VIMA 的全称是 **VisuoMotor Attention（Agent）**，直译就是 **“视觉-运动注意力（智能体）”**：

- **VisuoMotor**：视觉（vision）+ 运动控制（motor）
- **Attention**：注意力机制（用 Transformer 的 attention 去对齐“看到的东西”和“该怎么动”）

智能体（Agent）是一个在环境中为达成目标而进行多步决策与行动的闭环系统，通常包含感知、状态表示、策略/规划与执行，并根据反馈持续调整。

1️⃣ Token（词元/令牌）

顾名思义：Token 就是"信息的最小单位"，是模型处理的"基本颗粒"。

生活比喻：

- 你读一本书，最小单位是什么？**字**（或者词）
- 模型读数据，最小单位就是 **Token**

不同场景下的 Token：

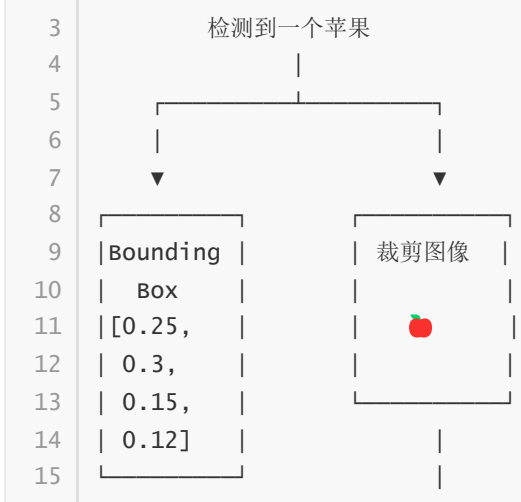
场景	Token 是什么	举例
NLP（文本）	一个词或子词	"机械臂" → ["机械", "臂"]
传统CV	一个像素？太小了！	224×224 图片 = 50176 个像素，太多了
ViT	一个图像块（patch）	把图片切成 16×16 的小块
VIMA	一个物体	检测出的每个物体是一个 token

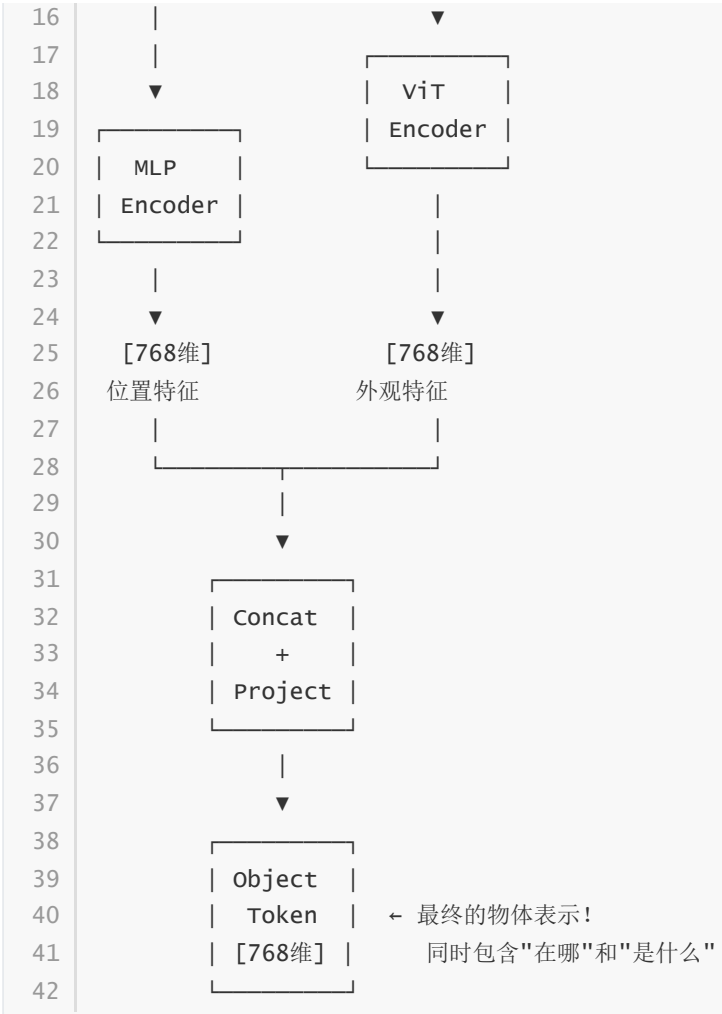
为什么需要 Token？ Transformer 模型只能处理**序列**（一串东西），所以必须把输入切成一个个 token 排成队。

这个拆解的一个又一个token指的就是这个张量

位置信息（MLP encoder）（多层感知机）和这个形状特点（Vision transform encoder）信息，两方面确定这个图像的特征，我们根据这个特征将其构造成一个信息单元token

🎨 图示总结





你之前那句话怎么修正更准确?

