PR #24994 完整报告

sgl-project/sglang

[diffusion] model: support a new model

合并时间：2026-05-27 08:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24994

执行摘要

- 一句话：支持 Cosmos3 世界模型，实现 T2V/I2V/T2I 生成
- 推荐动作：值得精读，特别是以下设计决策：
  - 双路径 DiT 的架构（Understanding 与 Generation 的分离）。
  - 权重映射函数的设计，展示了如何将 diffusers 格式转换为自定义并行模型。
  - 安全护栏的可插拔设计。
  - 讨论中关于注意力后端兼容性的取舍。

功能与动机

支持新的世界模型 Cosmos3，该模型尚未公开发布，旨在通过单个检查点同时支持 T2V、I2V 和 T2I 生成任务，以扩展 SGLang 的扩散模型能力。

实现拆解

1. 模型架构：在 python/sglang/multimodal_gen/runtime/models/dits/cosmos3video.py 中实现 Cosmos3OmniTransformer，包含双路径 DiT（Understanding 和 Generation）、mRoPE 位置编码、以及 USPAttention 集成。
2. 管道阶段：在 python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/cosmos3.py 和 cosmos3_guardrails.py 中定义 6 个阶段，包括图像预处理、tokenization、潜在准备、timestep 准备、去噪、解码，以及可选的文本 / 视频安全护栏。
3. 配置与权重映射：在 python/sglang/multimodal_gen/configs/models/dits/cosmos3video.py 中定义模型超参数和权重映射函数 _build_cosmos3_param_names_mapping，将 diffusers 格式 checkpoint 映射到 SGLang 命名空间。
4. 管道编排：在 python/sglang/multimodal_gen/runtime/pipelines/cosmos3_pipeline.py 中组装所有阶段，并通过环境变量控制 guardrails 启用。
5. 集成与测试：修改 registry.py 注册新模型，更新 hf_diffusers_utils.py 和 scheduling_unipc_multistep.py 增加兼容性；添加单元测试 test_cosmos3.py 验证权重映射，并更新文档。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/cosmos3video.py（模块 模型层；类别 source；类型 core-logic；符号 compute_mrope_position_ids_text,

compute_mrope_position_ids_vision, qwen3_rotate_half,qwen3_apply_rotary_pos_emb）：Cosmos3 双路径 DiT 模型的核心实现，包括Understanding 和 Generation 路径、mRoPE 位置编码、以及与 USPAttention 的集成，是 PR 最大的新增代码（+1359 行）。

- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/cosmos3.py（模块 管道阶段；类别 source；类型 core-logic；符号 Cosmos3ImagePreprocessStage, verify_input, forward, Cosmos3TokenizationStage）：定义了 Cosmos3 管道的所有核心阶段，包括图像预处理、tokenization、潜在准备、timestep 准备、去噪和解码，是管道流的核心。
- python/sglang/multimodal_gen/configs/models/dits/cosmos3video.py（模块 模型配置；类别 source；类型 data-contract；符号 is_layers, _build_cosmos3_param_names_mapping, Cosmos3VideoArchConfig, post_init）：定义模型架构配置和权重映射逻辑，将 diffusers 格式的 checkpoint 映射到 SGLang 模型命名空间，是模型加载的关键。
- python/sglang/multimodal_gen/runtime/pipelines/cosmos3_pipeline.py（模块 管道编排；类别 source；类型 core-logic；符号 Cosmos3Pipeline, create_pipeline_stages）：编排管道阶段的创建和组装，用户与模型交互的入口，控制 guardrails 开关。
- python/sglang/multimodal_gen/test/unit/test_cosmos3.py（模块 单元测试；类别 test；类型 test-coverage；符号 _apply, TestCosmos3ParamNamesMapping, setUpClass, test_lm_head_dropped）：单元测试覆盖权重映射函数，确保转换逻辑正确，验证配置文件的正确性。

