Behavioral Cloning

ESE 6510

Antonio Loquercio

Philadelphia, 1957

Behavioral Cloning: Agenda

- Theoretical Foundations
- Tools for Data Collection
- **Algorithms**
- Leveraging foundation models
- Challenges

Behavioral Cloning: Algorithms

- General optimization problem

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i | s_i)$$

- Instantiation example:

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \| a_i - NN_\theta(s_i) \|_2$$

  - we assume that π_θ (the conditional of actions with respect to states) is a gaussian with fixed variance and the prediction of a neural network (NN_θ) conditioned on the state as mean.

Behavioral Cloning: Algorithms

- General optimization problem

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

- Things we can decide:
  - **How to collect data?**
  - How to represent the input?
  - How to represent the actions?
  - How to model the training process?

Data Collection Algorithm: DAGGER

- What we can optimize (ρ_{π^*} is the trajectory distribution of the expert):

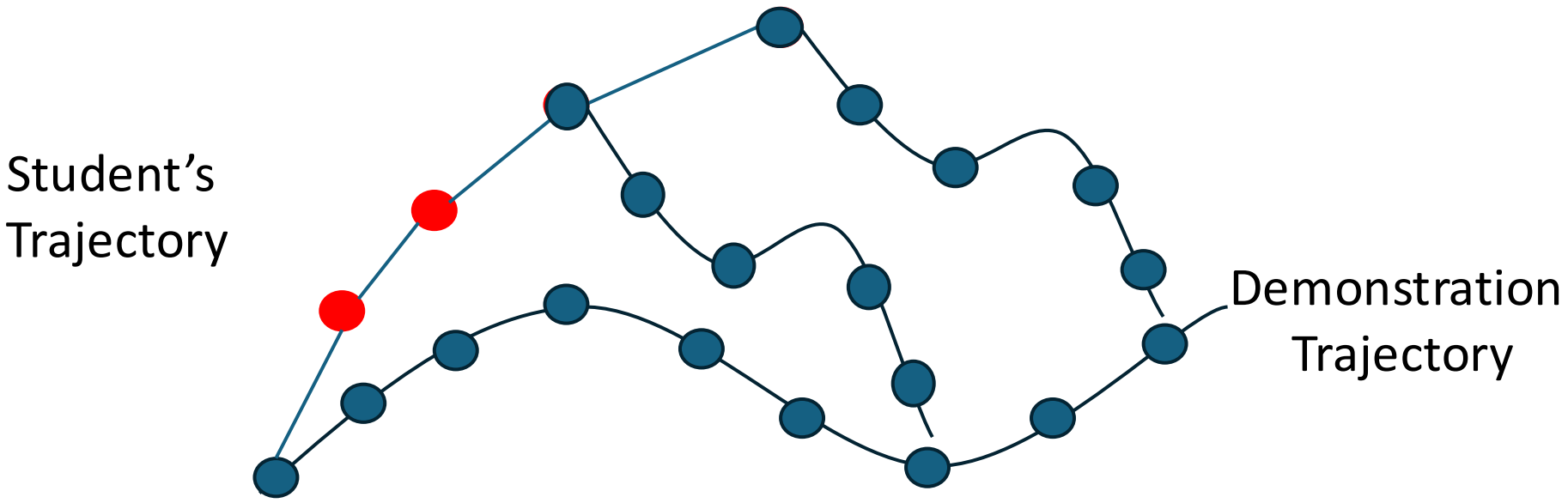
$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

- What we should to optimize (ρ_{π_θ} is the trajectory distribution of the student):

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi_\theta}} \log \pi_\theta(a_i|s_i)$$

Data Collection Algorithm: DAGGER

- Very simple idea: let the student control the robot, and collect demonstrations **from the states that the student observed,** not from the states that the expert observes.



Data Collection Algorithm: DAGGER

- Very simple idea: let the student control the robot, and collect demonstrations **from the states that the student observed,** not from the states that the expert observes.

- This process is called dataset aggregation (Dagger). Improved theoretical bound on performance (linear in the episode length).

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Adapted from S. Levine

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, 2010

Data Collection Algorithm: DAGGER

- What's the problem with DAGGER?

- You need to ask a human to recollect labeled data after each training step!

- Solution: Batch-Dagger.