你说：“先检测有用物体 → 找有用 patch → transform → 处理”。

更贴近 VIMA 的说法是：

> **先检测出物体（不一定只选“有用”的，而是把场景/提示里的物体都 token 化），对每个物体：用 ViT 编码外观、用 MLP 编码框位置，然后把这些 object tokens 作为序列喂给 Transformer。**

确实如此，应当刚开始的时候检测这个所有的这个物体，然后通过多层感知机和这个图像转换器构成这个信息单元token，根据transform转换成这个张量最后根据这个LLM进行决策

这就是这个VIMA的多模态的最大革新点，可以将这个图片当做这个指挥，我们的手段本质是将其转换为这个张量的格式

1) Multimodal Prompt：多模态提示 = “任务说明书”

图最上面那个框就是“题目”本身，例子像：

- “Sweep all [图:物体] into [图:容器]”
- “without touching [图:障碍物]”

你注意到了：这里的 **[图:物体]** 不是文字，而是**用图像中的物体来当变量。**
这就是 VIMA 的核心：**prompt 里可以直接“指物体”**，而不是只能说“the red block”。

Prompt Tokenizer好一个tokenizer，token化，这个名字起的不错

📍 第三层：Frozen T5 Encoder（编码器）

```
Frozen T5 Encoder（预训练）
输出: Prompt Tokens [L, d]
```

这是什么？ 把切碎的 tokens 进一步"理解"，输出一个**统一的表示**。

关键词解释：

- **Frozen（冻结）**：T5 是在海量文本上预训练好的，我们**不再更新它的参数**，直接拿来用
- **[L, d]**：输出的形状
  - L = 序列长度（有多少个 token）
  - d = 每个 token 的维度（通常是 768）

生活比喻： T5 就像一个**读过无数书的老学者**，你把句子给他，他能快速理解意思：

```
输入: [sweep] [all] [苹果token] [into] [容器token]
        ↓
```

这个第三步相当于一个新的编码器，相当于将这个向量（张量）形式转换为这个机器可理解的语言，相当于一层解码，一层语言转换，我们通过一个已经训练好的模型T5进行这个转换

📍 第四层：Robot Controller（机器人控制器/解码器）

这是**最核心**的部分！它包含两个子模块：

4.1 Cross-Attention（交叉注意力）

```
Cross-Attention
(Q=History, K,V=Prompt)  ←——— Prompt Tokens
```

这是什么？ 让机器人的"记忆"去**查询指令**，确保每一步都在按指令做事。

有意思，根据这个已经规划好的上下文进行这个对应的操作（prompt）

4.3 两者如何配合？（重复 L 层）

```
for i in range(L层):
    # 第一步：问问指令说要干嘛
    x = CrossAttention(Q=我的记忆, KV=指令)

    # 第二步：回顾我之前做了什么
    x = CausalSelfAttention(Q=x, KV=x)
```

重复多次的原因：理解可以**层层加深**

- 第 1 层：粗略理解"要把东西放进容器"
- 第 2 层：更细理解"是苹果，放进碗"
- 第 3 层：最细理解"苹果在左边，碗在右边"

好一个理解可以这个层层加深

这是什么？ 把理解结果转换成具体的机械臂动作。

输出格式：

```
动作 = (抓取位置, 放置位置)
     = ((x1, y1, θ1), (x2, y2, θ2))
        ↑              ↑
     从哪里抓        放到哪里
```

