PR #24660 完整报告

sgl-project/sglang

[diffusion] fix: further align ltx2.3 accuracy with tp

合并时间：2026-05-11 13:42

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24660

执行摘要

- 一句话：修复 LTX-2/2.3 扩散模型多 GPU 精度对齐与 HQ 两阶段路径
- 推荐动作：建议所有使用 LTX-2/2.3 模型进行推理和 CI 测试的团队仔细阅读此 PR。其中关于 CFG 引导分支合并、Attention Backend 选择、RoPE 精度控制的决策值得在其它扩散模型推理框架中借鉴。

功能与动机

根据 PR 描述，主要动机是提升 LTX-2 / LTX-2.3 扩散模型与官方输出的精度对齐，尤其是多 GPU CI 用例和 LTX-2.3 HQ 两阶段路径。之前的 LTX-2.3 多 GPU CI 设置对某些用例的一致性阈值过松，且使用了较慢或对齐性较差的并行模式；HQ 路径在 PR#23366 的 SpongeBob 复现中出现了回归。

实现拆解

1. 重构 CFG 引导分支合并逻辑：引入 _ltx2_combine_guided_x0_parallel_av 方法替代旧的 _ltx2_combine_guided_x0_parallel，将视频和音频引导分支先分别计算为 x0 后再合并，通过一次 all-reduce 同步所有分支，避免折叠系数导致的 bf16 舍入差异。
2. 修复 RoPE 频率生成：移除基于 CPU/NumPy 的缓存函数 _ltx2_rope_freq_grid_np，改为在目标设备上使用指定精度（float32/float64）直接生成频率，保留与官方一致的舍入轨迹；Gemma3 编码器也改为使用预计算的 cos_sin_cache 并通过 index_select 查找，同时修复滑动注意力模式的检测逻辑（支持 sliding_window_pattern）。
3. 统一 Attention Backend：让 transformer_2 等次级组件继承基础组件的 attention backend，允许在 LocalAttention 和 USPAttention 中使用 cuDNN SDP 后端（通过 allow_cudnn_sdp 参数），以匹配官方 LTX 的 torch_sdpa 行为；同时处理旧版 FA3 varlen kernel 不支持 out= 关键字的情况。
4. 对齐 LTX-2.3 HQ 路径：恢复阶段 1 和阶段 2 的 res2s 噪声精度（保持 float64 轨迹），修复 sigma 空间数学计算，并确保 HQ 变长序列的 CFG 广播源使用全局 rank。
5. 收紧 CI 测试：调整多 GPU 用例的并行策略（TP/CFG Parallel 取代 SP/Ulysses），收紧一致性阈值（clip/SSIM/PSNR/mean_abs_diff），添加新用例到官方一致性 GT 集合，并改进测试失败时的 HTML 报告，包含生成图像链接。

