PR #2123 完整报告

radixark/miles

refactor(session): extract helpers shared by v1 and v2

合并时间：2026-08-06 07:13

原文链接：http://prhub.com.cn/radixark/miles/pull/2123

执行摘要

- 一句话：提取 v1/v2 共用会话原语，字节级测试锁定默认行为
- 推荐动作：值得细读。建议关注三点：一是 test_sessions_v1_pins.py 中“字节级断言 + 状态快照比对（records 与 accumulated_token_ids 前后相等）”的测试设计，这是重构默认路径时保证零漂移的高性价比手段；二是 prepare_chat_request / extract_completion / assert_pretokenized_prefix 的提取边界——凡是“会话无关”的处理收进 core.py，凡是“线性轨迹特有”的策略留在 linear_trajectory.py，这个边界决定了 v2 能否无缝复用；三是遗留的未落实 review 意见（函数内 import）。若要跟进 session v2 开发，本 PR 是必读的基石。

功能与动机

PR body 明确这是一次为共享而做的重构：Session v2（tree-serving）需要 stable token、序列化与 record 构建行为与 v1 保持一致，而这些行为此前内嵌在线性轨迹实现中。原文强调默认路径不可变："Session v2 needs the stable token, serialization, and record-building behavior already exercised by v1, while the default v1 path must keep the same observable behavior." 并声明提取边界：Token 与 record 原语移入 miles/rollout/session/core.py，线性轨迹策略保留在 linear_trajectory.py，同时用 test_sessions_v1_pins.py 的 byte-exact HTTP 特征测试证明行为保持（"adds byte-exact HTTP characterization of the v1 session wire behavior"）。

实现拆解

1. core.py 提取请求准备原语 prepare_chat_request(body, args, tito_tokenizer)：把 SessionCore.chat_completions Phase 1 中约 50 行的请求体处理逻辑整体搬出——JSON 解码、伪流式参数弹出（stream / stream_options）、强制写死 logprobs=True、return_meta_info=True、no_stop_trim=False、按 use_rollout_routing_replay / use_rollout_indexer_replay 注入回放字段、LoRA 模式下注入 lora_path、按请求 chat_template_kwargs 克隆 TITO tokenizer。核心变化是从依赖 self.args 改为显式 args 参数并将 tito_tokenizer 作为入参，函数不再绑定 SessionCore 实例，v2 核心可直接复用。
2. core.py 提取响应提取原语 extract_completion(result)：把 Phase 2 的响应解析与三道校验（meta_info.output_token_logprobs 存在性、assistant content 非 None、logprobs 长度与 completion_tokens 一致）封装为返回四元组 (response, choice,

assistant_message, completion_token_ids) 的纯函数，调用点从约 30 行内联逻辑收敛为一行赋值，v1 / v2 共用同一份校验语义。

3. linear_trajectory.py 提取前缀校验 assert_pretokenized_prefix(...)：update_pretokenized_state 中"旧 token 序列必须是新 checkpoint 前缀（容忍 max_trim_tokens 尾差，超限时定位首个 mismatch 并抛 TokenizationError）"的逻辑被抽成模块级纯函数，v1 的 checkpoint 更新与未来 v2 的 commit 路径共用；LinearTrajectory 的数据结构与回滚策略（MAX_ASSISTANT_ROLLBACK_STEPS=1、generated_checkpoint_message_ends）保持不变。
4. 新增 tests/fast/router/test_sessions_v1_pins.py（193 行）：以字节级断言锁定 v1 默认路径的回滚 / 重试行为，覆盖 deep rollback 的 400 错误文案逐字节一致、回滚后 records 与 accumulated_token_ids 完全不变、失败首轮不落 record、回滚副作用与 append-only 校验的执行顺序、collect_samples 与回滚联动、few-shot 场景只锚定后端生成的首个 assistant checkpoint 等边界；文件头注释明确这些 pin 用于防止 v2 悄悄改变默认路径。
5. 测试基建配套：新增 tests/fast/router/conftest.py 重新导出 test_sessions.py 中的类级 router_env fixture（mock SGLang upstream + session server），避免新测试模块导入时与自身测试签名互相遮蔽；test_sessions.py 的 fixture 补齐 sglang_speculative_algorithm=None 与 save_debug_trajectory_data=None 两个新增 args 字段。验证结果为 CPU 套件 269 个测试全部通过。

