PR #26247 完整报告

sgl-project/sglang

[diffusion] Fix diffusion serve warmup defaults

合并时间：2026-05-27 15:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26247

执行摘要

- 一句话：将 diffusion serve warmup 从请求预热改为服务器预热，并修复多模型边缘情况
- 推荐动作：值得精读。该 PR 展示了服务端预热的设计权衡：如何在不影响用户请求的前提下确保缓存就绪。should_include_warmup_image 的策略和 get_model_sampling_defaults 的回退机制是值得关注的模式。同时修复了多个模型特定的兼容问题，是理解 sglang diffusion 架构的好入口。

功能与动机

之前的 warmup 基于首请求复制导致服务器在 negative cache 构建前就报告就绪，第一个用户请求承担 warmup 延迟。需要通过服务器内部 warmup 来确保缓存预热后再响应外部请求。

实现拆解

1. 新增 server_warmup.py 模块：提供 build_warmup_reqs、get_first_generation_req、is_warmup_req 等工具函数，以及 get_model_sampling_defaults 获取模型默认采样参数，prepare_warmup_image_path 生成最小图像用于图像任务预热。
2. 修改 http_server.py 启动流程：在 FastAPI lifespan 中创建 asyncio.Event 作为 warmup 完成标志；启动后台任务 _run_server_warmup_after_http_ready，该任务首先轮询 /health 直到就绪，然后根据 should_include_warmup_image 决定是否准备图像，最后通过 build_warmup_reqs 构造 warmup 请求并经由 scheduler 转发，完成后设置 Event 并打印启动日志。同时定义 SERVER_WARMUP_BYPASS_PATHS，在 warmup 完成前只允许健康检查等轻量接口。
3. 重构 scheduler.py 中的 warmup 处理：将原先的 _first_generation_req、_is_warmup_item 等内联方法迁移到 server_warmup.py；_dispatch_single_request 中增加对 is_server_based_warmup 和 should_return_warmup_result 的判断，server-based warmup 的响应会被直接返回到 HTTP 层以确认完成。
4. 修改 server_args.py：新增 server_warmup 内部模式参数，在 sglang serve CLI 中默认启用（warmup=true 且 server_warmup=true），并保证显式 --warmup false 时同时禁用服务器 warmup；disagg_role 会自动禁用服务器 warmup。
5. 修复多个模型的兼容性问题：
   - 设置非空占位 prompt（"warmup"）而非空字符串，避免 Qwen-Image-Edit 处理 None。

- 修改 ImageVAEEncodingStage.retrieve_latents 支持 Flux2 等 VAE 返回的 latent/latents 键。
- 增加 preprocess_vae_encode 钩子允许 Flux2 Finetuned 模型在 VAE 编码前预处理。
- 细化 should_include_warmup_image 策略：仅当任务需要图像输入或明确指示时才附加图像，避免 TI2I 任务漏预热或 TI2V 任务错误缺失。

6. 配套测试：test_cfg_parallel_warmup.py 新增 server-based warmup 的单测；test_server_args.py 新增 CLI 参数解析测试，验证 --warmup 与 server_warmup 的联动。

关键文件：

- python/sglang/multimodal_gen/runtime/server_warmup.py（模块 预热模块；类别 source；类型 dependency-wiring；符号 get_first_generation_req, is_warmup_req, is_server_based_warmup, should_return_warmup_result）：新增的预热核心模块，定义了 warmup 请求构造、图像准备、采样默认值获取等公用函数。
- python/sglang/multimodal_gen/runtime/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _first_generation_req, _is_warmup_item, _log_warmup_result）：调度器主循环修改：导入并使用 server_warmup.py 工具，调整 warmup 请求的派发与日志。
- python/sglang/multimodal_gen/runtime/entrypoints/http_server.py（模块 服务入口；类别 source；类型 dependency-wiring；符号 _wait_until_http_ready, _run_server_warmup_after_http_ready, wait_for_server_warmup）：HTTP 服务入口：启动后台 warmup 任务，定义 warmup 完成前的路由屏障。
- python/sglang/multimodal_gen/runtime/server_args.py（模块 启动配置；类别 source；类型 core-logic）：服务器参数解析：新增 server_warmup 内部标志，控制 CLI 默认启用 / 禁用。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/image_encoding.py（模块 图像编码；类别 source；类型 core-logic；符号 scale_and_shift_encode_latents）：修复 VAE 编码阶段对不同输出格式的兼容性，支持 Flux2 等模型的 latents 属性。
- python/sglang/multimodal_gen/test/unit/test_cfg_parallel_warmup.py（模块 测试；类别 test；类型 test-coverage；符号 test_server_based_warmup_uses_model_default_negative_prompt, test_server_based_warmup_uses_model_default_resolution, test_server_based_warmup_keeps_lightweight_image_fallback, test_warmup_image_inclusion_policy_all_task_types）：新增 server-based warmup 单测，覆盖模型默认 negative prompt、分辨率、图像包含策略等。
- python/sglang/multimodal_gen/test/unit/test_server_args.py（模块 测试；类别 test；类型 test-coverage；符号 test_serve_cli_preserves_config_and_dynamic_unknown_args, test_serve_cli_defaults_warmup_on, test_serve_cli_preserves_explicit_warmup_false, test_serve_cli_preserves_config_warmup_false）：新增 CLI 参数解析测试，验证 --warmup 与 server_warmup 的联动行为。

