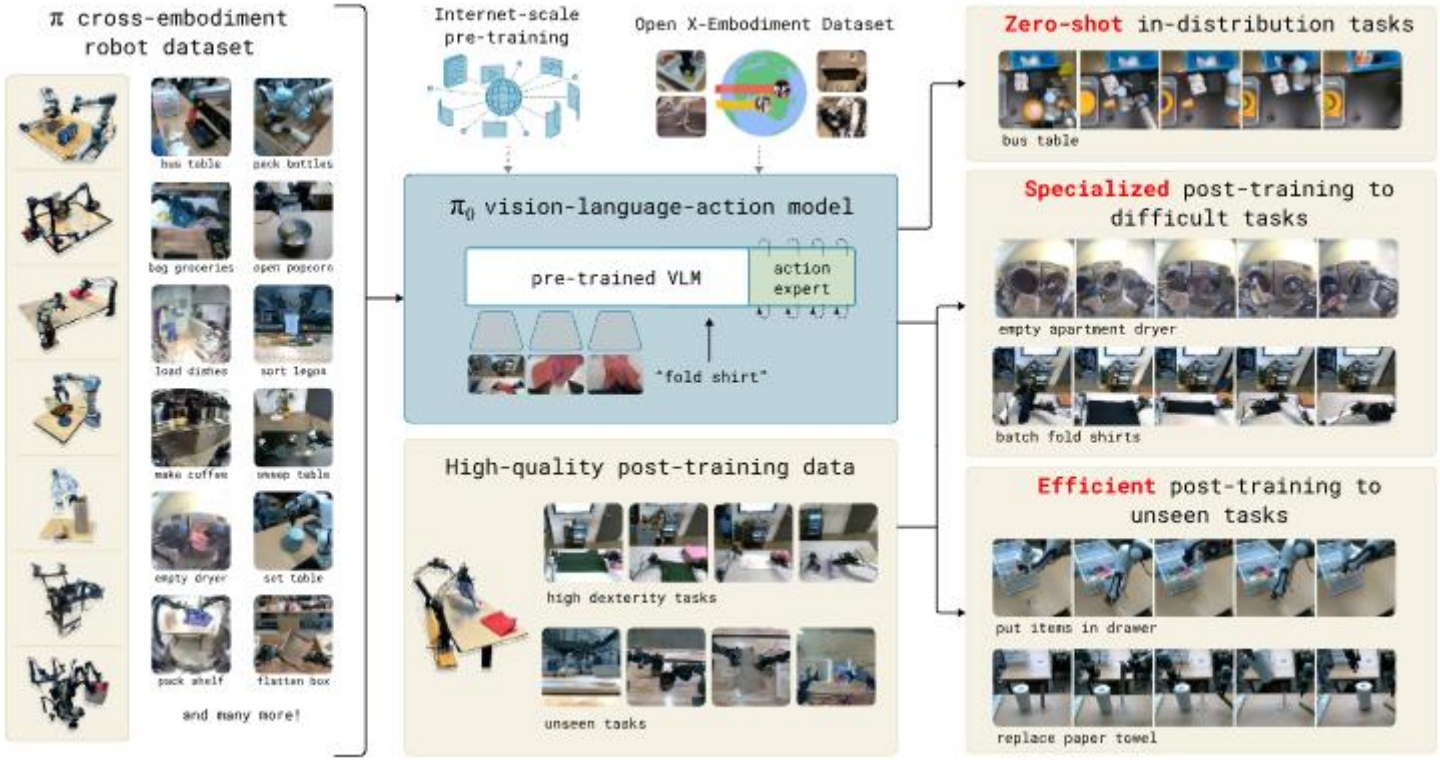
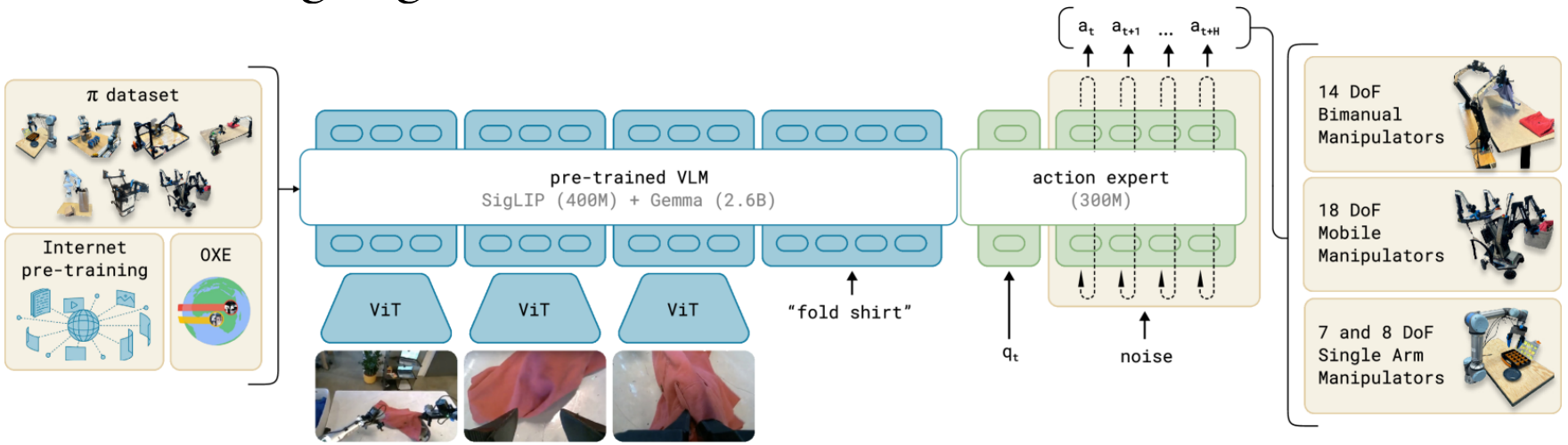
pi0: A Vision-Language-Action Flow Model for General Robot Control



