

✓ 特征匹配完之后能做什么呢？

✎ 两张图像匹配后，我们可以知道它俩的位姿(位移，角度等)变化

✎ 这就是咱们在机器人导航和三维重构中非常重要的一个模块

✎ 图像相似度计算(基于匹配到的点)，图像检索与匹配等

✎ 相当于可以根据关键点的匹配特征得到很多位置相关的信息

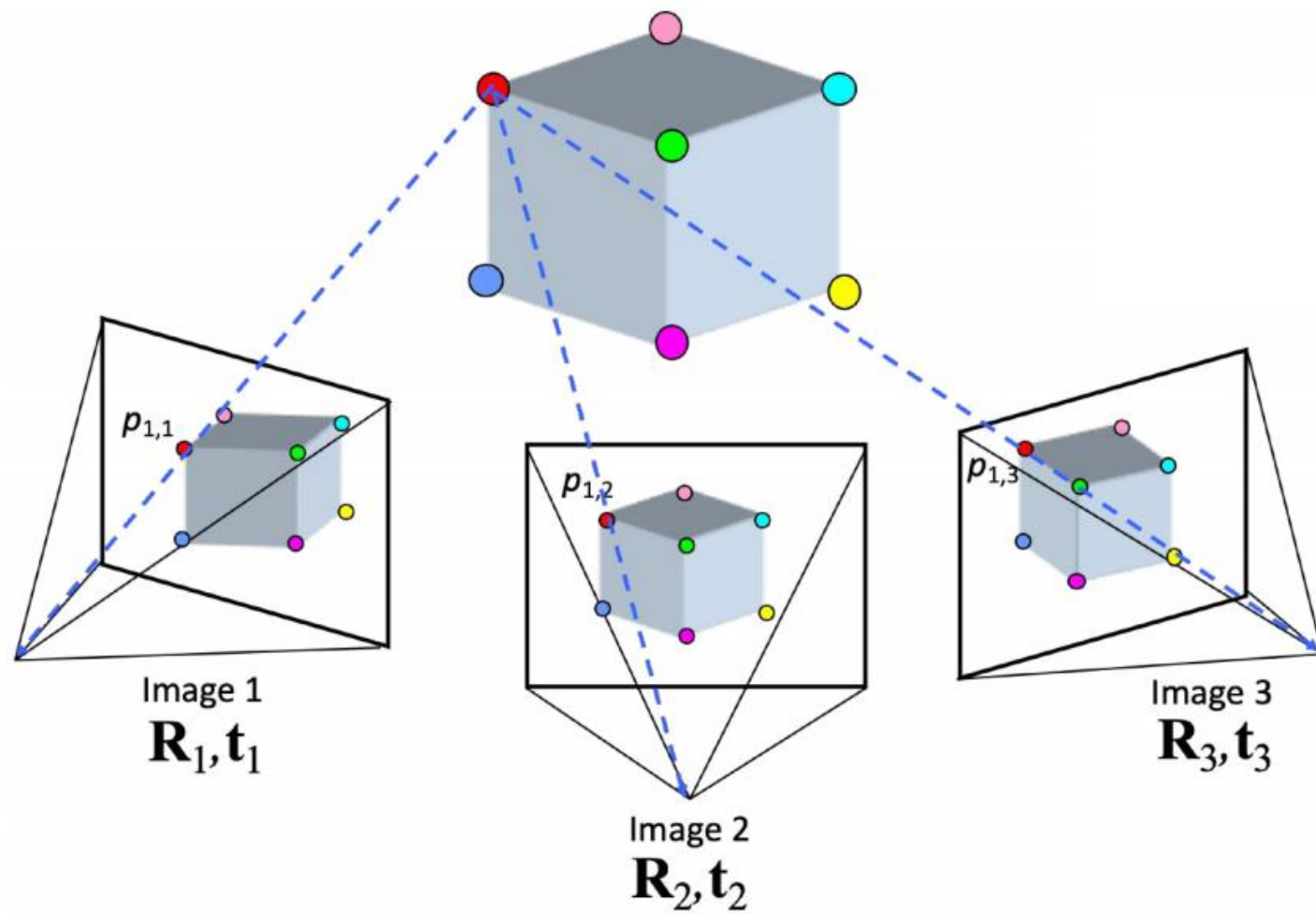
✓ 特征匹配应用

✎ 有了匹配就有位姿

✎ 相当于根据点求解RT

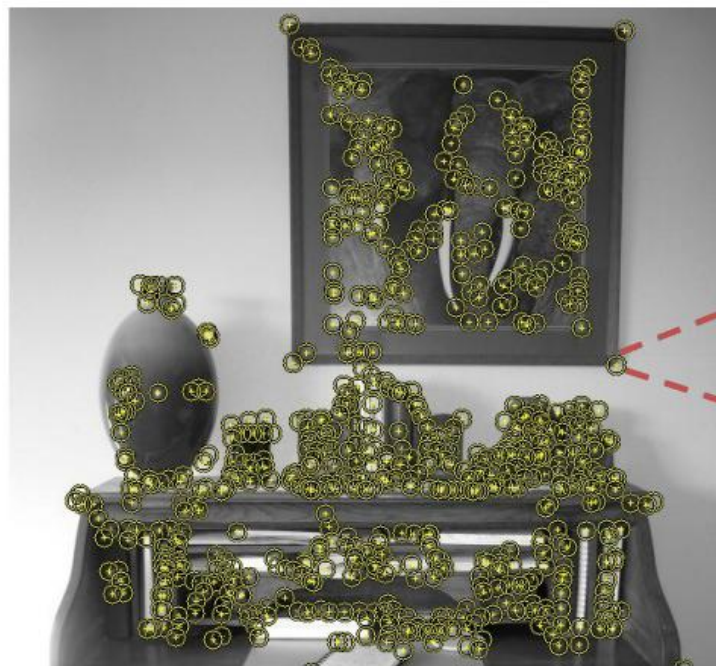
✎ 这样就可以知道很多信息

