

1. Flamingo (由 Google DeepMind 提出)

Flamingo 是 2022 年发布的模型，它最重大的贡献在于证明了“**冻结的大模型也能处理视觉信息**”。

- **核心思路**：如果你已经有一个很聪明的语言模型（比如 Chinchilla），没必要为了让它看图而从头训练。Flamingo 像是在 LLM 上加了几个“视觉插件”。
- **关键组件**：
 - **Perceiver Resampler**: 这是一个视觉特征提取器。它不管输入的图片是什么尺寸或多少张，都会把它们转换成固定数量的“视觉 Token”。
 - **Gated Cross-Attention (门控交叉注意力)**：这是 Flamingo 的精髓。它在原有的 LLM 层之间插入了新的层，让文本在生成时能够“注意到”视觉信息。
- **主要特点**：
 - **少样本学习 (Few-shot Learning)**: 它是第一个展现出极强少样本能力的 VLM。你给它看两三组“图片+描述”的例子，它就能学会这种任务模式。
 - **支持多图和视频**: 它可以处理一个对话中交替出现的图片和文本。

2. BLIP-2 (由 Salesforce 提出)

BLIP-2 是 2023 年发布的，它在 Flamingo 的基础上进一步解决了“**效率**”和“**对齐**”的问题。

- **核心思路**：视觉特征（来自图像编码器）和文本特征（来自 LLM）就像两种互不相通的语言。BLIP-2 设计了一个非常高效的“翻译官”——**Q-Former**，专门负责把视觉信息翻译成 LLM 听得懂的话。
- **关键组件: Q-Former (Querying Transformer)**
 - 这是一个轻量级的 Transformer。
 - **第一阶段 (预训练)**：它先学习如何用一小组“查询向量 (Queries)”去图片里提取最有用的特征。
 - **第二阶段 (对齐)**：它学习如何把提取出的特征“喂”给 LLM，让 LLM 觉得这些视觉特征就像是它熟悉的单词 Token 一样。
- **主要特点**：
 - **极高的效率**: BLIP-2 的可训练参数非常少（主要在 Q-Former），而图像编码器和 LLM 都是完全冻结 (Frozen) 不动的。
 - **性能强劲**: 尽管参数量比 Flamingo 小很多，但在很多视觉问答任务上表现更好。

3、LLaVA 的核心贡献是：它第一次将“**指令微调 (Instruction Tuning)**”成功应用到了多模态领域。它向世人证明了：**只要用一个简单的线性投影层，就能把强大的开源 LLM（如 LLaMA）转变为一个能看图、能聊天、能推理的视觉模型。**

SayCan（全称 **DoAsISay-CanAsIDo**）是由 Google Research 和 Everyday Robots 团队在 2022 年提出的一种结合了**大语言模型 (LLM) *与*机器人控制算法**的框架。

简单来说，它的核心目标是让机器人既能“听懂”人类复杂的自然语言指令（例如：“我把饮料洒了，你能帮帮我吗？”），又能根据自己当下的“能力范围”和“环境状态”去执行任务。

1. 核心公式：Say + Can

SayCan 的逻辑可以用一个非常直白的公式来表达：

$$\text{Probability} = P(\text{Say}) \times P(\text{Can})$$

- **Say (语义规划 - LLM)**：LLM 负责分析人类的指令，并将其拆解为一系列逻辑步骤。比如听到“打扫洒掉的水”，LLM 会认为“找海绵”或“拿纸巾”在语义上是高概率的后续动作。
- **Can (感知落地 - 价值函数)**：这部分由强化学习 (RL) 中的**价值函数 (Value Function)** 负责。它告诉机器人：在当前环境下，我真的能做到这件事吗？比如，如果海绵在另一个房间，或者机器人没有手拿纸巾，那么这些动作的 $P(\text{Can})$ 就会很低。

最终决策：机器人会选择那个“语义上合理”且“物理上可行”的动作。

2. 为什么 SayCan 很重要？

在 SayCan 出现之前，将 LLM 应用于机器人主要面临两个难点：

- **** LLM 不接地气 (Grounding) ****：语言模型可能建议你去“飞到天花板上擦灰”，它懂语言逻辑，但不了解物理规律或机器人的硬件限制。
- **** 任务太长容易失败 ****：机器人执行长序列任务 (Long-horizon tasks) 时，一个小错误就会导致全盘崩溃。

SayCan 通过将 **LLM 的高层逻辑规划** 与 **底层的技能原子库** 结合，让机器人具备了初步的“常识”和“自我认知”。

3. 一个实际例子

如果你对机器人说：“我刚运动完，能给我来点健康的零食和饮料吗？”