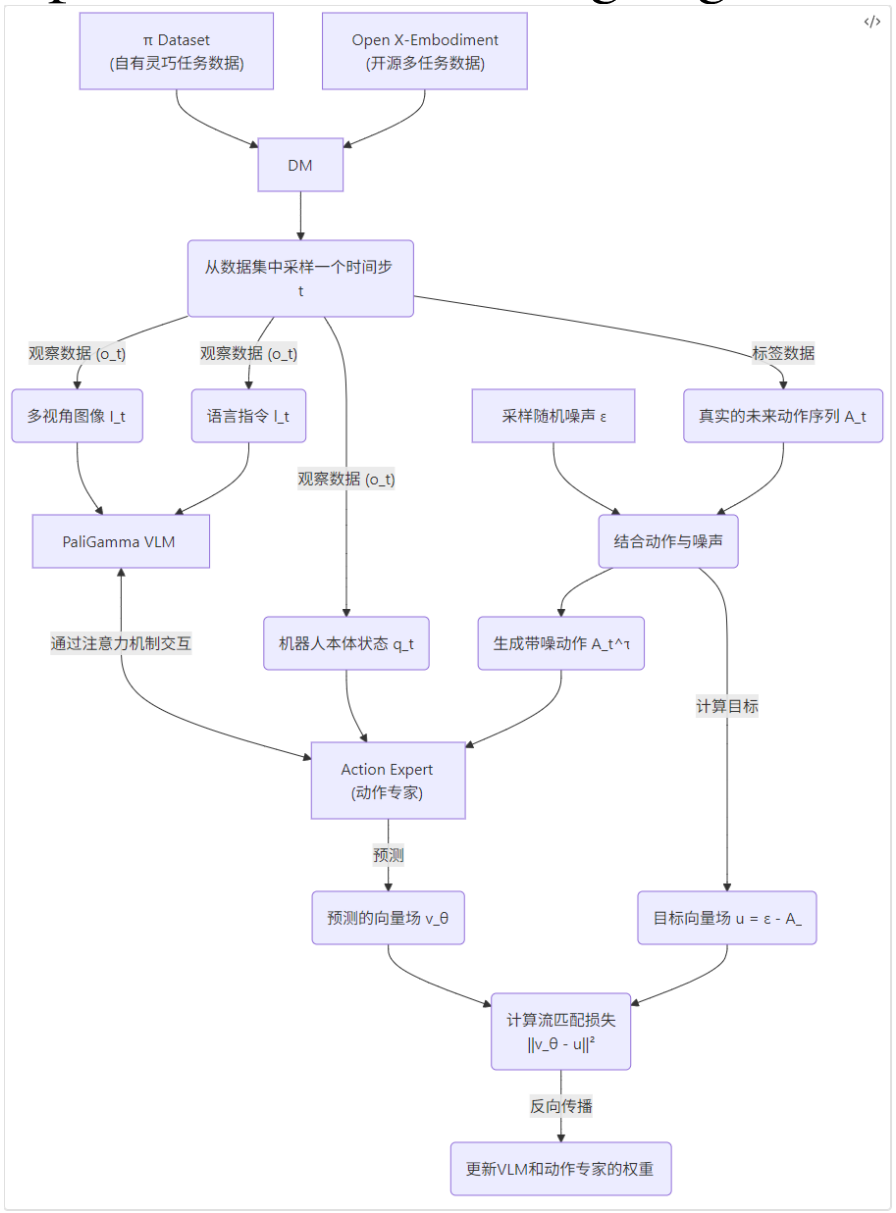
- 结合VLM和Flow Matching
- 可以做到长时间推理
- 有精细且高频的动作策略的同时, 让这个策略足够的泛化

