PR #22625 完整报告

sgl-project/sglang

[diffusion] model: support JoyAI-Image-Edit

合并时间：2026-05-02 14:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22625

执行摘要

- 一句话：支持 JoyAI-Image-Edit 图像编辑模型
- 推荐动作：推荐关注此 PR 的设计模式：它展示了如何通过配置驱动的方式将新扩散模型集成到 SGLang 框架中，特别是通过管道配置的 postprocess_text_funcs 实现后处理泛化。对模型集成者有参考价值。建议精读 qwen3vl.py 的编码器实现和 joy_image.py（runtime）的 DiT 双流块实现。

功能与动机

PR 由 JoyAI Team 提出，希望基于 SGLang 扩散框架提供图像编辑能力。JoyAI-Image 是一个统一的多模态基础模型，结合了 8B MLLM 和 16B 多模态扩散 Transformer。此 PR 使 SGLang 能够加载并运行 JoyAI-Image-Edit 模型进行指令引导的图像编辑推理。

实现拆解

1. 注册与配置：在 multimodal_gen/registry.py 中添加 JoyAI 模型路径检测和配置映射，新增 JoyImageEditPipelineConfig 和 JoyImageEditSamplingParams。
2. Qwen3-VL 文本编码器：新增 qwen3vl.py 配置文件（Qwen3VLArchConfig/Qwen3VLConfig）和运行时模型文件（runtime/models/encoders/qwen3vl.py），实现了完整的 Qwen3VLTextAttention、Qwen3VLTextModel、Qwen3VLModel 等模块，用于文本和视觉条件编码。
3. JoyImage DiT 模型：新增 joy_image.py 配置文件（JoyImageArchConfig/JoyImageDiTConfig）和运行时模型文件（runtime/models/dits/joy_image.py），包含 MMDoubleStreamBlock、ModulateWan、JoyTransformer3DModel 等，支持序列并行和权重名称映射。
4. 编辑管道：新增 JoyImageEditPipeline 类，通过 create_pipeline_stages 编排标准 T2I 阶段。JoyImageEditPipelineConfig 设置默认采样参数（guidance_scale=4.0, num_inference_steps=40, num_frames=1）。
5. 框架泛化：将图像编码阶段的后处理函数抽象为管道可配置的 postprocess_text_funcs，保留 Qwen-Image-Edit 原有的 drop_idx=64 行为。在 WanVAEConfig 中添加 get_vae_scale_factor() 和 post_init() 统一缩放因子获取。
6. 测试：新增 JoyAI 图像编辑 1 GPU 扩散 CI 烟雾测试，包含性能基准数据。

关键文件：

- python/sglang/multimodal_gen/runtime/models/encoders/qwen3vl.py（模块 编码器；类别 source；类型 data-contract；符号 Qwen3VLTextAttention, init, forward, Qwen3VLTextMLP）：新增的 Qwen3-VL 文本编码器运行时实现，包含 Attention、MLP、DecoderLayer、Model 等完整模块，是 JoyAI 模型理解文本和图像条件的关键组件。
- python/sglang/multimodal_gen/runtime/models/dits/joy_image.py（模块 DiT 模型；类别 source；类型 data-contract；符号 fused_add_gate, ModulateWan, init, forward）：新增的 JoyImage DiT 模型运行时实现，包含双流块 ModulateWan、MMDoubleStreamBlock、JoyTransformer3DModel 等，是生成图像编辑结果的核心扩散骨干。
- python/sglang/multimodal_gen/configs/pipeline_configs/joy_image.py（模块 管道配置；类别 source；类型 dependency-wiring；符号 joy_image_postprocess_text, JoyImageEditPipelineConfig, post_init, slice_noise_pred）：新增的 JoyImage 编辑管道配置，定义了任务类型、组件配置、采样默认值和桶生成策略，是模型集成的入口配置。
- python/sglang/multimodal_gen/configs/models/encoders/qwen3vl.py（模块 编码器配置；类别 source；类型 data-contract；符号 _is_transformer_layer, _is_embeddings, _is_final_norm, Qwen3VLArchConfig）：新增 Qwen3-VL 文本编码器的配置类，定义模型超参数、参数映射和 FSDP 分片条件。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/image_encoding.py（模块 图像编码阶段；类别 source；类型 dependency-wiring；符号 encoding_qwen_image_edit, encoding_image_edit）：泛化了图像编码阶段的后处理函数，使其支持管道配置的函数，同时保留了 Qwen-Image-Edit 原有的 drop_idx=64 行为。

关键符号：Qwen3VLTextAttention.forward, MMDoubleStreamBlock.forward, JoyImageEditPipeline.create_pipeline_stages, joy_image_postprocess_text, JoyImageEditPipelineConfig.post_init

