PR #51280 完整报告

vllm-project/vllm

[ROCm][CI] Solidify entrypoint LLM lifecycle

合并时间：2026-08-14 01:24

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51280

执行摘要

- 一句话：entrypoint 测试统一到 VllmRunner 生命周期，修复 ROCm CI 显存回收
- 推荐动作：值得精读，尤其适合负责测试基础设施与 CI 稳定性的工程师。三个设计点值得借鉴：① 把 engine 生命周期收敛到上下文管理器，失败路径也走完整清理；② 用 weakref.proxy 对抗 pytest fixture 缓存造成的强引用；③ 并发 engine 场景下把显存等待从每个 runner 退出即时等待调整为全部退出后的有界等待。阅读时可重点对照 test_chat.py 的 fixture 与 test_gpu_utilization.py 的并发写法。

功能与动机

PR body 明确指出问题根因：This addresses AMD CI instability caused by entrypoint tests constructing LLM directly and relying on partial or ad hoc cleanup. Those paths could leave engine processes or lazily reclaimed ROCm allocations alive long enough to starve a later model startup. 即直接构造 LLM 的测试只依赖零散的 finalizer 或 cleanup_dist_env_and_memory，一旦测试失败或路径不完整，残留的 engine 进程与延迟回收的 ROCm 分配会耗尽显存，导致同一 CI 作业中后续模型启动失败。

实现拆解

1. 入口测试统一迁移到 vllm_runner 上下文管理器：test_struct_output_generate.py、test_generate.py、test_chat.py、test_prompt_validation.py 等将 LLM(...) 直接构造改为 with vllm_runner(...) as runner:，测试体内所有 llm.generate/llm.chat 改为 runner.llm 调用。这样 engine shutdown、Dynamo reset、分布式清理和 ROCm 显存等待在成功或失败路径都会执行，替代原先 request.addfinalizer 与 cleanup_dist_env_and_memory 的临时清理。
2. 删除自建清理设施收敛路径：tests/entrypoints/multimodal/conftest.py 删除了 _shutdown_llm、managed_llm、_make_managed_llm_factory、multimodal_llm_factory 共 73 行，多模态测试如 test_mm_cache_external_injection.py 从 multimodal_llm_factory 改为直接注入全局 vllm_runner fixture，避免多套清理语义不一致。
3. fixture 用 weakref.proxy 对抗 pytest 缓存：test_chat.py、test_generate.py 及 pooling 各组测试的 fixture 在 with vllm_runner(...) 内 yield weakref.proxy(runner.llm)。原因是 pytest 会把 fixture 的 yield 值缓存到 teardown，直接 yield runner.llm 会形成强引用，使 VllmRunner.__exit__释放的 LLM 与 ROCm 显存无法回收。

4. 并发 runner 场景单独处理：test_gpu_utilization.py 用三个嵌套 vllm_runner 同时启动 3 个 engine，验证 gpu_memory_utilization 是 per-instance 限制；多 engine 并存时单个 runner 退出无法回到显存基线，因此每个退出只做有界等待，待所有 runner 退出后才算真正回收。
5. 保留默认值差异与特殊场景：offline_mode 通过 _create_runner 辅助函数把 MODEL_CONFIGS 逐条转成 vllm_runner 并显式处理 tokenizer_name；test_tiling_engine.py 显式传入 max_model_len=None、enable_chunked_prefill=None 保持旧 LLM 构造的默认行为；weight transfer 测试将 patch 与 vllm_runner 组合进同一个 with (...) 块，保证退出顺序为 engine 先关、patch 后撤。 测试配套说明：本 PR 全部为测试代码改动（+974/-1075），无生产代码路径变更，靠 entrypoint 测试在 CI 上的稳定运行来验证效果。

关键文件：

- tests/entrypoints/llm/test_struct_output_generate.py（模块 结构化输出；类别 test；类型 test-coverage；符号 test_structured_output, test_guidance_no_additional_properties, generate_with_backend, test_structured_output_with_structural_tag）：改动量最大的文件（+566/-562），结构化输出测试从直接 LLM 构造 + request.addfinalizer 迁移到 vllm_runner 上下文，删除手工 cleanup，是本 PR 迁移模式的典型代表
- tests/entrypoints/multimodal/conftest.py（模块 多模态夹具；类别 test；类型 test-coverage；符号 _shutdown_llm, managed_llm, _make_managed_llm_factory, make_llm）：删除整套自建 shutdown 工具（-73 行），是清理逻辑收敛到 VllmRunner 的关键证据，也直接回应了 review 中关于清理归属的讨论
- tests/entrypoints/weight_transfer/test_weight_transfer_llm.py（模块 权重迁移；类别 test；类型 test-coverage；符号 test_get_world_size_tp1, test_init_weight_transfer_engine_calls_engine, test_update_weights_calls_engine, test_full_weight_transfer_flow）：展示 patch 与 vllm_runner 组合进同一 with 块、并用 weakref.proxy 访问 llm 的模式，是本 PR 处理多上下文组合的代表
- tests/entrypoints/llm/test_chat.py（模块 聊天入口；类别 test；类型 test-coverage；符号 text_llm, llm_for_failure_test, thinking_llm）：典型 fixture 生命周期迁移，体现 weakref.proxy 对抗 pytest fixture 缓存的核心设计，是本 PR 最值得复用的模式
- tests/entrypoints/llm/test_gpu_utilization.py（模块 显存利用；类别 test；类型 test-coverage；符号 test_gpu_memory_utilization）：并发多 runner 场景的代表，体现 body 中提到的 deferred ROCm VRAM settling 设计
- tests/entrypoints/llm/offline_mode/test_offline_mode.py（模块 离线模式；类别 test；类型 test-coverage；符号 _create_runner, cache_models, test_offline_mode, test_model_from_huggingface_offline）：展示需要保留默认值差异与 monkeypatch env 组合的适配方式，通过 _create_runner 封装 MODEL_CONFIGS 迁移