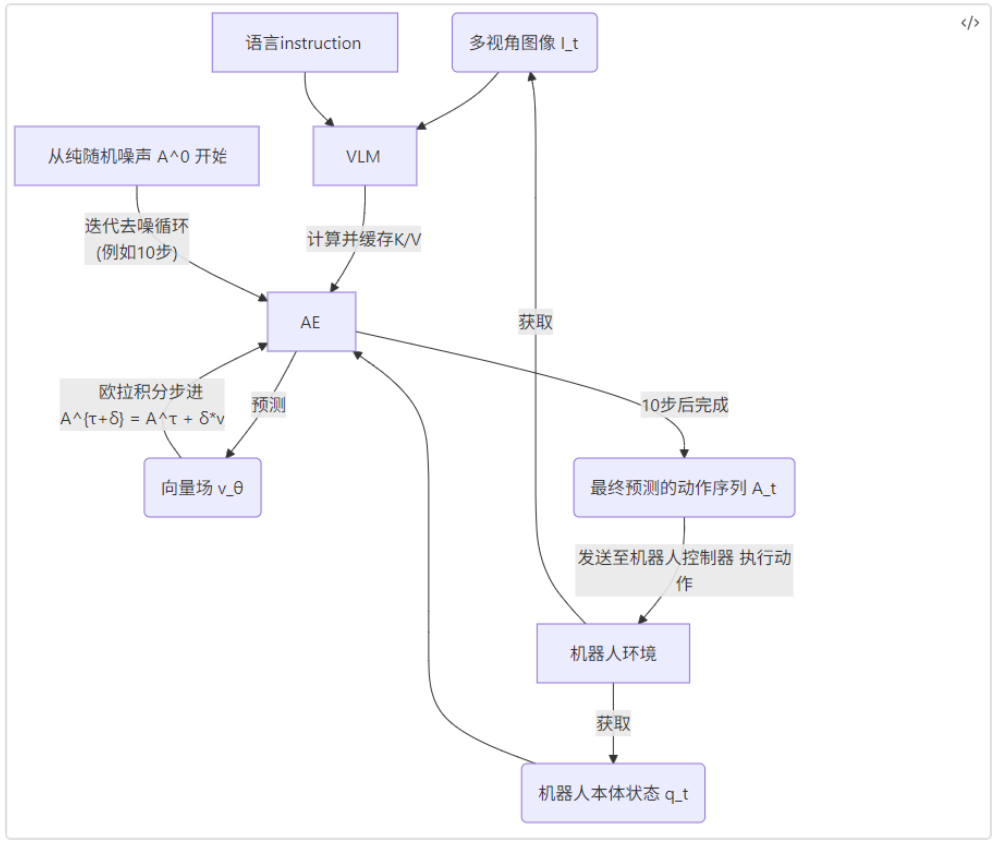
pi0: A Vision-Language-Action Flow Model for General Robot Control