✎ 相对位置，相对位姿等

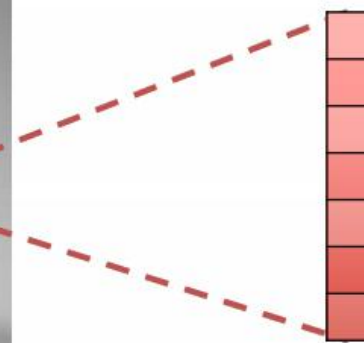


## ✓ 传统任务流程

✎ 传统任务中，完成特征匹配需要哪些已经条件呢？



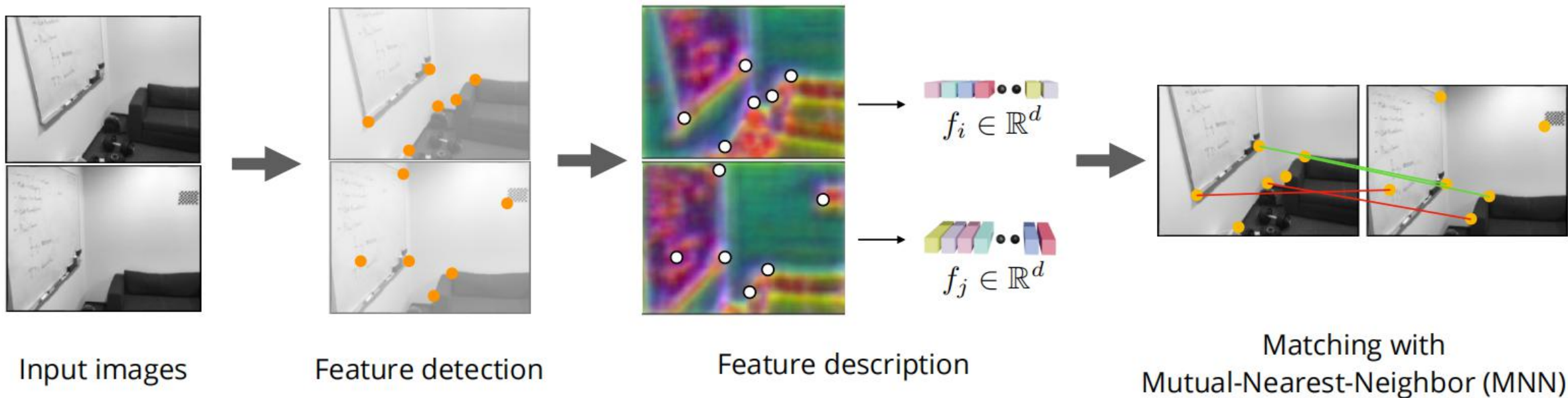
Feature detection



Feature description

## ✓ 基本流程

✎ 首先找到一些关键点(图像梯度较高, 角点检测等), 然后计算特征相似度来匹配



## ✓ 遇到的问题

✎ 这种方法很依赖检测到的特征点，一旦点找不到，那就不用说匹配了





