PR #7127 完整报告

verl-project/verl

[BREAKING][misc] feat: uv integration

合并时间：2026-08-17 14:10

原文链接：http://prhub.com.cn/verl-project/verl/pull/7127

执行摘要

- 一句话：全项目切换到 uv 统一管理多后端依赖与安装流程
- 推荐动作：值得精读，这是 verl 包管理架构的里程碑级迁移。重点看三处设计：manage_envs.py 的 universal lock + conflicts 方案（一个锁文件、运行期 materialize 一个无冲突组合）、prefetch 与 Docker 镜像"先烘焙缓存、运行时离线 sync"的配合、以及用 check_uv_gpu_only.py 静态分析强制 GPU/NPU 使用路径分离的做法。建议同时关注后续对 trtllm / veomni（cu12.9 世界）恢复解锁的演进，以及 DEVICE:-gpu 默认值对 NPU 用户的隐性要求。

功能与动机

PR body 只有一句话："Support uv with vllm, sglang x fsdp, megatron"，动机需要从实现中还原：此前 verl 依赖 setup.py 动态 optional-dependencies，每个后端一套独立 venv 与 Ray py_executable 切换，缺少统一锁文件，多后端并存时 Docker 镜像无法用一个镜像服务所有 backend。manage_envs.py 的文档串明确说明新设计目标："verl uses one uv.lock for the whole project ... There is no per-backend lockfile and no Ray py_executable switching"，并用 prefetch 让镜像预烘焙 uv 缓存、运行时再离线 sync 出目标组合。同时，check_uv_gpu_only.py 与 attention_utils.py 的注释揭示了两个配套动机：uv.lock 只解析 CUDA backend（vllm-ascend / sglang-ascend / mindspeed 无法来自 uv），因此 NPU 路径必须保持 ambient python；flash-attn 只发布 CUDA wheel，CPU-only 安装（uv 的 cpu extra、单元测试、开发机）需要纯 torch 等价实现保证 attention 路径可运行。

实现拆解

1. 依赖契约重写：pyproject.toml 从 setup.py 动态依赖改为 PEP 621 静态元数据，dynamic 字段移除 optional-dependencies；全部后端拆成 verl-core（公共运行时）+ vllm / sglang / fsdp / megatron / cpu，以及无 torch 依赖的 math / ci / veomni-sft 附加 extra；requires-python 收紧为 >=3.10,<3.13；torch 全家统一 ==2.11.0（cu130），并新增 [tool.uv].conflicts 声明互斥关系，最终提交 5119 行的 uv.lock。flash-attn 与 cupy 通过 flash-attn-cu130torch211、cupy-cu130 子 extra 由父 extra 引用，避免在多个 backend 中重复解析；trtllm（RC sdist 会让 uv lock 失败）与 veomni / nemoautomodel（cu12.9 世界）被注释 defer。
2. 新增 manage_envs.py 环境驱动：作为 uv 的统一入口，提供 lock / sync / run / shell / list / clean / prefetch 命令。核心设计有三点：一是命名 venv，--name 或

VERL_VENV_NAME 生成 .venv-<name>，默认 .venv 维护为指向最近一次 sync 组合的符号链接（_point_default_venv 负责重定向，_detach_symlink_target 防止写穿链接）；二是 VERL_UV_NO_INSTALL 让内部自建包（如内部 ray / wandb）跳过 uv 的精确同步（--no-install-package + --inexact）；三是 prefetch 面向 Docker 镜像构建，先在 uv cache 中烘焙全部后端依赖，镜像运行时再离线 sync 出目标组合。

