RoPE-Matrix
昇腾亲和的RoPE算法实现与算子开源

中央媒体技术院-图像工程部 陈敏琪
2025年12月20日

Content
目录

RoPE-Matrix算法原理

01

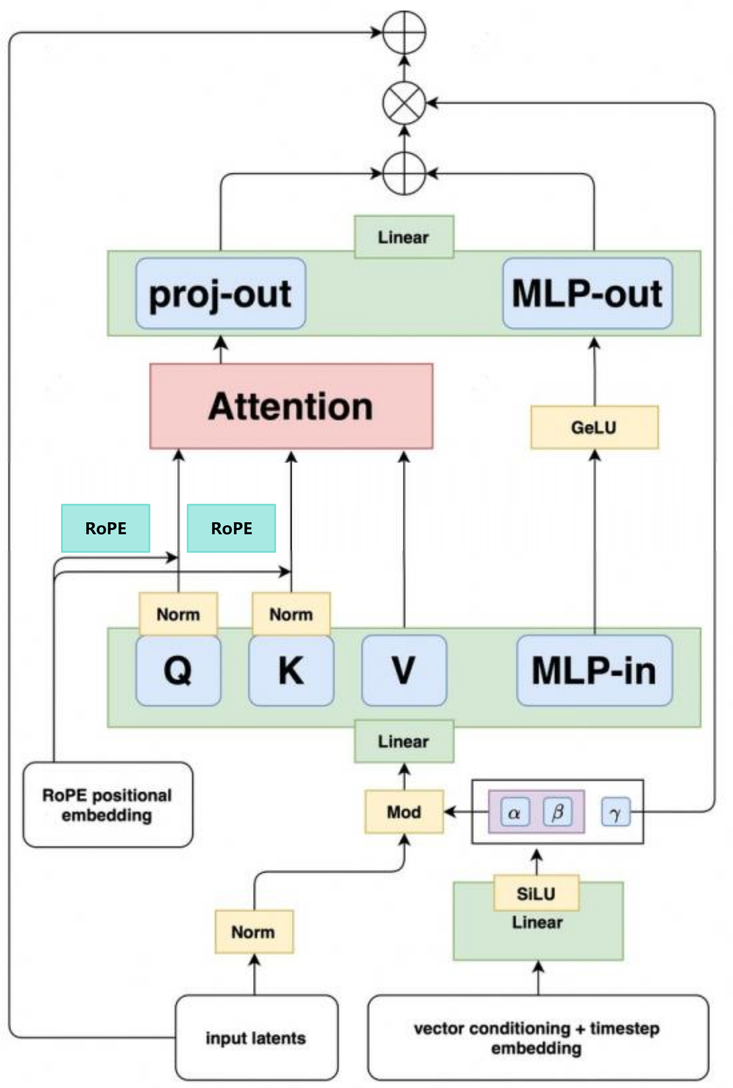
RoPE-Matrix：昇腾亲和的RoPE算法实现

多模态生成推理性能分析：Vector占比高，其中RoPE相关计算占到Vector的44%开销。

FLUX.1-Kontext性能分析

性能拆解	910B	占比
Cube类算子(MatMul等)	0.452	42.6%
CV融合算子(FlashAttention)	0.348	34.8%
Vector算子	**0.238**	**22.5%**
端到端耗时【秒】	**1.059**	**100%**

A fused DiT block equipped with rotary positional embeddings



基于单个block的vector前向计算打开分析：RoPE占到近一半开销

类别	耗时[ms]	占比
RoPE	**0.110**	**44%**
GeLU	0.026	11%
LN	0.022	9%
TransData	0.021	8%
Concat	0.021	8%
Mul	0.013	6%

RoPE-Matrix：昇腾亲和的RoPE算法实现

2021.04: Roformer-Enhanced Transformer With Rotary Position Embedding提出当前广泛使用的RoPE，旋转位置编码，**应考虑任意两个token直接的相对位置信息**，被广泛应用于LLaMA、ChatGLM、PaLM等当今最先进的大模型中。

①

$$f_q(\boldsymbol{x}_m, m) = (\boldsymbol{W}_q \boldsymbol{x}_m) e^{im\theta}$$
$$f_k(\boldsymbol{x}_n, n) = (\boldsymbol{W}_k \boldsymbol{x}_n) e^{in\theta}$$
$$g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) = \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}]$$

考虑token的dim=2场景，第m、n个token之间的相关性分数，可以通过**复平面上的旋转**，保证了**相对位置信息的显式编码**及**稳定性**（不改tensor的模长）。

②

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

复数计算对软硬件支持有要求，可以转换成**旋转矩阵乘**实现等价功能

③

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$$

对于token的dim=d场景，可以**对d做两两分组**，分别做不同角度的旋转，实现不同的相对位置编码

④

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{d-2} \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_0 \\ \cos m\theta_0 \\ \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2-1} \\ \cos m\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -x_1 \\ x_0 \\ -x_3 \\ x_2 \\ \vdots \\ -x_{d-1} \\ x_{d-2} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_0 \\ \sin m\theta_0 \\ \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2-1} \\ \sin m\theta_{d/2-1} \end{pmatrix}$$

由于R矩阵的**稀疏性**，直接用矩阵乘法实现会很浪费算力，可以转换为**逐元素Vector乘实现**

⑤

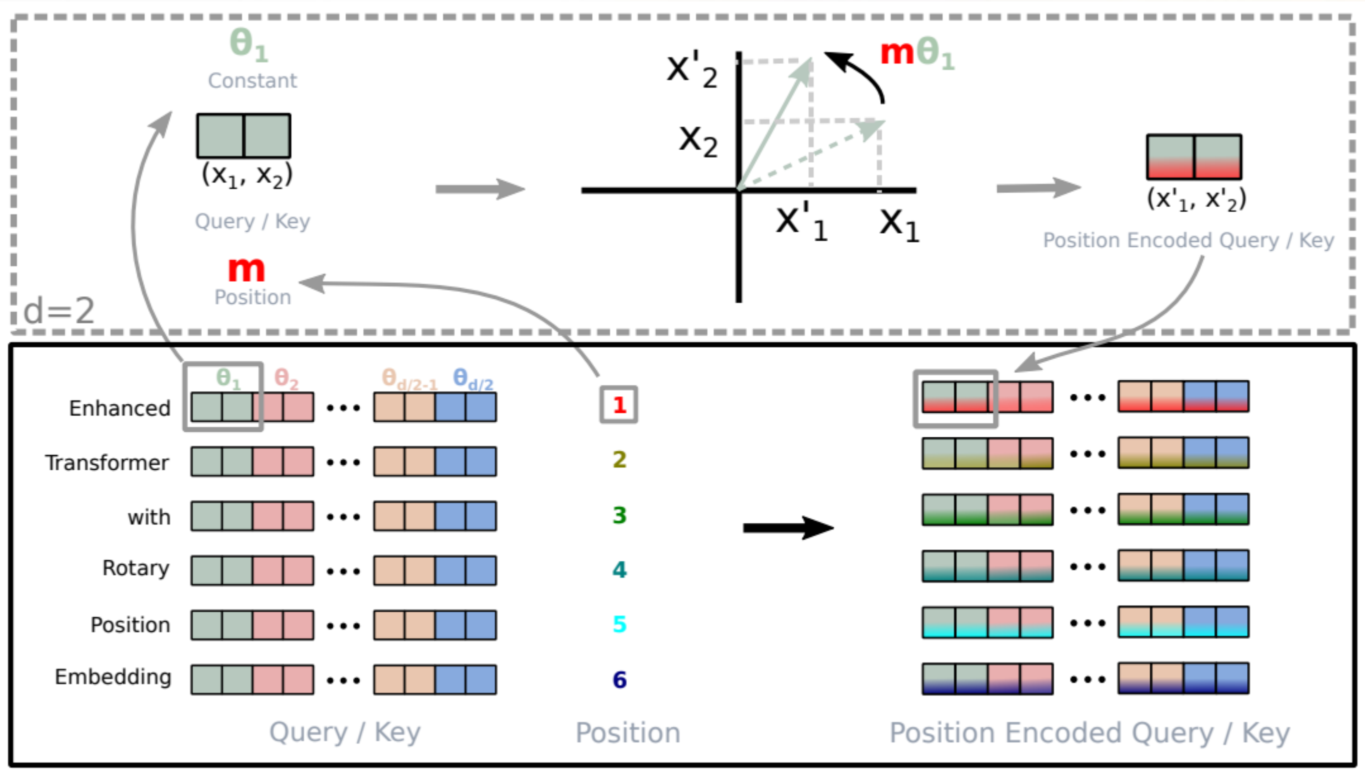
- half模式：

```
x1, x2 = torch.chunk(input, 2, -1)
x_new = torch.cat((-x2, x1), dim=-1)
output = r1 * input + r2 * x_new
```

- interleave模式：

```
x1 = input[..., ::2]
x2 = input[..., 1::2]
x_new = rearrange(torch.stack((-x2, x1), dim=-1), "... d two -> ...(d two)", two=2)
output = r1 * input + r2 * x_new
```

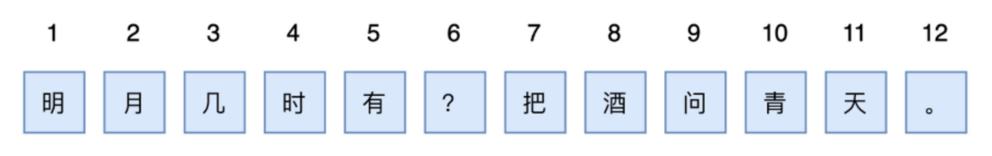
根据公式④的代码实现。实际社区中会分为**half**及**interleave**两种模式，区别仅在于dim维度两两分组的策略不同，数学意义等价。



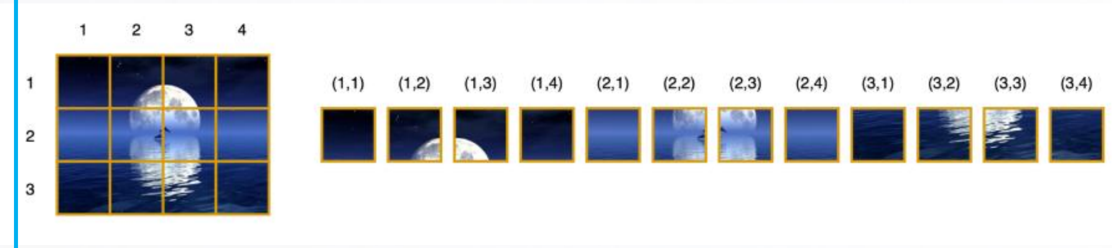
RoPE-Matrix：昇腾亲和的RoPE算法实现

2024.03： VisionLLaMA: A Unified LLaMA Backbone for Vision Tasks提出RoPE-2D，**位置编码信息考虑图像Patch的x,y坐标**，当前被广泛应用于多模态大模型中。

