PR #32683 完整报告

sgl-project/sglang

[Diffusion] Return scheduler sigmas snapshot in rollout dit_trajectory

合并时间：2026-07-31 15:29

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32683

执行摘要

- 一句话：diffusion rollout 返回调度器 sigmas 快照
- 推荐动作：建议精读本 PR 的语义设计：它给出了一个很好的范例——服务端把调度器内部的精确元数据直接外抛，而不是让下游从有限信息反推。尤其值得关注 _slice_rollout_trajectory_for_sample 中“元数据不做逐样本切片”的处理方式，以及作者后续将 timesteps 也切换到 scheduler.timesteps 的计划；若该计划落地，会彻底统一扩散 rollout 轨迹的时间轴语义。

功能与动机

PR body 明确指出：训练侧消费者目前从轨迹 timesteps 重算 sigmas，即 timesteps / num_train_timesteps，这会经历 sigma * 1000 / 1000 的 ULP 往返漂移，且在 bf16 除法时放大为可观测的 log-prob 差异；而调度器本身已经在 set_timesteps 中按请求实际 num_inference_steps 重建了精确的 self.sigmas，因此直接返回调度器持有的精确值，避免训练端二次计算引入数值偏差。

实现拆解

本 PR 是一条完整的数据通路增强，按以下 4 步实现：

1. 数据结构扩展：在 python/sglang/multimodal_gen/runtime/post_training/rl_dataclasses.py 的 RolloutDitTrajectory 中新增 sigmas: torch.Tensor | None = None 字段，注释明确其为 [T+1] 的 scheduler.sigmas 快照（post-shift，含终点 0）。默认值 None 保证旧构造不受影响，向后兼容。
2. 采集端快照：在 python/sglang/multimodal_gen/runtime/post_training/rollout_denoising_mixin.py 的 _maybe_finalize_denoising_env_collection 中，当 rollout_return_dit_trajectory 成立时，在构造 RolloutDitTrajectory 时追加 sigmas=batch.scheduler.sigmas.detach().cpu().clone()。选择 clone() 是为了避免调度器后续内部复用 tensor 时污染已返回数据。
3. 入口透传与序列化：在 python/sglang/multimodal_gen/runtime/entrypoints/post_training/rollout_api.py 中做三处配套改动：_slice_rollout_trajectory_for_sample 把 dit.sigmas 原样透传（不做 _extract_single_sample_tensor 切片，因为它是 batch 级共享的调度元数据，且可避免一维长度恰好等于 batch_size 时被误切）；_serialize_rollout_trajectory 新增 serialized_dit_sigmas 参数并写入响应字典；_build_response 在序列化前用

_maybe_serialize 统一处理 sigmas。

4. 测试配套：在 python/sglang/multimodal_gen/test/unit/test_rollout_api.py 中扩展两个用例：test_with_denoising_env 断言序列化结果中包含 sigmas 键且为 __tensor__；test_batch_dit_timesteps_on_each_row_one_serialize 断言批量响应中每行 sigmas 形状为 (T+1,) 且逐行相等，验证 batch 级共享语义。

关键文件：

- python/sglang/multimodal_gen/runtime/entrypoints/post_training/rollout_api.py（模块 接口层；类别 source；类型 entrypoint；符号 _slice_rollout_trajectory_for_sample, _serialize_rollout_trajectory, _build_response）：Rollout HTTP 入口：负责逐样本透传 sigmas、序列化并写入响应字典，是消费者最终拿到该字段的出口。
- python/sglang/multimodal_gen/runtime/post_training/rl_dataclasses.py（模块 数据结构；类别 source；类型 core-logic；符号 RolloutDitTrajectory）：定义 RolloutDitTrajectory 数据结构，新增 sigmas 字段是整条数据通路的源头。
- python/sglang/multimodal_gen/runtime/post_training/rollout_denoising_mixin.py（模块 去噪流程；类别 source；类型 core-logic；符号 _maybe_finalize_denoising_env_collection）：轨迹 finalize 的关键采集点，从这里直接快照 batch.scheduler.sigmas，是本 PR 数据来源。
- python/sglang/multimodal_gen/test/unit/test_rollout_api.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_with_denoising_env, test_batch_dit_timesteps_on_each_row_one_serialize）：覆盖 sigmas 的序列化存在性、批量响应共享性与形状一致性，验证入口层语义。

