PR #48646 完整报告

vllm-project/vllm

[ROCm][CI] Reuse equivalent ROCm CI images

合并时间：2026-08-09 16:25

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48646

执行摘要

- 一句话：ROCm CI 镜像按内容寻址复用，缓存命中率提升
- 推荐动作：值得精读，尤其适合维护多阶段 Docker 构建与共享 CI 缓存池的团队。三个设计决策最值得借鉴： 1) 用“信任边界 + 内容寻址”区分 canonical 与 preview 缓存写入，在安全与命中率之间取得平衡； 2) 用 digest 固定步骤间交接并在关键路径 fail closed，杜绝“以为复用了实际没复用”的隐性错误； 3) 按昂贵依赖的变更频率拆分 Dockerfile 阶段（PyTorch 全家桶 / csrc / Rust / Python 依赖），让各阶段缓存独立失效。注意本 PR 无实质 review 讨论，主要审阅信号来自提交历史与 CI 验证结果，合入前建议对照最终 head 提交再做一轮评估。

功能与动机

PR body 直接陈述了目标：“split the ROCm base build into independently cacheable dependency stages - reuse base and ci_base images by deterministic content identity, with trust-scoped writes and digest-pinned downstream handoffs - keep csrc and Rust compiler caches stable across PR commits when their real inputs are unchanged ... retain per-commit wheel/image packaging and the native artifact contract”。背景是：ROCm CI 每个 commit 都要重建昂贵的 GPU 依赖层（PyTorch 全家桶、csrc 原生扩展、Rust 工具链），且持久化 worker 与 Kubernetes worker 的工作区文件权限不一致，导致相同输入算出不同内容哈希，镜像无法跨 commit、跨 PR 复用，硬件 CI 排队和成本居高不下。

实现拆解

1. 内容身份哈希与 checkout 规范化：重写 .buildkite/scripts/ci-bake-rocm.sh 的 compute_content_hash，用 git ls-files 枚举 checkout 内文件（排除构建残留物），并把文件 mode、symlink target、目录结构纳入哈希；新增 normalize_ci_worktree_modes，在 Buildkite 且 REMOTE_VLLM=0 时把共享工作区暴露的 0664/0775 权限恢复为 git 索引中的 0644/0755，解决持久化 worker 与 K8s worker 哈希不一致、无法复用 ci_base 的问题。
2. 信任边界与缓存写作用域：ci-bake-rocm.sh 新增 is_trusted_ci_cache_writer（非 PR + main 分支 + 官方仓库 slug 归一化一致）与 ci_base_write_scope，fork/PR 只能写 preview-<repo-hash> 隔离引用；refresh-base-image.sh 的 rocm_base_layer_cache_scope 给出 main / pr-<PR号>-<hash> / preview-<分支>-<hash> 三级层缓存作用域，并始终把可信的 rocm-base-main 作为只读 cache-from 回

退，兼顾安全与命中率。

3. digest 固定交接与失败闭合：build-ci-base.sh、build-test-image.sh 改用 load_digest_handoff 从 Buildkite metadata 读取 rocm-ci-base-image/rocm-base-image，且必须匹配 @sha256:... 才继续；resolve_image_digest 增加重试（默认 4 次、间隔 2 s）；Buildkite 下缺 metadata 或引用非 digest 固定时直接失败，杜绝跨 commit 混用镜像。
4. Dockerfile 阶段拆分：docker/Dockerfile.rocm_base 将 PyTorch/TorchVision/TorchAudio 拆为 build_pytorch → build_pytorch_runtime → build_torchvision/build_torchaudio 独立阶段，sccache 构建与 wheel 安装分离；docker/Dockerfile.rocm 新增 build_vllm_dependencies 集中安装 requirements，csrc-build 与 build_vllm（wheel 打包）共用该层，避免无关 CI/ 测试改动使原生编译层失效。
5. 流水线串行化与产物契约：.buildkite/hardware_tests/amd.yaml 给 refresh 与 build-ci-base 步骤加 concurrency: 1 + concurrency_group 防止并发重复构建；测试镜像 tag 改为 build-scoped 的 rocm/vllm-ci:build-$BUILDKITE_BUILD_ID，同时保留 commit alias 以满足 native 工件契约（prepare_native_workspace 校验 artifact base 以 ci_base-build-<BUILDKITE_BUILD_ID> 结尾）；强制 REMOTE_VLLM=0 并 unset VLLM_BRANCH，让内容身份只描述本地 checkout。
6. 配套改动：run-amd-test.sh 新增有界失败诊断（25 s 探测预算、5 s 单命令超时、VLLM_CI_DIAGNOSTICS_DIR 路径穿越防护、结果上传为 artifact）；ci_config_rocm.yaml 把 rust/、tools/、pyproject.toml、.dockerignore 等加入 run_all_patterns；tests/tools/test_docker_build_metadata_args.py 收缩为行为契约断言；ci-rocm.hcl 简化 tag 语义，用 CI_BASE_TRUSTED_CONTENT_REF 取代三个附加 tag 变量。

