PR #32104 完整报告

sgl-project/sglang

[EPD][VLM] Fix Kimi-VL 2D encoder grids

合并时间：2026-07-29 11:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32104

执行摘要

- 一句话：修复 Kimi-VL 二维网格在 EPD 中的支持
- 推荐动作：该 PR 修复了一个模型特定但影响生产链路的问题，设计清晰（属性优先级 + 降维辅助函数），测试完备。值得精读，尤其关注如何在不破坏现有模型前提下扩展网格格式。

功能与动机

Kimi-VL 的图像网格是二维 (h, w) 而非 Qwen 风格的三维 (t, h, w)，并且编码器输出的网格元数据包含 NumPy 对象，在跨进程 Pickle 序列化时会被安全反序列化器拒绝，导致 EPD 模式下推理失败。PR body 中指出 "Kimi-VL image grids are 2-D (h, w), while SGLang's batched encoder helper assumed Qwen-style 3-D (t, h, w) grids. Kimi-VL grid metadata from the encoder must be converted away from NumPy objects before pickling"。

实现拆解

1. 修改网格属性选择逻辑(_get_mm_grid_dim): 在 python/sglang/srt/disaggregation/encode_server.py 中，为 Kimi-VL 和 Kimi-K2.5 分别定义不同的属性查找优先级（Kimi-VL 优先 image_grid_hws，Kimi-K2.5 优先 grid_thws），并通过 _convert 函数统一将 NumPy 数组转为 torch.Tensor，确保 Pickle 安全。
2. 新增辅助方法(_kimi_hw_from_patch_grid): 静态方法，接受 2D 或 3D 网格，统一提取最后两维作为 (h, w)，支持 torch.Tensor、np.ndarray、list 等输入类型，并校验维度合法性。
3. 扩展 get_num_patches: 当模型为 Kimi-VL 且模态为图片时，使用 _kimi_hw_from_patch_grid 计算 patch 数量 (h*w)，而非默认的 grid[0]*grid[1]*grid[2]。
4. 重构 _kimi_tokens_from_patch_grid: 内部调用 _kimi_hw_from_patch_grid 获取 h/w，去除对 3D 网格的假设。
5. 新增单元测试文件(test/registered/unit/disaggregation/test_encode_server.py): 覆盖网格选择、计数、切片、序列化安全等 6 个测试函数，注册为 CPU CI。
6. 扩展端到端测试(test/registered/disaggregation/test_epd_disaggregation.py): 新增 TestEPDDisaggregationKimiVL 类，启动真实 EPD 三节点服务验证多图片对话功能，CI 估计时间从 97s 提升至 300s。

关键文件：

- python/sglang/srt/disaggregation/encode_server.py（模块 编码服务器；类别 source；类型 core-logic；符号 _get_mm_grid_dim, get_num_patches,

_kimi_hw_from_patch_grid, _kimi_tokens_from_patch_grid）：核心修复文件，修改网格属性获取逻辑、新增辅助方法、扩展补丁计数和分片支持。

- test/registered/unit/disaggregation/test_encode_server.py（模块 网格测试；类别 test；类型 test-coverage；符号 TestKimiVLEPDGrid, _make_encoder, test_kimi_vl_prefers_and_normalizes_hw_grid, test_kimi_k25_keeps_thw_grid_preference）：新增单元测试覆盖 Kimi-VL 2D 网格的全部行为，是保证回归的重要配套。
- test/registered/disaggregation/test_epd_disaggregation.py（模块 EPD 测试；类别 test；类型 test-coverage；符号 TestEPDDisaggregationKimiVL, setUpClass, start_encode, start_prefill）：新增 E2E 测试类 TestEPDDisaggregationKimiVL，在真实服务环境中验证多图片对话。

关键符号：_get_mm_grid_dim, get_num_patches, _kimi_hw_from_patch_grid, _kimi_tokens_from_patch_grid, _convert

