PR #27524 完整报告

sgl-project/sglang

[diffusion] Progressive resolution growing for Image and Video models via GPU DCT upsampling with up to 2X+ speedup

合并时间：2026-06-08 20:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27524

执行摘要

- 一句话：渐进分辨率扩散加速，支持 5 种模型最高 2.32x加速
- 推荐动作：值得精读。核心设计模式（钩子基类 + 路由器）适合作为框架功能扩展的范本。DCT 上采样和贝叶斯最优转换的数学实现具有参考价值。建议关注频谱系数拟合流程和 Wan 视频的对齐逻辑，这两处最有可能需要根据实际场景调整。

功能与动机

Transformer attention 的复杂度是 O(n²) 序列长度。对于图像 / 视频扩散模型，早期去噪步骤几乎只在低频区域运行——高频空间细节尚未被扩散过程激活。在这些步骤上使用全分辨率计算是纯粹的浪费。基于 Spectral Progressive Diffusion (arXiv 2605.18736)。

实现拆解

1. 核心数学与频谱操作：在 python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/spectral_ops.py 中实现GPU DCT-II/IDCT-II（通过torch.fft）。在 scheduler_utils.py 中实现贝叶斯最优阶段转换计算函数 compute_stage_transitions、find_transition_steps 和 reset_scheduler_at_step。
2. 基类 ProgressiveDenoisingStage：在 denoising.py 中定义，继承 DenoisingStage，通过多个扩展钩子（_unpack_latent、_repack_latent、_latent_scale_factor、_on_resolution_change、_generate_initial_noise）支持模型特定子类。核心去噪循环在 forward 中执行多阶段粗到细流程，阶段间调用 apply_upsample。
3. 模型特定子类：为每个模型添加 FluxProgressiveDenoisingStage（flux.py）、Flux2ProgressiveDenoisingStage（flux_2.py，行优先 pack/unpack）、ZImageProgressiveDenoisingStage（zimage.py，5D 潜变量）、WanProgressiveDenoisingStage（wan.py，视频时空对齐）、QwenImageProgressiveDenoisingStage（qwen_image.py，2x2 patchify）。每个子类处理独有的潜变量布局、RoPE 更新和分辨率变化钩子。
4. Pipeline 路由与参数：在每个模型的 pipeline 文件（flux.py、flux_2.py、zimage_pipeline.py、wan_pipeline.py、qwen_image.py）中，将直接使用的 DenoisingStage 替换为 *DenoisingStageRouter，根据批次 progressive_mode 动态选择全分辨率或渐进阶段。修改 composed_pipeline_base.py 添加 add_progressive_denoising_stage 支持。在 sampling_params.py 添加

progressive_mode、progressive_levels、progressive_delta 字段和 CLI 标志。

5. 测试与文档：添加单元测试 test_progressive.py（覆盖格式转换、数学计算、参数验证），conftest.py 提供默认 ServerArgs。编写用户文档 progressive_resolution.mdx 和基准测试指南。

关键文件：

- python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/denoising.py（模块 核心逻辑；类别 source；类型 core-logic；符号 is_progressive_resolution_mode, unpack_2x2_latent, pack_2x2_latent, _P_omega）：核心基类 ProgressiveDenoisingStage 和多阶段去噪循环，定义了钩子扩展点，是渐进分辨率的逻辑中枢。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/flux_2.py（模块 核心逻辑；类别 source；类型 core-logic；符号 _flux2_unpack, _flux2_pack, Flux2ProgressiveDenoisingStage, init）：FLUX.2 特定的渐进去噪阶段，展示行优先潜变量 pack/unpack 和 _latent_scale_factor 钩子。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/wan.py（模块 核心逻辑；类别 source；类型 core-logic；符号 WanProgressiveDenoisingStage, init, _latent_scale_factor, _unpack_latent）：Wan T2V 视频模型渐进阶段，处理 5D 时空潜变量、空间对齐和 no-op 分辨率变化钩子。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/spectral_ops.py（模块 频谱运算；类别 source；类型 core-logic；符号 dct_1d, idct_1d, dct_2d, idct_2d）：GPU 原生 DCT-II / IDCT-II 实现，所有上采样的核心数学运算，精度匹配 scipy。
- python/sglang/multimodal_gen/runtime/pipelines_core/composed_pipeline_base.py（模块 Pipeline 基类；类别 source；类型 core-logic；符号 add_progressive_denoising_stage, create_stage）：修改 pipeline 基类以支持渐进阶段注册，是路由改造的关键入口。

