# PR #27698 完整报告

sgl-project/sglang

[diffusion] refactor realtime control state and adapters

合并时间：2026-06-10 12:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27698

## 执行摘要

- 一句话：重构 diffusion 实时控制状态与适配器，统一信号队列
- 推荐动作：值得精读，尤其是 control_signals.py 中脚本 / 状态两种控制模式的抽象设计和 states/ 子包的组织方式。它为后续新增实时模型（如更多 diffusion 模型）提供了清晰的基础设施。建议在合并后运行完整的 diffusion 测试套件并观察 CI 结果。

## 功能与动机

原有 ConditionEvent 与具体模型耦合紧密，LingBot 和 SANA-WM 两个适配器存在大量重复代码。通过提取模型无关的控制信号原语和状态模块，使新模型接入实时推理时只需实现少数接口，降低开发成本和出错概率。PR body 明确指出 'Replace ConditionEvent with model-agnostic realtime control signal queues' 和 'Share LingBot and SANA-WM adapter request prep through BaseRealtimeModelAdapter'。

## 实现拆解

1. 创建新的控制信号模块 control_signals.py，定义 ControlSignal、ControlStateTransition、ControlSignalSamplingParams、ParsedControlEventPayload 等数据结构，以及 ControlSignalQueue、ControlScriptQueue、ControlStateQueue 三个队列类，支持脚本模式（有限时间线）和状态模式（电平触发流）。
2. 将原先散布在 condition_events.py 和 camera_controls.py 中的摄像头控制逻辑抽取到 states/camera_control.py 中的 RealtimeCameraControlState 类，使用新队列；将 causal DiT 状态迁移到 states/causal.py 中的 RealtimeCausalDiTState。
3. 在 realtime_adapter.py 中提取 BaseRealtimeModelAdapter，封装 save_realtime_first_frame、build_realtime_sampling_params 等公共函数；将具体适配器（LingBotWorldRealtimeAdapter、SanaWMRealtimeAdapter）的重复代码下沉到基类。
4. 将实时 chunk 的 latent preparation 逻辑从原 latent_preparation.py 迁移到 stages/realtime/latent_preparation.py，新增 RealtimeChunkLatentPreparationStage。
5. 删除被替换的旧模块（condition_events.py、camera_controls.py、causal_state.py），更新所有模块的导入引用。
6. 配套调整测试文件（至少 7 个测试文件有改动），确保新逻辑的正确性。

关键文件：

- python/sglang/multimodal_gen/runtime/realtime/control_signals.py（模块 控制信号；类别 source；类型 core-logic；符号 ControlSignal, ControlStateTransition, ControlSignalSamplingParams, ParsedControlEventPayload）：新增的核心模块，定义了所有模型无关的控制信号数据结构和队列（ControlSignalQueue, ControlScriptQueue, ControlStateQueue），替代原有的 ConditionEvent。
- python/sglang/multimodal_gen/runtime/realtime/condition_events.py（模块 条件事件；类别 source；类型 deletion；符号 ControlSignal, ControlStateTransition, ConditionEvent, iter_signals）：原有的条件事件队列实现，被替换为 control_signals.py。该文件删除。
- python/sglang/multimodal_gen/runtime/realtime/camera_controls.py（模块 相机控制；类别 source；类型 deletion；符号 _identity_actions, RealtimeCameraControlState, init, clear）：原有的实时摄像头控制状态，被迁移至 states/camera_control.py。
- python/sglang/multimodal_gen/runtime/realtime/states/camera_control.py（模块 相机状态；类别 source；类型 core-logic；符号 _identity_actions, RealtimeCameraControlState, init, clear）：重新实现的相机控制状态，基于新的 control_signals 模块，内部使用 ControlStateQueue 和 ControlScriptQueue。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/realtime/latent_preparation.py（模块 潜空间准备；类别 source；类型 core-logic；符号 RealtimeChunkLatentPreparationStage, get_forward_latent_num_frames, get_latent_preparation_spec, should_scale_initial_noise）：新增 RealtimeChunkLatentPreparationStage，将实时 chunk 的 latent 准备逻辑从原 latent_preparation.py 中拆分出来。
- python/sglang/multimodal_gen/runtime/entrypoints/openai/realtime/realtime_adapter.py（模块 适配器基类；类别 source；类型 dependency-wiring；符号 RealtimeModelAdapter, save_realtime_first_frame, build_realtime_sampling_params, create_state）：提取 BaseRealtimeModelAdapter 基类，封装首帧保存、采样参数构造、输出发送等公共操作；简化具体适配器。
- python/sglang/multimodal_gen/runtime/entrypoints/openai/realtime/adapters/lingbot_world_realtime_adapter.py（模块 LingBot 适配器；类别 source；类型 core-logic；符号 receive_camera_event_payload, receive_camera_control_event_payload, LingBotWorldRealtimeAdapter, init）：适配新控制信号和基类，大量代码被移除并替换为对基类的调用。
- python/sglang/multimodal_gen/runtime/entrypoints/openai/realtime/adapters/sana_wm_realtime_adapter.py（模块 SANA-WM 适配器；类别 source；类型 dependency-wiring；符号 receive_camera_event_payload, receive_camera_control_event_payload, SanaWMRealtimeAdapter, init）：同上，SANA-WM 适配器同样简化。

