

PR #50890 完整报告

vllm-project/vllm

[BugFix][Pooling] Skip weight-prefix probe when model has WeightsMapper

合并时间: 2026-08-06 12:12

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/50890>

执行摘要

- 一句话: Pooling 探测用 Mapper 提前退出, 避免整包克隆
- 推荐动作: 值得精读。PR 体量小但展示了三个值得学习的决策: 为何不能整体删除前缀探测、为何不能对整份权重流直接 `apply()` mapper (双重映射 + `clone()` 不保留 `shard_id` 元数据)、以及如何在保持 #39650 `buffer-reuse` 保护的同时用模型自带 mapper 让探测提前退出。对理解 vLLM pooling 适配器与 `WeightsMapper` 的加载契约非常有帮助。关注点: `_map_name` 私有 API 的使用, 以及测试中 `Tensor.clone` 计数的验证手法。

功能与动机

`ModelForPooling.load_weights` 的通用前缀探测 (`candidate_prefixes` 为 `['', 'model.']`) 对 key 仅能在模型自带 `hf_to_vllm_mapper` 映射后对齐的 checkpoint (如 Gemma4 的 `model.language_model.* -> language_model.model.*`) 永不命中。PR body 指出探测 “walks the entire iterator, cloning every tensor”, 在 `runner: pooling / convert: embed` 下造成 “~44.3 GiB of extra host clones before engine init failed with CPU OOM”。由于 #39650 要求对 `buffer-reusing` 迭代器做 `clone` 保护, 这些 `clone` 会累积为整份 checkpoint 的主机副本; 又不能直接删掉探测 (无 mapper 的 `*ForCausalLM / *Model` 布局仍需要“空前缀 / `model.` 前缀”兼容), 也不能丢弃 `clone` 后重开迭代器 (单遍迭代器无法从 `adapters.py` 重开文件)。

实现拆解

1. 变更入口: `vllm/model_executor/models/adapters.py` 中 `ModelForPooling.load_weights` (由 `_create_pooling_model_cls` 动态构造的 pooling 模型类) 是唯一源码改动点。
2. 探测阶段集成 mapper: 在 `load_weights` 开头通过 `getattr(self, 'hf_to_vllm_mapper', None)` 读取模型自带映射; 探测循环里对每个 `(name, loaded_weight)` 仍先执行 `clone` 进 `seen_weights` (保留对 `buffer-reusing` 迭代器的保护), 随后若存在 mapper 则调用 `_map_name(name)` 得到 `lookup_name`, `mapped is None` 时 `continue` 跳过成员检查; 用 `prefix + lookup_name` 在 `params_dict` 完成 `['', 'model.']}` 命中判定, 命中即 `break`。
3. 转发契约不变: 命中后仍以原始 (未映射、未加前缀) 名字构造 `itertools.chain(seen_weights, weights)` 的 lazy 流交给父 loader, 避免父 loader 再次应用 mapper 导致双重映射, 也避免 `apply()` 给 packed weights 附加的 `shard_id` 元数据在 `Tensor.clone()` 中丢失。

4. 测试配套: `tests/models/test_adapters.py` 新增 `ModelWithWeightsMapper` (用 `WeightsMapper(orig_to_new_prefix={'model.language_model.': 'language_model.model.'})` 模拟 Gemma4 式 key 布局) 与 `test_pooling_prefix_probe_uses_hf_to_vllm_mapper`: 通过 monkeypatch 计数 `torch.Tensor.clone`, 断言 `clone_count == 1` 且小于 checkpoint 张量数; 同时校验父 loader 收到原始名字、映射后正确加载。原有两个 `buffer-reuse / clone` 保护测试保持不变并继续通过。
5. 验证与回归: PR 作者在 Gemma4-E2B 上 A/B: 新路径 0 次 clone / 0.7 s / 0 GiB, 旧路径 2011 次 clone / 9.543 GiB / 24.8 s; 端到端 `runner=pooling / convert=embed` 成功, 强制旧探测则 memcg OOM-kill。CI 中唯一失败 (HunYuanMoE `num_experts_per_token` 校验) 与本 PR 无关, 由 #51179 修复。

关键文件:

