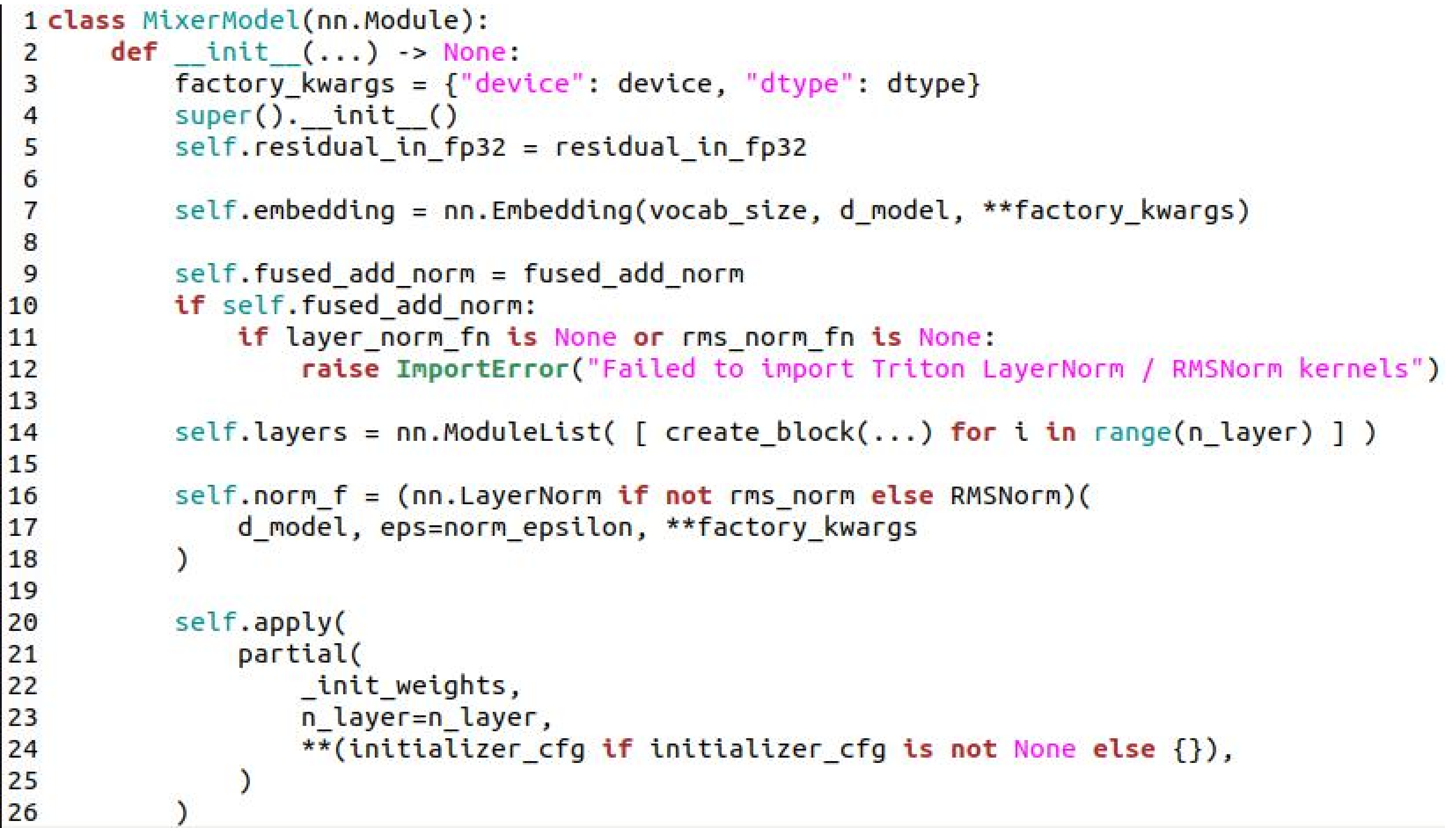
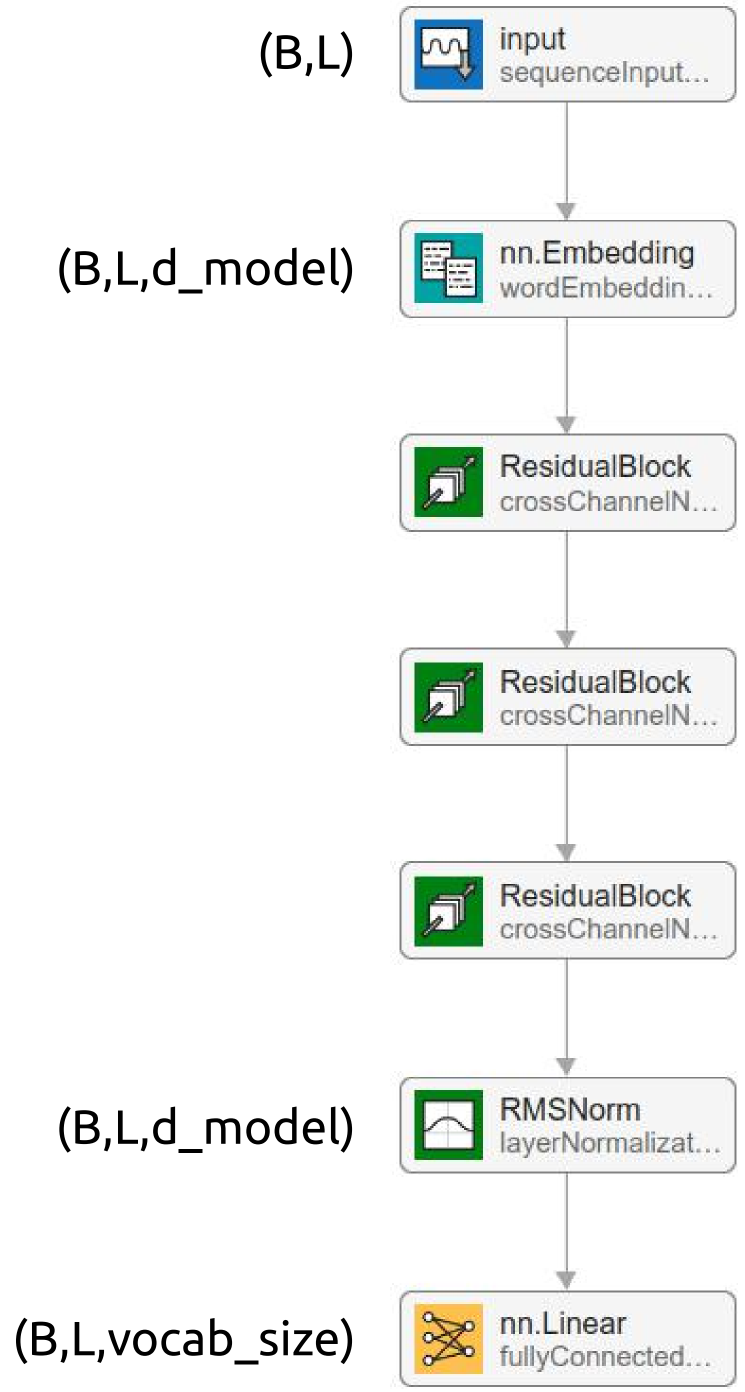
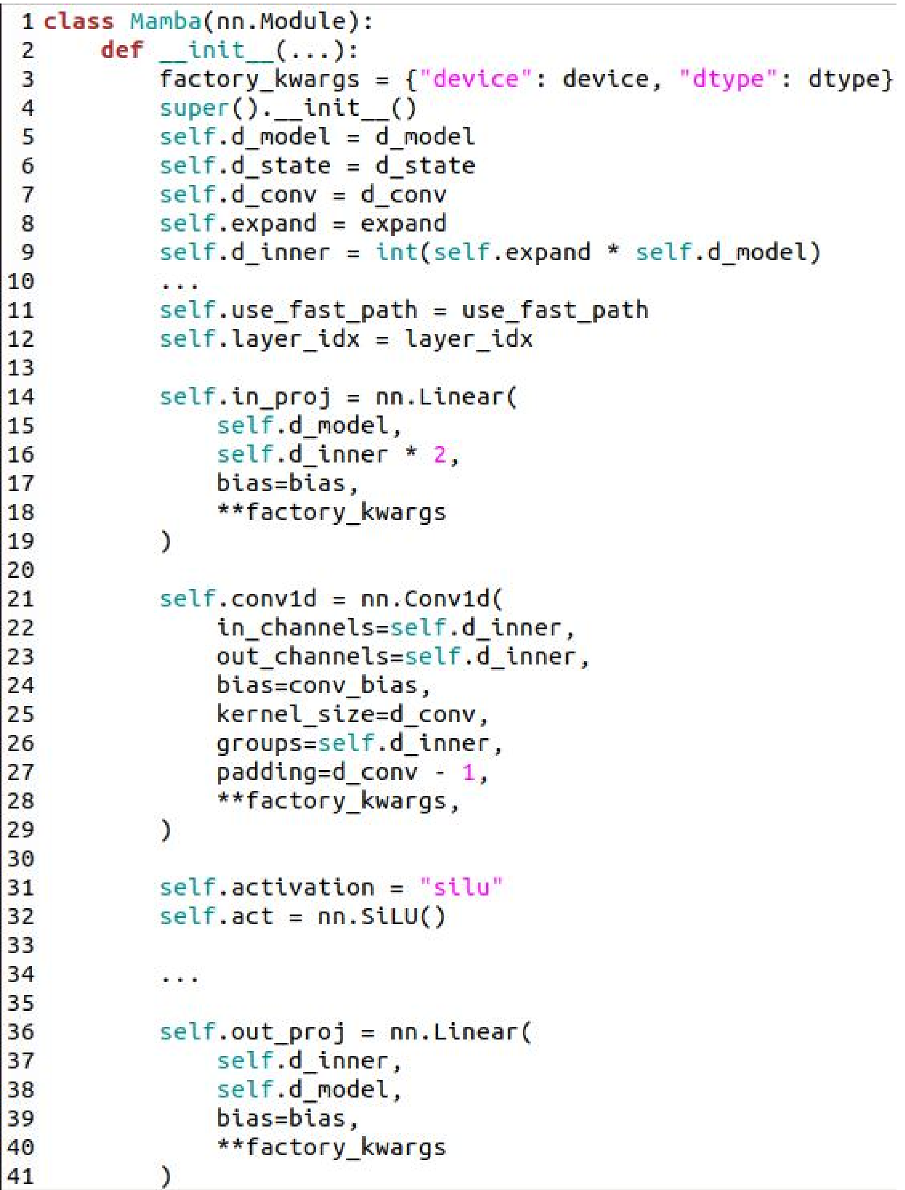
input_ids：词元化的句子，每个词元用id表示
B是batch_size，L是句子长度
input_ids作为输入送入embedding层
> self.embedding = nn.Embedding(vocab_size, d_model, **factory_kwargs)
vocab_size是词汇表的大小，d_model表示状态空间维数（词向量维数）
接下来送入若干个Block（ResidualBlock, named from mamba-minimal）
得到的结果作归一化（RMSNorm or nn.LayerNorm）后送入线性层得到logits
logits可以用于classify或者decode之后用于generate



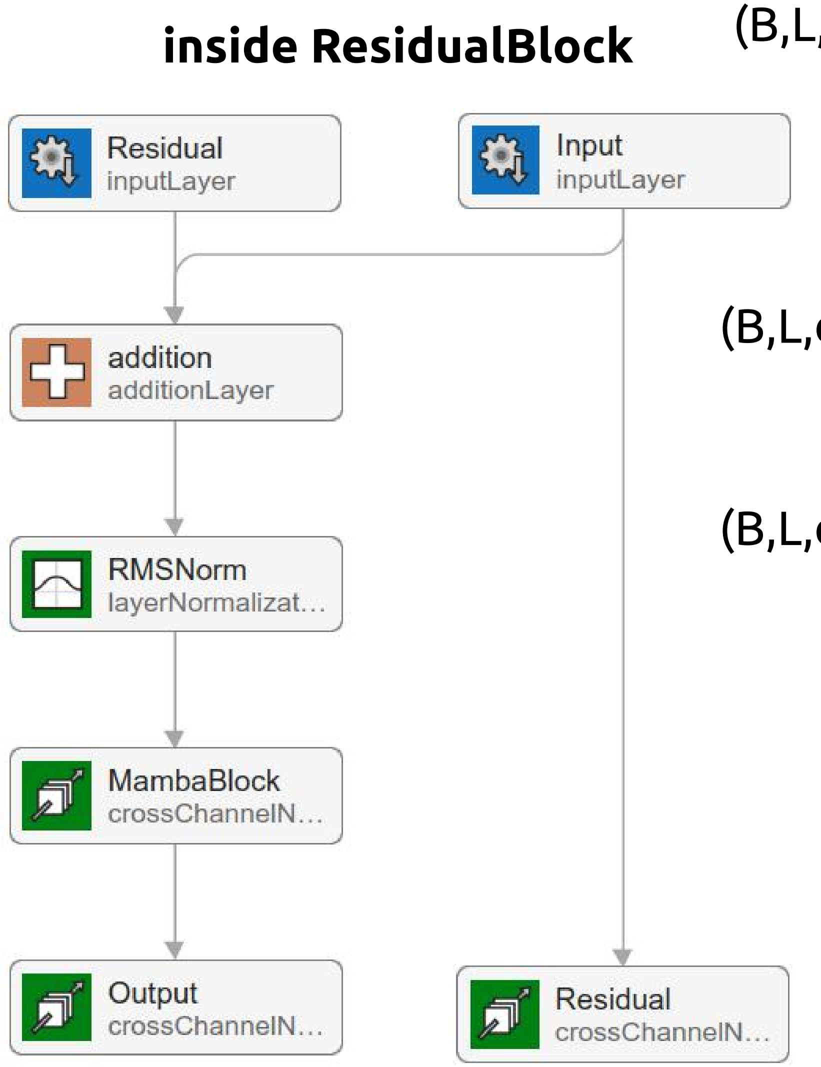
```python
class MixerModel(nn.Module):
    def __init__(...) -> None:
        factory_kwargs = {"device": device, "dtype": dtype}