关键符号：_run_server_warmup_after_http_ready, build_warmup_reqs, get_model_sampling_defaults, should_include_warmup_image, scale_and_shift_encode_latents, get_first_generation_req

关键源码片段

python/sglang/multimodal_gen/runtime/server_warmup.py

新增的预热核心模块，定义了 warmup 请求构造、图像准备、采样默认值获取等公用函数。

```
# python/sglang/multimodal_gen/runtime/server_warmup.py
# 预热请求构建与识别工具

from copy import copy
from typing import Any
from sglang.multimodal_gen.configs.sample.sampling_params import SamplingParams
from sglang.multimodal_gen.runtime.pipelines_core.schedule_batch import Req
from sglang.multimodal_gen.runtime.server_args import ServerArgs


def get_first_generation_req(req_or_group: Any) -> Req | None:
    """从单个 Req 或 Req 列表中提取第一个 Req，用于后续的 warmup 判断。"""
    if isinstance(req_or_group, Req):
        return req_or_group
    if isinstance(req_or_group, list) and req_or_group:
        first_req = req_or_group[0]
        if isinstance(first_req, Req):
            return first_req
    return None


def is_warmup_req(req_or_group: Any) -> bool:
    """判断是否为预热请求（包括服务器预热和请求级预热）。"""
    req = get_first_generation_req(req_or_group)
    return req.is_warmup if req is not None else False


def is_server_based_warmup(req_or_group: Any) -> bool:
    """判断是否为服务器内部预热请求（通过 extra.server_based_warmup 标记）。"""
    req = get_first_generation_req(req_or_group)
    return req is not None and req.is_warmup and bool(req.extra.get("server_based_warmup"))


def get_model_sampling_defaults(server_args: ServerArgs) -> SamplingParams:
    """优先从注册的 pipeline config 类获取默认采样参数，失败后从预训练模型加载。"""
    pipeline_class_name = server_args.pipeline_class_name
    try:
        if pipeline_class_name:
            config_classes = get_pipeline_config_classes(pipeline_class_name)
            if config_classes is not None:
                _, sampling_params_cls = config_classes
                return sampling_params_cls()
        return SamplingParams.from_pretrained(
            server_args.model_path,
```

```
            backend=server_args.backend,
            model_id=server_args.model_id,
        )
    except Exception:
        logger.debug("Falling back to base SamplingParams for server warmup")
        return SamplingParams()
```

python/sglang/multimodal_gen/runtime/managers/scheduler.py

调度器主循环修改：导入并使用 server_warmup.py 工具，调整 warmup 请求的派发与日志。

```
# python/sglang/multimodal_gen/runtime/managers/scheduler.py
# 调度器主循环中的 warmup 处理（部分）
from sglang.multimodal_gen.runtime.server_warmup import (
    build_warmup_reqs,
    get_first_generation_req,
    is_server_based_warmup,
    is_warmup_req,
    should_include_warmup_image,
    should_return_warmup_result,
)

class Scheduler(SchedulerDisaggMixin):
    def _dispatch_single_request(self, req_or_group: Any) -> OutputBatch:
        # ... 常规请求处理 ...
        if is_warmup_req(req_or_group):
            warmup_result = self._handle_warmup(req_or_group)
            # 如果是 server-based warmup 需要返回结果，则直接返回
            if should_return_warmup_result(req_or_group):
                return warmup_result
            return OutputBatch() # 否则不返回内容
        # ... 其他 handler ...
```