- `vllm/model_executor/models/adapters.py` (模块 适配器; 类别 source; 类型 data-contract; 符号 `load_weights`): PR 核心逻辑: 在 `ModelForPooling.load_weights` 的前缀探测中引入 `hf_to_vllm_mapper` 映射后的名字做成员检查, 命中即提前退出, 避免整包权重被 clone; 原始名字仍转发给父 loader, 防止双重映射。
- `tests/models/test_adapters.py` (模块 适配器; 类别 test; 类型 test-coverage; 符号 `_LanguageModelInner`, `_LanguageModelWrapper`, `ModelWithWeightsMapper`, `test_pooling_prefix_probe_uses_hf_to_vllm_mapper`): 新增 `ModelWithWeightsMapper` 桩与 clone 计数测试, 锁定“带 mapper 的模型探测只 clone 一次、原始名字转发”的行为契约, 防止回归。

关键符号: `load_weights`, `test_pooling_prefix_probe_uses_hf_to_vllm_mapper`, `counting_clone`, `ModelWithWeightsMapper.load_weights`

关键源码片段

`vllm/model_executor/models/adapters.py`

PR 核心逻辑: 在 `ModelForPooling.load_weights` 的前缀探测中引入 `hf_to_vllm_mapper` 映射后的名字做成员检查, 命中即提前退出, 避免整包权重被 clone; 原始名字仍转发给父 loader, 防止双重映射。

```
def load_weights(self, weights: Iterable[tuple[str, torch.Tensor]]):
    params_dict = dict(self.named_parameters())
    # 模型自带 key 重映射 (如 Gemma4 的 model.language_model.*
    # -> language_model.model.*) 时, 探测阶段用映射后的名字做成员检查,
    # 让 '' / 'model.' 扫描能提前退出, 避免整份 checkpoint 被 clone。
    hf_to_vllm_mapper = getattr(self, 'hf_to_vllm_mapper', None)

    # 兼容 *ForCausalLM 与 *Model 两种 checkpoint 布局
    candidate_prefixes = ['', 'model.']
    target_prefix = ''

    seen_weights = list[tuple[str, torch.Tensor]]()
    for name, loaded_weight in weights:
```

```

# 迭代器可能复用 tensor 缓冲区（如 runai_streamer），
# 必须先 clone，后续 lazy chain 才能安全读取（见 #39650）。
seen_weights.append((name, loaded_weight.clone()))

# 只把映射用于探测匹配；转发给父 loader 的仍是原始名字，
# 避免父 loader 再应用一次 mapper 造成双重映射。
lookup_name = name
if hf_to_vllm_mapper is not None:
    mapped = hf_to_vllm_mapper._map_name(name)
    if mapped is None:
        continue # 该 key 不在模型参数中，继续扫描
    lookup_name = mapped

try:
    target_prefix = next(
        prefix
        for prefix in candidate_prefixes
        if prefix + lookup_name in params_dict
    )
    break # 第一个命中即退出，剩余权重不再进入 seen_weights
except StopIteration:
    pass # 不存在的权重交给 AutoWeightsLoader 处理

# 命中后按 target_prefix 定位目标子模块，
# 最终以 (target_prefix + name, weight) 的 lazy chain 形式
# 把 seen_weights 与剩余 weights 一并转发给父 loader。

```

tests/models/test_adapters.py

新增 `ModelWithWeightsMapper` 桩与 clone 计数测试，锁定“带 mapper 的模型探测只 clone 一次、原始名字转发”的行为契约，防止回归。

```

def test_pooling_prefix_probe_uses_hf_to_vllm_mapper(monkeypatch):
    """Mapper 映射后的名字能让前缀探测在第一个 key 上命中并提前退出。"""
    ref = {
        'model.language_model.embed.weight': torch.randn(8, 4),
        'model.language_model.layer0.weight': torch.randn(8, 8),
        'model.language_model.extra0.weight': torch.randn(8, 8),
        'model.language_model.extra1.weight': torch.randn(8, 8),
        'model.language_model.extra2.weight': torch.randn(8, 8),
    }

    # 先确认通用的 '' / 'model.' 探测对这批 key 必然不命中
    model = _make_pooling_model(ModelWithWeightsMapper)
    params = dict(model.named_parameters())
    for name in ref:
        assert name not in params
        assert f'model.{name}' not in params

    # 用计数代理观察探测阶段实际 clone 的次数

