PR #6702 完整报告

verl-project/verl

[hardware] feat: add ROCm/HIP platform backend (PlatformROCm)

合并时间：2026-06-12 14:37

原文链接：http://prhub.com.cn/verl-project/verl/pull/6702

执行摘要

- 一句话：新增 AMD ROCm 平台后端 PlatformROCm
- 推荐动作：本 PR 设计清晰，通过类继承与最小化覆写策略为 AMD ROCm 添加了一等平台支持。建议关注以下设计点：1）PlatformROCm 继承 PlatformCUDA 而非完全从零实现，适用于 API 兼容性高的场景；2）通过 guard 在父类中短路避免误检测；3）在 rollout_env_vars 中提供用户可覆写的默认值。该模式可供后续新硬件平台（如 Intel GPU）参考。

功能与动机

在 AMD ROCm 硬件上训练时，verl 原本会错误地选中 PlatformCUDA 后端，导致部分行为不兼容。本 PR 的目的是为 ROCm 提供专用的平台后端，通过继承 CUDA 后端并覆盖差异点，实现自动检测和正确配置，避免静默回退。

实现拆解

1. 新增 PlatformROCm 类(verl/plugin/platform/platform_rocm.py)：定义 PlatformROCm(PlatformCUDA)，通过 @PlatformRegistry.register(platform="amd") 注册。覆盖以下方法：
   - vendor_name：返回 "amd"，device_name 保持 "cuda" 因为 PyTorch ROCm 通过 hipify 暴露 torch.device("cuda")。
   - is_platform_available：先检查 torch.version.hip 是否非空，以区分 AMD 和 NVIDIA；然后可选执行 rocm-smi 检测（镜像 PlatformCUDA 的 nvidia-smi 检测逻辑）。
   - rollout_env_vars：继承父类后添加 SGLANG_USE_AITER 变量，默认启用 AITER 内核，允许用户通过环境变量覆盖。
   - ray_noset_envvars：添加 HIP_VISIBLE_DEVICES 和 ROCR_VISIBLE_DEVICES 的 NOSET 变量，防止 Ray 误管理。
2. 修改 PlatformCUDA 平台检测(verl/plugin/platform/platform_cuda.py)：在 is_platform_available 方法中增加一段 guard，当 torch.version.hip is not None 时立即返回 False，确保 AMD 硬件不会误匹配到 CUDA 后端。
3. 注册平台(verl/plugin/platform/platform_manager.py)：在模块底部的导入区添加 from . platform_rocm import PlatformROCm，使装饰器在模块加载时生效。

4. 更新 CI 设备 API 检查(tests/special_sanity/check_device_api_usage.py)：将新文件 platform_rocm.py 加入 CUDA_KEYWORD_CHECK_WHITELIST，因为其内部合理引用了 torch.cuda。

关键文件：

- verl/plugin/platform/platform_rocm.py（模块 平台层；类别 source；类型 core-logic；符号 PlatformROCm, vendor_name, is_platform_available, rollout_env_vars）：新增的 ROCm 平台后端核心实现，定义 PlatformROCm 类并覆盖关键方法。
- verl/plugin/platform/platform_cuda.py（模块 平台层；类别 source；类型 core-logic；符号 is_platform_available）：修改了 is_platform_available 方法，增加 HIP 检测门控，防止自动检测选错平台。
- verl/plugin/platform/platform_manager.py（模块 平台层；类别 source；类型 dependency-wiring）：注册 PlatformROCm，使平台自动检测机制能够识别和创建新的平台实例。
- tests/special_sanity/check_device_api_usage.py（模块 设备 API 检查；类别 test；类型 test-coverage）：将 platform_rocm.py 加入白名单，允许其引用 torch.cuda，避免 CI 扫描报错。

关键符号：PlatformROCm.is_platform_available, PlatformROCm.rollout_env_vars, PlatformROCm.ray_noset_envvars, PlatformROCm.vendor_name, PlatformCUDA.is_platform_available

