

PR #54566 完整报告

vllm-project/vllm

[New model][Multimodal] Add DeepSeek-V4-Flash-Vision-Exp support

合并时间: 2026-09-03 02:09

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54566>

执行摘要

- 一句话: 为 DeepSeek-V4-Flash-Vision-Exp 多模态模型添加完整的推理支持, 涵盖视觉编码、多模态处理、注意力适配与 MoE 路由。
- 推荐动作: 本 PR 是一个里程碑级别的变更, 引入了完整的多模态模型支持, 涉及多个核心模块的协同修改。建议精读以下关键部分: 1) 多模态预处理中的哨兵令牌设计; 2) 注意力后端中图像可见性内核的实现; 3) MoE 路由器对图像令牌的适配。这些设计决策展示了如何在不破坏现有架构的前提下扩展新功能, 值得借鉴。

功能与动机

官方 DeepSeek-V4-Flash-Vision-Exp 模型提供了出色的视觉理解能力, 但 vLLM 此前不支持。本 PR 旨在将官方实现移植到 vLLM 框架, 使社区能够使用高性能推理引擎运行该模型。从 PR body 和评论中可知, 作者独立验证了图像预处理与参考实现的完全匹配, 并得到了其他贡献者的测试验证。

实现拆解

1. 多模态预处理模块: 新增 `vllm/models/deepseek_v4/common/mm_preprocess.py`, 实现图像变换、哨兵令牌块构建和处理器。关键符号包括 `image_sentinel_mask`、`grid_tokens`、`load_image` 等。该模块将图像占位符扩展为保留的 `in-vocab` 特殊令牌序列, 并确保令牌计数与参考实现一致。
2. 视觉编码器与对齐器: 新增 `vllm/models/deepseek_v4/common/vision.py`, 实现 DeepSeek-V4 ViT 和 Aligner 网络, 负责图像特征提取和下采样投影。关键类包括 `DeepseekV4ViT`、`DeepseekV4Aligner`。线性层支持张量并行和数据并行。
3. 多模态模型包装器: 新增 `vllm/models/deepseek_v4/nvidia/vl_model.py`, 实现 `DeepseekV4ForConditionalGeneration`, 作为多模态入口点, 整合视觉编码器、学习向量 (`image_start/pad` 等) 和文本骨干。通过 `embed_multimodal` 和 `embed_input_ids` 处理图像嵌入。
4. 注意力后端适配: 修改 `vllm/v1/attention/backends/mla/sparse_swa.py`, 增加图像可见性计算。新增 Triton 内核 `_compute_image_visibility_kernel` 和 `_compute_swa_indices_and_lens_kernel`, 在预填充时双向扩展滑动窗口以覆盖图像区域。元数据类新增 `prefill_left_visible` 和 `prefill_right_visible` 字段。

5. 架构路由与 MoE 适配：在 `vllm/transformers_utils/model_arch_config_convertor.py` 中新增 `DeepseekV4ModelArchConfigConvertor`，将带有视觉塔的配置路由到 VL 包装器。修改 `vllm/model_executor/layers/fused_moe/router/fused_topk_bias_router.py`，支持图像哨兵令牌使用 `bias_vl` 进行专家选择，确保视觉令牌的正确路由。
6. 测试与配套：新增多个测试文件，包括注意力可见性测试 (`test_deepseek_v4_swa_visible.py`)、MoE 路由器测试、配置转换测试等，确保功能正确性。修改现有测试以覆盖新场景。

关键文件：

- `vllm/models/deepseek_v4/nvidia/vl_model.py`（模块 VL 模型；类别 `source`；类型 `core-logic`；符号 `DeepseekV4ForConditionalGeneration`, `init`, `embed_multimodal`, `embed_input_ids`）：多模态模型包装器，整合视觉编码器和文本骨干，是新模型的主入口点。
- `vllm/models/deepseek_v4/common/mm_preprocess.py`（模块 多模态预处理；类别 `source`；类型 `data-contract`；符号 `image_sentinel_mask`, `validate_image_sentinel_ids`, `grid_tokens`, `load_image`）：多模态预处理核心，实现图像变换和哨兵令牌块构建，确保与参考实现完全匹配。
- `vllm/v1/attention/backends/mla/sparse_swa.py`（模块 注意力后端；类别 `source`；类型 `core-logic`；符号 `_build_image_visibility`, `_compute_image_visibility_kernel`, `_compute_swa_indices_and_lens_kernel`）：注意力后端关键修改，实现图像双向可见性计算，支持多模态预填充。