生活比喻： 前面所有的"思考"最终要落实成手的动作：

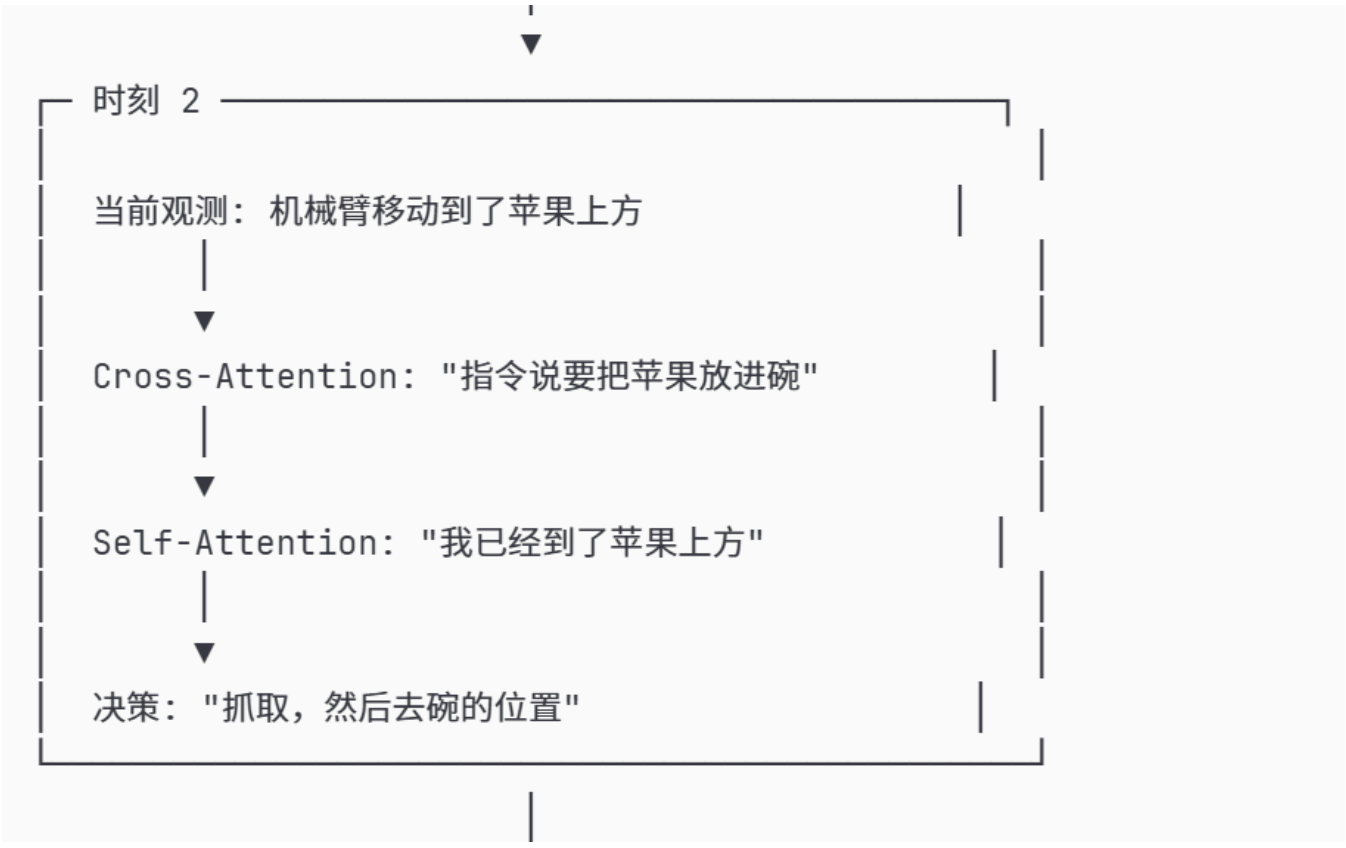
```
大脑思考结果: "苹果在左边(0.2, 0.3), 碗在右边(0.8, 0.6)"
      |
      ▼
Action Decoder
      |
      ▼
手的指令: "移动到(0.2, 0.3), 旋转0°, 抓取;
          移动到(0.8, 0.6), 旋转0°, 放下"
```

