Deepseek-v3 Op List

vLLM

共计是99个算子，需要算子实现的是54个

需要实现的**和llama2重复的有39个标为红色**

独有15个，标为蓝色

加粗字体的是和llama2 相同的算子（包含需要实现和不需要实现）

Operation Name	op 文档	Need Impl	Init	Prefill
_C::fused_add_rms_norm	**OP的开发文档**	X	X	
_C::rms_norm	**OP 的开发文档**	X	X	
_C::silu_and_mul	**OP 的开发文档**	X	X	
_C_cache_ops::concat_and_cache_mla	**OP 的开发文档**	X		
_moe_C::moe_align_block_size	OP 的开发文档	X	X	X
_moe_C::moe_sum	OP 的开发文档	X	X	X
_vllm_fa2_C::varlen_fwd	OP 的开发文档	X	X	X
aten::_log_softmax	**OP的开发文档**	X	X	
aten::_reshape_alias	OP 的开发文档	X	X	X
aten::_softmax	**OP的开发文档**	X	X	
aten::_to_copy	**CompositeExplicitAutograd**		X	X
aten::_unsafe_view	CompositeExplicitAutograd		X	X
aten::add.Tensor	DONE	X	X	X
aten::alias	**CompositeExplicitAutograd**		X	
aten::arange	**OP的开发文档**	X	X	
aten::arange.start_out	**OP的开发文档**	X	X	
aten::arange.start_step	**OP 的开发文档**	X	X	
aten::argmax	**OP的开发文档**	X	X	X
aten::as_strided	**OP的开发文档**	X	X	X
aten::bitwise_not	OP 的开发文档	X		X
aten::bmm	OP 的开发文档	X	X	
aten::cat	**OP的开发文档**	X	X	X
aten::chunk	CompositeImplicitAutograd		X	X
aten::clamp	OP 的开发文档	X	X	
aten::clone	CompositeExplicitAutograd			
aten::constant_pad_nd	CompositeExplicitAutograd		X	X
aten::contiguous	CompositeImplicitAutograd		X	X
aten::copy_	**CompositeExplicitAutograd**		X	X
aten::cos	**OP的开发文档**	X	X	
aten::cumsum	**OP的开发文档**	X	X	
aten::detach	**CompositeExplicitAutograd**		X	
aten::detach_	**CompositeExplicitAutograd**		X	X
aten::div.Tensor	**OP的开发文档**	X	X	
aten::div_.Tensor	**OP 的开发文档**		X	X
aten::embedding	**CompositeExplicitAutograd**		X	X
aten::empty.memory_format	**DONE**	X	X	

aten::empty_like	**CompositeExplicitAutograd**		X	X
aten::empty_strided	**DONE**	X	X	X
aten::expand	CompositeExplicitAutograd			
aten::exponential_	**OP的开发文档**	X	X	
aten::fill_.Scalar	**OP的开发文档**	X	X	X
aten::fill_.Tensor	**OP的开发文档**	X	X	
aten::flatten.using_ints	**CompositeImplicitAutograd**			
aten::full	CompositeExplicitAutograd		X	
aten::gather	**OP的开发文档**	X	X	
aten::index.Tensor	**OP的开发文档**	X	X	
aten::index_select	**OP的开发文档**	X	X	
aten::le.Tensor	**OP 的开发文档**	X	X	
aten::lift_fresh	**CompositeExplicitAutograd**		X	X
aten::linear	**CompositeImplicitAutograd**		X	
aten::log_softmax.int	**CompositeImplicitAutograd**		X	
aten::lt.Tensor	**OP 的开发文档**	X	X	
aten::masked_fill.Scalar	CompositeExplicitAutograd			
aten::masked_fill_.Scalar	**OP的开发文档**	X	X	X
aten::matmul	**CompositeImplicitAutograd**		X	X
aten::max.dim	OP 的开发文档	X		X
aten::mm	**OP的开发文档**	X	X	
aten::mul.Tensor	**OP的开发文档**	X	X	X
aten::narrow	CompositeImplicitAutograd		X	X
aten::neg	DONE	X	X	X
aten::ones	**CompositeExplicitAutograd**		X	
aten::pad	CompositeImplicitAutograd		X	X
aten::permute	**CompositeExplicitAutograd**		X	
aten::pow.Scalar	**OP 的开发文档**	X	X	
aten::pow.Tensor_Tensor_out	**OP 的开发文档**	X	X	
aten::reciprocal	**OP 的开发文档**	X	X	
aten::repeat_interleave.self_int	OP 的开发文档	X	X	X
aten::reshape	**CompositeImplicitAutograd**		X	X
aten::resize_	**OP 的开发文档**	X	X	X
aten::result_type.Scalar	**CompositeImplicitAutograd**		X	
aten::result_type.Scalar_Tensor	**CompositeImplicitAutograd**		X	
aten::rsub.Scalar	**OP 的开发文档**	X	X	
aten::scalar_tensor	**CompositeExplicitAutograd**		X	

aten::scatter_.src	**OP 的开发文档**		X	
aten::scatter_.value	scatter_.value			
aten::select.int	**CompositeExplicitAutograd**		X	X
aten::sin	**OP的开发文档**	X	X	
aten::slice.Tensor	**CompositeExplicitAutograd**		X	X
aten::softmax.int	**CompositeImplicitAutograd**		X	X
aten::sort	**OP 的开发文档**	X	X	
aten::sort.stable	**OP 的开发文档**	X	X	
aten::split.Tensor	CompositeExplicitAutograd		X	X
aten::split_with_sizes	**CompositeExplicitAutograd**		X	
aten::squeeze.dim	CompositeExplicitAutograd			X
aten::stack	CompositeExplicitAutograd		X	X
aten::sub.Tensor	**OP的开发文档**	X	X	X
aten::sum.IntList_out	OP 的开发文档	X	X	X
aten::t	**CompositeExplicitAutograd**		X	
aten::to.dtype	**CompositeImplicitAutograd**		X	
aten::to.dtype_layout	**CompositeImplicitAutograd**		X	
aten::topk	OP 的开发文档	X		
aten::transpose.int	**CompositeExplicitAutograd**		X	
aten::unsqueeze	**CompositeExplicitAutograd**		X	
aten::view	**OP 的开发文档**	X	X	X
aten::zero_	**OP 的开发文档**	X	X	X
aten::zeros	**CompositeExplicitAutograd**		X	X
aten::zeros_like	CompositeExplicitAutograd			
vllm::inplace_fused_experts	OP 的开发文档	X	X	X
vllm::unified_attention	OP 的开发文档	X	X	

HF transformers

attn_implementation = "eager" **共计是87个算子，需要算子实现的是55个（Need Impl)**

与llama2 op list 相比，需要算子实现的重复op有39个标为红色

Deepseek-v3 独有有16个 标为蓝色

attn_implementation="flash_attention_2" 会有一个额外的需要impl算子
flash_attn::_flash_attn_forward

