

PR #2836 完整报告

radixark/miles

examples: launcher for the in-process Harbor path

合并时间: 2026-09-09 09:52

原文链接: <http://prhub.com.cn/radixark/miles/pull/2836>

执行摘要

- 一句话: 新增 Harbor in-process 路径的启动器, 并提取共享的沙箱提供商凭证配置逻辑。
- 推荐动作: 此 PR 值得精读。它不仅仅是一个新功能(启动器)的添加, 更重要的是它通过重构解决了跨示例的代码重复和配置漂移问题。关键设计决策在于: 1) 将提供商配置序列提取为 `provision_provider`, 这是一个清晰的、单一职责的公共抽象; 2) 创建 `launch_common.py` 作为 Harbor 启动逻辑的共享层, 确保配方与测试的一致性; 3) 在 `harbor_env_vars` 中实施早期、明确的环境校验和规范化。这些模式(早期校验、共享配置层、行为保持的重构)可以借鉴到仓库中其他示例的维护中。建议关注 `provision_provider` 的实现和它如何简化了 `openenv_launch_common.py`, 以及 `harbor_env_vars` 中的环境校验逻辑。

功能与动机

动机源于 #2804 (追踪问题) 和 #2806 (基础 Harbor 代理函数)。在开发 Harbor in-process 路径时, 发现 OpenEnv 启动器中的沙箱提供商凭证配置逻辑即将在 Harbor 启动器中被重复实现。PR body 明确指出: 'A launcher for the in-process Harbor path, plus the one piece of launcher logic the OpenEnv and Harbor examples would otherwise duplicate.' 本 PR 的目标是:

- 1) 提供一个可运行的 Harbor GLM-4.7-Flash 启动器;
- 2) 将重复的提供商配置序列提取为共享函数;
- 3) 将 agentic 训练参数和 PYTHONPATH 配置集中管理, 消除配方与 GPU 端到端测试之间的配置漂移风险。

实现拆解

1. 核心库增强: 提取 `provision_provider`: 在 `miles/rollout/agentic/credentials.py` 中新增 `provision_provider` 函数。该函数封装了单个沙箱提供商的完整启动器侧配置流程: 通过 `sandbox_key_supply` 处理密钥文件路径传递(而非值), 调用 `preflight_sdk` 进行 SDK 可导入性检查, 遍历 `spec["forward"]` 列表将地址类环境变量按值转发到 worker, 并在有 `target` 配置时打印端点信息。此函数旨在被所有需要配置沙箱提供商的启动器复用。
2. OpenEnv 启动器重构: 修改 `examples/experimental/openenv/openenv_launch_common.py` 的 `apply_optional_env_vars` 函数, 将其内部 20 行的顺序调用 (`sandbox_key_supply`, `preflight_sdk`, `forward_address`, `print`) 替换为对 `provision_provider` 的单一调用。此变更行为保持不变, 但移除了 `openenv`: 日志前缀, 使输出与其他提供商消息格式统一。

3. 新增 Harbor 共享启动逻辑：创建 `examples/experimental/harbor/launch_common.py`。该模块定义了三个关键函数：`agentic_pythonpath_dirs()` 返回所有 Harbor 启动器都需要放入 worker PYTHONPATH 的目录列表；`agentic_train_args()` 生成传递给 `train.py` 的 `agentic` 相关参数字符串（如 `--custom-agent-function-path`, `--use-session-server`），确保配方与 GPU 端到端测试使用同一份配置；`harbor_env_vars()` 组装 Harbor 启动器所需的 worker 环境变量，包含环境类型校验（拒绝 docker）、依赖导入检查、以及通过 `provision_provider` 进行的提供商配置。
4. 新增 Harbor 启动器 `run.py`：实现 GLM-4.7-Flash GRPO 的 Harbor in-process 启动器。定义 `ScriptArgs` 数据类包含所有训练、Harbor 和 W&B 配置。`main` 函数依次调用 `cleanup()`（终止残留进程）、`prepare()`（模型转换）和 `execute()`（构建训练参数、组装环境变量并执行训练）。`execute` 函数通过导入并调用 `launch_common` 中的 `agentic_train_args` 和 `harbor_env_vars` 来获取共享配置。
5. 测试覆盖：新增 `tests/fast/examples/experimental/harbor/test_launch_common.py`，全面测试 `harbor_env_vars` 的逻辑，包括：空环境类型报错、docker 类型被拒绝（含大小写 / 空格变体）、`harbor` 模块缺失时报错、已知提供商（e2b）的密钥路径配置、未知提供商的透传提示、以及环境变量和 per-trial 参数的转发。测试通过 monkeypatch 模拟 `harbor` 和 SDK 模块。修改 `tests/fast/rollout/agentic/test_credentials.py`，新增 `test_provision_provider_wires_key_path_addresses_and_preflight` 测试用例，验证 `provision_provider` 的端到端行为。