## ✓ 遇到的问题

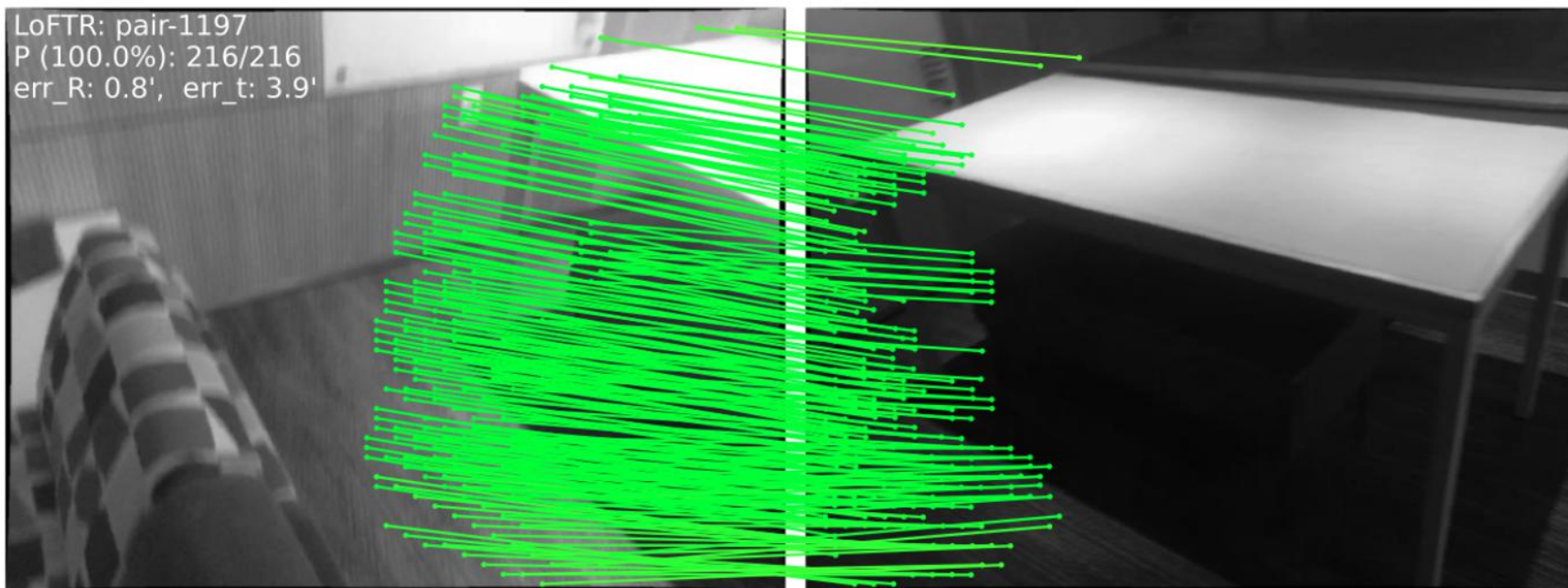
✎ 对于位置不同的两个点，如果它们的背景特征相似(与位置无关了)，也无法匹配



## ✓ Lofttr的优势

✎ 不需要先得到特征点，这也就解决了第一个问题

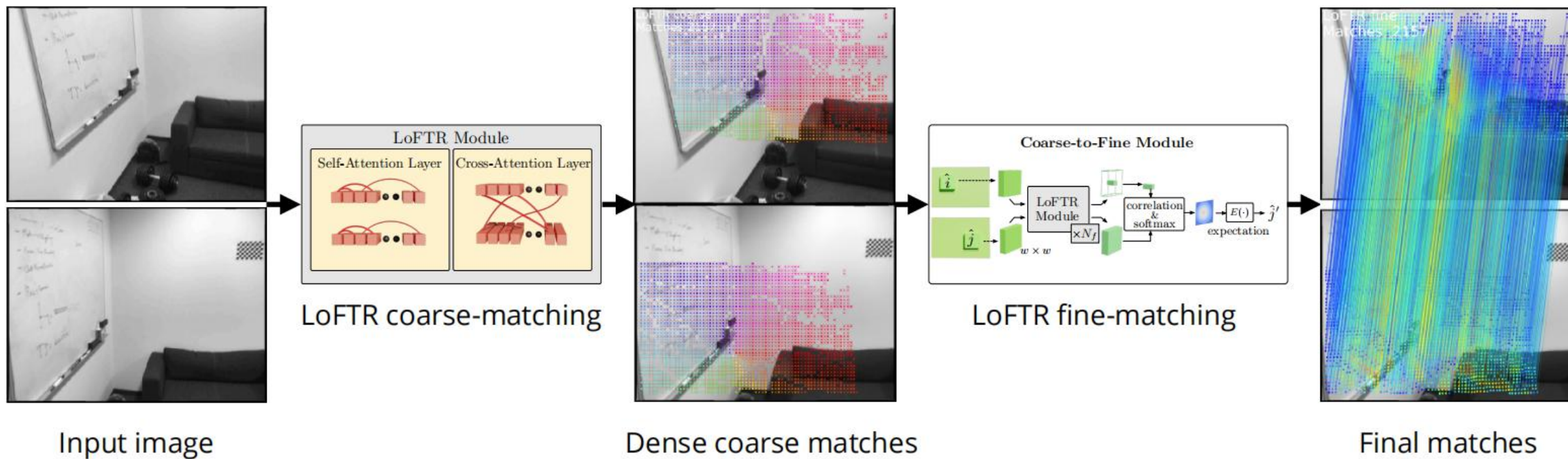
✎ End2End的方式，用起来比较方便，效果也更好



# LoFtr

✓ 整体流程

📎 核心就是transformer!!!