Operation Name	op 文档	Need Impl	Init	Prefil	Prefill F	Prefil
aten::_index_put_impl_	OP的开发文档	X			X	
aten::_local_scalar_dense	OP的开发文档	X			X	
aten::_softmax	OP的开发文档	X			X	
aten::add.Tensor	OP的开发文档	X			X	X
aten::any	OP的开发文档	X			X	
aten::any.dim	OP的开发文档	X			X	X
aten::arange	OP的开发文档	X			X	X
aten::arange.start_out	OP的开发文档	X			X	X
aten::arange.start_step	OP的开发文档	X			X	
aten::argmax	OP的开发文档	X				X
aten::as_strided	OP的开发文档	X			X	X
aten::bitwise_and.Tensor	OP的开发文档	X				X
aten::bitwise_not	OP的开发文档	X			X	X
aten::bitwise_or.Tensor	OP的开发文档	X				X
aten::bmm	OP的开发文档	X			X	
aten::cat	OP的开发文档	X			X	X
aten::cos	OP的开发文档	X			X	
aten::cumsum	OP的开发文档	X			X	
aten::div.Tensor	OP 的开发文档	X			X	
aten::empty.memory_forma	DONE	X			X	X
aten::empty_strided	DONE	X	X	X	X	X
aten::eq.Scalar	OP的开发文档	X			X	X
aten::eq.Tensor	OP的开发文档	X			X	X
aten::fill_.Scalar	OP的开发文档	X			X	X
aten::floor_divide	OP的开发文档	X			X	
aten::gather	OP的开发文档	X			X	
aten::index.Tensor	OP的开发文档	X			X	
aten::index_select	OP的开发文档	X			X	
aten::isin.Tensor_Tensor	OP的开发文档	X			X	X
aten::lt.Scalar	OP的开发文档	X			X	
aten::lt.Tensor	OP的开发文档	X			X	
aten::masked_fill_.Scalar	OP的开发文档	X			X	
aten::max	OP的开发文档	X				X
aten::mean.dim	OP的开发文档	X			X	
aten::mm	OP的开发文档	X			X	
aten::mul.Tensor	OP的开发文档	X			X	X

aten::neg	OP的开发文档	X			X	
aten::pow.Scalar	OP的开发文档	X			X	
aten::pow.Tensor_Scalar	OP的开发文档	X			X	
aten::pow.Tensor_Tensor_ou	OP的开发文档	X			X	
aten::reciprocal	OP的开发文档	X			X	
aten::resize_	OP的开发文档	X			X	X
aten::rsqrt	OP的开发文档	X			X	
aten::rsub.Scalar	OP的开发文档	X			X	X
aten::sigmoid	OP 的开发文档	X			X	
aten::silu	OP的开发文档	X			X	
aten::sin	OP的开发文档	X			X	
aten::sort	OP的开发文档	X			X	
aten::sort.stable	OP的开发文档	X			X	
aten::sub.Tensor	OP的开发文档	X			X	X
aten::sum.dim_IntList	OP的开发文档	X			X	
aten::topk	OP的开发文档	X			X	
aten::view	DONE	X			X	X
aten::where.self	OP的开发文档	X				X
aten::zero_	OP的开发文档	X			X	

llama2 op独有的为三个

Operation Nam	OP 文档	Need Impl	Init	Prefill Prepare	Prefill Forward	Prefill Postpr
aten::ge.Scalar	OP 的开发文	X			X	
aten::gt.Tensor	OP 的开发文	X			X	
aten::triu	OP 的开发文	X			X	

FP8 支持需求

通过DeepSeek开源的weights（https://huggingface.co/deepseek-ai/DeepSeek-V3-Base/tree/main）+ 源码（https://github.com/deepseek-ai/DeepSeek-V3/blob/main/inference/kernel.py https://github.com/deepseek-ai/DeepSeek-V3/blob/main/inference/model.py）分析

Embedding：

- model.embed_tokens.weight → 使用 BF16，不需要 FP8

Multi-Headed Attention Layer (MLA)：

- self_attn.kv_a_proj_with_mqa.weight → FP8 (Linear ->fp8_gemm)
- self_attn.kv_b_proj.weight → FP8（ColumnParallelLinear->Linear ->fp8_gemm)
- self_attn.o_proj.weight → FP8（RowParallelLinear ->Linear ->fp8_gemm)
- self_attn.q_a_proj.weight → FP8 （ColumnParallelLinear->Linear ->fp8_gemm)
- self_attn.q_b_proj.weight → FP8 （ColumnParallelLinear->Linear ->fp8_gemm)

FFN：

- **非 MoE 和 MoE 层：**
  - mlp.down_proj.weight → FP8 (ColumnParallelLinear /Linear -> linear->fp8_gemm)
  - mlp.gate_proj.weight → FP8 (Linear -> linear->fp8_gemm)
  - mlp.up_proj.weight → FP8 (RowParallelLinear / ColumnParallelLinear -> linear->fp8_gemm)

涉及到使用FP8数据类型计算的kernel只有**fp8_gemm_kernel**() https://github.com/deepseek-ai/DeepSeek-V3/blob/a878eada08ea6913f5a2ae80a43afeffdef082ef/inference/kernel.py#L115C5-L115C20

在**fp8_gemm_kernel中关键kernel为tl.dot()，参考文档：**https://triton-lang.org/main/python-api/generated/triton.language.dot.html

Deepseek开源的inference代码中的ColumnParallelLinear/RowParallelLinear/Linear 在调用fp8_gemm_kernel时，对应在pytorch框架中等价为**matmul**(gemm)**, mv**(gemv)**, bmm**三个算子

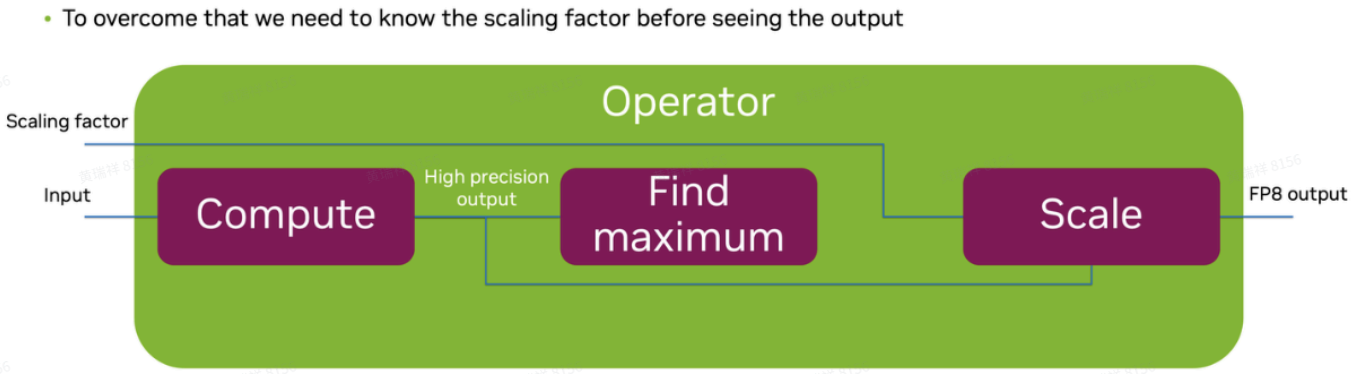
此外还需要实现**act_quant**() weight_dequant()量化和反量化操作https://github.com/deepseek-ai/DeepSeek-V3/blob/a878eada08ea6913f5a2ae80a43afeffdef082ef/inference/kernel.py

