PR #34690 完整报告

sgl-project/sglang

[BugFix][VLM] keep Qwen3-VL MoE inference deepstack order

合并时间：2026-08-28 17:02

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34690

执行摘要

- 一句话：Qwen3-VL FP8 视觉定位漂移修复：门控 deepstack 注入顺序
- 推荐动作：值得精读。这是一个典型且罕见的"浮点加法顺序影响推理精度"案例：同一模型、同样权重，仅改变残差加法次序就让 FP8 grounding 坐标漂移 285 px。设计上用训练专用标志 rl_on_policy_target 隔离数值路径，而不是简单回滚，保住了 RL 训练侧的 HF 对齐目标。对 VLM 精度调试、FP8 数值敏感性处理和训练 / 推理双路径设计都有参考价值。建议重点阅读 qwen3_vl_moe.py 的 forward 分支与测试用例的断言设计。

功能与动机

PR #14636 为了让 Qwen3-VL deepstack 注入顺序与 HuggingFace 一致以支持 RL on-policy 训练与 FSDP，把各 early decoder layer 末尾的 in-place add_ 改成了下一层 post_residual_addition，且没有用 rl_on_policy_target 门控，作用于所有 forward 路径。由于浮点加法不可结合，FP8 Qwen3-VL 推理中该重排使 grounding 坐标系统性漂移：在 1000 宽画布上目标中心 (750, 270)，错误顺序输出 (807, 555)，Y 方向漂移 +285 px，点选 / 指代任务打不中目标。本 PR 的目标是训练保持 HF 对齐、推理恢复原始精度顺序。

实现拆解

1. 引入运行时标志：在 Qwen3MoeLLMModel.init（python/sglang/srt/models/qwen3_vl_moe.py）与 Qwen3LLMModel.init（python/sglang/srt/models/qwen3_vl.py）中新增 self.use_hf_deepstack_order = get_exec().deterministic.rl_on_policy_target is not None，并新增 from sglang.srt.runtime_context import get_exec 读取运行时确定性配置。
2. 拆分 forward 注入路径：两个 forward 循环中，把原先无条件传入 post_residual_addition=deepstack_embeds 的调用改为 if/else 双分支。HF 顺序分支保留原逻辑；推理分支先正常调用 layer(...)，再在 input_deepstack_embeds 非空且 layer_idx 属于 deepstack_embed_to_decoder_layer 时执行 hidden_states.add_(input_deepstack_embeds[:, sep : sep + self.hidden_size])，还原 #14636 之前的数值顺序。
3. 修正末层 deepstack 处理：last_deepstack 只在 use_hf_deepstack_order 为 True 时计算，否则传 None；因为推理路径的 deepstack 已在各层结尾 add_ 完成，避免 norm 阶段二次注入。
4. 测试配套：在 test/registered/vlm/test_vision_openai_server_a.py 新增 _make_grounding_image 与 TestQwen3VLServer.test_deepstack_grounding_hits_targe

t_box。测试用 PIL 生成 1000x1000 白色画布并绘制红框 (620, 180, 880, 360)，通过 OpenAI 兼容接口让 Qwen3-VL-30B-A3B-Instruct 输出红框中心坐标，正则解析后断言落在 margin 60 的框区域内；该用例在修复前失败（Y 漂移 +285 px），修复后通过（(750, 275) 在框内）。

5. 配置与部署：无新增命令行参数，完全复用既有 rl_on_policy_target 运行时配置；推理路径 kernel 数量不变，无性能回归（PR body 实测确认）。

关键文件：

- python/sglang/srt/models/qwen3_vl_moe.py（模块 模型实现；类别 source；类型 data-contract；符号 Qwen3MoeLLMModel.init, Qwen3MoeLLMModel.forward, use_hf_deepstack_order）：MoE 版 Qwen3-VL 解码器主路径，deepstack 注入由无条件 post_residual_addition 改为按 rl_on_policy_target 门控的双路径，是 FP8 grounding 回归的直接修复点。
- python/sglang/srt/models/qwen3_vl.py（模块 模型实现；类别 source；类型 data-contract；符号 Qwen3LLMModel.init, Qwen3LLMModel.forward, use_hf_deepstack_order）：dense 版 Qwen3-VL 解码器，与 MoE 版本共享同等 deepstack 逻辑，做镜像门控改动，避免 dense 模型同样出现 grounding 回归。
- test/registered/vlm/test_vision_openai_server_a.py（模块 端到端测试；类别 test；类型 test-coverage；符号 _make_grounding_image, test_deepstack_grounding_hits_target_box）：新增 point-in-box 端到端回归测试，覆盖真实模型输出，能直接捕获 deepstack 顺序回归；测试依赖模型输出格式与坐标解析，需关注稳定性。