# Loftr

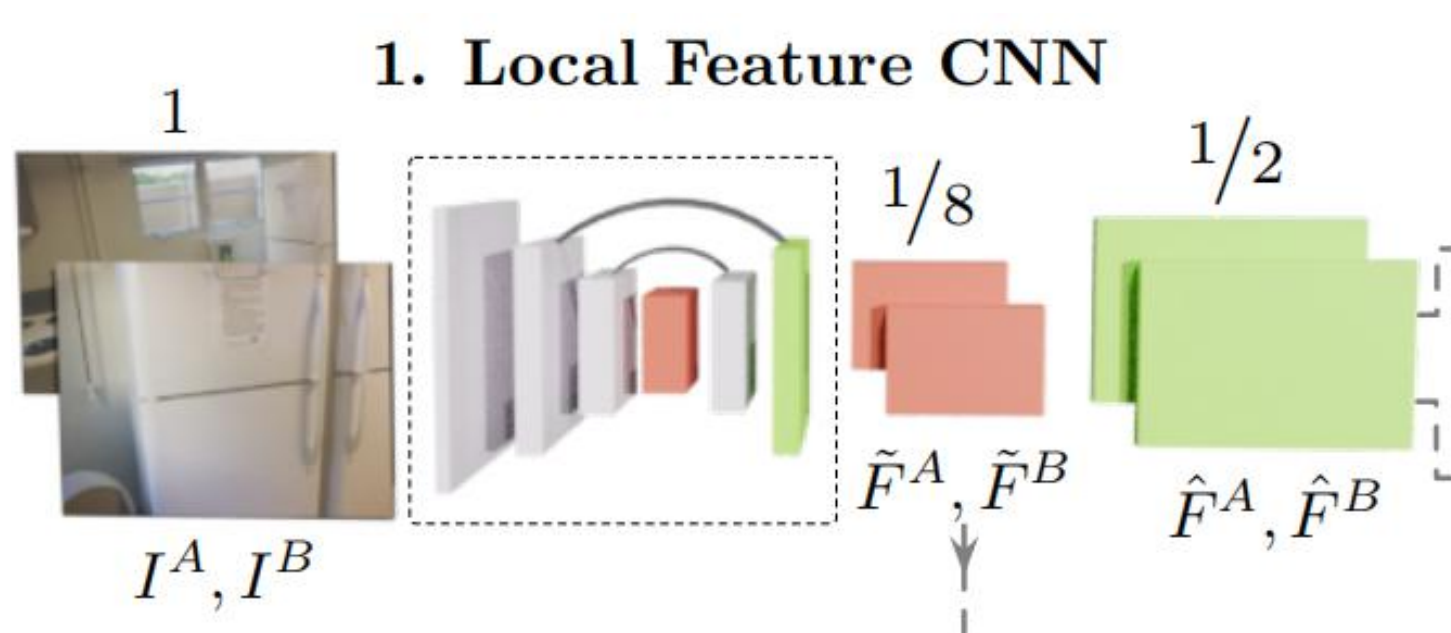
✓ 第一步: backbone特征提取

✎ 注意输入是两张图像

✎ 分别得到不同层特征

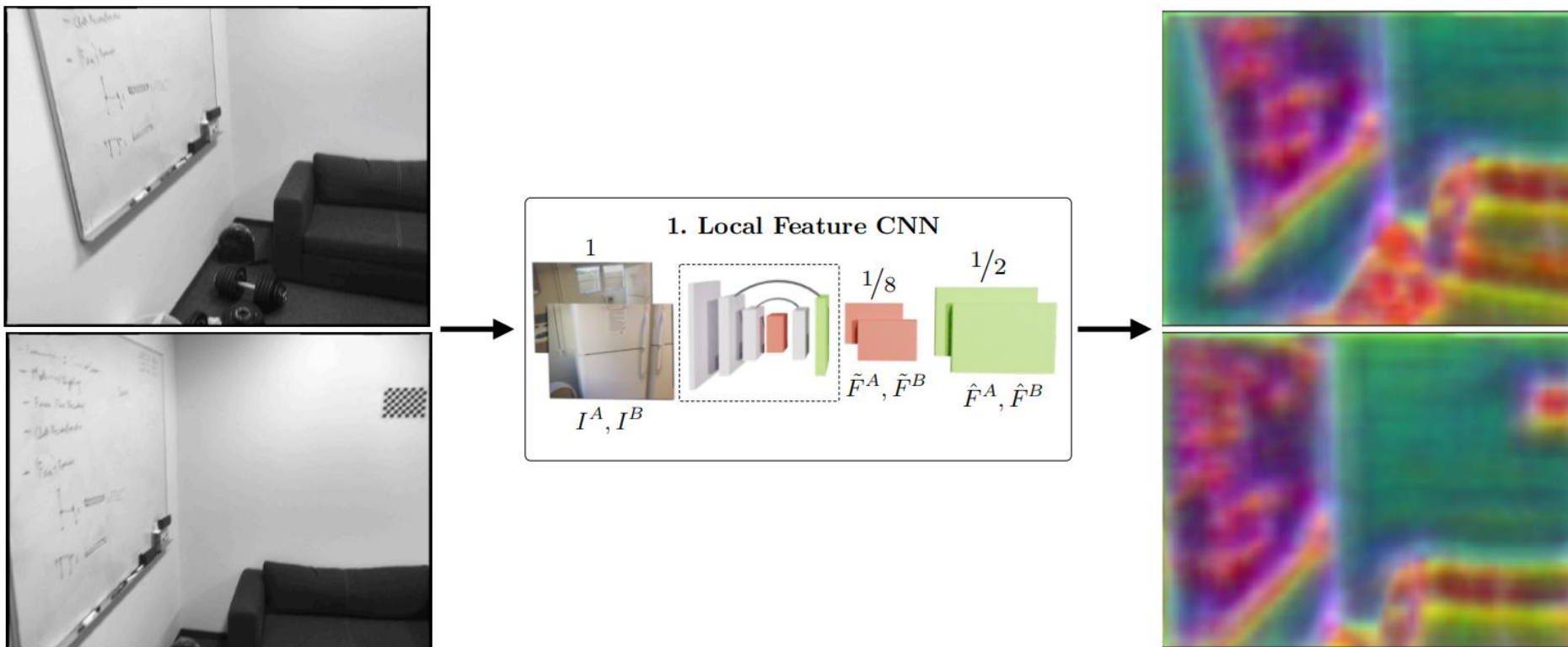
✎ 输入:[2,1,480,640]