- 模型:
  - VLM部分, 输入images和language instruction, 输出一个action token作为flow matching的condition
  - action expert部分接受最终输出的action token, 以及当前的状态q, 然后进行denoise生成一个action
- 数据:
  - pi0 自己收集的dataset
  - OXE dataset
- pre-train stage: general policy
- post-train stage: specific downstream task

pi0: A Vision-Language-Action Flow Model for General Robot Control

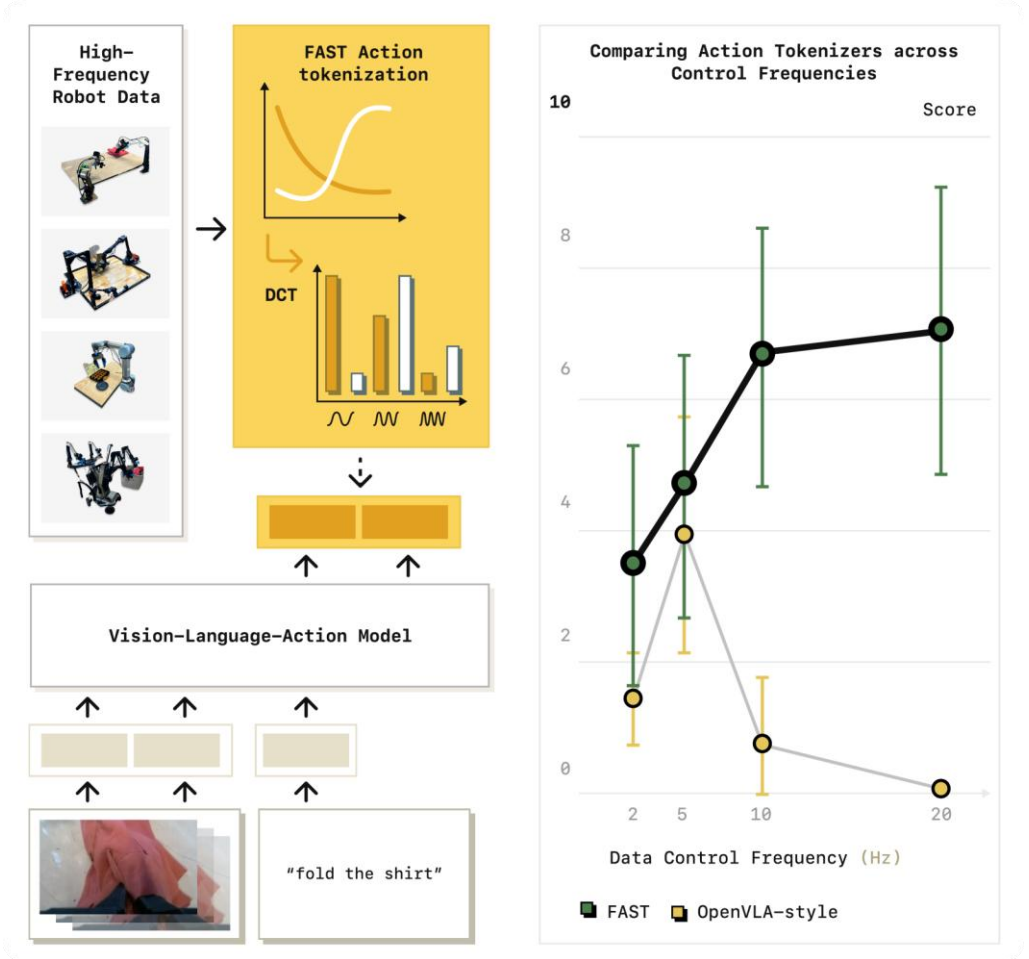




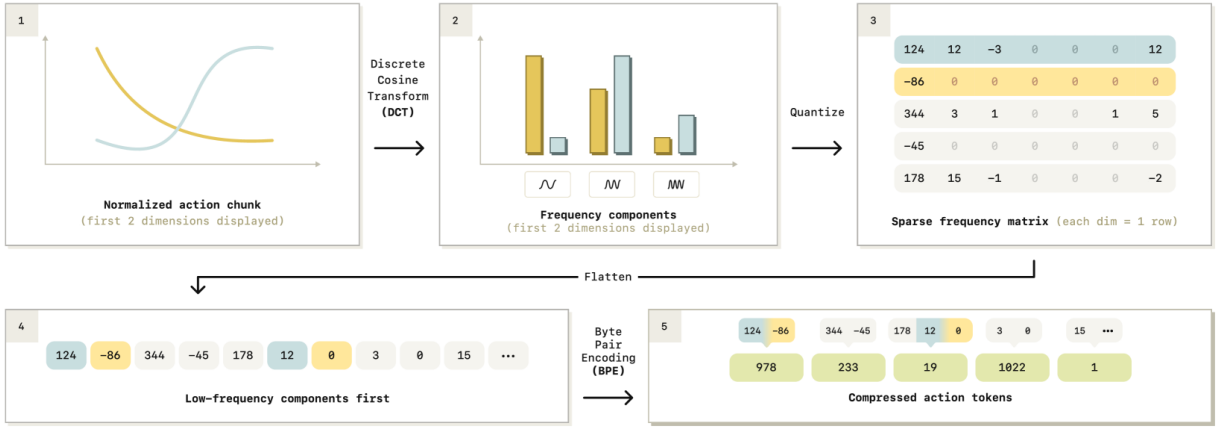
FAST: Efficient Action Tokenization for Vision-Language-Action Models