关键文件：

- .buildkite/scripts/ci-bake-rocm.sh（模块 镜像缓存；类别 other；类型 dependency-wiring；符号 docker_hub_repo, normalize_repo_slug, is_trusted_ci_cache_writer, ci_base_write_scope）：镜像构建与缓存编排的核心入口，重写最多（+702/-482）：内容哈希、信任边界、写作用域、digest 交接都在这里实现，是本次重构的中枢。
- .buildkite/scripts/rocm/refresh-base-image.sh（模块 基础镜像；类别 other；类型 core-logic；符号 rocm_base_layer_cache_scope, configure_rocm_base_layer_cache, resolve_image_digest, is_trusted_main_build）：base 镜像从‘按 Dockerfile 变化刷新’改为‘按内容身份选择、仅在缓存 miss 时构建’，定义了三级层缓存作用域与可信回退，是缓存命中率提升的关键。
- .buildkite/scripts/hardware_ci/run-amd-test.sh（模块 失败诊断；类别 infra；类型 infrastructure；符号 run_amd_diagnostic, collect_rocm_failure_diagnostics, clear_ci_orchestration_env, prepare_native_workspace）：新增 +247 行的有界失败诊断逻辑（amd-smi/rocm-smi/ 文件系统探测），并为 native artifact 契约增加 base 引用校验，是本次配套的稳定性保障。

- .buildkite/hardware_tests/amd.yaml（模块 流水线配置；类别 test；类型 configuration）：AMD 流水线定义：新增 concurrency 串行化、build-scoped 镜像 tag、REMOTE_VLLM=0 和 exit_status 2 重试，直接体现新的镜像流契约。
- docker/Dockerfile.rocm_base（模块 镜像定义；类别 infra；类型 infrastructure；符号 build_pytorch, build_pytorch_runtime, build_torchvision, build_torchaudio）：把 PyTorch/TorchVision/TorchAudio 构建拆成可独立缓存的阶段，是本次重构在镜像定义层的关键载体。
- docker/Dockerfile.rocm（模块 镜像定义；类别 infra；类型 infrastructure；符号 build_vllm_dependencies, csrc-build, build_vllm）：新增 build_vllm_dependencies 共享依赖安装层，让 csrc-build 与 wheel 打包共享昂贵层，原生编译缓存不再被无关改动击穿。
- docker/ci-rocm.hcl（模块 构建配置；类别 infra；类型 infrastructure；符号 ci-base-rocm-ci, CI_BASE_TRUSTED_CONTENT_REF）：简化 ci_base 目标的 cache/tag 语义，用 CI_BASE_TRUSTED_CONTENT_REF 取代三个附加 tag 变量，并关闭 provenance attestation。
- .buildkite/scripts/rocm/build-test-image.sh（模块 测试镜像；类别 other；类型 core-logic；符号 load_digest_handoff）：用 load_digest_handoff 替代原来的 use_ci_base_if_present/use_refreshed_base_if_present，交接必须 digest 固定，失败闭合。
- .buildkite/ci_config_rocm.yaml（模块 CI 触发；类别 config；类型 configuration）：run_all_patterns 大幅扩展（rust/、tools/、pyproject.toml、.dockerignore 等），保证缓存契约相关输入变化能触发完整验证。
- .buildkite/scripts/rocm/build-ci-base.sh（模块 CI 基础层；类别 other；类型 core-logic）：ci_base 构建前校验 base 交接 digest 并强制本地 checkout 身份，是内容身份一致性的重要一环。
- tests/tools/test_docker_build_metadata_args.py（模块 测试配套；类别 test；类型 test-coverage）：缓存契约测试从实现形断言收敛为行为契约断言，是作者在提交历史中主动“瘦身”的产物。
- .buildkite/scripts/rocm/smoke-test-image.sh（模块 冒烟测试；类别 other；类型 core-logic）：镜像冒烟检查随新 tag 语义与 build-scoped 引用调整，是镜像可用性的最后一层保障。
- .dockerignore（模块 构建上下文；类别 other；类型 core-logic）：排除构建产物，保证重复构建拥有稳定的源层与内容哈希。

