# PR #37148 完整报告

sgl-project/sglang

[CI] Fix stale GPU capability test patches

合并时间：2026-08-31 01:12

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37148

## 执行摘要

- 一句话：修复 DSV4 与 FP8 测试的失效平台能力 patch，恢复 CI 保护
- 推荐动作：建议快速阅读（10 分钟内），无需精读生产代码。值得关注的设计决策：测试 patch 运行时 API 而非模块级实现细节，让测试断言真正约束生产行为；同时 PR body 对 stale patch 静默失效陷阱的说明值得团队学习。引申建议：可在 CI 或 lint 阶段增加对 mock.patch/patch.object 目标存在性的静态检查，或定期核查测试 patch 目标是否仍被生产代码引用，防止同类问题再次出现。

## 功能与动机

PR body 明确说明：#37086 将平台能力读取从模块级缓存标志迁移到运行时 get_platform() API，涉及 DSV4 的 _is_sm120 和 FP8 的 _is_sm90_supported、_is_sm100_supported、_is_sm120_supported。这两个测试文件在迁移中被遗漏，继续 patch 已经不存在的名字。这里的关键风险是 mock.patch 对不存在的模块属性打补丁不会报错，而是让 patch 静默失效——测试继续跑且绿，但被 patch 的标志已无人读取，断言失去对生产代码的约束力，属于比测试失败更难发现的“假绿”问题。

## 实现拆解

1. 变更入口与背景：PR #37086 将平台能力读取从模块级缓存标志迁移为运行时 get_platform() API。模块级标志在 import 时求值缓存，无法反映运行时变化；新 API 让测试与运行时代码统一走同一读取路径。本 PR 是这次迁移遗漏的测试同步修复。
2. FP8 分发测试：test/registered/quant/test_fp8_utils.py 中 TestApplyFp8LinearScaleDispatch 的两个测试方法（test_native_scalar_a_static_prequant_and_dynamic_scale_shapes 与 test_without_native_scalar_a_static_scale_is_repeated）原本用 patch.multiple(fp8_utils, _is_sm90_supported=..., _is_sm100_supported=..., _is_sm120_supported=...) 控制平台能力，改为 patch.object(fp8_utils, "get_platform", return_value=SimpleNamespace(is_sm90=..., is_sm100=..., is_sm120=...))。能力名从 _is_sm90_supported 变为 is_sm90，去掉 _supported 后缀，反映新 API 直接以 SM 版本命名字段。改动使四个 apply_fp8_linear 调用分支（静态标量、动态 per-token、预量化输出、无 input_scale）的断言重新绑定到真实生产读取路径。
3. DSV4 测试：test/registered/attention/unittests/dsv4/test_deepseek_v4.py 中 TestDSV4BreakableCudaGraphMetadataContract 的两个测试方法原本 patch

sglang.srt.layers.attention.deepseek_v4_backend._is_sm120（模块级标志），改为 patch 同模块的 get_platform 并返回 SimpleNamespace(is_sm120=False)。此处只需提供 is_sm120 一个字段，因为 prepare_prefill_shared_read_snapshot 的 sparse prefill 分支只依赖该能力位。

4. 验证配套：PR body 说明两个文件均通过 pre-commit hooks 与 Python 3.11 字节码编译；本机 pytest collection 因环境中的 Transformers 重复注册 qwen3_asr 被阻塞，作者通过 /rerun-test 让 CI 在 4-gpu-b200 与 1-gpu-h100 上重跑两个目标测试，全部通过。

关键文件：

- test/registered/quant/test_fp8_utils.py（模块 FP8 量化；类别 test；类型 test-coverage；符号 TestApplyFp8LinearScaleDispatch.test_native_scalar_a_static_prequant_and_dynamic_scale_shapes, TestApplyFp8LinearScaleDispatch.test_without_native_scalar_a_static_scale_is_repeated）：FP8 标量 scale 分发测试的唯一修改文件；把四个能力位 patch 从模块级标志切换为 get_platform() 返回值，恢复 apply_fp8_linear 四条分发路径的断言约束力，改动量最大（+21/-12）。
- test/registered/attention/unittests/dsv4/test_deepseek_v4.py（模块 DSV4 后端；类别 test；类型 test-coverage；符号 TestDSV4BreakableCudaGraphMetadataContract.test_snapshot_builds_cache_only_for_sparse_prefill, TestDSV4BreakableCudaGraphMetadataContract.test_sparse_prefill_snapshot_marks_success_only_after_build）：DSV4 sparse prefill snapshot 测试的修改文件；两处 _is_sm120 patch 改为 patch get_platform()，恢复 SM120 边界行为验证（+4/-3），改动量小但涉及注意力后端核心能力位。

