PR #2479 完整报告

radixark/miles

examples: computer-use RL on HUD v6 environments

合并时间：2026-08-13 10:38

原文链接：http://prhub.com.cn/radixark/miles/pull/2479

执行摘要

- 一句话：HUD v6 计算机使用 RL 训练示例
- 推荐动作：值得精读。该 PR 最有价值的设计决策包括：① 用 httpx 传输层而非改 sglang 源码来补齐 token-id 契约，思路可复用甚至上游化；② 拼接前逐轮验证前缀属性，只容忍空白规范化的 resync，避免静默训练损坏；③ 失败按 write-off 而非异常处理，并保证样本字段完整（含 dummy image）以维持 FSDP collective 同步；④ “机制与数字分离”的分层设计，reward shape 通过数据而非代码决定。建议阅读 rollout.py 的 _stitch/_resync_whitespace 和 sglang_compat.py 的传输重写。

功能与动机

PR body 明确说明：Trains a VLM to operate a real GUI through HUD v6 environments. HUD owns the episode — it boots the environment (one Daytona sandbox per episode), runs the agent loop, and grades — while Miles owns training. 同时指出 HUD 自带的 computer-use 工具仅覆盖 Claude/Gemini/OpenAI 原生协议，其 OpenAI-compatible harness 缺少自托管模型（sglang/vLLM）所需的纯 function-calling 屏幕工具；sglang 也缺少 HUD 要求的 return_token_ids 扩展。本 PR 在示例层补齐这些缺口，并验证 reward 塑形、write-off 处理等可学习性设计。

实现拆解

1. 数据生成（make_hud_data.py）：从 HUD v6 env 包的 tasks.py 读取任务模板，生成 Miles 的 prompt jsonl；prompt 列只作样本分组用（真实 prompt 在环境内由模板 first yield 提供），通过 --repeat 复制行来满足 GRPO 组大小，--args-json 支持在不改动 env 包的情况下重调任务参数（如 target_score）。
2. 推理兼容层（sglang_compat.py）：用 httpx 自定义传输拦截 /chat/completions 请求，把 HUD 的 return_token_ids/prompt_token_ids 改写为 sglang 的 return_prompt_token_ids/return_meta_info，并在响应侧从 meta_info.output_token_logprobs 三元组合成 choice.token_ids，实现 sglang 与 HUD 双方零改动桥接。
3. 屏幕工具（computer_tool.py）：继承 HUD 的 RFBTool，暴露 computer 工具的 OpenAI function-calling schema；实现截图降采样与坐标回缩放、按键序列（逐个按键）与组合键（+ 连接）区分、首次键盘输入前的聚焦点击（_ensure_focus），并保证异常不会杀死 episode（返回 tool_err）。

4. 回放拼接与奖励（rollout.py）：作为 --custom-generate-function-path 接入，逐轮收集 HUD Trace 中 token 级 Sample，验证每轮 prompt 前缀与已拼接序列一致后拼接 token，并为 policy 输出置 loss mask 1、脚手架置 0；仅容忍空白规范化差异（_resync_whitespace），其余分歧按 turn 截断；视觉输入从工具返回的截图重建，并与 ids 中 image-token 数量按前缀校验，不一致则响亮失败；所有失败 episode 生成带 dummy image 的 remove_sample 样本，规避 FSDP 纯文本微批导致的 collective 不同步（#2406）。
5. 启动与配置（run_hud2048.py + hud2048_config.yaml）：组装 FSDP + GRPO + sglang 参数，含 preflight 检查（SDK、Daytona 凭证路径）与 smoke/rollout-only 模式；配置文件注释每个 trade-off 参数。测试配套：tests/ 下 28 个离线用例，通过 fakes 覆盖拼接、mask、resync 边界、视觉输入校验、工具 dispatch、sglang 重写和数据行，另外 MODE=smoke 提供 2 个真实 episode 的冒烟验证。

