PR #34891 完整报告

sgl-project/sglang

fix(diffusion): scope attention backend fallback

合并时间：2026-08-15 22:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34891

执行摘要

- 一句话：修复 diffusion 注意力后端回退过严，区分隐式偏好与显式覆盖
- 推荐动作：值得精读 selector.py 的“候选回退链”设计：它把严格校验、显式 / 隐式偏好区分、有界回退与可观测日志组合在一起，对多后端推理框架有普适借鉴意义。建议阅读时重点关注 selection_is_explicit 的判定来源和失败时的错误优先级。同时建议后续 PR 将 is_cross_attention 贯通到更多 diffusion 模型，并在 AMD/ROCm 上补充端到端验证。

功能与动机

Issue #34389 报告：PR #33707 将 desired attention backend 不在 supported 集合时的行为从默认 SDPA 改为直接抛错，ROCm 上默认请求后端是 AITER，导致几乎所有模型报错，MiniMax-H3 完全崩溃，其余模型回退到 diffusers 实现而非优化的 sgl-d 版本。PR body 说明根因：strict 校验正确暴露了显式选择错误，但同时也把 ROCm 的自动 AITER 偏好当作用户覆盖，并强制稀疏 Wan 自注意力后端用于仅支持稠密注意力的交叉注意力层。

实现拆解

1. 区分显式与隐式选择（selector.py）：get_attn_backend 新增 is_cross_attention 参数；在选择解析链路上逐级跟踪 selection_is_explicit。selected_attention_backend、全局强制后端、组件强制后端只要命中即视为显式；从 server_args.attention_backend 读取时，仅当 server_args 是完整 ServerArgs 实例且 is_arg_explicitly_set("attention_backend") 为真才算显式，避免 ROCm 平台自动偏好的 AITER 被当作覆盖。
2. 构建有界回退链：allowed_fallback_reason 在三种情况下放行：无 selected_backend（平台默认）、is_cross_attention and selected_backend.is_sparse（稀疏偏好用于交叉注意力）、非显式选择。候选队列先放 selected_backend，再追加 None 与 be_tuple 中的其他后端；循环中对每个候选依次做 _cached_get_attn_backend 解析、_is_backend_supported 范围校验、unsupported_requirements 能力校验，取第一个通过者，并记录 fallback_reason 用于日志。
3. 保持 fail-closed：全部候选失败时按“能力缺失 > 选择错误 > 无可用后端”顺序抛错；显式稠密不匹配场景仍抛出 not supported by this attention layer，测试断言该消息。日志对回退场景输出原因后缀（如 (dense cross-attention fallback)），提升可观测性。
4. 打通参数链路：layer.py 的 USPAttention.__init__ 新增 is_cross_attention 参数并透传给 get_attn_backend；wanvideo.py 的 WanSelfAttention 在构造 USPAttention 时传入 is_cross_attention=is_cross_attention，而 WanT2VCrossAttention 已经以

is_cross_attention=True 调用父类。

5. 测试配套：新增 test_attention_backend_selector.py（伪造 AITER/FA/SDPA 后端与平台，覆盖隐式回退、隐式能力缺失回退、轻量 SimpleNamespace 视为隐式、显式稠密 mismatch fail-closed、稀疏 cross-attention 回退、稀疏 self-attention fail-closed 六类场景）与 test_wan_attention_backend.py（验证 WanSelfAttention 把 is_cross_attention 与 skip_sequence_parallel 传给 USPAttention）。两组测试均为纯 CPU 单元测试。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/attention/selector.py（模块 选择器；类别 source；类型 core-logic；符号 get_attn_backend）：核心修复文件：重写 get_attn_backend 为候选回退链，新增 is_cross_attention 参数、显式 / 隐式选择判定、allowed_fallback_reason 与带原因的日志输出。
- python/sglang/multimodal_gen/test/unit/test_attention_backend_selector.py（模块 选择器；类别 test；类型 test-coverage；符号 TestAttentionBackendFallback, _ServerArgs, _FakePlatform, _FakeAITERBackend）：新增 188 行测试，用伪造后端与平台把回退契约固定下来，是理解本 PR 行为边界的最佳入口。
- python/sglang/multimodal_gen/runtime/layers/attention/layer.py（模块 注意力层；类别 source；类型 core-logic；符号 USPAttention.init）：USPAttention 是回退生效的接线点：新增 is_cross_attention 参数并透传给选择器。
- python/sglang/multimodal_gen/runtime/models/dits/wanvideo.py（模块 Wan 模型；类别 source；类型 data-contract；符号 WanSelfAttention.init, WanT2VCrossAttention）：WanSelfAttention 构造 USPAttention 时新增透传 is_cross_attention，让 Wan 交叉注意力层真正获得回退能力。
- python/sglang/multimodal_gen/test/unit/test_wan_attention_backend.py（模块 Wan 模型；类别 test；类型 test-coverage；符号 TestWanAttentionBackendRole, test_cross_attention_role_is_forwarded_to_usp）：新增回归测试，验证 Wan 将 cross-attention 角色（与 skip_sequence_parallel）正确传递给 USPAttention。