关键符号：TestApplyFp8LinearScaleDispatch.test_native_scalar_a_static_prequant_and_dynamic_scale_shapes, TestApplyFp8LinearScaleDispatch.test_without_native_scalar_a_static_scale_is_repeated, TestDSV4BreakableCudaGraphMetadataContract.test_snapshot_builds_cache_only_for_sparse_prefill, TestDSV4BreakableCudaGraphMetadataContract.test_sparse_prefill_snapshot_marks_success_only_after_build

## 关键源码片段

### test/registered/quant/test_fp8_utils.py

FP8 标量 scale 分发测试的唯一修改文件；把四个能力位 patch 从模块级标志切换为 get_platform() 返回值，恢复 apply_fp8_linear 四条分发路径的断言约束力，改动量最大（+21/-12）。

```
def test_without_native_scalar_a_static_scale_is_repeated(self):
    import sglang.srt.layers.quantization.fp8_utils as fp8_utils

    input, qinput, weight, input_scale, weight_scale = self._make_inputs()
    seen_scales = []

    def fake_fp8_scaled_mm(mat_a, mat_b, scales_a, scales_b, out_dtype, bias=None):
        seen_scales.append(scales_a)
        return torch.empty(
            (mat_a.shape[0], mat_b.shape[1]), dtype=out_dtype, device=mat_a.device
```

```
        )

    # 关键改动：生产代码已改走 get_platform() 运行时读取平台能力，
    # 因此测试不再 patch 模块级标志（那些名字已不存在），而是 patch get_platform 的返回值。
    with patch.object(
        fp8_utils,
        "get_platform",
        return_value=SimpleNamespace(
            is_sm90=False,
            is_sm100=False,
            is_sm120=False,
        ),
    ), patch.object(fp8_utils, "fp8_scaled_mm", side_effect=fake_fp8_scaled_mm):
        fp8_utils.apply_fp8_linear(
            input,
            weight,
            weight_scale,
            input_scale=input_scale,
            cutlass_fp8_supported=True,
        )
        fp8_utils.apply_fp8_linear(
            qinput,
            weight,
            weight_scale,
            input_scale=input_scale,
            cutlass_fp8_supported=True,
            pre_quant_output_dtype=input.dtype,
        )

    # 无原生 FP8 标量 scale 支持时，scale 应回退为 per-token 形状 (M, 1)
    # 断言验证回退行为，而不只是"不报错"
    self.assertEqual(tuple(seen_scales[0].shape), (input.shape[0], 1))
    self.assertEqual(tuple(seen_scales[1].shape), (input.shape[0], 1))
```

## test/registered/attention/unittests/dsv4/test_deepseek_v4.py

DSV4 sparse prefill snapshot 测试的修改文件；两处 _is_sm120 patch 改为 patch get_platform()，恢复 SM120 边界行为验证（+4/-3），改动量小但涉及注意力后端核心能力位。

```
  def test_snapshot_builds_cache_only_for_sparse_prefill(self):
      from sglang.srt.environ import envs
      from sglang.srt.layers.attention.deepseek_v4_backend import (
          _LARGE_INDEXER_QUERY_THRESHOLD,
          DeepseekV4AttnBackend,
          DSV4Metadata,
      )
      from sglang.srt.speculative.spec_info import SpeculativeAlgorithm

      batch = SimpleNamespace(forward_mode=ForwardMode.EXTEND)
```

```
        cache = object()
        # 边界行为验证：query token 数等于阈值时不建缓存，超过阈值才建
        for num_qo_tokens, builds in (
            (_LARGE_INDEXER_QUERY_THRESHOLD, False),
            (_LARGE_INDEXER_QUERY_THRESHOLD + 1, True),
        ):
            with self.subTest(num_qo_tokens=num_qo_tokens):
                backend = object.__new__(DeepseekV4AttnBackend)
                backend.model_runner = SimpleNamespace(
                    spec_algorithm=SpeculativeAlgorithm.DFLASH
                )
                backend.forward_metadata = DSV4Metadata(
                    self._make_core_metadata(0), indexer_metadata=None
                )
                backend._build_sparse_prefill_chunk_cache = mock.Mock(return_value=cache)

                # 关键改动：patch 运行时 get_platform 而不是已删除的模块级 _is_sm120
                with (
                    envs.SGLANG_ENABLE_PREFILL_WAR_READ_DONE.override(True),
                    envs.SGLANG_OPT_FLASHMLA_SPARSE_PREFILL.override(False),
                    mock.patch(
                        "sglang.srt.layers.attention.deepseek_v4_backend.get_platform",
                        return_value=SimpleNamespace(is_sm120=False),
                    ),
                ):
                    backend.prepare_prefill_shared_read_snapshot(
                        batch, num_qo_tokens=num_qo_tokens
                    )

                metadata = backend.forward_metadata
                if builds:
                    backend._build_sparse_prefill_chunk_cache.assert_called_once_with(
                        batch, num_qo_tokens=num_qo_tokens
                    )
                    self.assertIs(metadata.sparse_prefill_cache, cache)
                else:
                    backend._build_sparse_prefill_chunk_cache.assert_not_called()
                    self.assertIsNone(metadata.sparse_prefill_cache)
                # Dense 分支同样声明快照边界，只读取 init_forward_metadata 已快照的元数据
                self.assertTrue(metadata.prefill_shared_reads_snapshotted)
```