⑥ **LLM领域的输出为文本序列，为1维特性**

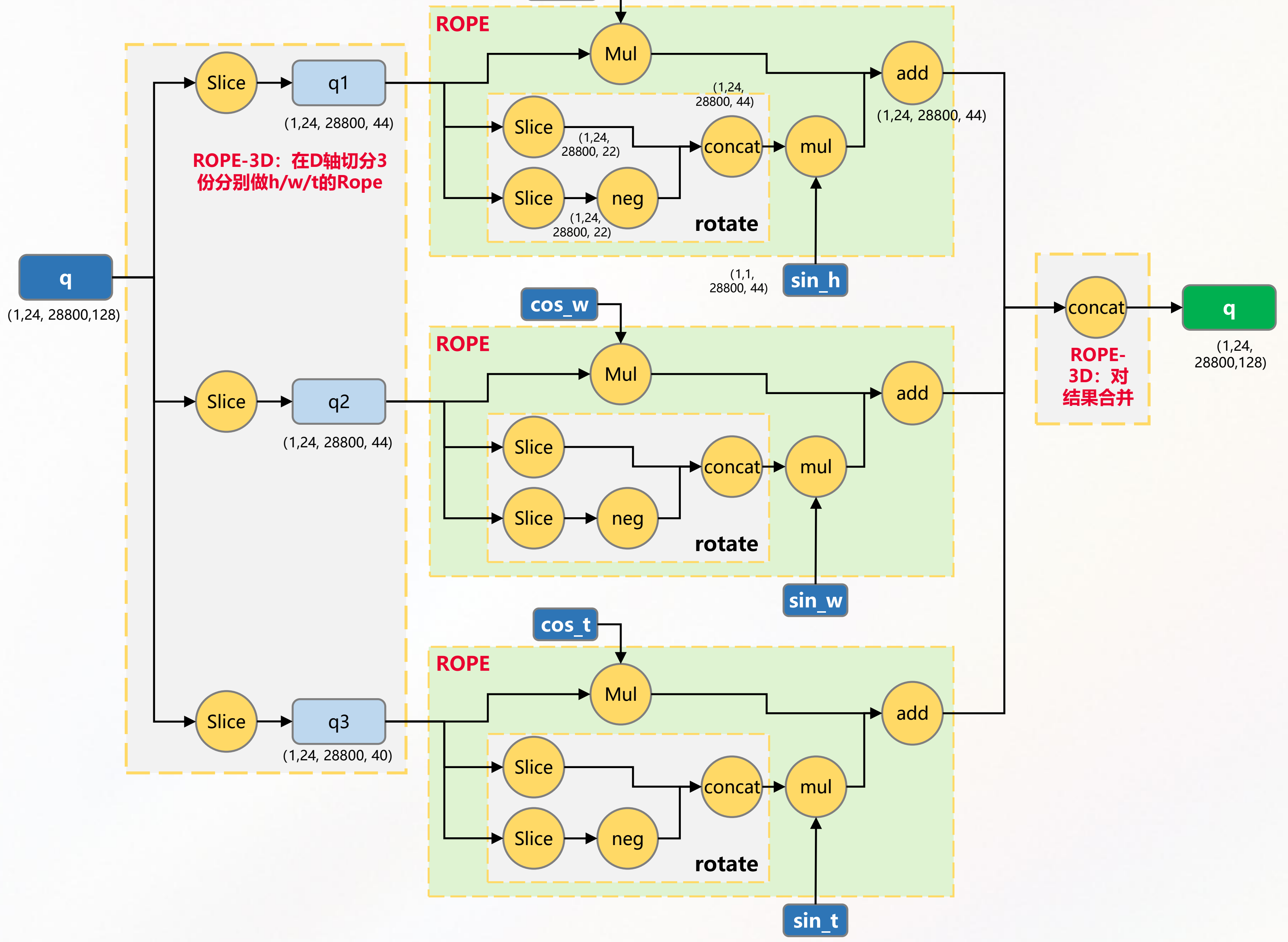


⑦ **针对多模态场景的输入为图像/视频，特征为2D/3D特性，ROPE实现也拓展为ROPE-2D/3D，同时注入HW或HWD位置信息。**



$$\mathcal{R}_{x,y} = \left(\begin{array}{cc|cc} \cos x\theta & -\sin x\theta & 0 & 0 \\ \sin x\theta & \cos x\theta & 0 & 0 \\ \hline 0 & 0 & \cos y\theta & -\sin y\theta \\ 0 & 0 & \sin y\theta & \cos y\theta \end{array}\right) = \begin{pmatrix} \mathcal{R}_x & 0 \\ 0 & \mathcal{R}_y \end{pmatrix}$$

⑧ **视频生成RoPE-3D实现**

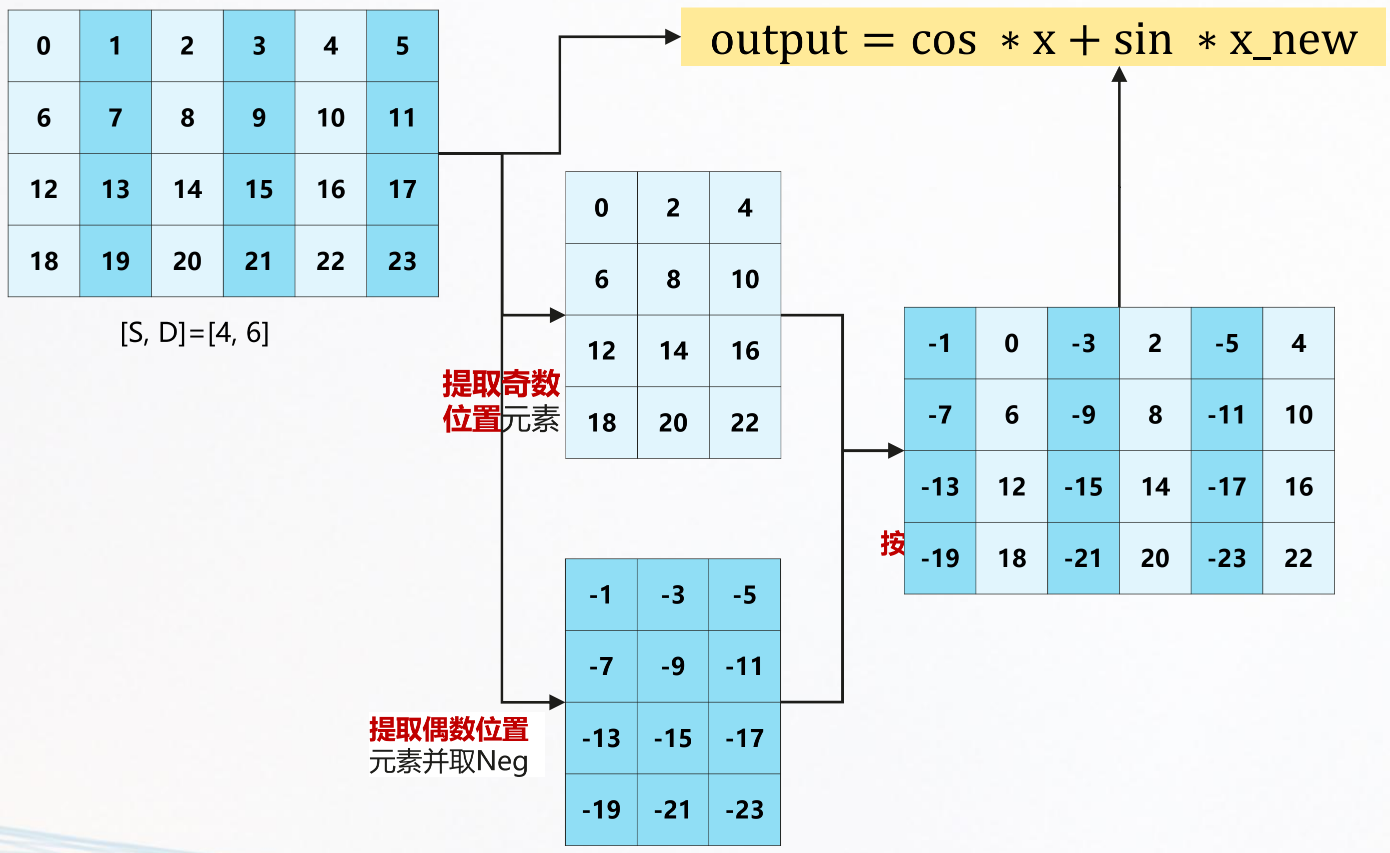


RoPE-Matrix：昇腾亲和的RoPE算法实现

Interleave模式

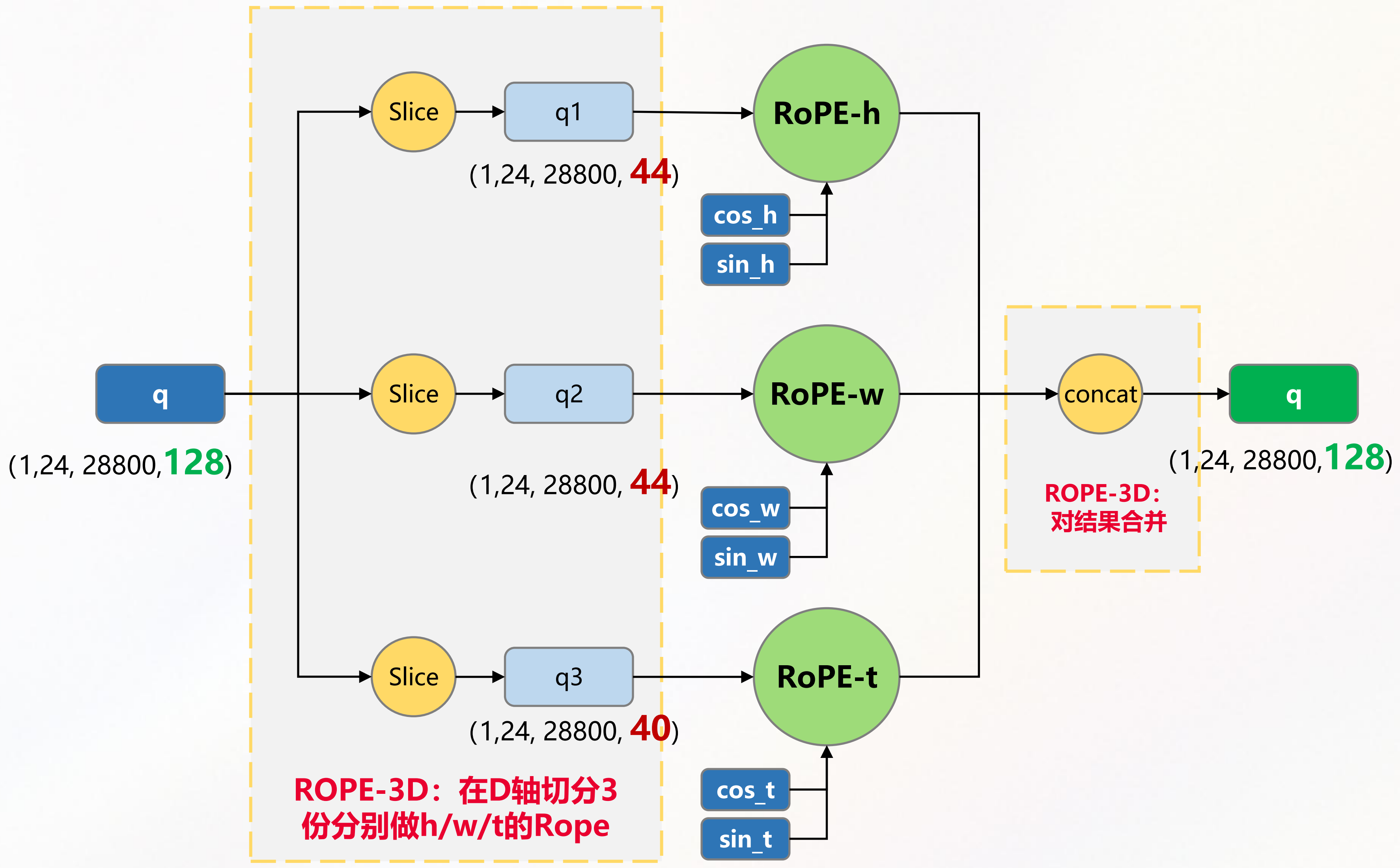
问题1：interleave模式涉及到对元素奇偶数位操作，昇腾不亲和；

$$
\boldsymbol{R}_{\Theta,m}^{d}\boldsymbol{x}=\begin{pmatrix}x_0\\x_1\\x_2\\x_3\\\vdots\\x_{d-2}\\x_{d-1}\end{pmatrix}\otimes\begin{pmatrix}\cos m\theta_0\\\cos m\theta_0\\\cos m\theta_1\\\cos m\theta_1\\\vdots\\\cos m\theta_{d/2-1}\\\cos m\theta_{d/2-1}\end{pmatrix}+\begin{pmatrix}-x_1\\x_0\\-x_3\\x_2\\\vdots\\-x_{d-1}\\x_{d-2}\end{pmatrix}\otimes\begin{pmatrix}\sin m\theta_0\\\sin m\theta_0\\\sin m\theta_1\\\sin m\theta_1\\\vdots\\\sin m\theta_{d/2-1}\\\sin m\theta_{d/2-1}\end{pmatrix}
$$