关键文件：

- miles/rollout/session/core.py（模块 会话核心；类别 source；类型 core-logic；符号 prepare_chat_request, extract_completion, chat_completions）：默认路径 chat_completions 的重构主体：提取 prepare_chat_request 与 extract_completion 两个 v1 / v2 共享原语，并改写调用点，是本次重构的核心。
- tests/fast/router/test_sessions_v1_pins.py（模块 行为固定；类别 test；类型 test-coverage；符号 TestRollbackPins, _turn, _get, _two_turn_session）：新增 193 行字节级 pin 测试，锁定 v1 默认路径的回滚 / 重试、400 错误文案、状态不变性等可观察行为，是证明重构零漂移的关键证据，也是后续 v2 开发的行为契约。
- miles/rollout/session/linear_trajectory.py（模块 线性轨迹；类别 source；类型 core-logic；符号 assert_pretokenized_prefix, update_pretokenized_state）：把 update_pretokenized_state 中的 pretokenized 前缀校验提取为模块级纯函数 assert_pretokenized_prefix，v1 checkpoint 更新改为委托调用，为 v2 commit 路径复用同一 token 级校验。
- tests/fast/router/conftest.py（模块 测试夹具；类别 test；类型 test-coverage）：新增共享 fixture 文件，重新导出 test_sessions.py 中的 router_env，使新增的 pin 测试模块可以复用而不与自身测试签名互相遮蔽。
- tests/fast/router/test_sessions.py（模块 会话测试；类别 test；类型 test-coverage）：router_env fixture 的 args 补齐 sglang_speculative_algorithm 与 save_debug_trajectory_data 两个新字段，保证 SessionServer 构造参数完整。

关键符号：prepare_chat_request, extract_completion, assert_pretokenized_prefix, update_pretokenized_state, chat_completions