关键符号：_maybe_finalize_denoising_env_collection, _slice_rollout_trajectory_for_sample, _serialize_rollout_trajectory, _build_response, RolloutDitTrajectory

关键源码片段

python/sglang/multimodal_gen/runtime/entrypoints/post_training/rollout_api.py

Rollout HTTP 入口：负责逐样本透传 sigmas、序列化并写入响应字典，是消费者最终拿到该字段的出口。

```
# 逐样本抽取时：latents 是 [B, T+1, ...] 的逐样本轨迹，需要切片；
# timesteps 与 sigmas 是 batch 级共享的调度元数据，直接透传不切片。
# 如果对 sigmas 也走 _extract_single_sample_tensor，
# 当一维 schedule 的长度恰好等于 batch_size 时会被误切，因此必须旁路。
dit_trajectory = None
if rtd.dit_trajectory:
    dit = rtd.dit_trajectory
    dit_trajectory = RolloutDitTrajectory(
        latents=_extract_single_sample_tensor(dit.latents, sample_idx, batch_size),
        timesteps=dit.timesteps,
        sigmas=dit.sigmas,
```

```
    )


# 序列化阶段：sigmas 与 timesteps 走同一路径，由调用方先 _maybe_serialize。
# 这样批量响应中每个样本共享同一份已序列化的调度元数据，避免重复编码。
def _serialize_rollout_trajectory(
    rtd: RolloutTrajectoryData | None,
    *,
    serialized_dit_timesteps: dict | None = None,
    serialized_dit_sigmas: dict | None = None,
) -> tuple[dict | None, dict | None, dict | None, dict | None]:
    """按顺序返回 rollout_log_probs、debug_tensors、denoising_env、dit_trajectory。"""
    if rtd is None:
        return None, None, None, None

    serialized_dit_trajectory = None
    if rtd.dit_trajectory:
        dit = rtd.dit_trajectory
        serialized_dit_trajectory = {
            "latents": _maybe_serialize(dit.latents) if dit.latents is not None else None,
            "timesteps": serialized_dit_timesteps,
            "sigmas": serialized_dit_sigmas,
        }
    return (
        serialized_log_probs,
        serialized_debug_tensors,
        serialized_denoising_env,
        serialized_dit_trajectory,
    )
```

python/sglang/multimodal_gen/runtime/post_training/rl_dataclasses.py

定义 RolloutDitTrajectory 数据结构，新增 sigmas 字段是整条数据通路的源头。

```
@dataclass
class RolloutDitTrajectory:
    """扩散模型 rollout 去噪轨迹的计时与噪声水平元数据。"""
    # [B, T+1, ...]: 每步带噪 latents x_{t_0..t_{T-1}}，末尾是最后一次
    # scheduler.step 输出的最终去噪 latents x_{t_T}。
    latents: torch.Tensor | None = None
    # [T]: 每步去噪使用的 timestep，会被 rollout_return_step_indices 过滤。
    timesteps: torch.Tensor | None = None
    # [T+1]: scheduler.sigmas 快照（post-shift，含终点 0）。
    # 注意它是调度元数据而非逐样本数据：完整保留，不做逐样本切片，
    # 也不随 rollout_return_step_indices 过滤，消费方需按绝对 step index 对齐。
    sigmas: torch.Tensor | None = None
```