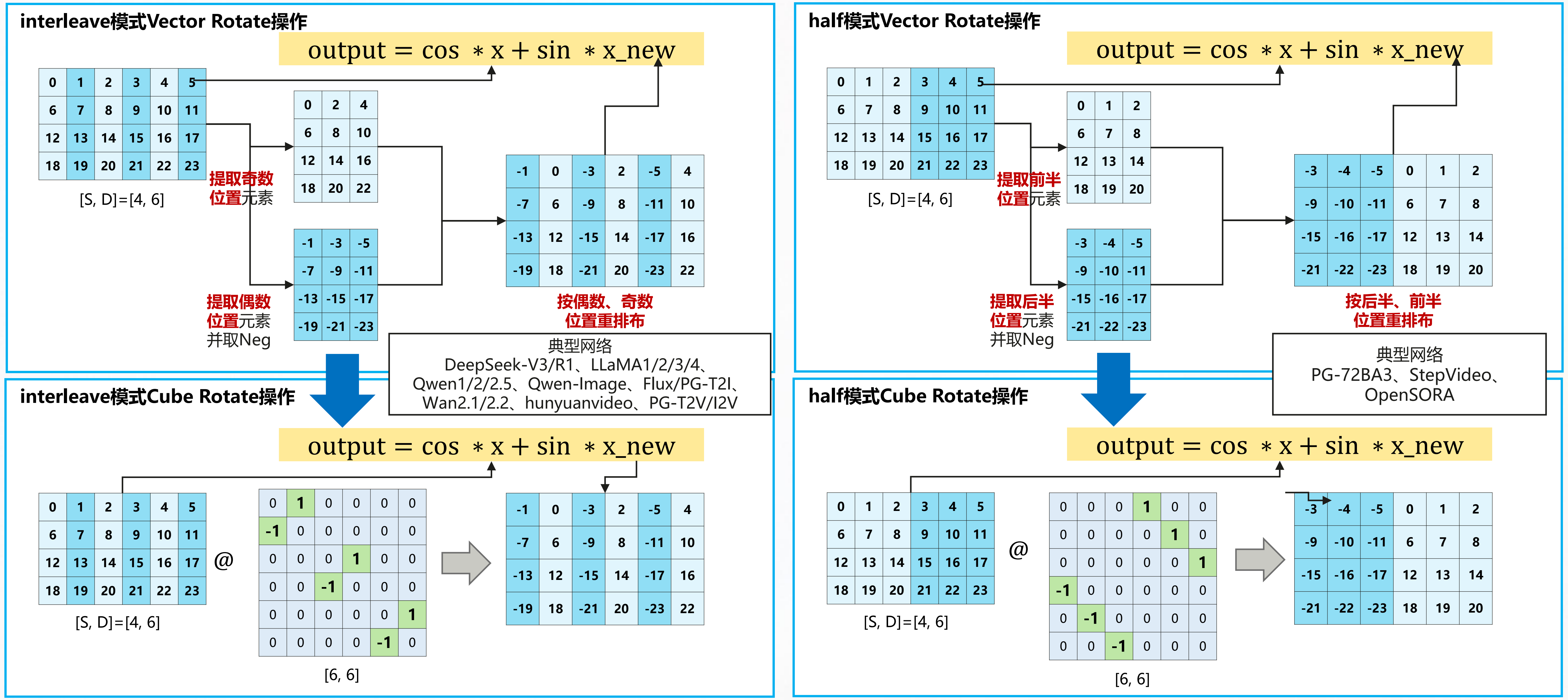
RoPE-3D模式

问题2：RoPE-3D模式下，会对数据进行切片与Concat操作，引入新开销
问题3：切片后的数据，尾轴为44/44/40，后续RoPE-h/w/t计算/搬运效率低



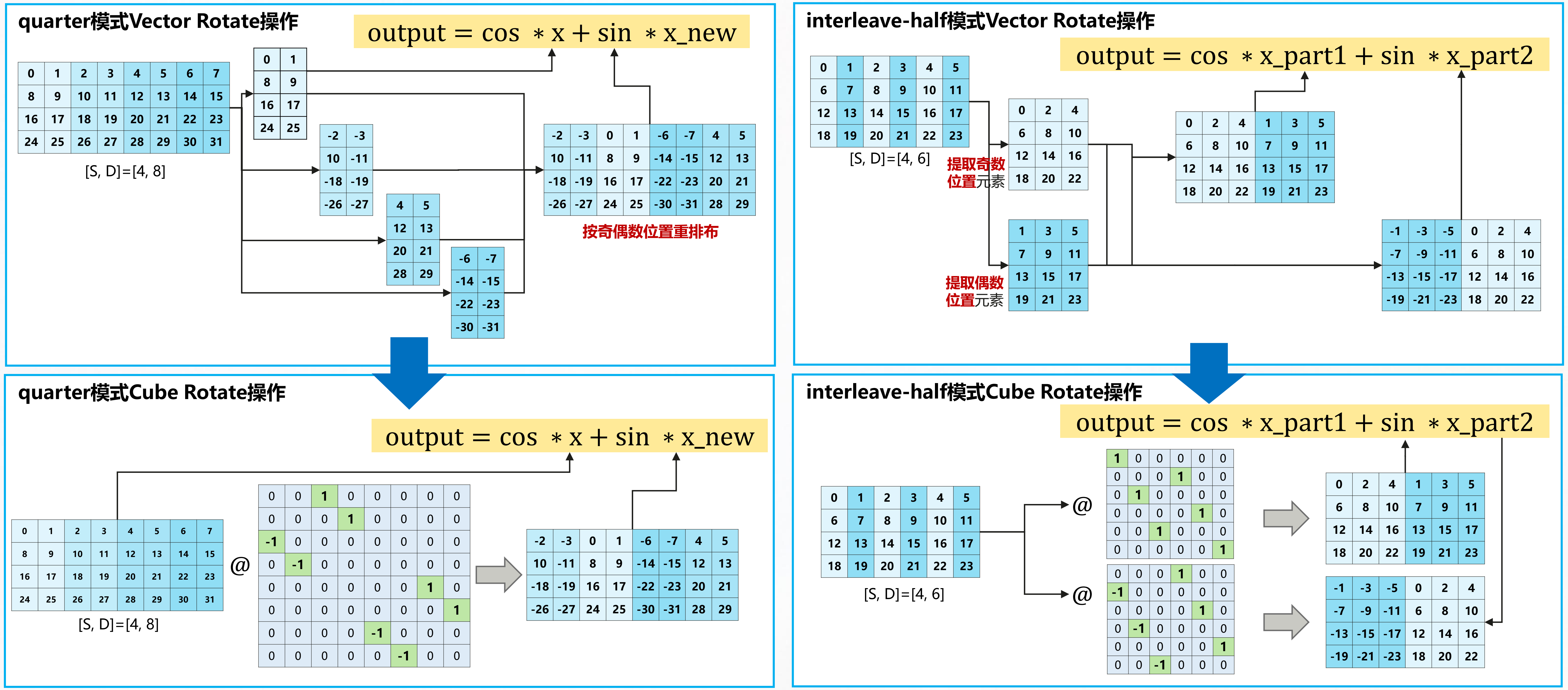
RoPE-Matrix：昇腾亲和的RoPE算法实现

Step1：以Cube计算替代Vector计算的新RoPE实现：针对原RoPE中存在大量的对数据切分与合并操作，通过构造对应的操作矩阵以矩阵乘实现相关功能，以此替代昇腾不亲和的Vector操作。



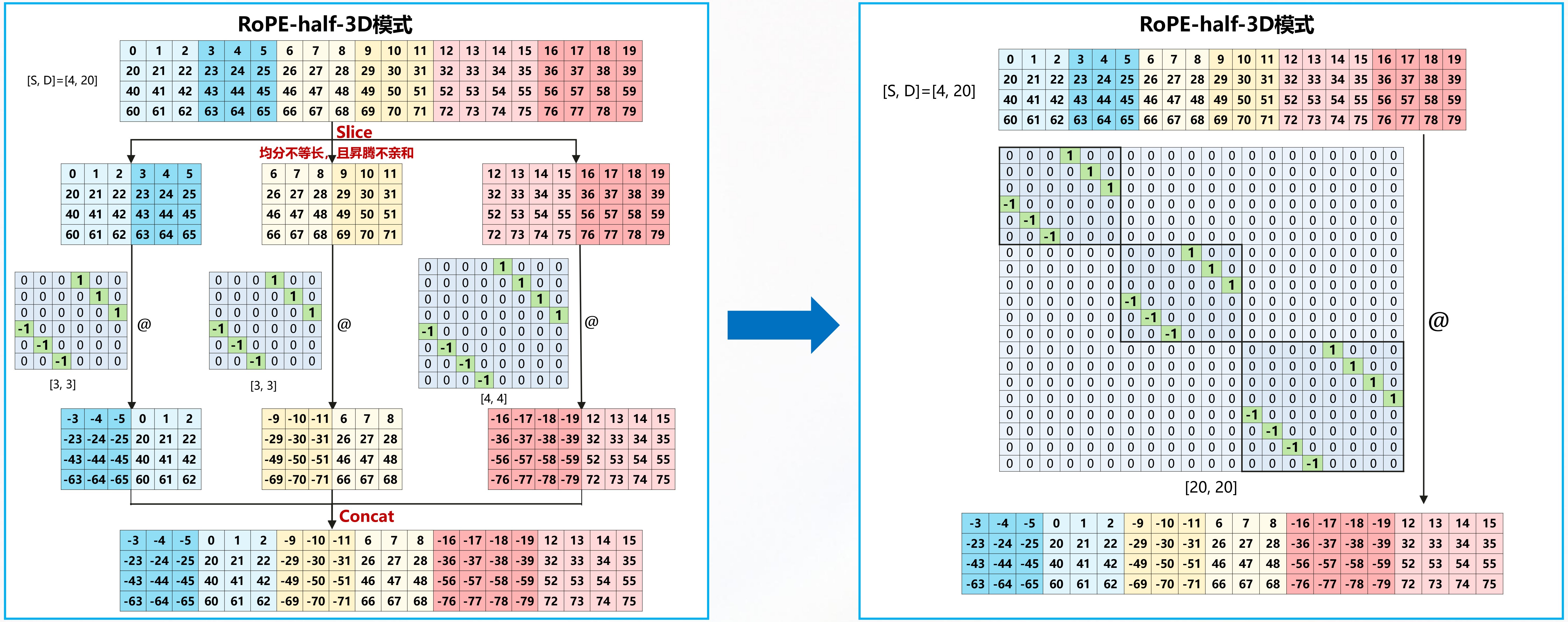
RoPE-Matrix：昇腾亲和的RoPE算法实现

Step1：以Cube计算替代Vector计算的新RoPE实现：针对原RoPE中存在大量的对数据切分与合并操作，通过构造对应的操作矩阵以矩阵乘实现相关功能，以此替代昇腾不亲和的Vector操作。