✎ 两张图像拼一起进行backbone



✓ 第一步: backbone特征提取

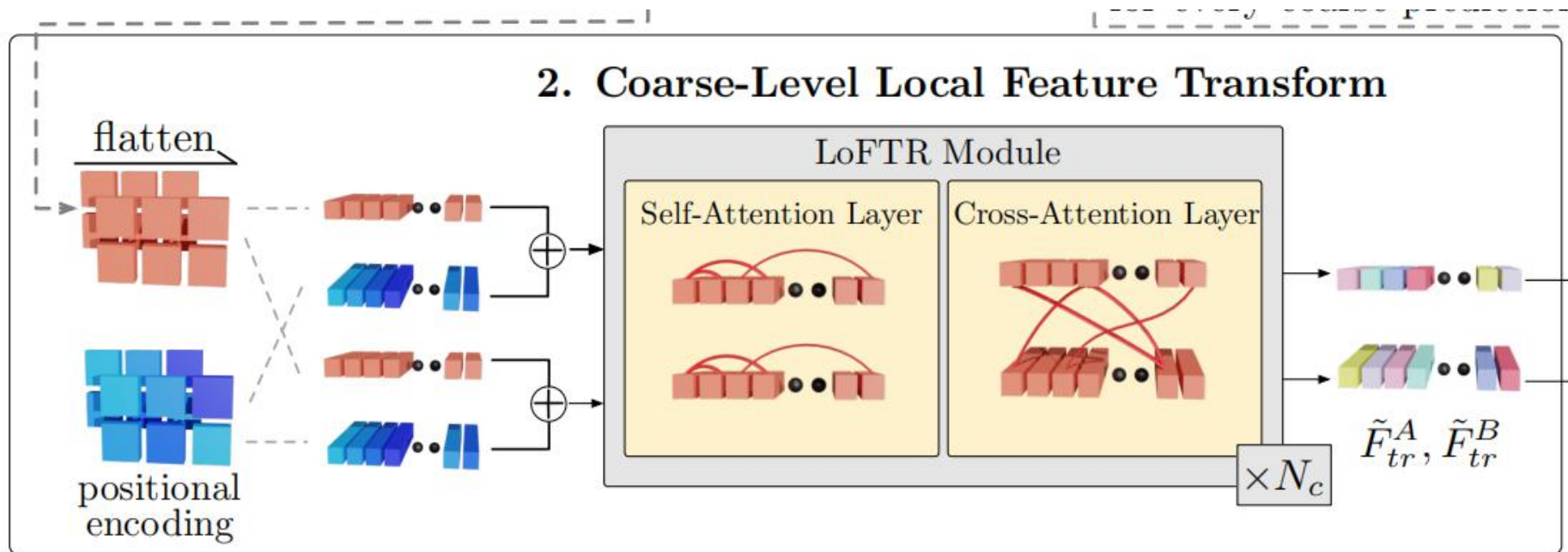
📎 Backbone得到的结果



## ✓ 第二步：transformer

✎ 这完全是专业对口啊，特征点匹配那不就是往transformer里面套嘛

✎ 首先进行位置编码，然后注意它是有两种方法：self和cross Attention



# Loftr

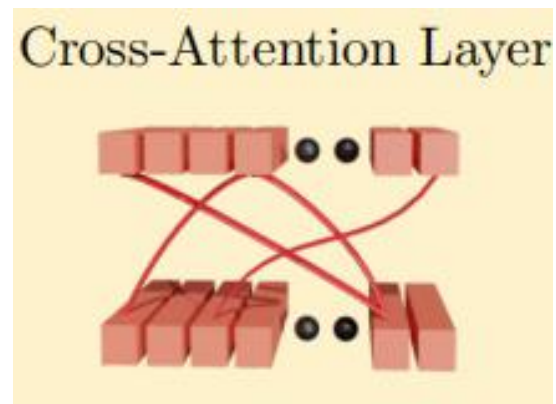
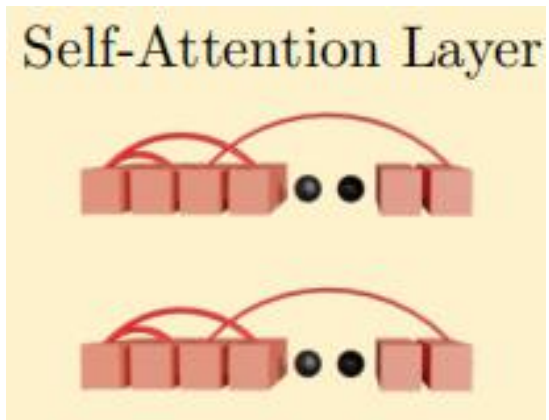
✓ 两种方法分别要做啥呢

✎ Self-Attention: 咱们自己家人先分工好

✎ 就相当于一会要打匹配了, 选好阵容别重了

✎ Cross-Attention: 如何对位呢? 打野对打野

✎ 这个就是Loftr的核心了, 找到每个点的对应关系



✓ 两种方法分别要做啥呢

✎ 不同Attention的可视化效果

✎ 其实每个点与自己家的事和每个点与别人家的事



Attention weight visualization of **self-attention**



Attention weight visualization of **cross-attention**



✓ transformer后得到的结果

✎ 位置关系和自身特征更明确(CNN后结果类似，但transformer后鲜明了)



## ✓ 第三步：粗粒度匹配

✎ 刚才一顿Attention咱们已经得到了两个图重构后的特征

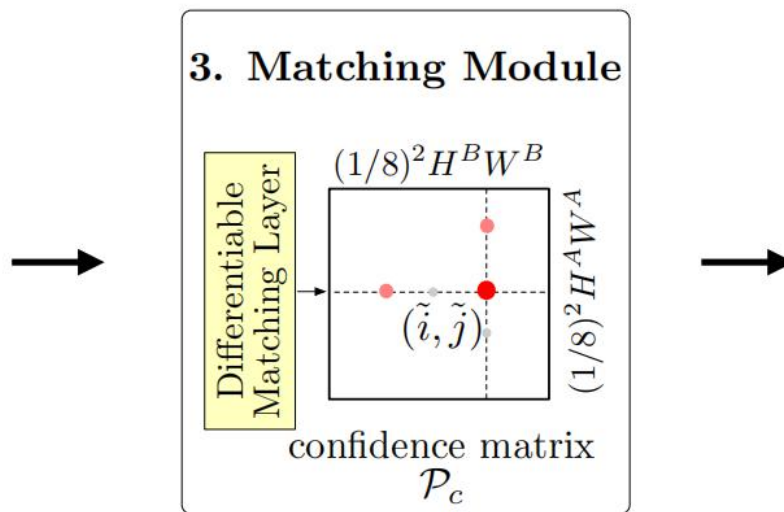
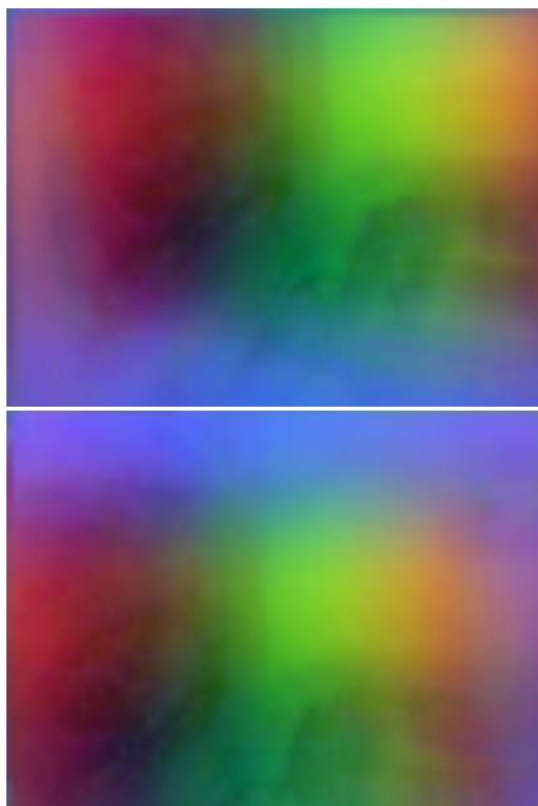
✎ 接下来计算它们之间的关系： $\mathcal{P}_c(i, j) = \text{softmax}(\mathcal{S}(i, \cdot))_j \cdot \text{softmax}(\mathcal{S}(\cdot, j))_i$

