

PR #51665 完整报告

vllm-project/vllm

Fix weight tying

合并时间: 2026-08-20 18:31

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51665>

执行摘要

- 一句话: 修复 weight tying 忽略 checkpoint 导致 lm_head 被静默丢弃
- 推荐动作: 值得精读。PR 将“config 声明”与“checkpoint 实际内容”的对账集中到新模块, 用“先解绑加载、再按相等性重绑”的策略同时兼容冗余副本与真实独立 head, 是 vLLM 权重加载架构上的一次重要收敛。建议重点阅读 weight_tying.py 的类型探测逻辑、AutoWeightsLoader 的别名跳过与缺 canonical 抛错设计, 以及 RL 场景下重绑定必要性的讨论。

功能与动机

Issue #51063 报告: 复合 VLM 包装器 `Mistral3ForConditionalGeneration` 从错误的顶层 config 解析 `tie_word_embeddings`, 导致 checkpoint 中真实的 `lm_head.weight` 被静默丢弃, 输出“词表一致但语义混乱”的结果。PR body 明确指出根因: vLLM 只信任 config 而不检查 checkpoint, 'a config claiming tied weights against a checkpoint with a real `lm_head` silently discards it and generates gibberish'。同时反向场景 (config 未绑定但 checkpoint 无 `lm_head`) 会遗留未初始化的 `lm_head`。

实现拆解

1. 配置层对账入口: 在 `vllm/config/model.py` 新增 `maybe_untie_word_embeddings()`, 当 `hf_config.tie_word_embeddings=True` 且 `checkpoint_has_lm_head()` 返回 `True` 时, 将顶层与文本层 config 的 `tie_word_embeddings` 同时置为 `False`, 并记录 `word_embeddings_untied_by_checkpoint=True` 供加载后对账使用。
2. checkpoint 探测: 在 `vllm/transformers_utils/config.py` 新增带 `@cache` 的 `checkpoint_has_lm_head()`, 读取 `safetensors` 参数元数据, 判断是否存在以 `lm_head.weight` 结尾的张量; `repo_utils.py` 同步调整 `try_get_safetensors_metadata`, 将 `NotASafetensorsRepoError` 设为致命错误不再重试, 避免对非 `safetensors` 仓库做无谓的重试。
3. 加载后重绑定: 新增 `vllm/model_executor/model_loader/weight_tying.py`, `_get_untied_lm_head` 按类型而非名字定位唯一的 `ParallelLMHead` 与 `VocabParallelEmbedding` (多模态模型的 head 嵌套在语言模型内部, 且量化 head 被 `UnquantizedEmbeddingMethod` 过滤排除); `maybe_retie_word_embeddings` 在加载后比较 CPU 上的权重 (规避 ROCm sleep mode 下设备上 `torch.equal` 段错误), 相同则重新共享存储回收显存, 不同则告警并保留独立 head。

4. 加载器别名跳过：在 `vllm/model_executor/models/utils.py` 的 `AutoWeightsLoader` 构造时用 `_get_tied_embedding_params` 建立参数 id 到首个 qualname 的映射，`_filter_skipped` 在遍历权重时跳过别名，`_check_skipped_aliases` 在加载结束后校验被跳过的别名对应的 canonical 参数确已加载，否则抛 `ValueError` 防止共享参数未初始化；对 MoE 路由器这类非 embedding 共享参数保持不跳过，由 `ModuleWithSharedParam` 测试保证。
5. 模型配套清理：约 50 个模型文件删除手写 `lm_head` 跳过逻辑；`gemma`、`gemma2`、`commandr`、`cohere2_moe`、`mpt` 五个硬编码绑定模型去掉 `assert config.tie_word_embeddings`，未绑定 `config` 时继续直接使用 `embeddings`（这些模型没有真实 `lm_head`，无法满足 checkpoint 中携带的 `head`，留待后续跟进）。
6. 测试配套：新增 `tests/model_executor/model_loader/test_weight_tying.py`，覆盖相同 / 不同权重、量化 `head` 豁免、无 checkpoint 覆盖三种场景；`tests/models/test_utils.py` 新增覆盖别名权重只加载一次、缺 canonical 时抛错、非 embedding 共享参数仍加载三类行为。

