PR #33275 完整报告

sgl-project/sglang

[diffusion] model: support minimax-h3

合并时间：2026-08-02 22:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33275

执行摘要

- 一句话：新增 MiniMax-H3 原生音视频生成管线与部署支持
- 推荐动作：值得精读。该 PR 展示了大型多模态生成模型在 SGLang 中落地的完整工程范式，尤其是 video_adapter.py 的 fail-closed 请求校验、packed_sequence.py 的布局构造、denoise_loop.py 的静态分支预计算与避免热点循环 gather/scatter 的优化思路，以及 VAE 的 tiling/blend 机制，都是高质量的设计参考。建议重点阅读 minimax_h3.py、klvae.py、denoise_loop.py 与 material_io.py。

功能与动机

MiniMax-H3 是面向联合音视频生成的新模型，需要原生 serving 支持以提供 FL2VA（第一帧到视频）、Ref2VA（参考媒体到视频）和 T2VA（文本到视频）等任务能力。PR body 明确说明目标是“add a fully native joint video/audio generation pipeline with text, endpoint-frame, and reference-media conditioning”，并强调“integrate distributed execution, mixed-precision loading, caching, and online quantization hooks”，同时需要“focused runtime contracts”和“PR-blocking GPU coverage”以保证交付质量。

实现拆解

1. 模型实现：在 python/sglang/multimodal_gen/runtime/models/ 下新增 MiniMax-H3 的 DiT 模型（dits/minimax_h3.py）、视频 VAE（vaes/minimax_h3_video_vae/）与音频 VAE（vaes/minimax_h3_audio_vae/）。DiT 采用 packed-token 契约输入，严格校验 forward 关键字；VAE 实现 3D causal CNN 编码器与 ViT3D 解码器、DAC-lineage 音频编解码器，并支持 tiling 与 tile 并行解码。
2. 管线编排：在 pipelines_core/stages/model_specific_stages/minimax_h3/ 下新增一整套模型特定阶段，包括 text_encoding.py、visual_encoding.py、denoising.py、decoding.py 等，并通过 resolved_plan.py、packed_sequence.py、denoise_loop.py 组织请求级执行计划、打包序列构造与 CFG-distilled 单分支去噪循环。denoise_loop.py 将参考行与目标行分离，每步只更新目标行以避免重复 gather/scatter。
3. 请求适配与交付：video_adapter.py 实现 OpenAI 视频 API 的 lowering 与严苛校验（拒绝 fps、num_frames、CFG 字段等），material_io.py 负责 base64/tar URI 的本地化与临时目录管理，保证请求级生命周期。
4. 集成与部署：新增 Docker 构建（docker/ 下相关配置）、CI 工作流调整（如 fix(ci) 系列提交）、平台检测修复（Fix non-NVML CUDA platform detection）以及环境变量控制的

混合精度与确定性选项（如 MINIMAX_H3_VAE_DECODER_TEMPORAL_CAT_DTYPE）。

5. 测试与文档：配套新增 / 调整至少 26 个测试文件，覆盖模型单元契约、VAE 编解码、任务校验等；文档侧新增 docs_new/cookbook/diffusion/ 部署 cookbook 与配置 snippets。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 扩散模型；类别 source；类型 data-contract；符号 _required_kwarg, _ulysses_ctx, _ring_world_size, _reorder_grouped_qkv_to_qkv）：MiniMax-H3 核心 DiT 模型实现，定义 packed-token forward 契约、TP/ 序列并行辅助函数、QK-norm/RoPE/ 调制等关键算子，是整个生成管线的执行核心。
- python/sglang/multimodal_gen/runtime/models/vaes/minimax_h3_video_vae/klvae.py（模块 视频 VAE；类别 source；类型 data-contract；符号 _resolve_temporal_cat_dtype, _resolve_temporal_stream_cat, get_tile_parallel_state, DiagonalGaussianDistribution）：MiniMax-H3 视频 VAE 的核心推理类，封装时空 chunking、tiling、encode/decode 入口与 tile 并行状态，是视频编解码的关键路径。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/denoising.py（模块 去噪阶段；类别 source；类型 data-contract；符号 minimax_h3_condition_noise_aug, _validate_fl2va_keyframe_payload, _imgvid_condition_shapes, _ref2va_payload_entry）：MiniMax-H3 去噪 sink 与 payload 校验，处理 CFG-distilled 单分支执行、fl2va keyframe 验证、ref2va 参考块排序等任务级逻辑。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/denoise_loop.py（模块 去噪循环；类别 source；类型 data-contract；符号 minimax_h3_update_target_rows, _ulysses_ctx, _build_local_embedding_layout, local_ids）：实现 CFG-distilled 全量去噪循环，预计算静态 packed 布局与 forward kwargs，避免热点循环中的 gather/scatter 与索引重算。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/packed_sequence.py（模块 打包序列；类别 source；类型 data-contract；符号 _keyframe_cond_frame_indices, _resolve_keyframe_frame_indices, _temporal_position_span, minimax_h3_packed_sequence）：负责从 validated workspace 构造 packed 序列（text/cond/audio/video/pad），含 fp64 位置网格与 token tags，是模型 forward 的输入基础。
- python/sglang/multimodal_gen/runtime/models/vaes/minimax_h3_audio_vae/audio_vae.py（模块 音频 VAE；类别 source；类型 data-contract；符号 GeGluMlp, init, forward, CausalAttention）：新增 DAC-lineage 音频 VAE（波形编码器 + BigVGAN 解码器），支持 16k/32k 采样率配置，是音频条件与目标合成的基础。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/video_adapter.py（模块 API 适配；类别 source；类型 data-contract；符号 _extra_value, _parse_extra_value, _format_video_seconds, MiniMaxH3VideoModelAdapter）：OpenAI 视频 API 的模型适配层，实现任务门控、严格字段校验、CFG 字段拒绝与 delivery 契约，是服务端入口的关键防线。