关键符号：parse_control_event_payload, _control_state_transitions_from_event_payload, save_realtime_first_frame, build_realtime_sampling_params, BaseRealtimeModelAdapter.wait_for_next_chunk, RealtimeCameraControlState.receive_camera_control_event_payload, RealtimeCameraControlState.sample_camera_actions, ControlScriptQueue.push_script,

ControlStateQueue.push_many, RealtimeChunkLatentPreparationStage.verify_input

## 关键源码片段

### python/sglang/multimodal_gen/runtime/realtime/control_signals.py

新增的核心模块，定义了所有模型无关的控制信号数据结构和队列（ControlSignalQueue, ControlScriptQueue, ControlStateQueue），替代原有的 ConditionEvent。

```
# 关键函数：将外部控制事件解析为脚本或状态模式的内部表示
# 模型无关，通过回调参数支持不同模型的校验 / 归一化

def parse_control_event_payload(
    payload: Any,
    *,
    event_id: int | None,
    kind: str,
    normalize_state_payload: ControlStatePayloadNormalizer,
    validate_script_payload: ControlScriptPayloadValidator,
) -> ParsedControlEventPayload:
    """解析外部控制事件（来自 endpoint），区分脚本模式和状态模式"""
    if isinstance(payload, dict) and payload.get("mode") == "state":
        # 状态模式：将 transitions 结构化为 ControlStateTransition 列表
        return ParsedControlEventPayload(
            mode="state",
            payload=_control_state_transitions_from_event_payload(
                payload, event_id=event_id, kind=kind,
                normalize_state_payload=normalize_state_payload,
            ),
        )
    # 脚本模式：由调用方提供的验证函数处理
    return ParsedControlEventPayload(
        mode="script",
        payload=validate_script_payload(payload),
    )

def _control_state_transitions_from_event_payload(
    payload: dict[str, Any],
    *,
    event_id: int | None,
    kind: str,
    normalize_state_payload: ControlStatePayloadNormalizer,
) -> list[ControlStateTransition]:
    """展开状态模式下的 transitions 列表"""
    transitions = payload.get("transitions")
    if not isinstance(transitions, list):
        raise ValueError(f"{kind} state payload requires transitions")
    result = []
    for transition in transitions:
        if not isinstance(transition, dict):
```

```
            raise ValueError(f"{kind} transition must be a map")
        actions = transition.get("actions")
        if not isinstance(actions, list):
            raise ValueError(f"{kind} transition actions must be a list")
        timestamp_ms = transition.get("client_ts_ms")
        if timestamp_ms is not None:
            timestamp_ms = int(timestamp_ms)
        result.append(
            ControlStateTransition(
                payload=normalize_state_payload(actions),
                seq_id=event_id,
                timestamp_ms=timestamp_ms,
            )
        )
    return result
```

