PR #20863 完整报告

sgl-project/sglang

[Diffusion] Add mixed-resolution benchmark support (for #20762)

合并时间：2026-04-22 14:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/20863

执行摘要

- 一句话：为扩散模型 benchmark 工具添加混合分辨率请求支持，提升测试真实性。
- 推荐动作：该 PR 值得精读，特别是 RandomDataset 中的权重采样实现和 per-request 参数传递的设计，展示了如何扩展 benchmark 工具以支持多样化负载。关注点包括：权重采样算法（random.choices 的使用）、接口变更（generate_batch 从字典到列表）的处理、以及混合分辨率下像素吞吐量的正确计算方式。

功能与动机

根据 PR body，动机是 'To test server with different prompts sizes'，即测试服务器在不同提示大小（如图像分辨率）下的表现，以更真实地模拟实际使用场景。

实现拆解

1. 添加 CLI 参数：在 bench_offline_throughput.py 和 bench_serving.py 中新增 --random-request-config（JSON 字符串定义请求配置文件，包含 width、height、num_inference_steps 和 weight 字段）和 --random-request-seed（随机种子）参数，用于接收混合分辨率配置。
2. 核心逻辑实现：在 datasets.py 的 RandomDataset 类中，解析 JSON 配置，使用 random.choices 基于权重采样请求配置文件，并新增 get_sampling_params 方法提供 per-request 参数；同时修改 __getitem__ 以根据采样配置覆盖默认参数。
3. benchmark 逻辑调整：修改 bench_offline_throughput.py 的 generate_batch 函数，使其接受 per-request 采样参数列表（原为单个字典），并调整 calculate_metrics 以正确计算混合分辨率下的像素吞吐量（累加每个请求的实际像素）。
4. HTTP 请求更新：在 bench_serving.py 中，确保 num_inference_steps 等参数被包含在图像和视频生成的 JSON payload 中，以传递给服务器。
5. 验证与错误处理：添加验证逻辑，确保 --random-request-config 仅与 --dataset random 一起使用，避免与其他数据集（如 VBenchDataset）冲突，并处理 JSON 解析异常。

关键文件：

- python/sglang/multimodal_gen/benchmarks/datasets.py（模块 数据集模块；类别 source；类型 core-logic；符号 get_sampling_params）：核心逻辑变更，实现了混合分辨率请求的权重采样和 per-request 参数管理，新增 get_sampling_params 方法，是功能实现的关键。

- python/sglang/multimodal_gen/benchmarks/bench_offline_throughput.py（模块 离线吞吐量；类别 source；类型 core-logic；符号 generate_batch, calculate_metrics）：关键入口文件，添加了混合分辨率支持的 CLI 参数，并修改了 generate_batch 和 calculate_metrics 函数以处理 per-request 参数和正确计算像素吞吐量。
- python/sglang/multimodal_gen/benchmarks/bench_serving.py（模块 服务端测试；类别 source；类型 core-logic）：服务端 benchmark 脚本，添加了相同的 CLI 参数并确保 num_inference_steps 等参数被包含在 HTTP 请求中，以支持混合分辨率负载测试。

关键符号：get_sampling_params, generate_batch, calculate_metrics

关键源码片段

python/sglang/multimodal_gen/benchmarks/datasets.py

核心逻辑变更，实现了混合分辨率请求的权重采样和 per-request 参数管理，新增 get_sampling_params 方法，是功能实现的关键。

```
import random
import json

class RandomDataset(BaseDataset):
    def __init__(self, args, api_url: str = "", model: str = ""):
        super().__init__(args, api_url, model)
        self.num_prompts = args.num_prompts or 100
        self.random_request_config = args.random_request_config # 直接访问属性，避免 getattr
        if self.random_request_config:
            # 解析 JSON 配置字符串
            self.random_request_config = json.loads(self.random_request_config)
            # 使用 pop 提取权重并同时从配置中移除 weight 键，提高效率
            weights = [p.pop("weight") for p in self.random_request_config]
            seed = args.random_request_seed
            rng = random.Random(seed) # 使用指定种子确保可重现性
            # 基于权重采样请求配置文件，采样数量等于请求数
            self._sampled_requests = rng.choices(
                self.random_request_config, weights=weights, k=self.num_prompts
            )
        else:
            self._sampled_requests = None # 未配置时保持原行为

    def get_sampling_params(self, idx: int) -> dict:
        """返回每个请求的采样配置字典，若无混合配置则返回空字典。"""
        if self._sampled_requests:
            return self._sampled_requests[idx]
        return {}

    def __getitem__(self, idx: int) -> RequestFuncInput:
        profile = self._sampled_requests[idx] if self._sampled_requests else {}
        # 根据采样配置覆盖默认参数，支持 width、height、num_inference_steps 等字段
        return RequestFuncInput(
```

```
            prompt=f"Random prompt {idx} for benchmarking diffusion models",
            api_url=self.api_url,
            model=self.model,
            width=profile.get("width", self.args.width),
            height=profile.get("height", self.args.height),
            num_frames=profile.get("num_frames", self.args.num_frames),
            num_inference_steps=profile.get(
                "num_inference_steps", self.args.num_inference_steps
            ),
            fps=profile.get("fps", self.args.fps),
        )
```

