PR #2738 完整报告

radixark/miles

feat: resolve --hardware from the node in every launcher

合并时间：2026-08-26 04:14

原文链接：http://prhub.com.cn/radixark/miles/pull/2738

执行摘要

- 一句话：--hardware 自动探测节点并推导 GPU 数
- 推荐动作：值得精读：它展示了如何把分散在 19 个脚本里的机器假设收敛为单一事实来源，并用「替换前字面量一致性核对 + 快照不变 + pin 机制」三重手段保证行为不漂移。最值得学习的两点：resolve_hardware() 用 typing.get_args 反射字段 Literal 做显式校验，避免 match 静默穿透；测试用 _fake_torch 在无 GPU runner 上完整覆盖硬件探测矩阵。合并前建议在 B300/GB300/H200 实机各跑一次探测。

功能与动机

PR body 指出每个 launcher 都硬编码了 num_gpus_per_node 与 hardware 默认值，存在两种静默错误路径：

1. 忘记传 --hardware 时采用 recipe 首次编写所在机器的配置（#2735 中 4-GPU H200 smoke test 被路由到 Grace TP8 rollout profile 并卡在 4/8 clients joined）；
2. 传了 --hardware GB300 但未传 --num-gpus-per-node 4 时拓扑仍按 8 卡每节点缩放。目标是把 --hardware 变成机器 profile 的唯一入口，默认探测节点，num_gpus_per_node 从它派生。

实现拆解

实现分五步推进：

1. 统一硬件解析原语（miles/utils/external_utils/command_utils.py）。补全 GENERATION_HARDWARE 的 H200/B200/B300 映射；新增 detect_hardware()，按 torch.version.hip 区分 AMD/NVIDIA，AMD 侧按设备名匹配 MI350X/MI355X，NVIDIA 侧按 compute capability 匹配 (9,0)/(10,0)/(10,3)，用 aarch64 区分 Grace 与 PCIe 同 die 芯片，用 100 GiB 以上 HBM 区分 H100 与 H200，识别失败时断言并提示显式传 --hardware；新增 resolve_hardware(config)，auto 时调用探测，并用 typing.get_args 反射读取 hardware 字段的 Literal，拒绝 launcher 未验证过的硬件，避免 match args.hardware 静默落空。
2. 19 个 launcher 批量接线（scripts/与 scripts/amd/）。所有声明 hardware 的脚本把默认值改为 auto、Literal 加入 auto 前缀、num_gpus_per_node 改为 int | None = None，并在各自 post_init() 最前面统一执行两行推导：先 resolve_hardware 再按 NUM_GPUS_OF_HARDWARE 填充 GPU 数。代表文件包括

scripts/run_glm45_355b_a32b.py、scripts/run_glm47_flash.py、scripts/run_qwen3_5_35b_a3b_mtp.py、scripts/run_qwen3_6_35b_a3b_mtp.py；不声明 hardware 的 33 个脚本保持字面量不动，因为那里的 num_gpus_per_node 往往是刻意的部分节点配置。

3. run_deepseek_v4.py 的 rollout 拓扑键控重构。删除手写的 _is_blackwell() 与 _BLACKWELL_HARDWARE，MXFP8 校验改用 U.GENERATION_HARDWARE[args.hardware] == "Blackwell"；rollout 引擎大小不再用 num_gpus_per_node == 4 当 GB300 的代理信号，而是直接按 hardware 键控，并用 min() 把 Grace 引擎 clamp 到 rollout 池实际大小，避免 4 卡 H200 冒烟测试再次误入 TP8/EP8。
4. 测试配套。tests/fast/utils/test_command_utils.py 新增 _fake_torch 与 TestDetectHardware，用参数化矩阵覆盖 Grace/PCIe 区分、H100/H200 内存区分与 ROCm 设备名路径；_hardware_launchers_accept() 从 launcher_hardware_literals 收集所有 launcher 接受的硬件并校验都在 NUM_GPUS_OF_HARDWARE 中。新增 tests/fast/launch_scripts/test_run_deepseek_v4.py，4 组 overrides 验证 --sglang-tp-size、--sglang-ep-size、--rollout-num-gpus-per-engine 跟随 hardware 与 GPU 数组合。tests/manual/launch_scripts/test_py_launch_scripts.py 引入 _HARDWARE_A_RECORDING_REPRESENTS 与 FROZEN_HARDWARE 固定快照录制硬件，并新增两条 pin 一致性测试；tests/fast/launch_scripts/py_harness.py 的 freeze_environment 支持指定硬件，新增 launcher_hardware_literals()。
5. 行为保持验证。替换前字面量逐一核对：声明 hardware 的 launcher 原先硬编码的 GPU 数与 NUM_GPUS_OF_HARDWARE[默认硬件] 全部一致（13/13），因此默认行为不变；快照测试 182 通过，fast 测试 80 通过，pre-commit 通过；CI 还复跑了 tests/e2e/megatron/test_qwen3_4B_ppo.py 通过。

