Swin Transformer 从零解读

主讲人：DASOU

回复【Swin】获取对应PPT

扫码关注微信公众号

文章周更

知识分享

一起进步

求关注，求点赞，求一切！！

1. 回顾TRM和VIT模型，理清SwinTRM的创新点

2. SwinTRM使用到相对位置编码介绍

3. 窗口移动注意力机制SW-MSA介绍

4. Patch Merging介绍

回顾TRM模型

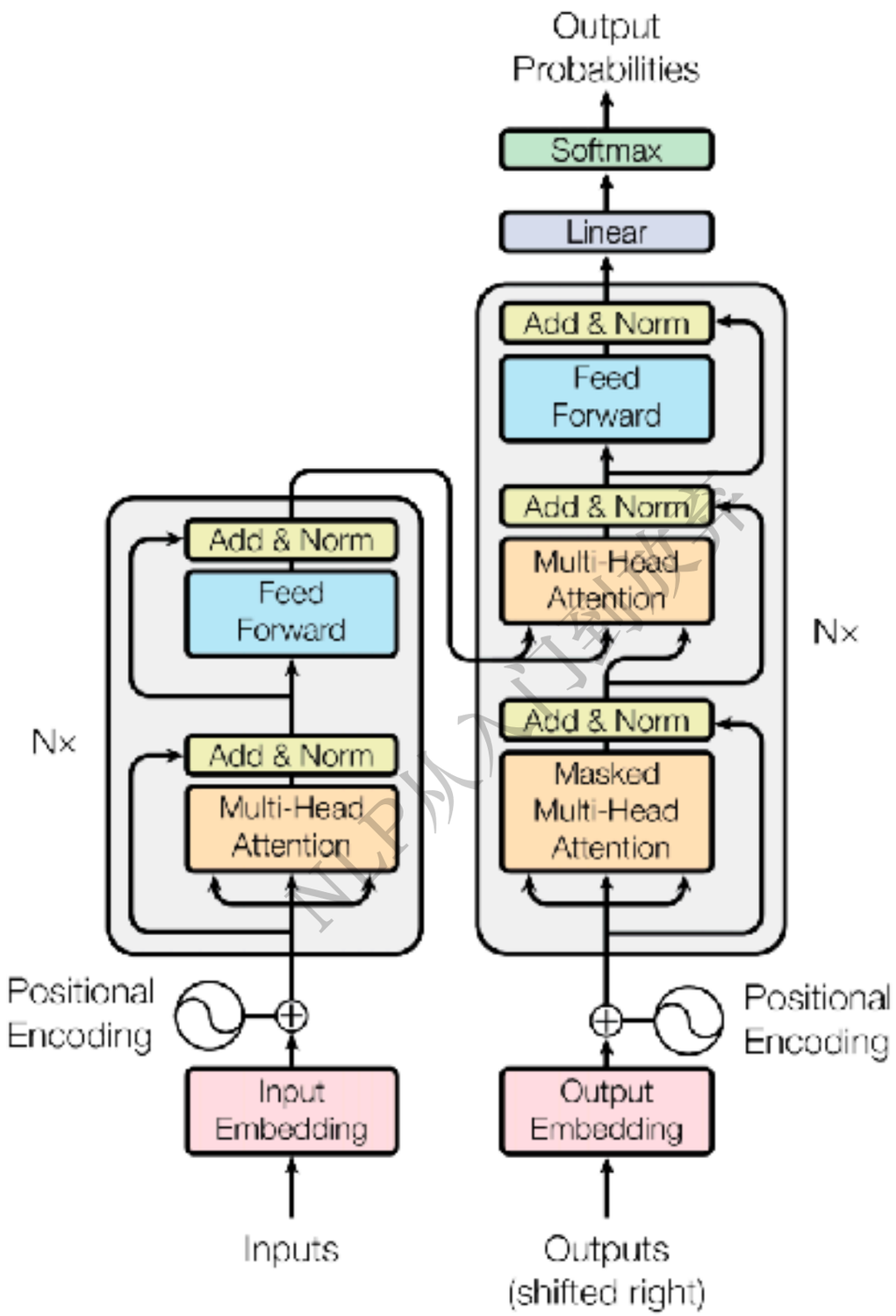
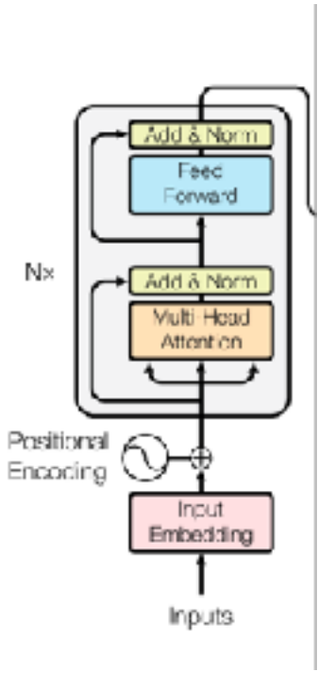


Figure 1: The Transformer - model architecture.



.....

encoder

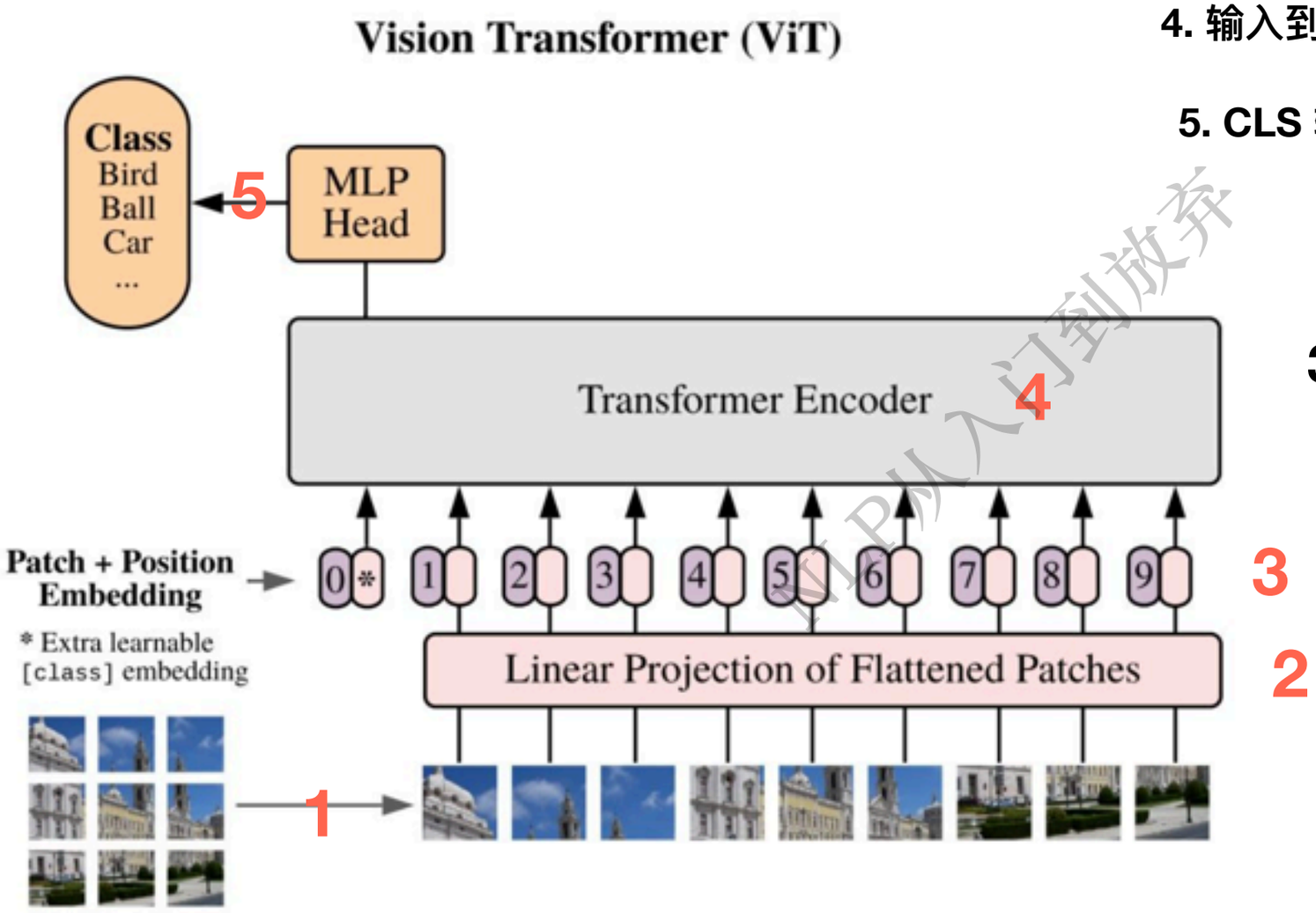
768 768 768

encoder

768 768 768

我 爱 你

回顾VIT模型



1. 图片切分为patch

2. patch转化为embedding

3.位置embedding和tokensembedding相加

4. 输入到TRM模型

5. CLS 输出做多分类任务

3.1 生成CLS符号的TokenEMB

3.2 生成所有序列的位置编码

3.3 tonken+位置编码

SwinTRM做到了两点

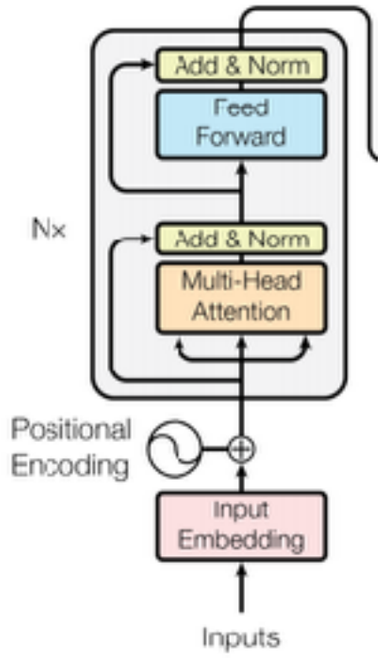
1. 金字塔形状：感受野是在不停的变大的

2. 注意力机制放在一个窗口内部

当时我们VIT模型中怎么把图片变成token的时候聊过

大部分人的思路

224

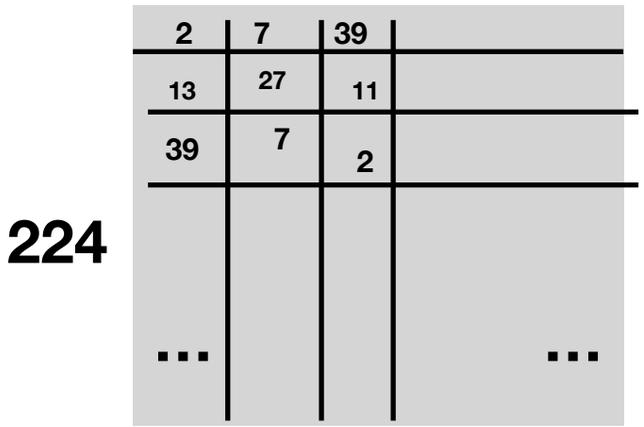


2,7,39…13,27,11…

复杂度的问题

224*224*1

224



→ 序列长度=224*224=50176

BERT的最大长度是512，相当于100倍

如何处理复杂度的问题？：本质上是去解决随着像素增加，复杂度平方级增长的问题；

1.局部注意力机制

有很多中方法：

2. 改进attention公式

3……..

VIT是图片化整为零：由一个像素点转为一块像素点

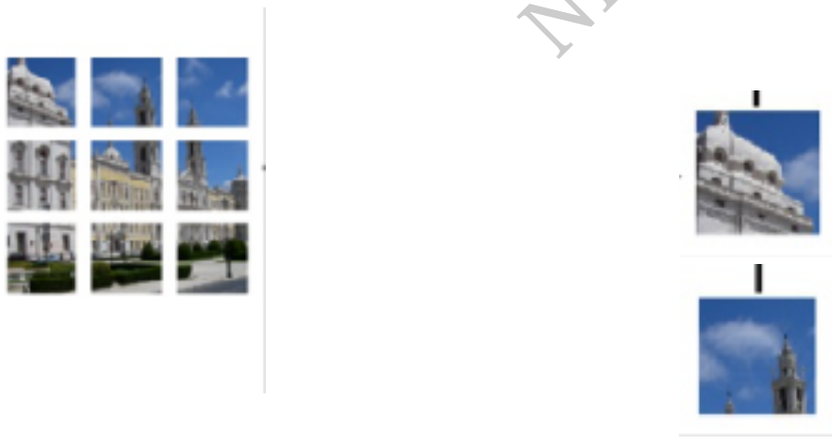
一个简单的改进方式：图像化整为零，切分patch

也就说原来是一个像素点代表一个token，
现在是一大块的token一个patch作为一个token

VIT的问题就是如果图像过大，patch就变大了，复杂度还是不太好

VIT

Swintrm



1： 224*224*3到(224/4)*(224/4)*(4*4*3)

来看SwinTRM的架构图　　2. (224/4)*(224/4)*(4*4*3)到(224/4)*(224/4)*(96)

3. (224/4)*(224/4)*(96)到(224/4)*(224/4)*(96)

4. (224/4)*(224/4)*(96)到(224/8)*(224/8)*(96*2=192)

