace, workspace_out = (
    current_workspace_manager().get_simultaneous(*self._flashkda_buffer_specs)
)
# 本次只有 num_prefill_seqs 条序列需要写回状态, 按实际长度裁剪
final_state = final_state[: initial_state.shape[0]]
if out is None:
    # 混合步 (同时有投机 token) 走这里: 输出落 workspace 缓冲,
    # 之后由 non_spec_token_indx 散射回 core_attn_out
    out = workspace_out[:, : q.shape[1]]
# FlashKDA 硬编码 dense q/k/v/g stride; beta 允许 row-strided 故不拷贝
torch.ops._flashkda_C.fwd(
    q.contiguous(),
    k.contiguous(),
    v.contiguous(),
    g.contiguous(),
    beta,
    self.head_dim**-0.5, # softmax 缩放, 在 kernel 内作用于 q
    out,
    workspace,
    self.A_log.view(-1),
    self.dt_bias.view(-1, self.head_dim),
    self.kda_lower_bound, # 有界门的上 / 下界, 由模型 config 提供
    initial_state.contiguous(),
    final_state,
    cu_seqlens.contiguous(),
    None,
    None,
)
return out, final_state

```

评论区精华

讨论的核心是一处被 review 逼出来的实现补全。ZJY0516 在 prefill 分支的 `if self.kda_prefill_backend == "flashkda" and not use_spec`: 上追问「I remember flashkda also supports spec decoding?」, 作者最初回答「I remember not, need more profiling and optimization on SpecDecode path tho..」; ZJY0516 随后直接给出同仓库参照「you can check this: [vllm/models/kimi_k3/nvidia/kda.py](#), no such guard」, 作者回复「thanks, will work on it」, 并在 commit [674dd959](#) 中移除了守卫: 混合步改为 FlashKDA 写 workspace 缓冲、再由 `non_spec_token_indx` 散射回 `core_attn_out`, 与 Kimi-K3 KDA 层方案对齐。另一条线索来自 gaby 对评测配置的疑问「你用了 `--attention-backend FLASHINFER_MLA_SPARSE` 但官方 recipe 都没用, 这样性能更好吗?」, 作者澄清该模型在 vLLM 只有一个注意力后端, 加 flag 只为清晰, 说明基准没有靠非默认后端取巧。此外

NNNtrance 给出了跨平台现场复现：三台 DGX Spark GB10 (sm_121、arm64、CUDA 13.0) TP3+EP 跑 EXL3 4.05bpw, kernel 2.47ms→0.97ms (2.55x, 含三次 contiguous 拷贝), 数值 mean|Δ| 1.2e-5、max 9.8e-4, 端到端 prefill +6.5%、TTFT -5.1%、单流 decode 不变、质量门禁全过。Copilot 则提示这是性能关键路径变更, 需要人工复核执行模式兼容性, 最终 ZJY0516 APPROVED。

- FlashKDA 是否应覆盖投机解码 (spec-decode) 步 (design): 移除 not use_spec 守卫: 混合步改为 FlashKDA 写 workspace 输出缓冲、再由 non_spec_token_indx 散射回 core_attn_out, 投机 token 本身仍保留带 num_accepted_tokens 回滚语义的 fused_recurrent_kda。这一改动带来 MTP k=1 场景 TTFT -5.5~-8.8%、decode +6.3~+11.3%。
- auto 默认启用 FlashKDA 的门槛是否足够保守 (design): 采用「auto 自动、triton 可回退、flashkda 强制即校验」三段式策略, 默认变更但保留显式逃生舱; ROCm 分支不受影响。
- 基准中 --attention-backend FLASHINFER_MLA_SPARSE 是否属于非默认配置 (question): 作者澄清 GLM-5.3-Flash 在 vLLM 目前只有一个注意力后端即 FLASHINFER_MLA_SPARSE, 显式加 flag 只为清晰, 基准没有依赖特殊后端设置。
- 第三方跨平台现场复现 (DGX Spark GB10) (testing): 独立环境复现验证了收益与数值一致性, 也侧面确认 major in (9, 10, 12) 的 SM121 覆盖是有效的。