关键文件：

- python/sglang/multimodal_gen/runtime/pipelines_core/stages/ltx_2_denoising.py（模块 去噪流水线；类别 source；类型 core-logic；符号 _ltx2_combine_guided_x0_parallel, _ltx2_combine_guided_x0_parallel_av, _move_ltx2_scheduler_tensors_to_device）：核心逻辑变更：重构 CFG 引导分支合并，移除旧方法，引入新方法 _ltx2_combine_guided_x0_parallel_av，支持视频 / 音频分离预处理后再合并，改变舍入路径
- python/sglang/multimodal_gen/runtime/models/encoders/gemma_3.py（模块 文本编码器；类别 source；类型 data-contract；符号 rotary_emb, _apply_rotary_pos_emb）：Gemma3 编码器 RoPE 和滑动注意力修复：改用预计算缓存，支持 sliding_window_pattern，提升精度
- python/sglang/multimodal_gen/runtime/models/dits/ltx_2.py（模块 扩散 Transformer；类别 source；类型 data-contract；符号 _ltx2_rope_freq_grid_np）：LTX DiT 模型 RoPE 频率生成方式变更，移除 NumPy 缓存，改用设备端生成；attention 后端开放 cuDNN SDP
- python/sglang/multimodal_gen/runtime/layers/attention/layer.py（模块 注意力层；类别 source；类型 dependency-wiring）：Attention 层增加 allow_cudnn_sdp 参数，控制 sdpa_kernel 上下文，使 LTX 可以使用 cuDNN 加速
- python/sglang/multimodal_gen/runtime/layers/attention/backends/sdpa.py（模块 SDPA 后端；类别 source；类型 dependency-wiring）：SDPA 后端同样增加 allow_cudnn_sdp 参数，使官方 LTX 的 torch_sdpa 路径可选择 cuDNN 后端
- python/sglang/multimodal_gen/test/test_utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 _save_generated_artifact_images）：测试工具增强：添加 _save_generated_artifact_images 函数保存生成帧图像，并在 HTML 报告中添加链接，便于调试
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising_av.py（模块 去噪流水线；类别 source；类型 core-logic）：伴随 ltx_2_denoising.py 的接口调整，更新配置键
- python/sglang/multimodal_gen/runtime/server_args.py（模块 服务参数；类别 source；类型 core-logic）：量化配置优先级调整：显式 --quantization 优先于检查点 metadata
- python/sglang/multimodal_gen/test/server/perf_baselines.json（模块 性能基线；类别 test；类型 test-coverage）：更新多 GPU LTX 性能基线，匹配新的并行策略和精度对齐
- python/sglang/multimodal_gen/runtime/pipelines_core/executors/parallel_executor.py（模块 并行执行器；类别 source；类型 core-logic）：修复 CFG 并行阶段的广播源：使用全局 source rank 而非本地 rank
- docs_new/src/snippets/diffusion/ltx-deployment.jsx（模块 部署文档；类别 source；类型 core-logic）：文档示例更新，反映新的部署配置
- python/sglang/multimodal_gen/test/server/consistency_threshold.json（模块 一致性阈值；类别 test；类型 test-coverage）：收紧多个 LTX 用例的一致性阈值，提升 CI 质量门禁

关键符号：_ltx2_combine_guided_x0_parallel_av, _move_ltx2_scheduler_tensors_to_device, _apply_rotary_pos_emb,