- python/sglang/multimodal_gen/runtime/models/vaes/minimax_h3_video_vae/processor.py（模块 VAE 预处理；类别 source；类型 data-contract；符号 get_norm_constants, get_normalize_transform, get_denormalize_transform, VAEProcessor）：VAE 张量预处理 / 后处理，负责像素归一化、对齐裁剪、帧长对齐与 numpy 到 tensor 的紧凑转换。

关键符号：MiniMaxH3DiTModel.forward, MiniMaxH3Rope.forward, _reorder_grouped_qkv_to_qkv, _copy_grouped_qkv_tp_shard, AutoencoderKL.setup_forward, AutoencoderKL.split_tiles, AutoencoderKL.blend, minimax_h3_packed_sequence, MiniMaxH3DenoiseBranch.forward_kwargs, MiniMaxH3VideoModelAdapter.lower_video_request_kwargs, VAEProcessor.transform_tensor, _AudioVAEDeterminismContext.enter

关键源码片段

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/packed_sequence.py

负责从 validated workspace 构造 packed 序列（text/cond/audio/video/pad），含 fp64 位置网格与 token tags，是模型 forward 的输入基础。

```
# MiniMax H3 packed-sequence 构造：布局为 [text L | imgvid_cond C | audio A(=t*2ch) | video_target V | pad P]。
# 位置网格使用 fp64，视频 t 计数从 text_len 延续，空间每轴均匀分布并排除右端点。
def minimax_h3_packed_sequence(
    *,
    text_len: int,
    latent_t: int,
    latent_h: int,
    latent_w: int,
    audio_t: int,
    audio_channel: int = 2,
    include_keyframe_cond: bool,
    keyframe_frame_indices: list[int] | tuple[int, ...] | None = None,
    frame_count: int | None = None,
) -> dict[str, Any]:
    ph, pw = latent_h // _PATCH_H, latent_w // _PATCH_W
    frame_rows = ph * pw
    cond_frame_indices = _keyframe_cond_frame_indices(
        include_keyframe_cond=include_keyframe_cond,
        keyframe_frame_indices=keyframe_frame_indices,
    )
    resolved_cond_frame_indices = _resolve_keyframe_frame_indices(
        cond_frame_indices,
        frame_count=frame_count,
    )
    cond_rows = len(cond_frame_indices) * frame_rows
    video_rows = latent_t * frame_rows
    audio_rows = audio_t * audio_channel
    used = text_len + cond_rows + audio_rows + video_rows
```

```
# 总长度对齐到 64
seq_len = (
    (used + MINIMAX_H3_PACKED_SEQUENCE_ALIGNMENT - 1)
    // MINIMAX_H3_PACKED_SEQUENCE_ALIGNMENT
    * MINIMAX_H3_PACKED_SEQUENCE_ALIGNMENT
)

text_sl = slice(0, text_len)
cond_sl = slice(text_len, text_len + cond_rows)
audio_sl = slice(cond_sl.stop, cond_sl.stop + audio_rows)
video_sl = slice(audio_sl.stop, audio_sl.stop + video_rows)
target_img_pos = torch.arange(video_sl.start, video_sl.stop)
img_pos = (
    torch.cat([torch.arange(cond_sl.start, cond_sl.stop), target_img_pos])
    if cond_rows
    else target_img_pos
)
update_mask = torch.zeros(img_pos.shape[0], dtype=torch.bool)
update_mask[cond_rows:] = True
audio_pos = torch.arange(audio_sl.start, audio_sl.stop)
text_pos = torch.arange(0, text_len)

# 位置网格 g: [seq_len, 3] fp64，三列分别是 t、h、w 坐标
g = torch.zeros(seq_len, 3, dtype=torch.float64)
g[text_sl, 0] = torch.arange(text_len, dtype=torch.float64)

t_grid = _video_t_grid(latent_t, float(text_len))
sqrt_area = np.sqrt(latent_h * latent_w)
h_grid = _axis_from_sqrt_area(latent_h, _PATCH_H, sqrt_area)
w_grid = _axis_from_sqrt_area(latent_w, _PATCH_W, sqrt_area)
hh, ww = torch.meshgrid(h_grid, w_grid, indexing="ij")
frame = torch.stack([hh.reshape(-1), ww.reshape(-1)], dim=-1)
video_g = g[video_sl].view(latent_t, frame_rows, 3)
video_g[:, :, 0] = t_grid[:, None]
video_g[:, :, 1:] = frame[None]
...

# token tags：1 文本、2 音频、0 视频、-1 填充
token_tags = torch.full((seq_len,), -1, dtype=torch.long) # PADDING
token_tags[text_sl] = 1 # TEXT
token_tags[audio_sl] = 2 # AUDIO
token_tags[img_pos] = 0 # VIDEO

cu = torch.tensor([0, used, seq_len], dtype=torch.int32)
return {
    "seq_len": seq_len,
    "img_pos": img_pos,
    "audio_pos": audio_pos,
    "text_pos": text_pos,
```

```
        "update_mask": update_mask,
        "img_position_ids": g,
        "token_tags": token_tags,
        "cu_seqlens": cu,
    }
```