3. 运行期兼容层：verl/utils/attention_utils.py 为 flash_attn.bert_padding 增加 4 个纯 torch 等价实现（_fallback_index_first_axis 等），在 ImportError 时降级；verl/checkpoint_engine/__init__.py 与 base.py 给每个可选引擎模块记录 ImportError 到 CheckpointEngineRegistry._import_errors，get() 报错时直接列出缺失依赖；verl/workers/engine/megatron/utils.py 的 set_random_seed 用 is_cuda_available or is_npu_available 门控 model_parallel_cuda_manual_seed，避免 CPU-only 下报 "Torch not compiled with CUDA enabled"；megatron_checkpoint_manager.py 的 RNG 存取增加 is_device_available() 保护；verl/single_controller/ray/base.py 在 Ray 首次导入前设置 RAY_ENABLE_UV_RUN_RUNTIME_ENV=0，规避 uv 环境下 Ray 的 uv runtime-env hook 处理 working_dir=None 时崩溃；verl/workers/engine/megatron/transformer_impl.py 的 value head 关闭 tie-embeddings 改为直接写 tf_config / provider_overrides，绕开部分 Megatron 构建对 TransformerConfig(**kwargs) 的兼容问题。
4. 启动脚本与 CI 门控改造：全部 examples/*.sh、tests/special_e2e/*.sh 的 launch 段统一为 ${VERL_USE_UV:-1} != 0 && ${DEVICE:-gpu} = gpu 双条件 gate：GPU 下用 uv run --frozen --all-packages --extra ... 并把 py_executable 传给 Ray，NPU 回落 ambient python。新增 tests/special_sanity/check_uv_gpu_only.py 静态分析所有 shell 脚本，强制 uv 命令必须位于 GPU gate 的 then 分支、NPU 专属脚本（路径或文件名含 ascend / npu / mindspeed 标记）不得出现 uv；新增 docker/Dockerfile.uv.cu130 配套镜像。
5. 测试与文档配套：vllm 相关 update_weights_from_ipc_* 测试用 pytest.importorskip("vllm") 保护并迁移到 vllm.yml 的 venv 中运行；新增 examples/rewards/custome_reward_fn/aime_reward.py，将自定义 AIME data_source 归一化为 aime 后委托 default_compute_score，保证训练（dapo-math）与验证（AIME）走同一 math_dapo 验证器；大量 workflow 与安装文档同步更新。

关键文件：

- manage_envs.py（模块 环境管理；类别 source；类型 dependency-wiring；符号 _resolve_name, _venv_dir, _proj_env, _detach_symlink_target）：本 PR 的核心新增文件：universal-lock uv 环境驱动，定义了 backend/conflict 常量、命名 venv 与 .venv 符号链接管理、sync/prefetch 语义，是整个依赖迁移的入口。
- pyproject.toml（模块 依赖配置；类别 config；类型 configuration）：依赖契约本体：PEP 621 extras 组织全部后端，声明 [tool.uv].conflicts 与 wheelhouse 路由，requires-python 收紧，是 uv.lock 的输入与 BREAKING 的具体载体。
- verl/utils/attention_utils.py（模块 注意力工具；类别 source；类型 dependency-wiring；符号 _fallback_index_first_axis, _fallback_pad_input, _fallback_unpad_input, _fallback_rearrange）：CPU-only 兼容的关键补丁：flash-attn 只有 CUDA wheel，新增纯 torch fallback 保证 uv cpu extra、单测与开发机可运行 attention padding 路径。