- 缺少仓库内自动化回归测试 (testing): 无 glm5next 专属测试落地, _resolve_kda_prefill_backend 的分支矩阵与 FlashKDA 预填路径的回归只能靠人工基准; 审核以 ZJY0516 的 APPROVED 收尾, 测试补齐仍是开放项。

风险与影响

- 风险: 1) 默认行为变更: auto 在 SM90/10/12 + bf16 + head_dim 128 上默认选中 FlashKDA, 存量 GLM-5.3-Flash 部署不改配置即换 kernel; 虽然 gsm8k 93.33→93.25、prompt-logprob A/B 在噪声内、社区报告 mean|Δ| 1e-5 量级, 但 bf16 ULP 级差异对严格 logprob 比对与回归基线仍可能造成抖动。2) workspace 静态上界契约: 缓冲按构造期的 max_num_batched_tokens / max_num_seqs 规划, _flashkda_prefill 里 workspace_out[:, :q.shape[1]] 与 final_state[:, initial_state.shape[0]] 都是窄化切片而非校验, 一旦某步 token 数或序列数超过规划值, 可能得到形状不足的输出缓冲而非清晰报错, 建议补断言。3) 硬编码 stride 的拷贝开销: FlashKDA 要求 q/k/v/g dense, 导致四个 contiguous() 在 16k chunk 下约 115us/ 层 (作者自述的后续优化点), 长 prefill 场景这部分成本会稀释收益。4) 外部扩展耦合: 依赖 vllm._flashkda_C 的 get_workspace_size / fwd 私有算子签名, 扩展 ABI 漂移会在构造期 import 或运行期失败 (构造期提前 import 已做 fail-fast)。5) 测试覆盖缺口: 无 glm5next 专属单测 /CI, _resolve_kda_prefill_backend 的分支矩阵 (auto/triton/flashkda × 支持 / 不支持) 没有自动化验证, 回归只能靠人工基准。6) 合并路径的隐式契约: core_attn_out.index_copy_ 原地写成立的前提是混合步输出落在 workspace (ns_out is None), 未来若新增后端把结果写回 core_attn_out 会相互覆盖。
- 影响: 用户侧: GLM-5.3-Flash 在 H100/GB200/GB300 等受支持 GPU 上 prefill TTFT 下降约 8-13% (8x2048 -12.2%、2x8192 -13.2%、8x32768 -8.1%、2x65536 -7.9%) , MTP k=1 下 TTFT -5.5~-8.8%、decode 吞吐 +6.3~+11.3%, 单流 decode 基本不变

(+0.7%)；混合步不再整段退回 Triton 是并发场景收益的主要来源。系统侧：prefill 计算量显著下降意味着同卡可承载更多并发 prefill，但每层新增常驻三段 workspace（final_state、uint8 工作区、输出缓冲），其显存占用需确认已计入 gpu_memory_utilization 的 profiling 预算。其他模型与 ROCm 路径零影响（只改 vllm/models/glm5next/nvidia/kda.py，AMD 分支未动）。团队侧：GLM-5.3-Flash KDA 层与 Kimi-K3 KDA 层在实现上进一步对齐（同一扩展、同一 spec 混合步散射方案），但也引入了 additional_config.kda_prefill_backend 这个临时配置面，等 #55364 的引擎参数落地后需要迁移去重。

- 风险标记：核心预填路径切换，默认行为变更（auto 启用），缺少仓库内回归测试，workspace 缓冲静态上界未断言，硬编码 contiguous 拷贝开销，依赖外部 CUDA 扩展 ABI

关联脉络

- PR #55364 [Perf] Integrate FlashInfer KDA kernels: 同为 KDA 预填 / 解码后端接入工作，且为 Kimi-K3 引入了引擎级 --kda-prefill-backend 参数；本 PR body 明确说明「Once #55364 lands, the additional_config.kda_prefill_backend knob here should move to that arg」，两者在配置入口上需要收敛。
- PR #55736 GLM-5.3-Flash decode 性能 PR（本系列三篇之一）：本 PR 的全部性能对照都以 main+#55736 为基线（#55736 只动 decode），两者构成 GLM-5.3-Flash prefill/decode 的成对优化，评测结论需要合并阅读。
- PR #55224 KDA conv 权重 refit 修复：PR 的 duplicate-work check 中主动核对过该 PR（「#55224 is the KDA conv weight refit fix」），确认不存在重复实现，属于同一 KDA 模块的邻近修复。
- PR #55235 [ROCm][Perf] Tune MiniMax-M3 decode top-k for short contexts: 同类范式：单模型专属 kernel 性能调优 + 微基准给出倍数收益 + 小范围测试补充，可作为对比理解 vLLM 近期「按模型选最优 kernel 后端」的演进节奏。