PR #43477 完整报告

vllm-project/vllm

Enable DeepSeek V4 and GLM-5.1 on SM120

合并时间：2026-06-23 02:54

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43477

执行摘要

- 一句话：为 Blackwell SM120 添加 DSv4 和 GLM-5.1 推理支持
- 推荐动作：该 PR 是面向 Blackwell 用户的重要功能合入，建议以下角色重点关注：
  - 推理引擎开发者：注意力后端基类重构的设计决策值得学习，如何分离 SM100/SM120 代码。
  - 模型优化工程师：kernel warmup 和 autotune 的实现方法可复用。
  - 部署工程师：注意构建依赖（DeepGEMM 和 FlashInfer 的特殊分支）以及推荐的运行参数。

功能与动机

为了在最新的消费级 Blackwell GPU（SM120）上高效运行 DeepSeek V4 和 GLM-5.1 模型，需要支持 FlashInfer 的稀疏 MLA 解码和 DeepGEMM 的 MXFP4 计算。该 PR 解决了此前 SM120 上无法使用这些模型的问题，提供了完整的推理路径和性能优化。

实现拆解

1. SM120 稀疏 MLA 注意力后端 - 新增 vllm/v1/attention/backends/mla/flashinfer_mla_sparse_sm120.py，实现 FlashInferMLASparseSM120Impl，仅支持 fp8_ds_mla KV 缓存和 decoder self-attention，集成 FlashInfer 的 SM120 稀疏 MLA API。 - 修改 flashinfer_mla_sparse.py，引入基类 _FlashInferMLASparseBackendBase，并创建 FlashInferMLASparseSM120Backend 和 FlashInferMLASparseTRTLLMBackend 分别对应 SM120 和 SM100，分离通用代码。
2. DeepSeek V4 模型适配 SM120 - 修改 vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py，在为 DeepseekV4FlashInferMLASparseBackend 扩展 supports_compute_capability（SM10x 和 SM12x）、supports_combination（区分 SM10/12 的 KV 缓存类型）、get_kv_cache_shape 等方法。 - 新增 DeepseekV4FlashInferSM120Attention 类（位于 flashinfer_sparse.py 中），独立处理 SM120 的 forward 逻辑，避免与 SM100 代码混合。 - 修改 vllm/models/deepseek_v4/nvidia/model.py 的 _select_dsv4_attn_cls 函数，将 FLASHINFER_MLA_SPARSE_SM120 后端路由到上述新类。
3. Kernel Warmup 与 Autotune - 新增 vllm/model_executor/warmup/flashinfer_sparse_mla_warmup.py，提供 _run_flashinfer_sparse_mla_decode_autotune 和

_flashinfer_sparse_mla_decode_autotune 函数，用于在服务启动前预编译 FlashInfer 稀疏 MLA 解码内核并执行 autotune（通过 flashinfer 的 AutoTuner）。 - 新增 deepseek_v4_mhc_warmup.py，为 DeepSeek V4 的 mHC 层提供 TileLang kernel warmup，通过查找模型中的 hc_* 属性自动触发。 - 新增 flashinfer_autotune_cache.py，管理 autotune 缓存哈希和文件读写，避免重复 autotune。 - 修改 vllm/model_executor/warmup/kernel_warmup.py，将新 warmup 函数集成到现有的 warmup 流程中。

4. DeepGEMM 与 MoE 支持 - 修改 vllm/utils/deep_gemm.py，将设备能力检查从仅 SM100 扩展到 SM120，并添加 pack_ue8m0_to_int、get_mn_major_tma_aligned_packed_ue8m0_tensor 等函数，用于 MXFP4 权重的紧凑格式转换。 - 修改 vllm/model_executor/layers/fused_moe/deep_gemm_utils.py，重构 compute_aligned_M 并添加 compute_aligned_M_and_alignment，优化 SM120 上的 Mk 对齐。 - 修改 vllm/model_executor/layers/fused_moe/oracle/mxfp4.py，使用 DeepGEMM 的 pack 函数代替手工转换，删除调试日志。
5. 构建系统与平台适配 - 修改 cmake/external_projects/deepgemm.cmake，添加 SM120 架构选项（根据 CUDA 版本设置 12.0f 或 12.0a;12.1a），并支持本地 / 远程源码构建。 - 修改 vllm/platforms/cuda.py，在 AttentionBackendEnum 中添加 FLASHINFER_MLA_SPARSE_SM120，并在 _get_attn_backend_class 中为其分发到对应的后端类。 - 修改 vllm/utils/flashinfer.py，添加 has_flashinfer_sparse_mla_sm120 函数检查 FlashInfer 是否支持 SM120 稀疏 MLA。
6. 测试文件（部分被移除） - 最初新增了 tests/models/test_deepseek_v4_backend_selection.py、tests/test_deep_gemm_wrapper.py、tests/model_executor/test_flashinfer_autotune_cache.py，但在 review 中被 reviewer 要求移除，作者已执行。剩下一个测试文件 v1/engine/test_deepseek_v4_sparse_swa.py 用于 sparse SWA 的测试。