- 问题:
  - 对于action chunk的预测, 每一个chunk内的action高度相关, 因此一个chunk可能会生成完全相同的action
  - 这样是比较差的local optima

- 解决:
  - 使用离散token
  - 用DCT转换
  - 在解决问题的基础上, 有更好的training efficiency, 并且表现和连续的action不相上下



FAST: Efficient Action Tokenization for Vision-Language-Action Models



- DCT: discrete cosine transform 离散余弦变换

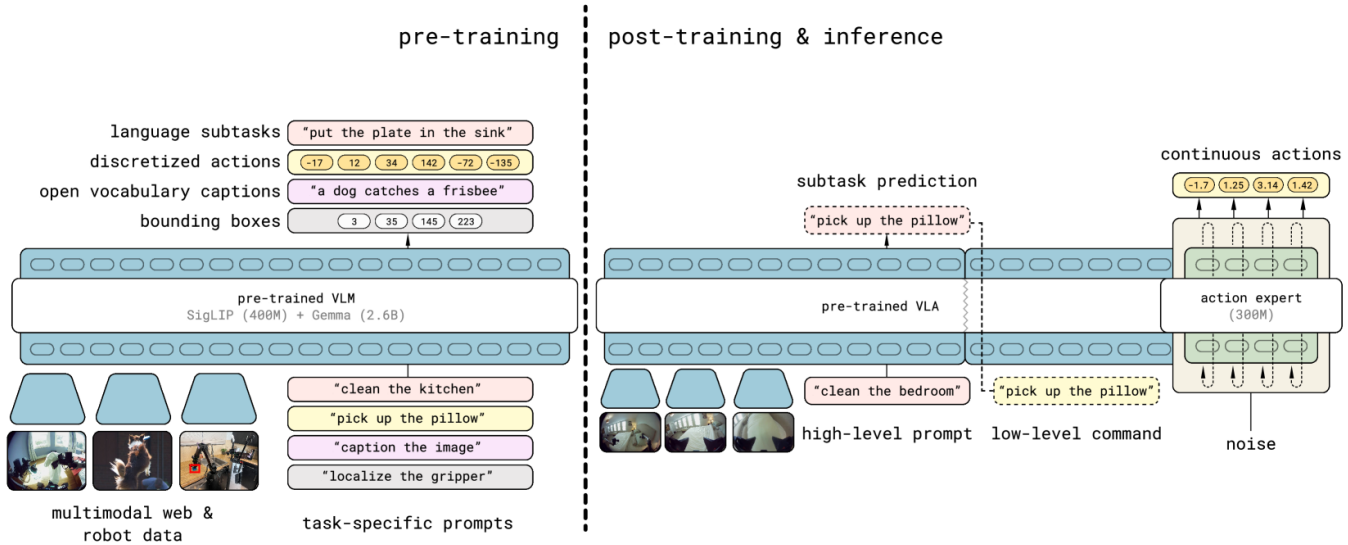
- FAST:
  - Normalize
  - DCT
  - quantize
  - BPE tokenize

