PR #50089 完整报告

vllm-project/vllm

[Model] Add Kimi K3 support: model files and kernels [1/N]

合并时间：2026-07-29 14:10

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50089

执行摘要

- 一句话：新增 Kimi K3 多模态模型推理支持（第一阶段）
- 推荐动作：建议核心架构师与模型组重点审阅：（1）vllm/models/kimi_k3/ 的模块划分是否合适；（2）CuTe DSL 内核与现有 custom_ops 的边界；（3）Latent MoE 融合路径的正确性。对于一般开发者，该 PR 是学习 vLLM 模型扩展与高性能内核的绝佳案例。

功能与动机

Kimi K3 是 Kimi 推出的新一代多模态大模型，vLLM 社区需要集成该模型以扩展支持的模型生态。PR#50089 是原 PR#50000 的第一部分拆分，用于提前合入模型定义与内核代码，后续将补充前端集成与测试使其可运行（参见 PR body 说明）。

实现拆解

1. 模型架构：在 vllm/models/kimi_k3/ 下按平台组织，amd/linear.py 定义 KimiDecoderLayer、KimiMoE、KimiMLAAttention 等通用模块；nvidia/model.py 提供 NVIDIA 专用逻辑。
2. 注意力机制：NVIDIA 端引入 KDA（Gated Delta Net Attention），通过 nvidia/kda.py 实现线性注意力核算子，nvidia/kda_metadata.py 构建运行时元数据，支持 PDL 提前启动。
3. MoE 与激活：新增 SituAndMul 激活函数（vllm/model_executor/layers/activation.py），LatentMoERunner 将 latent MoE 的 allreduce 与输出变换融合，减少通信开销；配套 CuTe DSL fused 内核 KimiK3LatentMoETailOp。
4. 低延迟 GEMM：cute_dsl/_skinny_gemm.py 编写 CuTe 模板内核，skinny_gemm.py 封装 ShapeDynamicSkinnyGemm，根据 (M, N, K) 动态选择最优配置。
5. 推测解码：nvidia/dspark_mla.py 实现 K3DSparkModel，配合复制版 Markov Head 完成 DSpark 草稿模型；amd/mtp.py 实现多 token 预测（MTP）。
6. 多模态预处理：common/mm_preprocess.py 提供图像缩放、Processor 初始化等，注册到模型配置。
7. 配套变更：更新模型注册表（registry.py）、Kimi Linear 配置解析、新增 16+ 个测试文件、Buildkite CI 入口。

关键文件：

- vllm/models/kimi_k3/nvidia/kda.py（模块 KDA 注意力；类别 source；类型 data-contract；符号 a_log_weight_loader, _KimiGDNMergedColumnParallelLinear, is_fused_kda_decode_supported, is_flashkda_supported）：K3 核心注意力机制 KDA 的实现，包含自定义 a_log_weight_loader、_KimiGDNMergedColumnParallelLinear、is_fused_kda_decode_supported 等关键逻辑。
- vllm/models/kimi_k3/nvidia/low_latency_gemm.py（模块 低延迟 GEMM；类别 source；类型 data-contract；符号 ProjectionSpec, cute_config, _cute, _backend_for）：K3 模型解码阶段 GEMM 后端的动态选择机制，根据 (M, N, K) 决定使用 CuTe 或 dsv3_fused_a 后端，包含精细的 ProjectionSpec 配置表。
- vllm/model_executor/layers/fused_moe/runner/latent_moe_runner.py（模块 MoE 运行器；类别 source；类型 data-contract；符号 LatentMoERunner, _get_zero_residual, _use_fused_path, forward）：Latent MoE 融合运行器，将 routed 与 shared expert 的 allreduce 合并为一次，并支持 CuTe DSL 融合尾操作，是 K3 MoE 性能关键。
- vllm/model_executor/kernels/linear/cute_dsl/skinny_gemm.py（模块 CuTe 封装；类别 source；类型 data-contract；符号 SkinnyGemmConfig, ShapeDynamicSkinnyGemm, is_available, _config）：ShapeDynamicSkinnyGemm 封装，管理 CuTe 内核的编译缓存与动态配置。
- vllm/model_executor/kernels/linear/cute_dsl/_skinny_gemm.py（模块 CuTe 内核；类别 source；类型 data-contract；符号 CuteSkinnyGemm, init, call, kernel）：CuTe DSL 内核主体，使用 cute.jit 和 cute.kernel 装饰器实现形状动态的低延迟 GEMM。
- vllm/models/kimi_k3/amd/linear.py（模块 AMD 模型层；类别 source；类型 data-contract；符号 KimiMLP, KimiRoutedOutputTransform, _apply_attn_res, KimiMoE）：AMD 后端 K3 模型基础模块，定义 KimiMLP、KimiMoE、KimiDecoderLayer 等，复用现有 MoE/MLA 框架。
- vllm/models/kimi_k3/common/mm_preprocess.py（模块 多模态预处理；类别 source；类型 data-contract；符号 navit_resize_image, KimiK3ProcessingInfo, get_supported_mm_limits, get_max_image_size）：多模态预处理统一入口，负责图像缩放、Processor 初始化、图像尺寸校验等。