关键符号：`image_sentinel_mask`, `validate_image_sentinel_ids`, `grid_tokens`, `load_image`, `build_image_block`, `DeepseekV4ForConditionalGeneration`, `embed_multimodal`, `embed_input_ids`, `_build_image_visibility`, `_compute_image_visibility_kernel`, `_compute_swa_indices_and_lens_kernel`, `fused_topk_bias`, `dsv4_topk`, `flatten_content_blocks`

关键源码片段

`vllm/models/deepseek_v4/nvidia/vl_model.py`

多模态模型包装器，整合视觉编码器和文本骨干，是新模型的主入口点。

```
class DeepseekV4ForConditionalGeneration(
    nn.Module, SupportsMultiModal, SupportsPP, SupportsEagle3
):
    """Multimodal entry point for DeepSeek-V4 checkpoints with a vision tower."""

    # MoE router needs raw token ids for image sentinel detection
    requires_raw_input_tokens = True

    def __init__(self, *, vllm_config, prefix: str = "") -> None:
        super().__init__()
        model_config = vllm_config.model_config
        config = model_config.hf_config
        # Vision tower enabled only when config has vision layers and limit > 0
        image_enabled = (
```

```

        config.vision_n_layers > 0
        and self.multimodal_config.get_limit_per_prompt("image") > 0
    )
    with self._mark_tower_model(vllm_config, {"image"}):
        self.vision: DeepseekV4ViT | None = None
        self.aligner: DeepseekV4Aligner | None = None
        # Learned vectors for sentinel tokens (start, pad, newline, end)
        self.image_start: nn.Parameter | None = None
        self.image_end: nn.Parameter | None = None
        self.image_newline: nn.Parameter | None = None
        self.image_pad: nn.Parameter | None = None
        if image_enabled:
            self.vision = DeepseekV4ViT(config)
            self.aligner = DeepseekV4Aligner(config)
            for name in (
                "image_start", "image_end", "image_newline", "image_pad"
            ):
                setattr(
                    self, name,
                    nn.Parameter(torch.empty(config.hidden_size, dtype=torch.float32))
                )
        # Initialize text backbone with marker to avoid recursion
        config._dsv4_vl_inner = True
        self.language_model = init_vllm_registered_model(
            vllm_config=vllm_config, hf_config=config,
            prefix=maybe_prefix(prefix, "language_model"),
            architectures=["DeepseekV4ForCausalLM"],
        )
        del config._dsv4_vl_inner

def embed_input_ids(
    self, input_ids: torch.Tensor,
    multimodal_embeddings: MultiModalEmbeddings | None = None, *,
    is_multimodal: torch.Tensor | None = None,
) -> torch.Tensor:
    # Merge vision embeddings into sentinel positions
    inputs_embeds = self.language_model.embed_input_ids(input_ids)
    if self.image_start is not None:
        sentinel_mask = image_sentinel_mask(input_ids)
        if is_multimodal is not None:
            # IMAGE positions get vision embeddings via merge below
            sentinel_mask = sentinel_mask & ~is_multimodal.to(input_ids.device)
        # Table of learned vectors for sentinel types
        table = torch.stack([
            self.image_start, self.image_pad, self.image_pad,
            self.image_newline, self.image_end,
        ]).to(inputs_embeds.dtype)
        idx = (input_ids - IMAGE_SENTINEL_BASE_ID).clamp(0, 4)
        inputs_embeds = torch.where(

```

```

        sentinel_mask.unsqueeze(-1), table[idx], inputs_embeds
    )
    # Merge vision embeddings (for IMAGE positions)
    if multimodal_embeddings is None or len(multimodal_embeddings) == 0:
        return inputs_embeds
    return _merge_multimodal_embeddings(
        inputs_embeds=inputs_embeds,
        multimodal_embeddings=multimodal_embeddings,
        is_multimodal=is_multimodal,
    )