关键符号：is_trusted_ci_cache_writer, ci_base_write_scope, configure_ci_base_write_scope, compute_content_hash, normalize_ci_worktree_modes, rocm_base_layer_cache_scope, configure_rocm_base_layer_cache, resolve_image_digest, load_digest_handoff, run_amd_diagnostic, collect_rocm_failure_diagnostics, prepare_native_workspace

关键源码片段

.buildkite/scripts/ci-bake-rocm.sh

镜像构建与缓存编排的核心入口，重写最多（+702/-482）：内容哈希、信任边界、写作用域、digest 交接都在这里实现，是本次重构的中枢。

```
# ----------------------------------------------------------------------
# 信任边界：谁能写 canonical 的 ci_base 缓存？
# 只有官方仓库 main 分支的非 PR 构建才是“可信写入者”，
# 可以直接发布 canonical 引用；fork 与 PR 只能写隔离的 preview 作用域，
# 防止不可信内容污染共享缓存。
# ----------------------------------------------------------------------

# 校验运行环境是否满足“可信构建”的全部条件：
# 1. 运行在 Buildkite 上；
# 2. 不是 PR 构建；
# 3. 分支为稳定主分支（默认 main）；
# 4. 仓库身份与官方 vllm-project/vllm 一致。
is_trusted_ci_cache_writer() {
    local actual_repo=""
    local trusted_repo=""

    [[ "${BUILDKITE:-false}" == "true" ]] || return 1
    [[ "${BUILDKITE_PULL_REQUEST:-false}" == "false" ]] || return 1
    [[ "${BUILDKITE_BRANCH:-}" == "${CI_BASE_STABLE_BRANCH:-main}" ]] || return 1

    actual_repo=$(normalize_repo_slug "${BUILDKITE_REPO:-}")
    trusted_repo=$(normalize_repo_slug \
        "${CI_BASE_STABLE_REPO_SLUG:-${DEFAULT_REPO_SLUG}}")
    [[ -n "${actual_repo}" && "${actual_repo}" == "${trusted_repo}" ]]
}

# 为不可信来源计算隔离写入作用域：
# 取源仓库 slug 的 sha256 前 12 位作为身份标识，
# 保证不同 fork 之间缓存互不串扰，同一 fork 则可稳定复用。
ci_base_write_scope() {
    local identity=""
    local source_repo="${BUILDKITE_PULL_REQUEST_REPO:-${BUILDKITE_REPO:-local}}"

    if is_trusted_ci_cache_writer; then
        return 0
    fi
    identity=$(printf '%s\n' "${source_repo}" | sha256sum | cut -c1-12)
    printf 'preview-%s\n' "${identity}"
}

# 将作用域写入全局变量并导出，供后续 docker bake 的 cache-to 引用使用。
configure_ci_base_write_scope() {
    local scope=""

    scope=$(ci_base_write_scope)
    if [[ -n "${scope}" ]]; then
```

```
        CI_BASE_WRITE_SCOPE=$(clean_docker_tag "${scope}")
        echo "Non-canonical cache writes use source scope: ${CI_BASE_WRITE_SCOPE}"
    else
        CI_BASE_WRITE_SCOPE=""
        echo "Trusted main build: publishing canonical ci_base refs"
    fi
    export CI_BASE_WRITE_SCOPE
}
```