_ltx2_rope_freq_grid_np, apply_split_rotary_emb, _save_generated_artifact_images

关键源码片段

python/sglang/multimodal_gen/runtime/pipelines_core/stages/ltx_2_denoising.py

核心逻辑变更：重构 CFG 引导分支合并，移除旧方法，引入新方法 _ltx2_combine_guided_x0_parallel_av，支持视频 / 音频分离预处理后再合并，改变舍入路径

```
# python/sglang/multimodal_gen/runtime/pipelines_core/stages/ltx_2_denoising.py

@classmethod
def _ltx2_combine_guided_x0_parallel_av(
    cls,
    *,
    video_latents: torch.Tensor,
    audio_latents: torch.Tensor,
    local_video_velocities: dict[str, torch.Tensor],
    local_audio_velocities: dict[str, torch.Tensor],
    video_sigma: float | torch.Tensor,
    audio_sigma: float | torch.Tensor,
    video_cfg_scale: float,
    video_stg_scale: float,
    video_rescale_scale: float,
    video_modality_scale: float,
    audio_cfg_scale: float,
    audio_stg_scale: float,
    audio_rescale_scale: float,
    audio_modality_scale: float,
) -> tuple[torch.Tensor, torch.Tensor]:
    """
    在CFG并行中，跨rank重建完整的引导分支，然后分别计算视频和音频的guided x0。
    不再合并系数后all-reduce，而是先将每个分支的x0通过all-reduce同步，
    再使用官方标准的组合公式，以消除bf16下系数折叠导致的数值漂移。
    """
    # 获取第一个 velocity 来构造模板张量（用于 zero 填充）
    first_video_velocity = next(iter(local_video_velocities.values()))
    first_audio_velocity = next(iter(local_audio_velocities.values()))
    video_template = cls._ltx2_velocity_to_x0(
        video_latents, first_video_velocity, video_sigma
    )
    audio_template = cls._ltx2_velocity_to_x0(
        audio_latents, first_audio_velocity, audio_sigma
    )
    video_numel = video_template.numel()
    # 对 4 个分支（cond, neg, perturbed, modality）分别收集并 all-reduce
    branches: dict[str, tuple[torch.Tensor, torch.Tensor]] = {}
    for name in ("cond", "neg", "perturbed", "modality"):
```

```
            if name in local_video_velocities:
                # 该 rank 拥有此分支的真实输出
                local_video = cls._ltx2_velocity_to_x0(
                    video_latents, local_video_velocities[name], video_sigma
                )
                local_audio = cls._ltx2_velocity_to_x0(
                    audio_latents, local_audio_velocities[name], audio_sigma
                )
            else:
                # 该 rank 不负责此分支，用零填充
                local_video = torch.zeros_like(video_template)
                local_audio = torch.zeros_like(audio_template)
            # 拼接后 all-reduce，使每个 rank 都获得完整的分支 x0
            flat = torch.cat((local_video.reshape(-1), local_audio.reshape(-1)))
            flat = cfg_model_parallel_all_reduce(flat)
            branches[name] = (
                flat[:video_numel].reshape_as(video_template),
                flat[video_numel:].reshape_as(audio_template),
            )
        # 分别计算视频和音频的 guided x0（使用官方组合公式，不折叠系数）
        guided_video = cls._ltx2_calculate_guided_x0(
            cond=branches["cond"][0],
            uncond_text=branches["neg"][0],
            uncond_perturbed=branches["perturbed"][0],
            uncond_modality=branches["modality"][0],
            cfg_scale=video_cfg_scale,
            stg_scale=video_stg_scale,
            rescale_scale=video_rescale_scale,
            modality_scale=video_modality_scale,
        )
        guided_audio = cls._ltx2_calculate_guided_x0(
            cond=branches["cond"][1],
            uncond_text=branches["neg"][1],
            uncond_perturbed=branches["perturbed"][1],
            uncond_modality=branches["modality"][1],
            cfg_scale=audio_cfg_scale,
            stg_scale=audio_stg_scale,
            rescale_scale=audio_rescale_scale,
            modality_scale=audio_modality_scale,
        )
        return guided_video, guided_audio
```

python/sglang/multimodal_gen/runtime/models/encoders/gemma_3.py

Gemma3 编码器 RoPE 和滑动注意力修复：改用预计算缓存，支持 sliding_window_pattern，提升精度

```
# python/sglang/multimodal_gen/runtime/models/encoders/gemma_3.py

# 在 __init__ 中：
```

```
# 之前的代码直接从 layer_types 列表索引，现在兼容 sliding_window_pattern 配置
sliding_window_pattern = getattr(
    config.text_config, "sliding_window_pattern", None
)
self.is_sliding = (
    bool((layer_id + 1) % sliding_window_pattern)
    if sliding_window_pattern
    else False
)
self.layer_type = "sliding_attention" if self.is_sliding else None

# RoPE 初始化时，将 self.rotary_emb 重命名为 self.rotary_pos_emb
self.rotary_pos_emb = get_rope(
    self.head_dim,
    rotary_dim=self.head_dim,
    max_position=config.text_config.max_position_embeddings,
    base=self.rope_theta,
    rope_scaling=rope_scaling,
    is_neox_style=True,
)

# 新增方法 _apply_rotary_pos_emb，使用预计算 cos_sin_cache，避免每步重新计算 inv_freq
def _apply_rotary_pos_emb(self, positions, q, k):
    positions_flat = positions.flatten().to(
        device=self.rotary_pos_emb.cos_sin_cache.device, dtype=torch.long
    )
    cos_sin = self.rotary_pos_emb.cos_sin_cache.index_select(0, positions_flat)
    cos, sin = cos_sin.chunk(2, dim=-1)
    # 扩展半维度频率以匹配 head_dim（HF Gemma3 风格）
    cos = torch.cat((cos, cos), dim=-1).to(device=q.device, dtype=q.dtype)
    sin = torch.cat((sin, sin), dim=-1).to(device=q.device, dtype=q.dtype)
    cos = cos.unsqueeze(1)
    sin = sin.unsqueeze(1)
    # 应用旋转
    q = q.reshape(num_tokens, -1, self.head_dim)
    k = k.reshape(num_tokens, -1, self.head_dim)
    q = q * cos + _rotate_half(q) * sin
    k = k * cos + _rotate_half(k) * sin
    return q, k
```