TE中的实现进行了算子融合，会涉及到更多的FP8情况

1. FP8 量化与反量化算子

https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html#Mixed-precision-training-with-FP8

- **Delayed scaling**，即假设我们提前知道了 scale 值，那么计算过程就可以在一个 kernel 上完成，而 amax 的计算和 scale 值的更新与其独立，不会中断计算进程，因此这种方式能完全发挥 FP8 的性能。但“提前知道的” scale 值需要额外的空间来记录，同时会引入一定的误差。

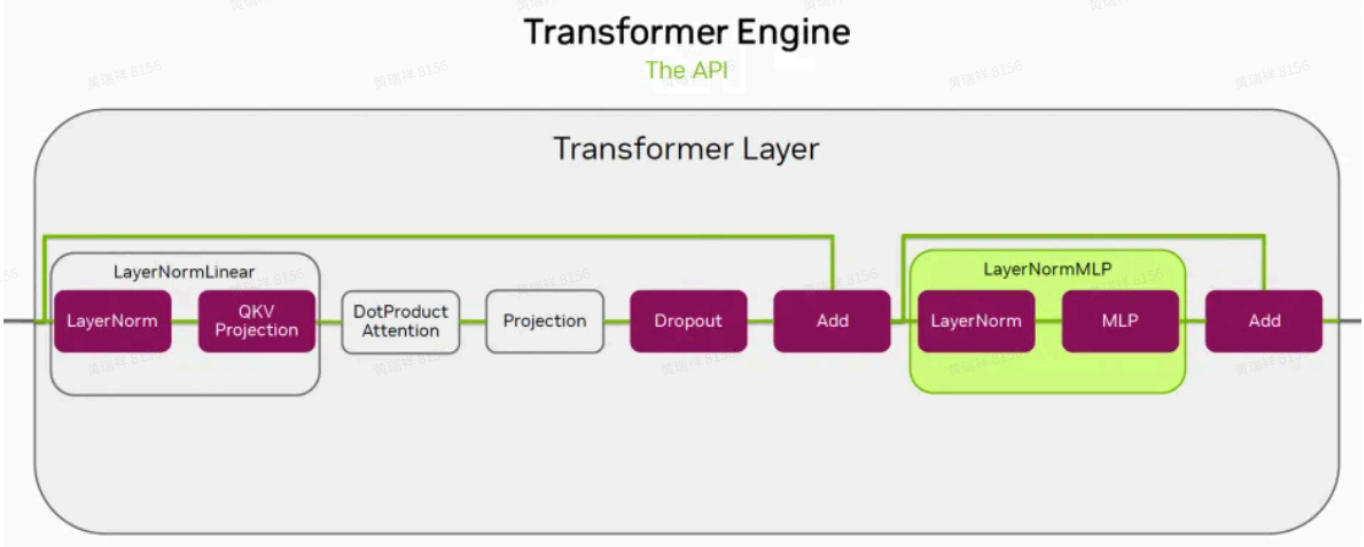


对每个 GEMM 算子用到的 tensor 记录一个 amax history 数组，当我们需要 scale 值时，就从这个数组中取出最近一段时间窗口内 amax 的最大值，以此近似现在这个 tensor 的 amax，并默认用以下方式计算 scale 值：

代码块

```
FP8_MAX = maximum_representable_value(fp8_format)
new_scaling_factor = (FP8_MAX / amax) / (2 ^ margin)
```

2. FP8 Cast 与转置算子/padding结合
   https://github.com/NVIDIA/TransformerEngine/blob/db2aaa9e7765a3bcd1e652e3539d57e85cdbd5bc/transformer_engine/common/include/transformer_engine/transpose.h#L79
   https://github.com/NVIDIA/TransformerEngine/blob/db2aaa9e7765a3bcd1e652e3539d57e85cdbd5bc/transformer_engine/pytorch/module/fp8_padding.py#L69
3. GEMM FP8（矩阵乘法）算子
4. GEMM+activation+quantize fusion
   https://github.com/NVIDIA/TransformerEngine/blob/db2aaa9e7765a3bcd1e652e3539d57e85cdbd5bc/transformer_engine/pytorch/module/layernorm_mlp.py#L715
5. FP8 融合 Bias-Dropout-Add 算子，将FP8 casting操作和Bias-Dropout-Add结合



Attention 推理

precompute_freqs_cis + apply_rotary_emb part（ROPE先放在cpu上运行）

1. arange() -> **aten::arange.start_out** (MID)
2. ones() -> **fill_.Scalar** ((MID))
3. **clamp.out** (MID)
4. **data type cast** (MID)
    a. Data type cast for direct_copy_kernel_sipu is not supported now

```
代码块
1   默认input 为bf16，但是view_as_complex/view_as_real计算要求input为float32，如果不添加sipu的type cast，需要转到cpu上做
```

5. **copy size restrict (BUG) @ yunyi WIP**

```
代码块
1   copy size > 8k 会coredump，ds中会遇到[32, 16384] case
2   t = fake_arange(0, seqlen, 1, dtype=torch.bfloat16)
3   ones = fake_ones(freqs.shape, dtype=torch.bfloat16)
```

6. **polar.out** 或者 cos.out + sin.out (MID)

```
代码块
```

```
ROPE中位置计算需要
# torch.polar 的文档
# https://pytorch.org/docs/stable/generated/torch.polar.html
# 计算结果是个复数向量
# 假设 freqs = [x, y]
# 则 freqs_cis = [cos(x) + sin(x)i, cos(y) + sin(y)i]

freqs_cis = torch.polar(torch.ones_like(freqs), freqs)
```

7. ~~torch.view_as_complex（dont need kernel，done）~~
8. ~~torch.view_as_real（donot need kernel，done）~~

RMSNorm (先放在cpu上运行，MID) 目前已有rmsnorm的整体实现，composition的算子优先级降低

1. aten::pow.Tensor_Scalar_out
2. aten::result_type.Scalar
3. aten::to.dtype
4. aten::mean.out
5. aten::sum.IntList_out
6. aten::fill_.Scalar
7. div_.Scalar + div_.Tensor
8. aten::rsqrt.out

Linear

1. ~~***act_quant（must）done***~~
2. ~~***fp8_gemm（must）done***~~
3. ~~If world size > 1 还会涉及到 dist.all_reduce(tensor, op=ReduceOp.SUM)~~

