PR #23482 完整报告

sgl-project/sglang

[Diffusion][NPU]Add attention backends for diffusion models for Ascend NPU

合并时间：2026-05-19 17:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23482

执行摘要

- 一句话：NPU 扩散模型新增三种注意力后端
- 推荐动作：值得精读，尤其是如何在现有注意力抽象框架中新增后端，以及平台选择逻辑的 try-except 降级处理。对于 NPU 相关开发者，Rain Fusion 和 Block Sparse 的实现细节需关注 review 中的性能优化建议。

功能与动机

PR 描述指出 'Only SDPA and FIA backends available for Ascend NPU'，新增 Laser、Block Sparse 和 Rain Fusion Attention 可提供更灵活的稀疏注意力方案，提高扩散模型的推理效率。

实现拆解

1. 新增注意力后端类：在 backends/ 下创建三个文件，每个实现 AttentionBackend、AttentionImpl 及可选的 AttentionMetadata/AttentionMetadataBuilder。Laser Attention 封装 torch.ops.attentions.la，Block Sparse 封装 torch.ops.attentions.sparse_block_attention，Rain Fusion 组合 Laser 与自定义池化 / 索引。
2. 扩展平台选择逻辑：在 npu.py 的 get_attn_backend_cls_str 中添加三个 elif 分支，通过 try-import 检查依赖，返回对应后端类路径，缺失时抛出清晰安装指引。
3. 更新枚举与配置：在 interface.py 的 AttentionBackendEnum 中添加三个新枚举，并在 is_sparse 中标记。在 configs/models/adapter/base.py 和 dits/base.py 的支持列表中加入新枚举。
4. 集成到扩散流水线：在 denoising.py 的 _build_attn_metadata 中为 Block Sparse 和 Rain Fusion 构建元数据（当前时间步、稀疏度、潜空间形状等），Laser 无元数据。
5. 更新文档：在 docs_new 和 docs 目录下的注意力后端文档和兼容性矩阵中添加新后端说明，包括 CLI 参数示例和依赖安装方式。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/attention/backends/rain_fusion_attn.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 RainFusionAttentionBackend, RainFusionAttentionMetadata, RainFusionAttentionMetadataBuilder, RainFusionAttentionImpl）：核心新增文件：实现

Rain Fusion Attention 后端，包含完整的 Backend、Metadata、MetadataBuilder 和 Impl 类，通过激光注意力与自定义池化 / 索引融合提高稀疏注意力性能。

- python/sglang/multimodal_gen/runtime/layers/attention/backends/block_sparse_attn.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 BlockSparseAttentionBackend, BlockSparseAttentionMetadata, BlockSparseAttentionMetadataBuilder, BlockSparseAttentionImpl）：核心新增文件：实现 Block Sparse Attention 后端，支持块级稀疏注意力，通过元数据构建器将时间步、稀疏度与潜空间块对齐。
- python/sglang/multimodal_gen/runtime/layers/attention/backends/laser_attn.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 LaserAttentionBackend, LaserAttentionImpl）：核心新增文件：实现 Laser Attention 后端，封装 sgl-kernel-npu 的 la 内核，作为块稀疏和雨融合的基础依赖。
- python/sglang/multimodal_gen/runtime/platforms/npu.py（模块 平台层；类别 source；类型 dependency-wiring；符号 get_attn_backend_cls_str）：平台选择扩展：在 get_attn_backend_cls_str 中添加三个 elif 分支，通过 try-import 返回新后端类路径，并提供安装错误提示。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/denoising.py（模块 流水线；类别 source；类型 core-logic；符号 _build_attn_metadata）：流水线集成：在 _build_attn_metadata 中为 Block Sparse 和 Rain Fusion 构建元数据，包括时间步、稀疏度、潜空间形状等。
- python/sglang/multimodal_gen/runtime/platforms/interface.py（模块 配置枚举；类别 source；类型 core-logic；符号 AttentionBackendEnum, is_sparse）：枚举扩展：在 AttentionBackendEnum 中添加三个新枚举值，并在 is_sparse 方法中标记为稀疏后端。
- python/sglang/multimodal_gen/configs/models/adapter/base.py（模块 配置；类别 source；类型 data-contract）：配置更新：将新枚举加入支持的注意力后端列表，使其可通过 CLI 访问。
- python/sglang/multimodal_gen/configs/models/dits/base.py（模块 配置；类别 source；类型 data-contract）：配置更新：与 adapter/base.py 类似，将新枚举加入 DiT 架构支持列表。
- docs_new/docs/sglang-diffusion/attention_backends.mdx（模块 文档；类别 docs；类型 documentation）：文档更新：在新版文档站点中添加新后端的使用说明和 CLI 参数示例。
- docs_new/docs/sglang-diffusion/compatibility_matrix.mdx（模块 文档；类别 docs；类型 documentation）：文档更新：在新版文档的兼容性矩阵中添加新后端与各模型的兼容性标记。
- docs/diffusion/compatibility_matrix.md（模块 文档；类别 docs；类型 documentation）：文档同步：在旧版文档的兼容性矩阵中同步添加新后端支持信息。
- docs/diffusion/performance/attention_backends.md（模块 文档；类别 docs；类型 documentation）：文档更新：在旧版性能文档中添加新后端配置说明和性能提升提示。

