PR #26356 完整报告

sgl-project/sglang

[NPU]Support torch_npu profiler patch API drift

合并时间：2026-06-06 21:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26356

执行摘要

- 一句话：新增 torch_npu 补丁 API 兼容层并更新调用方
- 推荐动作：建议 NPU 相关开发者阅读，该 PR 展示了如何平滑处理上游库的 API 漂移，并提供了可复用的兼容函数。单元测试覆盖了三种场景，值得参考。

功能与动机

根据 PR 描述，新版 torch_npu 不再暴露 _apply_patches(patches) 入口点，导致现有的 NPU Profiler 设置代码在新版本上崩溃。需要一个兼容层来保持旧有行为，并在必要时回退到 _apply_all_patches()。

实现拆解

1. 新增兼容函数：在 python/sglang/srt/utils/torch_npu_patch_utils.py 中创建 apply_torch_npu_patches 函数，优先调用 _apply_patches(patches)，若不可用则回退到 _apply_all_patches()，两者都不可用时抛出 AttributeError。
2. 更新 SRT Profiler 模块：修改 python/sglang/srt/managers/scheduler_components/profiler_manager.py 和 python/sglang/srt/utils/profile_utils.py，将直接调用 torch_npu._apply_patches(patches) 替换为调用 apply_torch_npu_patches(torch_npu, patches)。
3. 更新多模态 Profiler 模块：修改 python/sglang/multimodal_gen/runtime/utils/profiler.py，同样替换为新的兼容函数。
4. 添加单元测试：新增 test/registered/unit/utils/test_torch_npu_patch_utils.py，覆盖三种场景：_apply_patches 优先、回退 _apply_all_patches、以及两者都缺失时抛出异常。测试已在 CPU CI 中注册。

关键文件：

- python/sglang/srt/utils/torch_npu_patch_utils.py（模块 NPU 兼容；类别 source；类型 core-logic；符号 apply_torch_npu_patches）：核心兼容函数所在文件，新增 apply_torch_npu_patches 函数，是整个 PR 的逻辑核心。
- test/registered/unit/utils/test_torch_npu_patch_utils.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestTorchNpuPatchUtils, test_apply_torch_npu_patches_uses_targeted_api_when_available, test_apply_torch_npu_patches_uses_all_patches_api_when_targeted_api_missing,

test_apply_torch_npu_patches_requires_supported_api）：新增单元测试，覆盖三种关键场景，验证兼容函数的正确性。

- python/sglang/srt/managers/scheduler_components/profiler_manager.py（模块 调度器；类别 source；类型 dependency-wiring）：调度器 Profiler 模块，更新调用点以使用兼容函数。
- python/sglang/srt/utils/profile_utils.py（模块 分析工具；类别 source；类型 dependency-wiring）：SRT 通用 Profiler 工具，更新调用点以使用兼容函数。
- python/sglang/multimodal_gen/runtime/utils/profiler.py（模块 多模态生成；类别 source；类型 dependency-wiring）：多模态生成 Profiler 模块，更新调用点以使用兼容函数。

关键符号：apply_torch_npu_patches

关键源码片段

python/sglang/srt/utils/torch_npu_patch_utils.py

核心兼容函数所在文件，新增 apply_torch_npu_patches 函数，是整个 PR 的逻辑核心。

```
from collections.abc import Sequence
from typing import Any


def apply_torch_npu_patches(torch_npu: Any, patches: Sequence[Sequence[Any]]) -> None:
    # 兼容新旧版本 torch_npu 的 patch API:
    # 1. 优先使用更精细的 _apply_patches ( 传入 patches 列表 )
    # 2. 如果不存在，回退到 _apply_all_patches ( 不传参，应用所有默认 patches)
    # 3. 如果两个 API 都缺失，则抛出 AttributeError 提示需要支持的 API
    if hasattr(torch_npu, "_apply_patches"):
        torch_npu._apply_patches(patches)
        return

    if hasattr(torch_npu, "_apply_all_patches"):
        torch_npu._apply_all_patches()
        return

    raise AttributeError(
        "torch_npu must provide either _apply_patches or _apply_all_patches"
    )
```

test/registered/unit/utils/test_torch_npu_patch_utils.py

新增单元测试，覆盖三种关键场景，验证兼容函数的正确性。

```
import types
import unittest

from sglang.srt.utils.torch_npu_patch_utils import apply_torch_npu_patches
from sglang.test.ci.ci_register import register_cpu_ci

register_cpu_ci(est_time=1, suite="base-a-test-cpu")
```

```
class TestTorchNpuPatchUtils(unittest.TestCase):
    # 测试当 _apply_patches 可用时优先使用它
    def test_apply_torch_npu_patches_uses_targeted_api_when_available(self):
        calls = []
        # 模拟 torch_npu 同时拥有两个 API
        torch_npu = types.SimpleNamespace(
            _apply_patches=lambda patches: calls.append(("_apply_patches", patches)),
            _apply_all_patches=lambda: calls.append(("_apply_all_patches", None)),
        )
        patches = [["profiler.profile", object()]]
        apply_torch_npu_patches(torch_npu, patches)
        self.assertEqual(calls, [("_apply_patches", patches)])

    # 测试当 _apply_patches 缺失时回退到 _apply_all_patches
    def test_apply_torch_npu_patches_uses_all_patches_api_when_targeted_api_missing(self):
        calls = []
        # 模拟 torch_npu 只有 _apply_all_patches
        torch_npu = types.SimpleNamespace(
            _apply_all_patches=lambda: calls.append("_apply_all_patches")
        )
        apply_torch_npu_patches(torch_npu, [["profiler.profile", object()]])
        self.assertEqual(calls, ["_apply_all_patches"])

    # 测试两个 API 都不存在时抛出 AttributeError
    def test_apply_torch_npu_patches_requires_supported_api(self):
        with self.assertRaises(AttributeError):
            apply_torch_npu_patches(types.SimpleNamespace(), [])


if __name__ == "__main__":
    unittest.main()
```

评论区精华

自动审核机器人最初指出测试使用了动态模块加载（importlib.util + 硬编码相对路径），建议改为直接导入 apply_torch_npu_patches。作者回复 “already resolved”，最终测试采用了直接导入，使代码更简洁健壮。

- 测试代码中动态加载的使用 (style): 作者采纳建议，将测试改为直接导入，简化了代码。

风险与影响

- 风险：该变更主要影响 NPU 环境下的 Profiler 模块，风险较低。主要风险点：
  - 兼容性边界：若未来 torch_npu 再次更改 API，当前保护逻辑可能不适用，但可类似扩展。
  - 异常路径：当两个 API 都缺失时，抛出 AttributeError 会导致模块启动失败，但这是预期行为，可清晰提示用户更新 torch_npu。
  - 影响：用户侧：NPU 用户升级 torch_npu 后 Profiler 功能恢复正常；非 NPU 用户无影响。系统侧：改动集中在补丁调用点，无运行时开销。团队侧：统一的兼容函数便于后续

维护，测试覆盖了关键路径。

- 风险标记：上游 API 依赖，仅 NPU 环境

关联脉络

- PR #26731 [NPU] Update documentation for software version upgrades: 同为 NPU 模块的持续改进，增强了 NPU 平台兼容性。