- tests/special_sanity/check_uv_gpu_only.py（模块 脚本检查；类别 test；类型 test-coverage；符号 is_npu_only, strip_comment, uv_command_lines, enclosing_ifs）：新增的静态检查测试：保证所有 shell 脚本中的 uv 命令位于 GPU gate 内、NPU 脚本完全不用 uv，是这次大规模脚本改动的护城河。
- verl/checkpoint_engine/base.py（模块 检查点引擎；类别 source；类型 core-logic；符号 record_import_error）：配合可选引擎依赖拆分：新增 record_import_error 与 _import_errors，让缺失传输依赖（cupy / nixl / torch_npu）时能给出明确报错而非笼统的 not registered。
- verl/checkpoint_engine/__init__.py（模块 检查点引擎；类别 source；类型 core-logic）：所有可选 checkpoint 引擎模块的 ImportError 从静默吞掉改为记录到 registry，是 base.py 新机制的唯一数据来源。
- verl/workers/engine/megatron/utils.py（模块 Megatron 适配；类别 source；类型 dependency-wiring）：CPU-only 环境下的 Megatron 种子初始化修复：用 is_cuda_available / is_npu_available 门控 model_parallel_cuda_manual_seed，避免无加速器时报 CUDA 编译错误。
- verl/single_controller/ray/base.py（模块 运行时适配；类别 source；类型 entrypoint）：uv 集成后 Ray 2.47+ 会自动执行 uv runtime-env hook，导致 working_dir=None 崩溃；此处禁用该 hook（RAY_ENABLE_UV_RUN_RUNTIME_ENV=0），是 uv 下多节点运行的关键修复。
- docker/Dockerfile.uv.cu130（模块 部署脚本；类别 infra；类型 infrastructure）：新增的镜像构建文件：配合 manage_envs.py prefetch 在镜像层内烘焙 uv 缓存，使一个镜像离线服务任意 backend 组合。
- uv.lock（模块 锁文件；类别 other；类型 configuration）：全项目统一锁文件（5119 行），是本次迁移的产物与后续所有 sync 的解析依据；体积大、冲突解决成本高。

关键符号：_resolve_name, _venv_dir, _proj_env, _detach_symlink_target, _point_default_venv, _pop_name, _require_uv, _expand, _fallback_index_first_axis, _fallback_pad_input, _fallback_unpad_input, _fallback_rearrange, _get_attention_functions, record_import_error, CheckpointEngineRegistry.get, set_random_seed, check_script, enclosing_ifs, is_gpu_gate, compute_score