关键符号：RainFusionAttentionBackend.get_supported_head_sizes, RainFusionAttentionMetadataBuilder.build, BlockSparseAttentionBackend.get_metadata_cls, BlockSparseAttentionMetadataBuilder.build, LaserAttentionBackend.get_enum,

LaserAttentionImpl._la_preprocess_input, NPUPlatform.get_attn_backend_cls_str,DenoisingStage._build_attn_metadata

关键源码片段

python/sglang/multimodal_gen/runtime/layers/attention/backends/rain_fusion_attn.py

核心新增文件：实现 Rain Fusion Attention 后端，包含完整的 Backend、Metadata、MetadataBuilder 和 Impl 类，通过激光注意力与自定义池化 / 索引融合提高稀疏注意力性能。

```
import math
from dataclasses import dataclass
from typing import Any, List, Optional

import attentions # noqa: F401
import torch
from einops import rearrange

from sglang.multimodal_gen.runtime.layers.attention.backends.attention_backend import (
    AttentionBackend, AttentionImpl, AttentionMetadata, AttentionMetadataBuilder,
)
from sglang.multimodal_gen.runtime.layers.attention.backends.laser_attn import
LaserAttentionBackend
from sglang.multimodal_gen.runtime.platforms import AttentionBackendEnum
from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger

logger = init_logger(__name__)


class RainFusionAttentionBackend(AttentionBackend):
    """Rain Fusion 注意力后端，封装了融合池化与激光注意力的稀疏方案"""
    accept_output_buffer: bool = True

    @staticmethod
    def get_supported_head_sizes() -> list[int]:
        return [32, 64, 96, 128]

    @staticmethod
    def get_enum() -> AttentionBackendEnum:
        return AttentionBackendEnum.RAIN_FUSION_ATTN

    @staticmethod
    def get_impl_cls() -> type["RainFusionAttentionImpl"]:
        return RainFusionAttentionImpl

    @staticmethod
    def get_metadata_cls() -> type["RainFusionAttentionMetadata"]:
        return RainFusionAttentionMetadata
```

```
    @staticmethod
    def get_builder_cls() -> type["RainFusionAttentionMetadataBuilder"]:
        return RainFusionAttentionMetadataBuilder


@dataclass
class RainFusionAttentionMetadata(AttentionMetadata):
    current_timestep: int
    skip_first_steps: int
    sparsity: float
    latent_shape: list[int]


class RainFusionAttentionMetadataBuilder(AttentionMetadataBuilder):
    """构建 Rain Fusion 所需的元数据：验证稀疏度与步数，计算 latent 形状"""
    def __init__(self) -> None:
        pass

    def prepare(self) -> None:
        pass

    def build(self, current_timestep: int, skip_first_steps: int, sparsity: float,
              raw_latent_shape: list[int], patch_size: tuple[int, int, int],
              **kwargs: dict[str, Any]) -> RainFusionAttentionMetadata:
        if not (skip_first_steps >= 0 and 0.0 <= sparsity < 1.0):
            raise ValueError(
                f"Sparsity 应在 [0, 1) 范围，skip_first_steps 应非负，"
                f"当前：sparsity={sparsity}, skip_first_steps={skip_first_steps}"
            )
        if sparsity == 0.0:
            logger.warning("稀疏度为 0，建议使用 Laser Attention 或增加稀疏度以提升性能。")
        # 将 raw_latent_shape 按 patch_size 分割得到 latent 形状
        latent_shape = raw_latent_shape[-3:]
        latent_shape = [latent_shape[i] // patch_size[i] for i in range(3)]
        return RainFusionAttentionMetadata(
            current_timestep=current_timestep,
            skip_first_steps=skip_first_steps,
            sparsity=sparsity,
            latent_shape=latent_shape,
        )
```

python/sglang/multimodal_gen/runtime/layers/attention/backends/block_sparse_attn.py