pi0.5: a Vision-Language-Action Model with Open-World Generalization

- 继承自pi0

- 分成了high-level的subtask instruction以及low-level的action expert

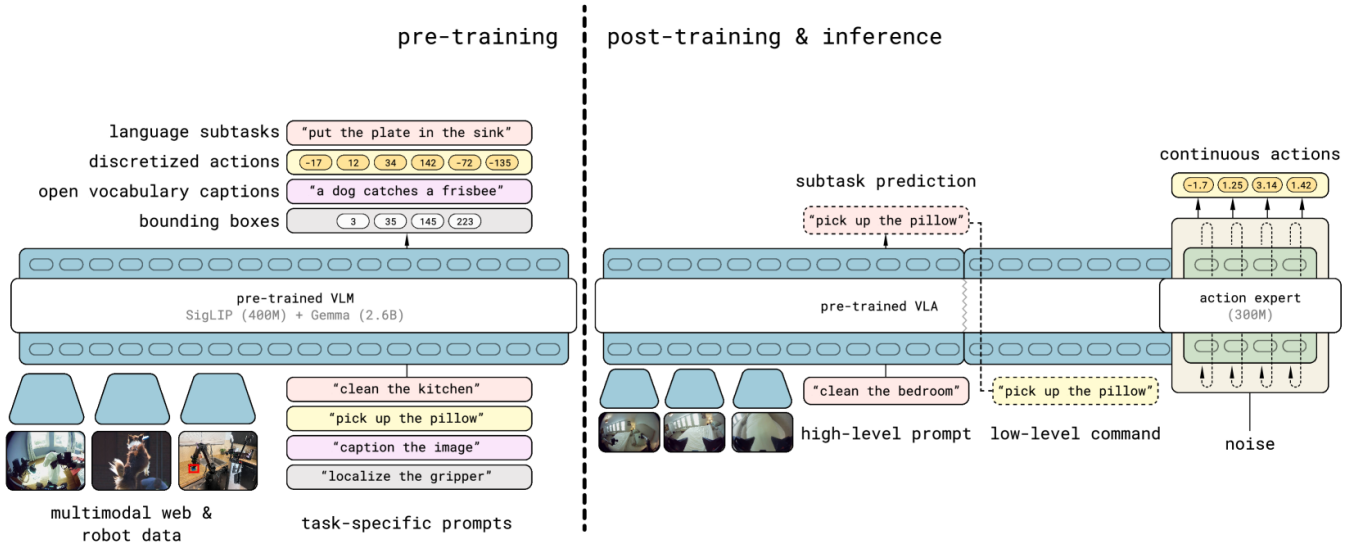
pi0.5: a Vision-Language-Action Model with Open-World Generalization



- training:
  - pre-training: 在不同数据中训练VLA, 生成high-level的subtask的分解
  - post-training: 根据high-level semantic subtask指导flow matching生成action

- inference:
  - 根据场景信息和任务结构预测semantic subtask
  - 根据subtask预测robot的low-level action

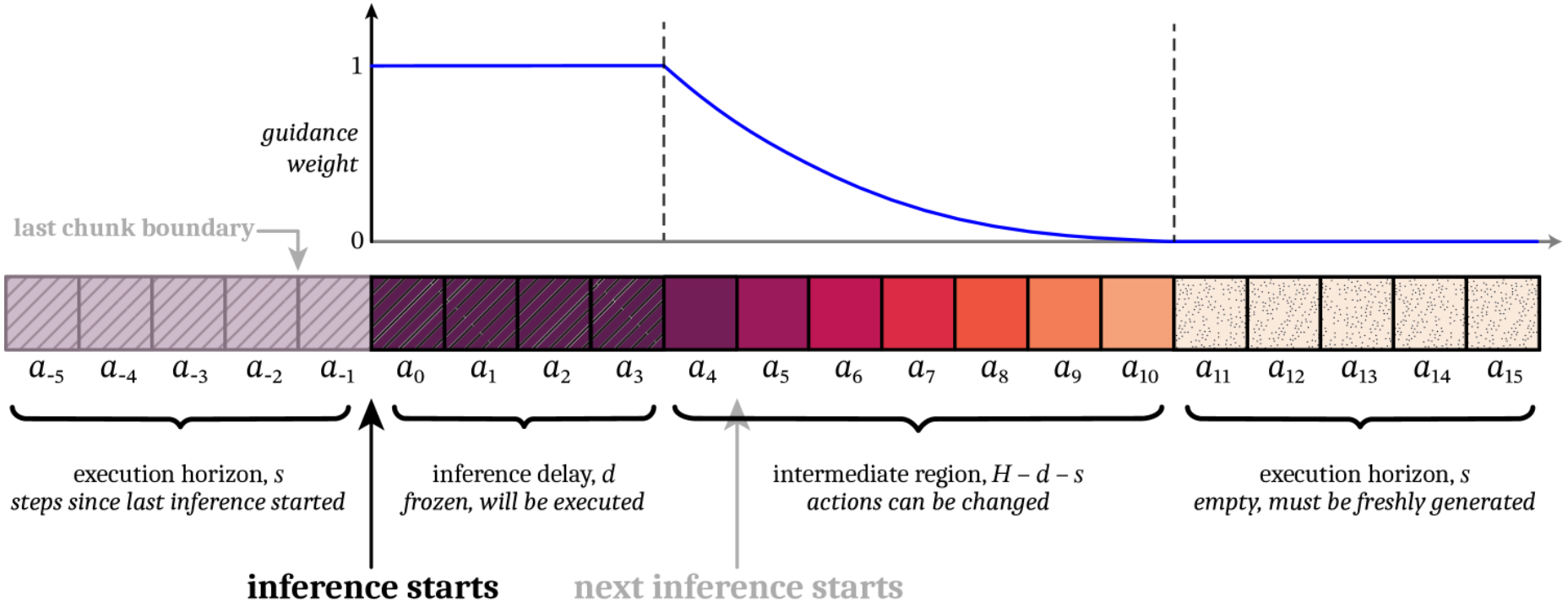
pi0.5: a Vision-Language-Action Model with Open-World Generalization