        super().__init__()
        self.residual_in_fp32 = residual_in_fp32

        self.embedding = nn.Embedding(vocab_size, d_model, **factory_kwargs)

        self.fused_add_norm = fused_add_norm
        if self.fused_add_norm:
            if layer_norm_fn is None or rms_norm_fn is None:
                raise ImportError("Failed to import Triton LayerNorm / RMSNorm kernels")

        self.layers = nn.ModuleList( [ create_block(...) for i in range(n_layer) ] )

        self.norm_f = (nn.LayerNorm if not rms_norm else RMSNorm)(
            d_model, eps=norm_epsilon, **factory_kwargs
        )

        self.apply(
            partial(
                _init_weights,
                n_layer=n_layer,
                **(initializer_cfg if initializer_cfg is not None else {}),
            )
        )
```

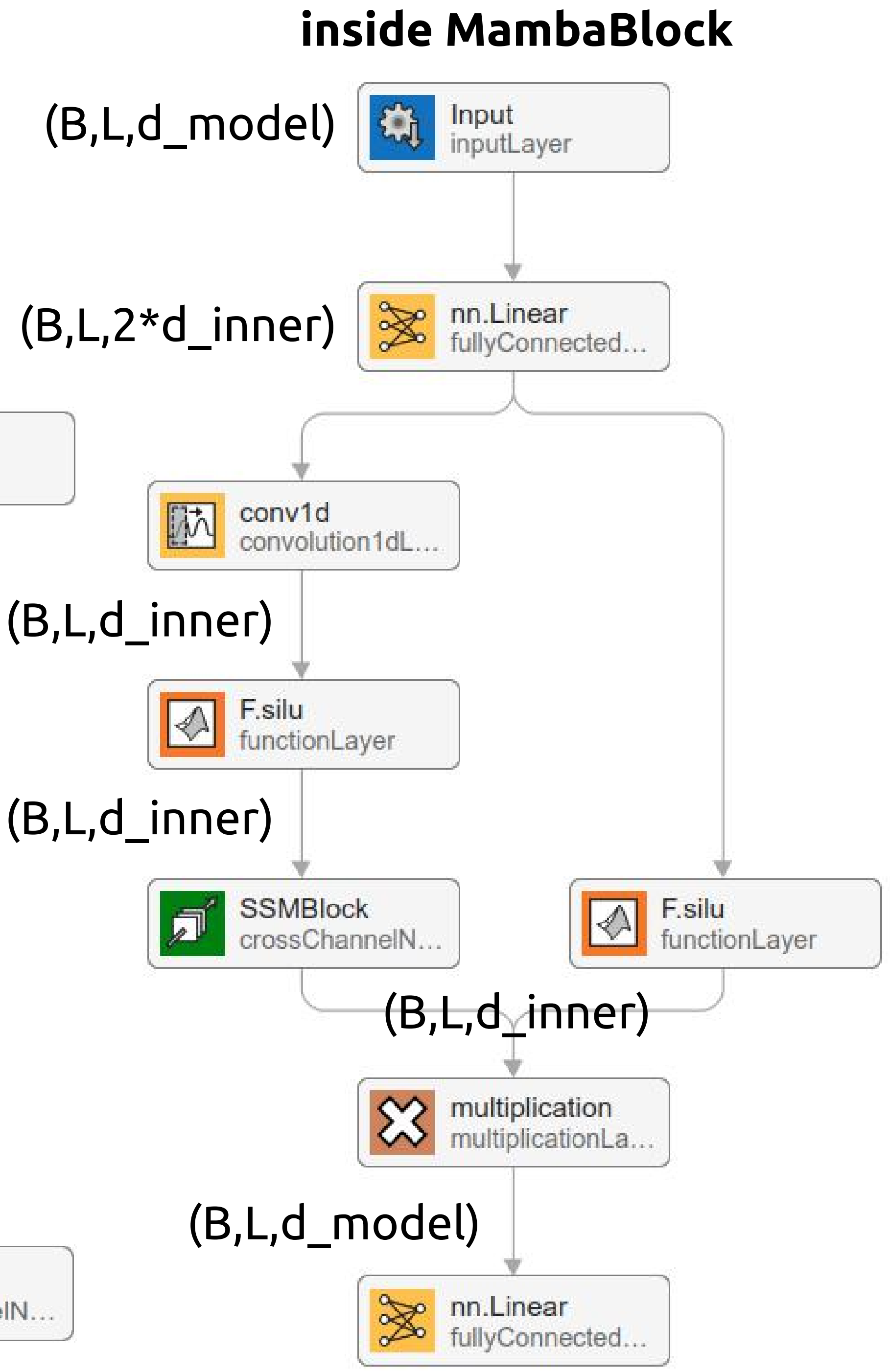


d_inner是内部维度，d_inner=d_model * expand（expand=2 by default）
在MambaBlock内部，输入经过第一个线性层之后等分为d_inner的两份，
分别交付两条计算路线。F.silu是激活函数。



```python
class Mamba(nn.Module):
    def __init__(...):
        factory_kwargs = {"device": device, "dtype": dtype}
        super().__init__()
        self.d_model = d_model
        self.d_state = d_state
        self.d_conv = d_conv
        self.expand = expand
        self.d_inner = int(self.expand * self.d_model)