好一个根据上述指令进行这个动作编码，这就是最终的目的，就是这个进行这个最终的执行动作，根据已知的信息

从接受这个指令，到这个信息的具体执行的这个流程

```
用户指令: "把 [图:🍎] 放进 [图:🥣]"

═══════════════════════════════════════════════════════
Step 1: Prompt Tokenizer 切碎
═══════════════════════════════════════════════════════
"把" → [把]
"放进" → [放][进]
[图:🍎] → [苹果token: 位置(0.2,0.3) + 红色圆形]
[图:🥣] → [碗token: 位置(0.8,0.6) + 白色凹形]

═══════════════════════════════════════════════════════
Step 2: T5 Encoder 理解
═══════════════════════════════════════════════════════
输入: [把, 苹果token, 放, 进, 碗token]
输出: 一串向量，表示"把左边的红苹果放进右边的白碗"

═══════════════════════════════════════════════════════
Step 3: Robot Controller 决策
═══════════════════════════════════════════════════════

  ┌─ 时刻 1 ──────────────────────────────────────┐
  │                                               │
  │  当前观测: 看到桌面上有苹果和碗                  │
  │       |                                       │
  │       ▼                                       │
  │  Cross-Attention: "指令说要把苹果放进碗"         │
  │       |                                       │
```

```
   |       ▼                                          |
   |  Self-Attention: "我还没做任何事"                  |
   |       |                                          |
   |       ▼                                          |
   |  决策: "先去抓苹果"                                |
   └──────────────────────────────────────────────────┘
                           |
                           ▼
   ┌─ 时刻 2 ──────────────────────────────────────────┐
   |                                                  |
   |  当前观测: 机械臂移动到了苹果上方                    |
   |       |                                          |
   |       ▼                                          |
   |  Cross-Attention: "指令说要把苹果放进碗"            |
   |       |                                          |
   |       ▼                                          |
   |  Self-Attention: "我已经到了苹果上方"               |
   |       |                                          |
   |       ▼                                          |
   |  决策: "抓取，然后去碗的位置"                       |
   └──────────────────────────────────────────────────┘
                           |
                           ▼
                          ...

═══════════════════════════════════════════════════
Step 4: Action Decoder 输出
═══════════════════════════════════════════════════
最终动作: pick(0.2, 0.3, 0°) → place(0.8, 0.6, 0°)
          "在(0.2,0.3)抓取，在(0.8,0.6)放下"
```



交叉注意和这个自我注意，一个是始终注意这个历史，一个是始终注意这个指令的执行

4.2 关键设计选择：Cross-Attention vs Direct Modeling

这是论文的**核心贡献之一**，必须理解！

方法	做法	问题
Direct Modeling (GPT 风格)	把 Prompt + History 拼成一个大序列，用 causal self-attention	序列太长，计算量大；Prompt 信息容易被"稀释"
Cross-Attention (VIMA 用的)	Prompt 单独编码，Controller 通过 cross-attention "查询"它	Prompt 信息保持完整；每一步都能"重新审视"指令

好一个查询它，真是天才的思想，足够的富有这个创新性，具体指令——规划上下文prompt——执行——查询上下文——继续执行

你不需要深入理解四元数的数学，只需知道：

> 四元数是一种**更稳定**的旋转表示方法，避免了角度跳变的问题

世界的专业性质越来越强，我们有时候无需了解这个东西的具体的含义，只需要知道他的这个具体的用法即可，为了快速跟上这个时代的发展

4.3 动作空间（Action Space）

VIMA 继承自 Ravens 仿真器的动作空间：

```
动作 = (pick_pose, place_pose)
     = ((x1, y1, θ1), (x2, y2, θ2))
```

- **pick_pose**：从哪里抓取
- **place_pose**：放到哪里
- 每个 pose 是 SE(2)：平面位置 (x, y) + 旋转角度 θ

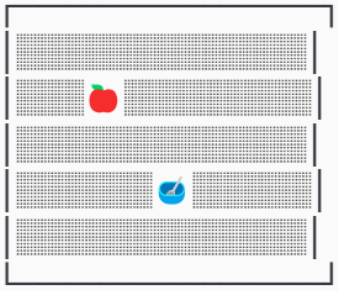
离散化：

- X 轴：50 个 bins
- Y 轴：100 个 bins
- 旋转：50 个 bins（四元数表示）

这些都是具体操作的时候的动作规范，规范这个操作精度，规范这个如何操作，二维特殊欧几里得群等等，好一个动作空间，这个名字起的不错

对比两种方法：

方法 A（像素级，如 VIMA-Gato）：



模型需要从海量像素中
自己学习"哪里有物体"
→ 需要大量数据！

方法 B（物体级，VIMA）：

直接告诉模型"这里有苹果"
模型只需要学习"怎么操作物体"
→ 学习任务简单，数据需求少！

特殊的算法使得这个对数据集的需求量降低了

总结：VIMA 成功的三个秘诀

秘诀	具体做法	效果
看得准	Object Tokens（物体级视觉）	数据效率提升 100×
记得牢	Cross-Attention（每步查指令）	泛化能力更强
懂得多	预训练 T5（站在巨人肩膀上）	语义理解更好

