PR #29100 完整报告

sgl-project/sglang

[NPU] fix: reach torch>=2.8 CUDA memory-pool APIs lazily via torch._C

合并时间：2026-08-30 06:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29100

执行摘要

- 一句话：修复 pynccl 分配器在 torch 2.7/NPU 启动崩溃
- 推荐动作：值得精读。核心看点有三个：(1) 用「调用点 torch._C 惰性查找」替代「try/except + None 哨兵」的设计取舍，alexnails 的评论点出了 None fallback 会掩盖真实 CUDA 安装损坏的深坑，这个思考对任何跨版本私有 API 兼容处理都有借鉴价值；(2) 静态 AST 回归测试是平台兼容性测试的优秀范本——不需要目标硬件、不修改共享模块状态、可在任何版本上防回归；(3) 平台 hook 的 declare_resolution 声明式降级模式，与同文件 handle_nccl_pre_warm 形成一致的设备门控风格。建议合并后配合 issue #28999 的验证者（Ascend 910B）确认端到端推理不受 FlagGems 外部 bug 影响。

功能与动机

issue #28999 明确指出：SGLang 在 Ascend NPU（torch 2.7.0 + torch_npu）上启动即崩溃，报 ImportError: cannot import name '_cuda_beginAllocateCurrentThreadToPool' from 'torch.cuda.memory'。原因是 pynccl_allocator.py 在模块作用域执行 from torch.cuda.memory import (...)，而其中两个符号是 torch 2.8 才加入的（2.7 中第二个叫 _cuda_endAllocateCurrentStreamToPool），from ... import 的 all-or-nothing 语义让整个导入失效。该模块又被 model_runner、linear、dp_attention、fp8、MoE/quant 层甚至 NPU graph runner 等广泛急切导入，导致错误在任何设备分发发生前就触发。Issue 建议用 try/except + None 兜底，但 reviewers 指出 None 会把损坏的 CUDA 安装变成深处 NoneType is not callable 的困惑错误，最终采纳了直接经由 torch._C 惰性访问的方案。

实现拆解

实现分为四步：

1. 移除模块级私有 CUDA 符号导入（python/sglang/srt/distributed/device_communicators/pynccl_allocator.py）：将原来 from torch.cuda.memory import (CUDAPluggableAllocator, _cuda_beginAllocateCurrentThreadToPool, _cuda_endAllocateToPool, _cuda_releasePool) 中的三个 _cuda_* 私有名称删除，只保留 CUDAPluggableAllocator（它是 torch 2.0 就存在的 Python 类）。理由：torch.cuda.memory 中的这些名称只是从 torch._C re-export，且该文件自身在 __enter__/__exit__ 中已经通过 torch._C._cuda_endAllocateToPool、torch._C._cuda_beginAllocateCurrentThreadToPool 访问其中两个，模块级 import 从未真正 load-bearing。

2. 调用点改走 torch._C：SymmetricMemoryContext.__enter__ 中的 _cuda_beginAllocateCurrentThreadToPool(...) 改为 torch._C._cuda_beginAllocateCurrentThreadToPool(...)；__exit__ 中的 _cuda_endAllocateToPool 和 _cuda_releasePool 同样改为 torch._C 路径。这样 torch 2.7 上导入模块不再失败，只有真正调用 --enable-symm-mem 路径时才需要这些符号。
3. 增加两层前置守卫：
   - 新增 handle_symm_mem_device_support(server_args)（python/sglang/srt/arg_groups/platform_hook.py）：当 enable_symm_mem 为真且设备不是 CUDA/HIP 时，打印 warning 并通过 declare_resolution 声明 enable_symm_mem=False。这模仿同文件 handle_nccl_pre_warm 的设备门控模式，因为 NCCL symmetric memory 需要编译 CUDA 插件并链接 -lnccl，在 NPU 上会在构建步骤深处爆炸，而不是在参数校验时明确失败。
   - 在 pynccl_allocator.get_nccl_mem_pool() 开头新增 assert after_2_8_0，前置声明 --enable-symm-mem requires torch>=2.8，避免用户在半途收到裸 AttributeError。
4. 接线与回归测试：
   - 在 python/sglang/srt/arg_groups/pipeline.py 的 run_resolution_pipeline 中注册 handle_symm_mem_device_support，放在其他平台 hook 之后、handle_gpu_memory_settings 之前（其 symm-mem prealloc 默认值依赖 enable_symm_mem，必须保证降级先发生）。
   - 新增 test/registered/unit/distributed/test_pynccl_allocator_import.py：用 ast 静态解析 pynccl_allocator.py，遍历所有 import-time 节点（跳过函数 / 类定义体，但包括 try/if 体），断言不存在 from torch.cuda.memory import _cuda_*。该测试无需 GPU/NPU，可在任意 torch 版本上运行，且不修改共享 torch 模块状态。