关键符号：is_progressive_resolution_mode, unpack_2x2_latent, pack_2x2_latent, _P_omega, _activation_time, compute_stage_transitions, find_transition_steps, reset_scheduler_at_step, dct_1d, idct_1d, dct_2d, idct_2d, dct_upsample_2d, apply_upsample, ProgressiveDenoisingStage.forward, FluxProgressiveDenoisingStage._unpack_latent, Flux2ProgressiveDenoisingStage._latent_scale_factor, WanProgressiveDenoisingStage.forward, ZImageProgressiveDenoisingStage._generate_initial_noise, QwenImageProgressiveDenoisingStage._on_resolution_change, _FluxDenoisingStageRouter.component_uses

关键源码片段

python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/denoising.py

核心基类 ProgressiveDenoisingStage 和多阶段去噪循环，定义了钩子扩展点，是渐进分辨率的逻辑中枢。

```
def compute_stage_transitions(
    delta: float,
    n_levels: int,
    A: float,
    beta: float,
    H_lat: int,
    W_lat: int,
) -> dict[int, float]:
    # 根据贝叶斯最优准则计算每个阶段的转换 sigma 阈值
    # P(omega) = A * |omega|^(-beta) 为功率谱经验拟合
    stage_sigmas: dict[int, float] = {1: 1.0}
    num_stages = n_levels + 1
    for stage in range(2, num_stages + 1):
        # 计算上一阶段的空间频率 omega（取最小维的一半）
        H_prev = H_lat // (2 ** (num_stages - stage + 1))
        W_prev = W_lat // (2 ** (num_stages - stage + 1))
        w = min(H_prev, W_prev) // 2
        stage_sigmas[stage] = _activation_time(_P_omega(w, A, beta), delta)
    return stage_sigmas


def find_transition_steps(
    scheduler_sigmas: torch.Tensor,
    stage_sigmas: dict[int, float],
    n_steps: int,
) -> dict[int, int]:
    # 将 sigma 阈值映射到具体去噪步数
    sigmas_list = scheduler_sigmas.cpu().tolist() # 提前转 list 避免逐元素 Python 开销
    transition_steps: dict[int, int] = {}
    for stage, threshold in stage_sigmas.items():
        if stage == 1:
            continue
        found = n_steps
        for step_index in range(n_steps):
            if sigmas_list[step_index] <= threshold:
                found = step_index
                break
        transition_steps[stage] = found
    return transition_steps
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/flux_2.py

FLUX.2 特定的渐进去噪阶段，展示行优先潜变量 pack/unpack 和 _latent_scale_factor 钩子。

```
def _flux2_unpack(latent: torch.Tensor, h_lat: int, w_lat: int) -> torch.Tensor:
    # packed [B, H_lat*W_lat, C] -> spatial [B, C, H_lat, W_lat]（行优先）
    B, _S, C = latent.shape
```

```
    return latent.permute(0, 2, 1).reshape(B, C, h_lat, w_lat)

def _flux2_pack(x: torch.Tensor) -> torch.Tensor:
    # spatial [B, C, H_lat, W_lat] -> packed [B, H_lat*W_lat, C]
    B, C, H, W = x.shape
    return x.reshape(B, C, H * W).permute(0, 2, 1)

class Flux2ProgressiveDenoisingStage(ProgressiveDenoisingStage):
    """FLUX.2 渐进去噪阶段，行优先 pack/unpack + RoPE 更新"""

    def _latent_scale_factor(self, server_args: ServerArgs) -> int:
        # FLUX.2 潜变量空间维度是像素的 1/(vae_scale_factor * 2)
        return server_args.pipeline_config.vae_config.arch_config.vae_scale_factor * 2

    def _unpack_latent(self, latent, h_lat, w_lat):
        return _flux2_unpack(latent, h_lat, w_lat)

    def _repack_latent(self, x_spatial, h_lat, w_lat, batch, server_args):
        return _flux2_pack(x_spatial)
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/progressive_resolution/wan.py