关键文件：

- vllm/v1/attention/backends/mla/flashinfer_mla_sparse.py（模块 注意力后端；类别 source；类型 core-logic；符号 FlashInferMLASparseBackend, get_supported_kernel_block_sizes, _FlashInferMLASparseBackendBase, get_impl_cls）：核心文件，重构了注意力后端基类，分离了 SM100 和 SM120 的后端实现，是 SPARSE_MLA_SM120 后端的入口。
- vllm/v1/attention/backends/mla/flashinfer_mla_sparse_sm120.py（模块 注意力后端；类别 source；类型 core-logic；符号 _kv_scale_format_for_model, FlashInferMLASparseSM120Impl, init, forward_mqa）：新增的 SM120 具体注意力实现，负责 FlashInfer 稀疏 MLA 解码的前向传播和缓存格式转换。
- vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py（模块 模型层；类别 source；类型 core-logic；符号 get_supported_kernel_block_sizes, get_supported_head_sizes, supports_sink, is_sparse）：DSv4 模型注意力类的容器，扩展了 backend 的兼容性检查、缓存形状和 SM120 专用的注意力类。
- vllm/model_executor/warmup/flashinfer_sparse_mla_warmup.py（模块 预热模块；类别 source；类型 core-logic；符号 _attention_backend_name, _has_deepseek_v4_sparse_mla_backend, _flashinfer_sparse_mla_decode_label,

_clamp_warmup_tokens）：新增的 FlashInfer 稀疏 MLA warmup 模块，负责 autotune 和混合步预热，对首次请求延迟至关重要。

- vllm/utils/deep_gemm.py（模块 工具库；类别 source；类型 core-logic；符号 get_theoretical_mk_alignment_for_contiguous_layout, set_mk_alignment_for_contiguous_layout, mk_alignment_scope, pack_ue8m0_to_int）：DeepGEMM 工具层扩展，添加 SM120 平台检测和 UE8M0 紧凑格式支持。
- vllm/platforms/cuda.py（模块 平台适配；类别 source；类型 core-logic；符号 _backend_cls_path, _get_attn_backend_class, _BackendCandidate）：CUDA 平台注意力后端选择，新增 FLASHINFER_MLA_SPARSE_SM120 枚举并路由到正确后端类。

关键符号：_attention_backend_name, _has_deepseek_v4_sparse_mla_backend, _flashinfer_sparse_mla_decode_label, _clamp_warmup_tokens, _uses_v2_model_runner, _run_flashinfer_sparse_mla_decode_autotune, _flashinfer_sparse_mla_decode_autotune, _deepseek_v4_sparse_mla_decode_autotune, _compute_mhc_pre_num_split, _normalize_token_sizes, _select_mhc_warmup_token_sizes, _find_first_mhc_layer, _find_deepseek_v4_model, _warmup_layer_mhc, _warmup_hc_head, deepseek_v4_mhc_warmup, get_supported_kernel_block_sizes, get_supported_head_sizes, supports_sink, is_sparse, supports_compute_capability, supports_combination, get_kv_cache_shape, DeepseekV4FlashInferSM120Attention, _kv_scale_format_for_model, FlashInferMLASparseSM120Impl, FlashInferMLASparseBackend, get_supported_kernel_block_sizes, _FlashInferMLASparseBackendBase, get_impl_cls, FlashInferMLASparseTRTLLMBackend, FlashInferMLASparseSM120Backend, get_name, supports_compute_capability, get_theoretical_mk_alignment_for_contiguous_layout, set_mk_alignment_for_contiguous_layout, mk_alignment_scope, pack_ue8m0_to_int, get_mn_major_tma_aligned_packed_ue8m0_tensor, get_k_grouped_mn_major_tma_aligned_packed_ue8m0_tensor, compute_aligned_M, compute_aligned_M_and_alignment, round_up_128, run_mixed_prefill_decode_warmup, _alloc_blocks, _backend_cls_path, _get_attn_backend_class, _BackendCandidate