关键源码片段

manage_envs.py

本 PR 的核心新增文件：universal-lock uv 环境驱动，定义了 backend/conflict 常量、命名 venv 与 .venv 符号链接管理、sync/prefetch 语义，是整个依赖迁移的入口。

```
# ---- 全局常量：单一 uv.lock 内的所有 backend 组合 ----
# 推理引擎与训练框架各自是 PEP 621 extra，互斥项在 [tool.uv].conflicts 中声明。
# 方案核心：运行时只 materialize 一个无冲突组合进 .venv，所有 Ray worker 共用。
INFERENCE_BACKENDS: list[str] = ["vllm", "sglang"]
TRAINING_BACKENDS: list[str] = ["fsdp", "megatron"]
# DEFERRED（cu12.9 / torch 2.9.1 世界）：veomni、nemoautomodel 待上游支持 torch 2.11 后再启用
```

```
CU129_BACKENDS: list[str] = []
# cpu 是不带 GPU 依赖的 CI / 单测切片
DEV_BACKENDS: list[str] = ["cpu"]
# 无 torch 依赖的附加 extra，可叠加在任意 backend 组合上，CI 通过 sync <backend...> ci 组合
ADDON_EXTRAS: list[str] = ["math", "ci", "veomni-sft"]

# 互斥集合：同一个 .venv 最多装入每个集合中的一个成员（与 pyproject.toml 的 conflicts
保持一致）
CONFLICT_SETS: list[set[str]] = [
    {"vllm", "sglang", "cpu"},
    {"fsdp", "cpu"},
    {"megatron", "cpu"},
]



def _venv_dir(name: str | None) -> Path:
    """解析本次调用对应的项目 venv 目录。

    优先级：用户 --name 指定的 .venv-<name> -> 环境变量 UV_PROJECT_ENVIRONMENT
    -> 默认 .venv。所有子命令都通过 UV_PROJECT_ENVIRONMENT 指向同一目标目录，
    保证 sync / run / shell / clean / list 语义一致。
    """
    if name:
        return VERL_DIR / f".venv-{name}"
    env = os.environ.get("UV_PROJECT_ENVIRONMENT")
    if env:
        return Path(env).expanduser().resolve()
    return DEFAULT_VENV_DIR



def _proj_env(venv_dir: Path) -> dict[str, str | None]:
    """把 uv 的项目环境变量指向目标 venv 目录。"""
    return {"UV_PROJECT_ENVIRONMENT": str(venv_dir)}



def _detach_symlink_target(venv_dir: Path) -> None:
    """若 venv_dir 当前是符号链接则先摘除，让 uv sync 建真实目录。

    只有默认 .venv 会成为指向最近一次 sync 命名环境的符号链接，
    无命名 sync 时先消费掉旧链接，避免写穿到旧命名环境。
    """
    if venv_dir.is_symlink():
        prev = os.readlink(venv_dir)
        venv_dir.unlink()
        print(f"note: {venv_dir} was a symlink (-> {prev}); replacing it with a real env", flush=True)



def _point_default_venv(venv_dir: Path) -> None:
    """让默认 .venv 符号链接指向刚 sync 完成的环境组合。
```

```
    这样 source .venv/bin/activate 与 CI 无需感知命名环境；重复 sync 指向
    同一目标时静默跳过，若 .venv 已是真实目录则告警并保留，绝不删除已物化环境。
    """
    link = DEFAULT_VENV_DIR
    if venv_dir == link or not venv_dir.exists():
        return
    if link.is_symlink():
        try:
            if link.resolve() == venv_dir.resolve():
                return # 已指向此处，保持安静
        except OSError:
            pass
        link.unlink()
    elif link.exists():
        print(
            f"warning: {link} is a real directory; not repointing it at {venv_dir.name}. "
            f"Activate {venv_dir}/bin/activate directly, or `python manage_envs.py clean` "
            "to replace it with a link.",
            file=sys.stderr,
        )
        return
    # 同仓库内的命名环境用相对链接（checkout 移动后依然可用），外部路径用绝对链接
    rel = os.path.relpath(venv_dir, VERL_DIR)
    target = rel if not rel.startswith("..") else str(venv_dir)
    link.symlink_to(target, target_is_directory=True)
    print(f"pointed {link} -> {target} (latest composition)", flush=True)
```