1. Train a model with expert data
2. Run experiments to see where the student fails
3. Collect data in that region of the state space

- Done in practically all BC papers today (either consciously or unconsciously)

Behavioral Cloning: Algorithms

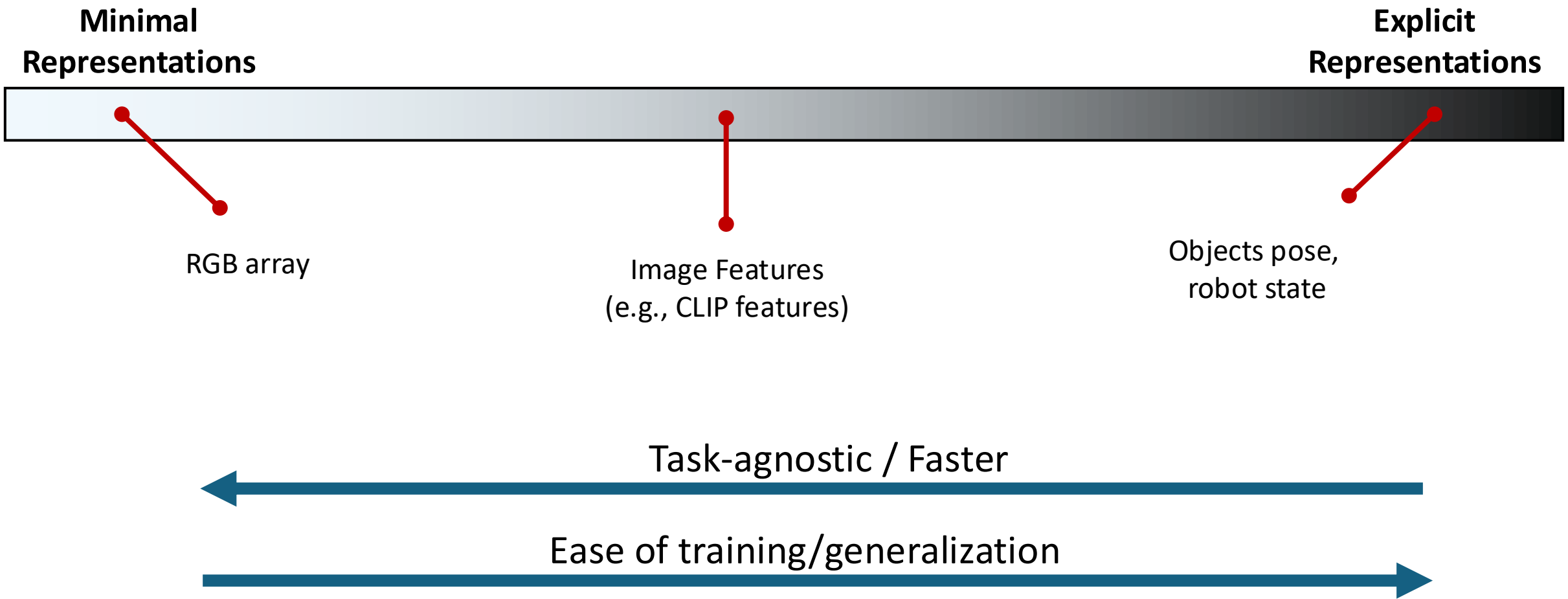
- General optimization problem

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

- Things we can decide:
  - How to collect the data?
  - **How to represent the input?**
  - How to represent the actions?
  - How to model the training process?

Input Representations

- **Remark**: We never process “raw” data. There is always some processing of sensory data before it is fed to a decision module.



Minimal Input Representations

a_t

- Advantages:
  - General and simple (often a ResNet-18 or VIT-Small)
  - Trained specifically for the task(s) of interest
- Disadvantages:
  - Large sample complexity
  - Requires a high-capacity (and potentially high-latency) NN

Intermediate Representations

(Pre-trained) Encoder z_t Fancy NN a_t

- Popular Encoders:
  - (Dense image features): CLIP, SigLIP, Dyno, …
  - (Sparse image features): SIFT, SURF, …
  - People often use the pre-trained image encoders even for non-visual data (e.g., tactile).
  - (V)AE trained on unlabeled robot data.
  - Depth/Optical Flow, etc.

