PR #36249 完整报告

sgl-project/sglang

[diffusion] feat: support out-of-tree torch.compile backends

合并时间：2026-08-26 09:53

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36249

执行摘要

- 一句话：支持 Diffusion 模块自定义 torch.compile 后端与选项
- 推荐动作：值得精读，尤其是 build_torch_compile_kwargs 中 backend/options/mode 的互斥与回退逻辑，以及 Platform 基类“默认实现 + 可选覆写”的扩展模式。该设计低侵入、向后兼容，可为其他子系统的平台适配提供参考。

功能与动机

PR body 明确指出：此前 diffusion 编译只对 NPU 平台做了后端特判，其他 out-of-tree 平台会静默使用默认 Inductor 后端，且无法提供 per-module 编译器选项（例如 persistent buffers 这类需要根据模块状态生成的后端配置）。因此需要把“选哪个后端、带什么选项”从 torch_compile 工具函数中抽离出来，交给平台层通过钩子决策。

实现拆解

1. 定义平台钩子（python/sglang/multimodal_gen/runtime/platforms/interface.py）：在 Platform 基类新增 get_compile_backend(mode) 与 get_compile_options(module) 两个默认实现，前者返回 simple_compile_backend，后者返回 None。默认实现保持“Inductor + 无额外选项”的旧行为，子类可覆写。
2. 改造编译参数构造（python/sglang/multimodal_gen/runtime/utils/torch_compile.py）：build_torch_compile_kwargs 新增 module 可选参数；优先分支从 is_npu() 改为 is_out_of_tree()，通过钩子取 backend 与 options；当未提供 options 且 backend 为 inductor 且 mode 非空时，才回退设置 mode，保证 mode/options 互斥。NPU 分支保留原逻辑，并将 get_compiler_backend 的导入移入分支内实现延迟导入。
3. 透传 DiT 模块（python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py）：_maybe_torch_compile 在非 NPU 分支调用 build_torch_compile_kwargs(mode=mode, module=module)，使 out-of-tree 平台能拿到 DiT 模块并生成 persistent buffers 等模块级配置。
4. 测试配套（python/sglang/multimodal_gen/test/unit/test_regional_torch_compile.py）：新增参数化测试 test_out_of_tree_platform_controls_compile_kwargs，覆盖 custom_backend + options、inductor + options、inductor + None 三种组合，并断言钩子的入参；同时把 test_denoising_stage_selects_regional_compile 改为 mock build_torch_compile_kwargs，验证其以 (mode, module) 被调用。

关键文件：

- python/sglang/multimodal_gen/runtime/utils/torch_compile.py（模块 编译配置；类别 source；类型 core-logic；符号 build_torch_compile_kwargs）：核心改造：build_torch_compile_kwargs 新增 module 参数和 out-of-tree 分支，决定 backend/options/mode 的优先级，是本次功能的主入口。
- python/sglang/multimodal_gen/runtime/platforms/interface.py（模块 平台抽象；类别 source；类型 core-logic；符号 get_compile_backend, get_compile_options）：定义平台扩展点：新增 get_compile_backend / get_compile_options 默认实现，是 out-of-tree 平台接入的接口契约。
- python/sglang/multimodal_gen/test/unit/test_regional_torch_compile.py（模块 编译单测；类别 test；类型 test-coverage；符号 test_out_of_tree_platform_controls_compile_kwargs）：新增参数化单测验证三种 backend/options 组合，并扩展区域编译测试验证 DiT 模块透传。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 扩散管线；类别 source；类型 core-logic；符号 _maybe_torch_compile）：把 DiT 模块透传给编译 helper，是 out-of-tree 平台获得模块状态的前提。

关键符号：build_torch_compile_kwargs, get_compile_backend, get_compile_options, _maybe_torch_compile, test_out_of_tree_platform_controls_compile_kwargs

关键源码片段

python/sglang/multimodal_gen/runtime/utils/torch_compile.py

核心改造：build_torch_compile_kwargs 新增 module 参数和 out-of-tree 分支，决定 backend/options/mode 的优先级，是本次功能的主入口。

```
def build_torch_compile_kwargs(
    *, mode: str | None, module: nn.Module | None = None
) -> dict[str, object]:
    """构造传给 torch.compile 的关键字参数。

    out-of-tree 平台可同时指定 backend 与 per-module options；
    未提供 options 且使用 Inductor 时保留既有 mode 语义。
    """
    compile_kwargs: dict[str, object] = {"fullgraph": False, "dynamic": None}
    if current_platform.is_out_of_tree():
        # 通过平台钩子选择后端；基类默认返回 simple_compile_backend
        backend = current_platform.get_compile_backend(mode)
        compile_kwargs["backend"] = backend
        if module is not None:
            # 传入 DiT 模块，便于外部平台读取 persistent state 等模块状态
            options = current_platform.get_compile_options(module)
            if options is not None:
                compile_kwargs["options"] = options
        # options 与 mode 互斥：有显式 options 时不再下发 mode，
        # 仅当回退到 Inductor 且无 options 时才保留原 mode 行为
```

```
        if (
            "options" not in compile_kwargs
            and backend == "inductor"
            and mode is not None
        ):
            compile_kwargs["mode"] = mode
    elif current_platform.is_npu():
        # NPU 沿用 torchair 后端，并关闭 dynamic；延迟导入避免顶层耦合
        from sglang.srt.utils.common import get_compiler_backend

        compile_kwargs["backend"] = get_compiler_backend()
        compile_kwargs["dynamic"] = False
    elif mode is not None:
        # 内置平台（CUDA 等）维持原有 mode 路径
        compile_kwargs["mode"] = mode
    return compile_kwargs
```