关键源码片段

vllm/v1/attention/backends/mla/flashinfer_mla_sparse.py

核心文件，重构了注意力后端基类，分离了 SM100 和 SM120 的后端实现，是 SPARSE_MLA_SM120 后端的入口。

```
# vllm/v1/attention/backends/mla/flashinfer_mla_sparse.py

class _FlashInferMLASparseBackendBase(AttentionBackend):
    """Common metadata for concrete FlashInfer sparse MLA backends."""

    # 所有稀疏 MLA 后端共享的 metadata builder
    @staticmethod
    def get_builder_cls() -> type[FlashInferMLASparseMetadataBuilder]:
        return FlashInferMLASparseMetadataBuilder
```

```
    @classmethod
    def is_mla(cls) -> bool:
        return True

    @classmethod
    def is_sparse(cls) -> bool:
        return True


class FlashInferMLASparseTRTLLMBackend(_FlashInferMLASparseBackendBase):
    """SM100 path: uses TRTLLM-gen launcher with bf16/fp8 KV cache."""
    supported_dtypes = [torch.float16, torch.bfloat16]
    supported_kv_cache_dtypes = ["auto", "float16", "bfloat16", "fp8", "fp8_e4m3"]

    @staticmethod
    def get_supported_kernel_block_sizes() -> list[int | MultipleOf]:
        return [32, 64]

    @staticmethod
    def get_impl_cls() -> type[SparseMLAAttentionImpl]:
        return FlashInferMLASparseImpl


class FlashInferMLASparseSM120Backend(_FlashInferMLASparseBackendBase):
    """SM120 path: uses FlashInfer sparse MLA decode API with fp8_ds_mla cache."""
    supported_dtypes = [torch.bfloat16]
    supported_kv_cache_dtypes = ["fp8_ds_mla"]

    @staticmethod
    def get_supported_kernel_block_sizes() -> list[int | MultipleOf]:
        return [256]

    @staticmethod
    def get_impl_cls() -> type[SparseMLAAttentionImpl]:
        return FlashInferMLASparseSM120Impl
```

vllm/v1/attention/backends/mla/flashinfer_mla_sparse_sm120.py

新增的 SM120 具体注意力实现，负责 FlashInfer 稀疏 MLA 解码的前向传播和缓存格式转换。

```
# vllm/v1/attention/backends/mla/flashinfer_mla_sparse_sm120.py

class FlashInferMLASparseSM120Impl(SparseMLAAttentionImpl[FlashInferMLASparseMetadata])
:
    """SM120 FlashInfer sparse-MLA implementation."""

    def __init__(self, num_heads, head_size, scale, num_kv_heads, ..., indexer=None, **mla_args)
    :
        # SM120 只支持 fp8_ds_mla 格式，且不允许 alibi_slides / sliding_window
        if self.kv_cache_dtype != "fp8_ds_mla":
```

```
            raise NotImplementedError(
                "SM120 requires fp8_ds_mla KV cache layout")
        # 检查 FlashInfer 是否支持 SM120 稀疏 MLA
        if not has_flashinfer_sparse_mla_sm120():
            raise RuntimeError("FlashInfer sparse MLA SM120 API not available")

    def forward_mqa(self, q, kv_c_and_k_pe_cache, attn_metadata, layer):
        # 将 topk_indices 转换为物理索引
        topk_indices_physical = triton_convert_req_index_to_global_index(
            attn_metadata.req_id_per_token[:num_actual_toks],
            attn_metadata.block_table,
            topk_indices,
            BLOCK_SIZE=attn_metadata.block_size,
            NUM_TOPK_TOKENS=topk_indices.shape[1],
        )
        # 分配输出 tensor
        output = q.new_empty((num_actual_toks, self.num_heads, self.kv_lora_rank), dtype=q.
        dtype)
        # 调用 FlashInfer 的 SM120 sparse MLA 解码函数
        flashinfer_trtllm_batch_decode_sparse_mla_sm120(
            output, q, kv_c_and_k_pe_cache, topk_indices_physical, ...)
        return output, None
```

vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py

DSv4 模型注意力类的容器，扩展了 backend 的兼容性检查、缓存形状和 SM120 专用的注意力类。

```
# vllm/models/deepseek_v4/nvidia/flashinfer_sparse.py

class DeepseekV4FlashInferMLASparseBackend(DeepseekV4FlashMLABackend):
    # ...
    @classmethod
    def supports_compute_capability(cls, capability: DeviceCapability) -> bool:
        return capability.major in [10, 12]

    @classmethod
    def supports_combination(cls, head_size, dtype, kv_cache_dtype, ...):
        if capability.major == 10:
            # SM100 允许 auto / bf16 / fp8
            return None
        if capability.major == 12:
            # SM120 必须使用 fp8_ds_mla
            if kv_cache_dtype not in ("fp8", "fp8_e4m3", "fp8_ds_mla"):
                return "kv_cache_dtype not supported"
            # 检查 FlashInfer 是否支持 SM120
            if not has_flashinfer_sparse_mla_sm120():
                return "SM120 support not available in FlashInfer"
            return None
        return "requires SM10x or SM12x"
```

```
@staticmethod
def get_kv_cache_shape(num_blocks, block_size, num_kv_heads, head_size, cache_dtype_str=
"auto"):
    device_capability = current_platform.get_device_capability()
    if device_capability is not None and device_capability.major == 12:
        # SM120 使用与 FlashMLA V4 相同的 packed shape
        return DeepseekV4FlashMLABackend.get_kv_cache_shape(...)
    else:
        # SM100 使用简单的 (num_blocks, block_size, head_size)
        assert num_kv_heads == 1
        return (num_blocks, block_size, head_size)
```

评论区精华

1. Warmup 后端名称遗漏：gemini-code-assist 指出 _DEEPSEEK_V4_SPARSE_MLA_BACKENDS 缺少新添加的 SM120 后端名称，导致 warmup 跳过。作者后续提交中已更新该集合，加入 FLASHINFER_MLA_SPARSE_DSV4。
2. V0 Runner 兼容性：gemini-code-assist 指出 warmup 函数只检查 V1 的 attn_groups，对 V0 runner 不可用。作者 later 移除了对 V0 的支持，只保留 V1 路径。
3. 代码分离与重构：zyongye 多次要求将 SM120 实现与 SM100 代码完全分离，避免混杂。作者最终将 flashinfer_sparse.py 中的 SM120 逻辑独立到新的 flashinfer_mla_sparse_sm120.py 和 flashinfer_mla_sparse_sm120_attention.py。
4. 删除多余测试：zyongye 认为 test_deepseek_v4_backend_selection.py 等测试文件不必要，作者同意并删除。
5. 依赖上游未发布：PR 依赖 FlashInfer PR#3395 和 DeepGEMM PR#324 尚未合并，reviewer 询问是否必须，作者确认这些是硬依赖。

- Warmup 后端名称遗漏 (correctness): 作者后续提交将新后端名称加回集合。
- V0 Runner 兼容性 (correctness): 作者移除了对 V0 的支持并只保留 V1 路径。
- SM100 与 SM120 代码分离 (design): 作者将 SM120 独立为 flashinfer_mla_sparse_sm120.py 和 DeepseekV4FlashInferSM120Attention 类，基类重构为 _FlashInferMLASparseBackendBase。
- SWA 索引函数的来源 (design): 作者尝试后认为本地 Triton 内核同样有效且更通用，最终切换回原始 Triton 实现。
- 多余测试文件的增删 (testing): 作者删除了这些测试文件。

风险与影响

- 风险：
  1. 回归风险：对注意力后端基类 FlashInferMLASparseBackend 的重构可能影响现有 SM100 用户的推理路径。尽管引入了向后兼容的 FlashInferMLASparseTRTLLMBackend，但未经过充分测试。
  2. 首次请求 JIT 延迟：如果 warmup 未覆盖所有内核，首次请求可能触发 JIT 编译导致高延迟。虽然新增了专门的 warmup，但可能仍不完整。

3. 外部依赖未稳定：依赖 FlashInfer 和 DeepGEMM 的未合并分支，上游 API 可能变化，导致后续维护困难。
4. 性能差异：提供的数据仅针对特定硬件和配置，在其他 SM120 设备（如 RTX 5090）或不同 CUDA 版本上可能有差异。
5. 代码复杂度增加：大量新增 warmup 和 autotune 逻辑可能增加系统加载时间，尤其在多 GPU 环境下。

- 影响：
  1. 用户：DeepSeek V4 和 GLM-5.1 用户现在可以在 Blackwell GPU 上使用高质量推理，并获得高达 633 tok/s 的吞吐（TP=2, MTP=2）。
  2. 系统：增加了约 2.3K 行源码，新增多个 warmup 模块，对首次启动延迟有影响；但通过 autotune 缓存可减少后续影响。
  3. 团队：需要维护 SM120 专用后端和 warmup，跟踪上游 FlashInfer 和 DeepGEMM 的变化。注意力后端基类重构为后续支持更多架构提供了模板。 - 风险标记：依赖上游未合并分支，首次请求 JIT 延迟风险，SM100 回归风险，构建系统变更，大量新代码引入复杂性

关联脉络

- PR #46800 Rust Frontend Add Harmony Renderer for GPT-OSS: 虽然不直接相关，但同属代码库的前端渲染部分，且两者都没有相互依赖。