Intermediate Representations: An Example

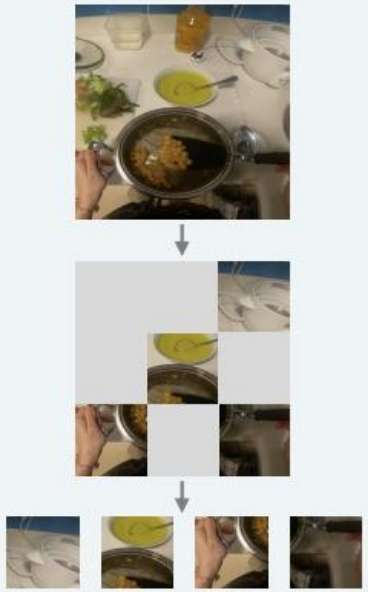
In-the-Wild Data

Over *4.5 million* images
Five diverse data sources

Masked Autoencoder

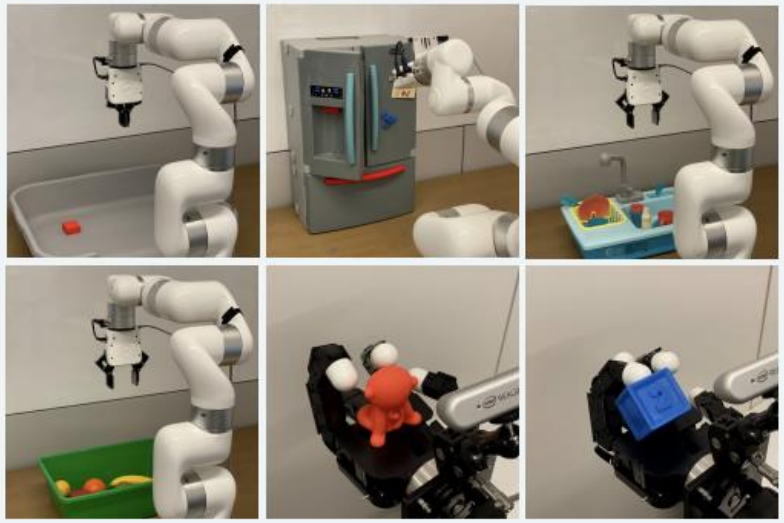
(a) Masking

(b) Autoencoder



Real-World Robotic Tasks

Two robots (xArm, Allegro hand)
Eight tasks (scenes, objects)



Real-World Robot Learning with Masked Visual Pre-training, 2022

Intermediate Representations

(Pre-trained) Encoder z_t Fancy NN a_t

- Advantages:
  - Simple and effective
  - Inherits robustness from pre-training (e.g., illumination invariance)
  - Most popular alternative for VLA-style networks
- Disadvantages:
  - Pre-trained encoders might not capture the features you need for the task
  - Still large sample complexity

Explicit Representations

Sensing Module

Obj. Pose,
Robot pose,
obstacles,
etc.

Fancy NN

a_t

Explicit Representations

Reconstructing Hand-Held Objects in 3D; J. Wu, G. Pavlakos, G. Gkioxari, J.Malik. Arxiv 2024.

Explicit Representations

- Advantages:
  - Sample efficient if well designed
  - Robust to input variations (if the sensing module is good enough)
  - Interpretable (if that's important for your task)
- Disadvantages:
  - Often quite specific for the task. Multi-task explicit representations are hard to design.
  - Highly reliant on the (separately trained) sensing module.
    - Latency
    - Compute
    - Failure cases

Recap: Input Representations

- Minimal (e.g., RGB Array)
- Intermediate (e.g., a pre-trained vision encoder like CLIP)
- Explicit (e.g., estimated object pose)

