PR #27420 完整报告

sgl-project/sglang

Diffusion: add JoyEcho multi-shot A/V generation support

合并时间：2026-06-26 15:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27420

执行摘要

- 一句话：新增 JoyEcho 多镜头音视频生成支持
- 推荐动作：值得精读。JoyEcho 的内存银行设计和 DMD 去噪流程是典型的视频生成模型实现示例，对于需要实现类似多镜头机制的开发者有参考价值。SP 处理中对复制前缀与分片后缀的 attention 处理方式（避免 all_gather）也值得关注。建议后续跟踪该 PR 引入的新模型路径的稳定性，以及社区对多镜头生成的支持反馈。

功能与动机

JoyEcho（jdopensource/JoyAI-Echo）是一个基于 LTX-2 的长格式音视频生成模型，核心思想是使用成对音视频内存银行：每个镜头将解码后的帧和音频潜变量存入滚动银行，后续镜头基于该内存前缀进行条件生成，从而支持多镜头、分钟级视频生成。该 PR 旨在让 SGLang Diffusion 能够加载 JoyEcho 并执行多镜头生成，同时保持与现有 LTX-2 基础设施的兼容。

实现拆解

步骤 1：扩展 LTX-2 VAE 与 Transformer 以支持 JoyEcho 特性

在 ltx_2_vae.py 中新增 video_encoder_variant="ltx_2_3_condition" 分支，使用 LTX23VideoConditionEncoder 编码内存帧。在 ltx_2.py 中增加逐块（per-block）音频自注意力掩码覆盖机制，用于 JoyEcho 后层掩码控制。

步骤 2：实现 JoyEcho 专用管道阶段

- JoyEchoMultishotSetupStage（setup.py）：管理多镜头会话重置与每次生成的种子增量（seed + shot_idx）。
- JoyEchoSigmaPreparationStage（setup.py）：安装 JoyEcho DMD sigma 调度（8 步，无 LTX-2 流偏移重映射）。
- JoyEchoMemoryBankFetchStage（memory.py）：从 PairedAudioVideoMemoryBank 获取之前镜头的解码帧与音频潜变量，并重新编码为内存前缀。
- JoyEchoDMDDenoisingStage（denoising.py）：DMD Euler 去噪逻辑，支持可选内存前缀、自定义音视频交叉 / 自注意力掩码、后层音频掩码覆盖，以及序列并行（SP）下的视频时间分片与音频全复制。
- JoyEchoAVDecodingStage（memory.py）：标准 LTX-2 音视频解码，并在启用内存银行时将新解码结果提交到内存槽。

步骤 3：实现成对音视频内存银行

PairedAudioVideoMemoryBank 类（memory.py）管理最多 7 个内存槽。辅助函数包括音频峰值窗口选择（select_max_response_audio_window_with_bounds）、视频帧索引选择（select_video_frame_indices_from_time_range）、内存槽范围计算（memory_slot_ranges）及成对内存跨注意力掩码构建（build_paired_memory_cross_mask）。

步骤 4：注册模型与管道

在 registry.py 中添加 jdopensource/JoyAI-Echo 到 overlay 映射，指向 Niehen6174/JoyAI-Echo-overlay。新建 JoyEchoPipeline（joy_echo_pipeline.py）继承 _BaseLTX2Pipeline，并在管道工厂中通过 EntryClass 导出。对应的管道配置 JoyEchoPipelineConfig 和采样参数 JoyEchoSamplingParams 分别定义在 configs/pipeline_configs/joy_echo.py 和 configs/sample/joy_echo.py。

步骤 5：添加验证与使用文档

添加单 GPU 烟雾测试用例（test/server/gpu_cases.py）和多 GPU SP 一致性测试（含一致性检查）。新增 JoyEcho Cookbook 文档（docs_new/cookbook/diffusion/JoyEcho/JoyEcho.mdx），介绍模型概述、部署方式、单镜头 / 多镜头用法及内存银行控制参数。

