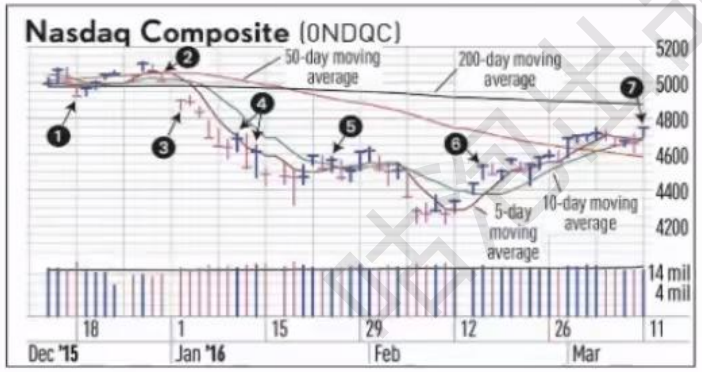
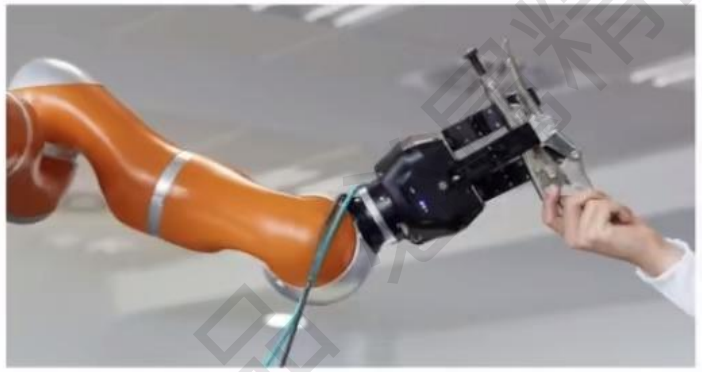
Informer