Behavioral Cloning: Algorithms

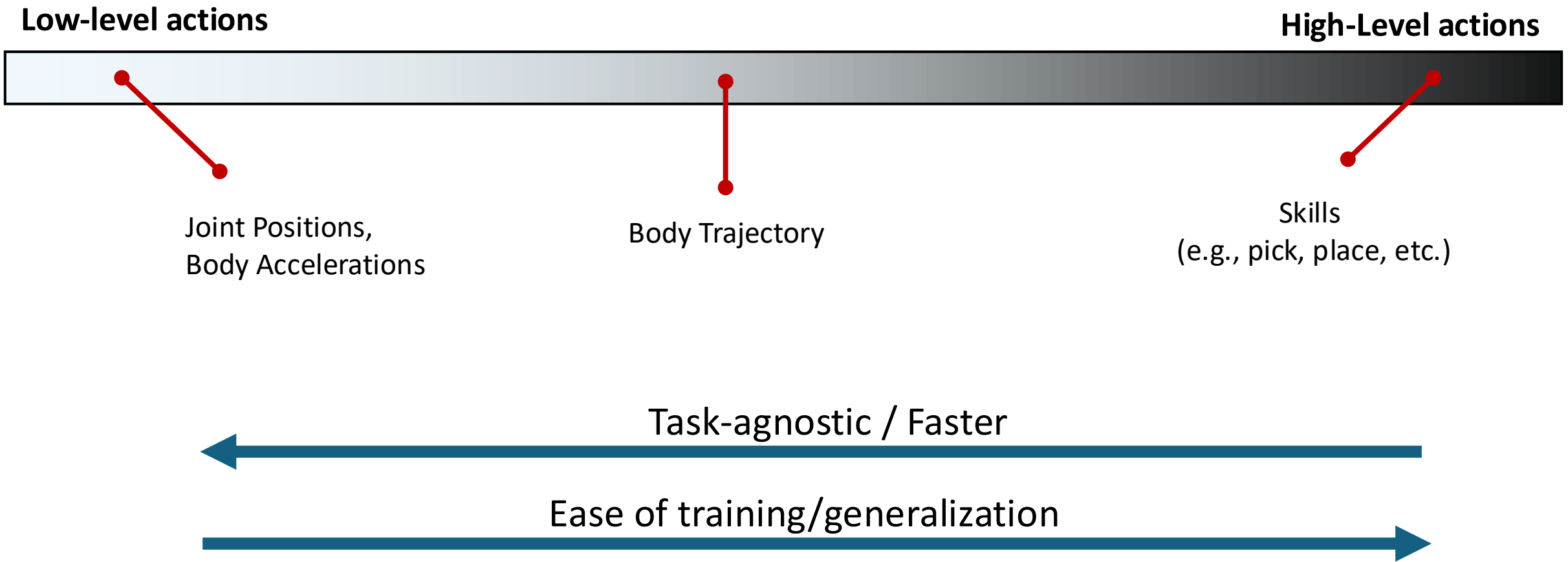
- General optimization problem

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

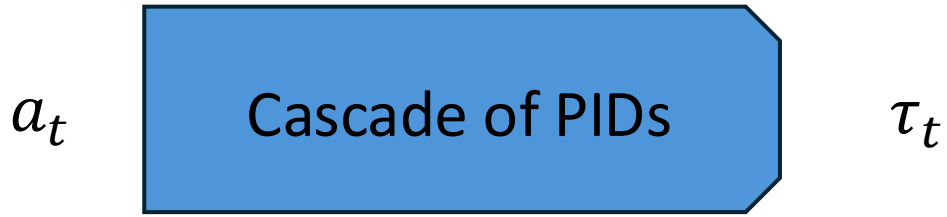
- Things we can decide:
  - How to collect the data?
  - How to represent the input?
  - **How to represent the actions?**
  - How to model the training process?

Action Representations

- **Remark**: We (almost) never output direct motor torques. There is often some layer of low-level controllers after a policy. The more high-level, the more complex the controllers need to be.

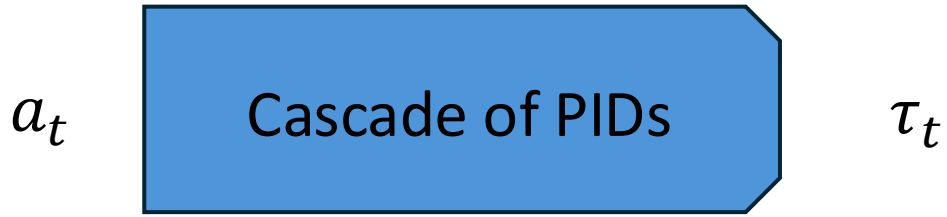


Low-Level Action Prediction



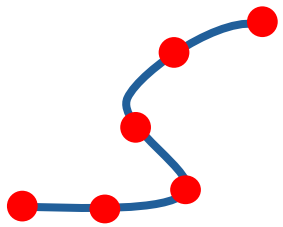
- Generally, joint position (e.g., for manipulators) or body accelerations (e.g., for drones)
- A bit more complicated than it looks. The cascade of PID controllers needs to be well-designed to function effectively.
- Often, we use more fancy controllers than a PID:
  - Adaptive controllers (if some parameters are unknown or hard to model)
  - Impedance controllers (very common in manipulation)

Low-Level Action Prediction



- Advantages:
  - General and simple
- Disadvantages:
  - Large sample complexity
  - Policy needs to be very fast to control at such low-level
- Currently, this works well for RL, but it's not as popular in BC.