核心新增文件：实现 Block Sparse Attention 后端，支持块级稀疏注意力，通过元数据构建器将时间步、稀疏度与潜空间块对齐。

```
from dataclasses import dataclass
from typing import Any
```

```
import attentions # noqa: F401
import torch

from sglang.multimodal_gen.runtime.layers.attention.backends.attention_backend import (
    AttentionBackend, AttentionImpl, AttentionMetadata, AttentionMetadataBuilder,
)
from sglang.multimodal_gen.runtime.layers.attention.backends.laser_attn import
LaserAttentionBackend
from sglang.multimodal_gen.runtime.platforms import AttentionBackendEnum
from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger

logger = init_logger(__name__)
BSA_BLOCK_SIZE = 128 # 块稀疏的块大小


class BlockSparseAttentionBackend(AttentionBackend):
    """Block Sparse Attention 后端，通过块级掩码实现稀疏注意力"""
    accept_output_buffer: bool = True

    @staticmethod
    def get_supported_head_sizes() -> list[int]:
        return [32, 64, 96, 128]

    @staticmethod
    def get_enum() -> AttentionBackendEnum:
        return AttentionBackendEnum.BLOCK_SPARSE_ATTN

    @staticmethod
    def get_impl_cls() -> type["BlockSparseAttentionImpl"]:
        return BlockSparseAttentionImpl

    @staticmethod
    def get_metadata_cls() -> type["BlockSparseAttentionMetadata"]:
        return BlockSparseAttentionMetadata

    @staticmethod
    def get_builder_cls() -> type["BlockSparseAttentionMetadataBuilder"]:
        return BlockSparseAttentionMetadataBuilder


@dataclass
class BlockSparseAttentionMetadata(AttentionMetadata):
    current_timestep: int
    skip_first_steps: int
    sparsity: float
    block_frame_stride: int


class BlockSparseAttentionMetadataBuilder(AttentionMetadataBuilder):
```

```
    """构建元数据：校验参数，计算 block_frame_stride 以保持首帧非稀疏"""
    def __init__(self) -> None:
        pass

    def prepare(self) -> None:
        pass

    def build(self, current_timestep: int, skip_first_steps: int, sparsity: float,
              raw_latent_shape: list[int], patch_size: tuple[int, int, int],
              **kwargs: dict[str, Any]) -> BlockSparseAttentionMetadata:
        if not (skip_first_steps >= 0 and 0.0 <= sparsity < 1.0):
            raise ValueError(
                f"Sparsity 应在 [0,1) 范围，skip_first_steps 应非负，"
                f"当前：sparsity={sparsity}, skip_first_steps={skip_first_steps}"
            )
        if sparsity == 0.0:
            logger.warning(
                "稀疏度为 0 将不丢弃任何 token，"
                "建议使用 Laser Attention 或增加稀疏度。"
            )
        # 从 raw_latent_shape 提取高宽，除以 patch_size 得到 block 数
        if len(raw_latent_shape) >= 5:
            latent_height, latent_width = raw_latent_shape[3:5]
        else:
            latent_height, latent_width = raw_latent_shape[-2:]
        latent_height //= patch_size[1]
        latent_width //= patch_size[2]
        frame_stride = latent_height * latent_width
        # 将 frame_stride 按 BSA_BLOCK_SIZE 对齐
        block_frame_stride = (frame_stride + BSA_BLOCK_SIZE - 1) // BSA_BLOCK_SIZE
        return BlockSparseAttentionMetadata(
            current_timestep=current_timestep,
            skip_first_steps=skip_first_steps,
            sparsity=sparsity,
            block_frame_stride=block_frame_stride,
        )
```