1. **LLM (Say)**：提炼关键词。它会觉得“拿苹果”、“拿矿泉水”比“拿薯片”、“拿可乐”的得分更高。
2. **机器人感知 (Can)**：机器人扫描厨房，发现苹果在桌子上（容易拿），矿泉水在冰箱里（需要开门，较难），而冰箱里其实已经没水了（无法完成）。
3. **综合判断**：机器人最后会决定：“既然水没了，拿苹果最稳妥”，于是它走过去抓起了苹果。

4. 局限性

虽然 SayCan 在当时非常惊艳，但它也有局限：

- **依赖预定义的技能库**：机器人只能执行它已经学会的底层动作（如抓取、放置、移动）。
- **感知限制**：如果视觉系统没识别出物体，LLM 再聪明也没用。
- **计算开销**：在当时，实时调用大型模型并结合价值函数计算仍有一定延迟。

VLM 的基本架构通常由三部分组成：

- **视觉编码器 (Vision Encoder)**：通常是 ViT (Vision Transformer)，负责将图像像素转换为特征向量。
- **适配器/投影层 (Adapter/Connector)**：由于视觉特征空间与 LLM 的文本特征空间不同，需要一个线性层或小型 Transformer 模块进行“翻译”。
- **大语言模型 (LLM)**：接收文本 Token 和处理后的视觉 Token，输出最终的文本响应。

VLA 的核心在于将机器人的**动作 (Action)** “词表化”。

- 动作令牌化 (Action Tokenization):** 机器人的关节角度、末端位姿等连续数值被离散化，映射成类似于单词的 Token。
- 统一预测:** 模型输入是“图像 + 任务描述 (文本)”，输出不再是文字，而是代表机器人下一步动作的 Token。
- 闭环控制:** VLA 运行在一个高频循环中，不断观察环境变化并输出下一跳动作。

VLA 是 VLM 的“进阶形态”。

- VLM 是大脑:** 它负责识别出“桌子上有一个红色的苹果”以及“红苹果通常比绿苹果更甜”。
- VLA 是大脑 + 小脑 + 神经:** 它不仅知道那是红苹果，还知道为了抓起这个苹果，自己的机械臂电机需要转动多少度，并在抓取过程中根据视觉反馈实时调整力度。

目前的趋势是利用 VLM 在互联网海量数据中习得的通用常识，通过**动作微调 (Action Fine-tuning)**，将其迁移到具体的机器人硬件上，从而解决机器人领域“数据匮乏”的难题。

流匹配 (Flow Matching, FM) 是近年来在生成模型 (Generative Models) 领域出现的一种新技术。简单来说，它是**扩散模型 (Diffusion Models)** 的一个更强、更快、数学上更简洁的替代方案。

如果说扩散模型是让数据在“迷雾”中随机漂移，那么流匹配就是为数据规划了一条“直达高速公路”。

1. 核心直觉：从“随机漂移”到“确定路径”

为了理解流匹配，我们可以对比一下它与扩散模型的区别：

- 扩散模型 (Diffusion):** 像是在水中滴入墨水。它通过添加随机噪声（高斯噪声）来打碎数据，然后训练模型学习如何“去噪”。这个过程是随机的、曲折的，因此推理（生成）时需要很多步才能还原出清晰的图像或动作。
- 流匹配 (Flow Matching):** 像是在地图上移动棋子。它定义了一个从“噪声分布”到“数据分布”的**向量场 (Vector Field)**。模型学习的是在每一个时刻，数据点应该以什么样的**速度 (Velocity) * 和 * 方向**移动，才能最快、最准地到达目的地。

2. 数学原理

流匹配的核心是学习一个由时间 t 参数化的速度场 $v_t(x)$ 。

- 概率路径 (Probability Path):** 我们定义一条路径 $p_t(x)$ ，将简单的噪声分布 p_0 （通常是高斯噪声）在 $t = 1$ 时转化为真实数据分布 p_1 。

- 常微分方程 (ODE):** 数据的演化遵循一个简单的方程：

$$\frac{dx}{dt} = v_t(x, t)$$

这意味着，只要模型学好了这个 v_t ，我们给定一个初始噪声 x_0 ，就可以通过数值求解（如欧拉法）一路“流”到目标数据 x_1 。

- 条件流匹配 (Conditional Flow Matching):** 在实际训练中，我们通常采用“条件流匹配”。其目标函数非常简单，就是最小化模型预测的速度与理想路径速度之间的差异：

$$\mathcal{L}_{CFM} = \mathbb{E}_{t, x_0, x_1, x_t} [\|v_t(x_t) - u_t(x_t | x_0, x_1)\|^2]$$