关键文件：

- `examples/experimental/harbor/run.py`（模块 Harbor 启动器；类别 source；类型 core-logic；符号 `ScriptArgs`, `cleanup`, `prepare`, `execute`）：本 PR 的主要交付物：Harbor in-process 路径的 GLM-4.7-Flash GRPO 启动器，包含完整的训练配置、清理、准备和执行流程。
- `examples/experimental/harbor/launch_common.py`（模块 启动逻辑共享层；类别 source；类型 dependency-wiring；符号 `agentic_pythonpath_dirs`, `agentic_train_args`, `harbor_env_vars`）：共享层，封装了 Harbor 启动器的环境变量组装、提供商配置和 `agentic` 参数生成，是消除重复、确保配置一致性的关键模块。
- `miles/rollout/agentic/credentials.py`（模块 Agentic 凭证管理；类别 source；类型 core-logic；符号 `provision_provider`）：核心库变更：新增 `provision_provider` 函数，将重复的沙箱提供商配置序列抽象为单一调用点，供所有启动器复用。
- `tests/fast/examples/experimental/harbor/test_launch_common.py`（模块 启动器测试；类别 test；类型 test-coverage；符号 `_args`, `fake_harbor_and_sdk`, `test_env_type_is_required`, `test_docker_is_refused_with_a_pointer_to_the_agent_server`）：对 Harbor 启动共享逻辑 `harbor_env_vars` 的全面测试，覆盖了环境校验、提供商配置、参数转发等关键路径，确保了重构的正确性。
- `examples/experimental/openenv/openenv_launch_common.py`（模块 OpenEnv 启动器；类别 source；类型 dependency-wiring）：现有 OpenEnv 启动器的重构，使用新的 `provision_provider` 替换重复的配置序列，验证了共享函数的复用性。

关键符号：`provision_provider`, `harbor_env_vars`, `agentic_train_args`, `agentic_pythonpath_dirs`, `run.py::execute`, `run.py::main`, `run.py::cleanup`,

run.py::prepare, apply_optional_env_vars

关键源码片段

examples/experimental/harbor/run.py

本 PR 的主要交付物：Harbor in-process 路径的 GLM-4.7-Flash GRPO 启动器，包含完整的训练配置、清理、准备和执行流程。

examples/experimental/harbor/run.py

Harbor in-process 路径的 GLM-4.7-Flash GRPO 启动器。main 函数编排启动流程。

```
def execute(args: ScriptArgs):
    """构建所有训练参数并执行训练。"""
    # ... ( 省略其他参数构建 ) ...
    agent_args = agentic_train_args(tito_model="glm47", session_server_workers=32) #
    导入共享的 agentic 参数
    # ... ( 省略 misc_args ) ...
    train_args = (f"{ckpt_args}{rollout_args}{optimizer_args}{grpo_args}{wandb_args}{trace_args}"
                  f"{perf_args}{sglang_args}{agent_args}{misc_args}{debug_args}")

    extra_env_vars = {
        "PYTHONPATH": ":".join([args.megatron_path, *agentic_pythonpath_dirs(), str(U.repo_
            base_dir)]),
        **harbor_env_vars(args), # 调用共享模块组装 Harbor 侧环境变量
    }
    if args.miles_host_ip:
        extra_env_vars["MILES_HOST_IP"] = args.miles_host_ip

    U.execute_train(train_args=train_args, config=args, num_gpus_per_node=args.num_gpus_
        per_node,
                    megatron_model_type=args.megatron_model_type, megatron_path=args.
                    megatron_path,
                    extra_env_vars=extra_env_vars)

@U.dataclass_cli
def main(args: ScriptArgs):
    cleanup() # 终止残留的 sglang/train 进程
    if not args.skip_prepare:
        prepare(args) # 调用 U.convert_checkpoint 进行模型转换
    execute(args) # 构建参数并启动训练
```