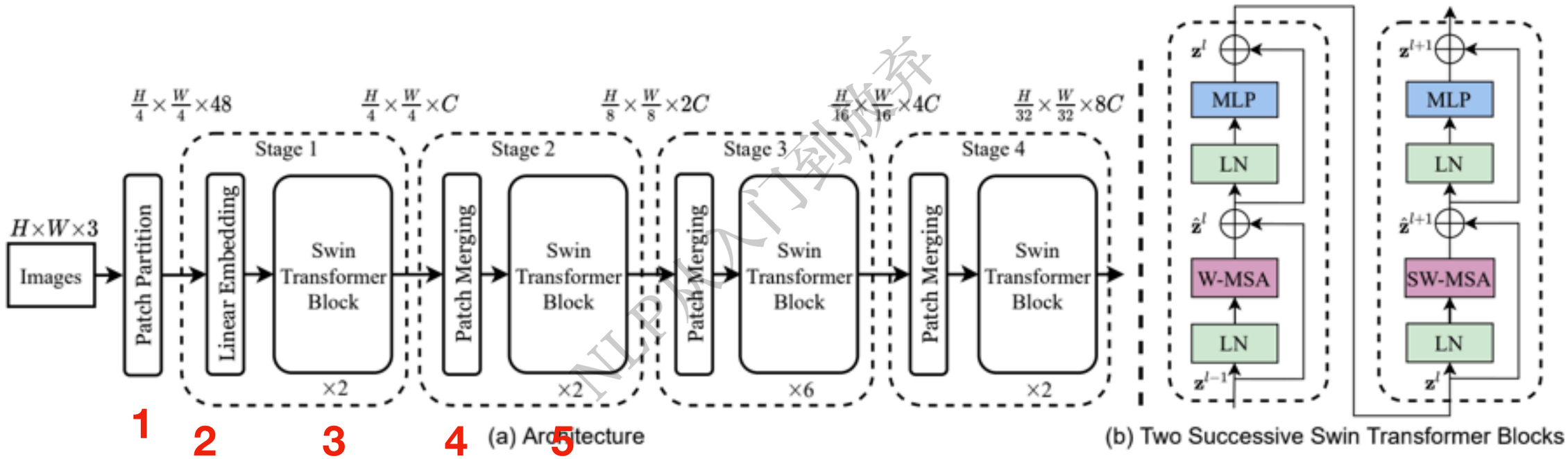
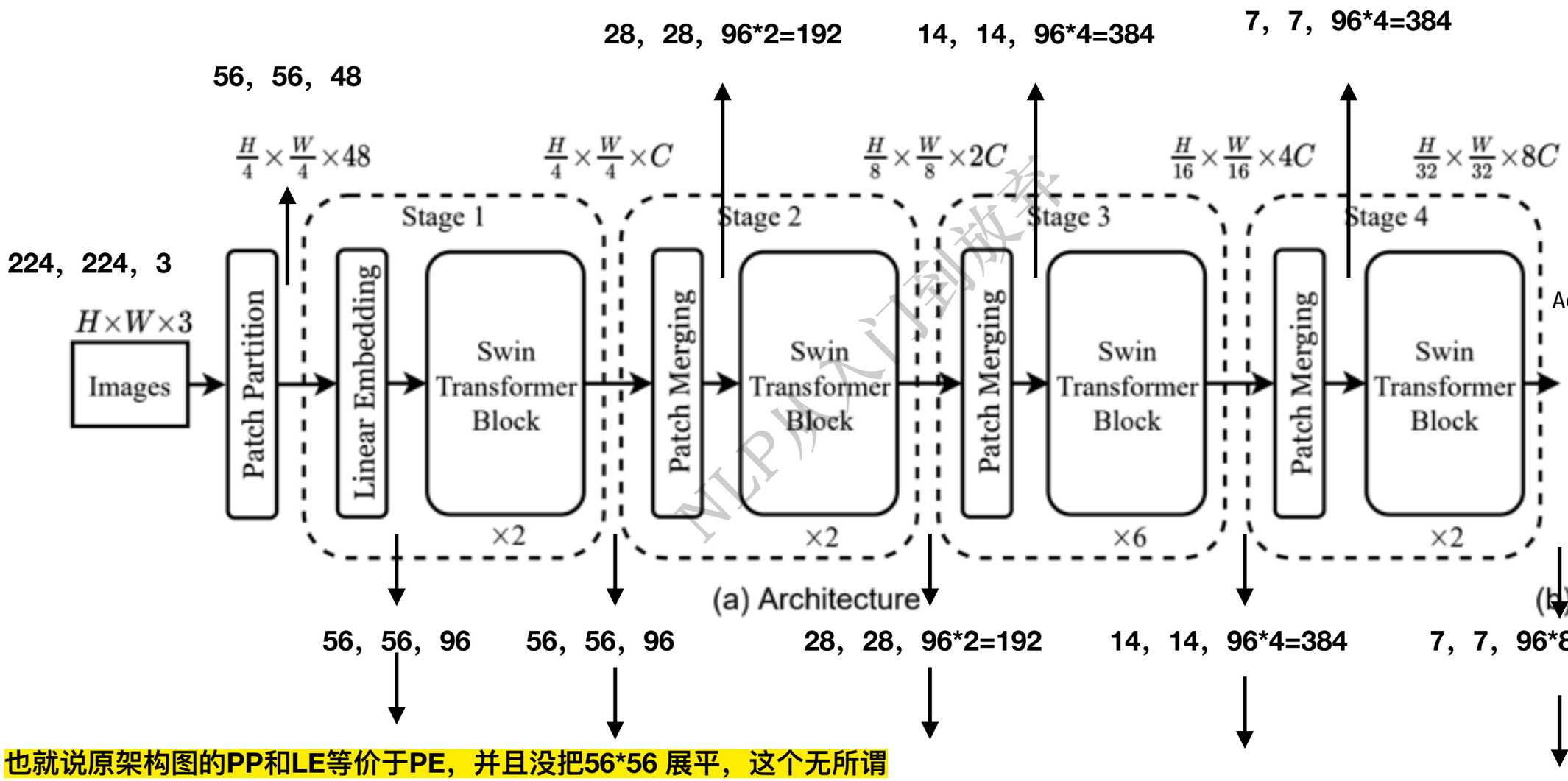


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

5. (224/8)*(224/8)*(96*2=192)到(224/8)*(224/8)*(96*2=192)

H是224，W是224



也就说原架构图的PP和LE等价于PE，并且没把56*56 展平，这个无所谓

来看SwinTRM的架构图

代码中实现的方式，按照模块划分是以下红框的形式进行的，和论文中的方式不同

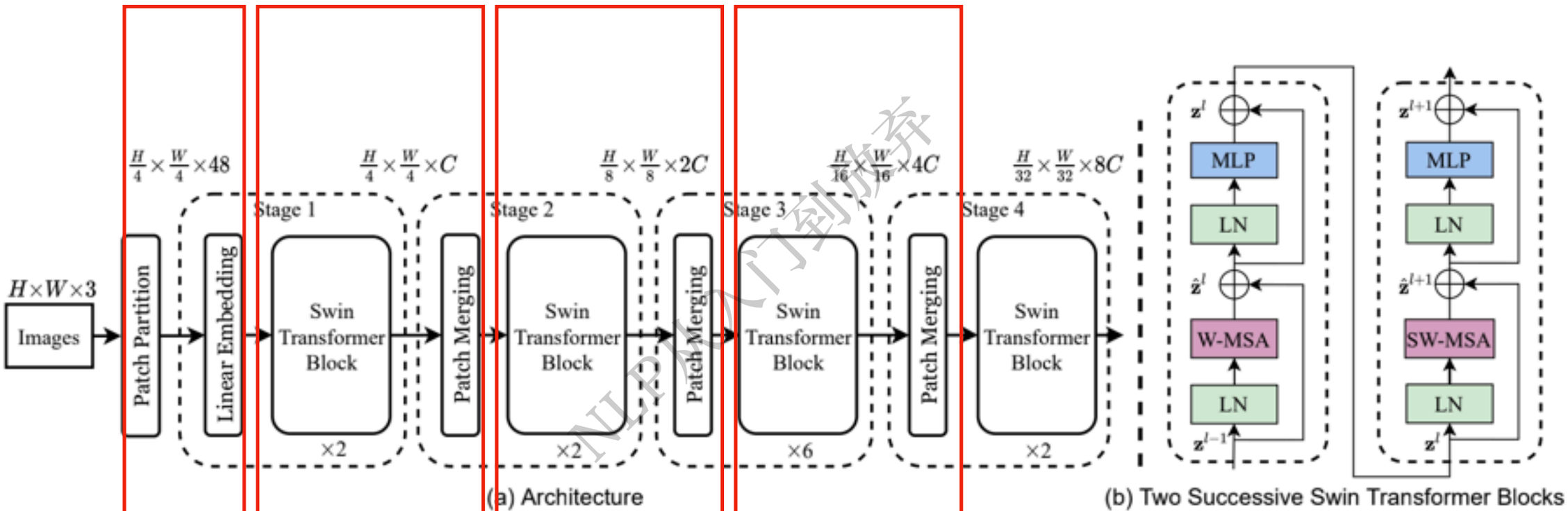
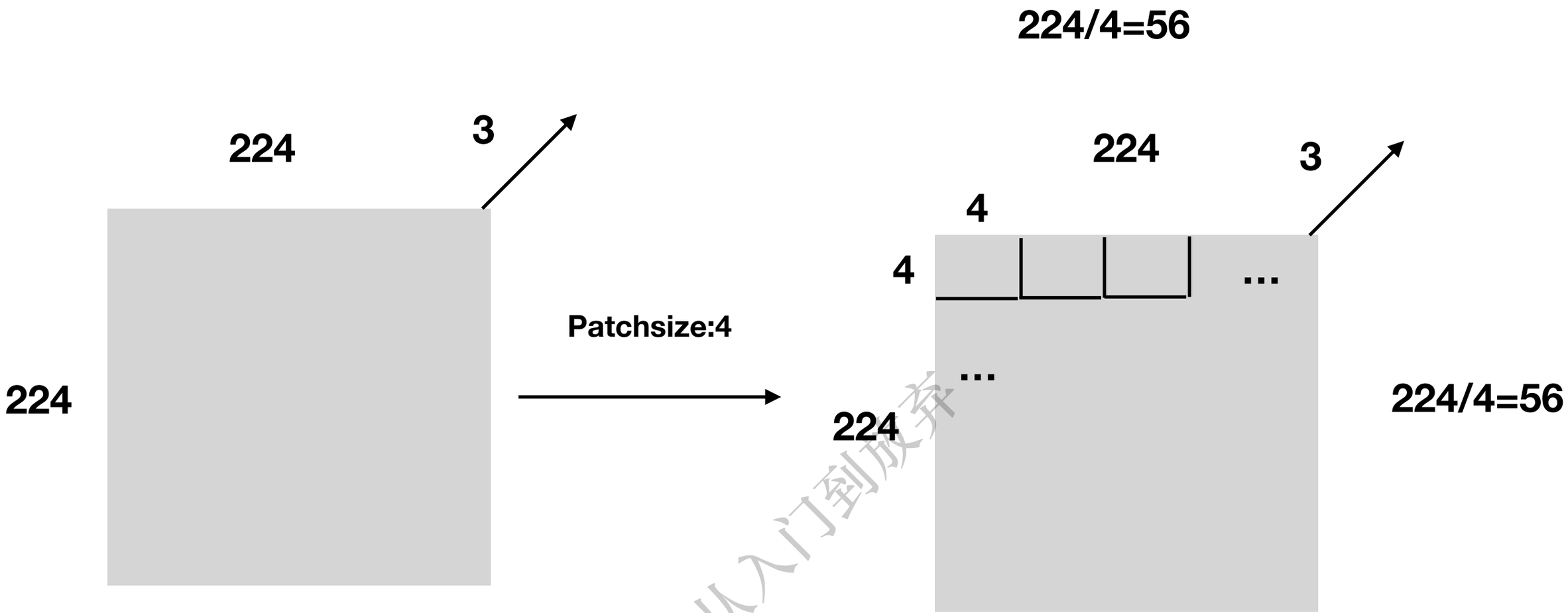


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

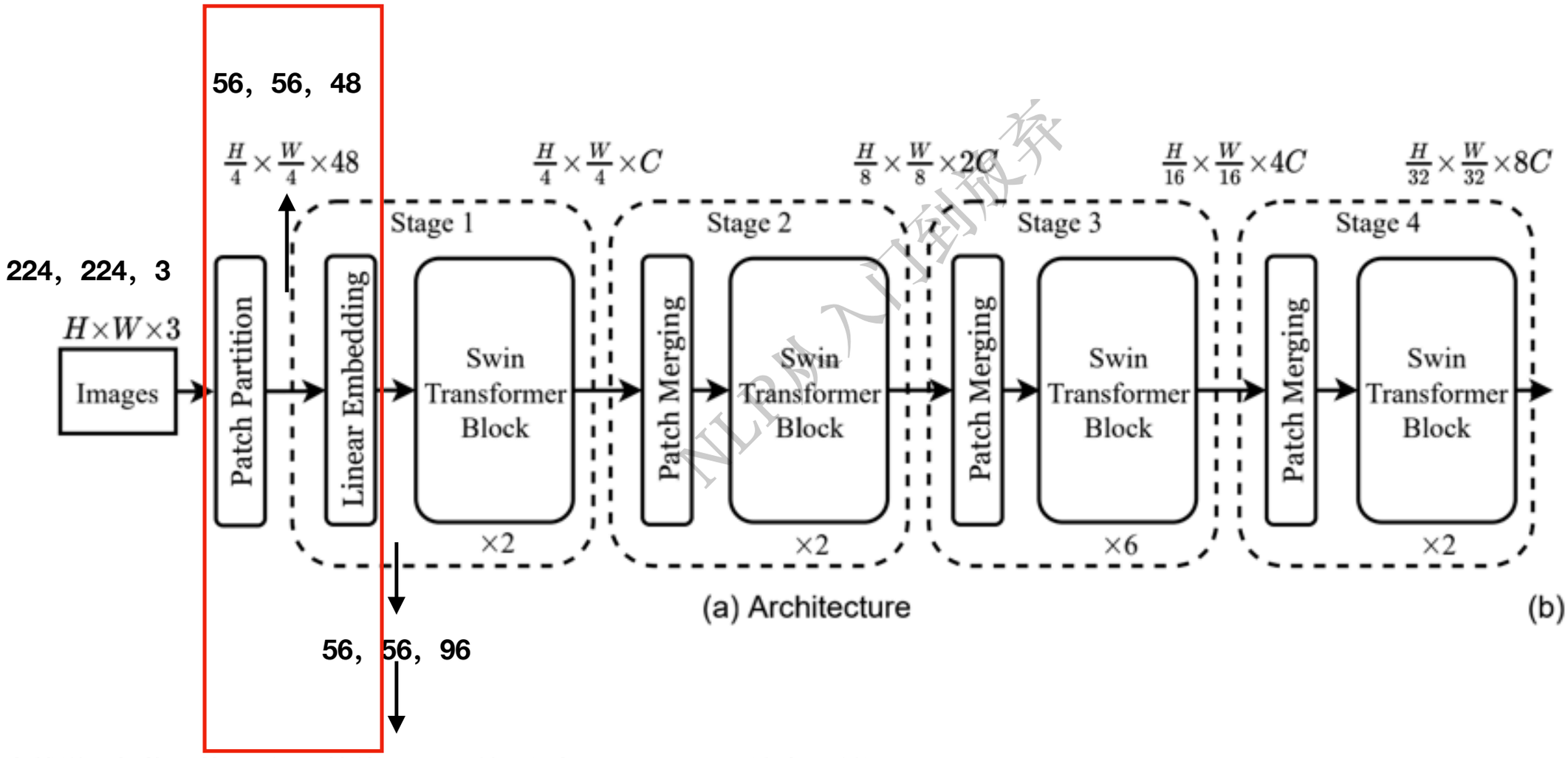


Patch数目就是：56*56=3136

1：224*224*3到(224/4)*(224/4)*(4*4*3)