关键符号：Qwen3MoeLLMModel.init, Qwen3MoeLLMModel.forward, Qwen3LLMModel.init, Qwen3LLMModel.forward, _make_grounding_image, test_deepstack_grounding_hits_target_box

关键源码片段

python/sglang/srt/models/qwen3_vl_moe.py

MoE 版 Qwen3-VL 解码器主路径，deepstack 注入由无条件 post_residual_addition 改为按 rl_on_policy_target 门控的双路径，是 FP8 grounding 回归的直接修复点。

```
class Qwen3MoeLLMModel(Qwen3MoeModel):
    def __init__(
        self,
        *,
        config: Qwen3VLMoeTextConfig,
        quant_config: Optional[QuantizationConfig] = None,
        prefix: str = "",
        decoder_layer_type=Qwen3MoeDecoderLayer,
    ):
        super().__init__(
            config=config,
            quant_config=quant_config,
            prefix=prefix,
            decoder_layer_type=decoder_layer_type,
```

```
    )
    self.hidden_size = config.hidden_size
    # 这里固定为 3，因为无法直接访问 config.vision_config.deepstack_visual_indexes，
    # 与原始实现保持一致；TODO 后续可通过 Qwen3VLMoeConfig 直接获取。
    self.deepstack_embed_to_decoder_layer = range(3)
    # 只有显式设置 rl_on_policy_target（RL on-policy / FSDP 训练）时才走 HF 顺序；
    # 普通推理保持原始原位 add_ 顺序，避免浮点加法不可结合导致 FP8 grounding 漂移。
    self.use_hf_deepstack_order = (
        get_exec().deterministic.rl_on_policy_target is not None
    )

def forward(
    self,
    input_ids: torch.Tensor,
    positions: torch.Tensor,
    forward_batch: ForwardBatch,
    input_embeds: torch.Tensor = None,
    pp_proxy_tensors: Optional[PPProxyTensors] = None,
    input_deepstack_embeds: Optional[torch.Tensor] = None,
) -> Union[torch.Tensor, PPProxyTensors]:
    # ...（省略 pp 分组与 aux_hidden_states 收集逻辑）...
    for layer_idx, layer in enumerate(
        self.layers[self.start_layer : self.end_layer]
    ):
        layer_idx += self.start_layer

        if self.use_hf_deepstack_order:
            # HF 顺序路径（RL on-policy / FSDP）：SGLang 在下一层开头才应用
            # residual，为匹配 HF 的 (hidden_states + residual) + deepstack，
            # 将上一层 deepstack 通过 post_residual_addition 注入。
            deepstack_embeds = self.get_deepstack_embeds(
                layer_idx - 1, input_deepstack_embeds
            )
            hidden_states, residual = layer(
                positions,
                hidden_states,
                forward_batch,
                residual,
                post_residual_addition=deepstack_embeds,
            )
        else:
            # 推理路径：本层结束后原位 add_ 注入 deepstack，与 #14636 之前的
            # 数值顺序一致，保证 FP8 视觉 grounding 精度。
            hidden_states, residual = layer(
                positions,
                hidden_states,
                forward_batch,
                residual,
            )
```

```
            if (
                input_deepstack_embeds is not None
                and layer_idx in self.deepstack_embed_to_decoder_layer
            ):
                sep = self.hidden_size * layer_idx
                hidden_states.add_(
                    input_deepstack_embeds[:, sep : sep + self.hidden_size]
                )

        # 最后一层 deepstack 仅在 HF 顺序路径下由 norm 的
        # post_residual_addition 处理；推理路径已在层内 add_ 完成。
        last_deepstack = (
            self.get_deepstack_embeds(self.end_layer - 1, input_deepstack_embeds)
            if self.use_hf_deepstack_order
            else None
        )
        # ...（省略 pp_proxy 返回与 norm 调用）...
```