关键文件：

- `vllm/model_executor/model_loader/weight_tying.py`（模块 权重绑定；类别 source；类型 core-logic；符号 `_UntiedLMHead`, `_get_untied_lm_head`, `maybe_retie_word_embeddings`）：新增核心对账模块：定位可绑定的 `lm_head` 与输入嵌入，加载后按相等性决定重新绑定或告警保留，是整个修复的核心。
- `vllm/model_executor/models/utils.py`（模块 加载器；类别 source；类型 core-logic；符号 `_get_tied_embedding_params`, `_filter_skipped`, `_check_skipped_aliases`）：`AutoWeightsLoader` 的别名跳过机制，替代约 50 处手写 `lm_head` 跳过逻辑并补齐约 20 个缺失场景，是加载路径的核心改造。
- `vllm/config/model.py`（模块 模型配置；类别 source；类型 data-contract；符号 `maybe_untie_word_embeddings`, `word_embeddings_untied_by_checkpoint`）：配置层新增 `maybe_untie_word_embeddings`，用 checkpoint 元数据推翻 `config` 声明，是整个对账链路的入口。
- `vllm/transformers_utils/config.py`（模块 配置工具；类别 source；类型 core-logic；符号 `checkpoint_has_lm_head`）：新增 `checkpoint_has_lm_head` 探测函数，从 `safetensors` 元数据判断 checkpoint 是否自带 `lm_head`。
- `tests/model_executor/model_loader/test_weight_tying.py`（模块 权重绑定；类别 test；类型 test-coverage；符号 `UntiedModel`, `test_retie_only_when_identical`, `test_quantized_lm_head_is_left_alone`, `test_no_retie_without_checkpoint_override`）：新增核心测试：覆盖重绑定的相等 / 不等分支、量化 `head` 豁免、无 checkpoint 覆盖时保持解绑。
- `tests/models/test_utils.py`（模块 加载器；类别 test；类型 test-coverage；符号 `ModuleWithTiedWeights`, `test_module_skip_tied_weights`, `test_module_skip_tied_weights_without_canonical`, `ModuleWithSharedParam`）：补充 `AutoWeightsLoader` 别名跳过行为的测试，包括缺 canonical 抛错与非 embedding 共享参数不受影响。

关键符号: maybe_untie_word_embeddings, checkpoint_has_lm_head, maybe_retie_word_embeddings, _get_untied_lm_head, _UntiedLMHead.tie, _get_tied_embedding_params, _filter_skipped, _check_skipped_aliases, AutoWeightsLoader.load_weights

关键源码片段

vllm/model_executor/model_loader/weight_tying.py

新增核心对账模块: 定位可绑定的 lm_head 与输入嵌入, 加载后按相等性决定重新绑定或告警保留, 是整个修复的核心。

```
"""把 word embedding 绑定与 checkpoint 实际内容对齐。"""
```

```
from dataclasses import dataclass
```

```
import torch
```

```
from torch import nn
```

```
from vllm.model_executor.layers.vocab_parallel_embedding import (
    ParallelLMHead,
    UnquantizedEmbeddingMethod,
    VocabParallelEmbedding,
)
```

```
@dataclass
```

```
class _UntiedLMHead:
```

```
    name: str
```

```
    lm_head: ParallelLMHead
```

```
    embed_tokens: VocabParallelEmbedding
```

```
    @property
```

```
    def weight_name(self) -> str:
```

```
        return f"{self.name}.weight"
```

```
    def tie(self, model: nn.Module) -> None:
```

```
        # 按名字路径找到父模块, 把 lm_head 替换为共享 embed_tokens 存储的版本
```

```
        parent_name, _, attr = self.name.rpartition(".")
```

```
        parent = model.get_submodule(parent_name) if parent_name else model
```

```
        setattr(parent, attr, self.lm_head.tie_weights(self.embed_tokens))
```

```
def _get_untied_lm_head(model: nn.Module) -> _UntiedLMHead | None:
```

```
    # 按类型而不是名字查找: 多模态模型的 lm_head 嵌套在语言模型内部,
```

```
    # 且 get_input_embeddings 在不同多模态模型上参数不统一
```

```
    heads = []
```

```
    embeddings = []
```

```
    for name, module in model.named_modules():
```

```

if not isinstance(module, VocabParallelEmbedding):
    continue
# 量化 head 的权重可能打包在别的名字下，且在线量化在加载后才创建，
# 无法在这里判断内容与加载状态，因此跳过
if not isinstance(module.quant_method, UnquantizedEmbeddingMethod):
    continue
if isinstance(module, ParallelLMHead):
    heads.append((name, module))
else:
    embeddings.append(module)

# 有多个 head（如 MTP）或已绑定的模型保持原样
if len(heads) != 1 or len(embeddings) != 1:
    return None

(name, lm_head), embed_tokens = heads[0], embeddings[0]
if lm_head.weight.shape != embed_tokens.weight.shape:
    return None
return _UntiedLMHead(name, lm_head, embed_tokens)