- Training stage:
- 使用FAST转换成离散的token
- 两个阶段, 使用同一个Loss函数, 使用超参数控制flow matching是否训练: $\mathcal{L} = \mathbb{E}_{D,\tau,\omega}[H(x_{1:M}, f_\theta^l(o_t, l)) + \alpha\|\omega - a_{t:t+H} - f_\theta^a(a_{t:t+H}^{\tau,\omega}, o_t, l)\|^2]$

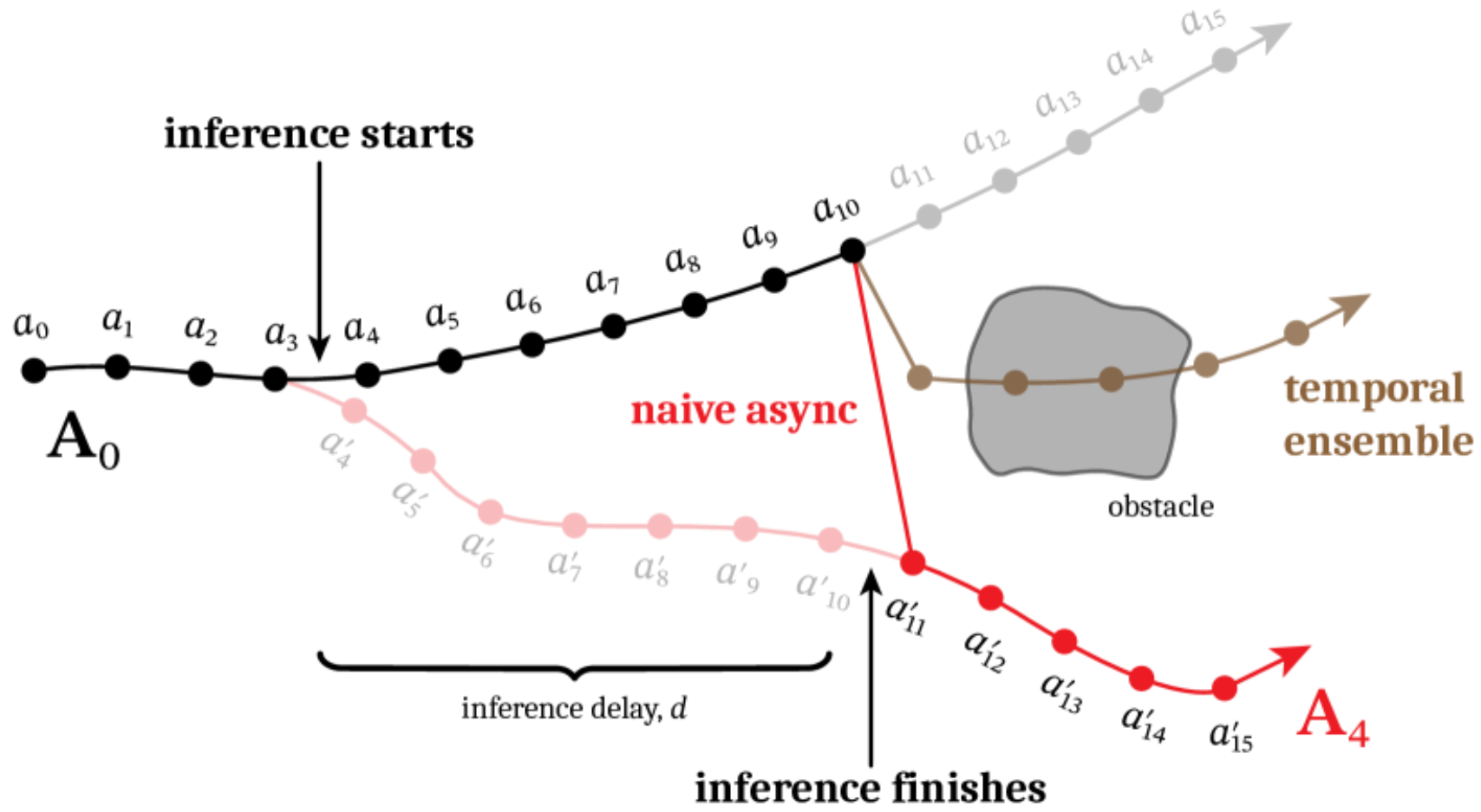
- Dataset:
  - 可移动机器人操作, 不可移动机器人操作, 在实验室中的操作, task 分解, web data, 等等
  - 类似FAST的方法, 将dataset中action归一化