Mid-Level Action Prediction



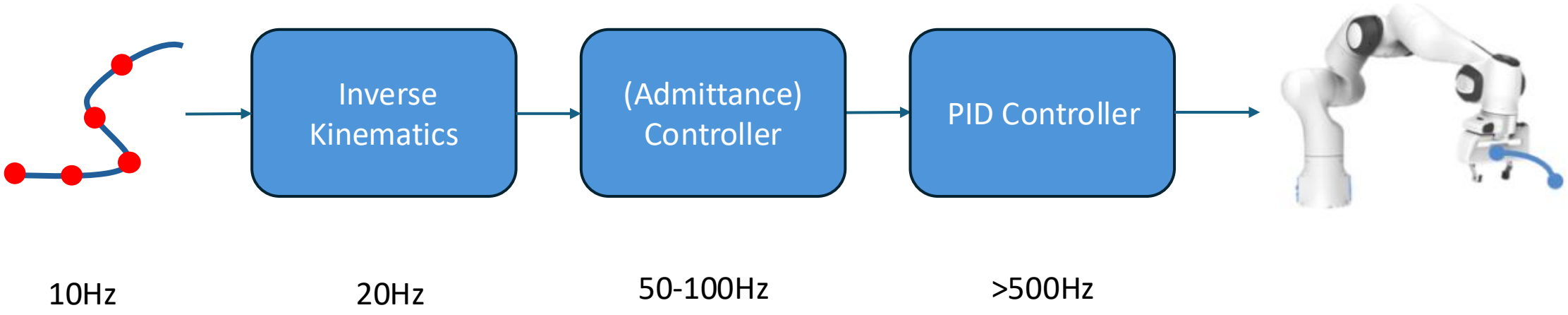
Controllers Cascade

$\tau_{t:t+K}$

- There is a lot of leeway in the definition of these actions:
  - Joint position sequences
  - Position/Velocity sequences of end effectors

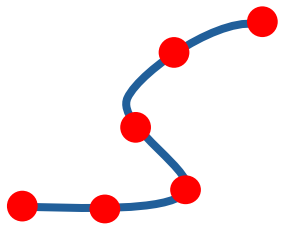
- The controllers' complexity increases…

Mid-Level Action Prediction: An example



- All blocks must be properly designed/tuned.
- Side note: these are the same blocks that are in place during data collection (see last lecture's slides). Therefore, the operator can potentially account for some limitations of the controllers.