```

```

def maybe_retie_word_embeddings(model: nn.Module, model_config: ModelConfig) -> None:
    # 只有 config 被 checkpoint 元数据推翻为解绑时才需要后续对账
    if not model_config.word_embeddings_untied_by_checkpoint:
        return
    if (untied := _get_untied_lm_head(model)) is None:
        return

    # 在 CPU 上比较：ROCm 开启 sleep mode 时设备上的 torch.equal 会段错误
    if not torch.equal(untied.lm_head.weight.cpu(), untied.embed_tokens.weight.cpu()):
        logger.warning(
            "The config for %s says the word embeddings are tied, but the "
            "checkpoint contains a different %s, which has been used instead "
            "of tying. Set tie_word_embeddings=False to silence this warning.",
            model_config.model,
            untied.weight_name,
        )
        return

    # 权重完全相同说明 checkpoint 只是保留了冗余副本，重新共享存储以回收显存
    logger.debug("Re-tying %s, which is identical to the input embeddings", untied.name)
    untied.tie(model)

```

vllm/model_executor/models/utils.py

AutoWeightsLoader 的别名跳过机制，替代约 50 处手写 lm_head 跳过逻辑并补齐约 20 个缺失场景，是加载路径的核心改造。

```

def _get_tied_embedding_params(module: nn.Module) -> dict[str, str]:
    """把每个被绑定的 word embedding 别名映射到它第一个出现的名字。"""

```

```

canonical = {}
aliased = {}
for prefix, submodule in module.named_modules(remove_duplicate=False):
    if not isinstance(submodule, VocabParallelEmbedding):
        continue
    for name, param in submodule.named_parameters(remove_duplicate=False):
        qualname = f"{prefix}.{name}" if prefix else name
        # setdefault 返回第一个登记该参数 id 的名字，后续同名参数记为别名
        if (first_name := canonical.setdefault(id(param), qualname)) != qualname:
            aliased[qualname] = first_name
return aliased

```

```

class AutoWeightsLoader:

```

```

    """自动加载器：遍历 checkpoint 权重并按模块名映射加载。"""

```

```

    def __init__(self, module, *, skip_prefixes=None, skip_substrs=None, ...):

```

```

        ...

```

```

        # weight tying 会让两个 qualname 指向同一个 nn.Parameter
        # （例如 lm_head.weight 与 model.embed_tokens.weight）。
        # 两次加载会重复写同一块存储，因此只加载遍历中先遇到的哪个名字
        self.aliased_params = _get_tied_embedding_params(module)

```

```

    def _filter_skipped(self, weights):

```

```

        for name, weight in weights:
            if name in self.aliased_params:
                # 记录被跳过的别名与它对应的 canonical 名，加载结束后校验
                self._skipped_aliases[name] = self.aliased_params[name]
                continue
            if any(name.startswith(p) for p in self.skip_prefixes) or any(
                substr in name for substr in self.skip_substrs
            ):
                continue
            yield name, weight

```

```

    def _check_skipped_aliases(self, autoloaded_weights):

```

```

        # 若 canonical 参数没有被加载（例如 checkpoint 只给了 lm_head.weight），
        # 直接跳过会让共享参数保持未初始化，必须报错而不是静默
        for name, canonical_name in self._skipped_aliases.items():
            if canonical_name not in autoloaded_weights:
                raise ValueError(
                    f"Weight {canonical_name} was skipped for {name} but never "
                    f"loaded. This usually means the checkpoint is missing the "
                    f"weight."
                )