Wan T2V 视频模型渐进阶段，处理 5D 时空潜变量、空间对齐和 no-op 分辨率变化钩子。

```
class WanProgressiveDenoisingStage(ProgressiveDenoisingStage):
    """Wan T2V 渐进去噪阶段，无 pack/unpack，仅空间上采样"""

    def _latent_scale_factor(self, server_args: ServerArgs) -> int:
        # WanVAEArchConfig 使用 spatial_compression_ratio 而非 vae_scale_factor
        arch = server_args.pipeline_config.vae_config.arch_config
        return getattr(arch, "vae_scale_factor", None) or getattr(
            arch, "spatial_compression_ratio", 8
        )

    def _unpack_latent(self, latent, h_lat, w_lat):
        return latent # 已经是 [B, C, T, H, W]

    def _repack_latent(self, x_spatial, h_lat, w_lat, batch, server_args):
        return x_spatial

    def _on_resolution_change(self, ctx, batch, server_args, new_h_pixel, new_w_pixel):
        pass # Wan T2V 没有空间位置编码需要更新

    def forward(self, batch: Req, server_args: ServerArgs) -> Req:
        # 确保每个渐进阶段潜变量 H/W 为偶数（Conv3d stride=(1,2,2) 要求）
        mode = getattr(batch, "progressive_mode", "fullres") or "fullres"
        if not is_progressive_resolution_mode(mode):
            return super().forward(batch, server_args)
        levels = getattr(batch, "progressive_levels", 1)
```

```
vae_scale = self._latent_scale_factor(server_args)
align_pixels = vae_scale * (2**levels) * 2
# 向下对齐，但至少为 align_pixels（防止零分辨率）
h_aligned = max((batch.height // align_pixels) * align_pixels, align_pixels)
w_aligned = max((batch.width // align_pixels) * align_pixels, align_pixels)
batch.height = h_aligned
batch.width = w_aligned
return super().forward(batch, server_args)
```

评论区精华

Gemini Code Assist 审查提出了 5 个技术问题：

- 重复测试类：TestProgressiveSamplingParams 出现在两个测试文件中，批注建议删除重复。
- .item() 效率：在 scheduler_utils.py 的嵌套循环中使用 .item() 逐个访问张量元素，批注建议替换为 .tolist() 以提升性能。
- torch.Generator 跨平台兼容性：在 upsample.py 中创建 Generator 时指定 device 可能导致非 CUDA 设备崩溃，批注建议在 CPU 上生成噪声再转移到目标设备。
- Wan 零分辨率边缘情况：如果 batch.height 或 width 小于对齐像素，对齐后可能得到 0，批注建议添加 max 防护。
- 负嵌入批处理维度：在 text_encoding.py 中，2-D 负嵌入直接返回导致 cfg 拼接形状不匹配，批注提供修复建议。

以上问题在后续提交中全部解决：重复类被移除，.item() 替换为 .tolist()，Generator 改为 CPU 创建，Wan 对齐添加 max 防护，负嵌入修复已包含。此外，mickqian 建议聚合分散的单元测试，已通过 commit 28c3b157 合并到 test_progressive.py。

- 重复测试类 TestProgressiveSamplingParams (testing): 后续提交中移除重复定义。
- .item() 在嵌套循环中的性能问题 (performance): 已替换为 .tolist()。
- torch.Generator 跨平台兼容性 (correctness): 已修改为在 CPU 上生成噪声。
- Wan 零分辨率边缘情况 (correctness): 已添加 max(..., align_pixels) 防护。
- 负嵌入批处理维度不匹配 (correctness): 已修复。
- 聚合单元测试 (testing): 测试已合并到 test_progressive.py。

风险与影响

- 风险：
  - 视觉质量回归风险：DCT 上采样是近似逆问题，高 δ 值可能导致细节损失。默认 fullres 模式不受影响。
  - 频谱系数依赖外部拟合：FLUX.1 和 Wan 的系数来自特定数据集，Qwen-Image 使用占位系数，可能不准确导致次优阶段转换。
  - 缺少多 GPU 测试覆盖：diffusion 模型通常运行在单卡，多 GPU 场景的 progressive 路由未充分测试。
  - Wan 分辨率对齐限制：Wan 的 patch 嵌入要求每个阶段潜变量 H/W 为偶数，对齐逻辑可能导致最终像素非标准。

    - 设备兼容性：DCT 操作依赖 torch.fft，在 NPU/AMD 等非 CUDA 设备上可能回退或精度不同。
- 影响：
    - 用户影响：默认行为完全不变。启用 progressive 模式后生成速度显著提升（最高 2.32x），但视觉质量可能略有下降（取决于 δ）。
    - 系统影响：新增约 2500 行核心源码，但模块化设计（钩子、路由器）对现有 pipeline 侵入性低。5 个 pipeline 文件修改为路由模式，但原有逻辑完全保留。
    - 团队影响：添加新扩散模型需要实现特定的钩子（pack/unpack、分辨率变化、噪声生成），还需要为 VAE 拟合功率谱系数 A 和 β。
    - 风险标记：视觉质量回归风险，频谱系数依赖外部拟合，缺少多 GPU 测试覆盖，Wan 分辨率对齐限制，设备兼容性

关联脉络

- PR #26961 [WIP] Spectral Progressive Diffusion (old PR): 该 PR 的旧版本链接，PR body 中明确引用，是同一功能线的早期提交。