关键源码片段

miles/rollout/session/core.py

默认路径 chat_completions 的重构主体：提取 prepare_chat_request 与 extract_completion 两个 v1 / v2 共享原语，并改写调用点，是本次重构的核心。

```
# miles/rollout/session/core.py
# 从 SessionCore.chat_completions 中提取的第一个共享原语：解析并规范化
# chat 请求体，是 v1 / v2 两条路径共用的"会话无关"前半段。
# 返回 (request_body, client_stream, tito_tokenizer)，其中 tokenizer 可能
# 是按请求 chat_template_kwargs 克隆出的临时实例。
def prepare_chat_request(body: bytes, args, tito_tokenizer) -> tuple:
    try:
        request_body = json.loads(body) if body else {}
    except json.JSONDecodeError as e:
        raise MessageValidationError(f"invalid JSON body: {e}") from e

    # 伪流式：后端必须保持非流式（TITO 需要完整 message 加 meta_info，
    # 且 sglang 在 stream=true 时会拒绝 return_meta_info），所以先弹出
    # 客户端 stream 意图，等渲染客户端响应时再兑现这个语义。
    client_stream = bool(request_body.pop("stream", False))
    request_body.pop("stream_options", None)

    # TITO token 追踪需要 Miles 自有 input_ids 加 SGLang 输出元数据：
    # logprobs=True 填充 meta_info.output_token_logprobs，return_meta_info
    # 把它包进 choice.meta_info。这里硬编码（而非 setdefault）是为防止
    # agent 侧覆盖破坏 token 累积。
    request_body["logprobs"] = True
    request_body["return_meta_info"] = True
    if getattr(args, "use_rollout_routing_replay", False):
        request_body["return_routed_experts"] = True
    if getattr(args, "use_rollout_indexer_replay", False):
        request_body["return_indexer_topk"] = True
    # 必须为 False，stop token 文本才会从 assistant content 中裁剪；
    # token ID 仍然来自下面的 logprobs。
    request_body["no_stop_trim"] = False
    # 服务训练中的 adapter 而非基座权重，否则训练中的 LoRA 永远无法
    # 影响它所生成的轨迹。
    if is_lora_enabled(args):
        request_body["lora_path"] = LORA_ADAPTER_NAME

    # chat_template_kwargs 需按请求克隆 tokenizer，非 dict 直接拒绝。
    request_ctk = request_body.get("chat_template_kwargs")
    if request_ctk is not None and not isinstance(request_ctk, dict):
        raise MessageValidationError("chat_template_kwargs must be an object")
    if request_ctk:
        try:
            tito_tokenizer = tito_tokenizer.clone_with_chat_template_kwargs(request_ctk)
        except ValueError as e:
```

```
            raise MessageValidationError(str(e)) from e
    if tito_tokenizer.chat_template_kwargs:
        request_body["chat_template_kwargs"] = dict(tito_tokenizer.chat_template_kwargs)
    else:
        request_body.pop("chat_template_kwargs", None)
    return request_body, client_stream, tito_tokenizer


# 第二个共享原语：解码并校验后端 chat 响应，v1 / v2 逐字共用。
# 返回 (response, choice, assistant_message, completion_token_ids)；
# 上游 payload 不合法时抛 UpstreamResponseError。
def extract_completion(result: dict) -> tuple:
    response = json.loads(result["response_body"])
    choice = response.get("choices", [{}])[0]

    # TITO 累积依赖 meta_info 中的输出 token logprobs，缺失立即报错。
    meta_info = choice.get("meta_info")
    if not isinstance(meta_info, dict) or "output_token_logprobs" not in meta_info:
        raise UpstreamResponseError(
            "meta_info and output_token_logprobs must be in choice (requires logprobs=True)"
        )
    assistant_message = choice.get("message") or {}
    if assistant_message.get("content") is None:
        # tool call 解析失败时 SGLang 应返回空 content 而非 None。
        raise UpstreamResponseError(
            "assistant message content is None, when tool call parser failed SGLang should still
            return "
            "an empty content rather than None. Please check your modified SGLang version."
        )

    output_token_logprobs = meta_info["output_token_logprobs"]
    completion_tokens = meta_info["completion_tokens"]
    actual_output_logprobs_len = len(output_token_logprobs)
    if actual_output_logprobs_len != completion_tokens:
        # 长度不一致通常意味着 tokenizer batch decode 分支用错，直接拒绝，
        # 避免静默吞掉部分 token 导致轨迹 token 数失真。
        raise UpstreamResponseError(
            "invalid chat completion response: "
            f"len(output_token_logprobs)={actual_output_logprobs_len} "
            f"!= completion_tokens={completion_tokens}. "
            f"Please check whether you use the correct SGLang branch which has fix the tokenizer
            batch decode issue."
        )

    # logprobs 元素是 (logprob, token_id) 二元组，这里只取 token_id。
    completion_token_ids = [t[1] for t in output_token_logprobs]
    return response, choice, assistant_message, completion_token_ids
```