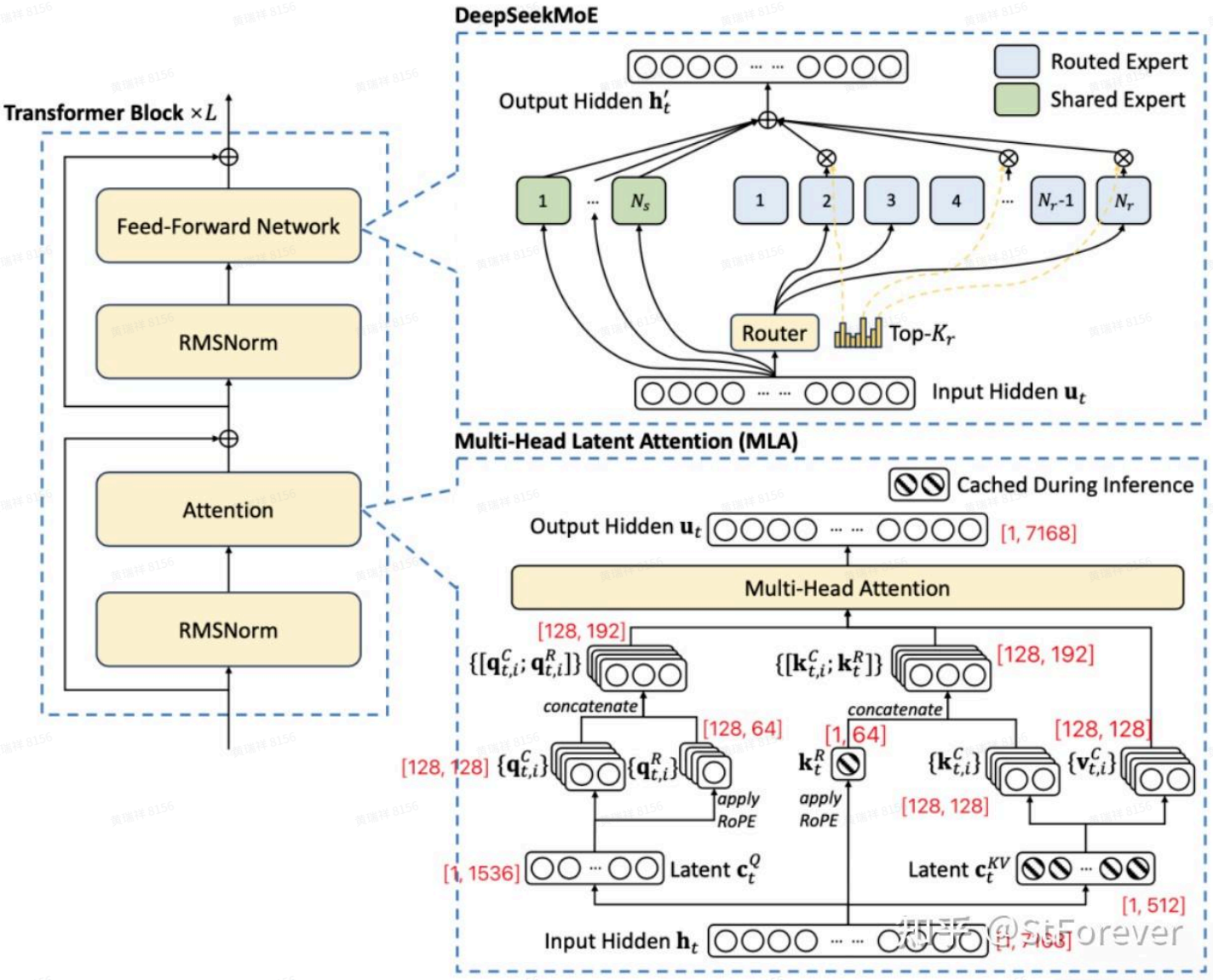
others

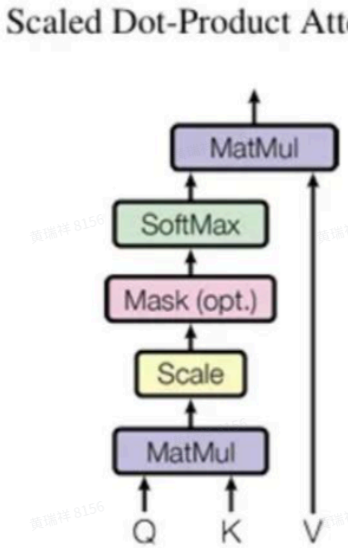
1. aten::zero_ (mid)

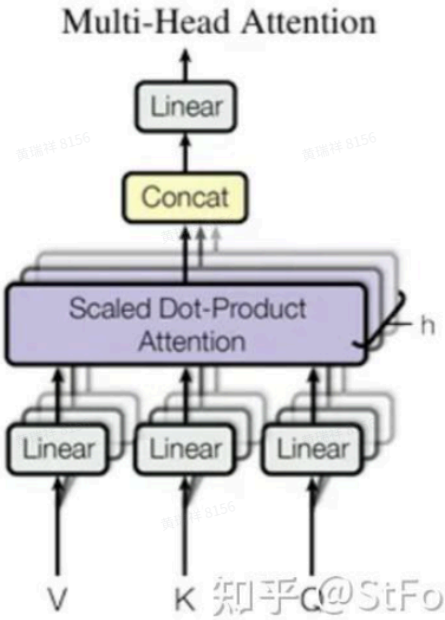
   用来初始化kv cache，可以使用aten::empty(sipu已经实现)代替
2. ~~**aten::bmm.out (must)done**~~