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

新增参数化单测验证三种 backend/options 组合，并扩展区域编译测试验证 DiT 模块透传。

```
@pytest.mark.parametrize(
    ("backend", "options", "expected"),
    [
        # 自定义后端 + per-module 选项
        (
            "custom_backend",
            {"pass_manager_config": {"persistent_buffers": ["weight"]}},
            {
                "backend": "custom_backend",
                "options": {"pass_manager_config": {"persistent_buffers": ["weight"]}},
            },
        ),
        # Inductor + 显式选项：options 存在时不再下发 mode
        (
            "inductor",
            {"max_autotune": True},
            {"backend": "inductor", "options": {"max_autotune": True}},
        ),
        # Inductor + 无选项：回退到 mode 语义
        (
            "inductor",
            None,
            {"backend": "inductor", "mode": "max-autotune-no-cudagraphs"},
        ),
    ],
)
def test_out_of_tree_platform_controls_compile_kwargs(backend, options, expected):
    """Out-of-tree 平台钩子能选出合法的 backend、mode、options 组合。"""
    module = _CompilableModule()
```

```
with patch(
    "sglang.multimodal_gen.runtime.utils.torch_compile.current_platform.is_out_of_tree",
    return_value=True,
), patch(
    "sglang.multimodal_gen.runtime.utils.torch_compile.current_platform.get_compile_
    backend",
    return_value=backend,
) as get_compile_backend, patch(
    "sglang.multimodal_gen.runtime.utils.torch_compile.current_platform.get_compile_options",
    return_value=options,
) as get_compile_options:
    compile_kwargs = build_torch_compile_kwargs(
        mode="max-autotune-no-cudagraphs",
        module=module,
    )

assert compile_kwargs == {"dynamic": None, "fullgraph": False, **expected}
get_compile_backend.assert_called_once_with("max-autotune-no-cudagraphs")
get_compile_options.assert_called_once_with(module)
```

评论区精华

mickqian 在 interface.py 两个新方法的 diff 上先后评论 "could we avoid the del xxx?" 与 "ditto 🤔"，指出用 del 清理未使用参数的做法不干净。xuzijian629 两次回应 "Reflected, thanks!" 与 "Thanks for catching this! Resubmitted a clean version"，最终移除 del 语句，review 状态转为 APPROVED。无未解决的疑虑。

- interface.py 新方法中 del 语句的写法 (style): 作者两次回复已修复并提交干净版本，最终方法体不再包含 del 语句；review 最终 APPROVED。

风险与影响

- 风险：
  - 行为风险：out-of-tree 平台现在会进入新分支，若某平台覆写 get_compile_backend 返回非 inductor 后端，且调用方仍按 mode 语义理解日志，可能产生误导；但代码已确保 options/backend 与 mode 不共存，避免 torch.compile 参数冲突。
  - 回归风险：PR body 说明未运行完整 pytest，且 CI 的 Extra 与 AMD ROCm 7.2 任务失败（状态为 x），虽然与本次改动无直接证据关联，仍需在合入后的 nightly 中关注 diffusion 编译路径。
  - 兼容性：build_torch_compile_kwargs 新增参数为可选，NPU 分支行为完全不变；唯一的耦合变化是把 get_compiler_backend 改成延迟导入，反而降低了 multimodal_gen 对 sglang.srt 的顶层依赖。
  - 扩展点风险：get_compile_options 默认返回 None，外部平台若覆写出错会影响启动，需要更完善的接口文档与错误处理。
  - 影响：对默认用户（CUDA、NPU 等内置平台）完全无感知，行为保持不变；out-of-tree 平台获得标准化扩展点，接入成本显著降低。对系统而言，diffusion 编译路

径新增了平台钩子层，后续第三方硬件接入不再需要改动主仓核心逻辑。对团队而言，新增了两个需要维护的默认方法，但它们是纯增量且向后兼容的。

- 风险标记：options/mode 互斥逻辑，AMD/Extra CI 未通过，完整 pytest 未运行，out-of-tree 平台行为变更

关联脉络

- PR #35850 [Diffusion][minimax-h3] Restrict MiniMax-H3 SubBlock sparsity to video queries: 同属 multimodal_gen diffusion 运行时（attention/denoise 管线）的适配与优化，说明 diffusion 子系统在持续收敛平台行为。
- PR #36327 [Diffusion] Bound reusable Ulysses A2A staging buffers across shapes: 同为 diffusion runtime 层改动并配套单测，与本次平台钩子一起体现 diffusion 平台扩展方向的延续。