关键文件：

- miles/utils/external_utils/command_utils.py（模块 启动工具；类别 source；类型 core-logic；符号 detect_hardware, resolve_hardware）：核心改动所在：新增 detect_hardware() 与 resolve_hardware()，补全 GENERATION_HARDWARE 映射，是全部 launcher 硬件解析的唯一事实来源。
- scripts/run_deepseek_v4.py（模块 启动脚本；类别 source；类型 dependency-wiring；符号 ScriptArgs, _is_blackwell）：rollout 拓扑键控重构的发生地：删除 _is_blackwell()，按 hardware 而非 GPU 数选择 TP/EP，是修复 4-GPU H200 挂起的直接受益者。
- tests/fast/utils/test_command_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 _fake_torch, TestDetectHardware, _hardware_launchers_accept, test_grace_is_what_separates_the_superchip_from_the_pcie_part）：用 _fake_torch 在无 GPU 的 CPU runner 上参数化覆盖全部硬件探测分支，并新增数据驱动校验保证 launcher 接受的硬件都已登记。
- tests/fast/launch_scripts/test_run_deepseek_v4.py（模块 启动测试；类别 test；类型 test-coverage；符号 test_the_rollout_profile_follows_the_hardware）：新增回归测试，专门验证 rollout 引擎 TP/EP 大小跟随 hardware 与 num_gpus_per_node 的组合，防止 4-GPU H200 误入 Grace 拓扑。

- tests/manual/launch_scripts/test_py_launch_scripts.py（模块 快照测试；类别 test；类型 test-coverage；符号 test_every_pinned_recording_names_a_launcher_that_accepts_that_hardware, test_a_launcher_the_frozen_default_covers_is_not_pinned_as_well, _HARDWARE_A_RECORDING_REPRESENTS）：快照录制环境的硬件 pin 机制：_HARDWARE_A_RECORDING_REPRESENTS 明确每份 recording 代表哪台机器，并新增两条 pin 一致性测试防止快照漂移。
- scripts/run_glm45_355b_a32b.py（模块 启动脚本；类别 source；类型 core-logic；符号 ScriptArgs, post_init）：代表 19 个 launcher 的批量接线模式：hardware 默认 auto、num_gpus_per_node 改 None、__post_init__统一推导。
- tests/fast/launch_scripts/py_harness.py（模块 测试基座；类别 test；类型 test-coverage；符号 launcher_hardware_literals, freeze_environment）：测试基础设施扩展：freeze_environment 支持指定硬件，新增 launcher_hardware_literals 供数据驱动测试收集各 launcher 的硬件集合。

关键符号：detect_hardware, resolve_hardware, ScriptArgs.post_init, freeze_environment, launcher_hardware_literals, _fake_torch, test_the_rollout_profile_follows_the_hardware, _is_blackwell