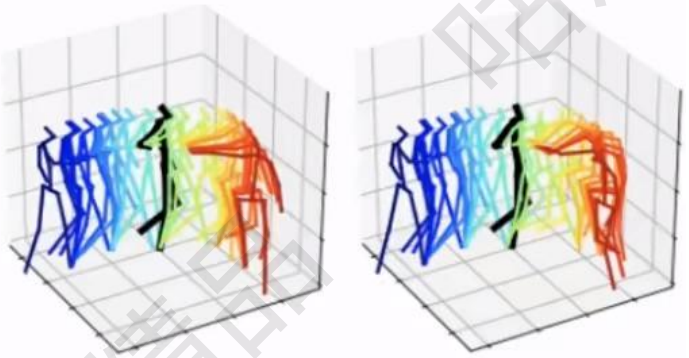
- 时间序列预测



Stock Market Prediction



Robots Action Prediction



Human Position Prediction

Weather Prediction

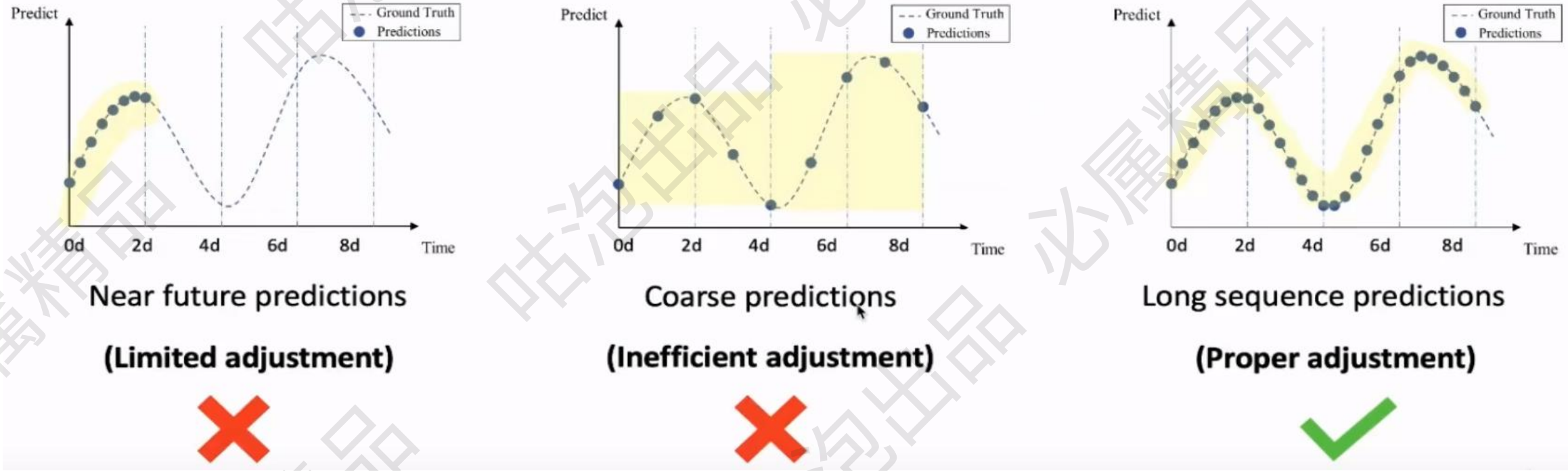
Supply Chain Prediction

Society Event Prediction

Informer

- 论文背景研究问题：

  - 1.短序列预测；2.趋势预测；3.精准长序列预测



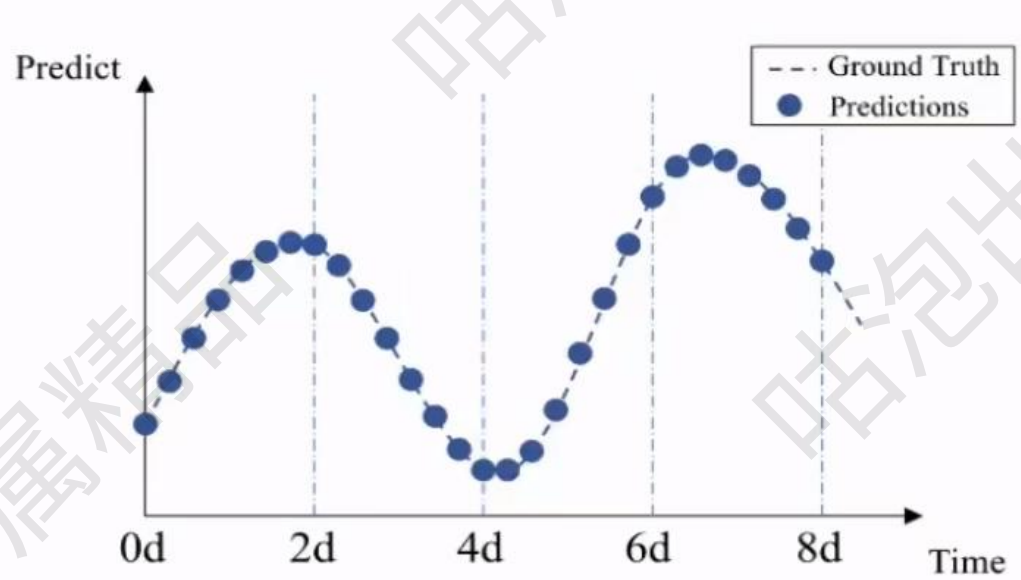
Informer

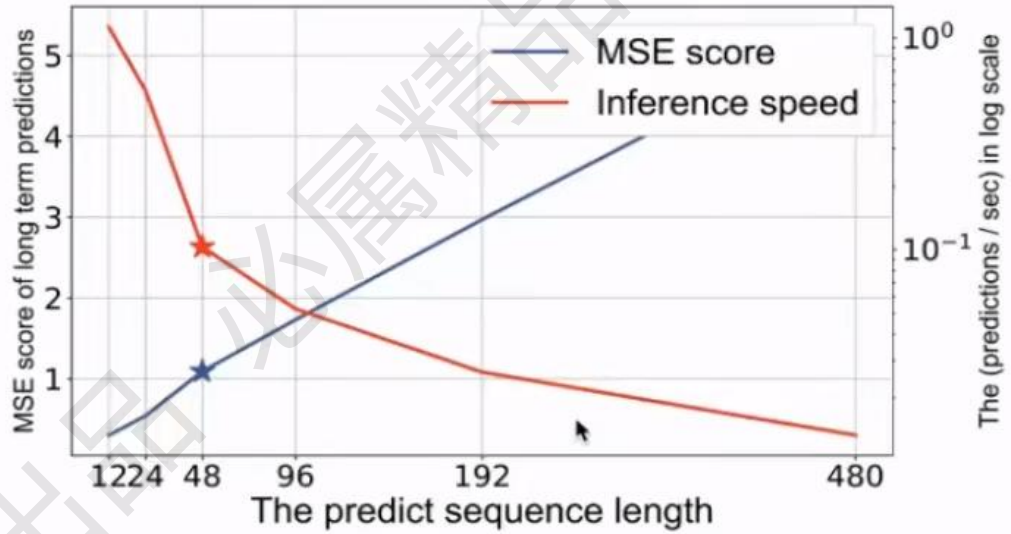
- 时间序列经典算法：
  - Prophet：非常实用的工具包，适合预测趋势，但不算精准
  - Arima：老牌算法了，短序列预测还算精准，但是趋势预测不准
  - 但是一旦涉及到长序列，他俩可能就都GG了
  - Informer中将主要致力于长序列问题的解决