关键源码片段

python/sglang/multimodal_gen/runtime/models/encoders/qwen3vl.py

新增的 Qwen3-VL 文本编码器运行时实现，包含 Attention、MLP、DecoderLayer、Model 等完整模块，是 JoyAI 模型理解文本和图像条件的关键组件。

```
class Qwen3VLTextAttention(nn.Module):
    # 使用 LocalAttention 替代原生 FlashAttention，兼容不同平台
    def __init__(self, config: Qwen3VLTextConfig, layer_idx: int):
        super().__init__()
        self.config = config
        self.layer_idx = layer_idx
        self.head_dim = config.hidden_size // config.num_attention_heads
        self.num_key_value_groups = config.num_attention_heads // config.num_key_value_heads
        self.scaling = self.head_dim ** -0.5
        self.attention_dropout = config.attention_dropout
        self.is_causal = True
        self.num_heads = config.num_attention_heads
        self.num_key_value_heads = config.num_key_value_heads
```

```
        # 标准线性投影
        self.q_proj = nn.Linear(config.hidden_size, config.num_attention_heads * self.head_dim,
        bias=config.attention_bias)
        self.k_proj = nn.Linear(config.hidden_size, config.num_key_value_heads * self.head_dim,
        bias=config.attention_bias)
        self.v_proj = nn.Linear(config.hidden_size, config.num_key_value_heads * self.head_dim,
        bias=config.attention_bias)
        self.o_proj = nn.Linear(config.num_attention_heads * self.head_dim, config.hidden_size,
        bias=config.attention_bias)
        # Q/K 各自独立应用 RMSNorm（区别于共用 norm 的做法）
        self.q_norm = Qwen3VLTextRMSNorm(self.head_dim, eps=config.rms_norm_eps)
        self.k_norm = Qwen3VLTextRMSNorm(self.head_dim, eps=config.rms_norm_eps)
        # 本地注意力层，支持 FA 和 torch_sdpa 后端
        self.attn = LocalAttention(
            num_heads=self.num_heads,
            head_size=self.head_dim,
            num_kv_heads=self.num_key_value_heads,
            softmax_scale=self.scaling,
            causal=True,
            supported_attention_backends=(AttentionBackendEnum.FA, AttentionBackendEnum.
            TORCH_SDPA),
        )

    def forward(self, hidden_states, position_embeddings, attention_mask=None, past_key_
    values=None, cache_position=None, **kwargs):
        input_shape = hidden_states.shape[:-1]
        hidden_shape = (*input_shape, -1, self.head_dim)
        # QKV 投影：Q 和 K 各自带归一化
        query_states = self.q_norm(self.q_proj(hidden_states).view(hidden_shape)).transpose(1, 2)
        key_states = self.k_norm(self.k_proj(hidden_states).view(hidden_shape)).transpose(1, 2)
        value_states = self.v_proj(hidden_states).view(hidden_shape).transpose(1, 2)
        # 应用旋转位置编码（RoPE）
        cos, sin = position_embeddings
        query_states, key_states = apply_rotary_pos_emb(query_states, key_states, cos, sin)
        # 可选：更新 past_key_values 缓存
        if past_key_values is not None:
            cache_kwargs = {"sin": sin, "cos": cos, "cache_position": cache_position}
            key_states, value_states = past_key_values.update(key_states, value_states, self.layer_
            idx, cache_kwargs)
        # 调用 LocalAttention 执行注意力计算
        attn_output = self.attn(query_states, key_states, value_states, attn_mask=attention_mask)
        # 输出投影
        attn_output = attn_output.transpose(1, 2).contiguous().reshape(hidden_states.shape)
        attn_output = self.o_proj(attn_output)
        return attn_output, None
```

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

新增的JoyImageDiT模型运行时实现，包含双流块ModulateWan、MMDoubleStreamBlock、JoyTransformer3DModel 等，是生成图像编辑结果的核心扩散骨干。

```
class MMDoubleStreamBlock(nn.Module):
    # 双流块：图像流和文本流各自独立进行调制、注意力、MLP，然后通过门控融合
    def forward(self, img: torch.Tensor, txt: torch.Tensor, vec: torch.Tensor, pe: torch.Tensor):
        # 从条件向量中切分出调制参数（6 个因子：缩放、平移、门控）
        img_mod1, img_mod2 = self.img_mod(vec).chunk(2, dim=1)
        txt_mod1, txt_mod2 = self.txt_mod(vec).chunk(2, dim=1)
        # 图像流：归一化 + 双流注意力
        img_norm1 = self.fused_modulate_img_norm1(img, img_mod1[:, :3])
        img_attn_out, _ = self.img_attn(img_norm1, pe)
        img = fused_add_gate(img, img_attn_out, img_mod1[:, 3])
        # 文本流：归一化 + 双流注意力
        txt_norm1 = self.fused_modulate_txt_norm1(txt, txt_mod1[:, :3])
        txt_attn_out, _ = self.txt_attn(txt_norm1, pe)
        txt = fused_add_gate(txt, txt_attn_out, txt_mod1[:, 3])
        # 图像 MLP
        img_norm2 = self.fused_modulate_img_norm2(img, img_mod2[:, :3])
        img_mlp_out = self.img_mlp(img_norm2)
        img = fused_add_gate(img, img_mlp_out, img_mod2[:, 3])
        # 文本 MLP
        txt_norm2 = self.fused_modulate_txt_norm2(txt, txt_mod2[:, :3])
        txt_mlp_out = self.txt_mlp(txt_norm2)
        txt = fused_add_gate(txt, txt_mlp_out, txt_mod2[:, 3])
        return img, txt
```