   ~~torch.einsum()会调用~~
3. slice操作

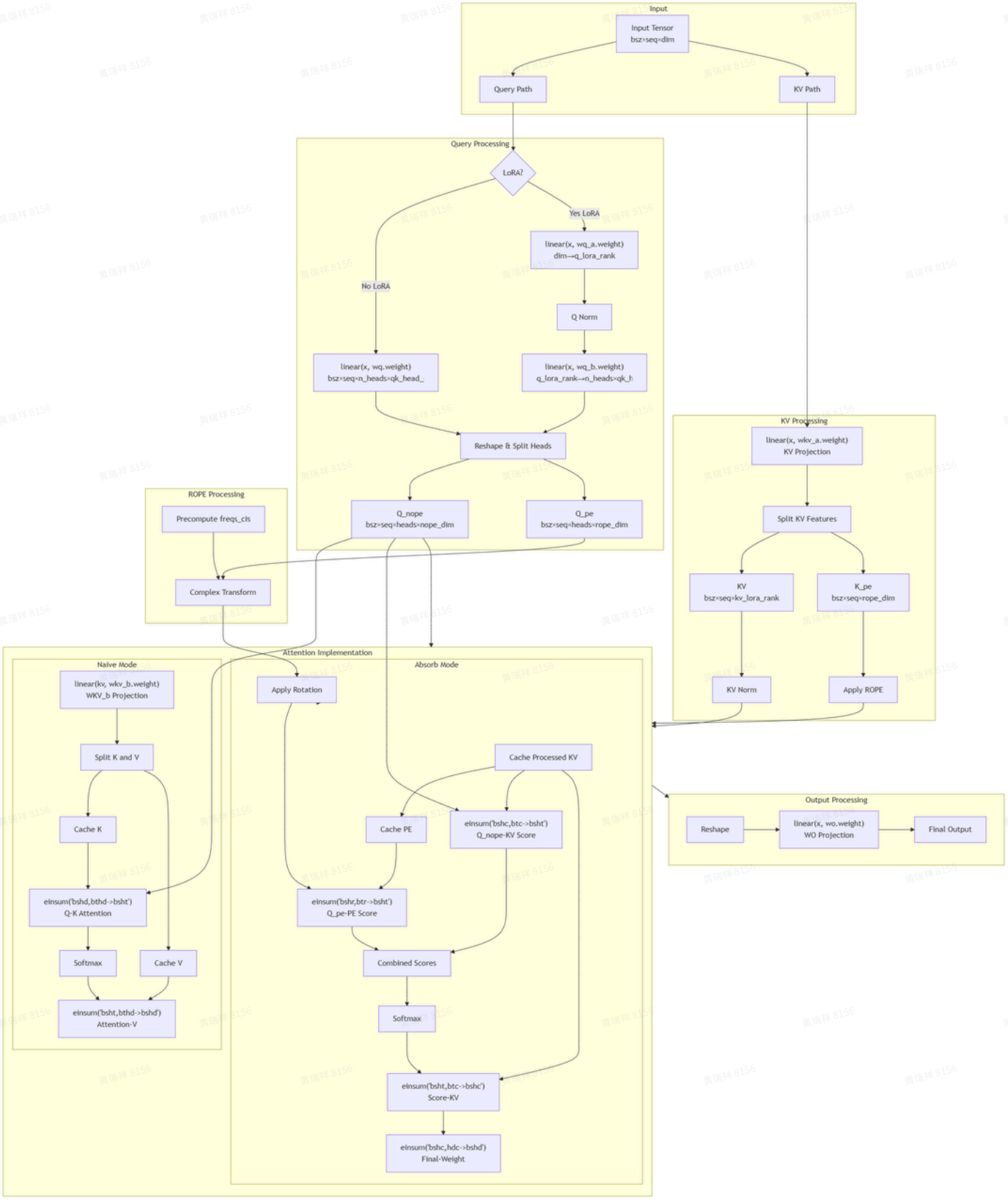
   bug，本身已经实现，但是会有core dump


DeepSeekMoE
Routed Expert
Shared Expert
Output Hidden $\mathbf{h}'_t$
Transformer Block ×L
Feed-Forward Network
RMSNorm
Attention
RMSNorm
Router
Top-K_r
Input Hidden $\mathbf{u}_t$
Multi-Head Latent Attention (MLA)
Cached During Inference
Output Hidden $\mathbf{u}_t$
[1, 7168]
Multi-Head Attention
[128, 192]
[128, 192]
concatenate
concatenate
[128, 64]
[1, 64]
[128, 128]
[128, 128]
apply RoPE
apply RoPE
[128, 128]
[1, 1536]
Latent $\mathbf{c}_t^Q$
Latent $\mathbf{c}_t^{KV}$
[1, 512]
Input Hidden $\mathbf{h}_t$
[1, 7168]


Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V


Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

实现流程：



放在SIPU的模块为 attention implementation（包含bmm，softmax），linear(和RMS_norm和ROPE穿插，调小了模型，目前可以在SIPU侧运行~~因为copy size restrict的问题，目前无法放在SIPU侧，单测bf16是ok的,~~) , RMS_norm(SIPU未实现), ROPE(SIPU未实现)目前只能先放在CPU侧处理

Embedding 推理

F.embedding() trace

```
 [call] op=[aten::embedding], key=[PythonTLSSnapshot]
  [redispatchBoxed] op=[aten::embedding], key=[AutogradPrivateUse1]
   [redispatch] op=[aten::embedding], key=[Python]
    [callBoxed] op=[aten::embedding], key=[Python]
     [callBoxed] op=[aten::embedding], key=[PrivateUse1]
      [call] op=[aten::reshape], key=[PrivateUse1]
       [call] op=[aten::view], key=[PrivateUse1]
      [call] op=[aten::index_select], key=[PrivateUse1]
       [call] op=[aten::_to_cpu], key=[PrivateUse1]
        [call] op=[aten::to.dtype_layout], key=[BackendSelect]
         [redispatch] op=[aten::to.dtype_layout], key=[PrivateUse1]
          [call] op=[aten::_to_copy], key=[BackendSelect]
           [redispatch] op=[aten::_to_copy], key=[PrivateUse1]
            [call] op=[aten::empty_strided], key=[BackendSelect]
             [redispatch] op=[aten::empty_strided], key=[CPU](sipu_torch已接入)
            [call] op=[aten::copy_], key=[PrivateUse1]
             [call] op=[aten::_copy_from], key=[PrivateUse1]
        [call] op=[aten::to.dtype_layout], key=[BackendSelect]
         [redispatch] op=[aten::to.dtype_layout], key=[PrivateUse1]
          [call] op=[aten::_to_copy], key=[BackendSelect]
           [redispatch] op=[aten::_to_copy], key=[PrivateUse1]
            [call] op=[aten::empty_strided], key=[BackendSelect]
             [redispatch] op=[aten::empty_strided], key=[CPU](sipu_torch已接入)
            [call] op=[aten::copy_], key=[PrivateUse1]
             [call] op=[aten::_copy_from], key=[PrivateUse1]
       [redispatchBoxed] op=[aten::index_select], key=[CPU] (未实现)
        [call] op=[aten::empty.memory_format], key=[BackendSelect]
         [redispatch] op=[aten::empty.memory_format], key=[CPU](sipu_torch已接入)
        [call] op=[aten::select.int], key=[CPU] (CompositeExplicitAutograd，依赖为as_strided，sipu_torch已接入)
         [call] op=[aten::as_strided], key=[CPU](sipu_torch已接入)
        [call] op=[aten::select.int], key=[CPU] (CompositeExplicitAutograd，依赖为as_strided，sipu_torch已接入)
         [call] op=[aten::as_strided], key=[CPU](sipu_torch已接入)
       [call] op=[aten::to.dtype_layout], key=[BackendSelect]
        [redispatch] op=[aten::to.dtype_layout], key=[PrivateUse1]
         [call] op=[aten::_to_copy], key=[BackendSelect]
          [redispatch] op=[aten::_to_copy], key=[PrivateUse1]
           [call] op=[aten::empty_strided], key=[BackendSelect]
            [redispatch] op=[aten::empty_strided], key=[PrivateUse1]
[14:01:10.016][SIRT][W][2260291] [device.cpp:120] Legacy compatibility mode: Converting SIPU to 1U16C
           [call] op=[aten::copy_], key=[PrivateUse1]
            [call] op=[aten::_copy_from], key=[PrivateUse1]
```

```
42        [call] op=[aten::view], key=[PrivateUse1]
```

1. ~~**aten::select.int**~~
   a. CompositeExplicitAutograd，依赖算子为as_strided，已经实现，经过测试可以完全在SIPU上运行

select trace

```
1  start select
2   [call] op=[aten::select.int], key=[PythonTLSSnapshot]
3    [redispatchBoxed] op=[aten::select.int], key=[AutogradPrivateUse1]
4     [redispatch] op=[aten::select.int], key=[ADInplaceOrView]
5      [redispatch] op=[aten::select.int], key=[Python]
6       [callBoxed] op=[aten::select.int], key=[Python]
7        [callBoxed] op=[aten::select.int], key=[PrivateUse1]
8         [call] op=[aten::as_strided], key=[PrivateUse1]
```