关键符号：test_structured_output, test_guidance_no_additional_properties, generate_with_backend, text_llm, llm_for_failure_test, thinking_llm, test_gpu_memory_utilization, _create_runner, test_init_weight_transfer_engine_calls_engine, test_max_model_len

关键源码片段

tests/entrypoints/weight_transfer/test_weight_transfer_llm.py

展示 patch 与 vllm_runner 组合进同一 with 块、并用 weakref.proxy 访问 llm 的模式，是本 PR 处理多上下文组合的代表

```
@create_new_process_for_each_test()
def test_init_weight_transfer_engine_calls_engine(vllm_runner):
    '''Verify LLM.init_weight_transfer_engine calls the mock engine.'''
    if torch.accelerator.device_count() < 1:
        pytest.skip('Need at least 1 GPU for this test')

    # 进程内运行以支持 mock.patch（spawn 子进程不会继承 mock 对象）
    os.environ['VLLM_ENABLE_V1_MULTIPROCESSING'] = '0'
    # collective_rpc 需要 pickle 函数，开启不安全序列化通道
    os.environ['VLLM_ALLOW_INSECURE_SERIALIZATION'] = '1'

    # patch 与 vllm_runner 组合进同一个 with 块，退出时
    # 按后进先出顺序先关 engine 再退出 patch，保证清理完整。
    with (
        patch(
            'vllm.v1.worker.gpu_worker.WeightTransferEngineFactory.create_engine',
            mock_create_engine,
        ),
        vllm_runner(
            MODEL_NAME,
            enforce_eager=True,
            load_format='dummy',
            tensor_parallel_size=1,
            weight_transfer_config=WeightTransferConfig(backend='nccl'),
        ) as runner,
    ):
        # 用 weakref.proxy 访问 llm，避免测试期间的强引用
        # 阻碍 VllmRunner.__exit__ 中的引擎关闭与显存回收。
        llm = weakref.proxy(runner.llm)

        # 先确认 engine 已创建
        def check_engine_exists(self):
            return self.weight_transfer_engine is not None

        results = llm.collective_rpc(check_engine_exists)
        assert all(results), 'Weight transfer engine should be initialized'

        # 再验证 init_weight_transfer_engine 是否把请求透传到引擎
        llm.init_weight_transfer_engine(
            WeightTransferInitRequest(init_info={'test_param': 'hello'})
        )

        def check_init_called(self):
```

```
            engine = self.weight_transfer_engine
            return (
                engine.init_transfer_engine_called,
                engine.last_init_info.test_param if engine.last_init_info else None,
            )

        results = llm.collective_rpc(check_init_called)
        for called, param in results:
            assert called, 'init_transfer_engine should have been called'
            assert param == 'hello', 'Expected hello, got ' + str(param)
```

tests/entrypoints/llm/test_chat.py

典型 fixture 生命周期迁移，体现 weakref.proxy 对抗 pytest fixture 缓存的核心设计，是本 PR 最值得复用的模式

```
import weakref

import pytest

from vllm.exceptions import VLLMValidationError
from vllm.sampling_params import SamplingParams


@pytest.fixture(scope='function')
def text_llm(vllm_runner):
    # VllmRunner 以上下文管理器接管 LLM 生命周期：退出 with 块时
    # 必然执行 engine shutdown、Dynamo reset、分布式清理，
    # 并在 ROCm 上等待显存回落，即使测试中途失败也一样。
    with vllm_runner(
        'meta-llama/Llama-3.2-1B-Instruct', enforce_eager=True, seed=0
    ) as runner:
        # pytest 会把 fixture 的 yield 值缓存到 teardown 结束，
        # 若直接 yield runner.llm 就会形成强引用，令 LLM 无法被
        # 垃圾回收，VllmRunner.__exit__ 释放的 ROCm 显存也回不来。
        # weakref.proxy 让测试只持有弱引用，生命周期完全交给 runner。
        yield weakref.proxy(runner.llm)


def test_chat(text_llm):
    prompt1 = 'Explain the concept of entropy.'
    messages = [
        {'role': 'system', 'content': 'You are a helpful assistant'},
        {'role': 'user', 'content': prompt1},
    ]
    outputs = text_llm.chat(messages)
    assert len(outputs) == 1
```