真是精彩绝伦的思想，也不知道这篇文章属于什么层次

7.1 策略函数：$\pi(a_t|\mathcal{P},\mathcal{H})$

先看符号含义

符号	读法	含义	比喻
π	"派"	策略/政策（Policy）	机器人的"大脑决策系统"
a_t	"a 下标 t"	第 t 步要执行的动作	"现在该做什么"
$\mathcal{P}$	"P 花体"	Prompt（指令）	"老板给的任务书"
$\mathcal{H}$	"H 花体"	History（历史）	"我之前做了什么"
$\|$	"竖线/给定"	条件概率的意思	"在...的条件下"

这个策略函数的这些字母用的还真是非常的形象

7.2 Cross-Attention 公式：

$$\mathcal{H}' = \text{softmax}\left(\frac{Q_{\mathcal{H}} K_{\mathcal{P}}^{\top}}{\sqrt{d}}\right) V_{\mathcal{P}}$$

这个公式看起来吓人，但其实就是在做一件事：**"带着问题去查手册"**

Step 0: 先理解 Q、K、V 是什么

符号	全称	含义	比喻
Q	Query	查询/问题	"我想问什么"
K	Key	钥匙/索引	"手册的目录"
V	Value	值/内容	"手册的正文"

它总是考虑了这个需要的和当下的进行的结合

```
问题：如果 d 很大（比如 768），Q×K 的结果会很大

例如：d = 768
  Q×K 的某个值可能 = 50（很大的数）

如果直接做 softmax：
  softmax([50, 2, 3, 1, 1]) ≈ [1.0, 0, 0, 0, 0]
  → 几乎把所有注意力都放在第一个上，其他的信息全丢了！

除以 √768 ≈ 27.7 后：
  [50, 2, 3, 1, 1] / 27.7 ≈ [1.8, 0.07, 0.11, 0.04, 0.04]
  softmax 后 ≈ [0.6, 0.1, 0.1, 0.1, 0.1]
  → 分布更平滑，能关注到多个相关信息
```

大白话：防止某一个匹配度太大，导致只看一个词而忽略其他词

则个softmax激活函数还是增强了其他元素的比重，是为了防止这个过拟合吗

Step 4: 乘以 $V_{\mathcal{P}}$（加权读取内容）

$$\mathcal{H}' = \text{softmax}(\cdots) \times V_{\mathcal{P}}$$

$V_{\mathcal{P}}$ 是什么？

- Prompt 的"内容向量"，包含每个词的详细语义信息

```
V_P 的形状: [5, d]  （5个词，每个词有 d 维的语义信息）

  V_把   = [0.1, 0.3, -0.2, ...]  ← "把"的语义
  V_苹果 = [0.8, 0.1, 0.5, ...]   ← "苹果"的语义
  V_放   = [0.2, 0.6, -0.1, ...]  ← "放"的语义
  V_进   = [0.1, 0.4, 0.0, ...]   ← "进"的语义
  V_碗   = [0.9, 0.2, 0.7, ...]   ← "碗"的语义
```

好一个详细的信息的转换，相当于是这个T5的功能是吗

公式全貌

$$\min_{\theta} \sum_{t=1}^{T} -\log \pi_\theta(a_t|\mathcal{P}, \mathcal{H})$$

一句话翻译：调整模型参数 θ，让模型预测的动作 **尽可能接近专家的动作**。

逐个符号拆解

符号	含义	比喻
$\min_\theta$	调整参数 θ，让后面的值最小	"调整大脑，让错误最少"
$\sum_{t=1}^{T}$	把每一步的损失加起来	"整个任务的总错误"
$-\log$	负对数（把概率变成损失）	"概率越低，惩罚越大"
$\pi_\theta(a_t\|\mathcal{P}, \mathcal{H})$	模型预测"正确动作"的概率	"模型觉得专家动作对不对"

好一个概率，负对数，总损失

$$\text{损失} = -\log(\text{预测正确答案的概率})$$

```
如果模型预测正确答案的概率是 70%:
  损失 = -log(0.7) ≈ 0.36  ← 损失较小 ✓

如果模型预测正确答案的概率是 10%:
  损失 = -log(0.1) ≈ 2.30  ← 损失很大 ×

如果模型预测正确答案的概率是 99%:
  损失 = -log(0.99) ≈ 0.01 ← 损失极小 ✓✓
```