关键符号：KimiMLP, KimiRoutedOutputTransform, KimiMoE, KimiMLAAttention, KimiDecoderLayer, CuteSkinnyGemm, ShapeDynamicSkinnyGemm, LatentMoERunner, KimiK3KDAMetadata, KimiK3ForConditionalGeneration, ReplicatedDSparkMarkovHead, K3DSparkModel, SharedHead, KimiK3MultiTokenPredictor, select_kimi_k3_backend

关键源码片段

vllm/model_executor/layers/fused_moe/runner/latent_moe_runner.py

Latent MoE 融合运行器，将 routed 与 shared expert 的 allreduce 合并为一次，并支持 CuTe DSL 融合尾操作，是 K3 MoE 性能关键。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
import torch
```

```
from vllm.config import get_current_vllm_config
from vllm.distributed import tensor_model_parallel_all_reduce
from vllm.model_executor.layers.fused_moe.runner.moe_runner import MoERunner


class LatentMoERunner(MoERunner):
    """MoE runner for latent MoE with a replicated routed up-projection.

    Fused path (tp>1, un-reduced combine output, shared expert, no SP):
    concatenates the un-reduced latent partial (dim d) and the un-reduced
    shared partial (dim D) into one contiguous buffer, all-reduces once, then
    splits. The latent half is normed and up-projected locally (replicated
    up-proj -> full hidden), and the shared add folds into the GEMM epilogue
    (``torch.addmm``). One collective total, no post-reduction communication.
    """

    def __init__(self, *args, enable_k3_latent_moe_tail_fusion: bool = False, **kwargs):
        super().__init__(*args, **kwargs)
        self.enable_k3_latent_moe_tail_fusion = enable_k3_latent_moe_tail_fusion
        # 检查是否满足 K3 尾融合的条件：TP=8 或 16 且不使用 ubatching/sleep 模式
        use_fused_path = self._use_fused_path()
        if (self.enable_k3_latent_moe_tail_fusion and use_fused_path
                and self.moe_config.tp_size not in (8, 16)):
            # TP 不匹配时回退默认路径
            self.enable_k3_latent_moe_tail_fusion = False

        if self.enable_k3_latent_moe_tail_fusion and use_fused_path:
            vllm_config = get_current_vllm_config()
            if vllm_config.parallel_config.use_ubatching:
                raise ValueError("K3 latent-MoE tail fusion does not support DBO or ubatching.")
            # 初始化 CuTe DSL 融合算子
            from vllm.models.kimi_k3.nvidia.ops.latent_moe_tail import KimiK3LatentMoETailOp
            self._k3_latent_moe_tail_op = KimiK3LatentMoETailOp.initialize(
                hidden_size=..., latent_size=..., dtype=..., device=..., rms_eps=...
            )

    def _use_fused_path(self):
        # 融合路径要求：TP>1、有 shared expert、未 reduce 输出且不是 sequence parallel
        return (self.moe_config.tp_size > 1
                and self._shared_experts is not None
                and not self._fused_output_is_reduced
                and not self.moe_config.is_sequence_parallel)

