PR #37090 完整报告

sgl-project/sglang

[Diffusion] Cache Qwen-Image modulation across serial CFG branches

合并时间：2026-08-31 01:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37090

执行摘要

- 一句话：Qwen-Image CFG 双分支调制投影缓存，GB300 端到端提速约 5.8%
- 推荐动作：值得精读。核心看点：如何利用串行 CFG 的结构冗余安全地消除确定性重复计算——用张量身份 + 版本号而非数值比较做缓存键、命中即消费的单槽语义、以及 grad/compile/graph/ 流捕获四类守卫的显式枚举。测试用计数投影在无完整模型的情况下精确断言「第二次前向不重算」，是轻量级验证缓存语义的好范例。该模式可推广到其他带串行 CFG 的扩散 DiT（如 GLM-Image、Flux 系列）以及任何「同一步内多次前向共享 timestep 派生计算」的场景。

功能与动机

Qwen-Image 默认串行 CFG 策略会在同一个 denoising 步内先后运行 conditional 与 unconditional 两次前向，且两次前向共享同一个 timestep 张量。PR body 明确指出："The modulation projections depend on the timestep embedding and fixed block weights, not prompt conditioning, so the conditional and unconditional branches produce the same values." 该优化思路引自已发布的 "Agentic Kernels in Production"（Baseten 博客），且 profiler 数据证实了冗余规模：每个 denoising 步存在 120 次重复的调制投影（60 transformer blocks × img/txt 两组投影），NVFP4 GEMM 与 unquantized addmm 合计 240 次可在两步采样内消除。

实现拆解

1. 缓存键构造（qwen_image.py 顶层新增）：新增 _safe_tensor_version 辅助函数——inference 张量没有可用的 _version 计数器，直接访问会抛错，因此对 tensor.is_inference() 返回 None；新增 frozen dataclass _QwenModulationCacheKey（eq=False，避免 dataclass 对张量做值比较），其 matches 方法用 is 比较张量身份、用版本号捕捉 in-place 修改，并比对 additional_t_cond 与 hidden 的 dtype/device；新增工厂函数 _qwen_modulation_cache_key，在 grad 开启、torch.compiler.is_compiling()、is_in_breakable_cuda_graph() 或 CUDA 流捕获期间返回 None，保证训练、编译与图执行路径完全走原有未缓存实现。
2. 单槽缓存写入与消费（QwenImageTransformerBlock）：__init__ 新增 _modulation_cache 属性（缓存键 + img_mod/txt_mod 两组投影输出），原始 forward 中直接调用 self.img_mod[1](temb_img_silu) / self.txt_mod[1](temb_txt_silu) 的两行改为调用新方法 _get_modulation_params：命中时返回缓存值并立即清空（一次性消费语义），

未命中时重算并写入；block 的 forward 签名新增可选参数 modulation_cache_key，向后兼容外部调用。

3. 模型前向贯通（QwenImageTransformer2DModel.forward）：在 timestep = (timestep / 1000).to(hidden_states.dtype) 归一化之前调用 _qwen_modulation_cache_key(timestep, additional_t_cond, hidden_states) 构造缓存键，并随 block 循环下传给每个 transformer block。同一步内 conditional / unconditional 两次前向拿到的是同一个 timestep 张量对象，第二次前向必然命中；prompt 相关的 hidden states、attention 与 MLP 计算仍完全独立。
4. 测试配套（test/registered/kernels/ops/diffusion/test_model_fast_paths.py）：新增 TestQwenImageModulationCache 四个用例，用 _CountingProjection（返回 (x + offset, None) 并自增 calls）配合 QwenImageTransformerBlock.__new__ 轻量化构造 block，覆盖：相同键两次调用只触发一次投影且返回同一对象、三类失效（in-place 修改 timestep.add_、clone 出的新张量、新增 additional_t_cond）、grad 开启时缓存键为 None 且旧缓存被清空、inference 张量可正常缓存以及可打断 CUDA graph 回退。
5. 验证配套：ruff、py_compile、git diff --check 通过；定向 pytest 4 passed；PR Test 与 PR Test (Extra) CI 通过，AMD ROCm 7.2 作业显示失败（PR 内未附原因）。基准采用 A-B-B-A 交替协议，5 组 saved-request 端到端对比，并用 SHA256、像素差、MSE、PSNR、SSIM、LPIPS 做了生成一致性验证。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py（模块 扩散模型；类别 source；类型 data-contract；符号 _safe_tensor_version, _QwenModulationCacheKey, matches, _qwen_modulation_cache_key）：本 PR 唯一源码变更文件，承载缓存键构造（_QwenModulationCacheKey、_qwen_modulation_cache_key）、单槽缓存属性与 _get_modulation_params 命中 / 消费逻辑，以及模型 forward 的键构造与下传，是全部性能收益的来源。
- test/registered/kernels/ops/diffusion/test_model_fast_paths.py（模块 前向测试；类别 test；类型 test-coverage；符号 _CountingProjection, TestQwenImageModulationCache, test_matching_cfg_key_reuses_both_modulation_projections, test_tensor_identity_version_and_condition_invalidate_cache）：新增 TestQwenImageModulationCache 四个用例，用计数投影轻量化验证命中复用、身份 / 版本 / 条件失效、grad 禁用与 graph 回退，覆盖了缓存语义的全部边界。