RoPE-Matrix：昇腾亲和的RoPE算法实现

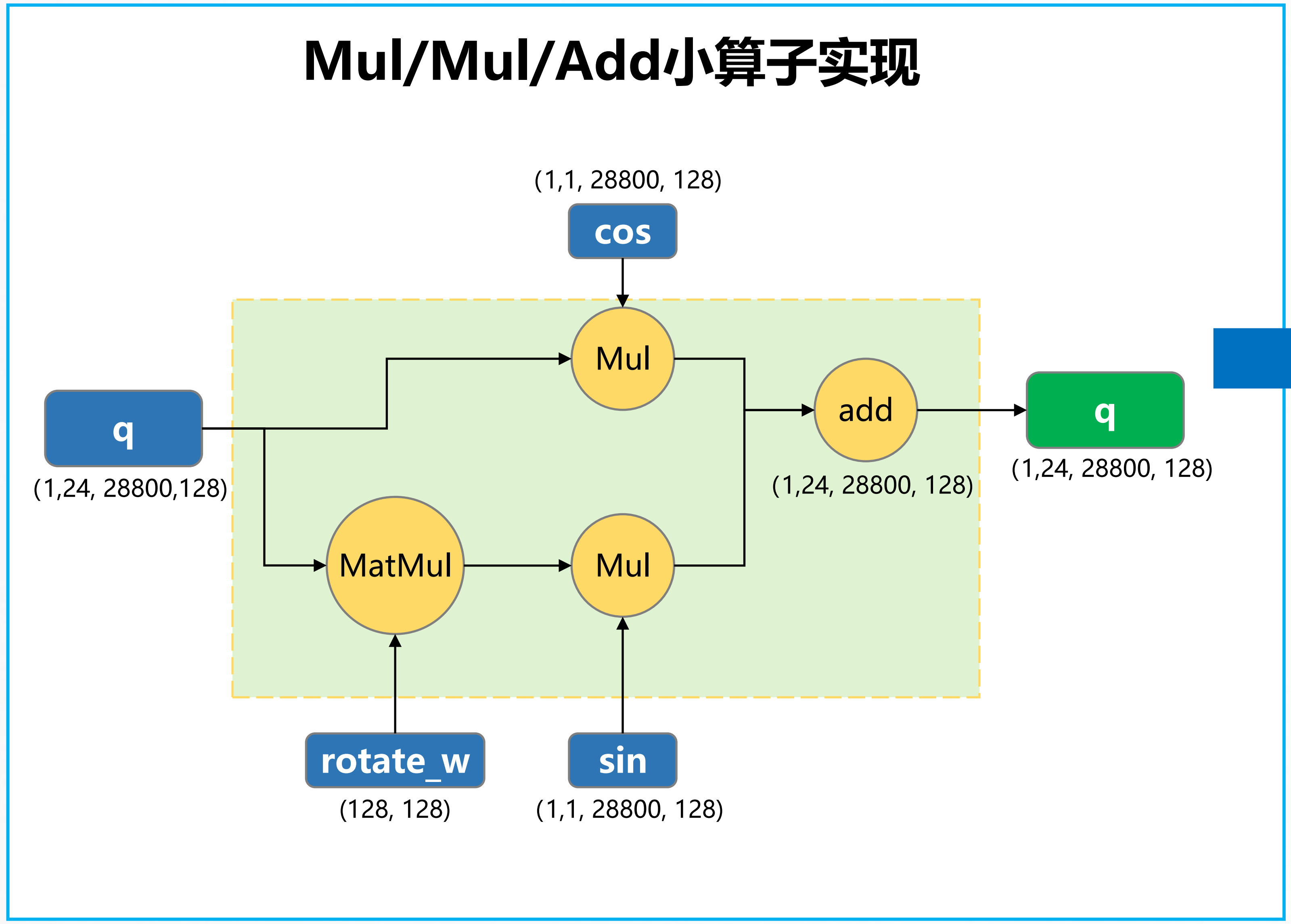
Step2：RoPE-2D/3D模式下， 不同数据的Cube Rotate可以合并为更大的rotate_mat，一次性实现完整输入的全部计算操作。

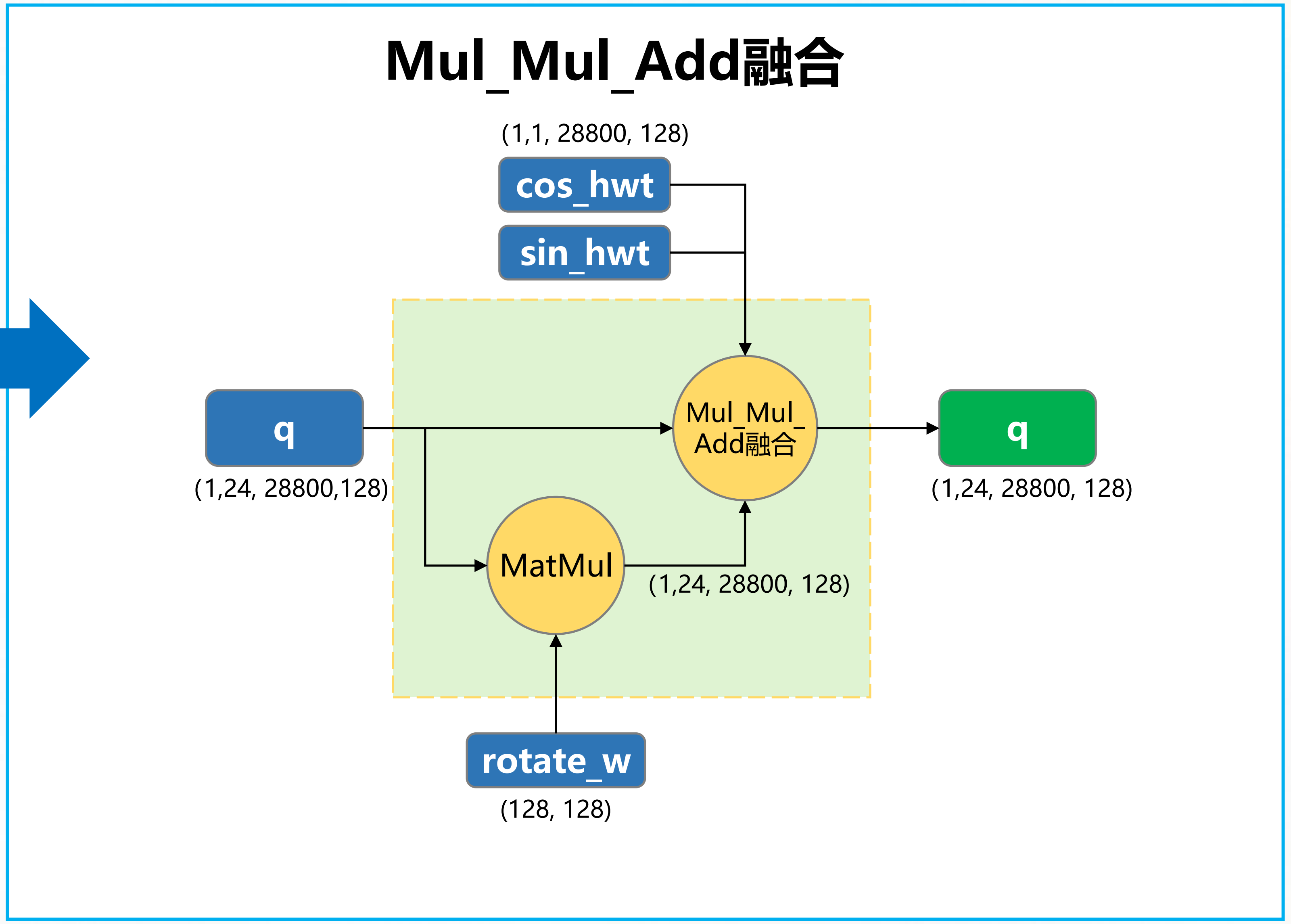


以S=4, D=20举例，通常D轴=128

RoPE-Matrix：昇腾亲和的RoPE算法实现

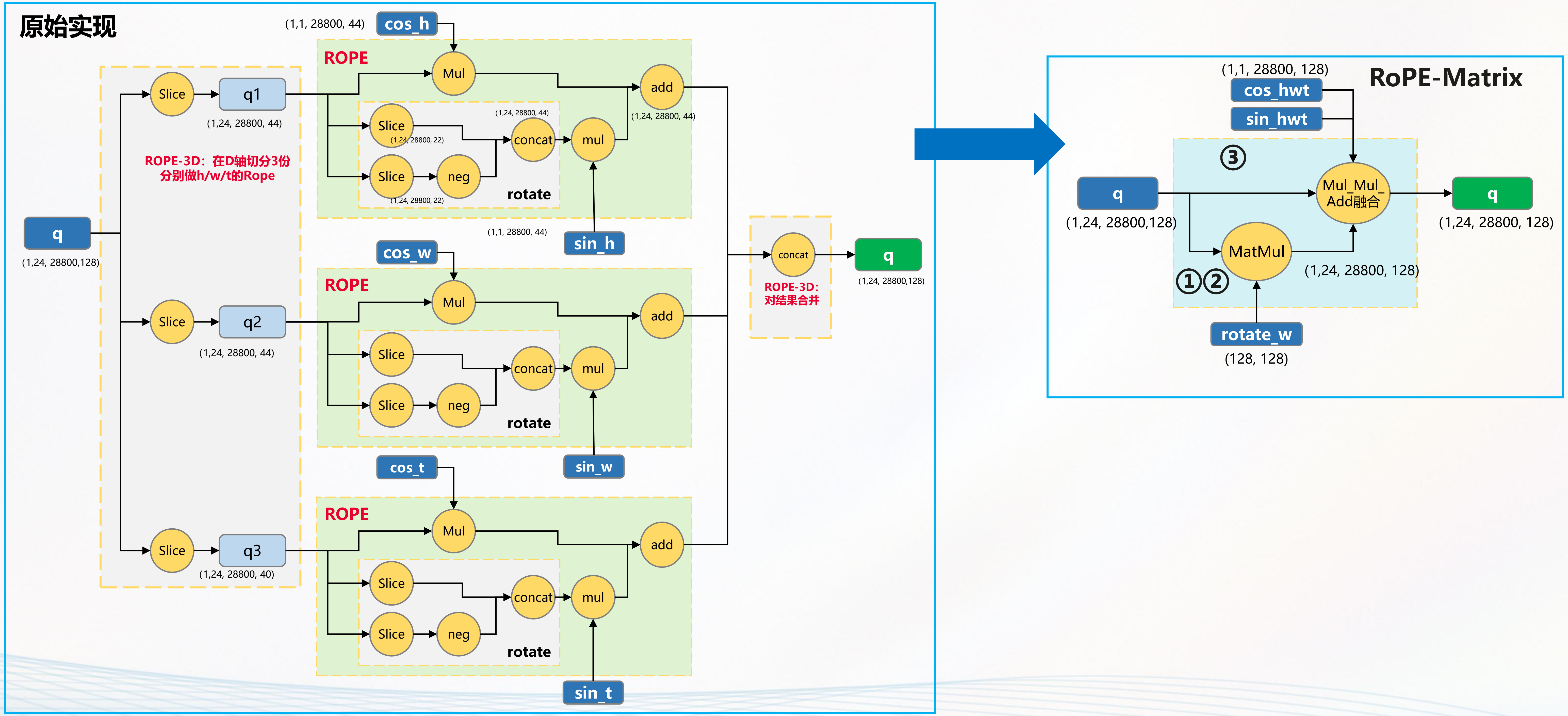
Step3：对Vector计算实现编译融合，进一步实现加速。





RoPE-Matrix：昇腾亲和的RoPE算法实现

Step1：以Cube计算替代Vector计算的新RoPE实现；
Step2：RoPE-2D/3D模式下， 用大旋转矩阵替换Slice-Concat操作；
Step3：引入torch.compile对Vector计算实现编译融合。



RoPE-Matrix：昇腾亲和的RoPE算法实现

单算子收益：相比小算子实现均有显著性正收益（平均提速300%），interleave（平均提速130%）和half场景（267%）下相比融合大算子也有明显收益，覆盖绝大部分典型网络。