2. (224/4)*(224/4)*(4*4*3)到(224/4)*(224/4)*(96)

H是224，W是224



也就说原架构图的PP和LE等价于PE，并且没把56*56 展平，这个无所谓

这里首先将图片的最小单位从像素转变为patch。论文中所给的示例为一个patch由4*4个pixel构成，每个patch的像素融入3通道，维度升至48 —— Patch Partition
然后通过线性embedding将维度提升至96维度 —— Linear Embedding

这里patch embedding和linear embedding是直接通过一层卷积实现的conv(3, 96, kernel=(4,4), stride=(4,4)), flatten和layernorm

第二个红色框框的操作

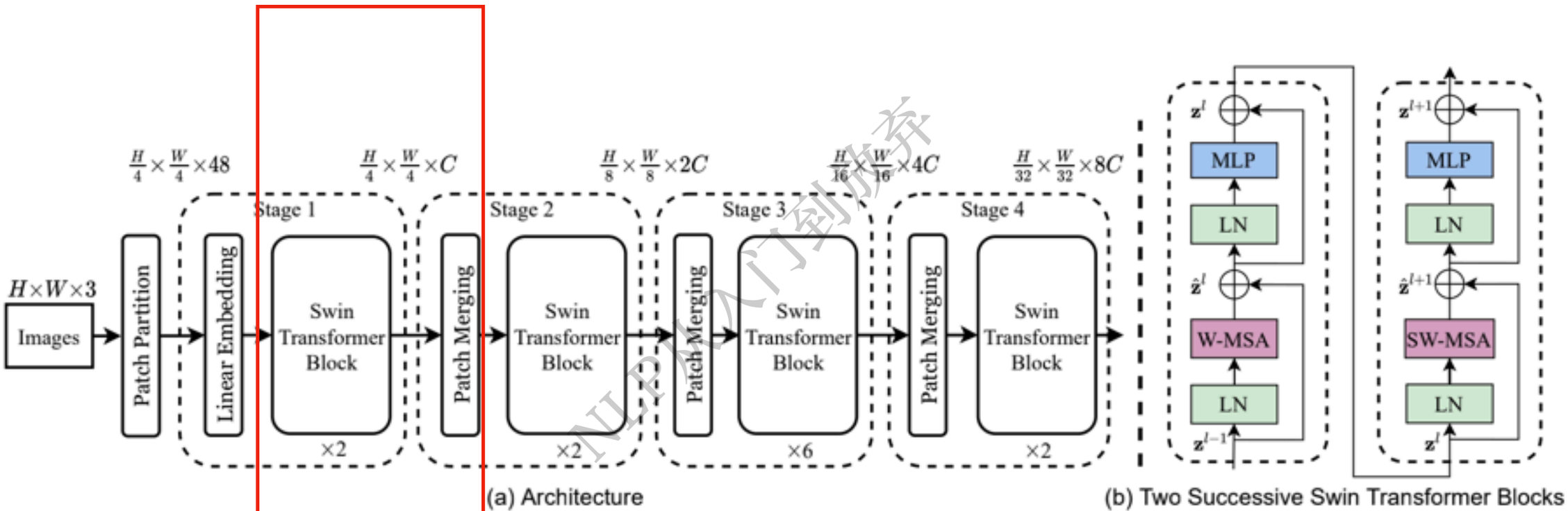
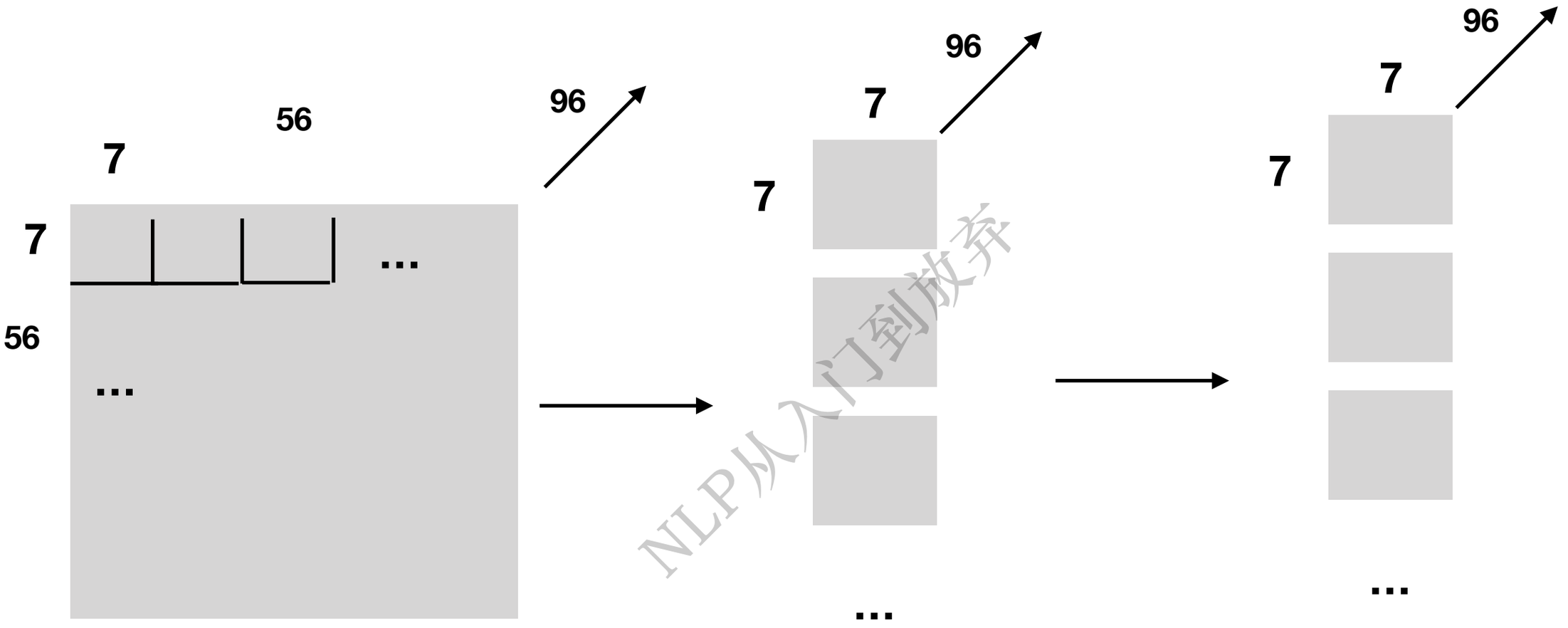