```

vllm/models/deepseek_v4/common/mm_preprocess.py

多模态预处理核心，实现图像变换和哨兵令牌块构建，确保与参考实现完全匹配。

```

# Sentinel roles: five consecutive reserved tokens never emitted from plain text
IMAGE_SENTINEL_BASE_ID = 129257
IMAGE_SENTINEL_TOKEN_NAMES = (
    "<|place_holder_mm_span_0431|>", # IMAGE_START
    "<|place_holder_mm_span_0432|>", # IMAGE_PAD
    "<|place_holder_mm_span_0433|>", # IMAGE
    "<|place_holder_mm_span_0434|>", # IMAGE_NEW_LINE
    "<|place_holder_mm_span_0435|>", # IMAGE_END
)

def image_sentinel_mask(token_ids: torch.Tensor) -> torch.Tensor:
    """Boolean mask for image-block sentinel positions (in-vocab ids)."""
    return (token_ids >= IMAGE_SENTINEL_BASE_ID) & (
        token_ids < IMAGE_SENTINEL_BASE_ID + len(IMAGE_SENTINEL_TOKEN_NAMES)
    )

def validate_image_sentinel_ids(tokenizer) -> None:
    """Check the borrowed sentinel ids against the tokenizer."""
    for i, name in enumerate(IMAGE_SENTINEL_TOKEN_NAMES):
        token_id = tokenizer.convert_tokens_to_ids(name)
        if token_id != IMAGE_SENTINEL_BASE_ID + i:
            raise ValueError(
                f"Image sentinel token {name!r} has id {token_id}, expected "
                f"{IMAGE_SENTINEL_BASE_ID + i}; the DeepSeek-V4 vision "
                "sentinel block requires these consecutive reserved ids."
            )

def build_image_block(n_llm_h: int, n_llm_w: int, start_pos: int):
    """Builds the N-layout token types (final order) and aligner-row order."""
    compress_pad = COMPRESS_PAD_TO - 1 - start_pos % COMPRESS_PAD_TO
    pad_h = n_llm_h % 2
    rows = n_llm_h + pad_h
    row_len = n_llm_w + 1
    pad_last = rows // 2 * row_len % 2 * 2
    types = torch.tensor(

```

```

        ([IMAGE] * n_llm_w + [IMAGE_NEW_LINE]) * n_llm_h
        + [IMAGE_PAD] * (row_len * pad_h),
        dtype=torch.int64,
    )
    order = torch.arange(rows * row_len).view(rows // 2, 2, row_len)
    order = order.transpose(1, 2).reshape(-1)
    image_idx = torch.full((rows * row_len,), -1, dtype=torch.int64)
    image_idx.view(rows, row_len)[:n_llm_h, :n_llm_w] = torch.arange(
        n_llm_h * n_llm_w
    ).view(n_llm_h, n_llm_w)
    perm = image_idx[order]
    perm = perm[perm >= 0]
    types = torch.cat([
        torch.full((compress_pad,), IMAGE_PAD, dtype=torch.int64),
        torch.tensor([IMAGE_START]),
        types[order],
        torch.full((pad_last,), IMAGE_PAD, dtype=torch.int64),
        torch.tensor([IMAGE_END]),
    ])
    return types, perm

```

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

注意力后端关键修改，实现图像双向可见性计算，支持多模态预填充。

```

def _build_image_visibility(
    self, num_reqs, mm_ranges, num_decode_tokens, num_prefill_tokens,
    seq_lens, query_start_loc, token_to_req_indices,
) -> tuple[torch.Tensor, torch.Tensor]:
    """Upload image spans and compute per-token in-image visibility."""
    indptr = [0] * (num_reqs + 1)
    starts, ends = [], []
    for req_idx in range(num_reqs):
        for span_start, span_end in mm_ranges.get(req_idx, ()):
            starts.append(span_start)
            ends.append(span_end)
        indptr[req_idx + 1] = len(starts)
    num_spans = len(starts)
    # Copy span CSR to device buffers
    self.span_indptr[: num_reqs + 1].copy_(torch.tensor(indptr, dtype=torch.int32))
    if num_spans > 0:
        self.span_starts[:num_spans].copy_(torch.tensor(starts, dtype=torch.int32))
        self.span_ends[:num_spans].copy_(torch.tensor(ends, dtype=torch.int32))
    # Launch kernel to compute left/right visibility for prefill tokens
    _compute_image_visibility_kernel[(num_prefill_tokens,)](
        self.left_visible, self.right_visible,
        self.span_indptr, self.span_starts, self.span_ends,
        query_start_loc, seq_lens, token_to_req_indices,
        self.max_image_tokens, token_offset=num_decode_tokens,
    )