<table>
<tr><th colspan="2">模式
bf16，[1, 24, 28800, 128]</th><th>典型网络</th><th>小算子实现
(无约束)
耗时：ms</th><th>CANN大算子
(约束条件)
耗时：ms</th><th>RoPE-Matrix
耗时：ms</th><th>相比
小算子</th><th>相比
CANN大算子</th></tr>
<tr><td rowspan="3">interleave</td><td>RoPE-1D
dim=128</td><td>DeepSeek-V3/R1
LLaMA1/2/3/4
Qwen1/2/2.5</td><td>2.3</td><td rowspan="3">1.0
(其中2D/3D按合并成整体高效执行)</td><td rowspan="6">0.8</td><td>2.9x</td><td rowspan="3">1.3x</td></tr>
<tr><td>RoPE-2D
dim=64+64</td><td>Qwen-Image
Flux
PG-T2I</td><td>2.6</td><td>3.3x</td></tr>
<tr><td>RoPE-3D
(dim=44+44+40)</td><td>Wan2.1/2.2
hunyuanvideo
PG-T2V/I2V</td><td>2.8</td><td>3.6x</td></tr>
<tr><td rowspan="3">half</td><td>RoPE-1D
dim=128</td><td>PG-72BA3</td><td>2.1</td><td>0.35</td><td>2.6x</td><td>0.4x
(Memory Bound+CV串行)</td></tr>
<tr><td>RoPE-2D
dim=64+64</td><td>-</td><td>2.5</td><td>1.1
含Slice+Concat</td><td>3.1x</td><td>1.4x</td></tr>
<tr><td>RoPE-3D
(dim=44+44+40)</td><td>StepVideo/OpenSORA</td><td>2.9</td><td>5.0
含Slice+Concat</td><td>3.6x</td><td>6.2x</td></tr>
</table>

RoPE-Matrix：昇腾亲和的RoPE算法实现

整网收益：a) 开源网络推理，LLM/多模态理解/多模态生成，分别实现4.7%/4.5%/4.5%加速

Task	Model	Parameters	QKShape[BNSD]	Baseline(ms)	**RoME(ms)**	**Speedup**
LLM	Llama3.1	8B	$[1, 24, 8192, 128]$	50.8	48.1	4.7%
VLM	internvl3	8B	$[1, 28, 1846, 128]$	174	166	4.5%
Edit	FLUX.1-Kontext	12B	$[1, 24, 8704, 128]$	1059	978	8.3%
T2I	Qwen-Image	20B	$[1, 24, 6032, 128]$	1457	1401	4%
T2V	HunyuanVideo	13B	$[1, 24, 42840, 128]/540P$	14410	14400	0.1%
			$[1, 24, 8192, 128]/720P$	40120	40100	0.1%
	Wan2.2	1.3B	$[1, 12, 6630, 128]/272P$	209	199	3.9%
			$[1, 12, 17550, 128]/480P$	790	759	3.4%
		14B	$[1, 40, 6630, 128]/272P$	1170	1120	4.3%
			$[1, 40, 17550, 128]/480P$	4500	4350	3.3%

b) 开源网络训练，多模态生成（wan2.2-1.3B），分别实现4%/2.3%加速

Inputs	RoPE (s)	**RoME (s)**	**Speedup**
$272P$	1.27	1.22	4%
$480P$	3.37	3.30	2.3%

RoPE-Matrix：昇腾亲和的RoPE算法实现

强泛化性：新方案支持任意多维RoPE的不同mode，且相比融合大算子，没有任何约束条件

torch_npu.npu_rotary_mul约束条件

约束说明

- **jit_compile=False场景**（适用Atlas A2 训练系列产品[*]，Atlas A3 训练系列产品[*]）：
  - half模式：

    input：layout支持：$BNSD$、$BSND$、$SBND$；$D < 896$，且为2的倍数；$B, N < 1000$；当需要计算cos/sin的反向梯度时，$B * N <= 1024$。

    r1、r2：数据范围：[-1, 1]；对应input layout的支持情况：
    - x为$BNSD$: $11SD$、$B1SD$、$BNSD$；
    - x为$BSND$: $1S1D$、$BS1D$、$BSND$；
    - x为$SBND$: $S11D$、$SB1D$、$SBND$。

    > **须知：**
    > *half*模式下，当输入*layout*是$BNSD$，且D为非*32Bytes*对齐时，建议不使用该融合算子（模型启动脚本中不开启*--use-fused-rotary-pos-emb*选项），否则可能出现性能下降。

  - interleave模式：

    input：layout支持：$BNSD$、$BSND$、$SBND$; $B * N < 1000$; $D < 896$, 且D为2的倍数;

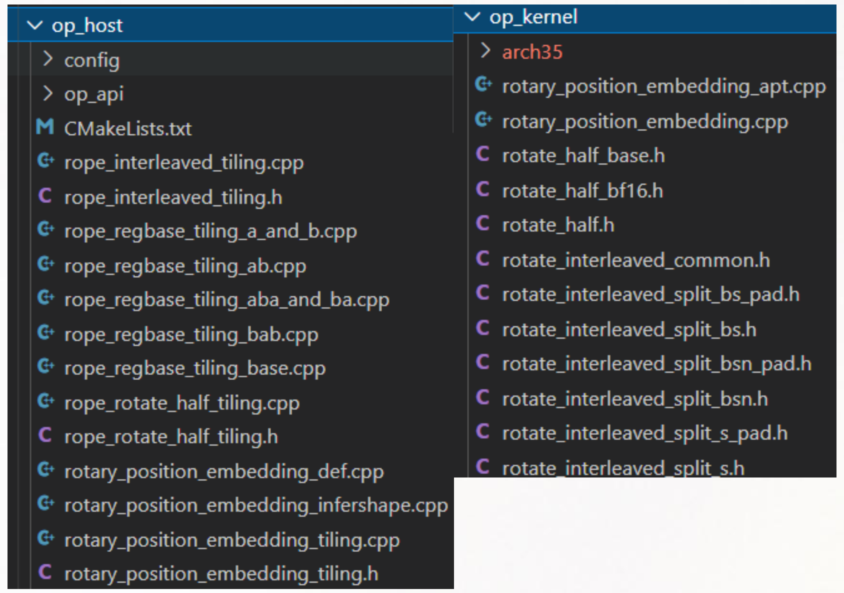
    r1、r2：数据范围：[-1, 1]；对应input layout的支持情况：
    - x为$BNSD : 11SD$;
    - x为$BSND : 1S1D$;
    - x为$SBND : S11D$

- **jit_compile=True场景**（适用Atlas 训练系列产品[*]，Atlas A2 训练系列产品[*]，Atlas 推理系列产品[*]）：

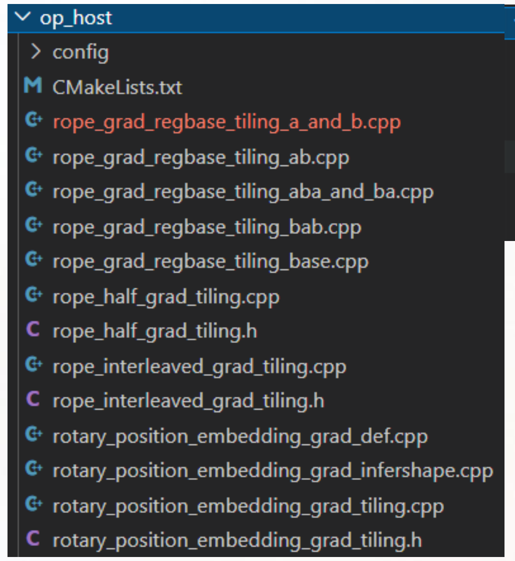
  仅支持rotary_mode为half模式，且r1/r2 layout一般为$11SD$、$1S1D$、$S11D$。

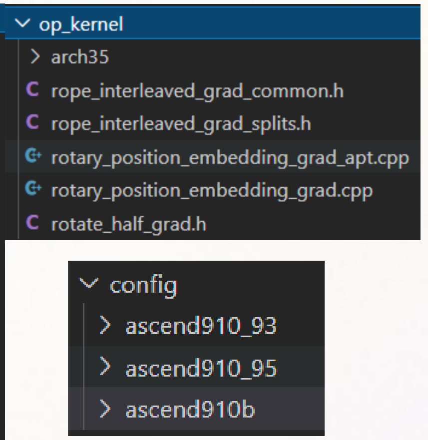
  shape要求输入为4维，其中B维度和N维度数值需小于等于1000，D维度数值为128。

CANN RotaryPositionEmbedding前向实现



CANN RotaryPositionEmbedding反向实现



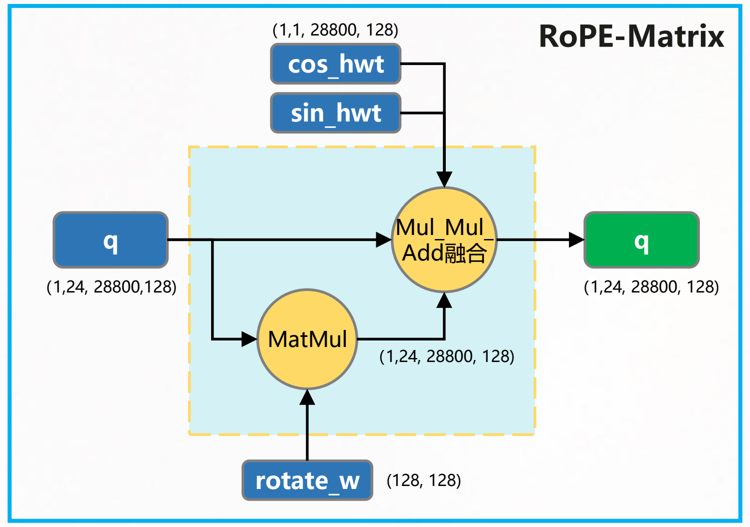


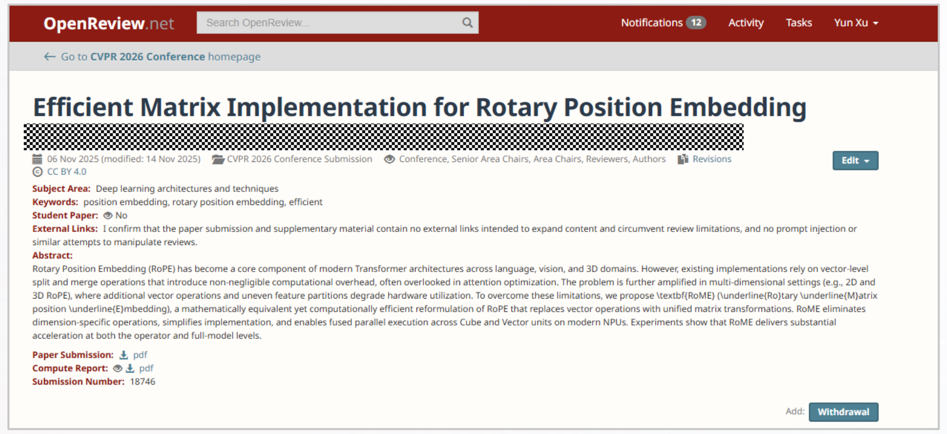
前反向共14k代码量
且不支持310P

RoPE-Matrix：昇腾亲和的RoPE算法实现

- **RoPE-Matrix**
  - 一种昇腾亲和的以Cube计算替代部分Vector计算的**RoPE矩阵乘**实现方式；
  - 算子维度取得CANN小算子**300%**加速、CANN大算子**148%**加速；
  - 整网收益：a) 开源网络推理，LLM/多模态理解/多模态生成，分别实现**4.7%/4.5%/4.5%**加速；b）开源网络训练，多模态生成（wan2.2-1.3B），分别实现4%/2.3%加速
  - 高泛化性，同时覆盖interleave/half/quarter/interleave-half等模式
  - 论文已投CVPR 2026

$$output = x * cos + (x@rotate\_mat) * sin$$





OpenReview.net Search OpenReview... Notifications 12 Activity Tasks Yun Xu

← Go to **CVPR 2026 Conference** homepage

Efficient Matrix Implementation for Rotary Position Embedding

[redacted]