遗留说明：提交历史中有两次 merge main 解决 arg_groups 重构冲突（ServerArgs._handle_gpu_memory_settings 被抽取到 platform_hook.py、_run_resolution_pipeline 被抽取到 pipeline.py），最终本分支不再触及 server_args.py。

关键文件：

- python/sglang/srt/distributed/device_communicators/pynccl_allocator.py（模块 通信分配器；类别 source；类型 dependency-wiring；符号 get_nccl_mem_pool, SymmetricMemoryContext.enter, SymmetricMemoryContext.exit）：问题根因所在文件。删除模块级 torch.cuda.memory 私有符号导入，改为调用点 torch._C 惰性访问，并新增 get_nccl_mem_pool 的 torch>=2.8 断言。这是修复的核心，决定了 NPU 启动是否还会崩溃。
- python/sglang/srt/arg_groups/platform_hook.py（模块 平台钩子；类别 source；类型 core-logic；符号 handle_symm_mem_device_support）：新增 handle_symm_mem_device_support 设备守卫，在非 CUDA/HIP 设备上自动禁用 --enable-symm-mem 并告警，把对称内存的失败从构建期提前到参数解析期。
- test/registered/unit/distributed/test_pynccl_allocator_import.py（模块 回归测试；类别 test；类型 test-coverage；符号 _import_time_nodes, TestPyncclAllocatorImportGuard, test_no_import_time_private_cuda_memory_symbols）：新增静态 AST 回归测试，无需 GPU/NPU 即可在任何 torch 版本上捕获模块级私有

CUDA 符号导入回归，是本次修复能否长期有效的关键保障。

- python/sglang/srt/arg_groups/pipeline.py（模块 参数流水线；类别 source；类型 core-logic；符号 run_resolution_pipeline）：注册 handle_symm_mem_device_support 到参数解析流水线的正确位置，保证在 handle_gpu_memory_settings 之前完成 symm-mem 降级，避免 prealloc 默认值错误生效。

关键符号：get_nccl_mem_pool, handle_symm_mem_device_support, SymmetricMemoryContext.enter, SymmetricMemoryContext.exit, _import_time_nodes, test_no_import_time_private_cuda_memory_symbols

关键源码片段

python/sglang/srt/distributed/device_communicators/pynccl_allocator.py

问题根因所在文件。删除模块级 torch.cuda.memory 私有符号导入，改为调用点 torch._C 惰性访问，并新增 get_nccl_mem_pool 的 torch>=2.8 断言。这是修复的核心，决定了 NPU 启动是否还会崩溃。

```
# python/sglang/srt/distributed/device_communicators/pynccl_allocator.py
import torch

# 只保留 torch 2.0 就存在的公共 Python 类；
# 三个私有 _cuda_* 名称在 torch 2.8 才出现（或改名），
# 因此不再在模块顶层导入，而是由调用点经 torch._C 惰性访问。
from torch.cuda.memory import CUDAPluggableAllocator

after_2_8_0 = torch_release >= (2, 8)


def get_nccl_mem_pool() -> torch.cuda.MemPool:
    """获取所有 group 共享的 MemPool，避免内存碎片化。"""
    # 前置断言：低于 2.8 的 torch 根本不含所需符号，
    # 与其让用户在 context manager 中途撞上 AttributeError，不如在此显式失败。
    assert after_2_8_0, (
        "--enable-symm-mem requires torch>=2.8 "
        "(torch._C._cuda_beginAllocateCurrentThreadToPool was added there)."
    )

    global _allocator, _mem_pool, _cur_device, _register_func
    if _allocator is None:
        # 编译 NCCL CUDA 插件并加载为可插拔分配器（原逻辑不变）
        ...


class SymmetricMemoryContext:
    def __enter__(self):
        # graph capture 分支保留 torch 2.7/2.8 的旧名兼容
        if self.is_graph_capture:
            assert _graph_pool_id is not None, "graph_pool_id is not set under graph capture"
```

```
            if after_2_8_0:
                torch._C._cuda_endAllocateToPool(_cur_device, _graph_pool_id)
            else:
                torch._C._cuda_endAllocateCurrentStreamToPool(_cur_device, _graph_pool_id)

        # 原模块级绑定等价物：torch.cuda.memory 只是从 torch._C re-export
        torch._C._cuda_beginAllocateCurrentThreadToPool(
            self._device_index, self._pool_id
        )
        global _active_symmetric_memory_context
        _active_symmetric_memory_context = self
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        torch._C._cuda_endAllocateToPool(self._device_index, self._pool_id)
        torch._C._cuda_releasePool(self._device_index, self._pool_id)
        self._register_segments_for_comm()
        ...
```