    def forward(self, hidden_states, router_logits, input_ids=None, shared_experts_input=None):
        if self._use_fused_path() and self.enable_k3_latent_moe_tail_fusion:
            return self._fused_forward(...) # 单次 allreduce + 融合 epilogue
        return super().forward(...) # 回退基类两条 collectives 路径
```

vllm/model_executor/kernels/linear/cute_dsl/skinny_gemm.py

ShapeDynamicSkinnyGemm 封装，管理 CuTe 内核的编译缓存与动态配置。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
from __future__ import annotations
from dataclasses import dataclass
from functools import partial
from typing import Any
import torch


@dataclass(frozen=True, slots=True)
class SkinnyGemmConfig:
    num_rows: int # M 维（token 数）
    block_size: int # 线程块大小
    outputs_per_block: int # 每个线程块处理的输出列数
    k_unroll: int = 1 # K 循环展开因子
    vector_width: int = 8 # 向量加载宽度


class ShapeDynamicSkinnyGemm:
    """形状动态的低延迟 GEMM，为小 token 数量优化。"""

    def __init__(self):
        self._compiled: dict[tuple[torch.dtype, SkinnyGemmConfig, bool], Any] = {}

    @staticmethod
    def is_available() -> bool:
        """检查 CuTe DSL 是否可用。"""
        try:
            import cutlass
            import cutlass.cute
            return True
        except ImportError:
            return False

    @staticmethod
    def _config(m: int, n: int, k: int) -> SkinnyGemmConfig:
        """根据形状启发式选择最佳配置。"""
        num_rows = m
        wide_block = 224
        if m == 1 and k >= 7168 and k % (wide_block * 8) == 0:
            # 宽 K 时使用大 block 以利用高并行
            if n % 3 == 0:
                k_unroll = 2 if n <= 2304 else 4
                return SkinnyGemmConfig(num_rows, wide_block, 3, k_unroll)
        # 默认：根据 K 大小选择 block_size 64 或 128
        block_size = 64 if 4096 <= n < 8192 else 128
        outputs_per_block = 1 if m == 1 and n <= 2304 else 2
```

```
        return SkinnyGemmConfig(num_rows, block_size, outputs_per_block, k_unroll=2)

    def _compile(self, dtype, config, has_residual):
        """编译 CuTe 内核并缓存。"""
        from ._skinny_gemm import CuteSkinnyGemm
        import cutlass.cute as cute
        # 创建 fake tensor 用于编译时形状推断
        k = cute.sym_int(divisibility=config.block_size * config.vector_width)
        a = ... ; b = ... ; c = ... ; residual = ...
        kernel = CuteSkinnyGemm(element_type=..., num_rows=config.num_rows, ...)
        self._compiled[(dtype, config, has_residual)] = cute.compile(kernel, a, b, residual, c, ...)
```

vllm/model_executor/kernels/linear/cute_dsl/_skinny_gemm.py

CuTe DSL 内核主体，使用 cute.jit 和 cute.kernel 装饰器实现形状动态的低延迟 GEMM。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
from __future__ import annotations
import cutlass
import cutlass.cute as cute
from cuda.bindings.driver import CUstream
from cutlass import const_expr


class CuteSkinnyGemm:
    """Shape-dynamic low-latency GEMM for small token counts.

    计算 C[M,N] = A[M,K] @ B[N,K].T + residual，支持 BF16/FP16 输入、
    FP32 累加。N 和 K 为运行时值，M 编译时固定并完全展开。
    """

    def __init__(self, *, element_type, num_rows: int, block_size: int,
                 outputs_per_block: int, vector_width: int = 8,
                 k_unroll: int = 1, has_residual: bool = False,
                 use_pdl: bool = False):
        if block_size % cute.arch.WARP_SIZE != 0:
            raise ValueError("block_size must be a multiple of warp size")
        self.num_rows = num_rows
        self.block_size = block_size
        self.outputs_per_block = outputs_per_block
        self.vector_width = vector_width
        self.k_unroll = k_unroll
        self.has_residual = has_residual
        self.use_pdl = use_pdl

    @cute.jit
    def __call__(self, gA: cute.Tensor, gB: cute.Tensor,
                 gResidual: cute.Tensor, gC: cute.Tensor,
                 stream: CUstream) -> None:
```

```
    n = cute.size(gB, mode=[0]) # 运行时 N
    k = cute.size(gA, mode=[1]) # 运行时 K
    # 加载配置原子操作：A 使用 ALWAYS cache，B 使用 STREAMING 以减小 L2 污染
    copy_a = cute.make_copy_atom(cute.nvgpu.CopyG2ROp(), self.element_type,
                                 num_bits_per_copy=self.vector_width * self.element_type.width,
                                 load_cache_mode=cute.nvgpu.LoadCacheMode.ALWAYS)
    copy_b = cute.make_copy_atom(cute.nvgpu.CopyG2ROp(), self.element_type,
                                 num_bits_per_copy=self.vector_width * self.element_type.width,
                                 load_cache_mode=cute.nvgpu.LoadCacheMode.STREAMING)
    self.kernel(gA, gB, gResidual, gC, k, copy_a, copy_b).launch(
        grid=[cute.ceil_div(n, self.outputs_per_block), 1, 1],
        block=[self.block_size, 1, 1],
        smem=self.num_rows * self.outputs_per_block * (self.block_size // cute.arch.WARP_
        SIZE) * 4,
        stream=stream, use_pdl=self.use_pdl, min_blocks_per_mp=1)