Informer

- 你肯定想到了LSTM

- 在长序列预测中，如果序列越长，那速度肯定越慢，效果也越差



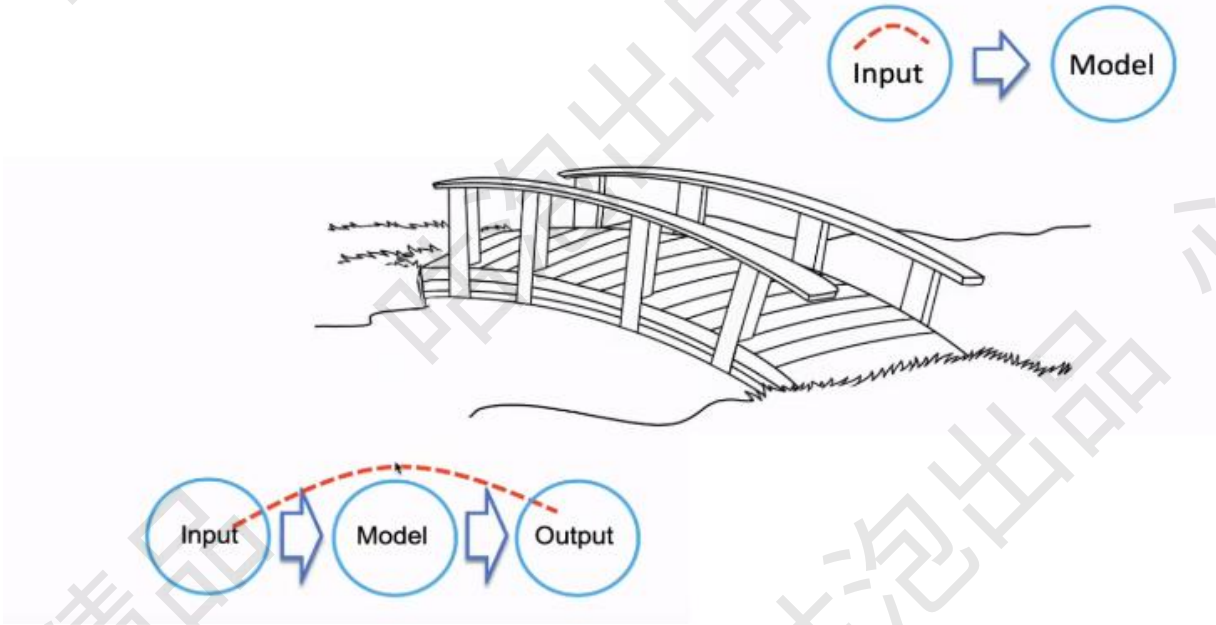


Informer

- 算法的核心思想

  - 看名字估计大家也能才出来了，肯定是套transformer架构

  - 建立好长输入（input）和长输出（output）之间的关系



Informer

- 传统transformer

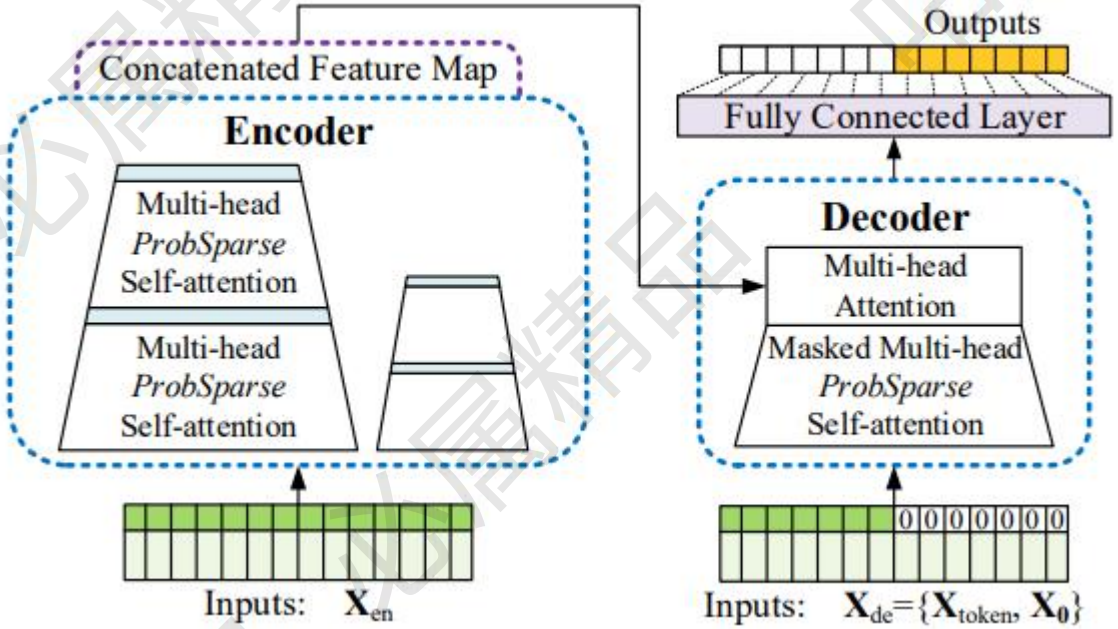
- 回顾一下经典QKV计算方法

Informer

- Transformer架构的优势与问题
  - 1.万能模型，直接套用，代码实现简单，现成例子一大片
  - 2.并行的，比LSTM快，全局信息丰富，注意力机制效果好
  - 3.长序列中attention需要每一个点跟其他点计算（如果序列太长，效率很低）
  - 4.Decoder输出挺墨迹的，要基于上一个预测结果来推断当前的预测结果