python/sglang/multimodal_gen/runtime/post_training/rollout_denoising_mixin.py

轨迹 finalize 的关键采集点，从这里直接快照 batch.scheduler.sigmas，是本 PR 数据来源。

```
# 去噪过程结束、轨迹落盘前，把调度器持有的精确 sigmas 一并快照。
# detach 防止梯度反传进调度器；clone 避免 scheduler 内部复用同一 tensor
# 导致已返回数据被后续生成破坏。
if step_latents and batch.rollout_return_dit_trajectory:
    step_latents_tensor = torch.stack(step_latents, dim=1)
    step_latents_tensor = gather_stacked_latents_for_sp(
        pipeline_config=pipeline_config,
        batch=batch,
        stacked_latents=step_latents_tensor,
    )
    batch.rollout_trajectory_data.dit_trajectory = RolloutDitTrajectory(
        latents=step_latents_tensor.cpu(),
        timesteps=torch.stack(step_timesteps, dim=0).cpu(),
        # [T+1] 完整噪声水平调度，含终点的 0；
        # 与 timesteps 不同，它不会因 step index 过滤而变短。
        sigmas=batch.scheduler.sigmas.detach().cpu().clone(),
    )
```

评论区精华

本 PR 没有实质性的 review 评论，四个 issue 评论中两个为 Gemini bot 关停提示，一个是 maintainer 的 /tag-and-rerun-ci，另一个是作者 Rockdu 自留的后续 TODO。核心讨论体现在 PR body 的三条语义说明中：

> sigmas 是 schedule 元数据而非逐样本数据，不受 rollout_return_step_indices 过滤影响；使用过滤时消费者必须按绝对 step index 对齐，不能与过滤后的 timesteps 位置一一对应。
>
> 当 num_outputs_per_prompt > 1 时，一个请求内所有样本共享同一份调度，因此 per-sample slicer 直接透传而不切片，同时避免 _extract_single_sample_tensor 误切长度恰等于 batch_size 的一维 schedule。
>
> 多次调用 set_timesteps 的多阶段流水线会快照最后一次调度，这与现有 timesteps 行为一致，且当前只支持单阶段 rollout 作为 RL 路径。

作者 TODO 为："TODO: ablation and switch to scheduler.timesteps as well"，即后续还要做消融实验，并让 timesteps 也直接取自 scheduler.timesteps，彻底消除训练侧重算路径。

- 后续切换 scheduler.timesteps 的 TODO (design): 未解决，属于后续演进方向；当前 PR 只处理 sigmas，timesteps 仍走重算 / 回传路径。
- sigmas 与 rollout_return_step_indices 的对齐语义 (design): 已由作者在 PR body 与代码注释中明确，作为既定语义被接受。
- 多阶段 set_timesteps 的快照语义 (other): 接受为已知限制，不留待办。

风险与影响

- 风险：

1. API 响应结构变更：POST /rollout/generate 的 dit_trajectory 对象新增 sigmas 字段（即使为 None 也会出现在序列化字典中），依赖严格 json schema 校验的旧消费方可能报未知字段错。
2. 对齐语义依赖：rollout_return_step_indices 过滤时，timesteps 被过滤而 sigmas 保持完整 [T+1]，消费者若位置对齐会导致错误的 sigma 取值，这是一个静默错误风险点。
3. 调度器状态依赖：_maybe_finalize_denoising_env_collection 直接访问 batch.scheduler.sigmas，若该路径在调度器重建 sigmas 之后执行，快照会变成最后一次调度；PR 已说明多阶段流水线只支持最后调度，当前单阶段 RL 路径不受影响。
4. 数据内存复制：新增 detach().cpu().clone() 对每批样例复制一份 fp32 [T+1] 数据，相比 latents 轨迹可以忽略不计，无实际性能风险。 - 影响：影响范围集中在多模态扩散模型的 RL 训练链路：rollout_denoising_mixin.py 是采集端，rollout_api.py 是服务端响应出口，下游训练侧消费者可获得与调度器完全一致的 fp32 sigma 序列，消除 bf16 重算引入的数值偏差。由于字段带默认值且无新增启动参数，存量未开启 rollout_return_dit_trajectory 的请求不受影响；开启该选项的请求响应会多出 sigmas 键，对已按 timesteps 反推的消费方需要切换口径。团队侧影响很小：变更共 20 行新增、0 删除，单 commit 提交，已通过单测与 H200 端到端验证。 - 风险标记：API 响应结构新增字段，sigmas 与过滤后 timesteps 对齐风险，多阶段调度快照语义未覆盖，依赖 batch.scheduler 内部状态

关联脉络

- 暂无明显关联 PR