Real-Time Execution of Action Chunking Flow Policies



- 是一个基本的独立的方法, 可以应用于任何action chunk的方法
- 问题: 两个chunk之间可能有突变, 推理延迟可能会导致卡顿

Real-Time Execution of Action Chunking Flow Policies



Real-Time Execution of Action Chunking Flow Policies

- 解决方案是使用Inpaint的方式, 让flow matching对action做重绘

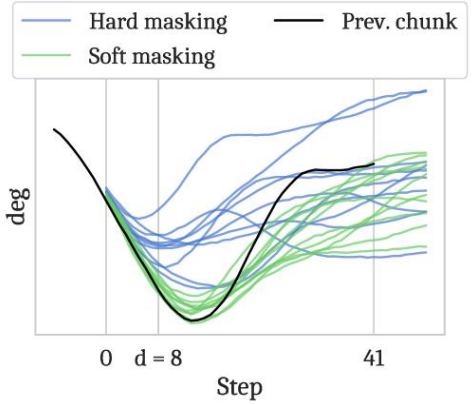
- 使用ΠGDM 的方法, 用train-free的方法进行inpaint. 因此denoise的公式变成了:

$$\mathbf{v}_{\Pi\text{GDM}}(A_t^\tau, o_t, \tau) = \mathbf{v}(a_t^\tau, o_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \text{diag}(\mathbf{W}) \frac{\partial \widehat{\mathbf{A}_t^1}}{\partial \mathbf{A}_t^\tau}$$

$$\widehat{\mathbf{A}_t^1} = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}(\mathbf{A}_t^\tau, o_t, \tau)$$

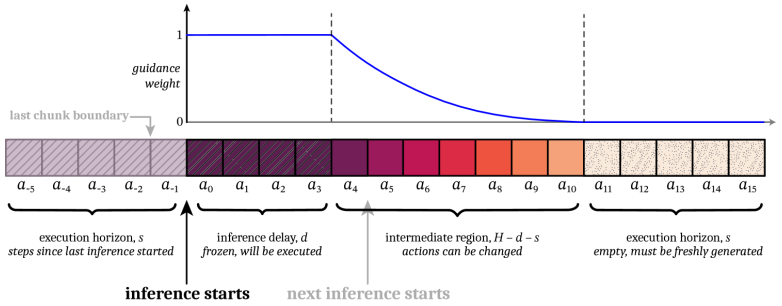
$$r_\tau^2 = \frac{(1-\tau)^2}{\tau^1 - (1-\tau)^2}$$

- mask有两种, hard mask和soft mask.
  经过实验比较, soft mask效果更好, 更贴近上一个action



Real-Time Execution of Action Chunking Flow Policies

- 计算:
  - flow matching进行原始的denoise
  - 复制上一个action的最后一部分, 到Y的最前面
  - 使用mask, 只关注需要重叠的部分



  - 计算vector-Jacobian product:
    - pytorch计算 $\widehat{\mathbf{A}_t^1} = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}(\mathbf{A}_t^\tau, o_t, \tau)$
    - pytorch计算 $\mathbf{J} = \left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \mathrm{diag}(\mathbf{W})$
    - 计算gradiant: guidance_term = torch.autograd.grad(L_pseudo, $\mathbf{A}_t^\tau$)[0]
    - 这里的guidance_term就是 $\left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \mathrm{diag}(\mathbf{W}) \frac{\partial \widehat{\mathbf{A}_t^1}}{\partial \mathbf{A}_t^\tau}$