Informer

- 要解决的三大问题
  - 1.Attention要算的更快
  - 2.Decoder要一次性输出所有预测
  - 3.堆叠encoder也得要更快
  - 论文的三大核心模块

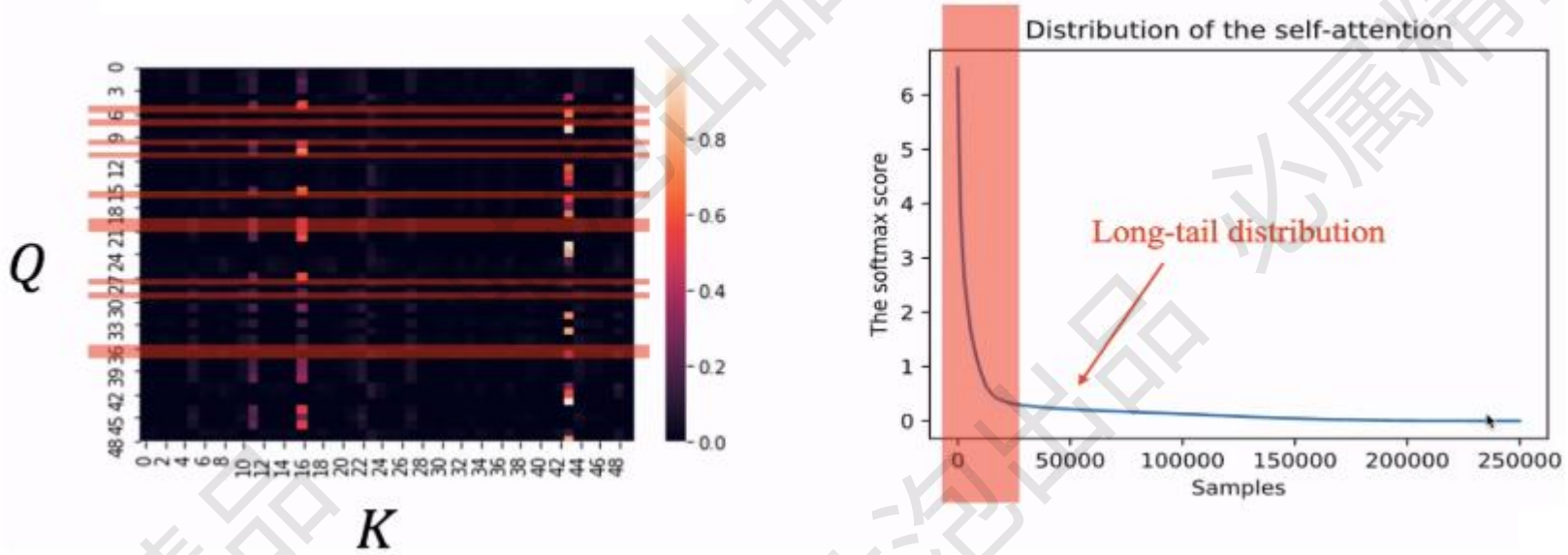


Informer

- Attention计算

  - 在长序列中，每一个位置的attention都很重要吗?