2. ~~**aten::index_select 需要实现**~~

MLP/MoE(ffn)推理

MLP

```python
class MLP(nn.Module):
    """
    Multi-Layer Perceptron (MLP) used as a feed-forward layer.

    Attributes:
        w1 (nn.Module): Linear layer for input-to-hidden transformation.
        w2 (nn.Module): Linear layer for hidden-to-output transformation.
        w3 (nn.Module): Additional linear layer for feature transformation.
    """
    def __init__(self, dim: int, inter_dim: int):
        """
        Initializes the MLP layer.

        Args:
            dim (int): Input and output dimensionality.
            inter_dim (int): Hidden layer dimensionality.
        """
        super().__init__()
        self.w1 = ColumnParallelLinear(dim, inter_dim)
        self.w2 = RowParallelLinear(inter_dim, dim)
        self.w3 = ColumnParallelLinear(dim, inter_dim)

```

```
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass for the MLP layer.

        Args:
            x (torch.Tensor): Input tensor.

        Returns:
            torch.Tensor: Output tensor after MLP computation.
        """
        return self.w2(F.silu(self.w1(x)) * self.w3(x))


涉及到的需要实现的算子
aten::silu
```

MoE

```
class Expert(nn.Module):
    """
    Expert layer for Mixture-of-Experts (MoE) models.
    """
    def __init__(self, dim: int, inter_dim: int):
        super().__init__()
        self.w1 = Linear(dim, inter_dim)
        self.w2 = Linear(inter_dim, dim)
        self.w3 = Linear(dim, inter_dim)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass for the Expert layer.
        """
        return self.w2(F.silu(self.w1(x)) * self.w3(x))
涉及到的需要实现的算子与MLP一致
aten::silu


class Gate(nn.Module):
    """
    Gating mechanism for routing inputs in a mixture-of-experts (MoE) model.

    Attributes:
        dim (int): Dimensionality of input features.
        topk (int): Number of top experts activated for each input.
        n_groups (int): Number of groups for routing.
        topk_groups (int): Number of groups to route inputs to.
```

```
        score_func (str): Scoring function ('softmax' or 'sigmoid').
        route_scale (float): Scaling factor for routing weights.
        weight (torch.nn.Parameter): Learnable weights for the gate.
        bias (Optional[torch.nn.Parameter]): Optional bias term for the gate.
    """
    def __init__(self, args: ModelArgs):
        """
        Initializes the Gate module.

        Args:
            args (ModelArgs): Model arguments containing gating parameters.
        """
        super().__init__()
        self.dim = args.dim
        self.topk = args.n_activated_experts
        self.n_groups = args.n_expert_groups
        self.topk_groups = args.n_limited_groups
        self.score_func = args.score_func
        self.route_scale = args.route_scale
        self.weight = nn.Parameter(torch.empty(args.n_routed_experts, args.dim))
        self.bias = nn.Parameter(torch.empty(args.n_routed_experts)) if self.dim == 7168 else None

    def forward(self, x: torch.Tensor) -> Tuple[torch.Tensor, torch.Tensor]:
        """
        Forward pass for the gating mechanism.

        Args:
            x (torch.Tensor): Input tensor.

        Returns:
            Tuple[torch.Tensor, torch.Tensor]: Routing weights and selected expert indices.
        """
        scores = linear(x, self.weight)
        if self.score_func == "softmax":
            scores = scores.softmax(dim=-1, dtype=torch.float32)
        else:
            scores = scores.sigmoid()
        original_scores = scores
        if self.bias is not None:
            scores = scores + self.bias
        if self.n_groups > 1:
            scores = scores.view(x.size(0), self.n_groups, -1)
            if self.bias is None:
                group_scores = scores.amax(dim=-1)
```

```
            else:
                group_scores = scores.topk(2, dim=-1)[0].sum(dim=-1)
            indices = group_scores.topk(self.topk_groups, dim=-1)[1]
            mask = scores.new_ones(x.size(0), self.n_groups,
dtype=bool).scatter_(1, indices, False)
            scores = scores.masked_fill_(mask.unsqueeze(-1), float("-
inf")).flatten(1)
        indices = torch.topk(scores, self.topk, dim=-1)[1]
        weights = original_scores.gather(1, indices)
        if self.score_func == "sigmoid":
            weights /= weights.sum(dim=-1, keepdim=True)
        weights *= self.route_scale
        return weights.type_as(x), indices

涉及到的需要实现的算子
aten::fill_.Scalar
aten::sigmoid
aten::amax.out
aten::topk.values
aten::gather.out
aten::sum.IntList_out
aten::div.out
to.dtype
Data type cast for direct_copy_kernel_sipu is not supported now


class MoE(nn.Module):
    """
    Mixture-of-Experts (MoE) module.

    Attributes:
        dim (int): Dimensionality of input features.
        n_routed_experts (int): Total number of experts in the model.
        n_local_experts (int): Number of experts handled locally in
distributed systems.
        n_activated_experts (int): Number of experts activated for each input.
        gate (nn.Module): Gating mechanism to route inputs to experts.
        experts (nn.ModuleList): List of expert modules.
        shared_experts (nn.Module): Shared experts applied to all inputs.
    """
    def __init__(self, args: ModelArgs):
        """
        Initializes the MoE module.

        Args:
            args (ModelArgs): Model arguments containing MoE parameters.
        """
```

```
        super().__init__()
        self.dim = args.dim
        assert args.n_routed_experts % world_size == 0, f"Number of experts must be divisible by world size (world_size={world_size})"
        self.n_routed_experts = args.n_routed_experts
        self.n_local_experts = args.n_routed_experts // world_size
        self.n_activated_experts = args.n_activated_experts
        self.experts_start_idx = rank * self.n_local_experts
        self.experts_end_idx = self.experts_start_idx + self.n_local_experts
        self.gate = Gate(args)
        self.experts = nn.ModuleList([Expert(args.dim, args.moe_inter_dim) if self.experts_start_idx <= i < self.experts_end_idx else None
                                      for i in range(self.n_routed_experts)])
        self.shared_experts = MLP(args.dim, args.n_shared_experts * args.moe_inter_dim)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass for the MoE module.

        Args:
            x (torch.Tensor): Input tensor.

        Returns:
            torch.Tensor: Output tensor after expert routing and computation.
        """
        shape = x.size()
        x = x.view(-1, self.dim)
        weights, indices = self.gate(x)
        y = torch.zeros_like(x)
        counts = torch.bincount(indices.flatten(), minlength=self.n_routed_experts).tolist()
        for i in range(self.experts_start_idx, self.experts_end_idx):
            if counts[i] == 0:
                continue
            expert = self.experts[i]
            idx, top = torch.where(indices == i)
            y[idx] += expert(x[idx]) * weights[idx, top, None]
        z = self.shared_experts(x)
        if world_size > 1:
            dist.all_reduce(y)
        return (y + z).view(shape)
```