关键符号：_safe_tensor_version, _QwenModulationCacheKey.matches, _qwen_modulation_cache_key, QwenImageTransformerBlock._get_modulation_params, QwenImageTransformerBlock.forward, QwenImageTransformer2DModel.forward

关键源码片段

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

本 PR 唯一源码变更文件，承载缓存键构造（_QwenModulationCacheKey、_qwen_modulation_cache_key）、单槽缓存属性与 _get_modulation_params 命中 / 消费逻辑，以及模型 forward 的键构造与下传，是全部性能收益的来源。

```
# ---- 缓存键构造：串行 CFG 两次前向共享同一个 timestep 对象 ----
def _safe_tensor_version(tensor: torch.Tensor) -> Optional[int]:
    """读取 tensor 版本号，但对 inference 张量返回 None。

    inference 张量没有可用的 `_version` 计数器，直接访问会抛错；
    这里返回 None，让缓存键仍能参与身份与 dtype/device 匹配。
    """
    return None if tensor.is_inference() else tensor._version


@dataclass(frozen=True, eq=False)
class _QwenModulationCacheKey:
    timestep: torch.Tensor
    timestep_version: Optional[int]
    additional_t_cond: Optional[torch.Tensor]
    additional_t_cond_version: Optional[int]
    hidden_dtype: torch.dtype
    hidden_device: torch.device

    def matches(self, other: "_QwenModulationCacheKey") -> bool:
        # 张量身份用 is 比较（必须是同一对象），版本号捕捉 in-place 修改；
        # eq=False 保证 dataclass 不会对张量做值比较，避免触发 autograd 记录
        return (
            self.timestep is other.timestep
            and self.timestep_version == other.timestep_version
            and self.additional_t_cond is other.additional_t_cond
            and self.additional_t_cond_version == other.additional_t_cond_version
            and self.hidden_dtype == other.hidden_dtype
            and self.hidden_device == other.hidden_device
        )


def _qwen_modulation_cache_key(
    timestep: Optional[torch.Tensor],
    additional_t_cond: Optional[torch.Tensor],
    hidden_states: torch.Tensor,
) -> Optional[_QwenModulationCacheKey]:
    """构造两个串行 CFG 前向共享的缓存键；不适合缓存的执行模式返回 None。"""
    # 训练、torch.compile、可打断 CUDA graph 与流捕获全部走原未缓存路径
    if (
        not isinstance(timestep, torch.Tensor)
        or torch.is_grad_enabled()
        or torch.compiler.is_compiling()
        or is_in_breakable_cuda_graph()
        or (timestep.device.type == "cuda"
            and torch.cuda.is_current_stream_capturing())
    ):
        return None
```

```
return _QwenModulationCacheKey(
    timestep=timestep,
    timestep_version=_safe_tensor_version(timestep),
    additional_t_cond=additional_t_cond,
    additional_t_cond_version=(
        _safe_tensor_version(additional_t_cond)
        if isinstance(additional_t_cond, torch.Tensor)
        else None
    ),
    hidden_dtype=hidden_states.dtype,
    hidden_device=hidden_states.device,
)

def _get_modulation_params(
    self,
    temb_img_silu: torch.Tensor,
    temb_txt_silu: torch.Tensor,
    cache_key: Optional[_QwenModulationCacheKey],
) -> Tuple[torch.Tensor, torch.Tensor]:
    cached = self._modulation_cache
    if (
        cache_key is not None
        and cached is not None
        and cached[0].matches(cache_key)
    ):
        # 命中即消费：立刻清空单槽缓存，避免陈旧条目被后续调用误用
        self._modulation_cache = None
        return cached[1], cached[2]

    # 未命中：重算 img_mod 与 txt_mod 两组调制投影并写入单槽缓存；
    # cache_key 为 None（训练 / 编译 / 图捕获）时不缓存，保持原行为
    img_mod_params, _ = self.img_mod[1](temb_img_silu)
    txt_mod_params, _ = self.txt_mod[1](temb_txt_silu)
    self._modulation_cache = (
        (cache_key, img_mod_params, txt_mod_params)
        if cache_key is not None
        else None
    )
    return img_mod_params, txt_mod_params

    hidden_states, _ = self.img_in(hidden_states)

    # 在 timestep 归一化与除法之前构造缓存键：同一 denoising 步的
    # conditional / unconditional 两次前向拿到同一个 timestep 张量对象，
    # 第二次前向才能命中各 block 的单槽缓存
    modulation_cache_key = _qwen_modulation_cache_key(
        timestep,
        additional_t_cond,
        hidden_states,
    )
```

```
        timestep = (timestep / 1000).to(hidden_states.dtype)
```