tests/fast/router/test_sessions_v1_pins.py

新增 193 行字节级 pin 测试，锁定 v1 默认路径的回滚 / 重试、400 错误文案、状态不变性等可观察行为，是证明重构零漂移的关键证据，也是后续 v2 开发的行为契约。

```
# tests/fast/router/test_sessions_v1_pins.py
# v1 是默认 session server，这些 pin 以字节级断言（连 400 错误文案里的
# 插值数字都逐一核对）锁定其生产行为。文件头注释明确目的：让 opt-in 的
# v2 树形服务（--use-session-server v2）永远无法悄悄改变默认路径。
class TestRollbackPins:
    U1 = {"role": "user", "content": "What is 1+2?"}
    T1 = {"role": "tool", "content": "tool-result-1", "tool_call_id": "t0"}
    T1_DIFF = {"role": "tool", "content": "tool-result-DIFFERENT", "tool_call_id": "t0"}

    def _two_turn_session(self, env) -> tuple[str, dict, dict]:
        # 构造两轮会话：存储历史为 [U1, a1, T1, a2]，共 2 条 record。
        session_id = _create_session(env.url)
        a1 = self._turn(env.url, session_id, [self.U1])
        a2 = self._turn(env.url, session_id, [self.U1, a1, self.T1])
        assert len(self._get(env.url, session_id)["records"]) == 2
        return session_id, a1, a2

    def test_deep_rollback_400_byte_exact_and_state_unchanged(self, router_env):
        # 深层回滚（需丢弃 2 个 assistant checkpoint）必须被拒绝，400 错误
        # 文案逐字节一致，且会话状态（records 与 accumulated_token_ids）
        # 与请求前完全相等。
        session_id, a1, a2 = self._two_turn_session(router_env)
        t2 = {"role": "tool", "content": "tool-result-2", "tool_call_id": "t1"}
        a3 = self._turn(router_env.url, session_id, [self.U1, a1, self.T1, a2, t2])
        before = self._get(router_env.url, session_id)
        assert len(before["records"]) == 3

        resp = _post_chat(router_env.url, session_id, {"messages": [self.U1, a1, self.T1_DIFF]})

        assert resp.status_code == 400
        # 错误消息中的 discard_count、max_assistant_rollback_steps、
        # 消息条数全部按真实语义插值，任何文案改动都会让测试失败。
        assert resp.json()["error"] == (
            "rollback failed: discard_count=2 exceeds max_assistant_rollback_steps=1 "
            "(stored has 6 messages, request has 3 messages)"
        )
        after = self._get(router_env.url, session_id)
        assert after["records"] == before["records"]
        assert after["metadata"]["accumulated_token_ids"] == before["metadata"]["accumulated_
        token_ids"]

        # 拒绝后继续沿原历史扩展仍是合法请求，证明会话未被破坏。
        t3 = {"role": "tool", "content": "tool-result-3", "tool_call_id": "t2"}
        extend = _post_chat(router_env.url, session_id, {"messages": [self.U1, a1, self.T1, a2, t2,
        a3, t3]})
        assert extend.status_code == 200
```

miles/rollout/session/linear_trajectory.py

把 update_pretokenized_state 中的 pretokenized 前缀校验提取为模块级纯函数 assert_pretokenized_prefix，v1 checkpoint 更新改为委托调用，为 v2 commit 路径复用同一 token 级校验。

```
# miles/rollout/session/linear_trajectory.py
# 从 LinearTrajectory.update_pretokenized_state 提取的纯函数：校验旧 token
# 序列必须是新 checkpoint 的前缀（容忍 max_trim_tokens 尾差）。
# v1 的 checkpoint 更新与将来 v2 的 commit 路径共用同一份语义。
def assert_pretokenized_prefix(
    prev: list[int],
    all_token_ids: list[int],
    *,
    max_trim_tokens: int,
    request_messages: list[dict[str, Any]],
    assistant_message: dict[str, Any],
) -> None:
    # 空历史无需校验（首轮请求）。
    if not prev:
        return
    # 允许最多 max_trim_tokens 个尾部 token 被裁剪（例如模板改版），
    # 只比较前 (len(prev) - max_trim_tokens) 个 token。
    check_len = len(prev) - max_trim_tokens
    if check_len > 0 and all_token_ids[:check_len] != prev[:check_len]:
        # 定位首个 mismatch 下标，用于生成可读的诊断信息。
        first_mismatch = next(
            (i for i, (a, b) in enumerate(zip(all_token_ids[:check_len], prev[:check_len], strict=True))
            if a != b),
            min(len(all_token_ids), check_len),
        )
        raise TokenizationError(
            f"pretokenized prefix mismatch: "
            f"stored {len(prev)} tokens (checking first {check_len}, "
            f"allowing {max_trim_tokens} trailing) are not a prefix of "
            f"prompt_token_ids + completion_token_ids "
            f"({len(all_token_ids)} tokens), "
            f"first mismatch at index {first_mismatch}, "
            f"matched {first_mismatch}/{check_len} prefix tokens\n"
            f"request_messages={request_messages}\n"
            f"assistant_message={assistant_message}"
        )


# 改造后的调用点：checkpoint 更新只负责追加 token、消息与 checkpoint 边界，
# 前缀一致性校验全部委托给上面的共享纯函数。
def update_pretokenized_state(
    self,
    request_messages: list[dict[str, Any]],
```

```
    assistant_message: dict[str, Any],
    prompt_token_ids: list[int],
    completion_token_ids: list[int],
    max_trim_tokens: int,
) -> None:
    all_token_ids = prompt_token_ids + completion_token_ids
    assert_pretokenized_prefix(
        self.token_ids,
        all_token_ids,
        max_trim_tokens=max_trim_tokens,
        request_messages=request_messages,
        assistant_message=assistant_message,
    )
    self.messages = list(request_messages) + [assistant_message]
    self.trajectory_token_ids.append(all_token_ids)
    self.generated_checkpoint_message_ends.append(len(request_messages) + 1)
    self.num_assistant = len(self.generated_checkpoint_message_ends)
```