        ...
        self.use_fast_path = use_fast_path
        self.layer_idx = layer_idx

        self.in_proj = nn.Linear(
            self.d_model,
            self.d_inner * 2,
            bias=bias,
            **factory_kwargs
        )

        self.conv1d = nn.Conv1d(
            in_channels=self.d_inner,
            out_channels=self.d_inner,
            bias=conv_bias,
            kernel_size=d_conv,
            groups=self.d_inner,
            padding=d_conv - 1,
            **factory_kwargs,
        )

        self.activation = "silu"
        self.act = nn.SiLU()

        ...

        self.out_proj = nn.Linear(
            self.d_inner,
            self.d_model,
            bias=bias,
            **factory_kwargs
        )
```


inside ResidualBlock
Residual
inputLayer
Input
inputLayer
addition
additionLayer
RMSNorm
layerNormalizat...
MambaBlock
crossChannelN...
Output
crossChannelN...
Residual
crossChannelN...


inside MambaBlock
(B,L,d_model)
Input
inputLayer
(B,L,2*d_inner)
nn.Linear
fullyConnected...
conv1d
convolution1dL...
(B,L,d_inner)
F.silu
functionLayer
(B,L,d_inner)
SSMBlock
crossChannelN...
F.silu
functionLayer
(B,L,d_inner)
multiplication
multiplicationLa...
(B,L,d_model)
nn.Linear
fullyConnected...