✎ 例如两张图分别有4800个点，现在咱们就能得到4800\*4800的关系矩阵

✎ 其中softmax表示分别对两张图中的内积结果做归一化，得到概率值

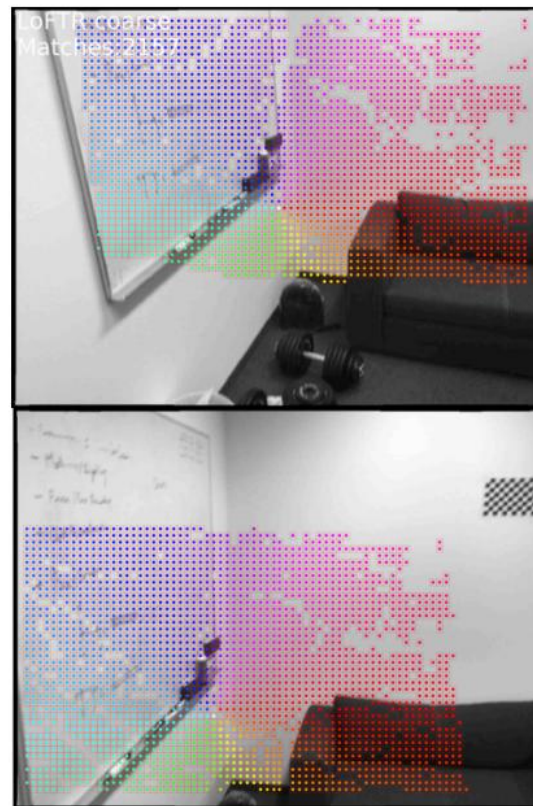
## ✓ 第三步：粗粒度匹配

✎ 特征都构建的挺好的了，这里要完成的就是粗粒度匹配



$$\mathcal{P}_c(i, j) = \text{softmax}(\mathcal{S}(i, \cdot))_j \cdot \text{softmax}(\mathcal{S}(\cdot, j))_i$$

Dual-softmax



## ✓ 第三步：粗粒度匹配

✎ 现在关系也有了，如何匹配呢？

✎ 先来一波过滤操作，如何关系不咋地的，肯定配对不了（阈值筛选）

✎ 然后再来一波互相最近邻，两个点如何匹配一定是互相的，并不是一厢情愿的

✎ 好了，现在我已经得到了两厢情愿的匹配结果，但是有木有啥问题呢？

## ✓ 第四步：细粒度匹配

✎ 我们刚才那个4800哪来的来着？是不是把原始图像中每个小区域当作一个点啦！

✎ 所以咱们这个4800和4800的匹配相当于找到了大概哪些区域是能匹配的

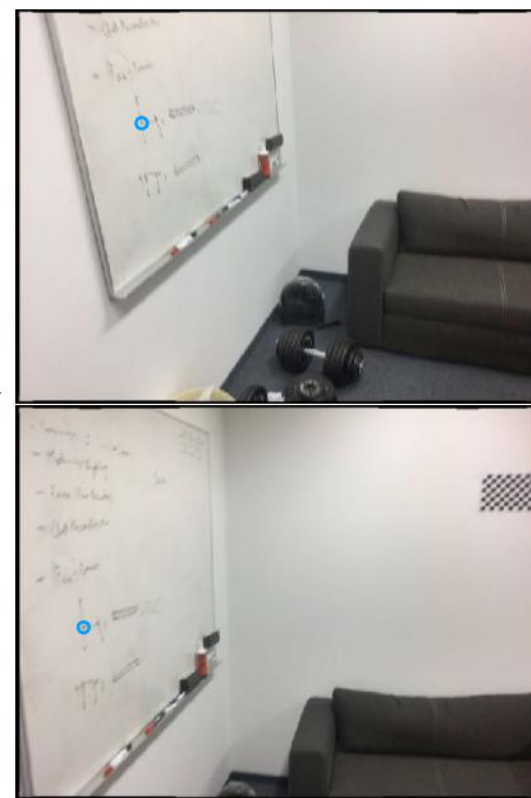
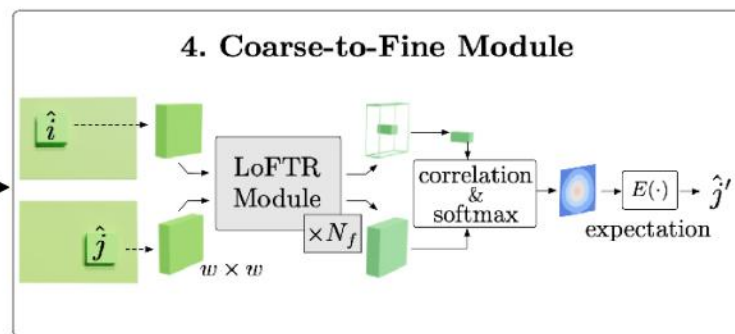
✎ 那么接下来我们就要看看更具体的位置是在哪了，相当于微调