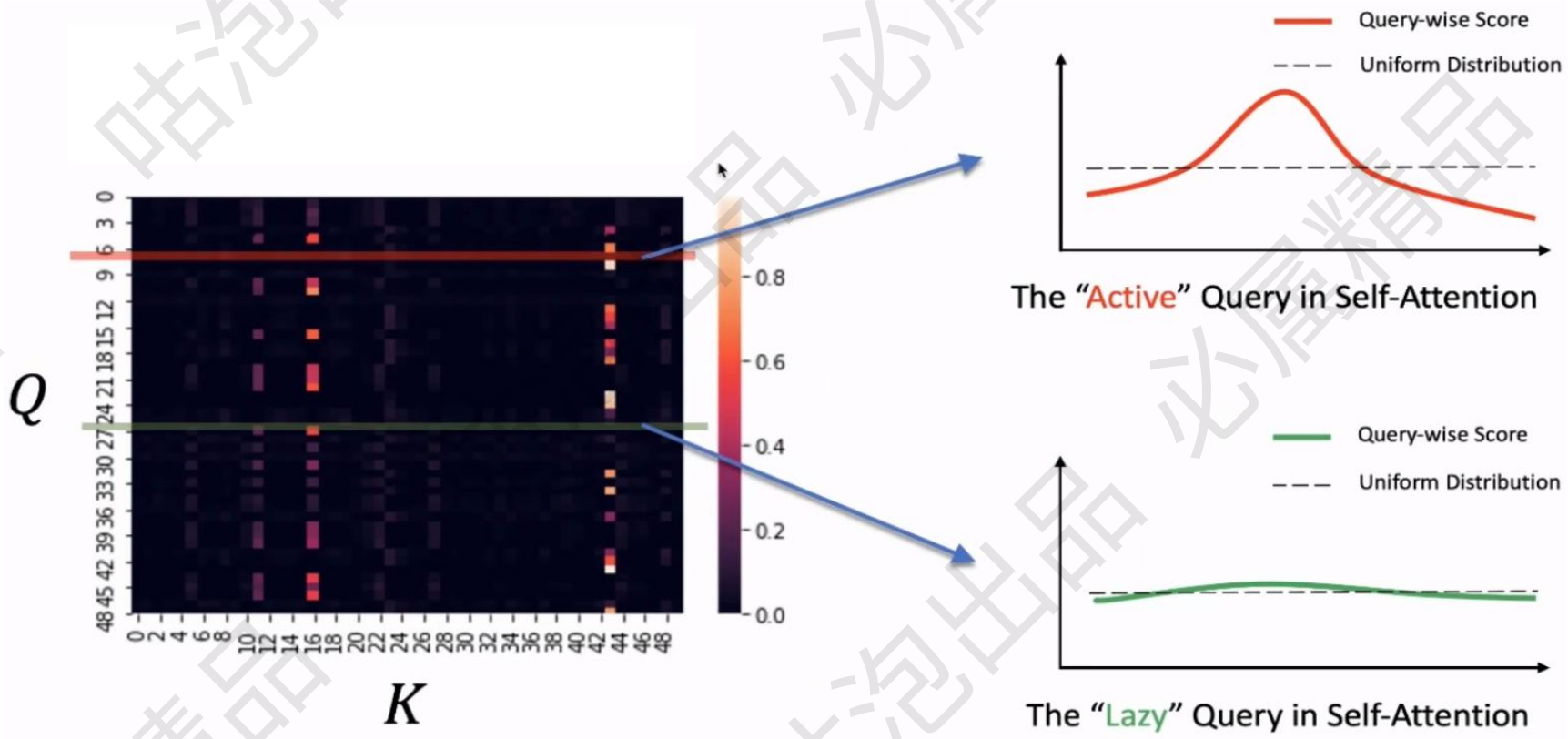
  - 对于每一个Q来说，只有一小部分的K是其它有较强关系



Informer

- 长序列中要不要进行采样呢?

- 群众里有坏人（有偷懒不干活的Q）



Informer

- 如何定义每一个Q是不是偷懒的
  - 偷懒的Q感觉就像是均匀分布，没啥特点，你有我有全都有
  - Active的Q明显在某些位置比较活跃，权重差异较大
  - 对于每一个Q，计算其与均匀分布的差异，差异越大则表示其越活越

$$M(\mathbf{q}_i, \mathbf{K}) = \ln \sum_{j=1}^{L_K} e^{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}$$

Informer

- ProbAttention计算方法
  - 输入序列长度为96，首先在K中进行采样，随机选25个K
  - 现在要选出来的是一些重要的Q，正常情况需每一个Q跟96个K计算
  - 重要的Q不用非得计算那么多，跟部分K计算的结果也可以当作其分布
  - 例如源码输出结果：32, 8, 96, 25表示8头，96个Q分别跟25个K计算的内积

Informer

- ProbAttention计算方法
  - 现在每一个Q有25个得分（分别跟25个K计算后得到的）

$$\overline{M}(\mathbf{q}_i, \mathbf{K}) = \max_j \{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}\} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}$$

  - 论文中做法比较绝，为了进一步加速，直接选最大值与均匀分布算差异
  - 在96个Q中，选出来差异最大的25个（根据序列长度来定的一个参数值）

Informer

- ProbAttention计算方法
  - 得到的QK内积为：32, 8, 25, 96，就是只选了25个Q
  - 那么其它位置的Q该咋办呢？没有计算其attention目前
  - 直接用V（96个，表示每一个位置的特征）的均值来替代
  - 也就是选出来的25个会更新，其他剩余的都是均值向量