评论区精华

本 PR 没有 review 评论或讨论线程（comments_count 与 review_comments_count 均为 0），由作者 BBuf 自行合并；head 分支名 codex/qwen-image-cfg-modulation-cache-b300 显示实现由 Codex agent 生成、经维护者确认提交。设计权衡集中在 PR body 的自述中，核心两点：其一，缓存键用张量身份（is）+ 版本号而非数值比较，"Key the one-slot cache by tensor identity/version plus the conditioning tensor and activation dtype/device"；其二，命中即消费，"consume the entry after the matching second pass"，避免陈旧条目被后续 miss 误用。守卫条件（grad / torch.compile / 可打断 CUDA graph / 流捕获返回 None）在 body 中明确承诺保持训练与图执行路径不变，并由测试逐一覆盖。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 依赖 torch 私有属性：_safe_tensor_version 读取 tensor._version，这是 PyTorch 私有接口，虽对 inference 张量做了兼容，但未来 PyTorch 变更可能影响该逻辑。
  2. 单槽缓存配对语义：缓存假设严格串行的 CFG 双分支且第二次前向必然发生；若某次执行只有单分支（如 CFG 关闭或策略变化），旧条目不会造成正确性问题（下一次 miss 会覆盖），但会错过优化；不同 timestep 张量对象之间也不会误命中。
  3. 验证范围有限：字节级一致性与基准只覆盖 NVFP4 量化 checkpoint 的 eager 路径，非量化、其他量化格式及非 CUDA 后端未做等价验证；缓存逻辑本身与量化无关，但收益量级可能因 GEMM 占比不同而不同。
  4. AMD ROCm CI 失败待确认：PR body 显示 Latest PR Test (AMD ROCm 7.2) 为红叉，无法从 PR 内确认是否与本改动相关，建议关注后续 CI 稳定性。
  5. 测试文件位置：缓存测试被放入 kernel 单元测试套件 test_model_fast_paths.py（CUDA kernel CI stage），而不是模型级测试目录，后续若增加端到端 CFG 测试需注意覆盖分层。 - 影响：影响面集中在 Qwen-Image 全系列推理前向路径（qwen_image.py 内所有配置、TP/SP 分片下每个局部 rank 独立生效），默认串行 CFG 策略直接受益约 5.8% 端到端；训练、torch.compile、CUDA graph 捕获用户透明（走原路径）；缓存键含张量身份，不跨请求共享，批处理与并发安全。对团队而言，这是 Qwen-Image 在 GB300 NVFP4 上系统化性能优化系列的一环，验证方法论（A-B-B-A 交替、SHA256/像素级一致性）值得作为扩散模型性能 PR 的参照模板。 - 风险标记：依赖 torch._version 私有接口，单槽缓存依赖串行 CFG 严格配对，字节级验证仅覆盖 NVFP4 eager 路径，AMD ROCm CI 失败待确认

关联脉络

- PR #37116 [diffusion] perf: absorb Qwen-Image output projection biases: 与本次 PR 修改同一个 qwen_image.py 文件、同属 Qwen-Image 在 GB300 NVFP4 上的性能优化线（bias 融合约 6% 与 CFG 调制复用约 5.8% 的收益互补叠加），两 PR 共同构成该模型的

系统化提速方向。

- PR #36916 [Diffusion] Detect quantized transformer replacements: 本 PR 基准使用的 lmsys/qwen-image-2512-modelopt-nvfp4-sglang 是预量化 NVFP4 模型，36916 修复了这类预量化 transformer 的加载识别，是本次基准可稳定运行的前提之一。
- PR #36991 [Diffusion] Add exact component precision overrides: 同属 multimodal_gen 扩散组件加载与推理基础设施演进（精度覆盖、组件驻留），与本 PR 的调制投影缓存共享 QwenImageTransformer 相关组件上下文。