tests/entrypoints/llm/test_gpu_utilization.py

并发多 runner 场景的代表，体现 body 中提到的 deferred ROCm VRAM settling 设计

```
def test_gpu_memory_utilization(vllm_runner):
    prompts = [
        'Hello, my name is',
        'The president of the United States is',
        'The capital of France is',
        'The future of AI is',
    ]
    sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=16)

    # 同时启动 3 个 runner，验证 gpu_memory_utilization 是 per-instance
    # 上限而非全局限制。多个 engine 并存时，单个 runner 退出无法回到
    # 显存基线，因此 VllmRunner 的 __exit__ 只做有界等待，
    # 等所有 runner 都退出后显存才算真正回收完毕。
    with (
        vllm_runner('facebook/opt-125m', gpu_memory_utilization=0.3, enforce_eager=True) as
        runner_0,
        vllm_runner('facebook/opt-125m', gpu_memory_utilization=0.3, enforce_eager=True) as
        runner_1,
        vllm_runner('facebook/opt-125m', gpu_memory_utilization=0.3, enforce_eager=True) as
        runner_2,
    ):
        for runner in (runner_0, runner_1, runner_2):
            outputs = runner.llm.generate(prompts, sampling_params)
            for output in outputs:
                print(output.outputs[0].text)
```

评论区精华

核心争议围绕清理逻辑的归属。DarkLight1337 在 tests/entrypoints/conftest.py 看到新增的 vllm_runner_factory fixture 后提问：Hmm why don't we just add proper cleanup behavior in VllmRunner itself? 作者解释：VllmRunner already owns the actual cleanup in exit... This fixture only manages the pytest lifetime and the concurrent-runner case, where the per-runner ROCm wait must be deferred until all peers exit. DarkLight1337 坚持：I prefer using explicit context managers with VllmRunner instance instead of putting the cleanup logic outside. 作者最终移除 factory，全局改为显式 with vllm_runner(...) as runner: 写法，并说明 fixture 用 weakref proxy 是因为 pytest 会缓存 yield 值到 teardown，允许 LLM 与 ROCm 内存被完整释放。

- 清理逻辑应放在 VllmRunner 还是外部 fixture (design): 作者移除 vllm_runner_factory，所有测试与 fixture 改为显式 with vllm_runner(...) as runner:，清理逻辑全部由 VllmRunner.__exit__承担
- weakref.proxy 规避 pytest fixture 缓存导致的显存不释放 (design): 保留 weakref.proxy 模式，并在 test_chat.py、test_generate.py、pooling 等 fixture 中统一采用

风险与影响

- 风险：回归风险：24 个文件、约 1075 行删除和 974 行新增的大面积测试重写，任何 fixture 迁移遗漏或参数默认值差异都可能在 NVIDIA/CPU 等非 ROCm 平台误报或漏报；test_tiling_engine.py 需要显式补 max_model_len=None、enable_chunked_prefill=None 说明 VllmRunner 默认值与直接构造存在差异。时序风险：ROCm 显存回收依赖 weakref.proxy 与垃圾回收时机，若测试代码某处意外强引用 runner.llm，将复现显存饿死问题。并发风险：test_gpu_utilization.py 多 runner 场景每个退出只做有界等待，若超时阈值过短可能掩盖显存回落慢的问题或拖慢 CI。兼容风险：offline_mode 测试中 HF_HUB_OFFLINE 与 _re_import_modules 配合新的 runner 构造路径，若 vllm_runner 构造时触发额外 import 或网络访问，会破坏离线测试语义。覆盖面：无生产代码改动，风险限于测试基础设施本身。
- 影响：对 AMD ROCm CI 有直接收益：解决 entrypoint 测试遗留 engine 进程与 ROCm 分配导致的后续模型启动饥饿。对团队形成统一规范：tests/entrypoints 下的测试统一走 vllm_runner 生命周期，删除重复的清理工具代码，后续新增 entrypoint 测试可直接复用。对用户与生产系统无影响，因为完全不涉及 vllm 生产代码；但 NVIDIA/CPU 平台的 entrypoint 测试同样受益于清理路径统一。
- 风险标记：24 个文件大范围测试迁移，依赖 weakref 语义释放显存，ROCm 显存回收时序敏感，并发 runner 延迟回收，默认参数差异需显式适配

关联脉络

- PR #51653 [ROCm] Enable V2 model runner for Kimi-K3 on ROCm: 同属 ROCm 平台 runner 相关行为调整，本 PR 的 entrypoint 测试迁移依赖 V2 runner 路径的稳定性，两者在同一 ROCm CI 工作线上
- PR #51862 [ROCm][Perf] Kimi-K3 Remove prefill pipeline stall in chunk KDA: 同为 ROCm 平台稳定性与性能系列改动，与本 PR 一起构成 AMD 平台 CI 可靠性改善的持续投入
- PR #52127 [CI/Build][CPU] Shrink triton-cpu-build layer by dropping build artifacts: 平台 CI 基础设施改进脉络的一部分，说明仓库正在系统性收敛各平台的 CI 稳定性与构建效率问题