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

切分窗口：win_size=7一直保持不变

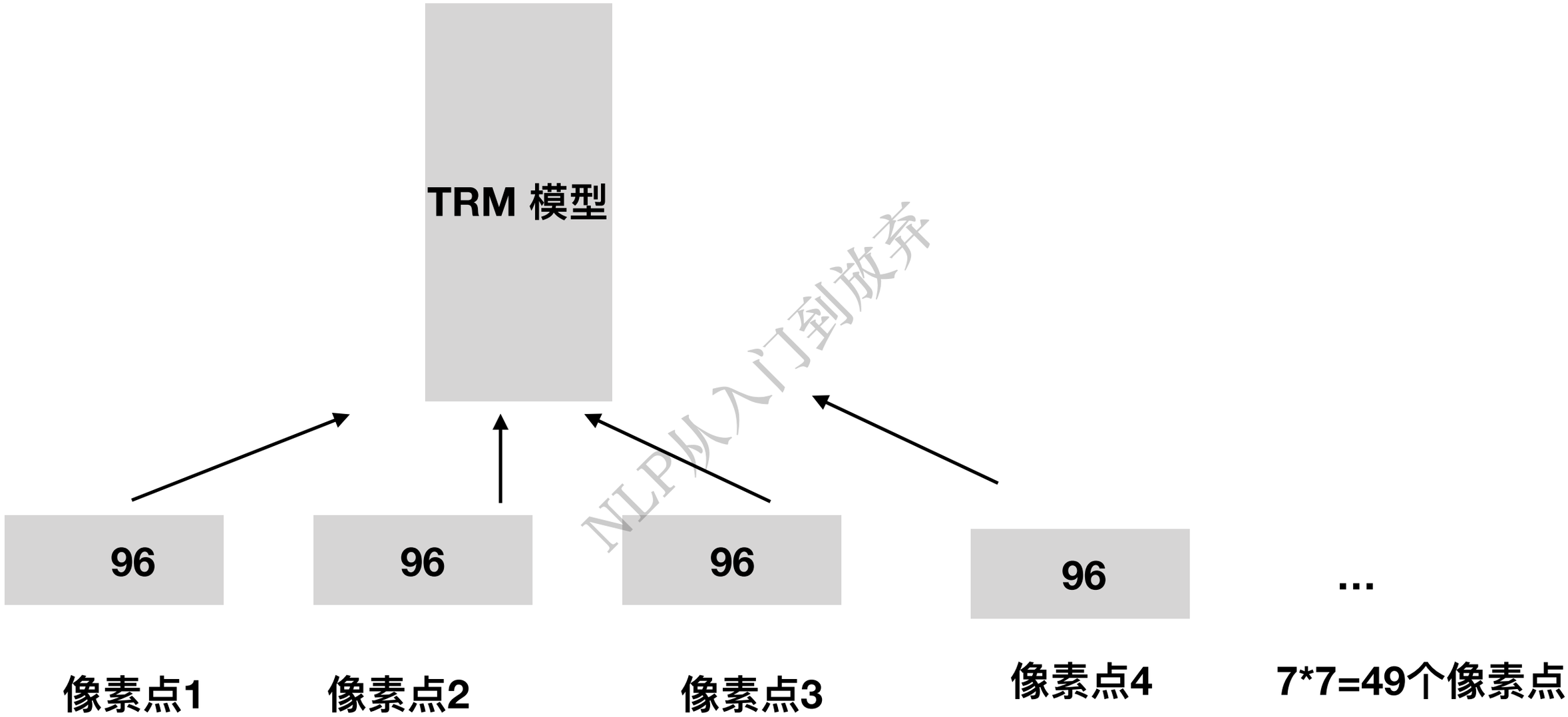


1,56,56,96

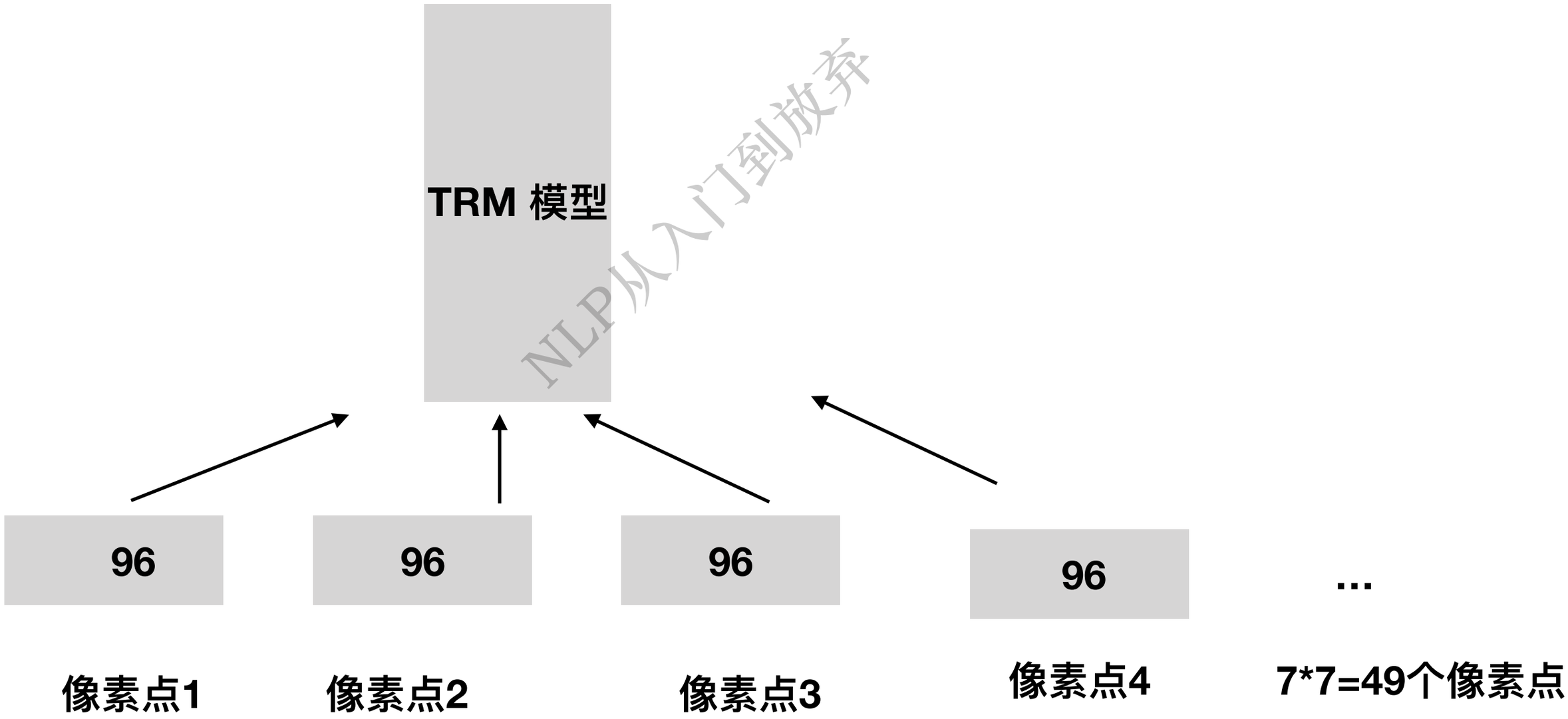
1,(56/7*56/7=64)，7，7，96

(56/7*56/7=64)，7，7，96

把一个图片的win数目移到bs维度上，就是整个patch有多少w



很简单：把每个元素点作为token，96个通道数作为token维度



位置编码问题

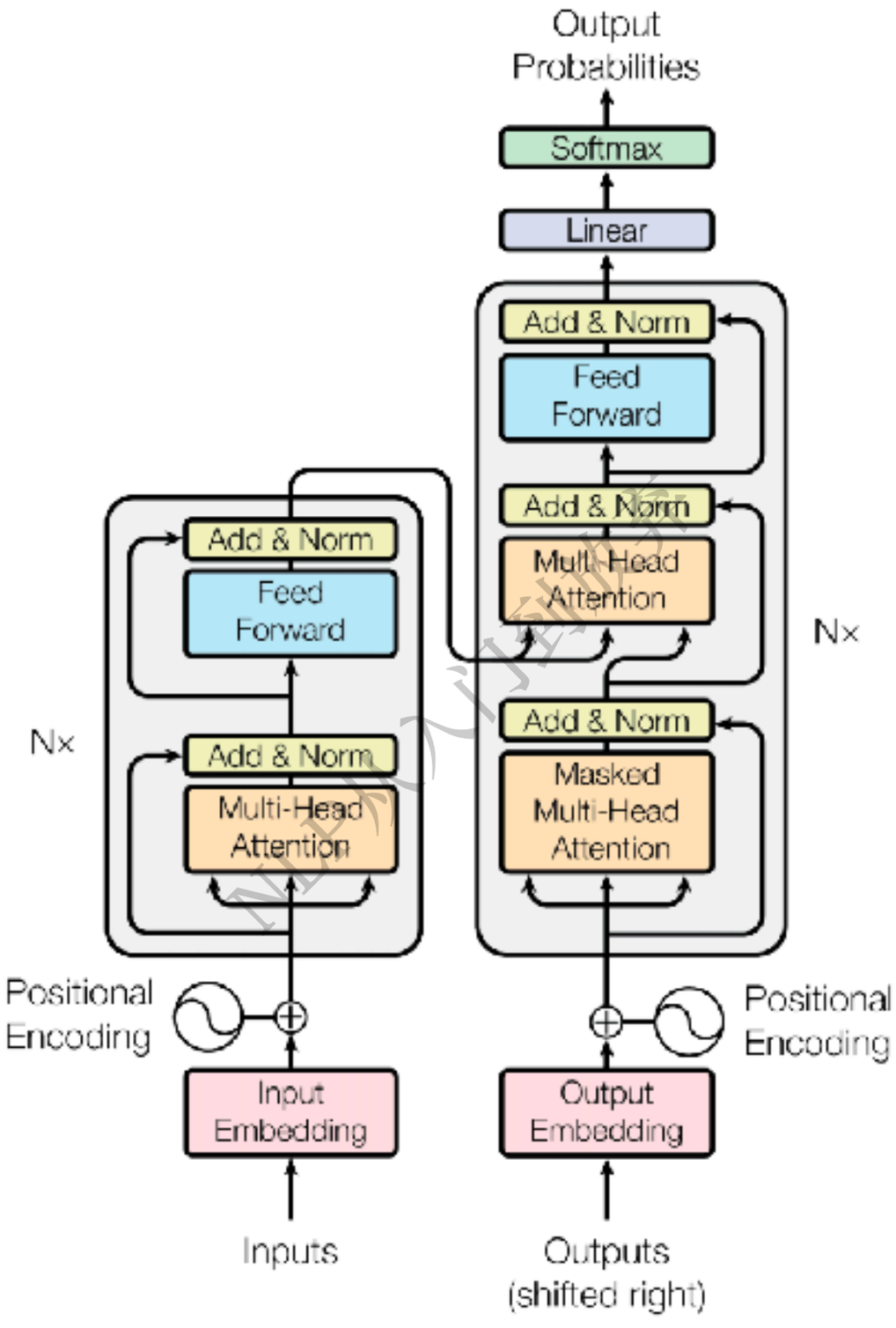
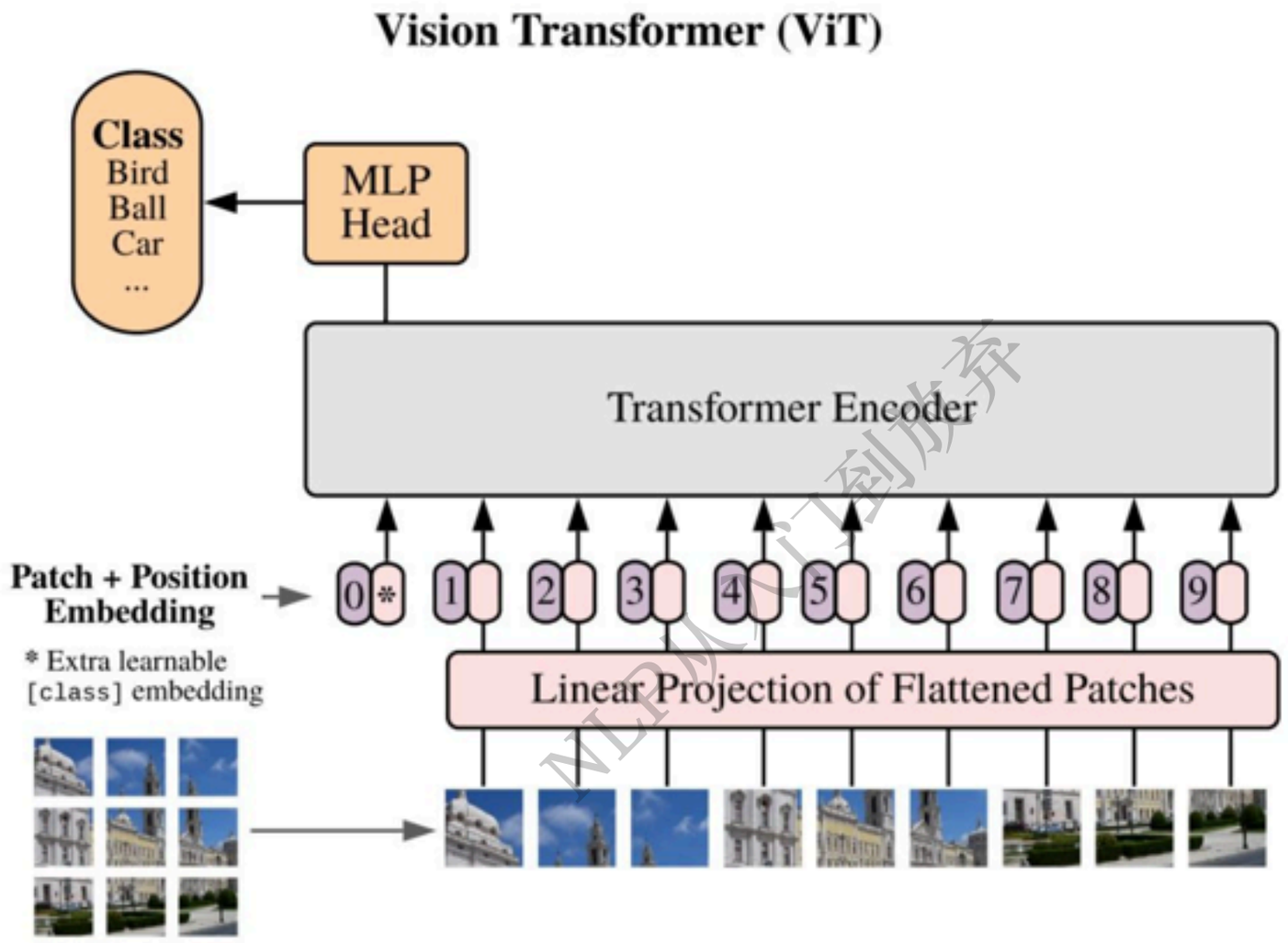


Figure 1: The Transformer - model architecture.

VIT中的位置编码：可学习的参数



1 2 3 4 5 6 7 8 9　　一维

【1，1】【1，2】 ……　　二维

这里一维和二维都是值得索引，通过索引从embedding获取向量

不同于原始Transformer的余弦位置信息（不可学习）和ViT中的一维位置信息（可学习），Swin Transformer的相对位置信息（可学习）是相对位置信息而非绝对

相对位置信息

SwinTRM位置编码有两点不同：

1. **加的位置不同：放在了att矩阵中**

ViT的位置信息是再embedding输入到Transformer前一步，但是这里作为B放在softmax前一步

2.**使用的是相对位置信息而不是绝对位置信息**

这里的B就是相对位置关系

最重要的相对位置编码讲解：

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T/\sqrt{d} + B)V, \quad (4)$$

QK转置相乘除以根号dk，然后加上相对位置编码

所以你需要知道QK转置相乘除以根号dk的输出形状是啥，你猜知道你的B是个啥形状

思考一下?

QK转置相乘除以根号dk的输出形状是啥?

softmax函数里面这个东西本质是计算每个字符对每个字符的相似性对吧

形状不就应该是下面这种吗?

Seq_len*seq_len

只有这种形状，才能遍历到每个单词对每个单词的形状

	卷	起	来	吧	
卷	20	5	4	9	softmax
起	5	30	8	12	softmax
来	4	8	15	14	softmax
吧	9	12	14	40	softmax

一个7*7的图像，每个像素点对每个像素点的相似性，形状不就应该是seq_le*seq_le

也就是49*49

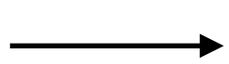
1，2，3，4，5，6，7

8，9，10，…

…

…

·，·，·，·，·，·，49



像素点1　像素点3

像素点2　像素点4　像素点49

像素点1	20	5	4	9	…	softmax
像素点2	5	30	8	12	…	softmax
像素点3	4	8	15	14	…	softmax
像素点4	9	12	14	40	…	softmax
…	…	…	…	…	…	
像素点49						