python/sglang/multimodal_gen/runtime/entrypoints/http_server.py

HTTP 服务入口：启动后台 warmup 任务，定义 warmup 完成前的路由屏障。

```
# python/sglang/multimodal_gen/runtime/entrypoints/http_server.py
# 服务器 warmup 启动逻辑

SERVER_WARMUP_BYPASS_PATHS = ("/health", "/health_generate", "/model_info", "/server_info")

async def _run_server_warmup_after_http_ready(
    server_args: ServerArgs, warmup_done: asyncio.Event
) -> None:
    """等待 HTTP 就绪，然后发送内部 warmup 请求。"""
    try:
        # 如果 warmup 被禁用或 warmup_resolutions 已指定（走静态预热），则直接标记完成
        if not server_args.warmup or not server_args.server_warmup or server_args.warmup_
        resolutions is not None:
            warmup_done.set()
```

```
            return

        await _wait_until_http_ready(server_args) # 轮询 /health 直到 200

        # 根据任务类型决定是否需要准备预热图像
        warmup_input_path = None
        if should_include_warmup_image(server_args, server_based_warmup=True):
            warmup_input_path = await prepare_warmup_image_path(server_args)

        # 构建预热请求（采用模型默认采样参数，标记为 server_based_warmup）
        warmup_reqs = build_warmup_reqs(
            server_args,
            warmup_resolutions=None,
            warmup_input_path=warmup_input_path,
            return_warmup_result=True,
            server_based_warmup=True,
            use_model_sampling_defaults=True,
        )
        for req in warmup_reqs:
            response = await async_scheduler_client.forward(req)
            if response.error is not None:
                raise RuntimeError(response.error)

        logger.info("The server is fired up and ready to roll!")
        warmup_done.set()
    except asyncio.CancelledError:
        raise
    except Exception as e:
        logger.error("Server warmup failed; aborting startup: %s", e, exc_info=True)
        os.kill(os.getpid(), signal.SIGTERM)
```

评论区精华

PR body 记录了详细的问题发现和决策过程，核心讨论点包括：

- 为什么改为服务器预热：首请求拷贝方式导致第一个用户请求性能异常，且服务器 readiness 信号与缓存就绪脱节。
- 性能基准阈值矛盾：Denoise Step 0 在服务器预热下不再是稳态信号，因此放宽了该步的 perf guard，转而依赖 *e2e* 和 median 指标。
- 图像输入策略反复：should_include_warmup_image 经过多次迭代，从全量图像到尽量轻量，最终按任务类型判断；TI2I 需要图像预热 TI2I 路径，但 TI2V 需要图像预热以避免图像编码冷启动。
- VAE 编码兼容性：Flux2 自定义 VAE 返回的 latent/latents 与标准 DiagonalGaussianDistribution 不同，需要额外属性检查。
    - 为何放弃 request-based warmup (design): 改为 server-based warmup，在 /health 就绪后自动触发，完成后才允许外部生成请求。

- 性能基线阈值调整 (performance): 步级别阈值放宽与 e2e 指标并行，长期需更精细的 warmup 检测机制。
- 图像预热包含策略 (design): 实现 should_include_warmup_image 函数，按 requires_image_input()、accepts_image_input() 及 TI2I/TI2V 具体标记决定。
- Flux2 VAE 输出格式兼容性 (bugfix): 修改 retrieve_latents 优先检查 latent_dist、latent、latents 属性，保证兼容。

风险与影响

- 风险：
  1. 性能回归风险：服务器预热增加启动延迟（~1 秒），但避免了首请求的冷启动损失；宽松的 Denoise Step 0 阈值可能掩盖真实性能问题。
  2. 多模型兼容性：不同 pipeline 的输入要求（如 Qwen 编辑需非空 prompt、Flux2 需特殊 VAE 预处理）当前以临时补丁解决，未来新模型可能再次引发类似问题。
  3. 测试依赖：perf guard 的 _validate_denoise_steps 逻辑调整后，可能无法捕获某些特定步数的退化。
  4. 分布式场景：disagg 角色已禁用服务器 warmup，但 mixed 模式启动顺序可能仍需验证。
  - 影响：用户影响：所有使用 sglang serve 启动的 diffusion 模型服务将经历约 1-2 秒的启动延迟（等待 warmup），但首次生成请求的延迟显著降低，因为 negative text embedding 缓存、VAE 等已预热。

系统影响：新增 ServerArgs.server_warmup 内部参数，通过 CLI 启动时默认启用；对显式使用 --warmup false 的配置保持兼容。

团队影响：引入新的模块 server_warmup.py，将 warmup 逻辑从 scheduler 抽离，便于后续维护和测试；Perf CI 的阈值需要与 warmup 行为绑定，未来修改 warmup 策略时需同步更新。

- 风险标记：核心路径变更，性能阈值松弛风险，多模型兼容性

关联脉络

- 暂无明显关联 PR