.buildkite/scripts/rocm/refresh-base-image.sh

base 镜像从‘按 Dockerfile 变化刷新’改为‘按内容身份选择、仅在缓存 miss 时构建’，定义了三级层缓存作用域与可信回退，是缓存命中率提升的关键。

```
# ------------------------------------------------------------------------
# ROCm base 镜像的层缓存作用域：
# - 可信主分支构建：直接复用并写入 canonical 的 rocm-base-main；
# - PR 构建：pr-<PR 号 >-< 仓库哈希 >，避免 fork 之间互相污染；
# - 其余分支构建：preview-< 分支短名 >-< 仓库 + 分支哈希 >。
# 所有非可信来源都只写自己作用域的引用，
# 同时把可信的 rocm-base-main 作为只读 cache-from 回退，
# 让 PR/ 分支也能吃到官方镜像的层缓存加速。
# ------------------------------------------------------------------------
rocm_base_layer_cache_scope() {
    local pull_request="${BUILDKITE_PULL_REQUEST:-false}"
    local branch="${BUILDKITE_PULL_REQUEST_HEAD_BRANCH:-${BUILDKITE_BRANCH:-local}}
    "
    local identity=""
    local repo_slug=""

    if is_trusted_main_build; then
        printf 'main\n'
        return 0
    fi
    if [[ "${pull_request}" != "false" && -n "${pull_request}" ]]; then
        repo_slug=$(normalize_repo_slug "${BUILDKITE_REPO:-local}")
        identity=$(printf '%s\n' "${repo_slug:-local}" | sha256sum | cut -c1-12)
        printf 'pr-%s-%s\n' \
            "$(tag_component "${pull_request}" 32)" "${identity}"
        return 0
    fi

    identity=$(printf '%s\n%s\n' "${BUILDKITE_REPO:-local}" "${branch}" \
        | sha256sum | cut -c1-12)
    printf 'preview-%s-%s\n' "$(tag_component "${branch}" 24)" "${identity}"
}

# 组装 BuildKit 的 Registry 层缓存参数：
# cache-to 只写当前作用域并允许失败（ignore-error），
# cache-from 优先当前作用域，非可信来源再追加可信回退。
```

```
configure_rocm_base_layer_cache() {
    local scope=""

    ROCM_BASE_CACHE_ARGS=()
    if [[ "${ROCM_BASE_NO_CACHE:-0}" == "1" ]]; then
        ROCM_BASE_CACHE_ARGS+=(--no-cache)
        ROCM_BASE_LAYER_CACHE_REF="disabled"
        return 0
    fi

    scope=$(rocm_base_layer_cache_scope)
    ROCM_BASE_LAYER_CACHE_REF="${CACHE_REPO}:rocm-base-${scope}"
    ROCM_BASE_CACHE_ARGS+=(
        --cache-from "type=registry,ref=${ROCM_BASE_LAYER_CACHE_REF}"
    )
    if [[ "${ROCM_BASE_LAYER_CACHE_REF}" != \
        "${ROCM_BASE_TRUSTED_LAYER_CACHE_REF}" ]]; then
        ROCM_BASE_CACHE_ARGS+=(
            --cache-from "type=registry,ref=${ROCM_BASE_TRUSTED_LAYER_CACHE_REF}"
        )
    fi
    ROCM_BASE_CACHE_ARGS+=(
        --cache-to \
        "type=registry,ref=${ROCM_BASE_LAYER_CACHE_REF},mode=max,ignore-error=true"
    )
}
```

评论区精华

本 PR 没有任何实质性 inline review 讨论（review_comments 为 0）。唯一的 reviewer 评论是 claude[bot] 的自动提示——来自 fork 的 PR 自动评审被禁用，需维护者手动触发；最终由 tjtanaa 空正文 APPROVED、vllm-bot 合入。有价值的"讨论"沉淀在提交历史中：作者与 OpenAI Codex 协作、历经 34 个提交的多轮收敛，期间曾引入 Buildx history 上传与大型内部测试，随后在 "Defer Buildx history artifacts""Minimize cache contract implementation""Remove unrelated Docker/runtime changes" 等提交中主动删减，把测试收敛为"行为契约"级断言；mergify[bot] 两次提示 merge conflict，头部分支多次合并 main、最终 rebase 后才合入。作者也手动触发了多轮 AMD CI（amd-ci builds 11254/11734 成功、Buildkite CI 82555/82885）验证缓存链路。

