PR #44645 完整报告

vllm-project/vllm

[Bugfix] Stream Llama4 weight loading to avoid host-OOM with copy-returning loaders

合并时间：2026-06-15 10:23

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44645

执行摘要

- 一句话：流式加载 Llama4 权重修复 host OOM
- 推荐动作：建议精读此 PR，尤其是处理大型模型权重加载时的内存管理。其中的迭代器流式消费模式值得借鉴。对于部署 Llama4 的团队，应优先合并此修复。

功能与动机

加载 Llama-4-Scout/Maverick 权重时，若使用 --load-format runai_streamer 等返回私有拷贝的加载器，由于权重迭代器被完全物化为列表，每 TP worker transient 持有整个语言模型检查点，多 GPU 节点上集体超限导致主机 OOM（issue #44430）。根因是 llama4.py 中列表推导式将整个迭代器转化为列表，mllama4.py 中 _separate_and_rename_weights 等方法也完全缓冲所有权重。

实现拆解

实现分为两大修改：

1. llama4.py：在 Llama4ForCausalLM.load_weights 中，将处理 Q/K 旋转的列表推导式改为生成器表达式，使 AutoWeightsLoader 惰性消费，避免一次性持有所有权重。
2. mllama4.py：重写 Llama4ForConditionalGeneration.load_weights，移除 _separate_and_rename_weights 和 _handle_expert_scale_broadcasting 辅助方法，新增内嵌生成器 regular_language_model_weights，它产出语言模型权重用于流式加载，同时将小量 vision/projector 权重和 expert scale 权重缓冲到列表中。随后分别用 AutoWeightsLoader 流式加载语言模型权重，然后用 _load_other_weights 加载非语言模型权重，最后处理 expert scale 广播。
3. 清理：删除两个未使用的辅助方法。

测试验证：在 8×H200 节点上 TP=8，原 runai_streamer 加载峰值 anon≈1019 GiB 导致 OOM；修复后峰值绑定在约 44 GiB/worker，无 OOM，且输出与默认 loader byte-identical。

关键文件：

- vllm/model_executor/models/llama4.py（模块 模型加载；类别 source；类型 data-contract；符号 Llama4ForCausalLM.load_weights）：核心修复之一：将列表推导式改为生成器表达式，实现惰性消费，是主机 OOM 的关键修复。

- vllm/model_executor/models/mllama4.py（模块 模型加载；类别 source；类型 refactor；符号 _separate_and_rename_weights, _handle_expert_scale_broadcasting, regular_language_model_weights）：第二大修改：重构加载逻辑，移除两个辅助方法，引入流式生成器，仅缓冲小量非语言模型权重。

关键符号：Llama4ForCausalLM.load_weights, Llama4ForConditionalGeneration.load_weights, regular_language_model_weights

关键源码片段

vllm/model_executor/models/llama4.py

核心修复之一：将列表推导式改为生成器表达式，实现惰性消费，是主机 OOM 的关键修复。

```
def load_weights(self, weights: Iterable[tuple[str, torch.Tensor]]) -> set[str]:
    loader = AutoWeightsLoader(
        self,
        skip_prefixes=(['lm_head.'] if self.config.tie_word_embeddings else None),
    )
    # 使用生成器（而非列表推导式）使 AutoWeightsLoader 惰性消费权重迭代器
    # 物化整个检查点会立刻导致 host OOM
    weights = (
        self.permute_qk_weight_for_rotary(name, loaded_weight)
        for name, loaded_weight in weights
    )
    return loader.load_weights(weights)
```

vllm/model_executor/models/mllama4.py

第二大修改：重构加载逻辑，移除两个辅助方法，引入流式生成器，仅缓冲小量非语言模型权重。

```
def load_weights(self, weights: Iterable[tuple[str, torch.Tensor]]) -> set[str]:
    other_weights: list[tuple[str, torch.Tensor]] = []
    expert_scale_weights: list[tuple[str, torch.Tensor]] = []

    def regular_language_model_weights() -> Iterable[tuple[str, torch.Tensor]]:
        '''产出语言模型权重（流式），缓存其余部分。'''
        for name, weight in weights:
            renamed = self._rename_weight_for_modelopt_checkpoint(name)
            attr = renamed.split('.', 1)[0]
            if isinstance(getattr(self, attr), StageMissingLayer):
                continue
            if renamed.startswith('language_model.'):
                yield renamed, weight
            else:
                other_weights.append((renamed, weight))

    # 流式加载语言模型权重
    updated_params.update(loader.load_weights(regular_language_model_weights()))
    # 加载非语言模型权重（vision / projector）
```

```
    updated_params.update(self._load_other_weights(other_weights, params_dict, stacked_
    params_mapping))
    # 加载 expert scale 权重（为简化片段省略广播细节）
    for name, loaded_weight in expert_scale_weights:
        param = params_dict[name]
        weight_loader = getattr(param, 'weight_loader', default_weight_loader)
        weight_loader(param, loaded_weight)
        updated_params.add(name)
    return updated_params
```

评论区精华

无实质性讨论。作者在 PR 描述中说明了根因和修复方案，维护者 DarkLight1337 直接批准。唯一人类评论为作者向维护者请求 review。

- 请求审阅与类比说明 (question): 维护者 DarkLight1337 批准合并。

风险与影响

- 风险：主要风险是迭代器消费模式从列表变为生成器，如果 AutoWeightsLoader 上游代码未来改为多次遍历 weights 参数，则生成器只能迭代一次，可能导致部分权重未加载。当前 AutoWeightsLoader 实现为单次消费，但需保持兼容。此外，mllama4 中重构后，expert scale 广播逻辑从单独方法移入生成器 + 后续循环，提高了耦合度但简化了控制流。没有新增单元测试，依赖现有测试覆盖。
- 影响：对用户：修复使用 runai_streamer 等私有拷贝加载器时的高负载节点 OOM 问题，使其能在多 GPU 节点上稳定加载 Llama4 模型。对系统：降低每 worker 主机内存占用从整个检查点（约 203 GiB）到缓冲大小约 44 GiB。对团队：删除冗余代码，降低维护成本。
- 风险标记：迭代器消费模式变更，生成器单次迭代假设，无新增测试覆盖

关联脉络

- 暂无明显关联 PR