关键源码片段

miles/utils/external_utils/command_utils.py

核心改动所在：新增 detect_hardware() 与 resolve_hardware()，补全 GENERATION_HARDWARE 映射，是全部 launcher 硬件解析的唯一事实来源。

```
# miles/utils/external_utils/command_utils.py
# 硬件 -> 每节点 GPU 数的唯一事实来源；launcher 的默认值都从这里派生。
NUM_GPUS_OF_HARDWARE = {
    "H100": 8,
    "H200": 8,
    "B200": 8,
    "B300": 8,
    "GB200": 4,
    "GB300": 4,
    "MI350X": 8,
    "MI355X": 8,
}

# 硬件 -> 架构代次；MXFP8 等精度开关靠它判断是否 Blackwell。
GENERATION_HARDWARE = {
    "H100": "Hopper",
    "H200": "Hopper",
    "B200": "Blackwell",
    "B300": "Blackwell",
    "GB200": "Blackwell",
    "GB300": "Blackwell",
}
```

```
def detect_hardware() -> str:
    """探测当前节点属于哪个 NUM_GPUS_OF_HARDWARE 条目；prepare 阶段无
    GPU，需在用到结果处现场调用。"""
    import torch

    assert torch.cuda.is_available(), "no visible GPU to detect the hardware from, pass --
    hardware explicitly"
    name = torch.cuda.get_device_name()
    if torch.version.hip is not None:
        # AMD 侧没有 compute capability，只能靠设备名匹配 MI350X / MI355X。
        detected = next((hardware for hardware in ("MI350X", "MI355X") if hardware in name),
        None)
    else:
        grace = platform.machine() == "aarch64"
        match torch.cuda.get_device_capability():
            case (9, 0):
                # H100 与 H200 同属 sm90，只有 HBM 容量能区分两者。
                detected = "H200" if torch.cuda.get_device_properties(0).total_memory > 100 *
                1024**3 else "H100"
            case (10, 0):
                # GB200 与 B200 是同一颗 die，宿主 CPU 是唯一区分信号。
                detected = "GB200" if grace else "B200"
            case (10, 3):
                # sm_103 目前是架构推导值，尚未在真实 B300 / GB300 节点实测。
                detected = "GB300" if grace else "B300"
            case _:
                detected = None
    assert detected is not None, f"cannot tell which hardware {name!r} is, pass --hardware
    explicitly"
    return detected

def resolve_hardware(config: ExecuteTrainConfig) -> str:
    # auto 由所在节点决定；显式传入的 --hardware 直接覆盖探测结果。
    if config.hardware == "auto":
        hardware = detect_hardware()
        print(f"detected --hardware {hardware}")
    else:
        hardware = config.hardware
    # 反射读取字段的 Literal，拒绝 launcher 没有验证过 profile 的硬件，
    # 避免后续 match args.hardware 静默穿透造成错误拓扑。
    supported = get_args(config.__dataclass_fields__["hardware"].type)
    assert hardware in supported, f"{type(config).__name__} has no verified profile for {hardware}
    "
    return hardware
```

scripts/run_deepseek_v4.py

rollout 拓扑键控重构的发生地：删除 _is_blackwell()，按 hardware 而非 GPU 数选择 TP/EP，是修复 4-GPU H200 挂起的直接受益者。

```
# scripts/run_deepseek_v4.py
@dataclass
class ScriptArgs(U.ExecuteTrainConfig):
    ...
    # performance configs
    # 每节点 GPU 数不再硬编码为 8；None 表示交给 __post_init__ 按硬件推导。
    num_gpus_per_node: int | None = None
    # auto 表示"由所在节点探测"，显式传入时覆盖探测结果。
    hardware: Literal["auto", "H100", "H200", "B200", "B300", "GB200", "GB300"] = "auto"
    ...

    def __post_init__(self):
        # 在所有拓扑派生之前把 hardware 解析成具体值，
        # 后续逻辑只需读 self.hardware，不再靠 GPU 数反推机器形态。
        self.hardware = U.resolve_hardware(self)
        self.num_gpus_per_node = self.num_gpus_per_node or U.NUM_GPUS_OF_HARDWARE[self.
        hardware]
        if not self.model_org:
            self.model_org = _DEFAULT_MODEL_ORG[self.model_name]
        ...
        # 显式 --num-gpus-per-node 仍然优先，这就是"在 8 卡节点只取 4 卡"的用法。
```