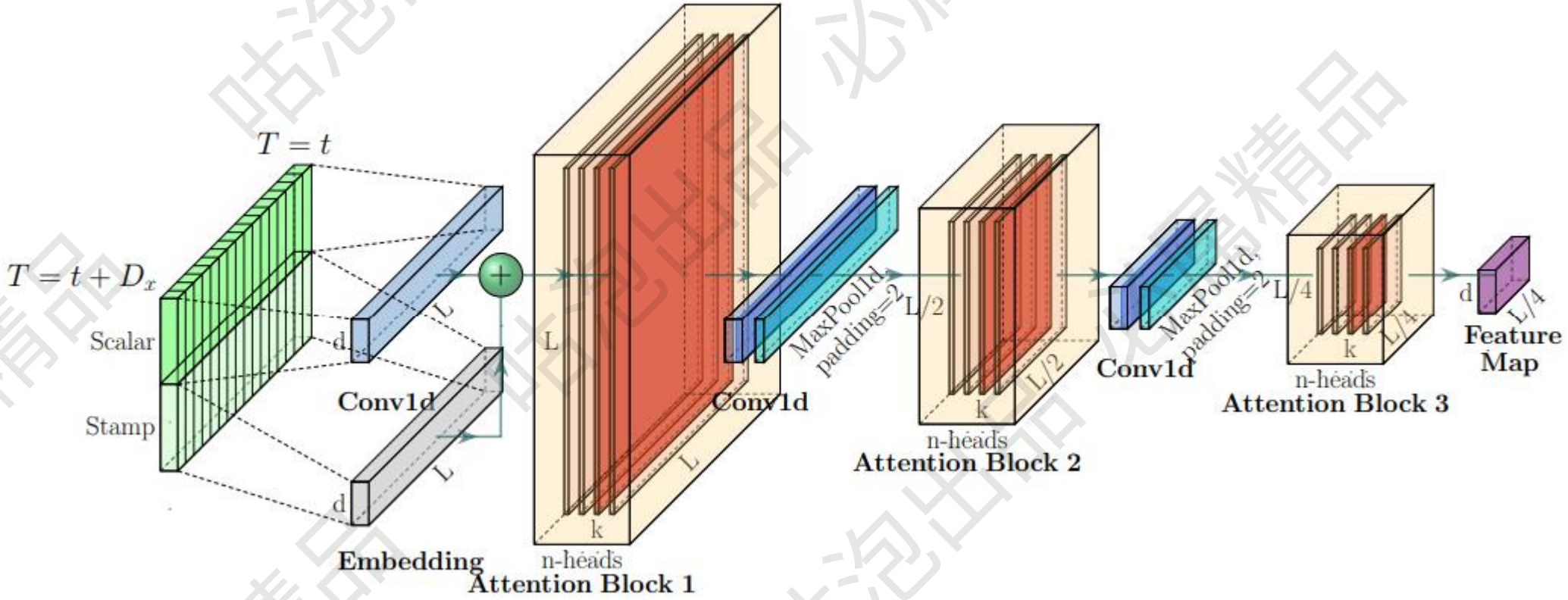
Informer

- Self-attention Distilling计算方法
  - 做完一次attention之后还要继续堆叠，只不过与传统transformer不同
  - 会先通过1D的maxpool操作来进行下采样，下次输入序列就为48了
  - 此时Q和K的采样由于序列长度变小，也会随之变小，例如由25->20
  - 重复堆叠多次就是Informer的Encoder架构了