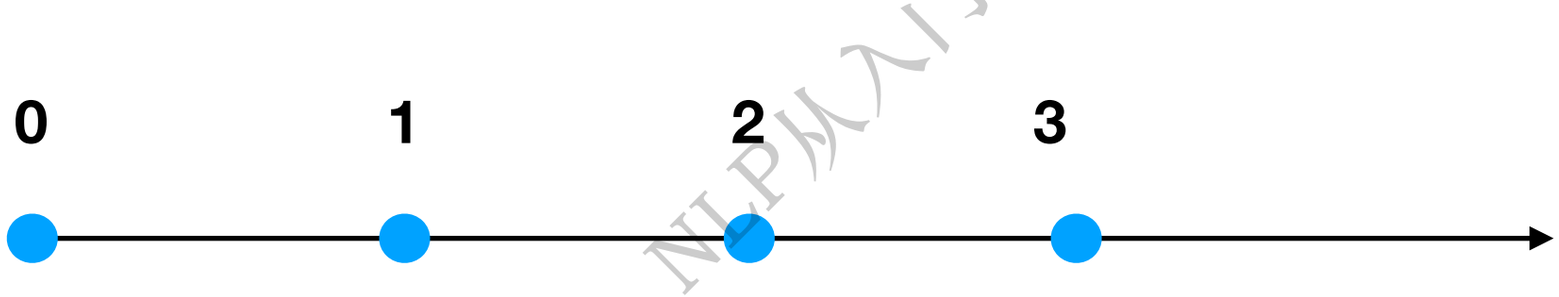
所以我的相对位置编码也必须是一个49*49的

7*7窗口对应的这个49*49太大了，我以winsize=2为例

winsize=2，attention矩阵就是：4*4对吧，在我接受范围之内啊

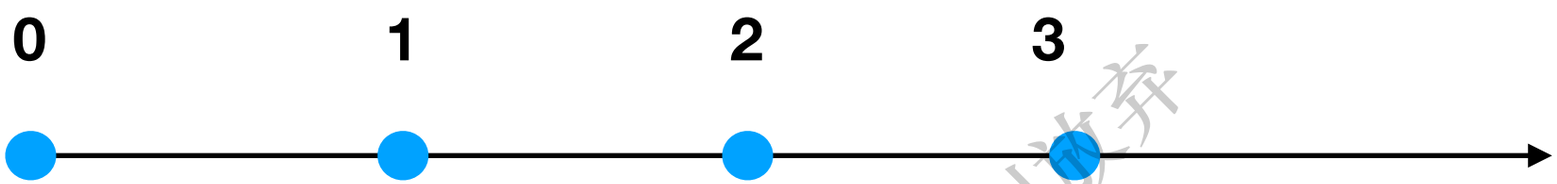
什么是绝对位置编码：一种绝对位置信息

绝对位置信息有很多种，就用最简单的例子来说啊

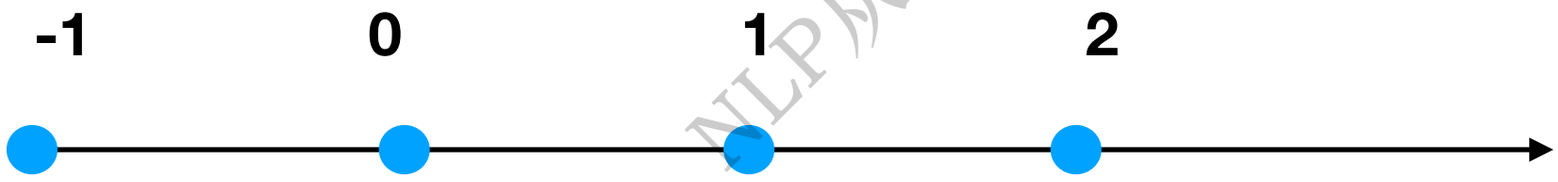


我的瞄点其实没动，我的原点这种位置没变啊

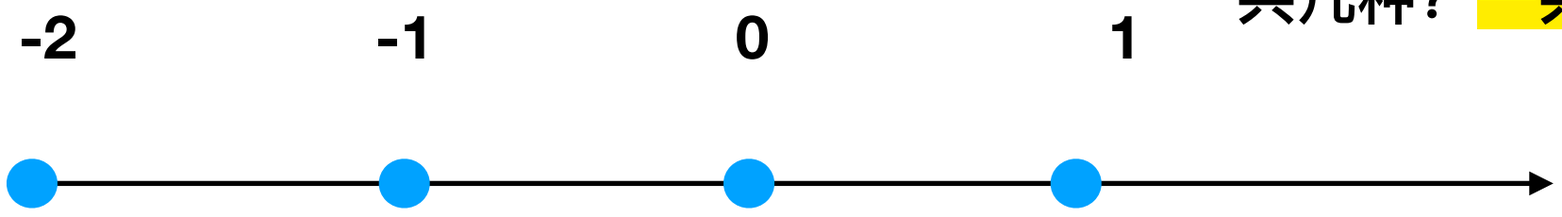
什么是相对位置信息

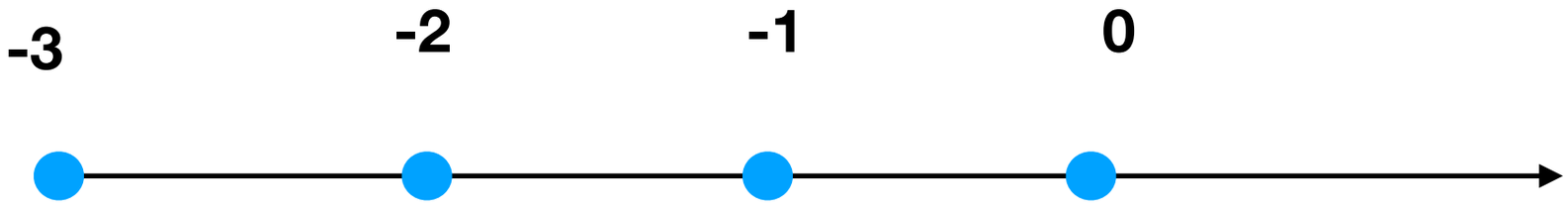


原点位置变了，相对坐标也就变了



一共几种？一共四种，四个不同原点位置





刚才是一个线段的绝对位置信息和相对位置，再看网格绝对

网格绝对位置

像素1	像素2
像素3	像素4

0	1
2	3

网格相对位置信息

像素1	像素2
像素3	像素4

0	1
2	3

-1	0
1	2

-2	-1
0	1

-3	-2
-1	0

怎么把四种相对位置信息融合起来，放入到我attention矩阵呢?

	像素1	像素2	像素3	像素4
像素1	20	5	4	9
像素2	5	30	8	12
像素3	4	8	15	14
像素4	9	12	14	40

0	1
2	3

-1	0
1	2

-2	-1
0	1

-3	-2
-1	0

	像素1	像素2	像素3	像素4	
像素1	20	5	4	9	以1为原点
像素2	5	30	8	12	以2为原点
像素3	4	8	15	14	以3为原点
像素4	9	12	14	40	以4为原点

将这里不同位置作为原点的相对信息拉平拼接在一起，每行代表以这一行表示的位置为原点，各个像素点的索引

	像素1	像素2	像素3	像素4
像素1	20	5	4	9
像素2	5	30	8	12
像素3	4	8	15	14
像素4	9	12	14	40

以1为原点

0	1
2	3

0	1	2	3

以2为原点

-1	0
1	2

-1	0	1	2

以3为原点

-2	-1
0	1

-2	-1	0	1

以4为原点

-3	-2
-1	0

-3	-2	-1	0

SwinTRM中的相对位置编码是怎么搞的呢？很简单，就是把我刚才讲的

位置信息全部变为二维

像素1	像素2
像素3	像素4

0，0	0，1
1，0	1，1

0，-1	0，0
1，-1	1，0

-1，0	-1，1
0，0	0，1

-1，-1	-1，0
0，-1	0，0

所以这里在代码中就是先分为x和y两个坐标信息的相对信息矩阵

0, 0	0, 1
1, 0	1, 1

0, -1	0, 0
1, -1	1, 0

0, 0	0, 1	1, 0	1, 1
0, -1	0, 0	1, -1	1, 0
-1, 0	-1, 1	0, 0	0, 1
-1, -1	-1, 0	0, -1	0, 0

-1, 0	-1, 1
0, 0	0, 1

-1, -1	-1, 0
0, -1	0, 0

因为Embedding索引只支持大于等于0的，这里所以需要加上M-1的偏置

同时为了区分诸如(0,1)和(1,0)索引位置上的区别，这里给0维乘以(2M-1)，区分x和y坐标的取值范围

0维度*(2M-1)

0, 0	0, 1	1, 0	1, 1
0, -1	0, 0	1, -1	1, 0
-1, 0	-1, 1	0, 0	0, 1
-1, -1	-1, 0	0, -1	0, 0

M-1

1, 1	1, 2	2, 1	2, 2
1, 0	1, 1	2, 0	2, 1
0, 1	0, 2	1, 1	1, 2
0, 0	0, 1	1, 0	1, 1

3, 1	3, 2	6, 1	6, 2
3, 0	3, 1	6, 0	6, 1
0, 1	0, 2	3, 1	3, 2
0, 0	0, 1	3, 0	3, 1

x+y

索引对应到参数

4	5	7	8
3	4	6	7
1	2	4	5
0	1	3	4

+

(2M-1)*(2M-1)

0.2	0.3	0.7	0.1	0.9	0.31	0.74	0.15

	像素1	像素2	像素3	像素4
像素1	20	5	4	9
像素2	5	30	8	12
像素3	4	8	15	14
像素4	9	12	14	40

1. 首先固定位置的二维坐标
window_size = (2,2)
coords_h = torch.arange(window_size[0])
coords_w = torch.arange(window_size[1])
coords = torch.meshgrid([coords_h, coords_w])
coords = torch.stack(coords)
coords_flatten = torch.flatten(coords, 1)
'''
x: [0, 0, 1, 1]
y: [0, 1, 0, 1]
'''

2. 计算相对位置坐标
relative_coords_first = coords_flatten[:, :, None]
relative_coords_second = coords_flatten[:, None, :]
relative_coords = relative_coords_first - relative_coords_second
'''
分别对coords_flatten的x和y轴在最后和中间添加一个维度，再进行广播和相减：
1. 首先进行广播：
first:　　second:
x：
[[0, 0, 0, 0],　　[[0, 0, 1, 1],
 [0, 0, 0, 0],　　 [0, 0, 1, 1],
 [1, 1, 1, 1],　　 [0, 0, 1, 1],
 [1, 1, 1, 1]],　　 [0, 0, 1, 1]],
y：
[[0, 0, 0, 0],　　[[0, 1, 0, 1],
 [1, 1, 1, 1],　　 [0, 1, 0, 1],
 [0, 0, 0, 0],　　 [0, 1, 0, 1],
 [1, 1, 1, 1]]　　 [0, 1, 0, 1]]
2. 然后相减，可以看作原坐标减去原每个的像素的坐标作为远点：
比如以[0,1]作为原点，这样所有的在[0,0],[0,1],[1,0],[1,1]都要减去[0,1]
x：
[[0, 0, -1, -1],
 [0, 0, -1, -1],
 [1, 1, 0, 0],
 [1, 1, 0, 0]],
y:
[[0, -1, 0, -1],
 [1, 0, 1, 0],
 [0, -1, 0, -1],
 [1, 0, 1, 0]]

'''

3. 然后维度转换后，加上偏置，确保坐标大于0：
relative_coords = relative_coords.permute(1, 2, 0).contiguous()
relative_coords[:, :, 0] += window_size[0] - 1
relative_coords[:, :, 1] += window_size[1] - 1
'''
　　[[1, 1],
　　 [1, 0],
　　 [0, 1],
　　 [0, 0]],

　　[[1, 2],
　　 [1, 1],
　　 [0, 2],
　　 [0, 1]],

　　[[2, 1],
　　 [2, 0],
　　 [1, 1],
　　 [1, 0]],

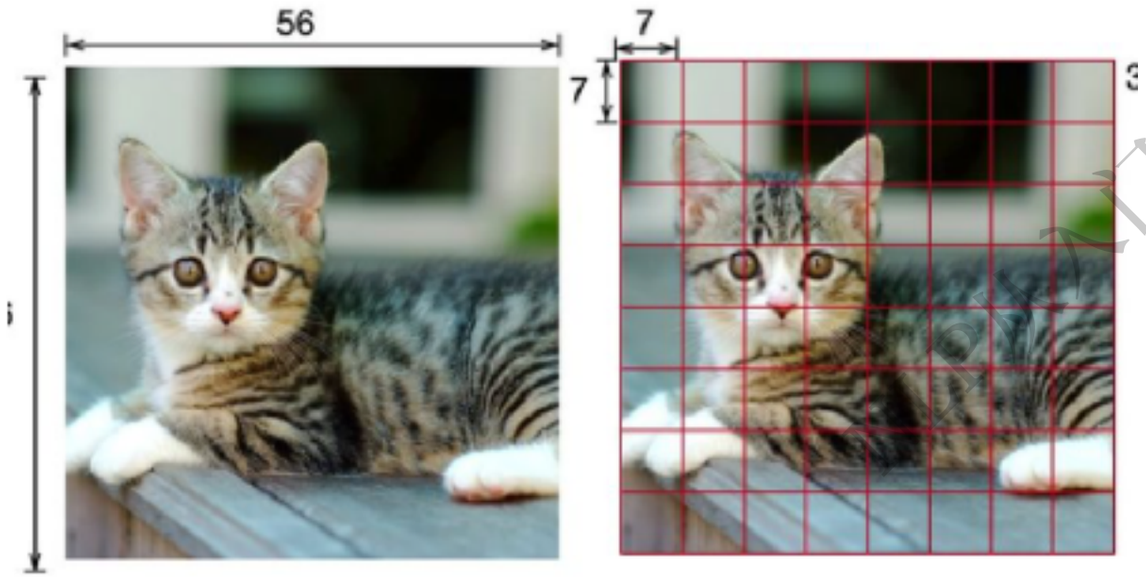
　　[[2, 2],
　　 [2, 1],
　　 [1, 2],
　　 [1, 1]]]
'''
4. 最后0维度乘以2M-1确保x和y坐标取值不一样，区分诸如[0,1]和[1,0]的问题
'''
　　[[3, 1],
　　 [3, 0],
　　 [0, 1],
　　 [0, 0]],

　　[[3, 2],
　　 [3, 1],
　　 [0, 2],
　　 [0, 1]],

　　[[6, 1],
　　 [6, 0],
　　 [3, 1],
　　 [3, 0]],

　　[[6, 2],
　　 [6, 1],
　　 [3, 2],
　　 [3, 1]]
'''

窗口注意力机制

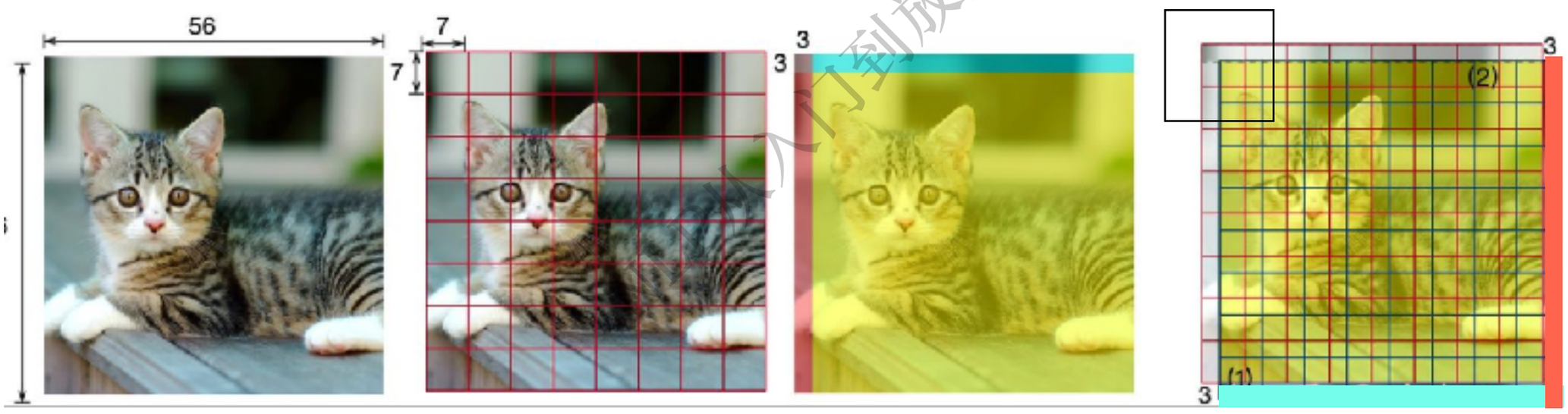


7*7的win直接输入TRM就可以

存在的一问题：窗口之间没有交互

但是我是这么理解的，没有交互只能说是在当前视野没有，
下一个阶段其实干感受野变大了，现在的窗口之间其实有交互，
但是在下一个阶段的窗口之间有没有交互了；所以为了在当前阶段就有交互，做了一个移动窗口注意力