涉及到的需要实现的算子

aten::bincount

aten::eq.Scalar_out

```
159   aten::nonzero
```

涉及到的需要sipu实现的算子

MLP module

1. ~~**aten::silu**~~

Gate module

1. ~~**aten::fill_.Scalar**~~
2. ~~*aten::sigmoid*~~
3. *aten::amax.out（new）*
4. ~~*aten::topk.values*~~
5. ~~aten::gather.out~~
6. ~~**aten::sum.IntList_out**~~
7. ~~**aten::div.out**~~
8. ~~to.dtype：Data type cast for direct_copy_kernel_sipu is not supported now~~

MoE module

1. ~~aten::bincount（new）~~
2. ~~*aten::eq.Scalar_out*~~
3. *aten::nonzero（new）*

630 DS 目前存在问题（0623更新）

~~Nonzero~~

1. ~~bool类型的支持**算子**~~

代码块

```
1   RuntimeError: nonzero only supports int32 or int64 tensors on sipu, but get:
    bool
```

2. ~~Running time问题 **框架**~~

~~已解决：这里是print(idx)打印sipu张量造成，非nonzero算子问题。待_reshape_alias算子接入后解决。~~

代码块

```
indices_i = (indices == i).cpu().to(torch.int32).to(device)
idx, top = indices_i .nonzero(as_tuple=True)

indices_i tensor([[0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [0, 0],
        [1, 0],
        [0, 0],
        [1, 0],
        [1, 0],
        [1, 0],
        [0, 0]], device='sipu:0', dtype=torch.int32)
indices_i.shape torch.Size([16, 2])
indices_i.dtype torch.int32
indices_i.device sipu:0
indices_i.stride (2, 1)


/data/users/liuhao2/pytorch/torch/_tensor_str.py:137: UserWarning: 0The operator aten::_reshape_alias appears to be a view operator, but it has no implementation for the backend "sipu:0". View operators don't support falling back to run on the CPU, since the tensor's storage cannot be shared across devices. (Triggered internally at /data/users/liuhao2/pytorch/aten/src/ATen/native/CPUFallback.cpp:304.)
  tensor_view = tensor.reshape(-1)
idx Traceback (most recent call last):
  File "/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py", line 1378, in <module>
    test_Transformer()
  File "/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py", line 1307, in test_Transformer
    output_prefill = model(tokens_sipu, device=torch.device("sipu"))
                     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1736, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1747, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py", line 1238, in forward
    h = layer(h, start_pos, freqs_cis, mask, device=device)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1736, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1747, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py", line 1174, in forward
    x = x.cpu().bfloat16().to(device) + self.ffn(
                                        ^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1736, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/nn/modules/module.py", line 1747, in _call_impl
    return forward_call(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py", line 628, in forward
    print("idx", idx)
  File "/data/users/liuhao2/pytorch/torch/_tensor.py", line 523, in __repr__
    return torch._tensor_str._str(self, tensor_contents=tensor_contents)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/_tensor_str.py", line 708, in _str
    return _str_intern(self, tensor_contents=tensor_contents)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/_tensor_str.py", line 625, in _str_intern
    tensor_str = _tensor_str(self, indent)
```

```
                ^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/_tensor_str.py", line 357, in _tensor_str
    formatter = _Formatter(get_summarized_data(self) if summarize else self)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/data/users/liuhao2/pytorch/torch/_tensor_str.py", line 137, in __init__
    tensor_view = tensor.reshape(-1)
                  ^^^^^^^^^^^^^^^^^^
RuntimeError: setStorage: sizes [4], strides [2], storage offset 0, and itemsize 8 requiring a storage size of 56 are out of bounds for storage of size 32

```

代码块

```
y[idx] += expert(x[idx].to(device), device).to(device) * weights[idx, top, None].to(device)


/data/users/liuhao2/torch_sipu/examples/run_deepseekv3_Transformer.py:631: UserWarning: 0The operator aten::_reshape_alias appears to be a view operator, but it has no implementation for the backend "sipu:0". View operators don't support falling back to run on the CPU, since the tensor's storage cannot be shared across devices. (Triggered internally at /data/users/liuhao2/pytorch/aten/src/ATen/native/CPUFallback.cpp:304.)
```

~~Topk~~

~~**框架**只接入了 float还有两个~~

~~mul ROPE put on CPU~~

~~RuntimeError: "mul_func" not implemented for 'ComplexFloat'~~

代码块

```
y = torch.view_as_real(x * freqs_cis).flatten(3)
```

~~amax~~

~~进展：已解决~~

~~**框架** RuntimeError: Only support 2-D tensor in sipu currently!~~

代码块

```
group_scores = scores.amax(dim=-1,keepdim=True)
```

当expert groups>1的情况下，scores为3维，目前框架这边只能接受2维的输入，目前只对last dim求amax，应该可以在框架侧处理合并外层维度

~~fill~~

~~进展：已解决~~

~~**算子** RuntimeError: "fill_func" not implemented for 'Bool' block by compiler~~

代码块

```
mask = scores.new_ones(x.size(0), self.n_groups, dtype=bool).scatter_(
                1, indices, False
            )
```

tile_format优化分析

MLA

linear之间有rms norm和cat，flatten等操作，难以优化

MLP

代码块

```
class MLP(nn.Module):
    """
    Multi-Layer Perceptron (MLP) used as a feed-forward layer.

    Attributes:
        w1 (nn.Module): Linear layer for input-to-hidden transformation.
        w2 (nn.Module): Linear layer for hidden-to-output transformation.
        w3 (nn.Module): Additional linear layer for feature transformation.
    """

    def __init__(self, dim: int, inter_dim: int, seed: Optional[int] = None):
        """
```

```
        Initializes the MLP layer.

        Args:
            dim (int): Input and output dimensionality.
            inter_dim (int): Hidden layer dimensionality.
            seed (Optional[int]): Seed for weight initialization to ensure identical
                                  weights across different devices. Default is None.
        """
        super().__init__()
        # Use predictable seeds for each layer to ensure identical initialization
        self.w1 = ColumnParallelLinear(
            dim, inter_dim, seed=seed if seed is None else seed + 1
        )
        self.w2 = RowParallelLinear(
            inter_dim, dim, seed=seed if seed is None else seed + 2
        )
        self.w3 = ColumnParallelLinear(
            dim, inter_dim, seed=seed if seed is None else seed + 3
        )

    def forward(self, x: torch.Tensor, device: torch.device = "cpu") -> torch.Tensor:
        """
        Forward pass for the MLP layer.

        Args:
            x (torch.Tensor): Input tensor.

        Returns:
            torch.Tensor: Output tensor after MLP computation.
        """

        res = self.w2(
            F.silu(self.w1(x, device=device).to(device))
            * self.w3(x, device=device).to(device),
            device=device,
        )
        return res

```

linear1（tile） -> silu(linear) -> linear2(tile)-> mul(linear)