关键源码片段

python/sglang/srt/disaggregation/encode_server.py

核心修复文件，修改网格属性获取逻辑、新增辅助方法、扩展补丁计数和分片支持。

```
# 获取多模态网格维度，根据模型类型调整属性优先级
def _get_mm_grid_dim(mm_inputs, modality, model_type: Optional[str] = None):
    attrs = _mm_grid_attrs[modality]
    model_type = (model_type or "").lower()
    if modality == Modality.IMAGE:
        # Kimi K2.5 发射 grid_thws，Kimi-VL 发射 image_grid_hws
        if model_type == "kimi_k25":
            attrs = ("grid_thws", "image_grid_thw", "image_grid_hws")
        elif model_type == "kimi_vl":
            attrs = ("image_grid_hws", "image_grid_thw", "grid_thws")
    for attr in attrs:
        if attr in mm_inputs and mm_inputs[attr] is not None:
            return _convert(mm_inputs[attr]) # 确保返回 torch.Tensor
    raise ValueError(f"Grid dim ({_mm_grid_attrs[modality]}) not found in {mm_inputs}")


@staticmethod
def _kimi_hw_from_patch_grid(
    grid: Union[torch.Tensor, np.ndarray, List[int], Tuple[int, ...]],
) -> Tuple[int, int]:
    """从 Kimi 2D 或 3D 网格元数据中提取 (height, width)。"""
    if isinstance(grid, torch.Tensor):
        values = grid.flatten().tolist()
    elif isinstance(grid, np.ndarray):
        values = grid.reshape(-1).tolist()
    else:
        values = np.asarray(grid).reshape(-1).tolist()
```

```
    if len(values) not in (2, 3):
        raise ValueError(
            f"Invalid Kimi image grid metadata: {values}; "
            "expected [h, w] or [t, h, w]"
        )
    return int(values[-2]), int(values[-1])


def get_num_patches(self, grid: Union[torch.Tensor, List[int]], modality: Modality) -> int:
    """计算原始 patch 数量（用于 pixel_values 切片）。"""
    if modality == Modality.AUDIO:
        return int(grid.item())
    if self.model_type == "kimi_vl" and modality == Modality.IMAGE:
        h, w = self._kimi_hw_from_patch_grid(grid)
        return h * w
    return int(grid[0] * grid[1] * grid[2])


def _kimi_tokens_from_patch_grid(self, grid: Union[torch.Tensor, List[int]]) -> int:
    """MoonViT + tpool：输出长度为 (h//mh)*(w//mw)；时间维度已合并。"""
    h, w = self._kimi_hw_from_patch_grid(grid)
    merge_h, merge_w = self.model_config.hf_config.vision_config.merge_kernel_size
    return (h * w) // (merge_h * merge_w)
```