这里的 u_t 通常被构造为一条**直线**（即 Optimal Transport 路径），这是流匹配最高效的原因。

3. 流匹配的优势

为什么像 π_0 这样的机器人大模型放弃了扩散模型而选择流匹配？

- **生成速度极快：** 因为学习的路径几乎是直线的 (Optimal Transport)，流匹配通常只需要很少的步数 (甚至 1 步，即 Consistency Models 方向) 就能生成高质量结果。对于机器人控制 (50Hz+ 的实时性要求) 至关重要。
- **训练更稳定：** 相比扩散模型复杂的采样过程，流匹配的损失函数更直观，数学形式更简洁，更容易收敛。
- **路径平滑：** 在机器人领域，动作的平滑性意味着电机的损耗更小、运动更自然。流匹配生成的“确定性”轨迹比扩散模型的“随机性”轨迹要平滑得多。

4. 在机器人领域 (VLA) 的应用

在 π_0 这种 VLA 模型中，流匹配的作用如下：

1. **输入：** 视觉特征 (看到杯子) + 语言指令 (“拿杯子”)。
2. **预测：** 模型不是直接输出一个关节角度，而是预测一个**动作流的速度场**。
3. **执行：** 机器人根据当前状态，顺着这个“流”的方向移动机械臂。

这种方式让机器人不仅能完成任务，还能在遇到外界干扰 (比如人推了一下机械臂) 时，迅速根据当前的向量场重新找到通往目标的路径。

总结

流匹配技术是生成式 AI 从“黑盒随机模拟”向“确定性几何优化”进化的产物。 它是目前构建高性能具身智能 (Embodied AI) 大脑最前沿的技术选择之一。

1. 四旋翼无人机：4 自由度

虽然四旋翼机器人在三维空间中是一个 6-DoF 的刚体 (3个位置坐标 + 3个姿态角)，但它是一个典型的**欠驱动系统 (Underactuated System)**。

- **独立控制量：** 在中高层控制 (如 VLA 模型) 中，我们通常只独立给定它的 **位置** (x, y, z) 和 **偏航角** (Yaw, ψ)。
- **非独立量：** 滚转角 (Roll, ϕ) 和 俯仰角 (Pitch, θ) 是与水平加速度强耦合的。为了向某个方向移动，无人机必须倾斜。因此，在控制指令空间里，它们通常不作为独立的输入，而是由轨迹自动决定的。
- **计算：** $3(\text{Position}) + 1(\text{Yaw}) = 4 \text{ DoF}$ 。

2. Franka Panda 机械臂：8 自由度

你提到的 Franka Panda 原生具有 7 个旋转关节，但在实际的任务执行 (特别是模仿学习和 VLA 框架) 中，通常会将**夹持器 (Gripper)** 也计入动作空间：

- **关节空间：** 7 个独立旋转关节 (q_1 到 q_7)。
- **末端工具：** 1 个夹持器的开合状态 (Open/Close)。
- **计算：** $7(\text{Joints}) + 1(\text{Gripper}) = 8 \text{ DoF}$ 。

pi0-FAST (全称 π_0 -FAST) 是机器人初创公司 **Physical Intelligence (π)** 在 2025 年初推出的一款**视觉-语言-动作 (VLA)** 基础模型。

它是对原版 **pi0** 模型的重大演进。如果说 pi0 证明了模型可以像 GPT 拥有语言常识一样拥有“物理常识”，那么 **pi0-FAST** 的核心突破就在于“**快**”和“**精准**”。

1. 核心差异：从“扩散”到“自回归”

要理解 pi0-FAST，首先要对比它和原版 pi0 的区别：

- pi0 (Flow-matching/Diffusion):** 采用类似生成图像的“扩散”机制来产生动作。虽然动作平滑且质量高，但训练和推理速度较慢，难以处理极高频率的精细操作。
- pi0-FAST (Autoregressive):** 采用了**自回归 (Autoregressive)** 架构（类似 GPT 预测下一个词）。它直接预测动作序列，训练速度提升了约 **5 倍**。

2. 秘密武器：FAST Tokenizer

pi0-FAST 之所以能实现自回归且保持高精度，关键在于它引入了一种全新的动作编码方式——**FAST (Frequency-space Action Sequence Tokenization)**。

以往的 VLA 模型（如 RT-1）将动作直接离散化为简单的数字（Binning），这在处理高自由度或精细任务时会丢失大量信息。FAST 另辟蹊径：

- 频域压缩：** 借鉴了 JPEG 图片压缩的原理，使用**离散余弦变换 (DCT)** 将机器人的连续动作轨迹转换到频域。
- 稀疏量化：** 只保留对动作影响最大的频率分量，去掉细微噪声。
- 高效解码：** 这使得模型可以用极少的 Token 表示极其复杂的动作序列（如叠衣服、剥大蒜）。

3. 技术架构

pi0-FAST 的“底座”非常豪华：

- 视觉：** 使用 **SigLIP** 作为视觉编码器，理解复杂的物理场景。
- 大脑：** 内核是 **Gemma 2B**（Google 的开源大模型），负责逻辑推理和指令理解。
- 推理优化：** 支持 **KV-Caching**，这意味着在实时控制机器人时，响应延迟极低。