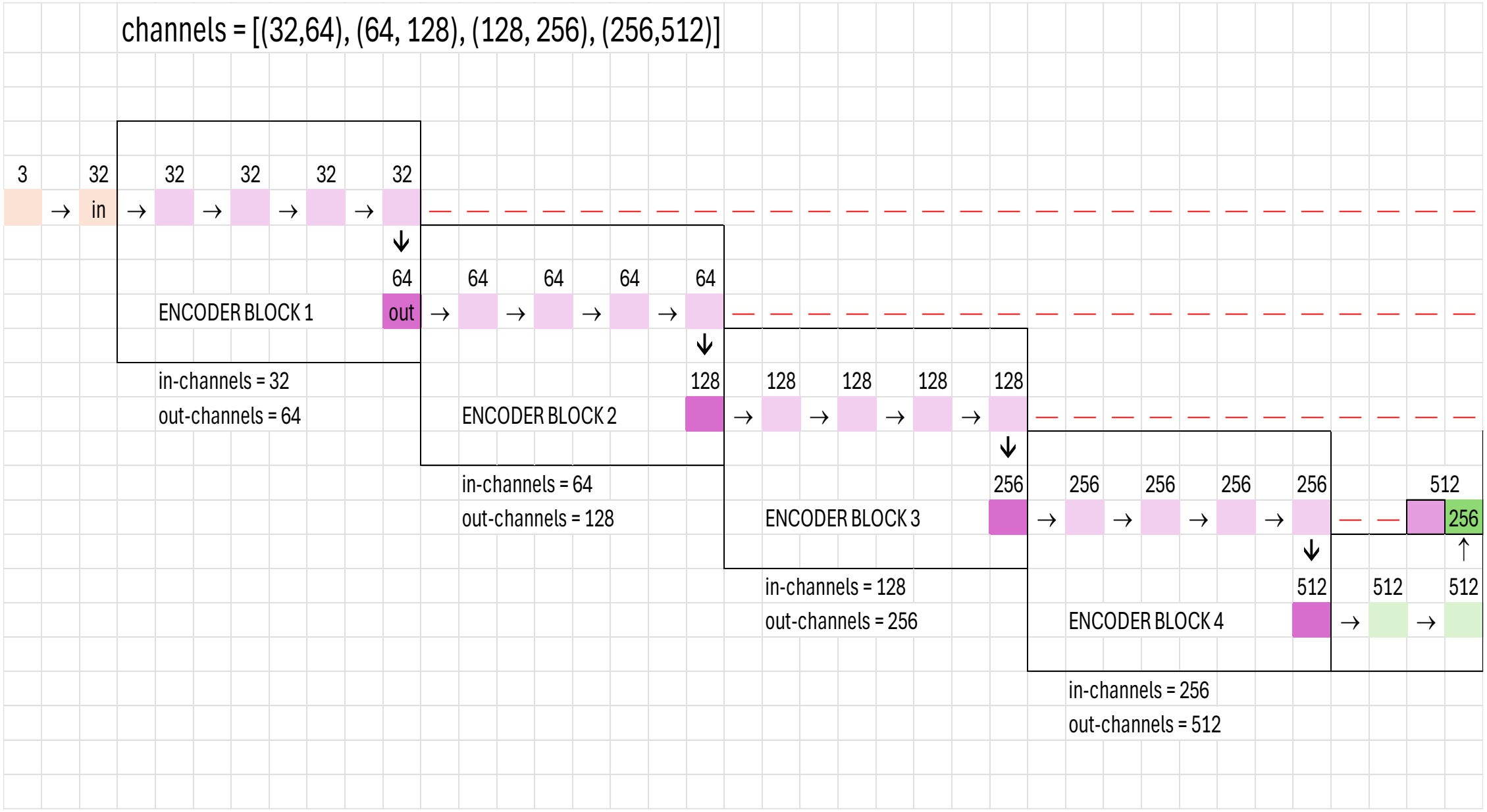
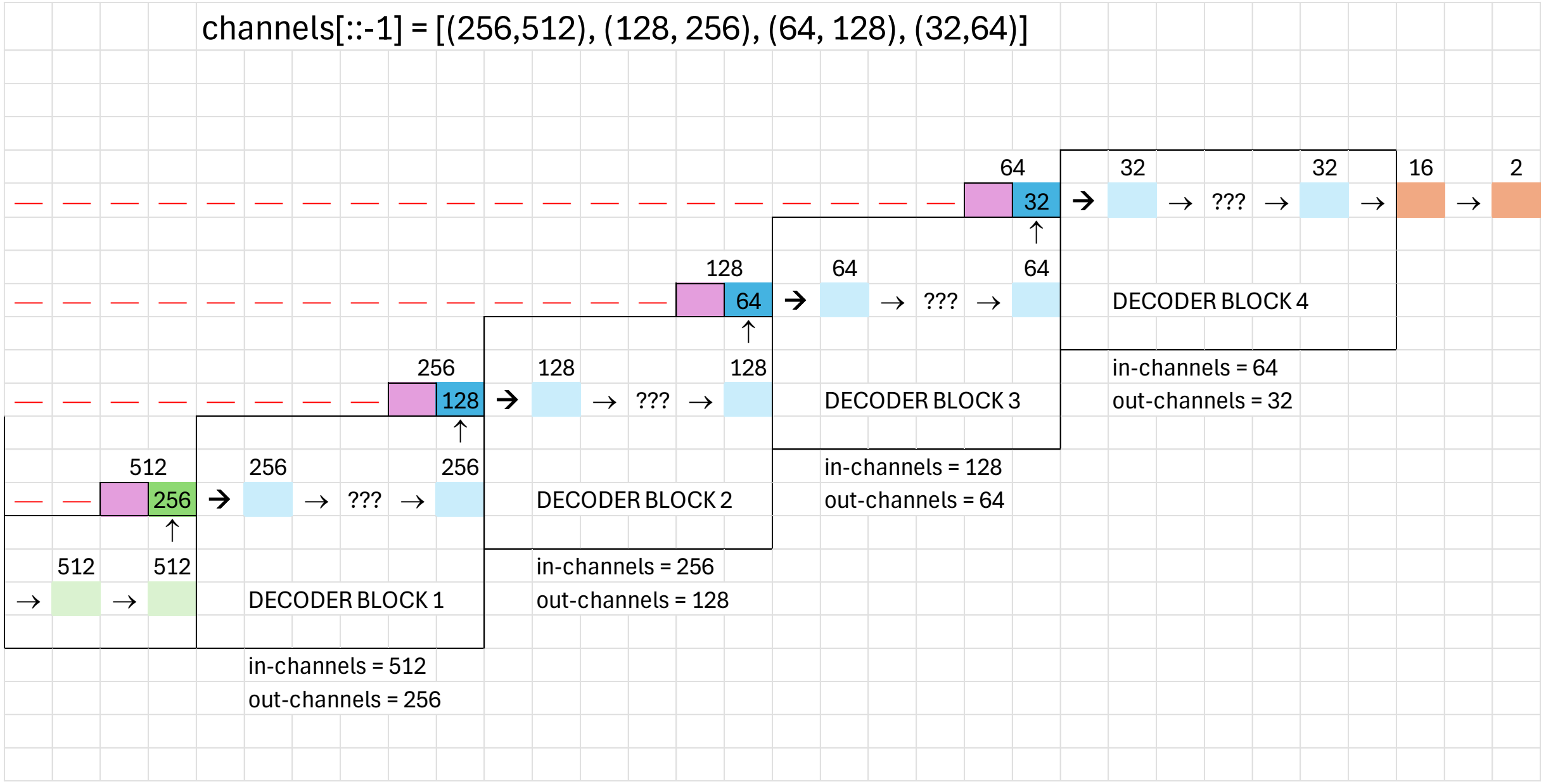
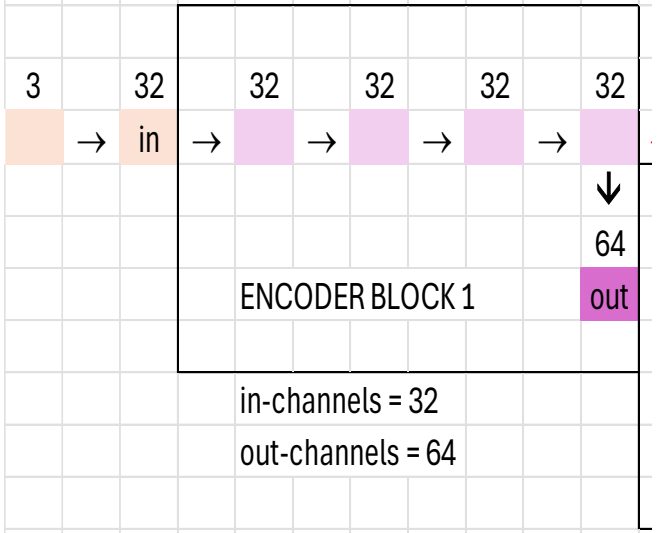