评论区精华

唯一的 review 评论来自 Shi-Dong，指向新增 pin 测试文件第 146 行：test_collect_samples_after_rollback_single_sample 内的函数级 import（decode_samples_and_merge_input_sample、Sample）被要求提升到脚本开头（"please hoist this import to the beginning of the script"）。该评论不阻塞，Shi-Dong 随后给出 APPROVED（"LGTM!"）。值得注意的是最终合入代码中该 import 仍保留在函数体内，意见未落实，讨论中未说明原因，后续提交可顺手清理。

- 函数级 import 应提升到脚本开头 (style): 未落实：最终合入代码仍保留函数内 import，讨论中未说明原因；该意见不阻塞，Shi-Dong 随后 APPROVED（"LGTM!"）。

风险与影响

- 风险：
  1. 默认路径重构依赖新测试兜底：chat_completions 是 session server 的核心入口（default v1 路径），重构后行为保障完全依赖新增 pin 测试；若 prepare_chat_request 或 extract_completion 与原内联逻辑存在细微出入（例如 stream_options 弹出顺序、tokenizer 克隆后 chat_template_kwargs 回填），线上轨迹采集会静默漂移。
  2. getattr 容错语义变化：原代码直接访问 self.args.use_rollout_routing_replay，属性缺失时抛 AttributeError 暴露配置缺口；新代码用 getattr(args, ..., False) 默认兜底，更宽容但也会掩盖配置项拼写错误类问题。
  3. 共享契约仅单侧验证：三个共享原语目前只有 v1 一个调用方，v2 尚未合入，提取边界（哪些属于“会话无关”）未被第二方消费验证，v2 落地时可能被迫回改签名。
  4. 测试夹具参数补齐：router_env 的 SimpleNamespace 新增 sglang_speculative_algorithm=None 与 save_debug_trajectory_data=None，若 SessionServer 内部对这两个参数存在分支，测试环境与生产配置可能不完全一致；None 表示未启用，风险较低。

5. pin 测试覆盖盲区：pin 覆盖回滚 / 重试 / 错误文案 /samples 联动，但未逐项验证 no_stop_trim、logprobs 强制注入等请求规范化分支的等效性（由原 test_sessions.py 兜底）。 - 影响：对用户与线上：本 PR 是行为保持型重构，v1 会话服务的 HTTP 可观察行为（400 错误文案、rollback 语义、record 结构）被 pin 测试锁定为契约，重构期间不应产生任何线上差异。对系统：session 模块开始形成“共享原语层”，tree-serving v2 将直接复用这三个原语，显著降低 v2 与 v1 行为分叉的风险。对团队：建立了一类可复用的“行为固定测试”（byte-exact pin）模板，后续任何 v1 语义调整都会在 CI 中显式暴露；同时明确了 v1 / v2 的代码边界（共享原语在 core.py，v1 策略在 linear_trajectory.py）。 - 风险标记：核心路径重构，行为锁定依赖新测试，review 意见未落实，共享契约待 v2 验证

关联脉络

- PR #2075 session: apply the trained LoRA adapter to session-server rollouts: 修改同一文件 miles/rollout/session/core.py，本 PR 提取出的 prepare_chat_request 中的 lora_path 注入逻辑正是 2075 引入的，属于同一 session 功能线的直接延续。
- PR #2202 fix(tito): prevent DeepSeek V4 system-tail mismatch: TITO tokenizer 链路相关，与本 PR 提取原语中的 tokenizer 克隆与 stable token 语义同属 session/chat template 基础设施。