python/sglang/multimodal_gen/benchmarks/bench_offline_throughput.py

关键入口文件，添加了混合分辨率支持的 CLI 参数，并修改了 generate_batch 和 calculate_metrics 函数以处理 per-request 参数和正确计算像素吞吐量。

```
def generate_batch(
    engine: DiffGenerator,
    bench_args: BenchArgs,
    prompts: List[str],
    user_sampling_params: List[Dict[str, Any]], # 改为 per-request 参数列表
) -> BatchOutput:
    """同步生成一批图像/视频，支持混合分辨率请求。"""
    assert len(user_sampling_params) == len(prompts), (
        f"user_sampling_params length ({len(user_sampling_params)}) must match "
        f"prompts length ({len(prompts)})"
    ) # 添加断言确保参数长度匹配，防止错误
    output = BatchOutput()
    start_time = time.perf_counter()
    torch.cuda.reset_peak_memory_stats()
    for prompt, params in zip(prompts, user_sampling_params): # 遍历每个请求及其参数
        try:
            sampling_params_kwargs = dict(params) # 复制 per-request 参数
            sampling_params_kwargs["prompt"] = prompt
            result = engine.generate(sampling_params_kwargs=sampling_params_kwargs)
            # ... 处理结果
        except Exception as e:
            # ... 错误处理
    output.latency = time.perf_counter() - start_time
    return output

def calculate_metrics(
    outputs: List[BatchOutput],
    total_duration: float,
    resolution: Tuple[int, int, int],
    num_requests: int,
    all_sampling_params: Optional[List[Dict[str, Any]]] = None, # 新增参数用于混合分辨率
) -> Dict[str, Any]:
    """计算生成指标，支持混合分辨率下的像素吞吐量。"""
```

```
    successful = [o for o in outputs if o.success]
    num_success = len(successful)
    width, height, frames = resolution
    if all_sampling_params: # 如果提供了 per-request 参数，则累加每个请求的实际像素
        total_pixels = sum(
            p.get("width", width)
            * p.get("height", height)
            * p.get("num_frames", frames)
            for p in all_sampling_params[:num_success]
        )
    else: # 否则使用全局分辨率计算
        total_pixels = num_success * width * height * frames
    metrics = {
        "num_requests": num_requests,
        "successful_requests": num_success,
        "request_throughput": num_success / total_duration,
        "pixel_throughput": total_pixels / total_duration, # 正确计算混合分辨率下的像素吞吐量
    }
    return metrics
```

评论区精华

- 权重提取效率：gemini-code-assist[bot] 建议使用 p.pop("weight") 替代先提取权重再移除键的双重迭代，以提高代码效率和简洁性，此建议被采纳。
- 避免 getattr/setattr：ping1jing2 指出应避免使用 getattr/setattr 以符合 SGLang 代码风格，作者改为直接属性访问（args.random_request_config）。
- num_inference_steps 传递问题：Makcum888e 报告服务器未使用配置的 num_inference_steps，作者修复了 bench_serving.py 中 HTTP payload 的构建，确保参数正确传递。
- 兼容性与像素计算：Ratish1 指出 bench_offline_throughput 缺少 random_request_config 字段导致崩溃，以及混合分辨率下像素吞吐量计算不准确；作者添加了字段、验证逻辑并修正了 calculate_metrics 中的像素累加逻辑。所有问题在后续提交中解决，PR 最终获得批准。
  - 权重提取效率优化 (performance): 建议被采纳，作者在后续提交中修改为 weights = [p.pop('weight') for p in self.random_request_config]。
  - 避免使用 getattr/setattr (style): 作者改为直接属性访问 self.random_request_config = args.random_request_config。
  - num_inference_steps 未正确传递 (correctness): 作者修复了 bench_serving.py，在图像和视频生成的 JSON payload 中添加了 num_inference_steps 字段。
  - 兼容性与像素计算问题 (correctness): 作者添加了字段和验证逻辑，并修正了 calculate_metrics 以累加每个请求的实际像素。

风险与影响

- 风险：

- 回归风险：bench_offline_throughput.py 中 generate_batch 的接口从接受单个 user_sampling_params 字典改为列表，可能影响现有调用，但该函数仅在模块内部使用，且添加了断言确保参数长度匹配。
- 性能风险：RandomDataset 初始化时使用 random.choices 进行权重采样，复杂度 O(n)，对于大量请求（如数万）可能轻微影响初始化时间，但 benchmark 中请求数通常有限（默认 100）。
- 安全风险：解析用户提供的 JSON 字符串（--random-request-config），若输入恶意或畸形 JSON 可能导致解析异常，现有代码使用 json.loads 但未显式捕获异常，依赖上层错误处理。
- 兼容性风险：新增参数为可选，默认行为不变，向后兼容；但若用户错误地将 --random-request-config 与非 random 数据集结合使用，会触发验证错误。

- 影响：
  - 对用户：benchmark 用户现在可以定义混合分辨率请求配置文件，模拟真实世界中的多样化负载，获得更准确的性能数据（如吞吐量、延迟），有助于评估服务器在不同工作负载下的表现。
  - 对系统：仅影响扩散模型 benchmark 工具（sglang/multimodal_gen/benchmarks/ 模块），不改变核心扩散模型推理逻辑或服务器运行时，对生产系统无直接影响。
  - 对团队：提升了测试工具的灵活性和真实性，支持更复杂的性能测试场景，有助于开发团队进行负载测试和性能调优；代码变更集中在 benchmark 模块，易于维护。
  - 风险标记：接口变更，JSON 解析风险，缺少单元测试

关联脉络

- PR #23443 [diffusion] ci: allow using prebuilt sgl-kernel wheel for GT regeneration: 同属 diffusion 模块的 CI 相关 PR，涉及扩散模型测试工具链的优化，与本 PR 的 benchmark 功能互补。
- PR #23422 [diffusion] support custom output folder name in GT generation workflow: 同属 diffusion 模块的 CI 工作流改进，展示了扩散模型测试工具的持续演进，与本 PR 的 benchmark 扩展相关。