关键文件：

- examples/experimental/hud/rollout.py（模块 回放拼接；类别 source；类型 core-logic；符号 _load_daytona_key, _shared_runtime, _sandbox_gate, _build_agent）：示例的核心拼接逻辑：将 HUD 每轮 token 级 sample 拼接成单条训练序列并计算 loss mask，含空白规范化 resync、视觉输入校验与失败样本兜底。
- examples/experimental/hud/computer_tool.py（模块 屏幕工具；类别 source；类型 core-logic；符号 to_keysym, ChatComputerTool, default_spec, to_params）：为自托管策略补齐 OpenAI function-calling 屏幕工具，处理截图降采样、坐标缩放、按键序列 / 组合键区分与键盘聚焦。
- examples/experimental/hud/sglang_compat.py（模块 兼容层；类别 source；类型 core-logic；符号 _SglangTokenIds, handle_async_request, sglang_client）：httpx 传输层重写，将 HUD 的 return_token_ids 契约翻译为 sglang 的 return_prompt_token_ids/return_meta_info 并合成 token_ids，是示例能拿到 token 级样本的基础。
- examples/experimental/hud/run_hud2048.py（模块 启动器；类别 source；类型 entrypoint；符号 preflight, execute）：HUD 2048 任务的 GRPO/FSDP 启动入口，组装训练参数，提供 preflight 与 smoke 模式，是示例可运行性的关键。
- examples/experimental/hud/make_hud_data.py（模块 数据生成；类别 source；类型 data-contract；符号 load_tasks, task_row, main）：负责把 HUD v6 taskset 转成 Miles 的 prompt jsonl，并通过 --args-json 支持 reward 重调，是 reward shaping 的配置点。
- examples/experimental/hud/agent.py（模块 智能体；类别 source；类型 core-logic；符号 ComputerChatAgent, for_shot_width）：基于 HUD OpenAIChatAgent 的计算机使用 agent，注入系统提示与可配置 shot width 的屏幕工具，连接 rollout 与工具层。
- examples/experimental/hud/tests/test_rollout.py（模块 回放测试；类别 test；类型 test-coverage；符号 test_stitch_two_turns_masks_only_policy_tokens, test_stitch_resyncs_when_the_rerender_canonicalizes_whitespace, test_vision_inputs_mismatch_fails_loudly, test_failed_sample_still_exercises_the_vision_tower）：离线覆盖拼接、mask、resync 边界、视觉输入校验与 write-off 样本形状，是拼接正确性的主要保障。

- *examples/experimental/hud/tests/test_sglang_compat.py*（模块 兼容测试；类别 test；类型 test-coverage；符号 test_shim_rewrites_flags_and_synthesizes_token_ids, test_shim_leaves_other_endpoints_alone）：验证传输层重写逻辑：请求标志改写与 token_ids 合成，防止桥接层回归。
- *examples/experimental/hud/hud2048_config.yaml*（模块 配方配置；类别 config；类型 configuration）：2048 配方配置，每个 trade-off 参数都有注释，体现"机制在代码、数字在配置"的设计原则。
- *examples/experimental/hud/README.md*（模块 示例文档；类别 docs；类型 documentation）：说明示例用法与 env 修改（kiosk、桌面高度），是复现该示例的入口文档。

关键符号：_stitch, _resync_whitespace, _vision_inputs, _failed, ChatComputerTool.execute, ChatComputerTool._ensure_focus, _SglangTokenIds.handle_async_request, sglang_client, ComputerChatAgent.for_shot_width, make_hud_data.main, run_hud2048.preflight, run_hud2048.execute