关键文件：

- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/memory.py（模块 内存银行；类别 source；类型 data-contract；符号 latent_window_size_to_pixel_window_size, select_max_response_audio_window_with_bounds, select_audio_window_with_bounds, mel_window_bounds_to_seconds）：核心内存银行实现，包含 PairedAudioVideoMemoryBank 类及音视频窗口选择、掩码构建等辅助函数；新增 1025 行，是 PR 中最大的单一文件。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/denoising.py（模块 去噪流程；类别 source；类型 data-contract；符号 JoyEchoDMDDenoisingStage, _prepare_denoising_loop, _zero_sp_shard_padding, _expand_sp_token_timestep）：JoyEcho DMD 去噪阶段，处理内存前缀注入、自定义注意力掩码、序列并行分片逻辑；新增 575 行。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/setup.py（模块 管道设置；类别 source；类型 data-contract；符号 JoyEchoMultishotSetupStage, init, _maybe_reset_multishot_session, forward）：多镜头会话管理和 DMD sigma 调度准备，控制 memory bank 重置和种子增量。
- python/sglang/multimodal_gen/runtime/pipelines/joy_echo_pipeline.py（模块 管道组装；类别 source；类型 dependency-wiring；符号 JoyEchoPipeline, init, _get_or_create_memory_bank, reset_memory_bank）：JoyEcho 管道定义，组装各阶段并暴露 EntryClass；新增 98 行。
- python/sglang/multimodal_gen/configs/pipeline_configs/joy_echo.py（模块 配置定义；类别 source；类型 core-logic；符号 _default_joy_echo_vae_config, JoyEchoPipelineConfig）：JoyEcho 管道配置，定义默认 sigma、memory bank 参数等。
- python/sglang/multimodal_gen/configs/models/dits/joy_echo.py（模块 模型配置；类别 source；类型 data-contract；符号 JoyEchoArchConfig, JoyEchoConfig）：JoyEcho

DiT 架构配置，设置 caption_proj_before_connector 等标志。

- python/sglang/multimodal_gen/configs/sample/joy_echo.py（模块 采样参数；类别 source；类型 core-logic；符号 JoyEchoSamplingParams）：JoyEcho 采样参数配置。
- python/sglang/multimodal_gen/runtime/models/vaes/ltx_2_vae.py（模块 VAE 适配；类别 source；类型 data-contract）：扩展 LTX-2 VAE 支持 ltx_2_3_condition 编码器变体，用于 JoyEcho 内存帧编码。
- python/sglang/multimodal_gen/runtime/models/dits/ltx_2.py（模块 Transformer 适配；类别 source；类型 data-contract）：修改 LTX-2 Transformer 支持逐块音频自注意力掩码覆盖，用于 JoyEcho 后层控制。
- python/sglang/multimodal_gen/registry.py（模块 模型注册；类别 source；类型 dependency-wiring）：注册 JoyAI-Echo overlay 映射，使加载模型时自动应用 overlay 分辨。
- docs_new/cookbook/diffusion/JoyEcho/JoyEcho.mdx（模块 文档；类别 other；类型 core-logic）：JoyEcho 使用文档，包含模型介绍、部署、单 / 多镜头用法示例。

关键符号：JoyEchoPipeline, PairedAudioVideoMemoryBank, JoyEchoDMDDenoisingStage, JoyEchoMultishotSetupStage, JoyEchoSigmaPreparationStage, JoyEchoMemoryBankFetchStage, JoyEchoAVDecodingStage, JoyEchoPipelineConfig, JoyEchoSamplingParams, select_max_response_audio_window_with_bounds, build_paired_memory_cross_mask, memory_slot_ranges