正确的反面就是这个损失，他这一层嵌套加的真是精彩

文阅读进度

$\sum_{t=1}$：整个轨迹的总损失

一个任务有 T 步，每一步都要算损失，然后加起来：

```
任务："把苹果放进碗"，共 4 步

时刻1：专家动作 = 向左移动
       模型预测概率 = 0.8
       损失₁ = -log(0.8) = 0.22

时刻2：专家动作 = 抓取
       模型预测概率 = 0.9
       损失₂ = -log(0.9) = 0.11

时刻3：专家动作 = 向右移动
       模型预测概率 = 0.6
       损失₃ = -log(0.6) = 0.51

时刻4：专家动作 = 放下
       模型预测概率 = 0.95
       损失₄ = -log(0.95) = 0.05
```

有意思这个真实情况和这个实际情况之间，因为已经知道了这个最后的这个结果，因此可以直接进行计算这个损失函数

✅ 总结

公式部分	含义	大白话
$\pi_\theta(a_t \| \mathcal{P}, \mathcal{H})$	模型预测正确动作的概率	"模型觉得专家动作有多对"
$-\log(\cdot)$	把概率转成损失	"概率越低，惩罚越重"
$\sum_{t=1}^{T}$	每一步损失求和	"整个任务的总错误"
$\min_\theta$	调整参数最小化损失	"不断练习，减少错误"

这个就是根据损失函数调整这个参数的过程，以达到这个行为模仿的最大化

📊 Figure 1：多模态 Prompt 示例（图文混排指令）

这张图展示了 VIMA 最核心的创新点——**指令不再是纯文字，而是文字和图片混在一起**。

图中展示了 4 种任务类型：

任务类型	中文名	Prompt 示例	通俗解释
Visual Goal: Rearrangement	视觉目标：重排	"Rearrange objects to match this scene: [图片]"	"把桌上的东西摆成这张图的样子"
One-shot Demonstration	单次演示模仿	"Follow this motion for [物体图]: [视频帧序列]"	"看我演示一遍，你照着做"
Novel Concept Grounding	新概念落地	"This is a blicket [图] This is a wug [图] Put a wug into a blicket"	"我现在定义新词汇，你学会后执行"
Visual Constraint	视觉约束	"Sweep all [物体] into [容器] without exceeding [边界线]"	"把东西扫进去，但别碰到红线"

带上图的四种指令类型

💡 对你课题的启发

当你以后评估自己的系统时，也要考虑：

- 训练时见过"螺丝刀"，测试时换成"扳手"，系统还能工作吗？（L3）
- 训练时只学过"放置"，测试时要求"组装"，系统能迁移吗？（L4）

好一个迁移能力和泛化学习

这是 VIMA 的核心机制：

通俗理解：机器人在决定下一步动作时，会**反复"回看"任务指令**。

用公式表示：

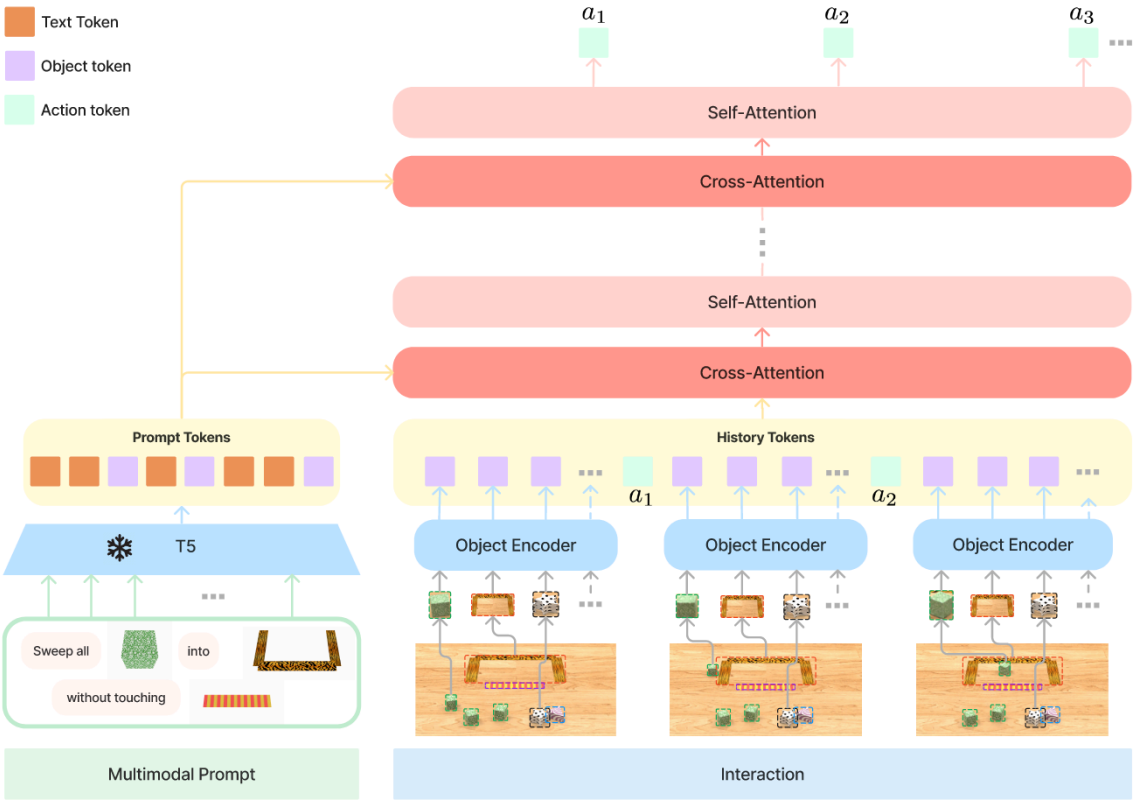
$$H' = \text{softmax}\left(\frac{Q_H K_P^T}{\sqrt{d}}\right) V_P$$