关键符号：get_attn_backend, USPAttention.init, WanSelfAttention.init, TestAttentionBackendFallback._resolve

关键源码片段

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

新增 188 行测试，用伪造后端与平台把回退契约固定下来，是理解本 PR 行为边界的最佳入口。

```
class TestAttentionBackendFallback(unittest.TestCase):
    def _resolve(
        self,
        backend: AttentionBackendEnum,
        *,
        explicit: bool,
        is_cross_attention: bool,
        supported: set[AttentionBackendEnum],
```

```
    attention_requirements: AttentionRequirements | None = None,
    server_args: object | None = None,
):
    # 默认用一个能区分“显式 / 隐式”的 ServerArgs 替身；
    # 显式时把 attention_backend 记入 _explicit_arg_names
    if server_args is None:
        server_args = _ServerArgs(backend.name.lower(), explicit=explicit)
    with (
        patch(f"{_SELECTOR}.get_global_forced_attn_backend", return_value=None),
        patch(f"{_SELECTOR}.get_component_forced_attn_backend", return_value=None),
        patch(f"{_SELECTOR}.get_global_server_args", return_value=server_args),
        patch(
            "sglang.multimodal_gen.runtime.platforms.current_platform",
            _FakePlatform,
        ),
        patch(
            f"{_SELECTOR}.resolve_obj_by_qualname",
            side_effect=_FAKE_BACKENDS.__getitem__,
        ),
    ):
        return get_attn_backend(
            128,
            torch.bfloat16,
            supported_attention_backends=supported,
            attention_requirements=attention_requirements,
            is_cross_attention=is_cross_attention,
        )

def test_sparse_backend_falls_back_for_cross_attention(self):
    # 稀疏后端（LASER_ATTN）用在交叉注意力层时允许回退到稠密后端
    backend = self._resolve(
        AttentionBackendEnum.LASER_ATTN,
        explicit=True,
        is_cross_attention=True,
        supported={AttentionBackendEnum.FA, AttentionBackendEnum.TORCH_SDPA},
    )
    self.assertIs(backend, _FakeFABackend)
    # 回退时不应再向平台查询所选后端，避免选中不支持的后端
    self.assertIsNone(_FakePlatform.selected_backend)

def test_sparse_backend_mismatch_fails_for_self_attention(self):
    # 同样的稀疏后端用于自注意力时保持严格，直接抛出选择错误
    with self.assertRaisesRegex(
        ValueError, "not supported by this attention layer"
    ):
        self._resolve(
            AttentionBackendEnum.LASER_ATTN,
            explicit=True,
            is_cross_attention=False,
```

```
            supported={AttentionBackendEnum.FA, AttentionBackendEnum.TORCH_SDPA},
        )
```

