PR #29306 完整报告

sgl-project/sglang

[diffusion] feat: enable compile warmup for vae decode

合并时间：2026-07-04 15:25

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29306

执行摘要

- 一句话：为 diffusion VAE decode 添加 torch.compile 预热机制
- 推荐动作：该 PR 值得精读，特别是 torch_compile.py 中的 ActiveTargetCompiledCallable 和 CompiledModuleRegistry 设计模式，以及 warmup 策略中"内部预热请求"的权衡。对于扩散模型开发者，可作为 torch.compile 集成的参考实现。

功能与动机

VAE compile 能降低稳态解码延迟，但第一个编译调用开销很大。另外普通 warmup 请求不足以覆盖真实请求路径，因为 warmup 和真实请求可能走不同分支。Z-Image 也暴露了 prompt-length guard 问题：caption embeddings 在编译的 transformer forward 内部 padding，导致不同原始 prompt 长度（即使属于同一 bucket）触发新图编译。

实现拆解

1. 提取通用 torch.compile 辅助工具（新增 torch_compile.py）：包含 build_torch_compile_kwargs（根据平台构建 kwargs）、resolve_torch_compile_mode（环境变量覆盖编译模式）、CompiledModuleRegistry（去重编译注册表）、CallableModule（包装非 forward callable）和 ActiveTargetCompiledCallable（按目标对象缓存编译结果）。重构 denoising.py 中的编译调用，统一使用新工具。
2. 在 VAE 解码中集成编译（修改 decoding.py）：在 DecodingStage.__init__ 中初始化 _compiled_vae_decode 缓存；新增 _get_vae_decode_fn 方法，根据 enable_torch_compile 和 vae 类型决定是否编译；使用 ActiveTargetCompiledCallable 自动管理 VAE 实例变化时的缓存失效。
3. 调整服务预热策略（修改 server_args.py、server_warmup.py、scheduler.py）：当 enable_torch_compile 且未显式指定 warmup 模式时，自动启用 server warmup；在 scheduler.py 中添加内部预热请求支持（_should_return_lightweight_warmup_result），区分非生成请求避免崩溃。
4. 修复 Z-Image caption padding（修改 zimage.py config 和 model）：将 caption embeddings 从在编译后的 transformer forward 内 padding 改为在外部 bucket-padded，并传入 tensor 有效长度，避免因原始 prompt 长度差异导致图重编译。
5. 配套测试与配置校验：更新 test_server_args.py 验证 torch.compile 时 warmup 模式自动切换；test_cfg_parallel_warmup.py 新增内部预热、显式 step 等场景的测试；

test_decoding_stage_parallelism.py 添加 VAE 编译缓存替换的回归测试。

关键文件：

- python/sglang/multimodal_gen/runtime/utils/torch_compile.py（模块 编译工具；类别 source；类型 dependency-wiring；符号 maybe_enable_inductor_compute_comm_overlap, build_torch_compile_kwargs, resolve_torch_compile_mode, CompiledModuleRegistry）：新增的编译辅助工具模块，提取了 torch.compile 的统一入口，包括构建 kwargs、解析模式、去重编译注册表、包装可调用模块、缓存按目标编译的功能，被 denoising 和 decoding 共用。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/decoding.py（模块 VAE 解码；类别 source；类型 core-logic；符号 _get_vae_decode_fn）：核心变更文件：在 DecodingStage 中集成 VAE 编译，引入 ActiveTargetCompiledCallable 缓存，新增 _get_vae_decode_fn 方法，在 decode 方法中根据 server_args 动态选择是否编译 VAE decode 函数。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 去噪阶段；类别 source；类型 dependency-wiring；符号 _maybe_torch_compile, _torch_compile_registry）：重构 torch.compile 逻辑，将内联的编译代码抽取为统一的工具函数，使用 CompiledModuleRegistry 代替散列的 set 管理已编译模块，并利用 build_torch_compile_kwargs 和 resolve_torch_compile_mode 减少重复代码。
- python/sglang/multimodal_gen/runtime/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic；符号 _should_return_lightweight_warmup_result）：添加内部预热请求的支持，新增 _should_return_lightweight_warmup_result 方法，区分生成请求和非生成请求（如 LoRA 控制请求），避免 AttributeError。
- python/sglang/multimodal_gen/test/unit/test_cfg_parallel_warmup.py（模块 CFG 预热；类别 test；类型 test-coverage；符号 test_server_warmup_keeps_minimum_image_steps_without_compile, test_torch_compile_respects_explicit_server_warmup_steps, test_torch_compile_server_warmup_repeats_each_bucket, test_lightweight_warmup_result_ignores_control_requests）：为 torch.compile warmup 添加大量单元测试，包括显式 warmup steps、多 resolution 重复预热、内部预热请求隔离等场景，同时增加了对非生成请求（SetLoraReq、UnmergeLoraWeightsReq）的保护测试。
- python/sglang/multimodal_gen/test/unit/test_server_args.py（模块 服务参数；类别 test；类型 test-coverage；符号 test_torch_compile_defaults_to_server_warmup, test_torch_compile_respects_explicit_warmup_off, test_torch_compile_uses_server_warmup_for_explicit_resolutions, test_torch_compile_server_warmup_disabled_for_disagg_role）：验证 enable_torch_compile 时自动启用 server warmup 的逻辑，包括默认行为、显式关闭 warmup、显式设置 resolution、disagg 角色下禁用等场景。
- python/sglang/multimodal_gen/runtime/warmup_request_builder.py（模块 预热构建；类别 source；类型 core-logic）：修复 SamplimgParams 浅拷贝导致多个 Req 共享同一实例的 bug，并为 torch.compile 场景生成额外的内部预热请求（每个 resolution 两份）。