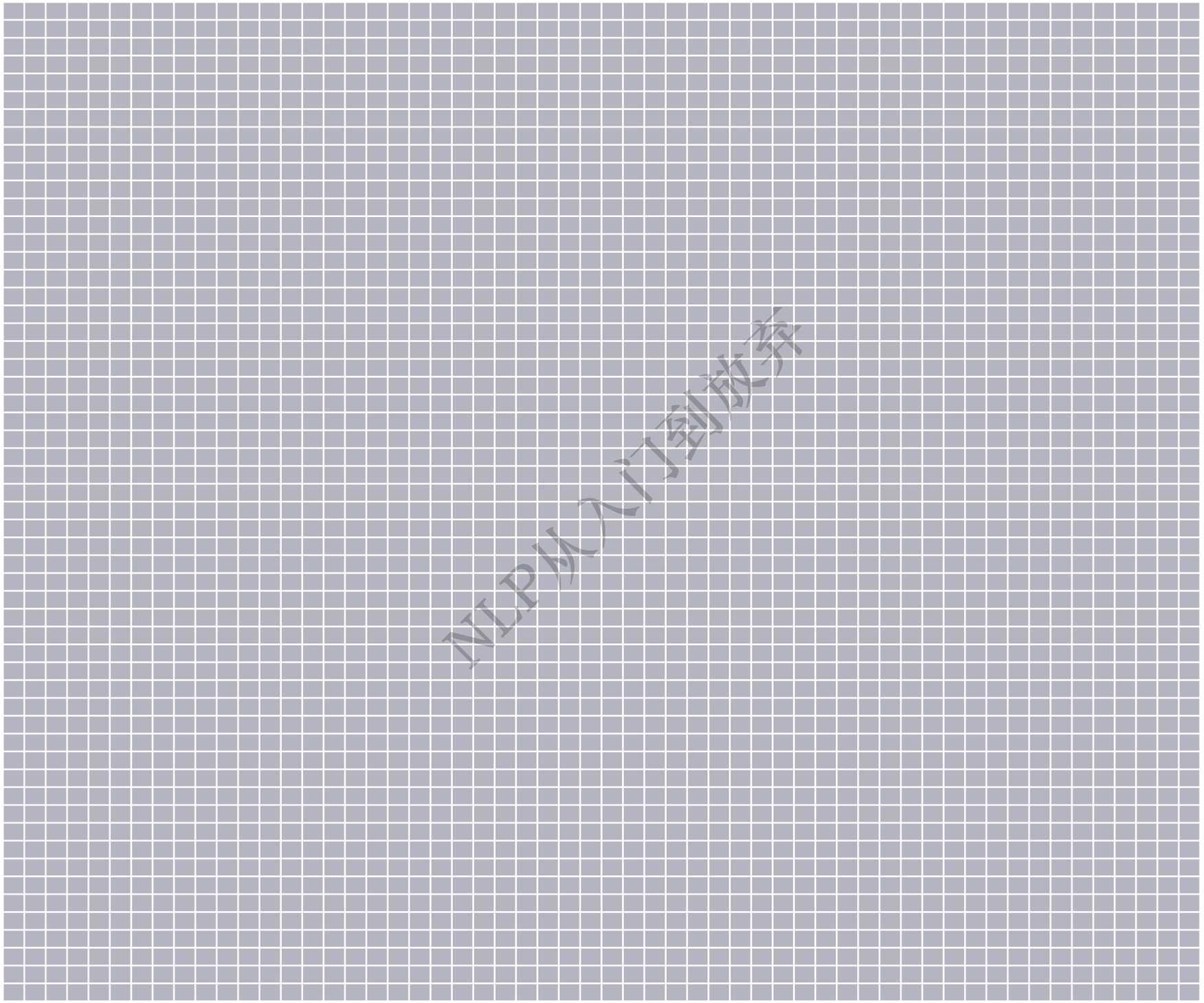
这里黄底框中的蓝框可以看出是灰底红框shift得到的
实现红框之间的像素attention



但是问题是计算窗口att的时候，有的本不相邻的也会被计算：mask的重要性就出来了

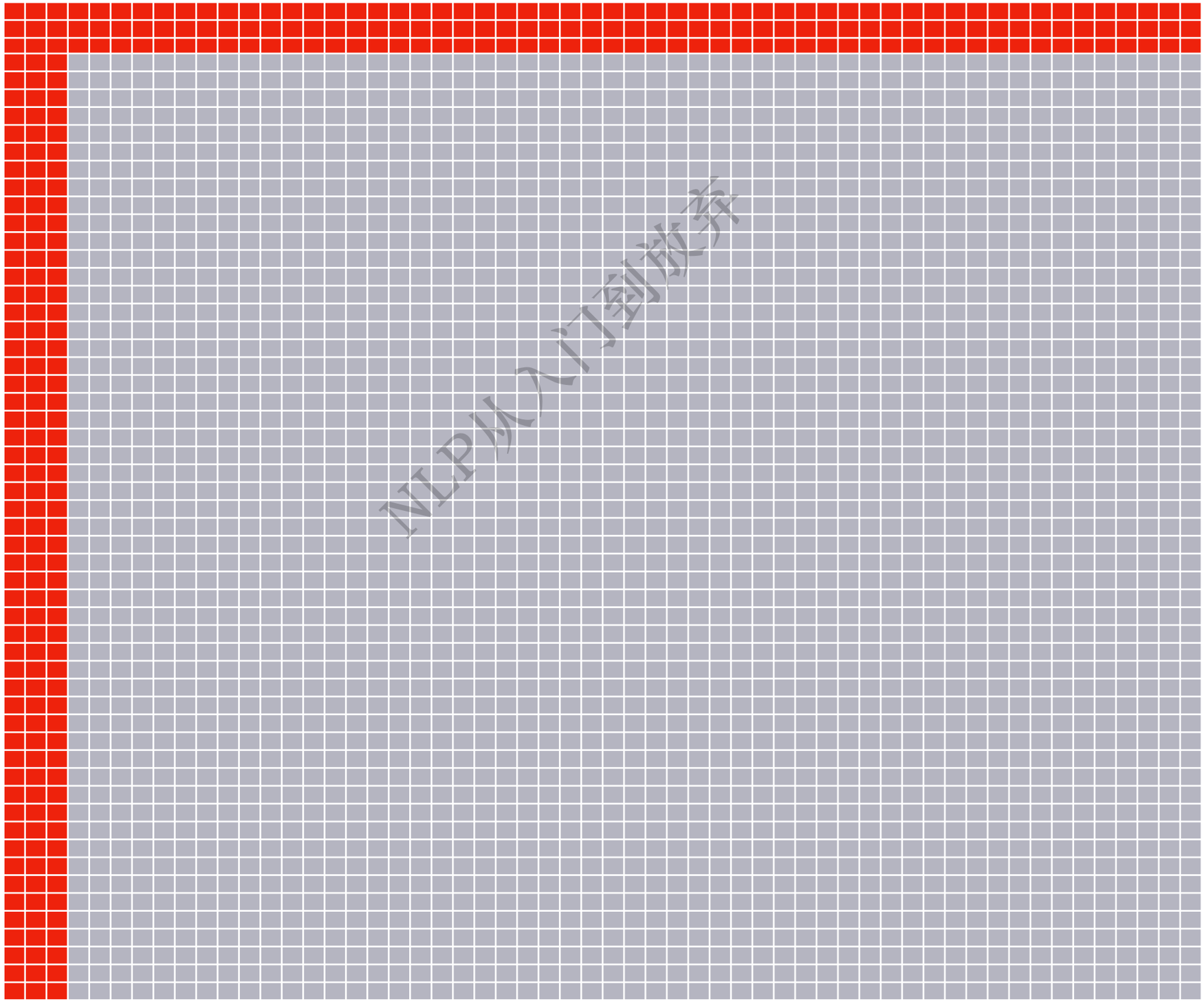
举个例子，我们从头说

56

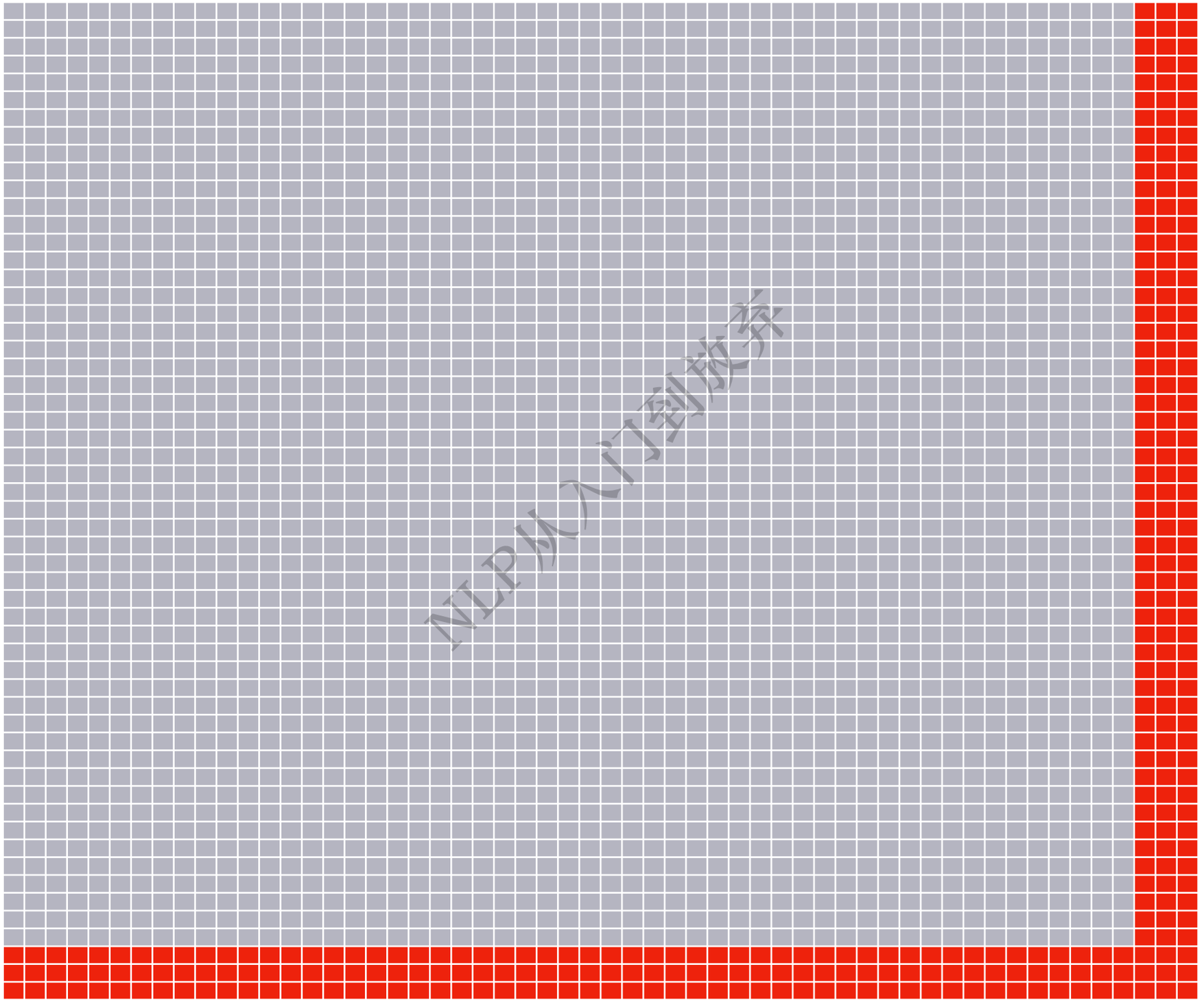


三行元素移动过去

56

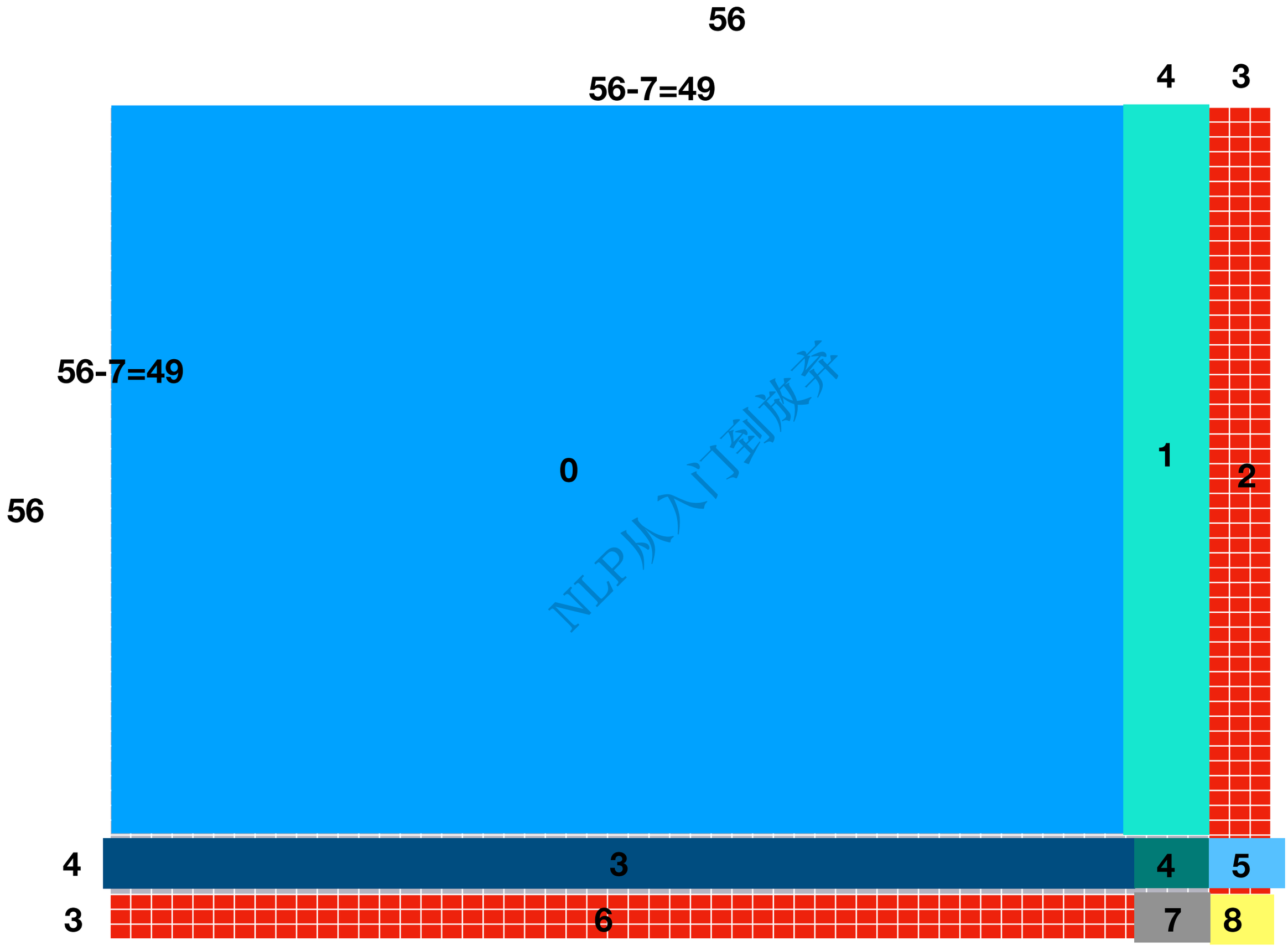


56



这里写错了，dims=0

有相邻的，有不相邻的，我们编上窗口索引


56
56-7=49
4
3
56-7=49
1
0
2
56
NLP从入门到放弃
4
3
4
5
3
6
7
8

56

56-7=49

4 3

7

7

56-7=49

56

0

1 2

4 3 4 5

3 6 7 8

7

这边是5不是2，
注释一下

7

4	4	4	4	2	2	2
4	4	4	4	2	2	2
4	4	4	4	2	2	2
4	4	4	4	2	2	2
6	6	6	6	8	8	8
6	6	6	6	8	8	8
6	6	6	6	8	8	8