## 评论区精华

本 PR 没有任何 review 评论，唯一交互发生在 issue 评论区：作者 /rerun-test 请求重跑两个目标测试，github-actions bot 在 4-gpu-b200（1 项）与 1-gpu-h100（2 项）上执行并回报全部通过。虽然没有代码评审交锋，但 PR body 对 stale patch 问题的定位很清晰：patch 不存在的名字不会报错，测试会静默失去约束力，因此本次修复是在恢复“假绿”测试的真实保护能力。

- GPU CI 重跑验证两个目标测试 (testing): patch 目标切换在 B200 与 H100 两个 GPU 平台上均验证通过，未发现平台相关差异。

## 风险与影响

- 风险：
  1. 隐式字段契约：新写法通过 SimpleNamespace(is_sm90=...) 模拟平台对象，字段名必须与生产代码 get_platform() 实际返回对象的属性一致。若生产代码新增读取字段而测试未提供，会抛 AttributeError 导致测试失败——这是显式失败而非静默失效，属于健康的保护方向，但字段列表的同步仍需人工维护。
  2. patch 位置的模块耦合：测试 patch 的是 fp8_utils 与 deepseek_v4_backend 模块命名空间里的 get_platform 名字，要求生产代码在这些模块内以全局名字调用。若未来平台能力改为实例方法、contextvar 或构造注入，测试需再次迁移——这正是本次修复试图避免的历史重演。
  3. 本地验证受限：作者本机无法跑通 pytest collection，验证依赖 CI 重跑；CI 只覆盖 B200 与 H100 两个平台，未覆盖 Ampere（sm80/sm86）等 GPU。但两个目标的测试逻辑本身与具体 GPU 无关（能力位由 mock 注入），风险有限。
  4. 流程性风险：本次暴露的 stale patch 问题说明仓库缺少对 mock.patch 目标存在性的检查机制，类似遗漏可能在未来的重构中再次发生。 - 影响：用户与系统：零运行时影响，纯测试文件变更，不触及任何生产代码路径。

CI 有效性：恢复了两条关键 GPU 路径的测试约束力——DSV4 sparse prefill snapshot 的 SM120 边界行为（阈值处不建缓存、阈值 +1 才建），以及 FP8 native scalar scale 分发的四条 scale 形状断言。CI 从“假绿”变为“真绿”。

团队协作：为后续平台能力测试提供了统一写法——patch get_platform() 返回值而非内部实现标志，涉及 deepseek 与 fp8 两个模块的测试维护者可直接复用该模式。

- 风险标记：测试曾静默失效（stale patch），依赖平台 API 字段契约，本地验证受限

## 关联脉络

- PR #37086 平台能力读取迁移（PR body 引用）：本 PR 的直接触发者：将 DSV4 _is_sm120 与 FP8 _is_sm90_supported/_is_sm100_supported/_is_sm120_supported 从模块级缓存标志迁移为运行时 get_platform() API；本 PR 补齐其遗漏的两个测试文件同步。
- PR #36985 test: re-enable FlashInfer per-token NVFP4 coverage: 同属量化 /GPU 能力测试维护线，恢复 FP8 相关路径的测试保护，与本次 FP8 分发测试修复形成同类演进趋势。