python/sglang/multimodal_gen/runtime/layers/attention/backends/laser_attn.py

核心新增文件：实现 Laser Attention 后端，封装 sgl-kernel-npu 的 la 内核，作为块稀疏和雨融合的基础依赖。

```
import torch

from sglang.multimodal_gen.runtime.layers.attention.backends.attention_backend import (
    AttentionBackend, AttentionImpl, AttentionMetadata,
)
from sglang.multimodal_gen.runtime.layers.attention.backends.sdpa import SDPABackend
from sglang.multimodal_gen.runtime.platforms import AttentionBackendEnum
```

```
from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger

# 导入 torch.ops.attentions，该模块由 sgl_kernel_npu 提供
try:
    import attentions # noqa: F401
except ImportError as e:
    raise ImportError(
        "需要安装 sgl_kernel_npu 以提供 attentions 包"
    ) from e

logger = init_logger(__name__)


class LaserAttentionBackend(AttentionBackend):
    """Laser Attention 后端封装，无元数据构建需求"""
    accept_output_buffer: bool = True

    @staticmethod
    def get_supported_head_sizes() -> list[int]:
        return [32, 64, 96, 128]

    @staticmethod
    def get_enum() -> AttentionBackendEnum:
        return AttentionBackendEnum.LASER_ATTN

    @staticmethod
    def get_impl_cls() -> type["LaserAttentionImpl"]:
        return LaserAttentionImpl


class LaserAttentionImpl(AttentionImpl):
    """Laser Attention 实现，通过 torch.ops.attentions.la 调用 NPU 内核"""
    def __init__(self, num_heads: int, head_size: int, causal: bool, softmax_scale: float,
                 num_kv_heads: int | None = None, prefix: str = "", **extra_impl_args) -> None:
        self.softmax_scale = softmax_scale
        # 预处理后张量布局预期为 BNSD
        self.seqlen_base = 256
        self.seqlen_index = 2
        self.dim_index = 3
        self.dim_base = 128
        self.max_token = 2**31 - 1
        self.seq_len_pad_base = 256
        # 激光注意力算子在短序列上存在问题，低于 2048 时使用 SDPA 回退
        self.min_seqlen = 2048
        self.sdpa_impl = SDPABackend.get_impl_cls()(
            num_heads, head_size, causal, softmax_scale,
            num_kv_heads, prefix, **extra_impl_args,
        )
```

评论区精华

讨论集中在代码质量、性能优化和依赖影响上。ping1jing2 担心 import attentions 会破坏 GPU CI，Napkin-AI 回应此库是 sgl-kernel-npu 的一部分且当前测试未使用，不会造成影响。对于 _avgpool、_get_mask_index 函数的性能，ping1jing2 建议使用 F.pad 替代 torch.zeros+cat、避免 torch.einsum，Napkin-AI 认为当前版本已较优但部分接受。BSND 布局支持问题被提出，Napkin-AI 确认内核有精度问题，暂时维持转置方案并添加 TODO。

- 导入 attentions 库对 GPU CI 的影响 (question): 确认不影响现有 CI，无需修改 pyproject.toml。
- 使用 F.pad 优化 torch.zeros+cat (performance): 已采纳，使用 pad 实现。
- BSND 布局支持讨论 (design): 暂不支持 BSND，添加 TODO 标注。
- Rain Fusion 实现中长上下文的性能优化 (performance): 未采纳具体优化，保持当前实现。
- 缺少单元测试 (testing): 待未来添加。

风险与影响

- 风险：
  - 缺少测试覆盖：三个新后端无对应单元测试，仅靠集成测试验证，回归风险较高。
  - 依赖自定义库：均依赖 sgl-kernel-npu 的 attentions 库，库接口变更或安装失败将导致后端不可用。
  - 性能风险：Rain Fusion 的 _avgpool 和 _get_mask_index 被指存在内存 / 延迟优化空间，长序列场景下可能不达预期。
  - 兼容性限制：Laser Attention 不支持交叉注意力，无法用于所有模型。
  - 影响：用户影响：NPU 扩散模型用户现在可选用三种新注意力后端，通过 --attention-backend 和 --attention-backend-config 灵活调整速度与精度。系统影响：代码量增加约 1150 行，新增三个后端模块，增加维护成本。团队影响：需跟进 sgl-kernel-npu 发布并规划集成测试。
  - 风险标记：缺少测试覆盖，依赖 sgl-kernel-npu 内核，长上下文性能风险，跨注意力兼容性限制

关联脉络

- PR #22338 [diffusion][npu][quant] Add MXFP4 quantization support for Wan2.2 Diffusion on Ascend NPU: 同为 NPU 扩散模型的性能优化功能，扩展了 NPU 上的推理能力，与当前 PR 共同完善 NPU 扩散加速方案。