✎ 在上一步的基础上继续调整，咱们就叫它Coarse-to-Fine



## ✓ 第四步：细粒度匹配

### ✎ 继续细化匹配任务



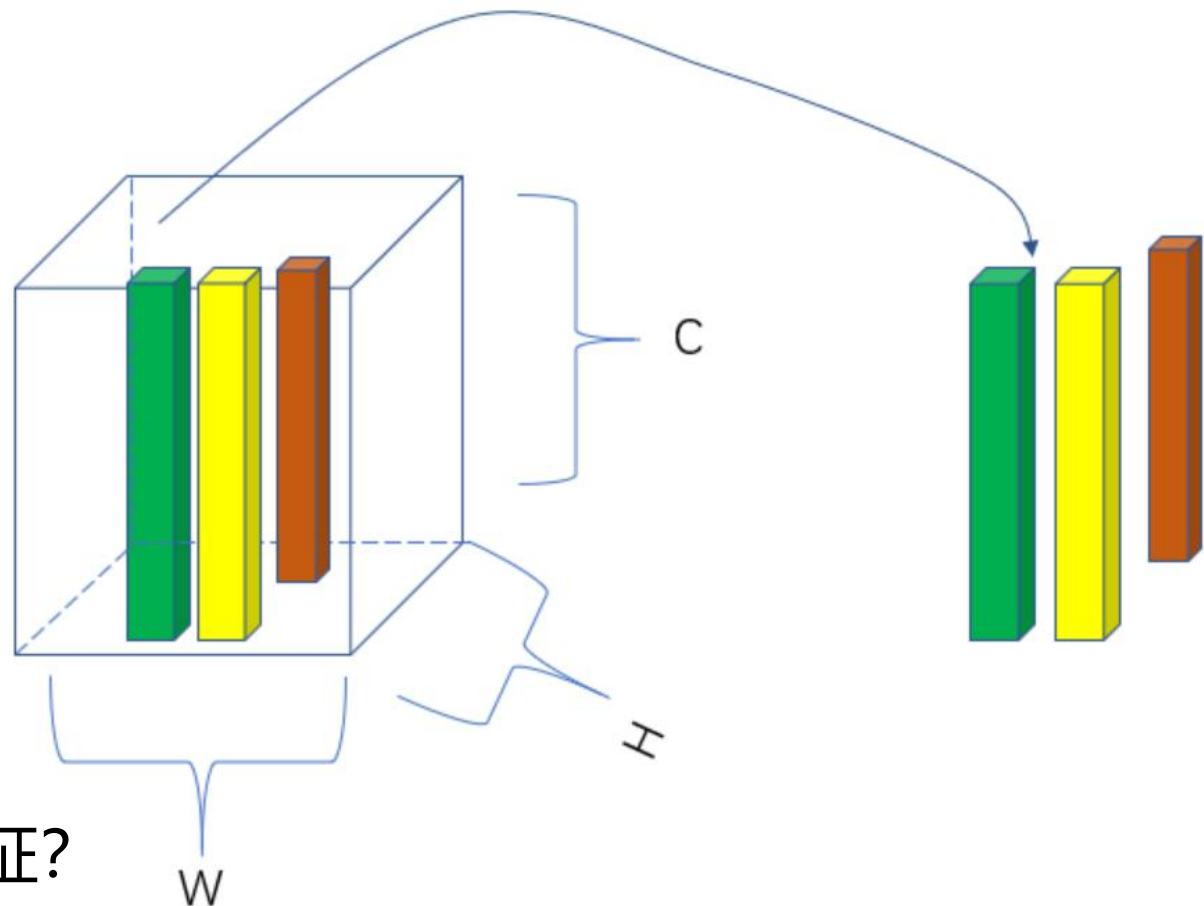
## ✓ 第四步：细粒度匹配

✎ 特征图拆解，F.unfold模块：

✎ 就是把特征图拆成多个特征块

✎ 例如输入 $128 \times 240 \times 320$

✎ 那么会输出多少块？ 每块多少个特征？



## ✓ 第四步：细粒度匹配

✎ 输出的每个长条（特征块）里面特征个数： $C * kH * kW$

✎ 例如  $128 * 5 * 5 = 3200$ ，表示每个长条有3200个特征

✎ 然后指定好步长，padding，也可以算出来一共得到多少块长条特征

✎ 源码中得到  $4800 * 3200$ （4800是长条的个数，3200是特征的个数）

## ✓ 第四步：细粒度匹配

✎ 转换维度得到：4800\*25\*128，也就是每个长条块是由25个点组成的

✎ 每个点现在是由128维的向量组成的，一会就该它们上场了

✎ 但是咱们现在不需要4800个区域，因为粗粒度中已经帮我筛选出来一部分了

✎ 例如筛选后得到4800个点里面只有142个是相互匹配的

## ✓ 第四步：细粒度匹配

✎ 现在咱们手里面有啥呢：142\*25\*128这样一个矩阵

✎ 对这142个已经匹配的区域，再做实际点的微调，也就是25个点再最匹配的

✎ 相当于一个区域由25个点组成，咱们要再这个区域里找到最准确的点位置

