PR #25848 完整报告

sgl-project/sglang

[diffusion] Add CFG gating for denoising

合并时间：2026-05-25 22:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25848

执行摘要

- 一句话：CFG 门控复用残差，减少去噪计算
- 推荐动作：值得精读。关键设计包括：缓存 delta 按模型身份失效确保多模型场景正确、与 cfg_parallel 互斥通过简单条件判断、状态字典统一管理。实现简洁，注释清晰，适合作为扩散推理加速的范例。

功能与动机

本 PR 移植自 FastVideo PR#1372（hao-ai-lab/FastVideo#1372）。Classifier-free guidance 每个去噪步骤需运行条件和无条件两个 forward；通过 Adaptive Guidance 方法，在指定比例步骤后缓存 cond - uncond 残差并复用，可减少一次 transformer forward 计算。

实现拆解

1. 在 envs.py 中声明环境变量 SGLANG_DIFFUSION_CFG_GATE_STEP，默认值 1.0（不启用），并在环境变量字典中注册。
2. 在 denoising.py 中添加 _init_cfg_gate_state 方法，在进入去噪循环前初始化门控状态字典，包括 fraction、gate_step、delta 缓存等。检查值与批量是否启用 CFG、是否与 CFG parallel 互斥。
3. 在 _predict_noise_with_cfg 方法中增加 cfg_gate_state 参数，当 active 且 step_index >= gate_step 时，无条件分支将使用缓存的 delta 重建预测，跳过模型 forward。若模型 id 与缓存不匹配则强制刷新并失效。
4. 在 _finalize_denoising_loop 中添加 _log_cfg_gate_summary，记录 fresh_uncond、reused、invalidations 等统计信息，便于调试。
5. 新增 test_cfg_gating.py，包含四个单元测试：① 默认无操作行为；② 门控步骤后复用残差；③ 模型切换时失效缓存；④ 无效门控配置验证。

关键文件：

- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 扩散引擎；类别 source；类型 core-logic；符号 _init_cfg_gate_state, _log_cfg_gate_summary）：核心实现，添加 CFG 门控的状态初始化和日志，并在去噪循环中传递门控状态
- python/sglang/multimodal_gen/test/unit/test_cfg_gating.py（模块 测试；类别 test；类型 test-coverage；符号 _PipelineConfig, TestCFGGating, _make_server_args,

_make_batch）：新增单元测试覆盖门控行为、模型切换失效、无效配置等，确保正确性

- python/sglang/multimodal_gen/envs.py（模块 配置；类别 source；类型 core-logic）：声明环境变量 SGLANG_DIFFUSION_CFG_GATE_STEP 及其默认值，作为门控功能的配置入口

关键符号：_init_cfg_gate_state, _log_cfg_gate_summary, _predict_noise_with_cfg

关键源码片段

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

核心实现，添加 CFG 门控的状态初始化和日志，并在去噪循环中传递门控状态

```
def _init_cfg_gate_state(
    self, ctx: DenoisingContext, batch: Req, server_args: ServerArgs
) -> None:
    """Initialize optional CFG residual reuse for the current denoising loop."""
    # 读取环境变量，默认 1.0（表示关闭）
    fraction = envs.SGLANG_DIFFUSION_CFG_GATE_STEP
    # 严格校验范围，防止无效配置
    if not 0.0 <= fraction <= 1.0:
        raise ValueError(
            "SGLANG_DIFFUSION_CFG_GATE_STEP must be between 0.0 and 1.0, "
            f"got {fraction}."
        )

    num_steps = len(ctx.timesteps)
    # 仅当 fraction < 1.0 且 batch 启用 CFG 时才请求门控
    requested = fraction < 1.0 and batch.do_classifier_free_guidance
    # 门控与 CFG parallel 互斥，确保正确性
    active = requested and not server_args.enable_cfg_parallel
    # 计算门控生效的 step 索引
    gate_step = int(num_steps * fraction) if active else num_steps + 1
    # 初始化门控状态字典，存放在 ctx.extra 中
    ctx.extra["cfg_gate_state"] = {
        "fraction": fraction,
        "requested": requested,
        "active": active,
        "gate_step": gate_step,
        "delta": None, # 缓存的 cond - uncond 残差
        "model_id": None, # 最后计算 delta 的模型 id
        "fresh_uncond": 0, # 统计：完整计算无条件次数
        "reused": 0, # 统计：复用残差次数
        "invalidations": 0, # 统计：因模型切换失效次数
    }

    # 预热阶段或非 rank0 节点不输出日志
    if ctx.is_warmup or get_world_group().local_rank != 0:
        return
```

```
    if active:
        logger.info(
            "CFG gating enabled: reuse unconditioned residual after step %d/%d "
            "(fraction=%.3f).",
            gate_step,
            num_steps,
            fraction,
        )
        # 当 guidance_rescale 非零时，建议用户自行 benchmark 质量
        if batch.guidance_rescale > 0:
            logger.warning(
                "CFG gating is enabled with guidance_rescale=%s; benchmark image "
                "quality before using this setting in production.",
                batch.guidance_rescale,
            )
    elif requested:
        logger.info(
            "CFG gating requested but skipped because CFG parallel is enabled."
        )
```