06 Nov 2025 (modified: 14 Nov 2025) CVPR 2026 Conference Submission Conference, Senior Area Chairs, Area Chairs, Reviewers, Authors Revisions Edit

CC BY 4.0

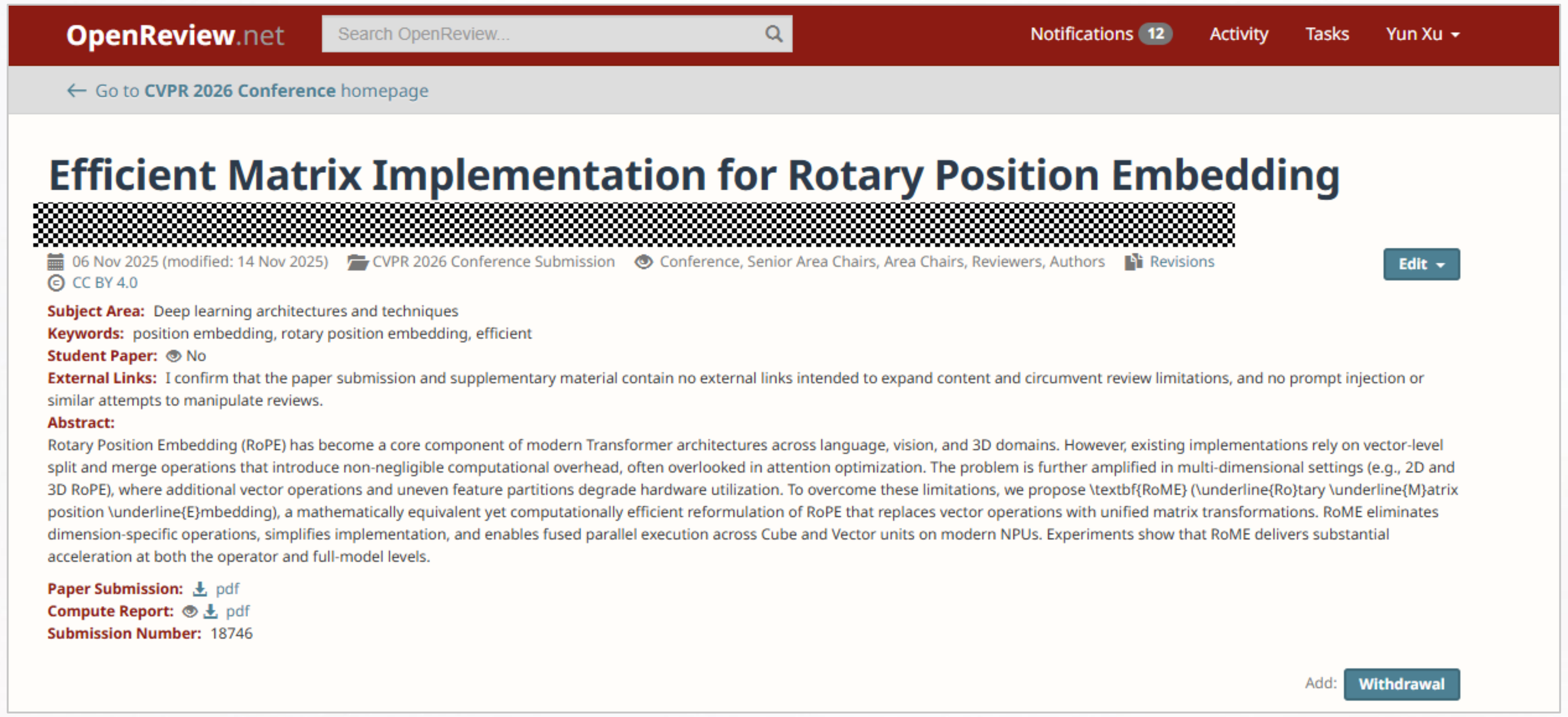
Subject Area: Deep learning architectures and techniques
Keywords: position embedding, rotary position embedding, efficient
Student Paper: No
External Links: I confirm that the paper submission and supplementary material contain no external links intended to expand content and circumvent review limitations, and no prompt injection or similar attempts to manipulate reviews.
Abstract:
Rotary Position Embedding (RoPE) has become a core component of modern Transformer architectures across language, vision, and 3D domains. However, existing implementations rely on vector-level split and merge operations that introduce non-negligible computational overhead, often overlooked in attention optimization. The problem is further amplified in multi-dimensional settings (e.g., 2D and 3D RoPE), where additional vector operations and uneven feature partitions degrade hardware utilization. To overcome these limitations, we propose \textbf{RoME} (\underline{Ro}tary \underline{M}atrix position \underline{E}mbedding), a mathematically equivalent yet computationally efficient reformulation of RoPE that replaces vector operations with unified matrix transformations. RoME eliminates dimension-specific operations, simplifies implementation, and enables fused parallel execution across Cube and Vector units on modern NPUs. Experiments show that RoME delivers substantial acceleration at both the operator and full-model levels.

Paper Submission: pdf
Compute Report: pdf
Submission Number: 18746

Add: Withdrawal

RoPE-Matrix：昇腾亲和的RoPE算法实现

- **RoPE-Matrix**
  - 一种昇腾亲和的以Cube计算替代部分Vector计算的**RoPE矩阵乘**实现方式；
  - 算子维度取得CANN小算子**300%**加速、CANN大算子**148%**加速；
  - 整网收益：a) 开源网络推理，LLM/多模态理解/多模态生成，分别实现**4.7%/4.5%/4.5%**加速；b）开源网络训练，多模态生成（wan2.2-1.3B），分别实现4%/2.3%加速
  - 高泛化性，同时覆盖interleave/half/quarter/interleave-half等模式
  - 论文已投CVPR 2026



OpenReview.net Search OpenReview... Notifications 12 Activity Tasks Yun Xu

← Go to **CVPR 2026 Conference** homepage

Efficient Matrix Implementation for Rotary Position Embedding

[redacted]

06 Nov 2025 (modified: 14 Nov 2025) CVPR 2026 Conference Submission Conference, Senior Area Chairs, Area Chairs, Reviewers, Authors Revisions Edit

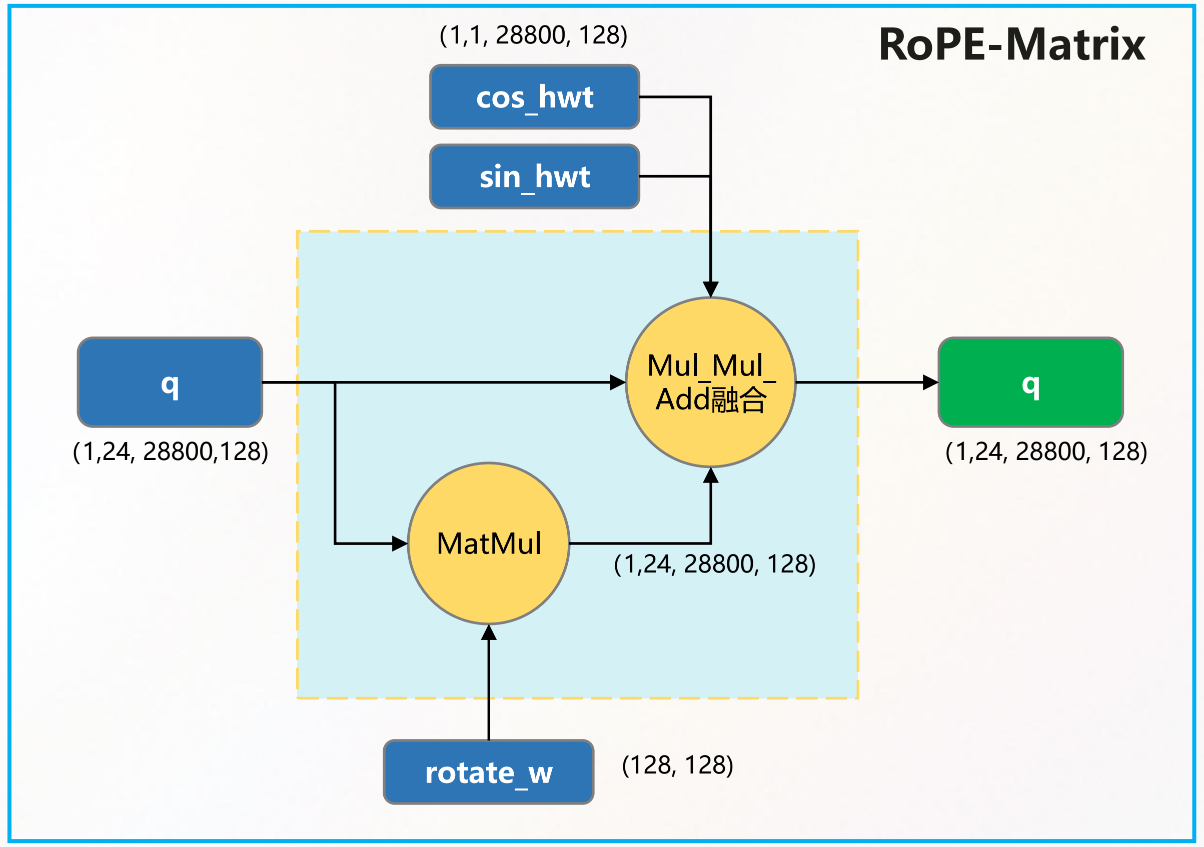
Subject Area: Deep learning architectures and techniques
Keywords: position embedding, rotary position embedding, efficient
Student Paper: No
External Links: I confirm that the paper submission and supplementary material contain no external links intended to expand content and circumvent review limitations, and no prompt injection or similar attempts to manipulate reviews.

Abstract:
Rotary Position Embedding (RoPE) has become a core component of modern Transformer architectures across language, vision, and 3D domains. However, existing implementations rely on vector-level split and merge operations that introduce non-negligible computational overhead, often overlooked in attention optimization. The problem is further amplified in multi-dimensional settings (e.g., 2D and 3D RoPE), where additional vector operations and uneven feature partitions degrade hardware utilization. To overcome these limitations, we propose \textbf{RoME} (\underline{Ro}tary \underline{M}atrix position \underline{E}mbedding), a mathematically equivalent yet computationally efficient reformulation of RoPE that replaces vector operations with unified matrix transformations. RoME eliminates dimension-specific operations, simplifies implementation, and enables fused parallel execution across Cube and Vector units on modern NPUs. Experiments show that RoME delivers substantial acceleration at both the operator and full-model levels.