python/sglang/multimodal_gen/runtime/models/dits/ltx_2.py

LTX DiT 模型 RoPE 频率生成方式变更，移除 NumPy 缓存，改用设备端生成；attention 后端开放 cuDNN SDP

```
# python/sglang/multimodal_gen/runtime/models/dits/ltx_2.py

# 在 RoPE 频率生成部分（原使用缓存函数 _ltx2_rope_freq_grid_np）：
# 新实现：直接在目标设备上生成频率，保留 float64 精度路径
freqs_dtype = torch.float64 if self.double_precision else torch.float32
```

```
pow_indices = torch.pow(
    self.theta,
    torch.linspace(
        start=0.0,
        end=1.0,
        steps=self.dim // num_rope_elems,
        dtype=freqs_dtype,
        device=device,
    ),
)
freqs = (pow_indices * torch.pi / 2.0).to(dtype=torch.float32)
# ... 后续与 grid 组合

# 在 TransformerBlock 初始化中，允许 cuDNN SDP 后端以匹配官方 LTX 的 torch_sdpa 行为
if use_local:
    self.attn = LocalAttention(
        # ... 其他参数
        allow_cudnn_sdp=True,
    )
```

评论区精华

本 PR 没有实质性的 review 讨论，主要作者独立完成调试和验证。PR body 中明确排除了 transformer fp8-cast，指出该 PR 的目的路径（CI 和 HQ 精度对齐）无需此变更，为后续独立分支留出空间。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. CFG 分支合并重构：新方法改变了 all-reduce 和系数组合顺序，可能影响单 GPU 和多 GPU 结果的数值一致性，已通过官方输出 PSNR 验证（20.722 vs 20.707）。
  2. RoPE 精度切换：从 float64 NumPy 改为设备端 float32/float64 生成，可能改变所有 LTX 模型的生成轨迹，但已在多 GPU CI 中验证。
  3. Attention Backend：默认启用 cuDNN SDP 可能在非 CUDA 设备上回退，且未覆盖所有后端（如 FlashInfer），需确保 FA3 varlen 无 out 关键字的兼容处理。
  4. CI 阈值收紧：新阈值更严格，可能导致后续升级引入波动，但提升了质量保障。 - 影响：对用户：LTX-2/2.3 模型的生成结果更稳定、与官方对齐度更高，特别是在多 GPU 和 HQ 两阶段场景下。对系统：引入少量性能优化（如 CFG Parallel 比 Ulysses 更快），但主要改进是数值精度。对团队：为后续 LTX 精度优化提供了清晰的调试路径和 CI 基准，并展示了跨文件精细调优的工程实践。 - 风险标记：核心路径变更，精度敏感逻辑，多 GPU 一致性，CI 阈值收紧

关联脉络

- PR #23366 [LTX-2.3] SpongeBob repro for HQ precision: PR#23366 是 LTX-2.3 HQ 精度对齐的原始复现用例，本 PR 在 body 中引用了该 PR 的 repro 结果来验证 HQ 路径的修复效果