python/sglang/multimodal_gen/configs/pipeline_configs/joy_image.py

新增的 JoyImage 编辑管道配置，定义了任务类型、组件配置、采样默认值和桶生成策略，是模型集成的入口配置。

```
@dataclass
class JoyImageEditPipelineConfig(ImagePipelineConfig):
    task_type: ModelTaskType = ModelTaskType.I2I # 图像到图像编辑
    dit_config: DiTConfig = field(default_factory=JoyImageDiTConfig)
    vae_config: VAEConfig = field(default_factory=WanVAEConfig)
    vae_tiling: bool = False # 编辑场景无需 VAE 分片
    vae_sp: bool = False
    flow_shift: float = 1.5
    # 使用 Qwen3-VL 作为文本 + 视觉编码器
    text_encoder_configs: tuple[EncoderConfig, ...] = field(default_factory=lambda:
    (Qwen3VLConfig(),))
    enable_torch_compile: bool = False
    # 全 bf16 精度
    precision: str = "bf16"
    vae_precision: str = "bf16"
    text_encoder_precisions: tuple[str, ...] = field(default_factory=lambda: ("bf16",))
    # 后处理函数可配置，允许不同编码器使用不同截断策略
    postprocess_text_funcs: tuple[Callable, ...] = field(default_factory=lambda: (joy_image_
```

```
    postprocess_text,))
    prioritize_frame_matching: bool = True
    bucket_configs: list[tuple[int, int, int, int]] = field(init=False)

    def __post_init__(self):
        # 初始化桶配置：1024 分辨率，单帧，批量大小 8，最多 6 项
        self.bucket_configs = self.generate_video_image_bucket(
            basesize=1024, min_temporal=1, max_temporal=1,
            bs_img=8, bs_vid=4, bs_mimg=8, min_items=1, max_items=6,
        )
```

评论区精华

Review 中最关键的讨论是 BBuf 提出的两点：

1) 通用化后处理导致 Qwen-Image-Edit 的 drop_idx 从 64 变为默认 34，可能改变现有模型的条件 token； 2) latents[0] 只取第一个输出的处理会导致批量请求时丢失其他输出。作者均已修复。此外，gemini-code-assist 建议将 WanVAEConfig.post_init 合并到 __post_init__ 并避免 bucket_configs 在请求处理中初始化，也已修复。变量 shadowing 问题（input_ids 和 config）在审查时未明确修复。

- 通用化后处理导致 Qwen-Image-Edit drop_idx 回归 (correctness): 作者恢复 edit 特定分支，使用 drop_idx=64，问题已解决。
- latents[0] 选择导致批量优化丢失 (correctness): 作者对齐了条件 latents 的批量维度，并在多处添加严格维度检查，问题已解决。
- WanVAEConfig.post_init 冗余设计 (design): 作者已将逻辑迁移到 post_init，问题已解决。

风险与影响

- 风险：主要风险在于： 1) 回归风险：对 image_encoding.py 的泛化改动了原有 Qwen-Image-Edit 路径，虽经修复，但可能仍有未覆盖的边角情况。 2) 依赖上游权重格式：JoyAI-Image-Edit 模型权重名称仍在变化（依赖 huggingface/diffusers#13444），合并后如权重变动可能导致加载失败。 3) 测试覆盖率有限：仅添加了烟雾生成测试，未包含单元测试或一致性校验，可能存在未发现的逻辑错误。 4) 批量请求支持虽已修补，但未经过充分验证。
  - 影响：对用户，新增了 JoyAI-Image-Edit 模型支持，可以用于指令引导的图像编辑。对系统，新增约 2300 行代码，主要是新模型实现和配置，对现有框架核心路径改动较小，不引入性能回归。对团队，需要维护新模型代码，并跟踪上游权重变化。
  - 风险标记：泛化改动影响现有路径，依赖上游权重格式，测试覆盖率有限，批量支持验证不足

关联脉络

- 暂无明显关联 PR