Paper Submission: pdf
Compute Report: pdf
Submission Number: 18746

Add: Withdrawal

$$output = x * cos + (x@rotate\_mat) * sin$$



RoPE-Matrix：融合算子开发与贡献

02

RoPE-Matrix：融合算子开发与贡献

代码开发：基于Ascend C示例代码及CANN RotaryPositionEmbedding开源仓代码，共3人周完成RoPE-Matrix算子开发，精度性能达标。

日期	事项
10.31	开工
11.04	算子方案设计
11.06	• 完成MIX算子框架搭建及调测链路 • 完成UT代码开发
11.07	Cube计算逻辑代码开发
11.08	• Vector计算逻辑代码开发 • MIX算子计算逻辑衔接
11.10	• 性能优化，达成预期性能 • ops-transformer库提PR

开源代码参考

算子编译工程参考：
https://gitee.com/ascend/samples/tree/master/operator/ascendc/0_introduction/13_matmulleakyrelu_kernellaunch

MIX算子框架及MatMul实现参考：
https://gitee.com/ascend/samples/blob/master/operator/ascendc/0_introduction/22_baremix_kernellaunch/

Vector实现参考CANN内置算子
https://gitcode.com/cann/ops-transformer/tree/master/posembedding/rotary_position_embedding

CV隔离MIX算子框架

```
KERNEL_TASK_TYPE_DEFAULT(KERNEL_TYPE_MIX_AIC_1_2);
if ASCEND_IS_AIC {
    ...
   AscendC::CrossCoreSetFlag<0x2, PIPE_FIX>(0x8); // set flag after matmul
}
if ASCEND_IS_AIV {
    AscendC::CrossCoreWaitFlag(0x8); // wait matmul outputs
    ...
}
```

PYBIND算子调用方式

```
at::Tensor rope_matrix_kernel_bf16(at::Tensor x, at::Tensor y, at::Tensor sin, at::Tensor cos)
{
    ...

    ACLRT_LAUNCH_KERNEL(rope_matrix_kernel_bf16)(
        blockDims, aclstream,
        (uint8_t*)(x.storage().data()),
        (uint8_t*)(y.storage().data()),
        (uint8_t*)(sin.storage().data()),
        (uint8_t*)(cos.storage().data()),
        (uint8_t*)(output.storage().data()),
        (uint8_t*)(workspace_tensor.storage().data()),
        tilingDevice);
}
```

此处预留直播画面LOGO
不放置内容

RoPE-Matrix：融合算子开发与贡献

代码开发：基于Ascend C示例代码及CANN RotaryPositionEmbedding开源仓代码，共3人周完成RoPE-Matrix算子开发，精度性能达标。

- 10.31：开工
- 11.04：算子方案设计
- 11.06：
  - 完成MIX算子框架搭建及调测链路
  - 完成UT代码开发
- 11.07：Cube计算逻辑代码开发
- 11.08：
  - Vector计算逻辑代码开发
  - MIX算子计算逻辑衔接
- 11.10：
  - 性能优化，达成预期性能
  - ops-transformer库提PR

开源代码参考

算子编译工程参考：
https://gitee.com/ascend/samples/tree/master/operator/ascendc/0_introduction/13_matmulleakyrelu_kernellaunch

MIX算子框架及MatMul实现参考：
https://gitee.com/ascend/samples/blob/master/operator/ascendc/0_introduction/22_baremix_kernellaunch/

Vector实现参考CANN内置算子
https://gitcode.com/cann/ops-transformer/tree/master/posembedding/rotary_position_embedding

CV隔离MIX算子框架

```
KERNEL_TASK_TYPE_DEFAULT(KERNEL_TYPE_MIX_AIC_1_2);
if ASCEND_IS_AIC {
   ...
  AscendC::CrossCoreSetFlag<0x2, PIPE_FIX>(0x8); // set flag after matmul
}
if ASCEND_IS_AIV {
   AscendC::CrossCoreWaitFlag(0x8); // wait matmul outputs
   ...
}
```

PYBIND算子调用方式

```
at::Tensor rope_matrix_kernel_bf16(at::Tensor x, at::Tensor y, at::Tensor sin, at::Tensor cos)
{
   ...

   ACLRT_LAUNCH_KERNEL(rope_matrix_kernel_bf16)(
      blockDims, aclstream,
      (uint8_t*)(x.storage().data()),
      (uint8_t*)(y.storage().data()),
      (uint8_t*)(sin.storage().data()),
      (uint8_t*)(cos.storage().data()),
      (uint8_t*)(output.storage().data()),
      (uint8_t*)(workspace_tensor.storage().data()),
      tilingDevice);
}
```

RoPE-Matrix：融合算子开发与贡献

算子开发：已开源至https://gitcode.com/cann/ops-transformer/tree/master/experimental/posembedding/rope_matrix

- posembedding
  - rope_matrix
    - inc
      - rope_matrix_extern.h
    - op_host
      - rope_matrix_tiling.h
    - op_kernel
      - rope_matrix.h
      - rope_matrix_cube.h
    - tests
      - test_rope.py
    - CMakeLists.txt
    - README.md
    - torch_interface.cpp

Tiling关键代码

```
optiling::TCubeTiling tilingData;
auto ascendcPlatform = platform_ascendc::PlatformAscendCManager::GetInstance(socVersion);
matmul_tiling::MultiCoreMatmulTiling tilingApi(*ascendcPlatform);

tilingApi.SetDim(usedCoreNum);
tilingApi.SetAType(leftPosition, leftFormat, leftDtype, isTransA);
tilingApi.SetBType(rightPosition, rightFormat, rightDtype, isTransB);
tilingApi.SetCType(resultPosition, resultFormat, resultDtype);

tilingApi.SetOrgShape(M, N, K); // 完成的MNK大小，单位为元素个数
tilingApi.SetShape(M, N, K); // matmul计算形状的MNK，考虑脏数据
tilingApi.SetSingleShape(calSingleCoreM, baseSize, baseSize);
tilingApi.SetFixSplit(baseM, baseN, -1);
tilingApi.SetBias(isBias);
tilingApi.SetBufferSpace(-1, -1, -1);

int64_t res = tilingApi.GetTiling(tilingData);
```

MatMul计算关键代码

```
int total_N = B * H;
for (int Nindex = 0; Nindex < total_N; ++Nindex) {
    int offsetA = 0, offsetB = 0, offsetC = 0;
    int tailM = 0, tailN = 0;
    bool isTransA = false, isTransB = false;

    CalcGMOffset(AscendC::GetBlockIdx(), offsetA, offsetB, offsetC,
                 tailM, tailN, B, H, M, N, K, Nindex);

    matmulObj.SetTensorA(aGlobal[offsetA], isTransA);
    matmulObj.SetTensorB(bGlobal[offsetB], isTransB);
    matmulObj.SetTail(tailM, tailN);
    matmulObj.IterateAll(cGlobal[offsetC]);
}
matmulObj.End();
```

Vector计算关键代码

```
__aicore__ inline void SingleStepProcess(uint32_t progress, uint64_t copyLength, uint64_t calcLength) {
    uint64_t xOffset;
    uint64_t rOffset;
    uint64_t bnLoopXStartOffset;
    uint64_t progressOffset;
    uint64_t batchOffset;
    rOffset = progress * this->sdBlockSize;
    CopyInR(rOffset, copyLength);
    LocalTensor<T> cosLocal = inQueueCos.DeQue<T>();
    LocalTensor<T> sinLocal = inQueueSin.DeQue<T>();
    LocalTensor<float> cosFp32 = cosBuf.Get<float>();
    LocalTensor<float> sinFp32 = sinBuf.Get<float>();

    Cast(cosFp32, cosLocal, RoundMode::CAST_NONE, calcLength);
    Cast(sinFp32, sinLocal, RoundMode::CAST_NONE, calcLength);
    inQueueCos.FreeTensor<T>(cosLocal);
    inQueueSin.FreeTensor<T>(sinLocal);

    bnLoopXStartOffset = progress * this->sdBlockSize;
    for (uint32_t bnLoop = 0; bnLoop < this->bnSize; bnLoop++) {
        xOffset = bnLoopXStartOffset + bnLoop * this->sdSizeTotal;
        CopyInX(xOffset, copyLength);
        Compute(cosFp32, sinFp32, calcLength);
        CopyOut(xOffset, copyLength);
    }
}
__aicore__ inline void ComputeInner(LocalTensor<float>& x, LocalTensor<float>& xNew,
                                    LocalTensor<float>& cos , LocalTensor<float>& sin,
                                    uint32_t calcLength) {
    Mul(x, x, cos, calcLength);
    Mul(xNew, xNew, sin, calcLength);
    Add(xNew, xNew, x, calcLength);
}
```

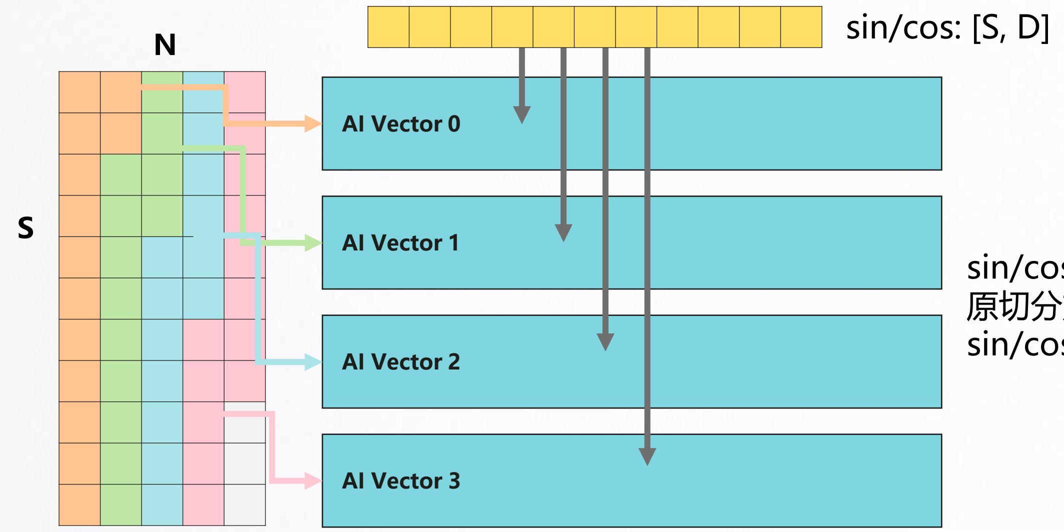
RoPE-Matrix：融合算子开发与贡献

性能优化，单算子耗时从10ms优化至0.8ms达到预期目标，达成memory bound，优化方法：a) 参考RotaryPositionEmbedding开源代码实现调整tiling，降低Vector搬运量；b) MatMul右矩阵大小仅32KB，设置常驻L1，降低Cube搬运量

优化点1：Vector搬运量降低

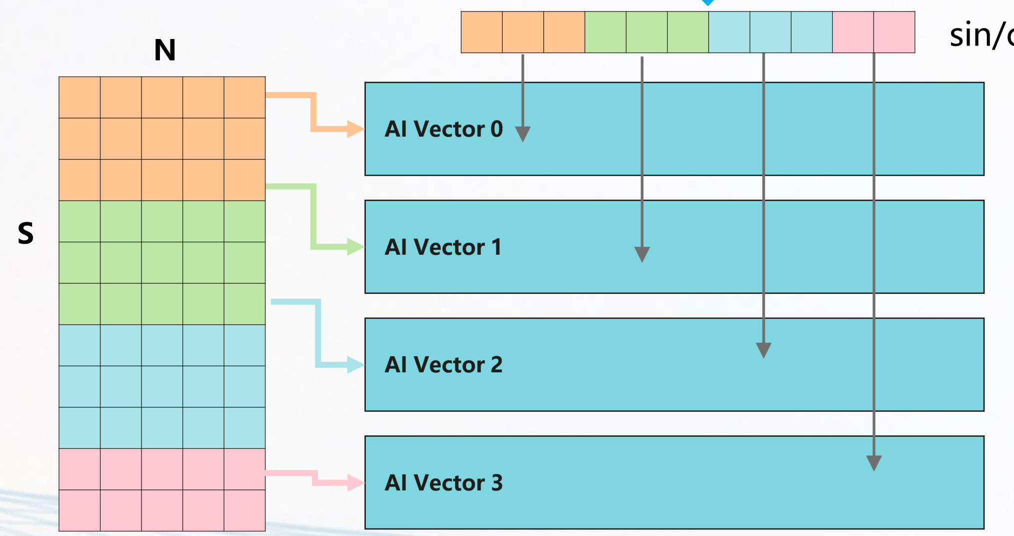
$$output = \boxed{x} * \boxed{cos} + (x@rotate\_mat) * sin$$