```

```

clone_count = 0
original_clone = torch.Tensor.clone

def counting_clone(self, *args, **kwargs):
    nonlocal clone_count
    clone_count += 1
    return original_clone(self, *args, **kwargs)

monkeypatch.setattr(torch.Tensor, 'clone', counting_clone)

loaded = model.load_weights(iter(ref.items()))

# 第一个 key 经 mapper 映射后命中参数，探测只 clone 一次就退出
assert clone_count == 1
assert clone_count < len(ref)
# 父 loader 收到的仍是原始（未映射、未加前缀）的 checkpoint 名字
assert model.seen_names == list(ref.keys())
assert loaded == {
    'language_model.model.embed.weight',
    'language_model.model.layer0.weight',
}

```

评论区精华

核心讨论围绕“跳过探测”与“在探测中集成 mapper”两种方案展开：DarkLight1337 首先建议直接在遍历循环中应用 `hf_to_vllm_mapper` 得到正确前缀，作者据此把初版方案从“整体跳过探测”改为“探测中用映射名做成员检查”，保留无 mapper 模型的兼容路径。关于直接调用 `hf_to_vllm_mapper.apply(weights)`，作者解释对整个权重迭代器应用不安全：父 loader 已应用过一次 mapper，会造成双重映射；且 `apply()` 可能给 packed weights 附加 `shard_id`，而 `Tensor.clone()` 不保留该元数据，DarkLight1337 接受解释。另外测试中的随机种子被评审质疑无必要，作者确认移除。

- 是否直接对整份权重流 `apply hf_to_vllm_mapper (design)`：采用仅映射探测用的 lookup 名、转发原始权重流的方案，DarkLight1337 接受解释。
- 跳过探测 vs 探测中集成 mapper (design)：保留探测路径并利用 mapper 提前退出，避免回归无 mapper 模型的 `['', 'model.']` 前缀兼容。
- 测试中的随机种子必要性 (testing)：移除 seed 设置（最终代码已无 seed）。
- CI 失败是否与本 PR 相关 (question)：确认无关问题，由 #51179 另行修复。

风险与影响

- 风险：
 1. 前置假设：新逻辑假设带 `hf_to_vllm_mapper` 的 pooling 模型其父 `load_weights` 会自行应用 mapper；若未来某模型声明了 mapper 但父 loader 不应用，转发原始名字会造成加载失败（当前带 mapper 的模型均符合该契约）。

2. 私有 API 依赖：直接调用 `WeightsMapper._map_name`（私有方法），`WeightsMapper` 内部结构调整时会同步影响此路径，建议后续改为公开封装或补充测试锁定。
3. 部分映射场景收益有限：若 checkpoint 开头若干 key 经 `_map_name` 返回 `None`，探测会继续扫描并持续 clone，仍可能出现整包克隆；对正常模型（首个 key 即映射）可提前退出。
4. 回归风险：改动集中在 pooling 加载路径的探测阶段，无 mapper 模型走原路径；`itertools.chain` 与 `clone` 语义未变，风险可控。新测试覆盖 mapper 命中场景，未覆盖 `mapped is None` 连续扫描分支。
 - 影响：使用 pooling / embed 且模型自带 key 重映射（如 Gemma4）的加载路径内存占用显著下降：实测消除约 9.5 GiB 主机副本（生产场景约 44.3 GiB），加载 wall time 从 24.8 s 降到 0.7 s，并避免 CPU OOM 导致的引擎初始化失败。源代码改动 14 行，附一个确定性单元测试，维护成本低；但对 `WeightsMapper` 内部结构新增了一处依赖点，后续 mapper 演进需要同步关注。
 - 风险标记：核心加载路径变更，依赖私有 API `_map_name`，部分映射场景收益有限

关联脉络

- PR #39650（上下文未提供标题，PR body 引用为 #39650）：PR body 明确引用 #39650：buffer-reusing 迭代器（`runai_streamer`）要求探测阶段对每个被扫描 tensor 做 clone 并用 `itertools.chain` 保证安全。本 PR 是它的互补修复：当探测永不命中时，这些 clone 会累积成整份 checkpoint 的主机副本并触发 CPU OOM。
- PR #51179 [CI Bug] Fix `pydantic_core._pydantic_core.ValidationError: Input should be a valid integer`：本 PR 的 CI 失败记录直接指向该问题：作者确认 `basic-models-tests-extra-initialization-2` 中 HunYuanMoE 的 `num_experts_per_token` 校验失败与本 PR 无关，由 #51179 修复。