test/registered/unit/disaggregation/test_encode_server.py

新增单元测试覆盖 Kimi-VL 2D 网格的全部行为，是保证回归的重要配套。

```
import pickle
import unittest
from types import SimpleNamespace
import numpy as np
import torch
from sglang.srt.disaggregation.encode_receiver import EmbeddingData
from sglang.srt.disaggregation.encode_server import MMEncoder, _get_mm_grid_dim
from sglang.srt.managers.schedule_batch import Modality
from sglang.srt.utils.common import safe_pickle_loads
from sglang.test.ci.ci_register import register_cpu_ci

register_cpu_ci(est_time=2, suite="base-a-test-cpu")


class TestKimiVLEPDGrid(unittest.TestCase):
    @staticmethod
    def _make_encoder(model_type="kimi_vl"):
        encoder = MMEncoder.__new__(MMEncoder)
        encoder.model_type = model_type
        encoder.model_config = SimpleNamespace(
            hf_config=SimpleNamespace(
                vision_config=SimpleNamespace(merge_kernel_size=(2, 2))
```

```
            )
        )
        return encoder

    def test_kimi_vl_prefers_and_normalizes_hw_grid(self):
        # 验证 Kimi-VL 优先选择 image_grid_hws 并转换为 Tensor
        mm_inputs = {
            "image_grid_hws": np.array([[40, 60]], dtype=np.int64),
            "image_grid_thw": torch.tensor([[1, 20, 30]]),
            "grid_thws": torch.tensor([[1, 10, 15]]),
        }
        grid = _get_mm_grid_dim(mm_inputs, Modality.IMAGE, "kimi_vl")
        self.assertIsInstance(grid, torch.Tensor)
        torch.testing.assert_close(grid, torch.tensor([[40, 60]]))

    def test_kimi_vl_2d_grid_counting_and_slicing(self):
        # 验证 2D 网格下的 patch 计数、token 计数和 embedding 切片
        encoder = self._make_encoder()
        grids = torch.tensor([[40, 60], [20, 40]])
        embedding = torch.arange(800 * 2).reshape(800, 2)
        self.assertEqual(encoder.get_num_patches(grids[0], Modality.IMAGE), 2400)
        self.assertEqual(encoder.get_num_tokens(grids[0], Modality.IMAGE), 600)
        slices = encoder.slice_embedding(embedding, grids, Modality.IMAGE)
        self.assertEqual([item.shape for item in slices], [(600, 2), (200, 2)])

    def test_grid_metadata_is_safe_to_deserialize(self):
        # 验证包含 NumPy 网格元数据的 EmbeddingData 可以安全 Pickle 往返
        grid = _get_mm_grid_dim(
            {"image_grid_hws": np.array([[40, 60]], dtype=np.int64)},
            Modality.IMAGE, "kimi_vl",
        )
        embedding_data = EmbeddingData(
            req_id="test-request", num_parts=1, part_idx=0,
            grid_dim=grid, modality=Modality.IMAGE,
            embedding=torch.zeros((600, 4)),
        )
        restored = safe_pickle_loads(
            pickle.dumps(embedding_data.copy_without_embedding())
        )
        torch.testing.assert_close(restored.grid_dim, torch.tensor([[40, 60]]))
```

评论区精华

在代码审查中，liusy58 提出疑问是否需要保留新增的单元测试文件（"This file is just for unit testing. Should we delete it?"）。xiaojun-zhang 回应：端到端测试（E2E）覆盖完整路径，单元测试直接验证 2D 网格逻辑并保护 Kimi-K2.5 的 3D 行为，建议保留。最终 reviewer ShangmingCai 批准，同时提醒关注新增测试的耗时，未来可能考虑跳过或移入额外 suite。

- 是否删除单元测试文件 (question): 单元测试文件保留，reviewer 同意并批准 PR。

风险与影响

- 风险：
  - 模型兼容性：所有 VLM 的网格属性查找逻辑均经过修改，但仅对 Kimi-VL 和 Kimi-K2.5 增加分支，其他模型仍沿用通用顺序，风险较低。
  - 序列化安全：通过 _convert 将 NumPy 转为 Torch Tensor，确保 Payload 可安全 Pickle 反序列化，但需确认 _convert 能处理所有边缘情况（如 object 类型）。
  - 性能：新增 _kimi_hw_from_patch_grid 在热点路径 get_num_patches 和 get_num_tokens 中调用，涉及多次类型检查和展平，但数据量小，影响可忽略。
  - 测试覆盖：单元测试覆盖 2D/3D 网格多种组合，端到端测试启动真实服务，风险可控。
- 影响：
  - 对用户：启用 Kimi-VL 模型在 EPD 模式下的多图片推理，此前该功能不可用；对已有 Kimi-K2.5 用户无影响。
  - 对系统：无性能退化或稳定性影响，CI 时间增加约 200s（est_time 从 97 增至 300），但后端可调整（如跳过或移入 extra suite）。
  - 对团队：新增约 300 行测试代码，维护成本增加，但降低了 Kimi-VL 回归风险。
  - 风险标记：模型特定修复，序列化兼容性，核心路径变更

关联脉络

- 暂无明显关联 PR