每次shift之后，都会再shift回去，还原原先的坐标

4	2
6	8

	4	2	6	8
4	**0**	**-2**	**2**	**4**
2	**2**	**0**	**4**	**6**
6	**-2**	**-4**	**0**	**2**
8	**-4**	**-6**	**-2**	**0**

其他几个位置的mask情况

一个标签序号的保留，其他的mask掉

这里给非0值赋值很小值（比如-100），可以在softmax避免非本区域之间进行softmax

	4	2	6	8
4	0	-100	-100	-100
2	-100	0	-100	-100
6	-100	-100	0	-100
8	-100	-100	-100	0

	4	2	6	8
4	20	5	4	9
2	5	30	8	12
6	4	8	15	14
8	9	12	14	40

4	2	6	8

4	0	-100	-100	-100
2	-100	0	-100	-100
6	-100	-100	0	-100
8	-100	-100	-100	0

7

7

4	4	4	4	2	2	2
4	4	4	4	2	2	2
4	4	4	4	2	2	2
4	4	4	4	2	2	2
6	6	6	6	8	8	8
6	6	6	6	8	8	8
6	6	6	6	8	8	8

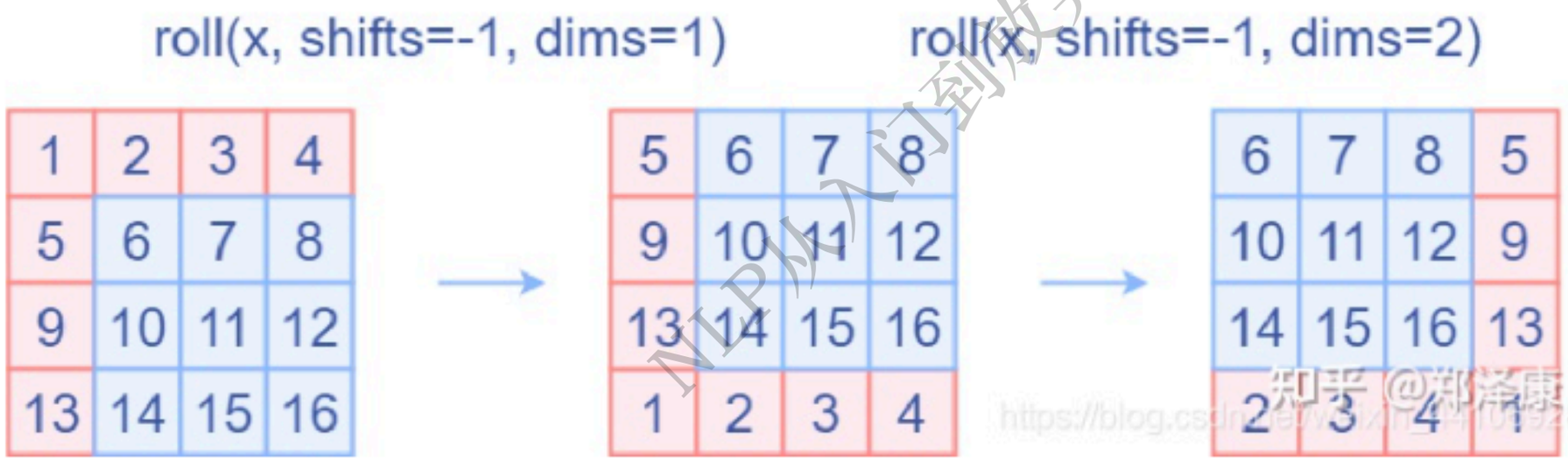
4 4 4 4 2 2 2 4 4 4 4 2 2 2 4 4 4 4 2 2 2 4 4 4 4 2 2 2 6 6 6 6 8 8 8 6 6 6 6 8 8 8 6 6 6 6 8 8 8

两个tensor相减

那三行元素究竟是怎么转的

注意swin transformer只有偶数行是shifted transformer，奇数行不用转

代码里对特征图移位是通过 `torch.roll` 来实现的，下面是示意图



如果是一个2乘以2 的窗口，问题在于8和5是没办法attention，他们没关系
但是8和12是在一个窗口的

复杂度的问题

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C, \tag{1}$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC, \tag{2}$$

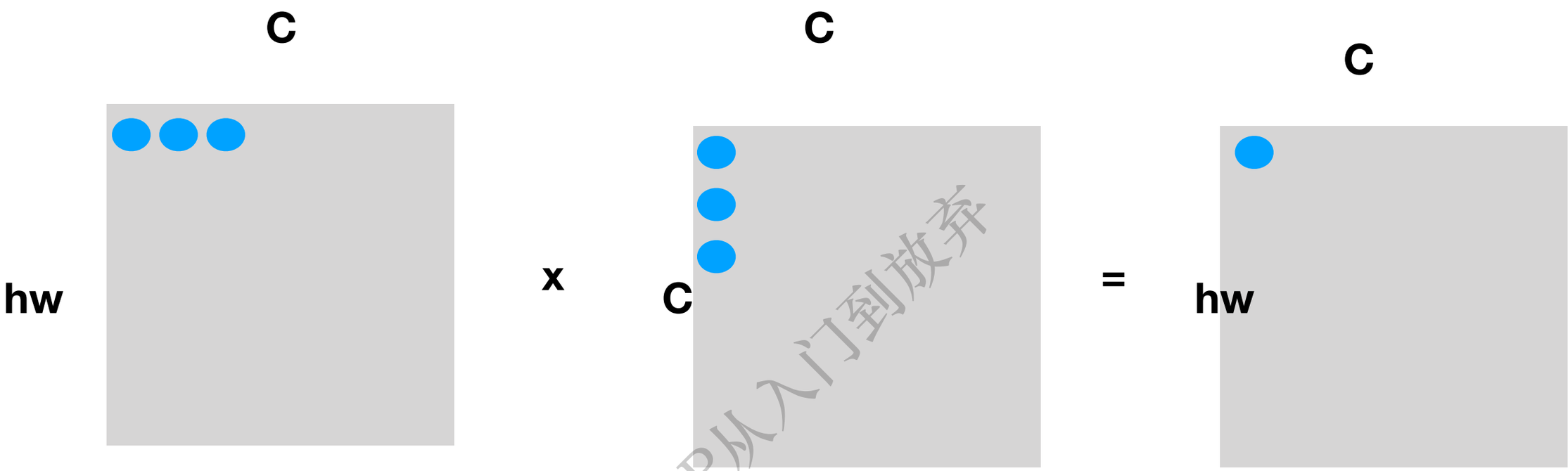
$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C, \tag{1}$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC, \tag{2}$$

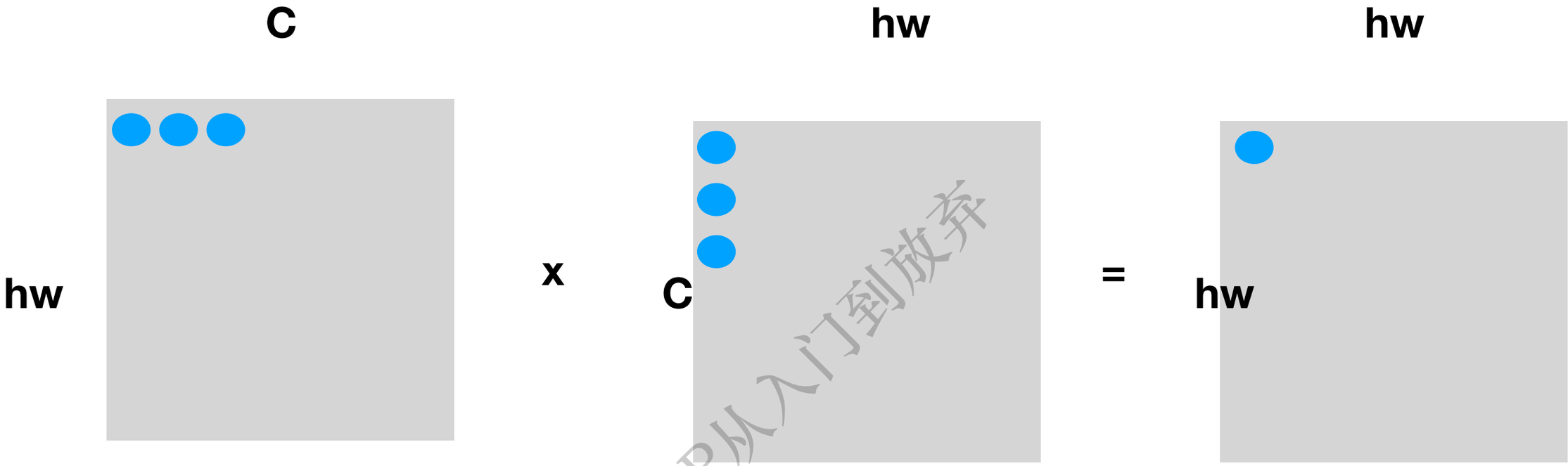
1. 代码中的 `to_qkv()` 函数，即用于生成 Q, K, V 三个特征向量：其中 $Q = x \times W^Q, K = x \times W^K, V = x \times W^V$ 。x 的维度是 (hw, C) ， W 的维度是 (C, C) ，那么这三项的复杂度是 $3hwC^2$ ；
2. 计算 QK^T ： Q, K, V 的维度均是 (hw, C) ，因此它的复杂度是 $(hw)^2C$ ；
3. softmax之后乘 V 得到 Z ：因为 QK^T 的维度是 (hw, hw) ，所以它的复杂度是 $(hw)^2C$ ；
4. Z 乘 W^Z 矩阵得到最终输出，对应代码中的 `to_out()` 函数：它的复杂度是 hwC^2 。