verl/utils/attention_utils.py

CPU-only 兼容的关键补丁：flash-attn 只有 CUDA wheel，新增纯 torch fallback 保证 uv cpu extra、单测与开发机可运行 attention padding 路径。

```
# 纯 torch 实现的 flash_attn.bert_padding 等价函数。
# 动机：flash-attn 只发布 CUDA wheel，CPU-only 安装（单元测试、开发机）没有它，
# 但 verl 的 attention 路径仍要能跑通，只是没有 CUDA 优化。


def _fallback_index_first_axis(tensor: torch.Tensor, indices: torch.Tensor) -> torch.Tensor:
    """等价于 flash_attn.bert_padding.index_first_axis：按 indices 取行。"""
    assert tensor.ndim >= 2
    return tensor[indices]


def _fallback_pad_input(hidden_states: torch.Tensor, indices: torch.Tensor, batch: int, seqlen:
int) -> torch.Tensor:
    """等价于 flash_attn.bert_padding.pad_input：把 unpadded 数据按 indices 放回稠密矩阵。"""
    other_shape = hidden_states.shape[1:]
    output = hidden_states.new_zeros(batch * seqlen, *other_shape)
    output[indices] = hidden_states
    return output.view(batch, seqlen, *other_shape)
```

```
def _fallback_unpad_input(hidden_states: torch.Tensor, attention_mask: torch.Tensor, unused_
mask=None):
    """等价于 flash_attn.bert_padding.unpad_input：返回 unpadded 数据、indices 与 cu_
    seqlens。"""
    all_masks = (attention_mask + unused_mask) if unused_mask is not None else attention_
    mask
    seqlens_in_batch = all_masks.sum(dim=-1, dtype=torch.int32)
    used_seqlens_in_batch = attention_mask.sum(dim=-1, dtype=torch.int32)
    indices = torch.nonzero(all_masks.flatten(), as_tuple=False).flatten()
    cu_seqlens = F.pad(torch.cumsum(seqlens_in_batch, dim=0, dtype=torch.int32), (1, 0))
    return (
        _fallback_index_first_axis(hidden_states.reshape(-1, *hidden_states.shape[2:]), indices),
        indices,
        cu_seqlens,
        seqlens_in_batch.max().item(),
        used_seqlens_in_batch,
    )


def _get_attention_functions() -> tuple[Callable, Callable, Callable, Callable]:
    """按硬件动态加载 attention 工具函数。

    优先级：NPU 实现 -> flash_attn CUDA 实现 -> 纯 torch fallback。
    新增的 try/except ImportError 分支是本次变更的关键：它让没有 flash-attn
    的环境（uv 的 cpu extra）也能 import 本模块并正常执行 pad / unpad。
    """
    from verl.utils.device import is_torch_npu_available

    global _index_first_axis, _pad_input, _rearrange, _unpad_input

    if is_torch_npu_available(check_device=False):
        from verl.utils.npu_flash_attn_utils import index_first_axis, pad_input, rearrange, unpad_
        input
    else:
        try:
            from flash_attn.bert_padding import index_first_axis, pad_input, rearrange, unpad_
            input
        except ImportError:
            # flash-attn 仅有 CUDA wheel，CPU-only 环境在这里降级为纯 torch 实现
            index_first_axis = _fallback_index_first_axis
            pad_input = _fallback_pad_input
            rearrange = _fallback_rearrange
            unpad_input = _fallback_unpad_input

    _index_first_axis, _pad_input, _rearrange, _unpad_input = index_first_axis, pad_input,
    rearrange, unpad_input
    return _index_first_axis, _pad_input, _rearrange, _unpad_input
```

