PR #27883 完整报告

sgl-project/sglang

Fix fp16 NaN flake in spec CI: bf16 eagle fixture; sanitize NaN logits in sampler

合并时间：2026-06-11 16:16

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27883

执行摘要

- 一句话：修复 spec CI fp16 NaN 问题：测试改用 bf16 并清理 NaN logits
- 推荐动作：值得精读。特别是 _AsyncNanWarner 的设计（利用 pin_memory 避免同步）和 sanitize_nan_logits 的数值选择（+-1e30 而非 dtype min/max 以避免温度缩放后溢出）。

功能与动机

原作提到：fp16 activation overflow on a degenerate draft branch during EAGLE verify becomes Inf -> NaN under RMSNorm。导致 CI 中 test_spec_eagle_fa3.py 出现不稳定的 NaN 设备断言失败。

实现拆解

1. 在 async_probe.py 中新增 _AsyncNanWarner 类，利用 pin_memory 实现无同步的 NaN 检测与限速警告。
2. 新增 sanitize_nan_logits 函数，在 CI 中做断言，在生产环境默认将 NaN/Inf 替换为安全数值。
3. 在 environ.py 注册 SGLANG_SANITIZE_NAN_LOGITS 环境变量（默认 True）。
4. 在 sampler.py 的 _preprocess_logits 和 eagle_info_v2.py 的 sample 方法开头加入 sanitize_nan_logits 调用。
5. 将测试 fixture 的 dtype 从 float16 改为 bfloat16，从根本上避免 fp16 溢出。

关键文件：

- python/sglang/srt/utils/async_probe.py（模块 异步断言；类别 source；类型 core-logic；符号 _AsyncNanWarner, init, check, maybe_warn_nan）：核心变更文件：新增 _AsyncNanWarner 异步 NaN 检测类、maybe_warn_nan 和 sanitize_nan_logits 函数，构成整个防御机制的核心。
- python/sglang/srt/speculative/eagle_info_v2.py（模块 推测解码；类别 source；类型 dependency-wiring）：在 EAGLE 验证的 sample 方法中调用 sanitize_nan_logits，确保 NaN 不会进入采样 kernel。
- python/sglang/srt/layers/sampler.py（模块 采样器；类别 source；类型 dependency-wiring）：在通用采样器的 _preprocess_logits 方法中调用 sanitize_nan_logits，覆盖所有采样路径。

- python/sglang/srt/environ.py（模块 配置层；类别 source；类型 core-logic）：新增环境变量 SGLANG_SANITIZE_NAN_LOGITS（默认 True）作为开关。
- python/sglang/test/server_fixtures/spec_eagle_fixture.py（模块 测试夹具；类别 test；类型 test-coverage）：测试 fixture 将 dtype 从 float16 改为 bfloat16，避免 fp16 溢出导致的 CI 不稳定。

关键符号：sanitize_nan_logits, _AsyncNanWarner.check, maybe_warn_nan, Sampler._preprocess_logits, EagleVerifyInput.sample

关键源码片段

python/sglang/srt/utils/async_probe.py

核心变更文件：新增 _AsyncNanWarner 异步 NaN 检测类、maybe_warn_nan 和 sanitize_nan_logits 函数，构成整个防御机制的核心。

```
class _AsyncNanWarner:
    """One-shot NaN monitor: device-side detection lands in pinned host
    memory without any stream sync; the host reads the (slightly stale) flag
    on a later call, warns once, and stops detecting."""

    def __init__(self):
        self._dev = None # device int32 tensor, initialized lazily
        self._host = None # pinned host mirror, read without sync
        self._warned = False

    def check(self, tensor: torch.Tensor, msg: str):
        # If already warned or tensor is not CUDA, skip
        if self._warned or not tensor.is_cuda:
            return
        # Lazily allocate device and pinned host buffers
        if self._dev is None:
            self._dev = torch.zeros(1, dtype=torch.int32, device=tensor.device)
            self._host = torch.zeros(1, dtype=torch.int32, pin_memory=True)

        # Report a hit enqueued on an earlier step (pinned read, no sync).
        if int(self._host[0]):
            logger.warning(
                "NaN detected in %s; values were sanitized before sampling. "
                "This usually indicates numerical overflow (e.g. fp16 "
                "activations) or an upstream bug producing NaN. "
                "Logged once; further occurrences are silent.",
                msg,
            )
            self._warned = True
            return

        # Enqueue this step's detection (async, no sync).
        self._dev.add_(torch.isnan(tensor).any().to(torch.int32))
```

```
        self._host.copy_(self._dev, non_blocking=True)


_nan_warner = _AsyncNanWarner()


def maybe_warn_nan(tensor: Optional[torch.Tensor], msg: str = ""):
    """Non-fatal counterpart of maybe_detect_nan: throttled sync-free warning
    instead of crashing. Callers sanitize the tensor themselves."""
    if envs.SGLANG_ENABLE_ASYNC_ASSERT.get():
        return # hard assert already covers detection
    if tensor is None:
        return
    _nan_warner.check(tensor, msg)


def sanitize_nan_logits(logits: torch.Tensor, msg: str = ""):
    """Detect NaN (assert in CI, throttled warning in prod), then sanitize in
    place: NaN logits (e.g. fp16 activation overflow) are undefined behavior
    in sampling kernels and can come back as out-of-vocab token ids. +-1e30
    rather than dtype min/max because callers divide logits by temperature,
    which would overflow dtype min/max to +-Inf and softmax back to NaN."""
    maybe_detect_nan(logits, msg)
    if not envs.SGLANG_SANITIZE_NAN_LOGITS.get():
        return
    maybe_warn_nan(logits, msg)
    torch.nan_to_num_(logits, nan=-1e30, posinf=1e30, neginf=-1e30)
```

python/sglang/srt/speculative/eagle_info_v2.py

在 EAGLE 验证的 sample 方法中调用 sanitize_nan_logits，确保 NaN 不会进入采样 kernel。

```
# 在 import 中新增导入
from sglang.srt.utils.async_probe import (
    maybe_detect_nan,
    maybe_detect_oob,
    sanitize_nan_logits,
)

# 在 sample 方法中，获取 logits 后立即清理
next_token_logits = logits_output.next_token_logits
sanitize_nan_logits(next_token_logits, "verify: target model logits")
# 之后才进行 penalty、grammar 等操作
```

评论区精华

无公开 review 讨论。作者通过提交序列逐步完善，从简单切 bf16 到加入通用 NaN 清理机制，并最终确定环境变量默认开启。

- NaN 处理方案 (design): 采用 bf16 测试 fixture 避免溢出，并通过 sanitize_nan_logits 在所有采样路径清理 NaN。

风险与影响

- 风险：sanitize_nan_logits 使用 torch.nan_to_num_ 会带来小幅 CUDA kernel 开销，但通常可忽略。默认启用可能掩盖上游 NaN 产生的 bug，但 CI 中的 maybe_detect_nan 会在启用 SGLANG_ENABLE_ASYNC_ASSERT 时捕获。建议在关键推理场景中保持此功能开启。
- 影响：直接影响：修复 spec EAGLE 测试的不稳定性，避免 CI 误报。间接影响：为所有采样路径提供 NaN 清理安全网，防止采样 kernel 产生越界 token。影响范围限定在 speculative decoding 和 sampler 模块。
- 风险标记：数值精度变更，新增默认开启环境变量

关联脉络

- 暂无明显关联 PR