examples/experimental/harbor/launch_common.py

共享层，封装了 Harbor 启动器的环境变量组装、提供商配置和 agentic 参数生成，是消除重复、确保配置一致性的关键模块。

examples/experimental/harbor/launch_common.py

Harbor 启动器的共享逻辑，独立于重型训练依赖，以便测试。

```

def harbor_env_vars(args) -> dict[str, str]:
    """组装 Harbor 启动器所需的 worker 环境变量。"""
    # 规范化环境类型，与 worker 侧行为一致，防止大小写 / 空格问题导致绕过检查
    env_type = args.harbor_env_type.strip().lower()
    if not env_type:
        raise ValueError("set --harbor-env-type / HARBOR_ENV_TYPE (e.g. e2b, daytona): ...")
    if env_type == "docker":
        raise ValueError("docker needs a Docker daemon next to Trial.run(); use examples/swe-agent-harbor-docker (agent server) for it")
    try:
        import harbor # noqa: F401 # the fork branch, see README
    except ImportError as e:
        raise RuntimeError("harbor is not importable ...; see the README install line") from e

    env = {"HARBOR_ENV_TYPE": env_type, "HARBOR_TASKS_DIR": args.harbor_tasks_dir, ...}
    # 根据提供商 spec 调用共享的 provision_provider 进行配置
    if spec := PROVIDER_CREDENTIALS.get(env_type):
        provision_provider(env, spec, arg_path=getattr(args, spec["arg_attr"], "") or "")
    else:
        print(f"harbor: no credential wiring known for {env_type!r}; ...", flush=True)
    # 转发 per-trial 可选参数
    for var in ("AGENT_MAX_INPUT_TOKENS", "HARBOR_RESPONSE_LENGTH_POLICY", ...):
        if value := os.environ.get(var, "").strip():
            env[var] = value
    return env

```

miles/rollout/agent/credentials.py

核心库变更：新增 `provision_provider` 函数，将重复的沙箱提供商配置序列抽象为单一调用点，供所有启动器复用。

```

# miles/rollout/agent/credentials.py
# 新增函数，封装单个提供商的完整配置序列。

```

```

def provision_provider(env: dict[str, str], spec: dict, *, arg_path: str = "") -> None:
    """One provider's whole launcher-side provisioning, from its PROVIDER_CREDENTIALS spec:
    key supply (the file's path, never the value), SDK preflight, forwarding the
    address-like vars by value, and echoing which endpoint is in effect."""
    sandbox_key_supply(
        env,
        provider=spec["provider"],
        key_env_vars=spec["key_env_vars"],
        file_env_var=spec["file_env_var"],
        arg_path=arg_path,
        default_path=spec["default_path"],
        provision_hint=spec["provision_hint"],
    )
    preflight_sdk(spec["sdk"], spec["sdk_hint"], spec.get("sdk_min_version"))
    for var in spec["forward"]:
        if value := os.environ.get(var, "").strip():

```

```
        forward_address(env, var, value)
    if spec["target"]:
        var, label, default_desc = spec["target"]
        print(f'{spec["provider"]} {label}: {env.get(var, default_desc)}', flush=True)
```