关键符号：compute_mrope_position_ids_text, compute_mrope_position_ids_vision, qwen3_rotate_half, qwen3_apply_rotary_pos_emb, Qwen3VLTextRotaryEmbedding.init, apply_interleaved_mrope, Cosmos3ImagePreprocessStage.forward, Cosmos3TokenizationStage._tokenize_prompt, SafetyClassifier.forward, _build_cosmos3_param_names_mapping, Cosmos3Pipeline.create_pipeline_stages, adjust_num_frames, _forward_with_replicated_kv_prefix

关键源码片段

python/sglang/multimodal_gen/runtime/models/dits/cosmos3video.py

Cosmos3 双路径 DiT 模型的核心实现，包括 Understanding 和 Generation 路径、mRoPE 位置编码、以及与 USPAttention 的集成，是 PR 最大的新增代码（+1359 行）。

```
def compute_mrope_position_ids_vision(
    grid_t: int, # 时间维度网格大小
    grid_h: int, # 高度网格大小
    grid_w: int, # 宽度网格大小
    temporal_offset: int | float, # 时间偏移量（与文本段衔接）
    device: torch.device,
    fps: float | None = None, # 实际帧率，用于 FPS 调制
    base_fps: float = 24.0, # 基础帧率
    temporal_compression_factor: int = 4, # VAE 时间压缩因子
) -> tuple[torch.Tensor, int | float]:
    """生成视觉 token 的 3D mRoPE 位置 ID（Qwen3VL 风格）。
```

```
创建 (T, H, W) 位置网格。空间索引在每个视觉段重置为 0。
按 T-major 顺序展平。

返回: (position_ids[3, N], next_temporal_offset)
"""
fps_modulation = fps is not None and grid_t > 1

if fps_modulation:
    # 当提供 FPS 且为视频时，按实际帧率调整时间索引
    tps = fps / temporal_compression_factor
    base_tps = base_fps / temporal_compression_factor
    frame_indices = torch.arange(grid_t, dtype=torch.float32, device=device)
    t_index = (
        (frame_indices / tps * base_tps + temporal_offset)
        .view(-1, 1)
        .expand(-1, grid_h * grid_w)
        .flatten()
    )
else:
    # 静态情况，时间索引为整数序列
    t_index = torch.arange(grid_t, dtype=torch.long, device=device).view(
        -1, 1
    ).expand(-1, grid_h * grid_w).flatten() + int(temporal_offset)

# 空间高度索引：沿 H 轴递增
h_index = (
    torch.arange(grid_h, dtype=torch.long, device=device)
    .view(1, -1, 1)
    .expand(grid_t, -1, grid_w)
    .flatten()
)
# 空间宽度索引：沿 W 轴递增
w_index = (
    torch.arange(grid_w, dtype=torch.long, device=device)
    .view(1, 1, -1)
    .expand(grid_t, grid_h, -1)
    .flatten()
)

if fps_modulation:
    # FPS 调制时使用 float32 类型以保持精度
    mrope_ids = torch.stack(
        [t_index, h_index.to(torch.float32), w_index.to(torch.float32)], dim=0
    )
else:
    mrope_ids = torch.stack([t_index, h_index, w_index], dim=0)

next_offset = math.ceil(mrope_ids.max().item()) + 1
return mrope_ids, next_offset
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/cosmos3.py

定义了 Cosmos3 管道的所有核心阶段，包括图像预处理、tokenization、潜在准备、timestep 准备、去噪和解码，是管道流的核心。

```
class Cosmos3ImagePreprocessStage(PipelineStage):
    """加载、缩放并中心裁剪 I2V 条件图像。