关键源码片段

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/memory.py

核心内存银行实现，包含 PairedAudioVideoMemoryBank 类及音视频窗口选择、掩码构建等辅助函数；新增 1025 行，是 PR 中最大的单一文件。

```
def select_max_response_audio_window_with_bounds(
    segment: Tensor,
    window_size: int,
) -> tuple[Tensor, Tensor, Tensor]:
    """在音频段 [B, C, T, F] 中，扫描候选窗口并返回 exp 响应最大的窗口及起止索引。"""
    if segment.dim() != 4:
        raise ValueError(
            f"Expected segment shape [B, C, T, F], got {tuple(segment.shape)}"
        )
    if window_size <= 0:
        raise ValueError(f"window_size must be positive, got {window_size}")
    num_time_steps = segment.shape[2]
    if num_time_steps <= 0:
        raise ValueError("Cannot select from an empty audio segment")
    # 扫描步长为 `window_size` // 4，确保覆盖最大范围
    scan_stride = max(1, window_size // 4)
    offsets = torch.arange(window_size, device=segment.device)
```

```
    max_start_idx = (
        num_time_steps - window_size
        if num_time_steps >= window_size
        else num_time_steps - 1
    )
    candidate_start_indices = list(range(0, max_start_idx + 1, scan_stride))
    if candidate_start_indices[-1] != max_start_idx:
        candidate_start_indices.append(max_start_idx)

    candidate_windows = []
    candidate_scores = []
    candidate_start_indices_tensor = torch.tensor(
        candidate_start_indices, device=segment.device, dtype=torch.long
    )
    for start_idx in candidate_start_indices:
        gather_indices = (start_idx + offsets).clamp(0, num_time_steps - 1).long()
        window = segment.index_select(dim=2, index=gather_indices)
        candidate_windows.append(window)
        # exp 求和作为响应分数
        candidate_scores.append(window.float().exp().sum(dim=(1, 2, 3)))

    scores = torch.stack(candidate_scores, dim=1)
    best_window_indices = scores.argmax(dim=1)
    best_start_indices = candidate_start_indices_tensor[best_window_indices]
    best_end_indices = torch.clamp(
        best_start_indices + window_size - 1, max=num_time_steps - 1
    )
    selected_windows = torch.cat(
        [
            candidate_windows[int(best_window_indices[batch_index])][
                batch_index : batch_index + 1
            ]
            for batch_index in range(segment.shape[0])
        ],
        dim=0,
    )
    return selected_windows, best_start_indices, best_end_indices
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/denoising.py

JoyEcho DMD 去噪阶段，处理内存前缀注入、自定义注意力掩码、序列并行分片逻辑；新增 575 行。

```
class JoyEchoDMDDenoisingStage(LTX2AVDenoisingStage):
    """JoyEcho DMD denoising with optional memory prefix and late-layer masks."""

    def _prepare_denoising_loop(
        self,
        batch: Req,
```

```
        server_args: ServerArgs,
    ) -> LTX2DenoisingContext:
        ctx = super()._prepare_denoising_loop(batch, server_args)
        if get_sp_world_size() <= 1:
            return ctx
        # JoyEcho DMD: 视频按时间分片，音频在每个 rank 上全复制
        # 若音频和视频同时分片，则每层需要大量 all_gather，开销过大
        ctx.replicate_audio_for_sp = True
        batch.ltx23_audio_replicated_for_sp = True
        batch.did_sp_shard_audio_latents = False
        return ctx

    def _prepare_ltx2_model_inputs(
        self,
        ctx: LTX2DenoisingContext,
        step: DenoisingStepState,
        batch: Req,
        server_args: ServerArgs,
        sigma: torch.Tensor,
    ) -> LTX2ModelInputs:
        model_inputs = super()._prepare_ltx2_model_inputs(
            ctx, step, batch, server_args, sigma
        )
        if not batch.did_sp_shard_latents:
            return model_inputs # 未使用 SP 时直接返回父类结果

        batch_size = int(model_inputs.latent_model_input.shape[0])
        seq_v = int(model_inputs.latent_model_input.shape[1])
        video_valid = batch.sp_video_valid_token_count
        # 构建视频自注意力掩码（padding mask）
        video_self_attention_mask = self._build_ltx2_sp_padding_mask(
            batch,
            seq_len=seq_v,
            batch_size=batch_size,
            key="sp_video_valid_token_count",
            device=model_inputs.latent_model_input.device,
        )
        # 后续处理 ...
        return model_inputs
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/joy_echo/setup.py

多镜头会话管理和 DMD sigma 调度准备，控制 memory bank 重置和种子增量。

```
class JoyEchoMultishotSetupStage(PipelineStage):
    """Apply official per-shot seeding before input validation and latent noise."""

    def __init__(self, pipeline: JoyEchoPipeline) -> None:
        super().__init__()
```

```
        self.pipeline = pipeline

    def _maybe_reset_multishot_session(self, batch: Req) -> None:
        """在 `generate()` 新会话开始时复位 shot index 和 memory bank。"""
        if not batch.reset_memory_bank:
            return
        session_id = batch.request_id
        if session_id is not None:
            if session_id == self.pipeline._multishot_session_id:
                return # 同一会话内不重复复位
            self.pipeline._multishot_session_id = session_id
            self.pipeline.multishot_index = 0
            self.pipeline.reset_memory_bank()
            logger.info(
                "JoyEcho memory bank reset for new multi-shot session (request_id=%s)",
                session_id,
            )
            return
        # 无 request_id 时，若 shot index 为 0 则复位
        if self.pipeline.multishot_index == 0:
            self.pipeline.reset_memory_bank()
            logger.info("JoyEcho memory bank reset for new multi-shot session")

    def forward(self, batch: Req, server_args: ServerArgs) -> Req:
        if not batch.enable_memory_bank:
            return batch
        self._maybe_reset_multishot_session(batch)
        shot_idx = self.pipeline.multishot_index
        self.pipeline.multishot_index += 1
        base_seed = batch.seed
        if isinstance(base_seed, list):
            if not base_seed:
                raise ValueError("seed list must not be empty for JoyEcho multi-shot")
            base_seed = base_seed[0]
        # 官方逻辑：prompt_seed = int(cfg.seed) + shot_idx
        batch.seed = int(base_seed) + shot_idx
        logger.info(
            "JoyEcho multi-shot setup: shot_idx=%d seed=%d",
            shot_idx,
            batch.seed,
        )
        return batch
```