Informer

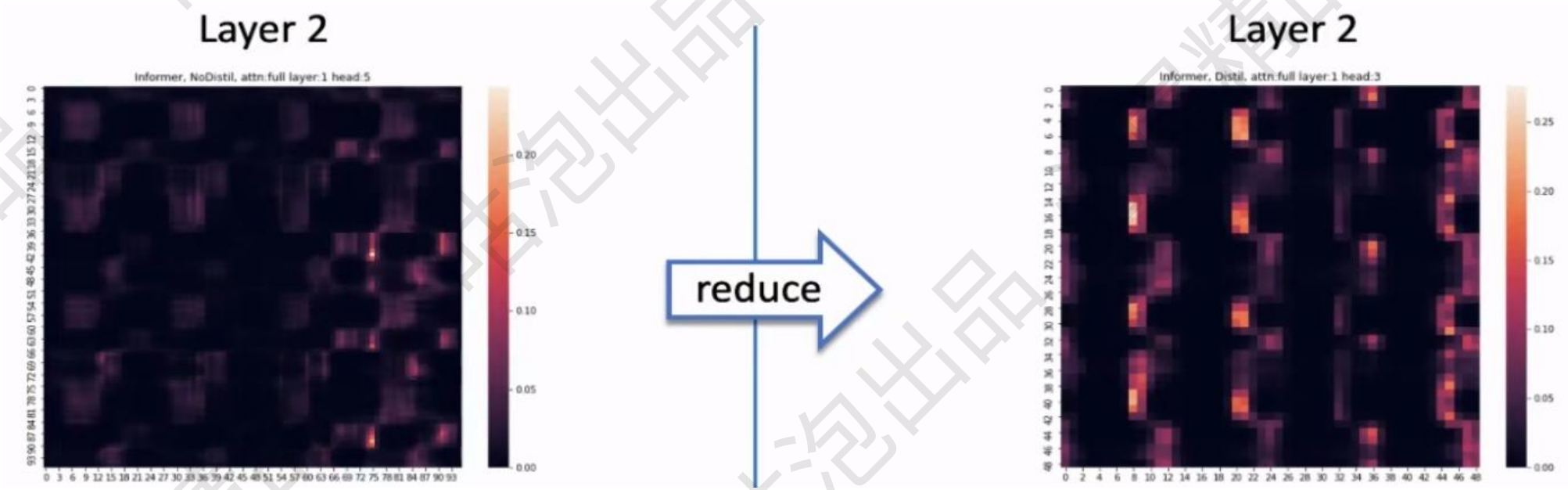
- Self-attention Distilling计算方法

  - 不仅是下采样，还把stamp特征也融合进来了（看来时间特征也很重要）



Informer

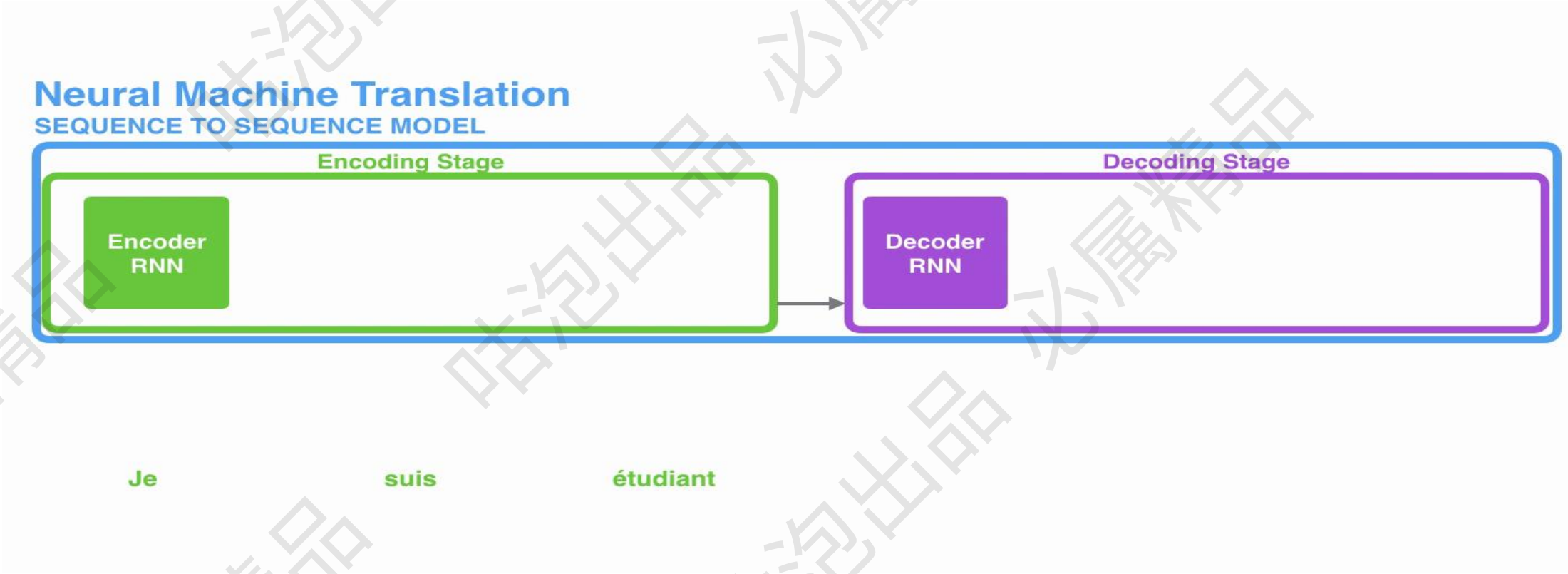
- Encoder改进后的效果
  - 一方面就是速度快效率高了，论文中计算复杂度由L^2->LlogL
  - 下采样之后，特征更明显，且跟之前的模式基本一致