```

评论区精华

Isotr0py (default_loader.py) : 如果 config 未绑定但 checkpoint 没有 lm_head.weight , 应该 raise 而不是尝试自动绑定, 因为 checkpoint 可能被错误导出。hmellor 回应: 最初认为这是通用场景, 最终同意“In hindsight, yes this is likely a case where we should not attempt to auto fix. I'll revert this part”。

Isotr0py (config.py) : 不能总是假设 head 名为 lm_head, 自定义模型可能有非标准 head 名。hmellor 回应: 该探测是 failsafe 的最佳努力, 失败只会让 maybe_untie_word_embeddings 提前返回, 不会破坏正确路径。

Isotr0py (weight_tying.py) : 重绑定后输出等价, 复杂性是否值得? hmellor 回应: 不相等时的 warning 有诊断价值; 若 config 声称绑定而 checkpoint 含两份相同权重, 不重绑时 RL 更新 embedding 不会传播到 lm_head。aoshen02 确认: “Yes weight will get updated in RL.”

Isotr0py 建议 maybe_retie_word_embeddings 应移入 process_weights_after_loading, 避免散落在 load_weights 路径中, 作者按建议调整了调用位置。

- config 未绑定但 checkpoint 缺 lm_head 时是否自动绑定 (design): 作者 revert 了 default_loader 中的 maybe_tie_word_embeddings 自动绑定逻辑。
- checkpoint_has_lm_head 对自定义 head 名字的假设 (question): 保留 lm_head.weight 后缀探测, 作为尽力而为的修复手段。
- 加载后重绑定的价值与复杂度 (design): 保留重绑定逻辑, 同时保留不相等时的告警。
- 依赖 .weight 对量化场景的健壮性 (correctness): 量化 head 直接豁免, 保持简单可维护。
- maybe_retie_word_embeddings 的调用位置 (design): 作者按建议调整了调用位置。
- ROCm 上 torch.equal 段错误 (correctness): 比较前将权重转 .cpu()。

风险与影响

- 风险:
 - 核心加载路径变更: AutoWeightsLoader 供所有模型权重加载使用, 别名映射若漏判会把 checkpoint 权重静默跳过; 已通过 _check_skipped_aliases 抛错兜底, 但覆盖完整性依赖 named_modules(remove_duplicate=False) 的遍历顺序与参数别名判定。
 - 启发式依赖: checkpoint_has_lm_head 只查 lm_head.weight 后缀且仅支持 safetensors; GGUF、bin 等格式返回 None 走旧逻辑, 自定义 head 名场景 failsafe 跳过解绑, 行为不变但问题依旧。
 - 全量比较开销: 加载后 torch.equal 对 lm_head 与 embed_tokens 做 CPU 全量比较, 大词表模型存在一次性拷贝与比较开销; ROCm 段错误已通过 .cpu() 规避, 但拷贝本身仍耗时。
 - 大面回归面: 70 个文件、约 50 处手写跳过逻辑被删除, 若有模型采用非标准绑定方式可能回归; gemma 等 5 个模型去掉 assert 后, 未绑定 config 的行为依赖 _get_untied_lm_head 找不到 head 时的降级路径。
 - 重试语义变化: NotASafetensorsRepoError 从可重试改为致命错误, 可能影响瞬态网络错误下的容错。
- 影响:

- 用户侧：修复 Mistral3ForConditionalGeneration 等复合模型输出乱码问题；Qwen3-0.6B 等含冗余 lm_head 的 checkpoint 会被重新绑定并回收显存；config 与 checkpoint 不一致时给出明确告警而非静默错误。
- 系统侧：所有模型权重加载路径行为统一收敛到 weight_tying.py，后续新增模型无需再手写 lm_head 跳过逻辑，加载器语义更接近 Transformers 的 PreTrainedModel.tie_weights。
- 团队侧：为 GGUF 等新模型支持扫清 tie_word_embeddings 混乱（Isotr0py 评论中提到），降低后续模型接入成本；需要关注 70 文件改动引入的回归面，以及新增启发式对远程仓库元数据读取的依赖。
- 风险标记：核心加载路径变更，跨 70 文件大范围改动，依赖 checkpoint 元数据启发式，新增全量权重比较开销，非 safetensors 仓库重试语义变化

关联脉络

- 暂无明显关联 PR