- Fork PR 自动评审被停用 (other): tjtanaa 以空正文 APPROVED 完成人工审批，PR 最终由 vllm-bot 合入。
- 重复 merge conflict 与 rebase (other): 作者在合入前完成最终 rebase（head 为 ae6251d72c2a45d3d226c470991275e2f09037cb），冲突全部解决。
- AMD CI 构建验证 (testing): 流水线验证通过后合入；无新增未解决问题。

风险与影响

- 风险：
  1. ci-bake-rocm.sh 一次性重写约 +702/-482，是镜像构建的唯一入口，涉及 awk 解析 Dockerfile 阶段、内容哈希管线与 docker bake 参数拼接，回归影响集中但排障成本高；现有测试只覆盖 Docker 构建元数据，没有直接覆盖这些 shell 脚本。
  2. fail closed 策略：Buildkite 下 build-ci-base.sh/build-test-image.sh 遇到缺 metadata 或非 digest 引用直接报错，若 buildkite-agent meta-data 异常或步骤依赖调整，会整体阻断 AMD 流水线（脚本注释中也明确 exit_status 2 对应"镜像已找到但 registry 交接失败"）。
  3. normalize_ci_worktree_modes 会对共享 checkout 执行 chmod，若并发 job 共用同一工作区且依赖可执行位，存在交叉影响；对 unmerged/ 异常 mode 返回失败属于安全侧设计。
  4. 缓存作用域依赖 normalize_repo_slug 对多种 URL 形态（https/ssh/git@）的解析，解析遗漏会造成 fork 身份碰撞或写错作用域。
  5. Docker tag 128 字符上限通过 ROCM_CACHE_BRANCH_TAG_MAX_LEN=111 预留，但长分支名场景仍有截断冲突风险。
  6. run-amd-test.sh 新增诊断会上传 artifact 并在失败路径增加耗时，VLLM_CI_DIAGNOSTICS_DIR 已有路径防御，但 25 s 预算在极端慢节点上可能覆盖不到文件系统探测。 - 影响：对 AMD/ROCm CI：base/ci_base 按内容身份复用后，内容未变的 commit 直接跳过重建（脚本注释称约 <30 s），显著降低硬件 CI 排队与成本；Rust/csrc 缓存跨 commit 稳定，减少重复 HIP/C++ 编译。对贡献者：PR 与 fork 获得隔离的 preview 缓存写入与可信回退，既避免污染 canonical 镜像，又能享受缓存加速；per-commit wheel/image 打包与 native artifact 契约保留，外部 AMD pod 模板不受破坏。对团队：CI 依赖 Buildkite metadata 的 digest 交接，未来调整步骤顺序需同步维护脚本；新增失败诊断为 ROCm 测试提供统一的 GPU 病理解剖工具。对用户侧无任何影响（不涉及运行时引擎、模型或前端代码）。 - 风险标记：CI 核心脚本大面积重写，失败闭合策略可能阻断流水线，跨 worker 权限规范化依赖 git 索引，缓存作用域与标签长度约束复杂

关联脉络

- PR #51058 [Build] Upgrade runtime image to Ubuntu 24.04, pick up rdma-core > 44: 同属 Docker 构建基础设施演进（docker/versions.json、Dockerfile、安装文档），为 ROCm 镜像的构建平台提供底层基础。
- PR #50805 [ROCm][CI] Baseline legacy extensions in the Torch ABI audit: 同为 ROCm CI 稳定性修复（Torch ABI 审计脚本），与本次镜像缓存重构同属 AMD 流水线演进脉络。
- PR #51457 [Test] Add ROCm AITER FP8 MLA prefill accuracy test: ROCm 测试覆盖扩展，可直接受益于本次镜像缓存加速带来的 CI 排队下降。