tests/fast/utils/test_command_utils.py

用 _fake_torch 在无 GPU 的 CPU runner 上参数化覆盖全部硬件探测分支，并新增数据驱动校验保证 launcher 接受的硬件都已登记。

```
# tests/fast/utils/test_command_utils.py
def _fake_torch(monkeypatch, *, capability, machine, total_memory=141 * 1024**3, name=
"fake", hip=None):
    # 在无 GPU 的 CPU runner 上伪造 torch 的 CUDA API，让探测逻辑可以被参数化测试。
    monkeypatch.setattr(platform, "machine", lambda: machine)
    monkeypatch.setitem(
        sys.modules,
        "torch",
        SimpleNamespace(
            version=SimpleNamespace(hip=hip),
            cuda=SimpleNamespace(
                is_available=lambda: True,
                get_device_name=lambda: name,
                get_device_capability=lambda: capability,
                get_device_properties=lambda device: SimpleNamespace(total_memory=total_
                memory),
            ),
        ),
    )

class TestDetectHardware:
    @pytest.mark.parametrize(
        ("capability", "machine", "expected"),
```

```
        [
            ((10, 0), "x86_64", "B200"),
            ((10, 0), "aarch64", "GB200"),
            ((10, 3), "x86_64", "B300"),
            ((10, 3), "aarch64", "GB300"),
        ],
    )
    def test_grace_is_what_separates_the_superchip_from_the_pcie_part(
        self, monkeypatch, capability, machine, expected
    ):
        # GB200 / GB300 与 B200 / B300 是同一颗 die，宿主 CPU 是唯一区分信号。
        _fake_torch(monkeypatch, capability=capability, machine=machine)

        assert command_utils.detect_hardware() == expected
```

评论区精华

只有一条实质设计讨论：guapisolo 提出硬件探测与 GPU 数推导逻辑应留在脚本层还是移入 miles launch（如 arguments.py），随后自答并拍板留在脚本层，理由是引擎不应知道太多业务逻辑（原文：We shouldn't let the engine know too many logics）。另有一条 CI 复跑线索：guapisolo 触发 /rerun-test tests/e2e/megatron/test_qwen3_4B_ppo.py，在 stage-c-4-gpu-h200 上 17m15s 通过。

- 硬件探测逻辑放在脚本层还是移入 miles launch (design): guapisolo 自答并拍板：留在脚本层的逻辑更好，不应让引擎知道太多业务逻辑，引擎保持简单。
- CI 复跑 e2e 测试验证回归 (testing): github-actions 报告在 stage-c-4-gpu-h200 上 17m15s 通过。

风险与影响

- 风险： 1) detect_hardware() 的能力表未在真实硬件验证：sm_103（B300/GB300）与 aarch64（Grace）是架构推导值，body 明确建议合并前在 B300/GB300/H200 节点实测，目前仍属残留风险。 2) 无 GPU 的 CPU runner 上 --hardware auto 会触发 no visible GPU 断言，任何直接构造 ScriptArgs 且未 pin 硬件的测试或脚本都会失败；测试用 FROZEN_HARDWARE 与 _HARDWARE_A_RECORDING_REPRESENTS 规避。 3) 19 个 launcher 批量修改，__post_init__接线若有遗漏会产生不一致的默认行为。 4) run_deepseek_v4.py 的 rollout 拓扑改为按 hardware 键控，边界组合行为改变，依赖参数化测试覆盖。
  - 影响：对使用这些 launcher 的开发者：默认行为从「recipe 定型时的机器」变为「所在节点」，跨节点误配事故（如 4/8 clients joined 挂起）被系统性消除，显式参数仍可覆盖，向后兼容。对 CI 与测试：快照录制环境必须显式 pin 硬件，py_harness 的 freeze_environment 扩展为通用设施，新增两条 pin 一致性测试防止快照漂移。对团队规范：新 launcher 将默认采用 hardware Literal（含 auto）加 __post_init__推导的模板，引擎层保持对硬件形态不感知。
  - 风险标记：核心启动路径变更，硬件探测未经真实硬件验证，无 GPU 环境 auto 探测断言失败，批量接线一致性风险

关联脉络

- PR #2735 *detect the node hardware in command_utils*: 本 PR 的前三个提交来自 #2735；*detect_hardware()* 在 #2735 引入，这里在其基础上把 *auto* 推导推广到全部 19 个 launcher。
- PR #2717 *add DeepSeek-V4-Flash-0731 support and mxfp4->fp8 converter*: 其把每个 4-GPU 节点当作 GB300 全模型 profile，导致 4-layer H200 smoke test 走 TP8/EP8 挂起；本 PR 以 *hardware* 键控 rollout 拓扑修复。