python/sglang/srt/arg_groups/platform_hook.py

新增 handle_symm_mem_device_support 设备守卫，在非 CUDA/HIP 设备上自动禁用 --enable-symm-mem 并告警，把对称内存的失败从构建期提前到参数解析期。

```
# python/sglang/srt/arg_groups/platform_hook.py
def handle_symm_mem_device_support(server_args: Any):
    cfg = resolving_view(server_args)
    # symm-mem 分配器需要编译 CUDA 插件并链接 -lnccl，
    # 在 NPU 等非 CUDA/HIP 设备上会在构建步骤深处才失败；
    # 这里把它提前到参数解析期，明确告知用户并降级为关闭。
    if cfg.enable_symm_mem and not (is_cuda() or is_hip()):
        logger.warning(
            "--enable-symm-mem is not supported on non CUDA/HIP devices "
            "(NCCL symmetric memory is unavailable). Disabling symmetric memory."
        )
        declare_resolution(
            server_args, "_handle_symm_mem_device_support", enable_symm_mem=False
        )
```

test/registered/unit/distributed/test_pynccl_allocator_import.py

新增静态 AST 回归测试，无需 GPU/NPU 即可在任何 torch 版本上捕获模块级私有 CUDA 符号导入回归，是本次修复能否长期有效的关键保障。

```
# test/registered/unit/distributed/test_pynccl_allocator_import.py
"""回归测试：pynccl_allocator 不得在模块作用域导入 torch.cuda.memory 私有符号。"""

import ast
import unittest
from pathlib import Path

# 定位仓库根目录下的目标源文件
```

```
REPO_ROOT = Path(__file__).resolve().parents[4]
SOURCE_PATH = (
    REPO_ROOT / "python/sglang/srt/distributed/device_communicators/pynccl_allocator.py"
)


def _import_time_nodes(tree: ast.Module):
    """产出所有 import 时执行的节点，包括 try / if 体，跳过函数与类定义体。"""
    stack = list(tree.body)
    while stack:
        node = stack.pop()
        if isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef, ast.ClassDef)):
            continue # 函数体内的导入不在 import 时执行，不构成启动风险
        yield node
        stack.extend(ast.iter_child_nodes(node))


class TestPyncclAllocatorImportGuard(unittest.TestCase):
    def test_no_import_time_private_cuda_memory_symbols(self):
        tree = ast.parse(SOURCE_PATH.read_text(), filename=str(SOURCE_PATH))
        # 找出所有 import-time 的 torch.cuda.memory 私有符号
        offenders = [
            alias.name
            for node in _import_time_nodes(tree)
            if isinstance(node, ast.ImportFrom) and node.module == "torch.cuda.memory"
            for alias in node.names
            if alias.name.startswith("_cuda_")
        ]
        # 这些符号在 torch<2.8 缺失，会阻断 Ascend NPU 启动；
        # 应改在调用点通过 torch._C.<name> 访问。
        self.assertEqual(offenders, [], f"模块级私有导入: {offenders}")
```

评论区精华

1. None 兜底 vs 惰性 torch._C 访问（design，已解决）：PR 最初采用 try/except + None fallback（issue 建议的方向），但 alexnails 提出异议："Setting None could cause odd behavior that would not make sense to bad CUDA install"，即 None 会把损坏的 CUDA 安装伪装成后续 NoneType is not callable 的费解错误。tech-cow 接受并推送了 alexnails 的替代分支，核心论据是 torch.cuda.memory 只是从 torch._C re-export，文件内已有调用点在使用 torch._C，方法体使用与模块级绑定等效，因此删除导入是运行时 no-op。
2. symm-mem 非 CUDA 设备的显式失败（design，已解决）：ronhuafeng 与 JosephNew 都指出仅有导入修复不够，应在 --enable-symm-mem 实际被请求时给出明确错误。JosephNew 原话："a clear 'NCCL symmetric memory unavailable on this backend' error at the call site would be a nice follow-up hardening"。最终该 PR 直接实现了 handle_symm_mem_device_support 的 warning + 自动禁用 + get_nccl_mem_pool 的 torch 版本断言，将二位的建议从"follow-up"变为本次交付。

3. 真实 NPU 验证承诺（question，未解决 / 外部依赖）：JosephNew 表示其团队拥有 Ascend 910B 环境，可在合并后验证 SGLang 在 NPU 上“starts + serves + runs inference”；但他同时指出还有另一个独立的 FlagGems argmax bug（flagos-ai/FlagGems#4466 / #4467）会阻断完整推理验证。因此本 PR 只保证导入成功，端到端 NPU 推理验证依赖外部修复。
4. 测试策略：静态 AST 而非运行时导入（testing，已解决）：alexnails 设计的测试避免在 torch 2.8 环境直接 import pynccl_allocator（那样会真的访问 CUDA API），也不修改共享 torch 模块状态，而是用 ast 做静态检查，能在任何 torch 版本捕获模块级私有符号回归。