tests/special_sanity/check_uv_gpu_only.py

新增的静态检查测试：保证所有 shell 脚本中的 uv 命令位于 GPU gate 内、NPU 脚本完全不用 uv，是这次大规模脚本改动的护城河。

```
# 背景：uv.lock 只解析 CUDA backend（x86_64 Linux / cp312 / cu130），vllm-ascend /
# sglang-ascend / mindspeed 不可能来自 uv。因此所有 shell 脚本里的 uv 命令必须位于
# GPU 专用 if 分支内，NPU 专属脚本则完全禁止 uv——本文件用静态分析强制这两条规则。

# 匹配 uv 调用：裸命令或嵌入字符串（py_executable="uv -v run ..."）都算
UV_COMMAND = re.compile(r"""(?:^|[\s;&|(="'])uv\s+(?:-\S+\s+)*(?:
run|sync|pip|lock|venv|tool|add|export)\b""")
# GPU gate 的两个必要子句：VERL_USE_UV 开关 + DEVICE 探针
USE_UV_CLAUSE = re.compile(r"\$\{VERL_USE_UV:-1\}")
GPU_CLAUSE = re.compile(r"""\[\s*"\$\{DEVICE:-gpu\}"\s*=\s*gpu\s*\]""")
# NPU 专属目录与文件名标记：这些脚本在任何情况下都不允许调用 uv
NPU_DIRS = ("examples/ascend_extras", "tests/special_npu")
NPU_NAME_MARKERS = ("_npu", "ascend", "mindspeed")


def is_npu_only(rel_path: str) -> bool:
    if any(rel_path == d or rel_path.startswith(d + "/") for d in NPU_DIRS):
        return True
    name = Path(rel_path).name.lower()
    return any(marker in name for marker in NPU_NAME_MARKERS)


def strip_comment(line: str) -> str:
    """丢弃整行注释，行尾注释保留：藏在行尾注释后的 uv 命令同样要被标记。"""
    return "" if line.lstrip().startswith("#") else line


def uv_command_lines(lines: list[str]) -> list[int]:
    return [i for i, line in enumerate(lines) if UV_COMMAND.search(strip_comment(line))]


def enclosing_ifs(lines: list[str], idx: int) -> list[int]:
    """找到包含第 idx 行的 if 行号，由内到外排列。

    只有 then 分支内的行才算作被 gate 保护：位于 else / elif 分支的
    uv 命令同样需要报错。实现为反向扫描并匹配 fi 深度。
    """
    found: list[int] = []
    depth = 0
    in_else = False
    for j in range(idx - 1, -1, -1):
        stripped = strip_comment(lines[j]).strip()
        if stripped == "fi":
            depth += 1
        elif depth == 0 and (stripped == "else" or stripped.startswith("elif ")):
            in_else = True # 该 if 的 then 分支并不包含当前行
```

```
        elif stripped.startswith("if ") and not stripped.endswith("fi"):
            if depth == 0:
                if not in_else:
                    found.append(j)
                in_else = False
            else:
                depth -= 1
    return found


def is_gpu_gate(line: str) -> bool:
    return bool(USE_UV_CLAUSE.search(line) and GPU_CLAUSE.search(line))


def check_script(lines: list[str], display: str) -> list[str]:
    """检查单个脚本：所有 uv 命令必须位于 GPU gate 的 then 分支内。

    py_executable 字符串透传给 Ray 时也算 uv 调用——如果漏检，worker actor
    就会在锁文件并不覆盖的设备上用 uv 启动。
    """
    uv_lines = uv_command_lines(lines)
    if not uv_lines:
        return []

    if is_npu_only(display):
        return [
            f"{display}:{i + 1}: NPU-only script must not invoke uv "
            f"(uv.lock covers CUDA backends only): {lines[i].strip()}"
            for i in uv_lines
        ]

    errors = []
    for i in uv_lines:
        gates = enclosing_ifs(lines, i)
        if any(is_gpu_gate(lines[j]) for j in gates):
            continue # 已被 GPU gate 覆盖，放行
        where = (
            f"gated only by line {gates[0] + 1} (`{lines[gates[0]].strip()}`)"
            if gates
            else "not inside a GPU-gated `then` branch"
        )
        errors.append(
            f"{display}:{i + 1}: uv command {where}; it must run inside "
            f'`if [ "${{VERL_USE_UV:-1}}" != 0 ] && [ "${{DEVICE:-gpu}}" = gpu ]; then`: {lines[i].
            strip()}'
        )
    return errors
```

评论区精华

评审整体很轻量，最终由 Luosuu 一句 “lgtm :)” 批准。两条实质讨论：一是 codex 自动化评论在 examples/sapo_trainer/run_qwen3_8b_fsdp.sh:126 提出 P2 级问题——${DEVICE:-gpu} 默认值为 gpu，NPU 环境未显式导出 DEVICE 时会误入 uv 分支并把 CUDA 的 py_executable 交给 Ray，建议默认关掉 gate 或先探测设备；合入时该模式被保留（check_uv_gpu_only.py 的正则也以它为基准），属于已接受的取舍。二是 wuxibin89 指出 verl/trainer/constants_ppo.py 中新增的 PYTHONPATH 转发可能与外置 PYTHONPATH 冲突，且已在 PR#7313 修复，作者回应 “OK, the rebase process got error.”，最终该改动在提交 7c7e1f9 中被回退，恢复 main 的实现。