其中：

- Q_H = 历史序列的 Query（"我现在在哪？做了什么？"）
- K_P, V_P = 提示序列的 Key 和 Value（"任务要我做什么？"）
- d = 特征维度（用于缩放，防止数值过大）

生活比喻：就像你做数学题时，做到一半会回头看题目确认"题目到底要我求什么"。

对事物进行创新，很大程度上就是模仿人的过程，确实这就是人工智能的核心之一，模仿人的特征，那么我们以后在做这个课题的时候，就要想想人，或者说自己是如何做的，然后自己的行为如何在这个计算机上进行这个实现



这张图就很好的解释了这个，交叉注意力机制corss-Attention，会一直回头看这个prompt

为什么 VIMA 更鲁棒?

论文给出的解释：

> **Object-centric representation（以物体为中心的表示）** 比直接从像素学习更不容易过拟合。

好一个更不容易进行这个过拟合，object-centric不容易将这个背景的无用信息给引入

这里的**"可变 token（variable #tokens）"**，指的不是"token 的维度变来变去"，而是：

> **每一条样本（每张图/每个场景）输入给模型的"token 个数"不固定，会随着场景里检测到的物体数量变化。**

你图里写的 **可变（3~8个）**，就是在他们的设置里：一张场景图通常会检测出大概 3 到 8 个关键物体，于是视觉侧就产生 3~8 个 **object tokens。**

这个概念也算是非常简单

只是有时候没有人用人话进行这个讲解

这里的 **baseline（基线方法/对照组）**，你可以把它理解成：

> **为了证明 VIMA 的设计"确实更好"，作者必须找一些"合理、常见、能跑通"的替代方案当对照组。**
> 这些对照组就是 baseline：**同样的任务、同样的数据、尽量相同的训练设置，只改某个关键设计点**，看性能差多少。

所以 **baseline 不是"随便找的模型"**，而是"用来对比、证明贡献"的参照物。

好一个基准模型，这个做实验这一块，让我回想起了这个高中的生物和这个物理，可惜我已经发生了很大的改变，已经不是过去的自己了，这些东西，随着我学习能力和认知的提升，已经，不可用过去的眼光进行这个衡量

当用户只能说话时，可以这样改造：

```
用户语音："把那个红色的东西放过来"
          ↓
    语音识别（ASR）转文字
          ↓
    视觉定位模块（Visual Grounding）
    "在当前画面中找到'红色的东西'"
          ↓
    生成物体的裁剪图
          ↓
    构造 VIMA 风格的多模态 Prompt:
    "Put the [检测到的物体图片] into [目标位置图片]"
```

对于纯语音指令，可以通过这个语音本身所含有的信息构造多模态

图片不一定要用户提供——可以由系统的摄像头自动捕获

工人说"把那个螺丝拧紧"→ 系统用 Grounding DINO 检测"螺丝" → 自动生成图片 token