^
|
linear3(tile)

tile -> linear 3 times
linear -> tile 1 times

如果silu 和 mul支持tileformat，MLP内部可以一直保持tileformat计算，不会再涉及转换问题

MoE & MoE中的Expert

代码块

```
class MoE(nn.Module):
    """
    Mixture-of-Experts (MoE) module.

    Attributes:
        dim (int): Dimensionality of input features.
        n_routed_experts (int): Total number of experts in the model.
        n_local_experts (int): Number of experts handled locally in distributed systems.
        n_activated_experts (int): Number of experts activated for each input.
        gate (nn.Module): Gating mechanism to route inputs to experts.
        experts (nn.ModuleList): List of expert modules.
        shared_experts (nn.Module): Shared experts applied to all inputs.
    """

    def __init__(self, args: ModelArgs):
        """
        Initializes the MoE module.

        Args:
            args (ModelArgs): Model arguments containing MoE parameters.
        """
        super().__init__()
        self.dim = args.dim
        assert (
            args.n_routed_experts % world_size == 0
        ), f"Number of experts must be divisible by world size (world_size={world_size})"
        self.n_routed_experts = args.n_routed_experts
        self.n_local_experts = args.n_routed_experts // world_size
        self.n_activated_experts = args.n_activated_experts
        self.experts_start_idx = rank * self.n_local_experts
```

```
        self.experts_end_idx = self.experts_start_idx + self.n_local_experts
        self.gate = Gate(args)
        self.experts = nn.ModuleList(
            [
                Expert(args.dim, args.moe_inter_dim)
                if self.experts_start_idx <= i < self.experts_end_idx
                else None
                for i in range(self.n_routed_experts)
            ]
        )
        self.shared_experts = MLP(args.dim, args.n_shared_experts * args.moe_inter_dim)

    def forward(self, x: torch.Tensor, device: torch.device = "cpu") -> torch.Tensor:
        """
        Forward pass for the MoE module.

        Args:
            x (torch.Tensor): Input tensor.

        Returns:
            torch.Tensor: Output tensor after expert routing and computation.
        """
        shape = x.size()
        x = x.view(-1, self.dim)
        weights, indices = self.gate(x, device=device)
        y = torch.zeros_like(x).to(device)
        #breakpoint()
        counts = torch.bincount(
            indices.flatten(), minlength=self.n_routed_experts
        ).tolist()
        for i in range(self.experts_start_idx, self.experts_end_idx):
            if counts[i] == 0:
                continue
            expert = self.experts[i].to(device)
            idx, top = torch.where(indices.cpu() == i)
            print("indices.cpu().shape", indices.cpu().shape)

            weights = weights.cpu()

            y[idx] += expert(x[idx].to(device), device).to(device) * weights[
                idx, top, None
            ].to(device)

        z = self.shared_experts(x, device=device)
        if world_size > 1:
```

```
            dist.all_reduce(y)
        return (y + z).view(shape)


class Expert(nn.Module):
    """
    Expert layer for Mixture-of-Experts (MoE) models.
    """

    def __init__(self, dim: int, inter_dim: int):
        super().__init__()
        self.w1 = Linear(dim, inter_dim)
        self.w2 = Linear(inter_dim, dim)
        self.w3 = Linear(dim, inter_dim)

    def forward(self, x: torch.Tensor, device: torch.device = "cpu") -> torch.Tensor:
        """
        Forward pass for the Expert layer.
        """
        return self.w2(
            F.silu(self.w1(x, device=device)).to(device) * self.w3(x, device=device).to(device),
            device=device,
        )

```

Expert和MLP本质上是一样的，只是feature dim不同，这里的情况和上面MLP中分析的一致
如果可以按照MLP中所说的那样进行优化，MoE的收益会更加明显
n_shared_experts: int = 2 #原始配置
n_activated_experts: int = 6 #原始配置
MoE中的每一个layer激活的路由专家（Expert模块）有6个，共享专家（MLP模块）2个

linear1（tile）-> silu(linear) -> linear2(tile)-> mul(linear)

^
|
linear3(tile)

tile -> linear 3 times
linear ->tile 1 times

如果silu 和 mul支持tileformat，MLP内部可以一直保持tileformat计算，不会再涉及转换问题，

每一个layer的MoE模块可以减少（1+3）*8=24次format的转换

Infer fallback 分析

除了**init weight** 和这个预计算ROPE 的**precompute_freqs_cis** fallback到cpu来做，

其余还fallback的还有

1. [redispatchBoxed] op=[aten::triu.out], key=[CPU]

```
代码块
1   mask = torch.full((seqlen, seqlen), float("-inf"),
    device=tokens.device).triu_(1)
```

2. [redispatchBoxed] op=[aten::zero_], key=[CPU]

```
代码块
1   y = torch.zeros_like(x)
```

3. [redispatchBoxed] op=[aten::index.Tensor_out], key=[CPU]

```
代码块
1   h = self.norm(h)[:, -1]
```

4. [redispatchBoxed] op=[aten::_index_put_impl_], key=[CPU] 算子 宋涵 正在开发

```
代码块
1   mask = (x < self.vocab_start_idx) | (x >= self.vocab_end_idx)
2   x = x - self.vocab_start_idx
3   x[mask] = 0
```

5. [redispatchBoxed] op=[aten::reciprocal.out], key=[CPU] 应该是一些除法自动触发了优化，按照倒数的方式乘法计算 接入int
6. Add type

```
代码块
```

```
class ParallelEmbedding(nn.Module):输出结果为F32
在class Block(nn.Module):当中
x = x + self.attn(self.attn_norm(x), start_pos, freqs_cis, mask, device=device)
x = x + self.ffn(self.ffn_norm(x), device=device)
x 为 f32， self.attn，self.ffn为BF16，直接相加 nan
```

7. 开源 infer code 修改 group_scores = scores.amax(*dim*=-1, ***keepdim*=True**) fixed amax支持keepdim=false

代码块

```
if self.n_groups > 1:
    scores = scores.view(x.size(0), self.n_groups, -1)
    if self.bias is None:
        group_scores = scores.amax(dim=-1)
    else:
        group_scores = scores.topk(2, dim=-1)[0].sum(dim=-1)
    indices = group_scores.topk(self.topk_groups, dim=-1)[1]
    mask = scores.new_ones(x.size(0), self.n_groups, dtype=bool).scatter_(1, indices, False)
    scores = scores.masked_fill_(mask.unsqueeze(-1), float("-inf")).flatten(1)
```

改为

代码块

```
        if self.n_groups > 1:
            scores = scores.view(x.size(0), self.n_groups, -1)
            if self.bias is None:
                group_scores = scores.amax(dim=-1, keepdim=True)
            else:
                group_scores = scores.topk(2, dim=-1)[0].sum(dim=-1)
            indices = group_scores.topk(self.topk_groups, dim=-1)[1]
            mask = scores.new_ones(x.size(0), self.n_groups, dtype=bool).to(device)
            indices_for_scatter = indices.squeeze(-1) if indices.dim() > 2 else indices
            mask = mask.scatter_(1, indices_for_scatter.to(device), False)
            scores = scores.masked_fill_(mask.unsqueeze(-1), float("-inf")).flatten(1)
```