- GPU gate 默认值：NPU 可能误入 uv 分支 (design): 合入时沿用双门控模式（check_uv_gpu_only.py 也以该 gate 为基准），依赖 NPU 用户显式设置 DEVICE=npu；该默认值风险在合入后仍存在，属于已知取舍。
- PYTHONPATH 转发与外部设置冲突 (correctness): 该改动在提交 7c7e1f9 中被回退，恢复 main 分支 get_ppo_ray_runtime_env 的实现（RAY_JOB_CONFIG_JSON_ENV_VAR working_dir 处理），避免与 PR#7313 的修复冲突，问题视为已解决。
- 整体审核节奏与最终批准 (other): 大规模 infra PR 的 review 密度较低，主要技术风险由 codex 自动化评论与合入后的 CI 暴露。

风险与影响

- 风险：
  1. 全局 BREAKING 安装流程变更：依赖来源从 setup.py 动态解析变为 pyproject.toml 静态 extras + uv.lock，Python 版本上限收紧到 <3.13，torch 统一锁定 2.11.0（cu130），存量环境（Python 3.10/3.11 或 torch 2.9 等）需要整体迁移。
1. 多后端强 pin 的升级脆弱性：vllm==0.24.0、sglang==0.5.12、transformers==5.5.3/5.3.0、megatron-bridge==0.5.2 等强版本绑定，任一上游发版节奏变化都会阻塞 verl 依赖升级；69 个 commit 中大量是 retry uv.lock / fix cudnn / fix flash_attn，说明锁文件的手工冲突解决成本很高。
2. NPU 误入 uv 分支风险：cpu 与 uv.lock 只覆盖 CUDA backend，NPU 正确性依赖 DEVICE 环境变量被显式设置；codex 指出的 :-gpu 默认值问题在合入后仍存在，未设 DEVICE 的 NPU 启动会静默拿到 CUDA 环境的 py_executable。
3. attention fallback 语义等价性：attention_utils.py 纯 torch 实现与 flash_attn 版本在极端 mask / 空 batch 场景下行为可能不一致，且 flash-attn 缺失时性能明显下降，但该回退只在 CPU-only 安装中触发。
4. 134 文件机械改动回归面：所有 e2e 脚本 launch 段统一改写，单个脚本 gate 笔误会导致 CI 静默使用错误解释器；有 check_uv_gpu_only.py 兜底，但它只做静态文本检查，无法验证运行时解释器选择。 - 影响：对用户 / 开发者：安装与运行方式改变，必须使用 uv（VERL_USE_UV=0 可回退 ambient python，NPU 路径默认不走 uv）；Python 3.12 成为事实上的标准解释器；多后端组合（vllm × fsdp、sglang × megatron）通过 extra 组合在单一 .venv 中实现，取代多 venv 切换。对系统 /CI：所有 GitHub workflow 改为 sync <backend...> ci 从锁文件安装；Docker 镜像通过 prefetch 层烘焙 uv 缓存，一个镜像可

服务任意后端组合。对团队：依赖升级流程变为“改 pyproject.toml + manage_envs.py lock + 提交 uv.lock”，维护成本转移到锁文件的冲突解决与 wheelhouse 路由上。 - 风险标记：全局 BREAKING 安装流程变更，NPU 误入 uv 分支风险（DEVICE 默认值），uv.lock 大文件手工维护成本高，多后端版本强绑定，升级窗口受限，134 文件机械改动回归面大

关联脉络

- PR #7313 （review 中引用的上游修复，标题未提供）：wuxibin89 在 review 评论中明确引用：constants_ppo.py 的 PYTHONPATH 转发与外部设置冲突的问题已在 PR#7313 修复，本 PR 最终回退了该处改动。
- PR #7357 [ci] test: migrate workflows from fully_async/one_step_off_policy to v1 separate_async: 与本 PR 都改动了 tests/special_e2e/run_v1_separate_async.sh，本 PR 在其中加入 uv 双门控 launch 段，两条 CI 迁移线在脚本上交叉。
- PR #7293 [ci] chore: Update ci image: 同一 CI/ 依赖基础设施演进线：本 PR 引入 Dockerfile.uv.cu130 与 prefetch 缓存烘焙，此前的 CI 镜像更新与 NPU 依赖调整为其铺路。