关键符号：maybe_enable_inductor_compute_comm_overlap, build_torch_compile_kwargs, resolve_torch_compile_mode, CompiledModuleRegistry.compile_once, ActiveTargetCompiledCallable.get_or_compile, DecodingStage._get_vae_decode_fn, Scheduler._should_return_lightweight_warmup_result, _pad_text_embed_for_dit, build_warmup_reqs

关键源码片段

python/sglang/multimodal_gen/runtime/utils/torch_compile.py

新增的编译辅助工具模块，提取了 torch.compile 的统一入口，包括构建 kwargs、解析模式、去重编译注册表、包装可调用模块、缓存按目标编译的功能，被 denoising 和 decoding 共用。

```
# SPDX-License-Identifier: Apache-2.0
# python/sglang/multimodal_gen/runtime/utils/torch_compile.py

import os
from collections.abc import Callable
from dataclasses import dataclass, field
from typing import Any
import torch.nn as nn

from sglang.multimodal_gen.runtime.platforms import current_platform
from sglang.srt.utils.common import get_compiler_backend


def build_torch_compile_kwargs(*, mode: str | None) -> dict[str, object]:
    """构造 torch.compile 参数字典，根据平台做差异化配置"""
    compile_kwargs: dict[str, object] = {"fullgraph": False, "dynamic": None}
    if current_platform.is_npu():
        # NPU 使用 torchair 后端且必须关闭 dynamic
        compile_kwargs["backend"] = get_compiler_backend()
        compile_kwargs["dynamic"] = False
    elif mode is not None:
        compile_kwargs["mode"] = mode
    return compile_kwargs


@dataclass
class CompiledModuleRegistry:
    """记录已被编译的 module id，避免重复编译"""
    module_ids: set[int] = field(default_factory=set)

    def is_compiled(self, module: nn.Module) -> bool:
        return id(module) in self.module_ids

    def compile_once(self, module: nn.Module, *,
                     compile_kwargs: dict[str, object]) -> bool:
        module_id = id(module)
```

```
        if module_id in self.module_ids:
            return False
        module.compile(**compile_kwargs)
        self.module_ids.add(module_id)
        return True


class CallableModule(nn.Module):
    """将普通 callable 包装为 nn.Module，以便使用 module.compile"""
    def __init__(self, fn: Callable[..., Any]) -> None:
        super().__init__()
        self.fn = fn

    def forward(self, *args, **kwargs):
        return self.fn(*args, **kwargs)


@dataclass
class ActiveTargetCompiledCallable:
    """缓存当前活动目标对象的编译版本，支持按目标更换"""
    target_id: int | None = None
    compiled_module: CallableModule | None = None

    def get_or_compile(self, target: object, fn: Callable[..., Any],
                       *, compile_kwargs: dict[str, object]) -> Callable[..., Any]:
        target_id = id(target)
        if self.target_id == target_id and self.compiled_module is not None:
            # 目标未变且已编译，直接返回缓存
            return self.compiled_module

        # 目标改变或首次编译
        module = CallableModule(fn)
        module.compile(**compile_kwargs)
        self.target_id = target_id
        self.compiled_module = module
        return module
```