评论区精华

1. 文件结构重组：reviewer mickqian 建议将 JoyEcho 相关 stage 文件放到 model_specific_stages/joy_echo/ 目录下，author 同意并后续重构。

2. SP 支持讨论：mickqian 建议添加序列并行（SP）支持及多 GPU 一致性测试，author 最初解释实现复杂度高，但在后续提交中实际实现了 SP，并添加了 2 GPU 测试用例。
3. 非 diffusers 注册：mickqian 询问 utils.py 中添加 joy-echo 非 diffusers 模式注册的必要性，author 确认冗余并计划清理。
4. USPAttention 接口：mickqian 建议使用 USPAttention.forward 的 num_replicated_prefix 参数，author 解释了 JoyEcho 的内存布局（复制前缀 + 分片后缀）与现有 API 不兼容，涉及 3D 掩码、交叉注意力等差异。
5. 测试治理：mickqian 要求上传一致性 ground truth 并启用检查，author 已创建 CI data PR，待合并。

- 文件结构重组 (design): author 同意并在后续提交中完成了重组。
- 序列并行支持 (performance): PR 中实际实现了 SP 支持（视频分片、音频全复制），并添加了 2 GPU 测试用例。
- 非 diffusers 模型注册必要性 (question): author 确认冗余，计划清理。
- USPAttention 接口兼容性 (design): 维持自定义 SP 路径，未使用通用接口，但讨论澄清了设计约束。
- 测试治理与一致性检查 (testing): CI data PR 待合并；测试用例已更新。

风险与影响

- 风险：
  1. 新模型路径回归：首次引入 JoyEcho 模型，推理路径可能存在未发现的边缘情况，特别是多镜头交互和内存银行状态转换。
  2. 序列并行新增路径：PR 中新增了 SP 支持，涉及视频分片、音频复制和特殊掩码构建，若未充分测试可能在其他 SP 配置下出现错误。
  3. LTX-2 基座改动影响：对 ltx_2_vae.py 和 ltx_2.py 的修改（条件编码器、后层掩码）虽为条件性，但可能影响其他基于 LTX-2 的模型（如 LTX-2.3 HQ）。
  4. 测试覆盖不足：烟雾测试仅覆盖单镜头、单 GPU 场景；多 GPU 一致性测试依赖 CI data PR 合并，当前尚未生效。
  5. 资源配置：JoyEchoPipelineConfig 中引入了多个新参数（如 memory_max_size, audio_window_size），若默认值不合理可能导致生成失败或性能问题。 - 影响：用户：可直接通过加载 jdopensource/JoyAI-Echo 使用 JoyEcho 生成多镜头音视频，支持 enable_memory_bank、reset_memory_bank 等参数控制多镜头行为。系统：新增约 2200 行代码，包含独立的 joy_echo/ 模块、配置注册和 overlay 映射。对现有 LTX-2 VAE 和 Transformer 的添加是可选分支，不影响原有模型路径。团队：模块化设计良好，JoyEcho 特定代码集中在 model_specific_stages/joy_echo/ 下，便于后续维护和扩展。
- 风险标记：新模型推理路径，内存银行状态机，序列并行新增路径，LTX-2 基础改动影响，测试覆盖待完善

关联脉络

- PR #29281 [KDA-Pilot] Add diffusion causal Conv3D cat-pad CUDA fast path for Cosmos3: 同为 diffusion 模型支持 PR，共享部分 pipeline 基础设施（如 stage 模式）。