关键源码片段

examples/experimental/hud/rollout.py

示例的核心拼接逻辑：将 HUD 每轮 token 级 sample 拼接成单条训练序列并计算 loss mask，含空白规范化 resync、视觉输入校验与失败样本兜底。

```
def _stitch(turns, tokenizer=None) -> tuple[list[int], list[int], list[float], int] | None:
    """将每轮 (prompt_ids, output_ids) 拼接成一条训练序列。

    返回 (tokens, loss_mask, rollout_log_probs, response_start):
    mask 和 logprobs 覆盖响应区域（首轮 prompt 之后的所有位置）：
    策略真正输出的 token 为 1/真实 logprob，重渲染进后续 prompt 的脚手架
    token 为 0/0.0。
    """
    tokens: list[int] = []
    loss_mask: list[int] = []
    logprobs: list[float] = []
    response_start = 0
    for k, s in enumerate(turns):
        prompt = list(s.prompt_token_ids)
        output = list(s.output_token_ids)
        if k == 0:
            # 第一轮的 prompt 就是序列前缀
            tokens = prompt
            response_start = len(prompt)
        else:
            # 服务器每轮重新渲染模板，因此第 k+1 轮 prompt 必须以已拼接
            # 的 tokens 为前缀；不一致时尝试仅空白差异的 resync，否则截断。
            if prompt[: len(tokens)] != tokens:
                resynced = (
```

```
                _resync_whitespace(tokens, loss_mask, logprobs, prompt, response_start,
                tokenizer)
                if tokenizer is not None
                else None
            )
            if resynced is None:
                # 真正的历史改写：保留该 turn 之前的精确序列，结束拼接
                break
            tokens, loss_mask, logprobs = resynced
        # 新 prompt 中多出的部分（脚手架 / 截图 tokens）mask 为 0
        delta = prompt[len(tokens):]
        tokens += delta
        loss_mask += [0] * len(delta)
        logprobs += [0.0] * len(delta)
    out_lp = list(s.output_logprobs)
    # 长度不匹配时兜底为 0，防止训练器索引越界
    if len(out_lp) != len(output):
        out_lp = [0.0] * len(output)
    tokens += output
    loss_mask += [1] * len(output) # 策略输出参与 loss
    logprobs += out_lp
if not loss_mask:
    return None # 连一个有效 turn 都没有，直接判失败
return tokens, loss_mask, logprobs, response_start
```

examples/experimental/hud/computer_tool.py

为自托管策略补齐 OpenAI function-calling 屏幕工具，处理截图降采样、坐标缩放、按键序列 / 组合键区分与键盘聚焦。

```
async def execute(self, arguments: dict[str, Any]) -> MCPToolResult:
    action = str(arguments.get("action", ""))
    # 以 INFO 记录每个动作，便于事后诊断 reward 为零的批量运行
    logger.info("[hud act] %s %s", action, {k: v for k, v in arguments.items() if k != "action"})
    try:
        if action == "screenshot":
            pass # 直接返回下方 _observation()
        elif action in ("click", "double_click"):
            if arguments.get("x") is None or arguments.get("y") is None:
                return tool_err("click needs x and y")
            # 模型在降采样帧里给出坐标，需换算回真实屏幕
            await self.click(
                self._to_screen(arguments["x"]),
                self._to_screen(arguments["y"]),
                button=arguments.get("button") or "left",
                count=2 if action == "double_click" else 1,
            )
            self._pointer_used = True
        elif action in ("type", "press"):
            await self._ensure_focus() # 首次键盘输入前确保焦点
```

```
            if action == "type":
                await self.type_text(str(arguments.get("text") or ""))
            else:
                keys = arguments.get("keys") or []
                if isinstance(keys, str):
                    keys = [keys]
                if not keys:
                    return tool_err("press needs keys, e.g. {'keys': ['Left', 'Down']}")
                # keys 数组中的每个元素是独立的按键序列；包含 '+' 的
                # 元素是组合键（hotkey），如 'ctrl+c'。逐次发送。
                for element in keys:
                    chord = [to_keysym(k) for k in str(element).split("+") if k.strip()]
                    if chord:
                        await self.press_keys(chord)
        elif action == "scroll":
            await self.scroll(
                self._to_screen(arguments["x"]) if arguments.get("x") is not None else None,
                self._to_screen(arguments["y"]) if arguments.get("y") is not None else None,
                scroll_y=int(arguments.get("scroll_y") or 0),
            )
        elif action == "wait":
            await self.wait(min(int(arguments.get("ms") or 500), 5000))
        else:
            return tool_err(f"unknown action {action!r}")
        # 等待 UI 稳定后再截图给模型看
        await self.wait(300)
        return await self._observation()
    except Exception as e: # 坏动作不能杀死 episode
        return tool_err(f"{action} failed: {e}")
```