channels = [(32,64), (64, 128), (128, 256), (256,512)]
3
32
in
32
32
32
32
64
64
64
64
64
out
ENCODER BLOCK 1
in-channels = 32
out-channels = 64
128
128
128
128
128
ENCODER BLOCK 2
in-channels = 64
out-channels = 128
256
256
256
256
256
ENCODER BLOCK 3
in-channels = 128
out-channels = 256
512
512
512
512
ENCODER BLOCK 4
in-channels = 256
out-channels = 512


channels[::-1] = [(256,512), (128, 256), (64, 128), (32,64)]
512
256
512
512
DECODER BLOCK 1
in-channels = 512
out-channels = 256
256
128
256
DECODER BLOCK 2
in-channels = 256
out-channels = 128
128
64
128
DECODER BLOCK 3
in-channels = 128
out-channels = 64
64
32
64
DECODER BLOCK 4
in-channels = 64
out-channels = 32
32
16
2
???

channels = [(32,64), (64, 128), (128, 256), (256,512)]




```python
class EncoderBlock(nn.Module):
    def __init__(self,in_channels,out_channels,time_embedding_dim):
        super().__init__()
        self.conv0=nn.Sequential(*[ResidualBottleneck(in_channels,in_channels) for i in range(3)]
                                   ResidualBottleneck(in_channels,out_channels//2))

        self.time_mlp=TimeMLP(embedding_dim=time_embedding_dim,hidden_dim=out_channels,out_dim=ou
        self.conv1=ResidualDownsample(out_channels//2,out_channels)

    def forward(self,x,t=None):
        x_shortcut=self.conv0(x)
        if t is not None:
            x=self.time_mlp(x_shortcut,t)
        x=self.conv1(x)

        return [x,x_shortcut]

```

in-channels = 256
out-channels = 512