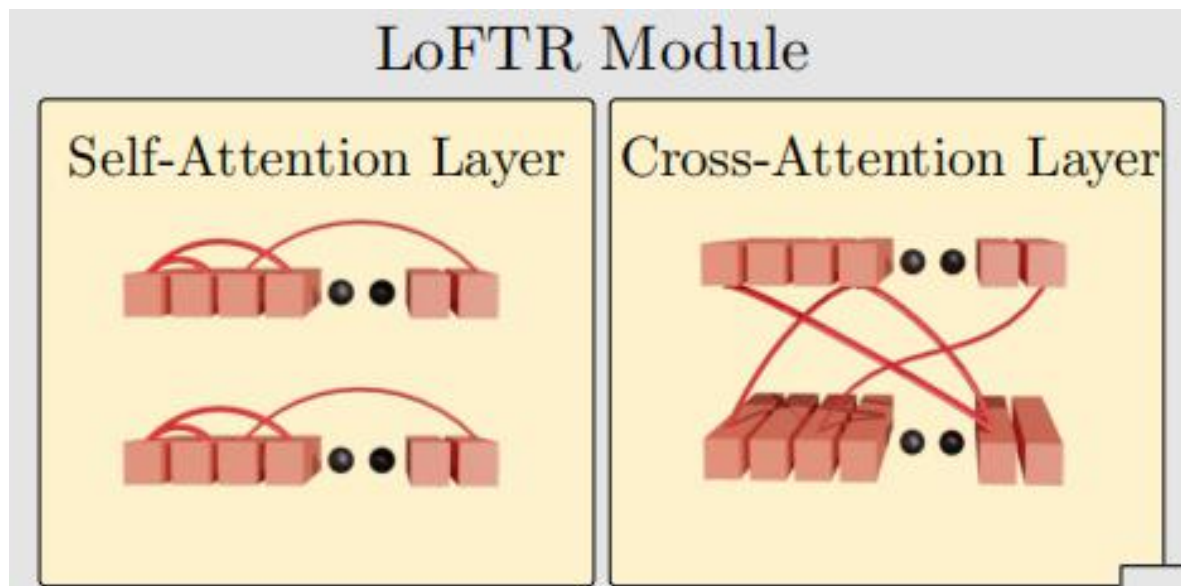
✎ 两张图，每个图有142个区域，每个区域有25个点，如何匹配呢？



## ✓ 第四步：细粒度匹配

✎ 25个点如何相互匹配呢？这不又回来了transformer了嘛！

✎ 在细粒度中，我们针对25个位置还要走Self和Cross Attention



## ✓ 第四步：细粒度匹配

✎ transformer过后，咱们只是把各个点的特征进行了重构

✎ 那接下来如何进行微调呢？现在我们要算这25个点与其中中心点的关系

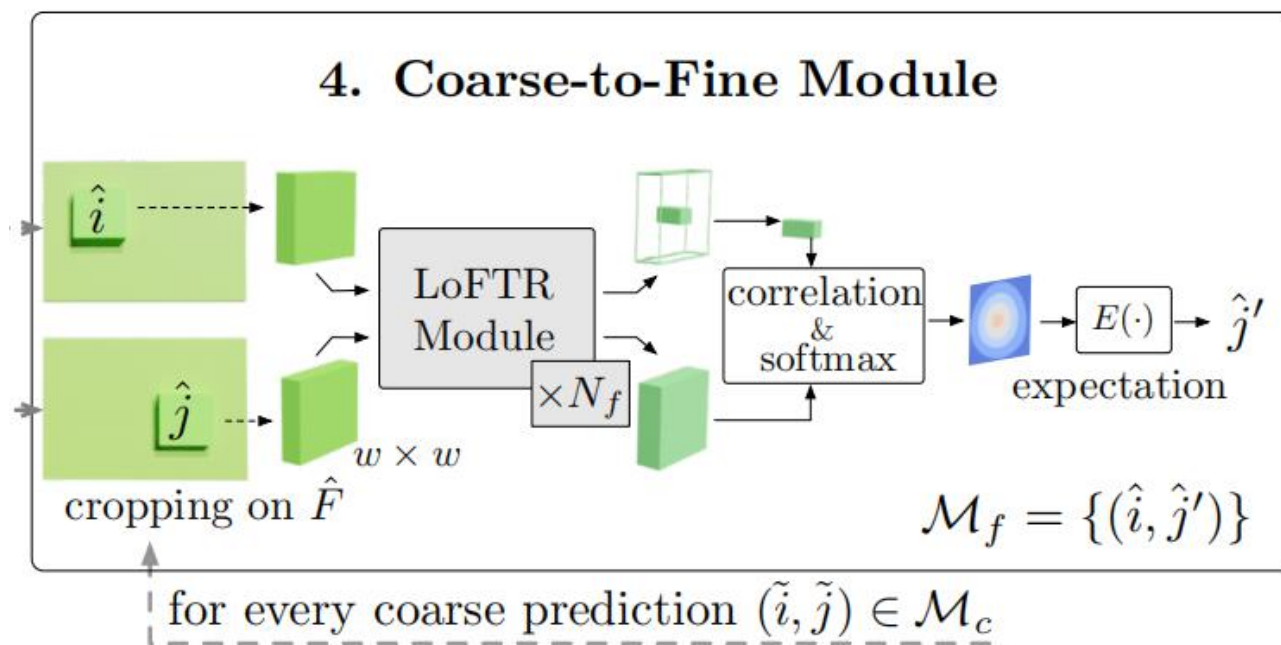
✎ 相当于我要以中心点为圆心，算周围点跟它的概率关系，这样会得到一个热度图

✎ 例如最后输出了 $142 \times 5 \times 5$ 的一个概率图，相当于哪块跟中心点关系紧密

## ✓ 第四步：细粒度匹配

✎ 再用这 $142 \times 5 \times 5$ 中个概率图进行期望，得到最终实际位置

✎ 相当于这25个点都会对最终结果产生影响，我们算其期望： $142 \times 2$ (实际位置)



## ✓ 效果对比

✎ 匹配到的点更全面，不依赖先验知识(检测到关键点)

✎ 稳定性更高，已经刷榜

SuperGlue



LoFTR



## ✓ 效果对比

✎ 匹配到的点更全面，不依赖先验知识(检测到关键点)

✎ 稳定性更高，已经刷榜

SuperGlue



LoFTR