```

```

end = num_decode_tokens + num_prefill_tokens
return self.left_visible[:end], self.right_visible[:end]

# In metadata builder __init__:
self.max_image_tokens = (
    getattr(hf_config, "vision_max_n_token", 0)
    if getattr(hf_config, "vision_n_layers", 0) > 0
    else 0
)
self.prefill_index_width = self.window_size + self.max_image_tokens
# Prefill SWA indices widened to include image tokens
self.prefill_swa_indices = torch.zeros(
    max_tokens, 1, self.prefill_index_width, dtype=torch.int32, device=self.device
)

```

评论区精华

- 架构转换器路由错误导致视觉塔未构建 (correctness): 作者在 `__init__` 中直接修改 `hf_config.architectures` 属性，确保模型类解析正确。
- 哈希层门偏置缺失导致权重加载失败 (correctness): 作者在权重映射后缀规则中添加 `.fnn.gate.bias` 到 `.ffn.gate.e_score_correction_bias` 的映射。
- SM120 硬件上解码失败和预填充内核不支持 (performance): 解码问题由主分支修复 (PR#53574)，预填充问题需 FlashInfer 更新 (flashinfer-ai/flashinfer#4850)。
- 图像预处理与参考实现的一致性验证 (correctness): 验证了算法正确性，确认 `grid_tokens`、`solve_resize_ratio`、`build_image_block` 等函数与参考一致。

风险与影响

- 风险：
 1. 算法正确性风险：图像预处理和哨兵令牌生成必须与参考实现完全一致，否则会导致模型输出偏差。作者已通过对比验证，但边界情况（如极端宽高比）仍需监控。
 2. 注意力后端回归风险：修改 `_COMPUTE_SWA_INDICES_AND_LENS_KERNEL` 以支持图像可见性可能影响文本模式的性能。测试确保了 `HAS_IMAGE=False` 时行为不变。
 3. MoE 路由兼容性风险：图像哨兵令牌的专家选择使用 `bias_vl`，若令牌 ID 范围判断错误可能导致路由错误。测试覆盖了混合令牌场景。
 4. 跨平台兼容性风险：视觉编码器仅支持 NVIDIA GPU (SM120+)，AMD 等平台通过 `vl_stub.py` 抛出 `NotImplementedError`。未来需扩展支持。
 5. 性能风险：图像可见性计算和扩展的滑动窗口索引可能增加预填充延迟，但测试表明在预期范围内。
 - 影响：对用户：可以部署和运行 DeepSeek-V4-Flash-Vision-Exp 模型进行图像理解，扩展了 vLLM 的多模态能力。对系统：引入了新的模型组件和注意力后端修改，增加了代码库的复杂性，但模块化设计保持了清晰性。对团队：需要维护视觉编码器、多模态处理器和相关测试，未来可能需要扩展到其他硬件平台。对生态：增强了 vLLM 在多模态推理领域的竞争力，可能吸引更多用户。
 - 风险标记：多模态处理器与参考实现对齐，注意力后端图像可见性计算，MoE 路由器图像哨兵专家选择，跨平台兼容性

关联脉络

- PR #45091 Fix DeepSeek V4 FlashMLA auto KV cache dtype: 修复 DeepSeek V4 FlashMLA 在默认 auto KV 缓存类型下的启动问题，与本 PR 共享模型基础架构。
- PR #54901 [Perf][Model Runner V2] Compact sampling masks on GPU instead of unpacking the full-vocab bitmask on CPU: 优化采样掩码紧凑操作，可能影响多模态推理的性能。
- PR #51381 [Core][KV Events] Echo session_id on GPU BlockStored events: 为 GPU BlockStored 事件添加 session_id 字段，支持会话级 KV 缓存观察，与本 PR 的多模态支持在 KV 缓存管理上有交集。
- PR #54908 [Bugfix][DCP] Materialize prefill keys on non-owner ranks: 修复 DCP 模式下融合内核在非所有者 rank 上未生成 K 输出的问题，可能影响 DeepSeek V4 的分布式推理。