@cute.kernel
def kernel(self, gA, gB, gResidual, gC, k_extent: cutlass.Int32,
           copy_a: cute.CopyAtom, copy_b: cute.CopyAtom):
    # 线程局部初始化累加器
    acc_layout = cute.make_layout((self.num_rows, self.outputs_per_block), stride=(self.
    outputs_per_block, 1))
    acc = cute.make_rmem_tensor(acc_layout, cutlass.Float32)
    acc.fill(0.0)
    if const_expr(self.use_pdl):
        cute.arch.griddepcontrol_wait() # PDL 等待前驱
    # 主循环：分块加载 K 维度，执行矩阵乘法
    for k_tile in cutlass.range(k_extent // (self.block_size * self.vector_width), unroll=self.k_
    unroll):
        # 加载 A 和 B 分块到寄存器，执行累加
        ...
    # 写回结果，可选加余量
    if const_expr(self.has_residual):
        cute.add(acc, ...) # 累加前已加载余量
    cute.copy(acc, gC)
```

评论区精华

- 模型注册绕过风险：chatgpt-codex-connector[bot] 指出 KimiLinearForCausalLM 仍映射到旧的 kimi_linear 模块，新定义的 KimiK3ForConditionalGeneration 不会被实例化，导致 K3 特定行为被忽略。（P1 严重性）
- Profiling 未初始化输出：同一 bot 发现 profiling 路径返回 torch.empty，未经写入直接被归一化投影，可能传播 NaN 值；建议恢复零初始化。（P2 严重性）
- CI 测试回归：AndreasKaratzas 报告该 PR 在 Buildkite 上引起了潜在回归，并已创建 revert PR#50304 还原部分测试文件改动。
  - 模型注册映射错误导致 K3 逻辑被绕过 (design): 未在该 PR 中修复，需后续 PR 修正注册表或条件加载逻辑。

- Profiling 路径未初始化输出导致的 NaN 风险 (correctness): 未确认修复；建议恢复零初始化或写入确定性值。
- CI 测试回归与部分文件 revert (testing): 已创建 revert PR #50304 应对回归，但未在此 PR 中直接修复。

风险与影响

- 风险：
  1. 模型注册绕过：若 registry.py 中 KimiLinearForCausalLM 未正确映射到 kimi_k3 模块，系统将使用旧代码运行 K3 模型，导致大量新增逻辑失效。
  2. Profiling 路径未初始化：KDA 的 profiling 分支可能产出 NaN，影响运行时状态，需立即修复。
  3. 跨平台兼容性：AMD 后端和 NVIDIA 后端差异大，AMD 端缺少 KDA 等内核，可能无法正确推理。
  4. 变更范围大：109 个文件 +28k 行新增，容易引入跨模块回归（如调度器或缓存交互）。
  5. 尚不可运行：PR 明确标注 not runnable，合并后可能影响主分支稳定性。 - 影响：对用户：获得 Kimi K3 模型基础支持，但需后续 PR 才能实际使用。对系统：新增约 28k 行代码，增加构建依赖（CuTe DSL、flash-linear-attention），并引入新的 CUDA 内核。对团队：需要多平台维护，且存在因测试 revert 导致的覆盖缺口。影响程度：高，涉及核心模型层和内核层。 - 风险标记：模型注册绕过风险，Profiling 未初始化，CI 测试回归，仅 NVIDIA 验证，尚不可运行，变更范围过大 (109 文件)

关联脉络

- PR #50000 [Model] Add Kimi K3 support (original): 该 PR 是从 #50000 拆分出来的第一部分，原 PR 包含更多前端与运行时集成。
- PR #50304 [Bug] Revert Kimi K3 test changes causing regression: 由于该 PR 在 CI 中引起回归，AndreasKaratzas 创建了该 revert PR 来还原部分测试文件。
- PR #50088 [MoE] Support DeepSeekV3 routing ... (merged into this PR): 从 commit 历史看，该 MoE 支持被合并到本 PR 中（Merge PR #50088）。
- PR #49731 [Spec Decode][Perf] Replicate DSpark Markov head across TP ranks: DSpark Markov head 的复制逻辑被本 PR 的 ReplicatedDSparkMarkovHead 重用。