## python/sglang/multimodal_gen/runtime/realtime/states/camera_control.py

重新实现的相机控制状态，基于新的 control_signals 模块，内部使用 ControlStateQueue 和 ControlScriptQueue。

```
class RealtimeCameraControlState:
    """Session-local camera-control buffer, using ControlScriptQueue and ControlStateQueue."""

    def __init__(
        self,
        *,
        min_pulse_items: int = 1,
        script_maxlen: int = 512,
        max_transitions: int = 512,
        normalize_state_actions: CameraActionNormalizer = _identity_actions,
    ) -> None:
        # 状态模式队列：保持最新状态，支持脉冲
        self.camera_state_queue = ControlStateQueue(
            default_item=[],
            min_pulse_items=min_pulse_items,
            max_transitions=max_transitions,
        )
        # 脚本模式队列：消费预定义时间线，优先级高于状态模式
        self.camera_script_queue = ControlScriptQueue(
            "camera_actions",
            max_events=script_maxlen,
            default_item=[],
        )
        self.latest_sampled_event_id: int | None = None
        self._normalize_state_actions = normalize_state_actions

    def receive_camera_control_event_payload(
        self,
        payload: Any,
```

```
        *,
        event_id: int | None,
        validate_camera_actions: CameraActionValidator,
    ) -> str:
        """解析外部相机事件并安装为脚本或状态。"""
        parsed = parse_control_event_payload(
            payload,
            event_id=event_id,
            kind="camera_actions",
            normalize_state_payload=self._normalize_state_actions,
            validate_script_payload=validate_camera_actions,
        )
        if parsed.mode == "state":
            transitions = parsed.payload
            self.receive_camera_state_transitions(transitions)
            return f"kind=camera_actions, mode=state, transitions={len(transitions)}"
        camera_actions = parsed.payload
        self.receive_camera_action_script(camera_actions, event_id=event_id)
        return f"kind=camera_actions, mode=script, frames={len(camera_actions)}"
```

## python/sglang/multimodal_gen/runtime/entrypoints/openai/realtime/realtime_adapter.py

提取 BaseRealtimeModelAdapter 基类，封装首帧保存、采样参数构造、输出发送等公共操作；简化具体适配器。

```
async def save_realtime_first_frame(
    session: GenerateSession,
    request: RealtimeVideoGenerationsRequest,
    *,
    required_error: str | None = None,
    cache_remote_urls: bool = False,
) -> None:
    """保存首帧图像到本地临时目录并更新 request.first_frame 路径。"""
    first_frame = request.first_frame
    if first_frame is None:
        if required_error is not None:
            raise ValueError(required_error)
        return

    server_args = get_global_server_args()
    if server_args.input_save_path is not None:
        uploads_dir = server_args.input_save_path
        os.makedirs(uploads_dir, exist_ok=True)
    else:
        if session.input_temp_dir is None:
            session.input_temp_dir = tempfile.mkdtemp(prefix="sglang_input_")
        uploads_dir = session.input_temp_dir

    if cache_remote_urls and isinstance(first_frame, str) and first_frame.lower().startswith(("http:
```

```
//", "https://")):
    # 远程 URL 缓存到本地
    suffix = os.path.splitext(first_frame.split("?", 1)[0])[1]
    digest = hashlib.sha256(first_frame.encode("utf-8")).hexdigest()[:16]
    target_path = os.path.join(uploads_dir, f"realtime_ref_{digest}{suffix}")
    if os.path.exists(target_path):
        request.first_frame = target_path
        return
else:
    target_path = os.path.join(uploads_dir, f"{session.id}_first_frame")

request.first_frame = await save_image_to_path(first_frame, target_path)
```

## 评论区精华

没有实质性的 review 讨论。PR 由作者 mickqian 直接合并，仅在 issue 中触发了 CI 重跑。该 PR 标记为依赖 #27697。

- 暂无高价值评论线程

## 风险与影响

- 风险：主要风险在于重构幅度大（51 个文件、±3k 行），核心的控制流和数据结构完全替换。尤其是脚本 / 状态模式互斥逻辑、采样行为（pulse、repeat_last）的改动，可能引入边缘情况错误。虽然测试文件同时改动，但未确保全部覆盖实时推理流程的所有分支（如空 chunk、默认值处理、状态模式下的 press/release 时序）。另外，删除旧模块后所有外部引用必须同步更新，存在遗漏风险。
- 影响：影响范围限定在 sglang/multimodal_gen 内的实时推理路径（realtime）。对外暴露的 API（HTTP/WebSocket endpoint）无变化，但内部模块重新组织。开发者在升级后需要导入新的 control_signals 模块代替原有的 condition_events 模块，且自定义适配器需继承 BaseRealtimeModelAdapter。整体影响面中等偏下。
- 风险标记：大规模重构（51 files），核心控制流替换，依赖前置重构 #27697，测试覆盖需要确认

## 关联脉络

- PR #27697 [diffusion] refactor realtime and model-specific stage modules: 依赖的前置重构，提供 stage 模块基础
- PR #27699 [diffusion] refactor: rename realtime timer module: 本 PR 引用的修复，重命名定时器模块以配合控制状态重构