python/sglang/multimodal_gen/test/unit/test_cfg_gating.py

新增单元测试覆盖门控行为、模型切换失效、无效配置等，确保正确性

```
  def test_reuses_unconditional_residual_after_gate_step(self):
    # 创建测试所需的 DenoisingStage 实例和辅助对象
    stage = DenoisingStage.__new__(DenoisingStage)
    batch = self._make_batch() # 返回 SimpleNamespace，do_classifier_free_guidance=True
    server_args = self._make_server_args() # 关闭 CFG parallel
    policy = CFGPolicy().build(batch, {}, {}, {})
    calls = []

    # 伪造 _predict_noise，记录调用顺序：pos 表示条件，neg 表示无条件
    def fake_predict_noise(**kwargs):
        calls.append("neg" if batch.is_cfg_negative else "pos")
        timestep = kwargs["timestep"]
        timestep_value = float(timestep.item())
        offset = 0.25 if batch.is_cfg_negative else 1.25
        return torch.tensor([timestep_value + offset])

    stage._predict_noise = fake_predict_noise
    model = torch.nn.Identity()
    latents = torch.zeros(1)
    state = self._make_gate_state(gate_step=1) # gate_step=1，即 step 0 之后开始复用

    # 第一次调用：step 0，未到达 gate_step，应执行两次 forward（pos + neg）
    first = stage._predict_noise_with_cfg(
        current_model=model,
        latent_model_input=latents,
        timestep=torch.tensor(0),
```

```
        batch=batch,
        timestep_index=0,
        attn_metadata=None,
        target_dtype=torch.float32,
        current_guidance_scale=4.0,
        cfg_policy=policy,
        cfg_gate_state=state,
        server_args=server_args,
        guidance=None,
        latents=latents,
    )
    # 第二次调用：step 1，已到达 gate_step，应复用残差，仅执行一次 forward（pos）
    second = stage._predict_noise_with_cfg(
        current_model=model,
        latent_model_input=latents,
        timestep=torch.tensor(1),
        batch=batch,
        timestep_index=1,
        attn_metadata=None,
        target_dtype=torch.float32,
        current_guidance_scale=4.0,
        cfg_policy=policy,
        cfg_gate_state=state,
        server_args=server_args,
        guidance=None,
        latents=latents,
    )

    # 验证输出值：经过 CFG 公式后，预期 first=4.25, second=5.25
    self.assertTrue(torch.equal(first, torch.tensor([4.25])))
    self.assertTrue(torch.equal(second, torch.tensor([5.25])))
    # 验证调用顺序：第一次应为 pos, neg；第二次只有 pos
    self.assertEqual(calls, ["pos", "neg", "pos"])
    # 验证统计计数：fresh_uncond=1, reused=1, invalidations=0
    self.assertEqual(state["fresh_uncond"], 1)
    self.assertEqual(state["reused"], 1)
    self.assertEqual(state["invalidations"], 0)
```

评论区精华

PR 仅获得一名审核者批准，没有公开讨论记录。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 质量退化风险：复用残差假设后验收敛，某些场景（尤其是 guidance_rescale>0）可能影响生成质量，PR 已发出警告建议 benchmark。

    - 与 CFG parallel 互斥：当 *enable_cfg_parallel* 为 True 时门控不会激活，用户需注意。
    - 环境变量强校验：值必须在 [0,1] 内，否则抛出 ValueError。
    - 模型切换失效机制：通过模型对象 id 判断，多进程或模块包装可能影响稳定性，但当前实现已验证。
- 影响：
    - 用户：默认无变化，设置环境变量即可获得性能提升，适合延迟敏感场景。
    - 系统：减少去后阶段 FLOPs，增加少量内存和日志输出。监控更透明。
    - 团队：新增一个配置参数，代码集中在 diffusion 模块，不影响其他模块。
    - 风险标记：质量退化风险，与 CFG parallel 互斥，配置校验严格

关联脉络

- 暂无明显关联 PR