examples/experimental/hud/sglang_compat.py

httpx 传输层重写，将 HUD 的 return_token_ids 契约翻译为 sglang 的 return_prompt_token_ids/return_meta_info 并合成 token_ids，是示例能拿到 token 级样本的基础。

```
class _SglangTokenIds(httpx.AsyncBaseTransport):
    """把 HUD 的 token-id 契约翻译成原生 sglang 字段的传输层。"""

    def __init__(self) -> None:
        self._inner = httpx.AsyncHTTPTransport()

    async def handle_async_request(self, request: httpx.Request) -> httpx.Response:
        is_chat = request.url.path.endswith("/chat/completions")
        if is_chat:
            body = json.loads(request.content or b"{}")
            # HUD 请求里带 return_token_ids，sglang 不认识；
            # 改为 sglang 自己的两个标志。
            if body.pop("return_token_ids", None):
                body["return_prompt_token_ids"] = True
```

```
            body["return_meta_info"] = True
        # HUD 的 KV-continuation 字段，sglang 改为重渲染模板，丢弃之
        body.pop("prompt_token_ids", None)
        content = json.dumps(body).encode()
        headers = {k: v for k, v in request.headers.items() if k.lower() != "content-length"}
        request = httpx.Request(request.method, request.url, headers=headers, content=
        content)

    response = await self._inner.handle_async_request(request)

    if is_chat and response.status_code == 200:
        data = json.loads(await response.aread())
        for choice in data.get("choices", []):
            # sglang 把输出 token 及对数概率放在 meta_info.output_token_logprobs
            # 三元组里，这里合成 HUD 期望的 choice.token_ids。
            triples = (choice.get("meta_info") or {}).get("output_token_logprobs") or []
            if triples and "token_ids" not in choice:
                choice["token_ids"] = [t[1] for t in triples]
        content = json.dumps(data).encode()
        headers = {
            k: v
            for k, v in response.headers.items()
            if k.lower() not in ("content-length", "content-encoding", "transfer-encoding")
        }
        response = httpx.Response(response.status_code, headers=headers, content=content,
        request=request)
    return response
```

评论区精华

该 PR 的 review 记录仅包含 Shi-Dong 的 'LGTM!' 批准，没有实质代码讨论。设计权衡主要在 PR body 中阐述，例如 whitespace resync 只接受“去空白后文本相等且后续精确对齐”的规范化差异、拒绝真正的历史改写；FSDP 纯文本微批的 dummy image 处理；Daytona 凭证以路径传输而非值（Ray 会原样记录 runtime_env 到日志和 job metadata）。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 拼接正确性依赖服务端行为：rollout.py 假设 sglang 每轮重渲染 chat template，且第 k+1 轮 prompt 以第 k 轮 prompt+output 为前缀。若 sglang 或 HUD 升级后改变渲染方式，可能大量截断 episode 或产生静默错位序列（虽有校验，但属强假设）。
  - sglang_compat.py 重写响应的潜在开销：重构响应时移除 content-length/content-encoding/transfer-encoding 头，对超大响应或压缩流可能造成解析变化；示例规模下可接受，但未来若用于大 batch 需关注。

- computer_tool.py 坐标与聚焦风险：坐标缩放依赖 display_width/shot_width 的准确性；_ensure_focus 只点击一次，若聚焦失败或界面将焦点移走，后续按键可能无效（仅日志可见，不报错）。
- 外部 SDK 版本未锁定：hud/daytona SDK 未固定版本，上游 API 变化可能导致示例失效。
- FSDP 纯文本微批陷阱：通过 dummy image 规避 collective 不同步，但这只是本示例的本地保护，通用修复仍需 #2406 落到 trainer。
- 影响：影响范围限于 examples/experimental/hud 目录和 docs/user-guide/environments.md 文档，miles/ 核心代码零改动，不影响现有训练、推理或 dashboard 功能。对用户 / 研究团队而言，提供了一个可直接复现的 computer-use GRPO 训练配方（含启动脚本、数据生成、离线测试），显著降低多轮 VLM 实验门槛；对团队而言，新增示例代码的维护成本主要由上游 SDK 的兼容性决定，但分层清晰（transport / computer use / HUD / recipe），便于后续扩展其他 HUD 任务。
- 风险标记：拼接逻辑对服务端行为敏感，FSDP 纯文本微批需 dummy image，外部 SDK 未锁版本，测试覆盖离线为主

关联脉络

- PR #2369 fix(rollout): normalize rewards per rollout: 本 PR 在 HUD 示例中讨论 reward shape（线性 reach_score 保持 GRPO 组内方差），与 #2369 的奖励归一化同属 rollout 奖励信号正确性主题。
- PR #2368 fix(rollout): group session v2 leaf samples: 本 PR 的 stitch 逻辑同样处理多轮 / 多样本序列拼接，与 session v2 样本分组问题相关。