test/registered/vlm/test_vision_openai_server_a.py

新增 point-in-box 端到端回归测试，覆盖真实模型输出，能直接捕获 deepstack 顺序回归；测试依赖模型输出格式与坐标解析，需关注稳定性。

```
_GROUNDING_IMG_SIZE = 1000
# 目标框像素坐标：(620, 180, 880, 360) 中心为 (750, 270)，在 1000x1000
# 画布上恰好等于 0-1000 归一化坐标，避免像素与归一化换算带来的歧义。
_GROUNDING_BOX = (620, 180, 880, 360)
_GROUNDING_MARGIN = 60
_GROUNDING_COORD_RE = re.compile(r'\(?\s*(\d{1,4})\s*,\s*(\d{1,4})\s*\)?')


def _make_grounding_image() -> str:
    # 生成白色画布 + 单个红色目标框的 base64 data URI。
    img = Image.new('RGB', (_GROUNDING_IMG_SIZE, _GROUNDING_IMG_SIZE), (255, 255,
    255))
    ImageDraw.Draw(img).rectangle(_GROUNDING_BOX, fill=(220, 30, 30))
    buf = io.BytesIO()
    img.save(buf, format='PNG')
    return '' + base64.b64encode(buf.getvalue()).decode('utf-8')


class TestQwen3VLServer(ImageOpenAITestMixin, VideoOpenAITestMixin):
    model = 'Qwen/Qwen3-VL-30B-A3B-Instruct'

    def test_deepstack_grounding_hits_target_box(self):
        # 回归守卫：预测点必须落在目标框内；若 deepstack 顺序再次被
        # post_residual_addition 污染，坐标会漂出框外（参考 PR #14636）。
        client = openai.Client(api_key=self.api_key, base_url=self.base_url)
        response = client.chat.completions.create(
            model='default',
```

```
        messages=[
            {'role': 'system', 'content': _GROUNDING_SYSTEM},
            {
                'role': 'user',
                'content': [
                    {'type': 'image_url', 'image_url': {'url': _make_grounding_image()}},
                    {'type': 'text', 'text': 'Point at the center of the red rectangle.'},
                ],
            },
        ],
        temperature=0,
        **(self.get_vision_request_kwargs()),
    )
    out = response.choices[0].message.content
    match = _GROUNDING_COORD_RE.search(out or '')
    self.assertIsNotNone(match, f'could not parse a coordinate from: {out!r}')
    x, y = int(match.group(1)), int(match.group(2))
    x0, y0, x1, y1 = _GROUNDING_BOX
    inside = (x0 - _GROUNDING_MARGIN <= x <= x1 + _GROUNDING_MARGIN) and (
        y0 - _GROUNDING_MARGIN <= y <= y1 + _GROUNDING_MARGIN
    )
    self.assertTrue(
        inside,
        f'grounding output {out!r} -> ({x}, {y}) fell outside target box',
    )
```

评论区精华

该 PR 没有留下 review 评论，合入者 yhyang201 直接 APPROVED。核心论证集中在 PR body：精度对比表显示 post_residual_addition 顺序输出 (807, 555)，Y 漂移 +285 px，而原始原位 add_ 输出 (750, 275) 落在框内；性能方面说明推理路径只是把融合 kernel 的 post_residual_addition 换成本层末尾等价 add_，无新增 kernel、分配或显存流量。CI 曾多次触发 /rerun-failed-ci，但没有留下失败根因分析，E2E 测试稳定性仍是未展开的讨论点。

- deepstack 注入顺序与 FP8 数值稳定性 (design): 推理路径恢复原位 add_，训练路径保留 HF 顺序，行为以 use_hf_deepstack_order 区分。

风险与影响

- 风险：
  1. 双路径分叉风险：训练（rl_on_policy_target 开启）与推理数值路径不同，后续修改 deepstack 注入时只改一条分支会造成训练 / 推理行为不一致；qwen3_vl.py 与 qwen3_vl_moe.py 的镜像改动也容易漏改其一。
  2. 运行时标志读取时机：use_hf_deepstack_order 在模型 __init__ 时通过 get_exec() 读取，若某些 pipeline（如多进程、pp 分组）下模型加载早于运行时配置设置或 get_exec() 不可用，可能走错分支。

3. 测试 flaky 风险：point-in-box 依赖真实 LLM 输出与正则解析，模型版本、prompt 或采样策略变化都可能影响稳定性；CI 运行模型（30B-A3B）与作者实测模型（235B-A22B-FP8）不同，漂移幅度不同，margin 60 的裕量未在讨论中确认。
4. 回归范围：两个解码器与 norm 尾部都涉及 deepstack，漏改 last_deepstack 会导致重复或缺失注入。 - 影响：用户侧：FP8 Qwen3-VL（30B-A3B、235B-A22B 等）的视觉 grounding / 点选任务坐标恢复准确；非 grounding 任务不受影响，因为只是前 3 层加法顺序还原。系统侧：改动局限在模型 forward 的 deepstack 注入点，不涉及 kernel、显存布局或调度器；训练侧 HF 对齐行为保持不变。团队侧：新增的"合成图 + point-in-box 断言"回归测试范式可复用到其他 VLM 精度问题；forward 中运行时标志分支带来一定维护成本。 - 风险标记：核心推理路径变更，双路径行为分叉，镜像改动同步风险，E2E 测试潜在 flaky

关联脉络

- PR #14636 Qwen3-VL deepstack injection order change: 本 PR 的直接上游：该 PR 将 deepstack 注入改为 post_residual_addition 并应用到所有 forward 路径，导致 FP8 推理 grounding 回归；本 PR 将该行为门控到 rl_on_policy_target 场景。