评论区精华

1. 安全 / 健壮性讨论（已解决）：Claude 机器人在 review 中指出 `harbor_env_vars` 中对 `HARBOR_ENV_TYPE` 的 docker 拒绝检查是区分大小写和空格敏感的。
`harbor_agent_function.py` 在 worker 侧会进行 `.strip().lower()` 规范化，但启动器侧直接比较原始值。这可能导致如 `Docker` 或 `docker` 的输入绕过启动器的明确拒绝，直接进入 worker 并在运行时失败。作者 nblintao 在后续提交中修复了此问题，在 `harbor_env_vars` 开头对 `env_type = args.harbor_env_type.strip().lower()` 进行规范化，并确保后续的空值检查、docker 拒绝和 `PROVIDER_CREDENTIALS` 查找都使用规范化后的值。Claude 在最终 review 中确认此修复已生效，并补充了对应的测试用例 `test_docker_is_refused_with_a_pointer_to_the_agent_server`。
2. 无其他实质性争议：Review 讨论主要集中在这一次安全修复上。Shi-Dong 给出了 "LGTM!" 的批准。PR 作者的后续提交（如修复 `repo_base_dir` 的类型转换、将 `session-server-port` 参数化）属于对实现细节的打磨，未引发进一步讨论。
- `HARBOR_ENV_TYPE` 大小写 / 空格敏感性导致 docker 拒绝检查失效 (correctness): 已修复。作者在 `harbor_env_vars` 开头添加了 `env_type = args.harbor_env_type.strip().lower()`，并确保后续所有检查使用此规范化值。同时添加了测试用例 `test_docker_is_refused_with_a_pointer_to_the_agent_server` 来验证修复。

风险与影响

- 风险：低风险，但有以下需关注点：
 1. 配置漂移风险（已通过共享层缓解）：本 PR 的核心价值之一是通过 `launch_common.py` 将 `agentic` 配置（`--custom-agent-function-path`，`--use-session-server` 等）集中管理。如果未来新增 Harbor 启动器但未使用此共享层，将导致配置不一致。`test_agentic_wiring_is_the_shipped_wiring` 测试通过导入 `miles` 侧的实际函数路径来防止此类拼写错误。
 2. 向后兼容性（OpenEnv 输出格式微调）：OpenEnv 启动器日志中移除了 `openenv:` 前缀（因为 `provision_provider` 使用统一的 `{provider} {label}: {value}` 格式）。这是预期的行为变更，但依赖此日志格式进行监控或解析的工具可能需要调整。
 3. 运行时环境依赖：`harbor_env_vars` 在启动时会执行 `import harbor` 检查，如果 `harbor` 包未正确安装，会立即抛出明确错误，这属于预防性检查，降低了运行时失败的可能性。
- 影响：影响范围：
 1. 开发者体验：显著改善了 Harbor 和 OpenEnv 相关示例的开发者体验。新增的 `run.py` 为 GLM-4.7-Flash 的 Harbor GRPO 训练提供了一键式启动入口。共享的 `launch_common.py` 简化了新启动器的创建，并确保配置一致性。

2. 系统健壮性：提升了沙箱提供商配置的健壮性，通过规范化的环境类型检查和明确的错误信息，减少了因配置错误导致的难以调试的运行时失败。
3. 测试可信度：通过 `test_agentic_wiring_is_the_shipped_wiring` 测试，确保了 GPU 端到端测试（#2876）所验证的配置与正式配方中使用的配置完全相同，增强了对 e2e 测试结果的信任度。
4. 代码复用：将 OpenEnv 启动器中的重复逻辑提取到 `provision_provider`，为未来新增其他沙箱后端的启动器奠定了可复用的基础。 - 风险标记：配置漂移风险（已通过共享层缓解），OpenEnv 日志格式微调

关联脉络

- PR #2806 examples: in-process Harbor agent function: 本 PR 的基础，提供了 `harbor_agent_function`，本 PR 的 `run.py` 启动器为其提供运行环境。
- PR #2804 (tracking issue): 本 PR 的追踪 Issue（在 PR body 中提及），用于跟踪 Harbor in-process 路径的整体进度。
- PR #2876 GPU e2e with real e2b sandboxes: 本 PR 的 `launch_common.py` 中的 `agentic_train_args` 和 `harbor_env_vars` 是 GPU 端到端测试（#2876）的依赖。本 PR 确保了测试与正式配方使用相同的配置。