Informer

- 传统Decoder输出

  - 先输出第一个，在基于第一个输出第二个，以此类推



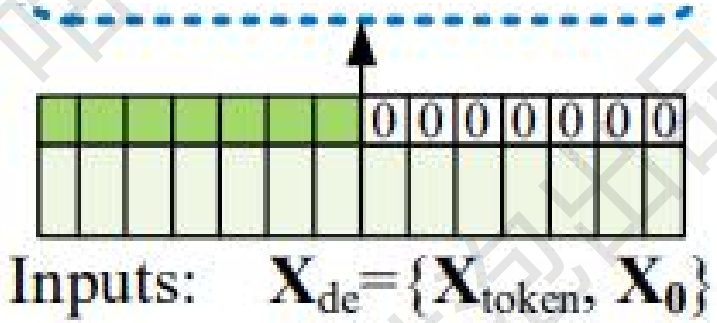
Informer

- Start标志位

  - 要让Decoder输出预测结果，你得先告诉它从哪开始输出

  - 先给一个引导，比如要输出20-30号的预测结果，Decoder中需先给出

  - 前面一个序列的结果，例如10-20号的标签值（下图0是待预测结果）

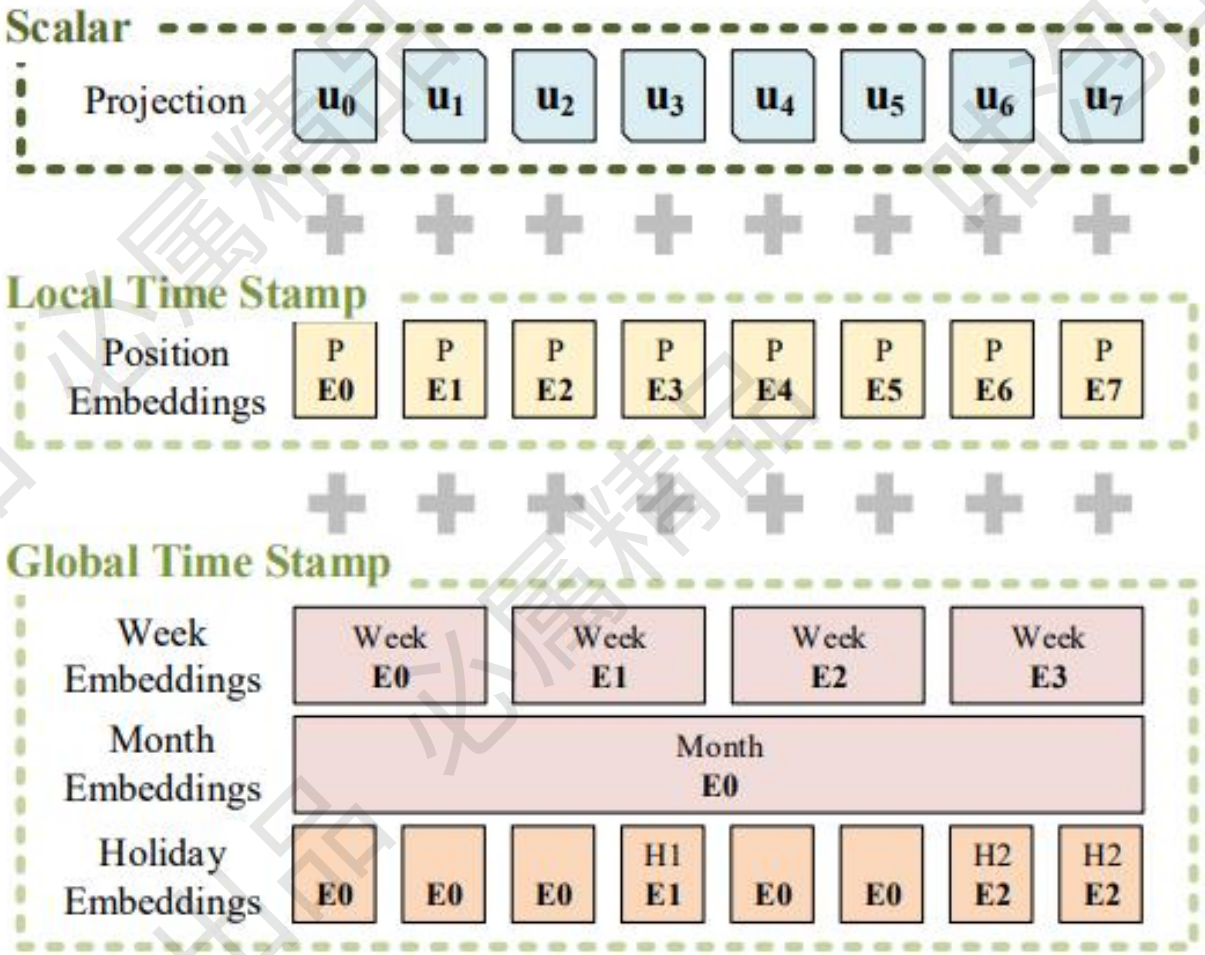


Informer

- Decoder输入
  - 源码中decoder输入长度为72，其中前48是真实值，后24是预测值
  - 第一步还是做自身的ProbAttention，注意这回需要加上mask
  - Mask的意思就是前面的不能看到后面的（不能透题）
  - 自身计算完Attention，再算与encoder的Attention即可

Informer

- 位置编码信息
  - 位置信息比较丰富这回
  - 不仅有绝对位置编码
  - 还包括了跟时间相关的各种编码
  - Encoder与Decoder都加入了



Informer

- 整体网络架构

- 主要改进就是编码和解码器的优化，速度更快，解决长序列问题