评论区精华

该 PR 没有人工 review 评论，仅有 bot 自动评论（Mintlify 预览、Gemini Code Assist 停用通知）与作者自行触发的两次 /tag-and-rerun-ci。从提交历史看，作者在合并前完成了多次自检修复，包括 CI 夹具更新、平台检测、TP-Ulysses 通信选择、AdaLN 集合通信优化以及临时禁用不稳定的 4-GPU H100 任务。整体缺乏外部评审的实质讨论，设计决策主要通过代码中的严格契约与注释表达。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 数据契约严格性：minimax_h3.py 中 _FORWARD_SUPPORTED_KWARGS 拒绝未列出的 kwarg，video_adapter.py 对 fps/num_frames/CFG 字段 fail-closed，可能在未来模型版本升级时需要同步调整契约，兼容性风险集中在契约面。
  2. 数值精度与确定性：klvae.py 的 VAE 解码涉及 tiling 重叠区域 blend，setup_forward 中的帧对齐算法复杂度高；reference_encoding.py 中的 _AudioVAEDeterminismContext 通过禁用 cuDNN/TF32 换取确定性，可能显著降低部分环境下的编码性能。
  3. 外部依赖：material_io.py 与 reference_encoding.py 依赖 ffmpeg、Pillow、torchaudio 等外部工具，路径与 URI 解析虽做了校验，但恶意构造的 tar/base64 URI 仍可能触发资源消耗或异常路径。
  4. CI 稳定性：PR 提交历史显示曾临时禁用不稳定的 4-GPU H100 job，且最终 PR Test (Extra) 为失败状态（:x:），新增 GPU 测试的长期稳定性有待观察。
  5. 代码规模：一次性新增 2.2 万行，缺少外部 review 的充分交叉验证，潜在边界条件可能在真实负载下暴露。 - 影响：影响范围集中在新增的 sglang/multimodal_gen 子系统中，对既有 SRT 核心路径无侵入。用户侧可获得 MiniMax-H3 的开箱即用服务与 OpenAI 兼容视频 API；系统侧新增大量模型与管线代码需维护，Docker 镜像与 CI 任务也会增加资源开销；团队侧该 PR 建立了 diffusion 模型集成的参考范式，后续类似模型可复用 model_specific_stages 与 packed_sequence 等基础设施。整体影响程度中等偏大，但风险隔离良好（新模块独立）。 - 风险标记：缺少人工 review 讨论，新增 2.2 万行大规模代码，CI 临时禁用不稳定任务，严格数据契约存在兼容性风险，依赖 ffmpeg 等外部工具

关联脉络

- PR #33277 [CI] Fix stale CPU test fixtures: 本 PR 提交历史中包含 fix(ci): update model override test fixture，与 #33277 修改同类测试夹具文件，属于同一 CI 稳定性修

复线。

- PR #33254 [CI] Add speculative_draft_attention_backend to the page-constraint test view: 同样涉及 test_model_overrides.py 等夹具更新，本 PR 的 CI 修复提交与之一致，反映近期 CI 夹具演进脉络。