python/sglang/multimodal_gen/runtime/pipelines_core/stages/decoding.py

核心变更文件：在 DecodingStage 中集成 VAE 编译，引入 ActiveTargetCompiledCallable 缓存，新增 _get_vae_decode_fn 方法，在 decode 方法中根据 server_args 动态选择是否编译 VAE decode 函数。

```
# python/sglang/multimodal_gen/runtime/pipelines_core/stages/decoding.py ( 部分 )

from sglang.multimodal_gen.runtime.utils.torch_compile import (
    ActiveTargetCompiledCallable,
    build_torch_compile_kwargs,
    resolve_torch_compile_mode,
)
```

```
class DecodingStage(PipelineStage):
    def __init__(self, vae, pipeline=None, component_name: str = "vae") -> None:
        super().__init__()
        self.vae: ParallelTiledVAE = vae
        self.pipeline = weakref.ref(pipeline) if pipeline else None
        self.component_name = component_name
        # 记录当前 VAE 实例的编译缓存，当 VAE 实例变化时自动失效
        self._compiled_vae_decode = ActiveTargetCompiledCallable()

    def _get_vae_decode_fn(self, vae, server_args: ServerArgs):
        """返回 VAE 的 decode 函数（原始或已编译版本）"""
        if not server_args.enable_torch_compile or not isinstance(vae, nn.Module):
            return vae.decode

        will_compile = (
            self._compiled_vae_decode.target_id != id(vae)
            or self._compiled_vae_decode.compiled_module is None
        )
        if current_platform.is_npu():
            compile_kwargs = build_torch_compile_kwargs(mode=None)
            if will_compile:
                logger.info("Compiling VAE decode with torchair backend on NPU")
        else:
            mode = resolve_torch_compile_mode(
                "SGLANG_VAE_TORCH_COMPILE_MODE",
                "SGLANG_TORCH_COMPILE_MODE",
                default="default",
            )
            compile_kwargs = build_torch_compile_kwargs(mode=mode)
            if will_compile:
                logger.info("Compiling VAE decode with mode: %s", mode)

        return self._compiled_vae_decode.get_or_compile(
            vae, vae.decode, compile_kwargs=compile_kwargs
        )

    @torch.no_grad()
    def decode(self, batch: Req, server_args: ServerArgs,
               output_batch: OutputBatch) -> None:
        # ... 其他逻辑 ...
        with temporary_module_dtype(self.vae, vae_dtype, enabled=should_cast_vae) as vae:
            decode_output = self._get_vae_decode_fn(vae, server_args)(latents)
        # ...
```

评论区精华

Review 中由 gemini-code-assist[bot] 指出了三个关键问题：

1. VAE 编译缓存内存泄漏：使用 id(vae) 做 key 但强引用导致旧 VAE 实例无法 GC，已通过改用 ActiveTargetCompiledCallable 自动失效修复，并添加测试验证。
2. 调度器非生成请求崩溃：scheduler 中直接访问 processed_req.extra 在 SetLoraReq 等请求上会引发 AttributeError，已通过 get_first_generation_req 保护修复。
3. Warmup 请求参数浅拷贝：SamplingParams 被浅拷贝导致多个 Req 共享同一实例，修改一个影响其他，已通过为每个请求单独拷贝修复。所有问题均已在相应 commit 中解决并添加回归测试。

- VAE 编译缓存内存泄漏 (correctness): 作者在 commit c5d9ce2561 中修复：编译缓存使用 ActiveTargetCompiledCallable 自动随目标对象变化失效，并添加测试确认。
- 调度器非生成请求崩溃 (correctness): 作者在 commit d4449cd263 中修复：使用 get_first_generation_req 保护并添加 regression 测试。
- Warmup 请求参数浅拷贝 (correctness): 作者在 commit c5d9ce2561 中修复：为每个请求单独拷贝 sampling_params，并添加测试验证隔离性。

风险与影响

- 风险： 1) VAE 编译缓存管理：若 VAE 实例重载或卸载，旧实例可能因强引用无法 GC（已在 c5d9ce2561 修复）。 2) 调度器崩溃：非生成请求缺少 extra 属性导致 AttributeError（已在 d4449cd263 修复）。 3) Warmup 参数共享：浅拷贝导致数据竞争（已在 c5d9ce2561 修复）。 4) 平台兼容性：NPU 使用 torchair 后端且关闭 dynamic，其他平台需验证编译模式。 5) 冷启动延迟：首次编译仍慢，但预热机制将其推至服务器就绪前，不影响用户请求。
  - 影响：对用户：启用 --enable-torch-compile 后 VAE decode 延迟显著降低，稳态性能提升。对系统：Server warmup 行为改变，默认启用 server warmup，增加冷启动时间但避免用户请求承受编译开销。对团队：扩散模型的编译优化路径更加清晰，辅助工具可复用，风险已充分修复。
  - 风险标记：VAE 编译缓存泄漏（已修复），非生成请求崩溃（已修复），Warmup 参数共享（已修复），NPU 平台差异化处理

关联脉络

- PR #29631 [diffusion][cache-dit] add cache-dit support for Ideogram 4: 本 PR 与 cache-dit (#29631) 共享 diffusion 路径中 torch.compile 集成的上下文，前者为加速 denoising 阶段编译，后者为 VAE decode 编译，形成互补。
- PR #30016 [diffusion] feat: performance_mode=speed enables torch.compile by default: #30016 将 speed 模式默认启用 torch.compile，本 PR 则进一步为 VAE decode 提供编译预热机制，两者共同完善 diffusion 领域的编译优化策略。