---

hw是长度，C是每个token的维度



hwC^2



v 矩阵

Z

hw

C

C

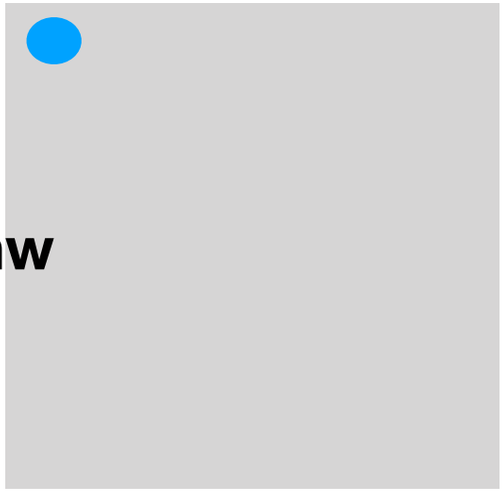
hw

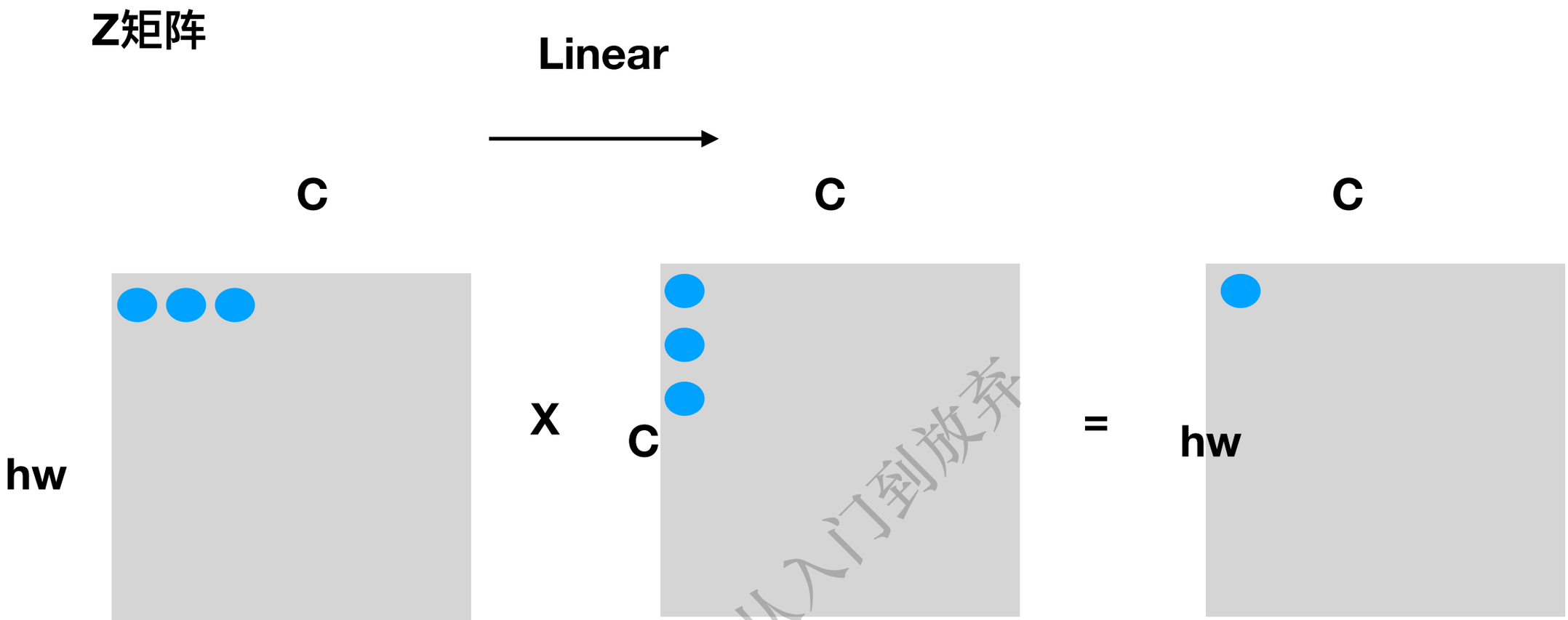
x

hw

=

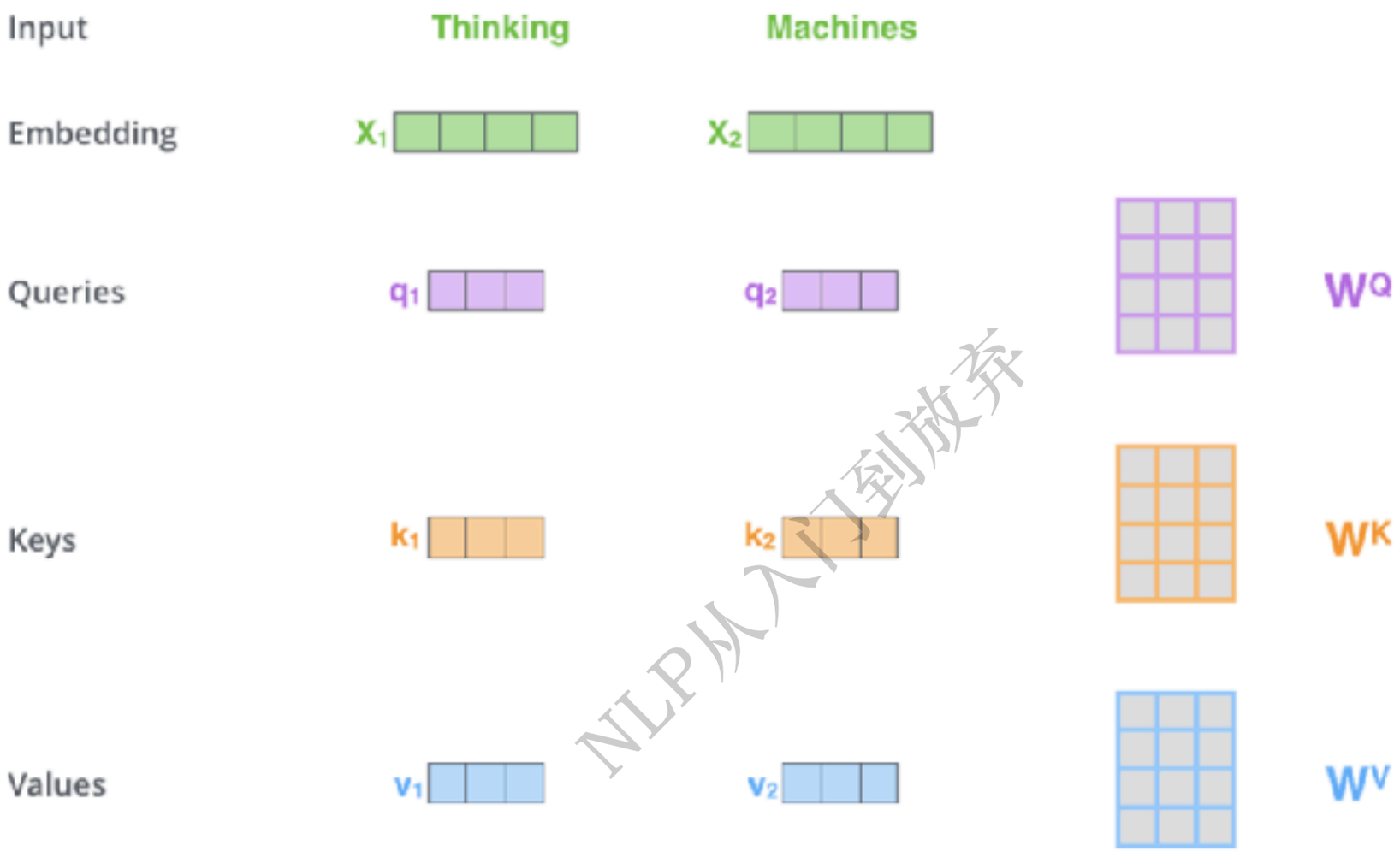
hw





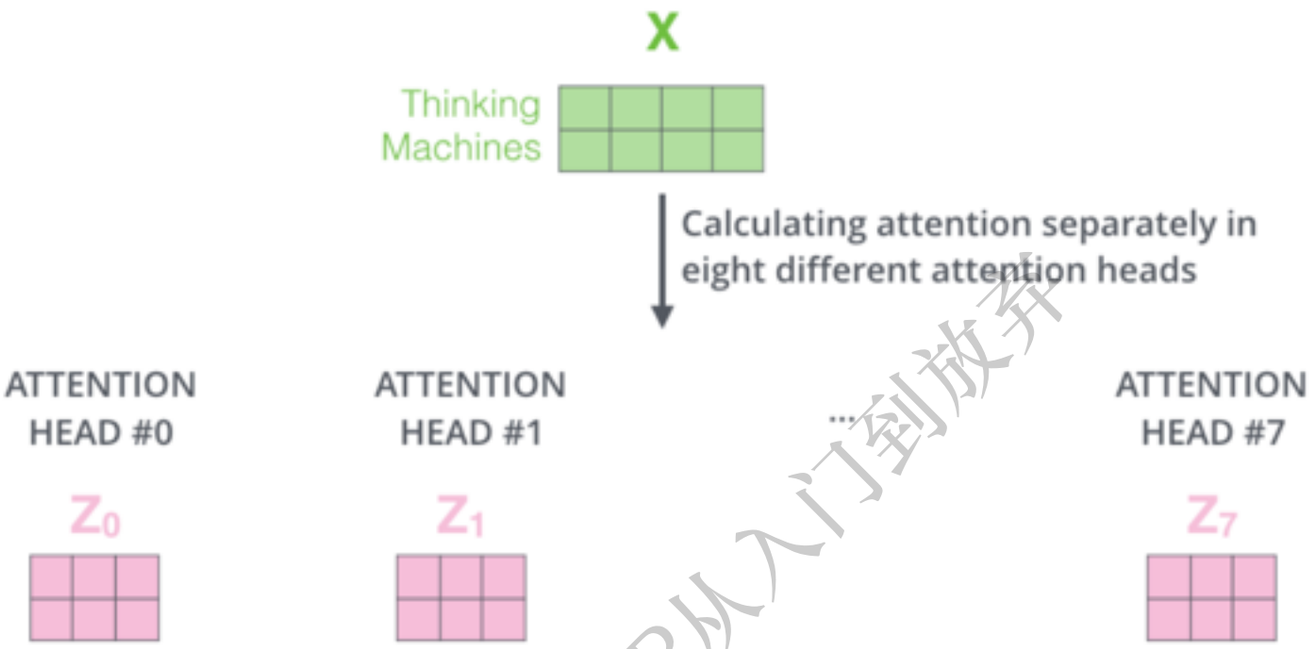
Z矩阵的最后的这个映射作用是什么？有的TRM代码实现可没这个步骤

transformer encoder需要确保输入和输出维度不变，这里所以需要一个线性层确保不变



一个头，四维度到了三维度

多头就是多个三维度拼在一起



TRM本身不改变形状的，不然不能堆叠，对从多个三拼接再到4维度就好了

窗口注意力机制

$$\Omega(\mathrm{MSA}) = 4hwC^2 + 2(hw)^2C$$

一个win

$$= 4\boxed{MM}C^2 + 2(\boxed{MM})^2C$$

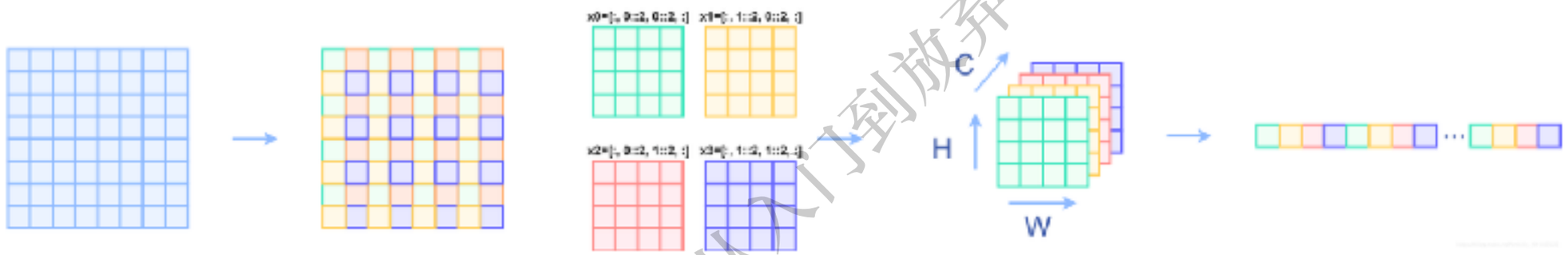
h/M *w/M个win

(h/M)*(w/M)

$$* \; 4\boxed{MM}C^2 + 2(\boxed{MM})^2C$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC, \qquad (2)$$

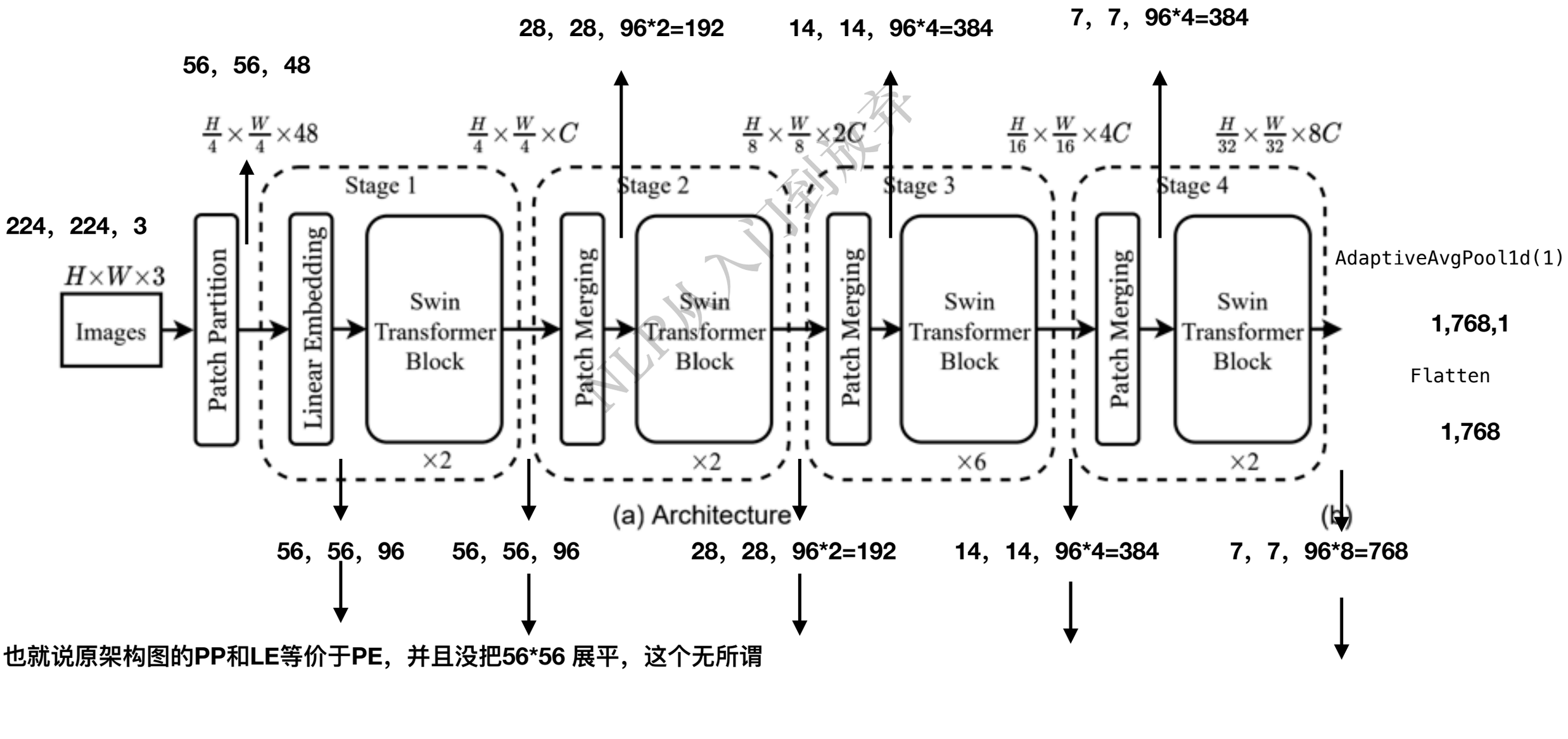
PatchMerging如何实现的：降采样



在行方向和列方向，间隔取2，通道维度会变成原先的4倍，接一个linear转为两倍

整体梳理一遍

H是224，W是224



也就说原架构图的PP和LE等价于PE，并且没把56*56 展平，这个无所谓