- None 哨兵 vs torch._C 惰性访问 (design): 采用 alexnails 的方案：删除模块级私有导入，调用点直接 torch._C.<name>；保留 CUDAPluggableAllocator 的公共导入。
- --enable-symm-mem 非 CUDA 设备的显式失败 (design): 实现 handle_symm_mem_device_support 守卫：非 CUDA/HIP 设备 warning + 自动禁用；get_nccl_mem_pool 增加 torch>=2.8 断言。
- 真实 NPU 端到端验证 (question): 本 PR 仅修复导入阶段；端到端 NPU 推理验证依赖外部 FlagGems 修复，未在本 PR 内完成。
- 静态 AST 测试设计 (testing): 采用静态 AST 测试，作为无需硬件的回归防线。

风险与影响

- 风险：
  1. CUDA 路径调用语义变化风险：_cuda_beginAllocateCurrentThreadToPool、_cuda_endAllocateToPool、_cuda_releasePool 从模块级绑定改为 torch._C 属性查找。虽然 torch 内部是同一 C 函数，但如果未来 torch 修改 torch._C 的暴露方式（极低概率），CUDA 上的 symmetric memory 会受影响。所有调用点均发生在 SymmetricMemoryContext.__enter__/__exit__ 与 get_nccl_mem_pool，没有运行时路径之外的间接依赖。

1. torch 2.7 与 2.8 分支路径的既有复杂性：__enter__/__exit__ 的 graph-capture 分支仍保留 after_2_8_0 的 if/else（2.8 前后函数名不同），本次变更没有消除该分叉，只是把边界推到 torch._C 查找。任何 torch 版本更新都需重测 --enable-symm-mem + CUDA graph 捕获组合。
2. AST 测试的脆弱性：test_no_import_time_private_cuda_memory_symbols 假定 pynccl_allocator.py 路径固定（parents[4]），仓库结构变化会导致测试误报；同时 AST 检查只覆盖 torch.cuda.memory 的 _cuda_* 名字，若未来新增其他模块级 CUDA 私有符号则不会被捕获。
3. 降级守卫的覆盖盲区：handle_symm_mem_device_support 只检查 is_cuda() or is_hip()，对 CPU/XPU/MLX 等其他后端也禁用 symm-mem——这是预期行为，但如果未来 NCCL symmetric memory 支持非 CUDA 平台，此守卫需同步更新。
4. 行为变更面：--enable-symm-mem 在非 CUDA/HIP 设备上从“静默不生效（因为无人传参）”变为“显式 warning + 禁用”，是用户可见的行为变化，但方向是更安全的失败。 - 影响：用户侧：修复 Ascend NPU（torch 2.7 + torch_npu）上 SGLang 完全无法启动的阻断性问题，影响所有 NPU 用户；对 CUDA/HIP 用户是纯内部重构（同一 C 函数、无运行时行

为变化），对 torch<2.8 的 CUDA 用户还顺带消除了模块导入期的潜在风险。系统侧：pynccl_allocator 被模型加载、量化、深度并行等多个模块急切导入，本次修复消除了启动路径上的一颗地雷；同时让 --enable-symm-mem 在错误设备上的失败从构建期深处提前到参数解析期。团队侧：静态 AST 测试作为新的回归防线，不依赖 GPU 即可在任意 torch 版本上运行，适合作为此类“导入期平台兼容性”问题的标准测试模式。影响范围集中在分布式通信与平台参数解析两个模块，风险可控。 - 风险标记：核心启动路径变更，跨版本私有 API 依赖，测试依赖源码路径固定，外部 FlagGems bug 阻塞端到端验证

关联脉络

- PR #28999 [Bug] Hardcoded torch.cuda.memory import in pynccl_allocator.py breaks Ascend NPU compatibility: 本 PR 直接修复该 issue 报告的 NPU 启动崩溃，issue 中的调用链和错误信息是变更的主要依据。
- PR #37108 [mem_cache] Share one ReqKvInfo between a streaming session slot and its request: 同属近期 mem_cache 与请求状态重构系列，本 PR 的 arg_groups 变更需与 main 上的重构（如 RequestKvInfo 移动）保持同步，且多次 merge main 与这些重构冲突。
- PR #37164 [mem_cache] Move mamba state and retraction_backup into ReqKvInfo: 同样改动 schedule_batch/streaming_session 等运行状态结构，与本 PR 共同构成对 distributed 状态管理的持续演进，虽不直接改同一文件，但属于同一演进脉络。
- PR #35281 [PD] Align defensive protocol behavior across Mooncake, NIXL, and Mori: 同为跨平台（含 NPU）的分布式通信兼容性修复，且都涉及 pynccl/PD 后端的防御性行为设计，可作为同类问题的参考。