python/sglang/multimodal_gen/runtime/layers/attention/layer.py

USPAttention 是回退生效的接线点：新增 is_cross_attention 参数并透传给选择器。

```
class USPAttention(nn.Module):
    def __init__(
        self,
        num_heads: int,
        head_size: int,
        num_kv_heads: int | None = None,
        softmax_scale: float | None = None,
        causal: bool = False,
        supported_attention_backends: set[AttentionBackendEnum] | None = None,
        prefix: str = "",
        dropout_rate: float = 0.0,
        skip_sequence_parallel: bool = False,
        enable_packed_qkv_input_a2a: bool = False,
        is_cross_attention: bool = False,
        **extra_impl_args,
    ) -> None:
        """
        is_cross_attention:
          稀疏后端偏好可为交叉注意力选择兼容的稠密后端，
          而自注意力仍保持严格校验。
        skip_sequence_parallel:
          当 KV 在所有 SP rank 上复制（例如对文本/图像编码器输出的交叉注意力）时，
          本地 Q 分片可直接访问本地完整 KV，无需走完整 USP 通信管线。
        """
        super().__init__()
        if softmax_scale is None:
            self.softmax_scale = head_size**-0.5
        if num_kv_heads is None:
            num_kv_heads = num_heads

        dtype = get_compute_dtype()
        # 把 cross-attention 角色传给选择器，稀疏偏好只在此类层上放行回退
        attn_backend = get_attn_backend(
            head_size,
            dtype,
            supported_attention_backends=supported_attention_backends,
            is_cross_attention=is_cross_attention,
        )
        if get_ring_parallel_world_size() > 1:
            if not attn_backend.supports_ring_rotation():
                raise RuntimeError(
                    "Ring Attention requires a backend whose kernel exposes the "
                    "softmax LSE for the per-hop merge; "
```

```
                f"{attn_backend.get_enum().name} does not declare support "
                "(see AttentionBackend.supports_ring_rotation)."
            )
        impl_cls: Type[AttentionImpl] = attn_backend.get_impl_cls()
        self.allow_cudnn_sdp = bool(extra_impl_args.get("allow_cudnn_sdp", False))
        self.attn_impl = impl_cls(
            num_heads=num_heads,
            head_size=head_size,
            causal=causal,
            softmax_scale=self.softmax_scale,
            num_kv_heads=num_kv_heads,
            prefix=f"{prefix}.impl",
            **extra_impl_args,
        )
```

评论区精华

PR 无任何 review 评论（review_comments_count = 0），唯一的 issue 评论是作者触发 CI 重跑的 /tag-and-rerun-ci。真正的讨论发生在 Issue #34389：报告者给出了 ROCm 环境、可复现命令与崩溃模型清单（MiniMax-H3 完全崩溃，FLUX 等回落 diffusers 路径）。提交历史显示分两步：fix(diffusion): scope attention backend fallback 先界定回退范围，随后的 fix(diffusion): harden attention backend fallback 再加固候选遍历、错误优先级与日志，说明作者在实现中主动补强了边界情况。

- 暂无高价值评论线程

风险与影响

- 风险：
  - selector.py 是 multimodal_gen 所有扩散模型注意力选择的唯一入口，回退链改动影响面大于 Wan 单模型；新增的候选遍历、错误保留与日志逻辑需要足够回归覆盖。
  - is_cross_attention 目前只贯通到 Wan 模型，FLUX、Hunyuan 等其他扩散模型若存在“稀疏自注意力 + 稠密交叉注意力”组合，仍需逐一接线，否则在显式稀疏后端下仍可能报错。
  - 隐式回退恢复意味着用户不显式指定后端时，实际可能使用 FA/SDPA 而非平台默认 AITER，但日志会带 (platform default fallback) 原因，可观测性有兜底。
  - 显式性判定依赖 ServerArgs.is_arg_explicitly_set，与 srt 侧正在进行的配置显式性 / 配置 bag 重构相关联；若该接口语义变化，需回归本 PR 的测试。
  - 无新增 GPU 端到端测试，PR test extra 显示 ❓，ROCm 实际行为依赖后续 AMD CI 验证。
  - 影响：影响范围集中在 multimodal_gen（diffusion）模块：ROCm/NPU 等平台上所有扩散模型不再因隐式平台默认后端不匹配而直接报错或回退到 diffusers 实现；Wan 模型的交叉注意力层可选择兼容的稠密后端。显式后端配置错误依旧快速失败，不掩盖用户意图。对 srt（LLM 推理）路径无影响。团队侧，该 PR 修复了 #34389 阻塞的 AMD 用户场景，并为后续其他扩散模型接入 cross-attention 角色提供了样板。

- 风险标记：核心回退逻辑变更，稀疏→交叉回退仅 Wan 贯通，隐式回退可能掩盖配置意图，无 GPU 端到端验证

关联脉络

- PR #34264 config: decisions keyed on the attention backend read the configured pair: 同属“注意力后端决策正确性”修复：srt 侧修复 prefill/decode 配置对误读，本 PR 修复 multimodal_gen 侧隐式偏好被误判为显式覆盖。
- PR #34376 [Fix] Make the linear-attn kernel choice per-runner, and pin draft/target loader-hook parity: 同为注意力后端选择隔离与配置污染修复，srt 侧按 runner 隔离，本 PR 按组件角色（cross/self-attention）限定回退范围。