$\boxed{x: [B, N, S, D]}$ $\boxed{cos: [1, 1, S, D]}$



sin/cos为broadcast Mul，原切分方案为切BNS，导致sin/cos存在重复拷贝

参考RotaryPositionEmbedding开源代码实现调整tiling，从切BNS轴变为切S轴，消除sin/cos重复拷贝

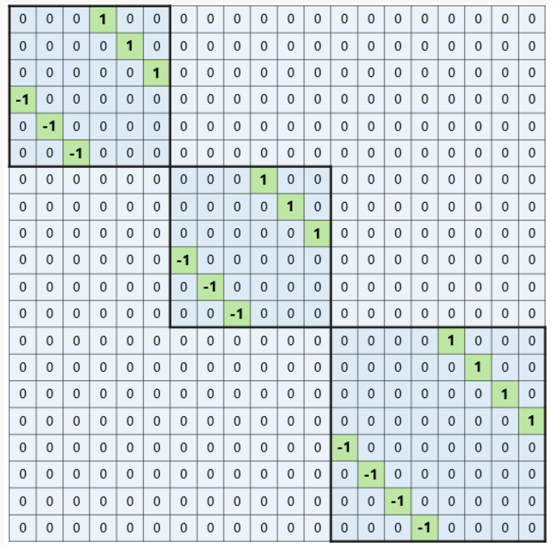


每个AI Core仅需拷贝对应S段的sin/cos

优化点2：Cube搬运量降低

$$output = x * cos + (\boxed{x}@\boxed{rotate\_mat}) * sin$$

$\boxed{x: [B, N, S, D]}$ $\boxed{rotate\_mat: [D, D]}$



MatMul计算右矩阵通常为128x128，且为固定值，可以常驻L1，降低搬运量

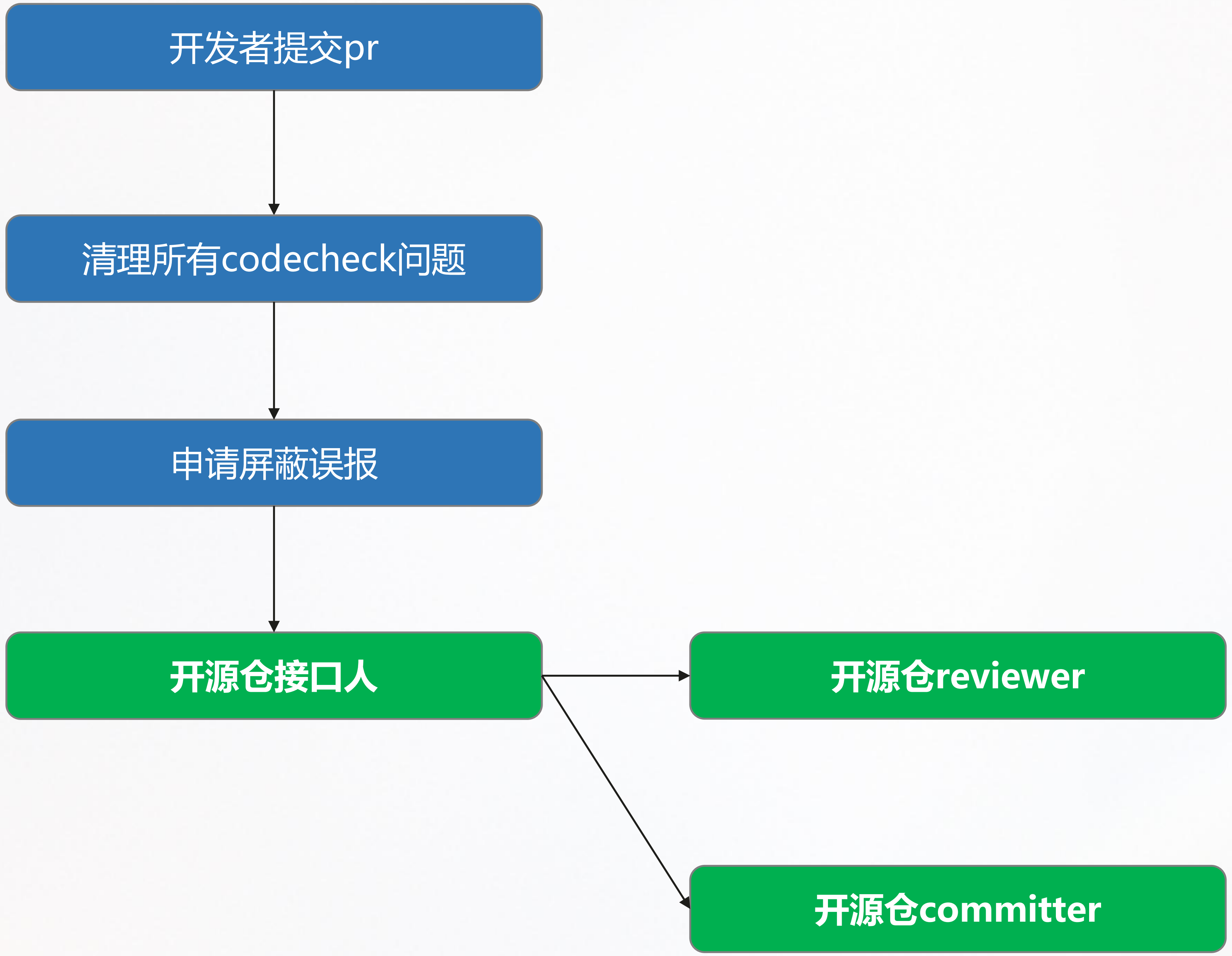
```
+    // Matmul<MatmulType<AscendC::TPosition::GM, CubeFormat::ND, aType>,
+    //        MatmulType<AscendC::TPosition::GM, CubeFormat::ND, bType>,
+    //        MatmulType<AscendC::TPosition::GM, CubeFormat::ND, cType>> matmulObj;
```

右矩阵复用：MatMul高阶API支撑同学建议用法，使能IBShare特性

```
+    static constexpr MatmulConfig MM_CFG = GetIBShareNormConfig();
+    typedef MatmulType<AscendC::TPosition::GM, CubeFormat::ND, A_T> aType;
+    typedef MatmulType<AscendC::TPosition::GM, CubeFormat::ND, B_T, false, LayoutMode::NONE, true> bType
;
+    typedef MatmulType<AscendC::TPosition::GM, CubeFormat::ND, C_T> cType;
+    typedef MatmulType<AscendC::TPosition::GM, CubeFormat::ND, C_T> biasType;
+    Matmul<aType, bType, cType, biasType, MM_CFG> matmulObj;
```

RoPE-Matrix：融合算子开发与贡献

代码合入：核心代码量0.7k，整体合入过程顺利，约1天。



RoPE-Matrix：融合算子开发与贡献

算子源码：https://gitcode.com/cann/ops-transformer/tree/master/experimental/posembedding/rope_matrix

算子编译与安装： https://gitcode.com/cann/ops-transformer/tree/master/experimental/npu_ops_transformer_ext

1. 进入目录，安装依赖

```
cd experimental/npu_ops_transformer_ext
pip install -r requirements.txt
```

2. 从源码构建.whl包

```
python -m build --wheel -n
```

3. 安装构建好的.whl包

```
pip install dist/xxx.whl
```

算子调用：https://gitcode.com/cann/ops-transformer/blob/master/experimental/posembedding/rope_matrix/tests/test_rope.py

```
import torch
import torch_npu
import npu_ops_transformer_ext

B, N, S, D = 1, 24, 28800, 128
dtype = torch.bfloat16
x = torch.randn((B, N, S, D), dtype= dtype).npu()
mat = get_interleave_matrix(dim=D).to(dtype).npu()
sin = torch.randn((1, 1, S, D), dtype=dtype).npu()
cos = torch.randn((1, 1, S, D), dtype=dtype).npu()
out = torch.ops.npu_ops_transformer_ext.rope_matrix(x, mat, sin, cos)
```

RoPE-Matrix：融合算子开发与贡献

RoPE-Matrix首个合入ops-transformer项目库的算子

已合并 **add new operator ropematrix** #299

xuyun15 创建于 23 天前 · 从 xuyun15/ops-transformer：xy_dev 合入到 cann/ops-transformer：master 落后目标分支 50 个提交

讨论 41 提交 1 检查 0 文件改动 9

xuyun15

描述

新增rope matrix算法实现 rope，数学等价，具体参见issue描述

关联的Issue

[Requirement|需求建议]: 新增interleave模式优化rope算子

测试

参考上库代码中test脚本，测试性能和精度

共提出10个issue，其中9个已被accept，6个已关闭

Issue编号	内容	是否accept	当前状态
61	[Requirement|需求建议]: 新增interleave模式优化rope算子	Y	close
119	[Documentation|文档反馈]: Matmul高阶API性能调优文档缺失	Y	close
162	[Requirement|需求建议]: 算子贡献默认规则未描述清晰	Y	close
184	[Documentation|文档反馈]: experimental路径下使用RunOpApiV2完成算子适配调用缺乏详细信息	Y	close
187	[Documentation|文档反馈]: experimental路径下编译依赖说明不全	Y	close
195	[Documentation|文档反馈]: 官网matmul高阶api的约束没有说明清楚	N	open
198	[Documentation|文档反馈]: experimental路径下一些编程规范没有说清楚	Y	open
203	[Documentation|文档反馈]: 特定场景下matmul引发精度问题，文档未提及	Y	open
206	[Documentation|文档反馈]: 高阶api对ASCENDC_CUBE_ONLY描述的不够清晰，与用例使用不一致	Y	close
230	[Requirement|需求建议]: experimental路径下部分编译能力提供	Y	open

算子开发流程及经验总结文档发布于社区

Ascend 开发者 主页 开发 文档 活动 学习 论坛 博客 开发者计划

博客 > 【开源贡献案例】端到端打通transformer仓experimental路径首个开源mix算子

【开源贡献案例】端到端打通transformer仓experimental路径首个开源mix算子

新人帖 CANN PyTorch Samples 算子 Ascend C

xuyun15 发表于11/24 452 15 13

一、背景

笔者主要业务为AI加速，在对业务相关的transformer网络做profiling分析时发现，当前常见的AIGC、多模态、VLM、LLM中都存在RoPE算法，且其中ROPE算子执行占比高，如下图所示，我们可以看到flux网络中，除了FA和matmul，第三就是rope，它由一些基础的vector操作组成，耗时远高于预期，占比整网10%。

ops-transformers仓discussion经验沉淀

已开启 **使用<<<>>>方式实现算子FAQ** #3

fanzijian · 创建于 8 天前

fanzijian 8 天前 创建讨论

使用<<<>>>实现新算子流程参考文档：https://gitcode.com/cann/ops-transformer/blob/master/experimental/npu_ops_transformer_ext/README.md

一、环境要求

1. gcc版本依赖>9，并需确认ccec -v 链接到的是对应的gcc版本；
2. torch_npu建议从链接下载安装，以免部分新特性不支持，建议下载2.1.0版本。

二、编译报错：缺少char_t类型

答：使用matmul高阶api tiling头文件之前需要“typedef uint8_t char_t”，否则编译会报错，缺少char_t类型。

三、编译报错：不用二级namesapce直接以::xxx::xxx引用

答：所有文件包括experimental工程所需的npu_ops_transformer_ext以及用户自己子工程的namespace，都需要包含二级namespace。

四、编译报错：存在重名文件

答：“using namespace matmul_tiling “、“ using namespace matmul “、“ using namespace AscendC “ 不能同时使用，它们内部有重名文件，会导致工程编译报错。

五、python -m build --wheel -n编译报错：找不到bisheng路径

答：source环境变量之后ascendhome路径是对的，但bisheng的路径不对。解决方案：通过python setup.py clean清理一下编译缓存再编译就可以了。

Thanks！

访问CANN开源社区

关注昇腾CANN公众号