关键源码片段

verl/plugin/platform/platform_rocm.py

新增的 ROCm 平台后端核心实现，定义 PlatformROCm 类并覆盖关键方法。

```
# Copyright (c) 2026 BAAI. All rights reserved.
"""AMD ROCm/HIP platform implementation.

ROCm is largely CUDA-compatible: PyTorch on ROCm reuses the ``torch.cuda.*``
API surface via hipify, so most of ``PlatformCUDA`` works unchanged. This class
therefore subclasses ``PlatformCUDA`` and only overrides the parts that differ
on ROCm, following a "CUDA base + ROCm extensions" model (always extend the
parent via ``super()`` rather than re-implementing it).
"""

import os
import shutil

import torch

from .platform_cuda import PlatformCUDA
from .platform_manager import PlatformRegistry


@PlatformRegistry.register(platform="amd")
```

```
class PlatformROCm(PlatformCUDA):
    """Platform backend for AMD ROCm/HIP GPUs (reuses PlatformCUDA where compatible)."""

    @property
    def vendor_name(self) -> str:
        # NOTE: device_name stays 'cuda' on purpose — PyTorch ROCm exposes the
        # device type string as "cuda" (torch.device("cuda") works via hipify).
        return "amd"

    def is_platform_available(self, use_smi_check=False) -> bool:
        if not hasattr(torch, "cuda"):
            return False
        # Only ROCm (HIP) torch builds qualify as the AMD platform.
        if torch.version.hip is None:
            return False
        if use_smi_check:
            # In CPU-only Ray actors, torch.cuda.is_available() may return False
            # even though the cluster has GPUs. Fall back to rocm-smi check,
            # and if that's also unavailable (e.g. not on PATH), treat
            # torch.cuda being built as sufficient evidence.
            cmd = "rocm-smi"
            cmd_path = shutil.which(cmd)
            if cmd_path is None:
                # Fallback to common absolute paths if not found in PATH
                common_paths = [
                    f"/usr/bin/{cmd}",
                    f"/usr/local/bin/{cmd}",
                    f"/opt/rocm/bin/{cmd}",
                ]
                for path in common_paths:
                    if os.path.isfile(path) and os.access(path, os.X_OK):
                        cmd_path = path
                        break
                if cmd_path is None:
                    return False # Consistent with PlatformCUDA behavior
            if self.check_smi_command(cmd_path):
                return True
        return torch.cuda.is_available()

    def rollout_env_vars(self) -> dict[str, str]:
        # Extend CUDA's rollout env vars with ROCm-specific ones. SGLANG_USE_AITER
        # routes SGLang's non-attention kernels (RMSNorm/RoPE/MoE/quant) through AITER.
        # Default to "1" but honor an explicit user override (e.g. SGLANG_USE_AITER=0
        # to fall back to vLLM kernels).
        return {
            **super().rollout_env_vars(),
            "SGLANG_USE_AITER": os.environ.get("SGLANG_USE_AITER", "1"),
        }
```

```
    def ray_noset_envvars(self) -> list[str]:
        # On ROCm, HIP_VISIBLE_DEVICES takes precedence over CUDA_VISIBLE_DEVICES,
        # and ROCR_VISIBLE_DEVICES is also relevant, so tell Ray not to manage them.
        return super().ray_noset_envvars() + [
            "RAY_EXPERIMENTAL_NOSET_HIP_VISIBLE_DEVICES",
            "RAY_EXPERIMENTAL_NOSET_ROCR_VISIBLE_DEVICES",
        ]
```

verl/plugin/platform/platform_cuda.py

修改了 is_platform_available 方法，增加 HIP 检测门控，防止自动检测选错平台。

```
  def is_platform_available(self, use_smi_check=False) -> bool:
      if not hasattr(torch, "cuda"):
          return False
      # On ROCm, torch.cuda is present too; defer to PlatformROCm so that
      # auto-detection does not pick CUDA on AMD hardware.
      if torch.version.hip is not None:
          return False
      if use_smi_check:
          cmd = "nvidia-smi"
          cmd_path = shutil.which(cmd)
          if cmd_path is None:
              common_paths = [
                  f"/usr/bin/{cmd}",
                  f"/usr/local/bin/{cmd}",
                  f"/usr/local/cuda/bin/{cmd}",
              ]
              for path in common_paths:
                  if os.path.isfile(path) and os.access(path, os.X_OK):
                      cmd_path = path
                      break
              if cmd_path is None:
                  return False
          if self.check_smi_command(cmd_path):
              return True
      return torch.cuda.is_available()
```

评论区精华

Gemini Code Assist Bot 在 review 中指出 platform_rocm.py 第 55 行，当 rocm-smi 找不到时返回 False 与注释中描述的“将 built torch.cuda 视为充分证据”矛盾，建议返回 True。作者 xiaohong42 回复表示保持 False 以与 PlatformCUDA.is_platform_available 行为一致，遵循最小改动原则，且父类在该场景下同样返回 False。该讨论无后续争议，状态为已解决。

- rocm-smi 不可用时返回 False 是否合理 (correctness): 保持 False，与 PlatformCUDA 行为一致，遵循最小改动原则。

风险与影响

- 风险：
  1. 平台检测正确性：若 ROCm 系统上 rocm-smi 不在任何搜索路径且 torch.cuda.is_available() 因某种原因返回 False（如 CPU-only actor），则 is_platform_available 会返回 False，导致平台检测失败。但此场景罕见，且与 CUDA 后端的现有行为一致（同样在 nvidia-smi 找不到时返回 False）。
  2. 继承风险：PlatformROCm 继承 PlatformCUDA 的多数方法（如 set_device、device_count），若未来 CUDA 后端引入不兼容 ROCm 的修改，可能波及 AMD 用户；需通过 CI 或手动回归覆盖。
  3. 测试覆盖缺失：由于需要真实 AMD ROCm 硬件，未添加单元测试；手动验证覆盖了主要功能，但边缘路径缺少保护。 - 影响：用户：AMD ROCm 用户在启动训练时会自动选用 PlatformROCm，环境变量 SGLANG_USE_AITER 默认开启，可通过显式导出覆盖。现有 CUDA 用户无感知，因 PlatformCUDA 新增的门控会跳过 AMD 硬件。系统：平台管理器当前支持三个后端：nvidia、npu、amd。get_platform() 的自动检测逻辑扩展为优先匹配 HIP 构建。团队：新增的 ROCm 后端基于继承实现，添加新方法时需注意同步。持续维护成本较低。 - 风险标记：缺少自动化测试（需硬件），继承 CUDA 可能引入兼容性问题，平台检测路径依赖

关联脉络

- PR #6668 [ci, hardware] feat: add AMD ROCm (MI300) e2e_ppo_trainer workflow: 同为 AMD ROCm 支持，该 PR 添加了 CI 工作流，本 PR 提供平台后端，两者共同构成 ROCm 支持的基础设施。