    当请求没有图像（T2V / T2I）时自动跳过。
    输出为 [1, 3, H, W] 张量，值域 [-1, 1]，写入 batch.preprocessed_image。
    """
    parallelism_type = StageParallelismType.REPLICATED

    def forward(self, batch: Req, server_args: ServerArgs) -> Req:
        # 获取图像路径，支持列表中的首个图像
        image_path = batch.image_path
        if isinstance(image_path, list):
            image_path = image_path[0] if image_path else None
        if not isinstance(image_path, str) or not image_path:
            return batch # no-op for T2V/T2I

        # 打开并转换为 RGB
        image = PIL.Image.open(image_path).convert("RGB")
        target_h, target_w = batch.height, batch.width
        # 计算缩放比例，保持宽高比填充目标区域
        scale = max(target_w / image.width, target_h / image.height)
        resize_w = int(np.ceil(scale * image.width))
        resize_h = int(np.ceil(scale * image.height))
        image = image.resize((resize_w, resize_h), PIL.Image.Resampling.LANCZOS)
        # 中心裁剪到目标尺寸
        left = (resize_w - target_w) // 2
        top = (resize_h - target_h) // 2
        image = image.crop((left, top, left + target_w, top + target_h))

        # 转换为 tensor 并归一化到 [-1, 1]
        arr = np.asarray(image, dtype=np.float32) / 127.5 - 1.0
        tensor = torch.from_numpy(arr).permute(2, 0, 1).unsqueeze(0).contiguous()

        batch.preprocessed_image = tensor
        self.log_info(f"Preprocessed conditioning image to {target_w}x{target_h}")
        return batch
```

评论区精华

Review 中核心讨论包括：

- mickqian 建议将 guardrails 文件从顶层移动到 model_specific_stages 目录以获得更好的组织，作者已执行。

- mickqian 询问是否可以适配 USPAttention 而非使用原生 torch 注意力，作者完成适配。
- mickqian 关注自定义注意力包装是否与其他注意力后端兼容，作者移除了包装器，回退到架构配置中默认的 _supported_attention_backends。
- 作者识别并移除了一段旧版多 GPU 支持的 artifact。
- 将 guardrails 文件移到 model_specific_stages 目录 (design): 作者同意并移动了文件。
- 适配 USPAttention 而非自定义注意力 (design): 作者完成了适配，整个模型使用 USPAttention。
- 注意力后端兼容性考虑 (design): 作者移除了自定义包装器，回退到架构配置中默认的 _supported_attention_backends，从而兼容更多后端。
- 移除旧版多 GPU 支持 artifact (other): 移除了该 artifact。

风险与影响

- 风险：主要技术风险包括：
  1. 新模型稳定性：Cosmos3 尚未公开发布，checkpoint 未经过充分准确性验证，可能产生异常输出。
  2. 依赖风险：安全护栏阶段依赖 cv2、nltk、better_profanity 和 Hugging Face 模型下载，若环境缺失可能运行时崩溃。
  3. 注意力层修改：python/sglang/multimodal_gen/runtime/layers/attention/layer.py 中修改了 _forward_with_replicated_kv_prefix，可能影响其他使用该路径的模型（如 MOVA）。
  4. 配置耦合：VAE 配置强制覆写了 use_parallel_encode/decode 为 False，若未来 WanVAE 升级可能产生冲突。 - 影响：影响范围限于 sglang/multimodal_gen 扩散模块。用户可以通过新的 Cosmos3Pipeline 调用 T2V/I2V/T2I 生成。修改了 registry.py 和通用组件（attention layer、scheduler、diffusers utils），但保持向后兼容。不影响核心推理引擎和其他非 diffusion 模型。 - 风险标记：新模型未经过大量准确性验证，安全护栏依赖外部组件（cv2/nltk/better_profanity），修改了基础 attention 层可能影响其他模型，使用未发布的模型 checkpoint

关联脉络

- PR #21544 [diffusion][mova]Enhance cfg parallel for mova and update CI configuration: COSMOS3 和 MOVA 都是扩散模型，架构和集成模式相似，可参考 MOVA 的管道设计和测试方式。