Mid-Level Action Prediction



Controllers Cascade

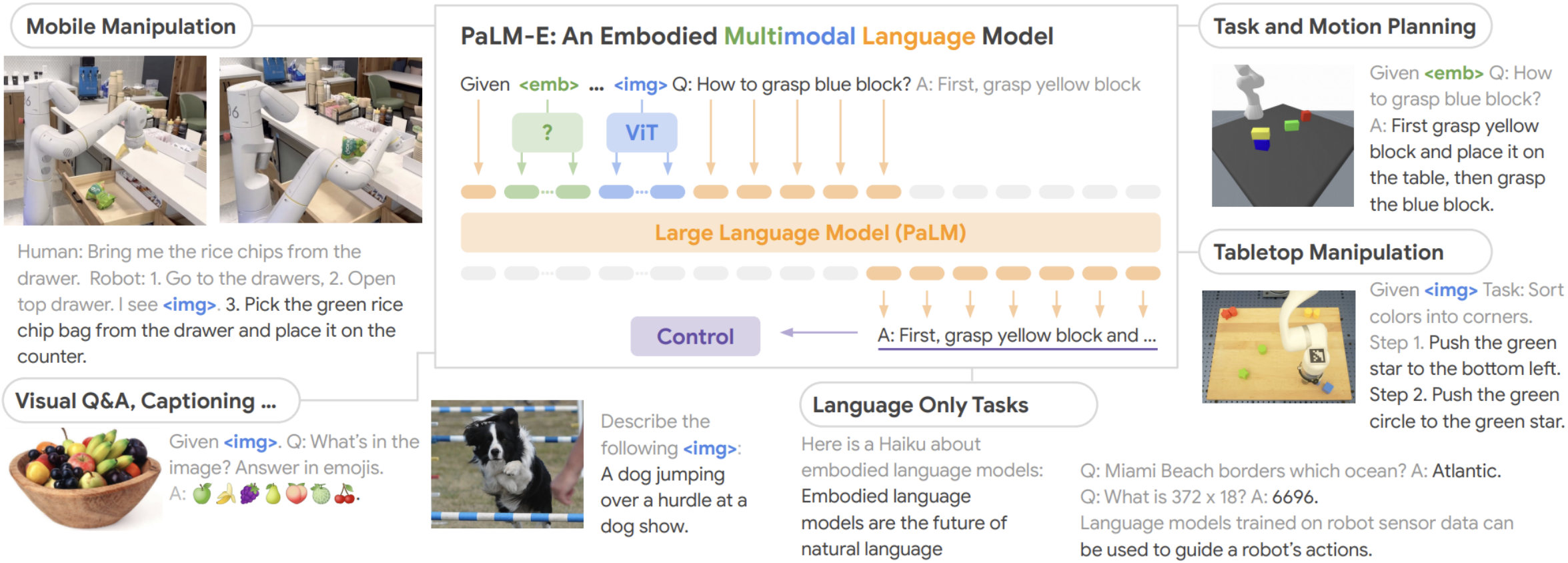
$\tau_{t:t+K}$

- Advantages:
  - Lowers the frequency at which the policy needs to operate.
  - Simplifies learning.
  - Easier to add constraints to the policy (e.g., a safety filter).
- Disadvantages:
  - Latency/Compute of the controllers.
  - Potentially bottlenecked by what the controllers can do.
- This is the most popular approach for BC right now.

High-Level Action Prediction

- The policy outputs a desired behavior (e.g., pick an apple), and the right skill is selected from a library to execute the desired behavior.
- The policy effectively works as a planner/state machine.
- The skill library comprises a set of lower-level policies, some of which could be trained using BC/RL or other optimization methods.

High-Level Action Prediction: An Example



PaLM-E: An Embodied Multimodal Language Model, 2023

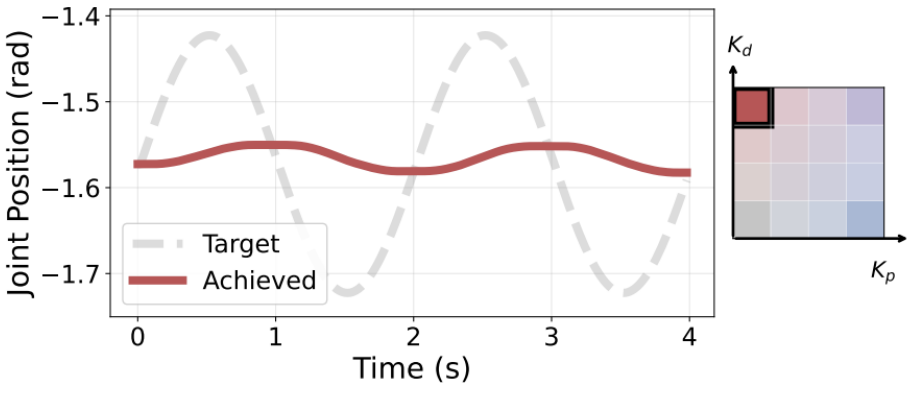
High-Level Action Prediction

- Advantages:
  - Policy can operate at a low frequency.
- Disadvantages:
  - If you don't have a skill for something, the policy can't do it or learn it.
- Most people don't call training these policies as BC. Often you need no training at all: you can use VLMs zero-shot for planning.

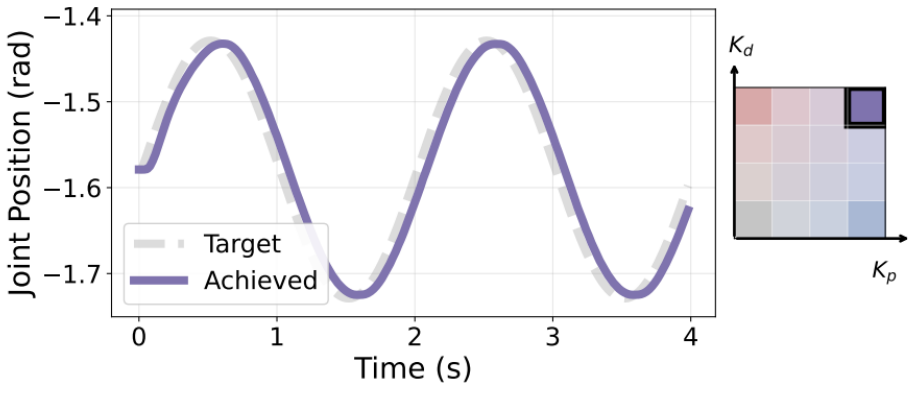
Recap: Action Representations

- Low-Level Control (e.g., body acceleration)
- Mid-Level Control (e.g., end-effector position sequence)
- High-Level Control (e.g., skills)

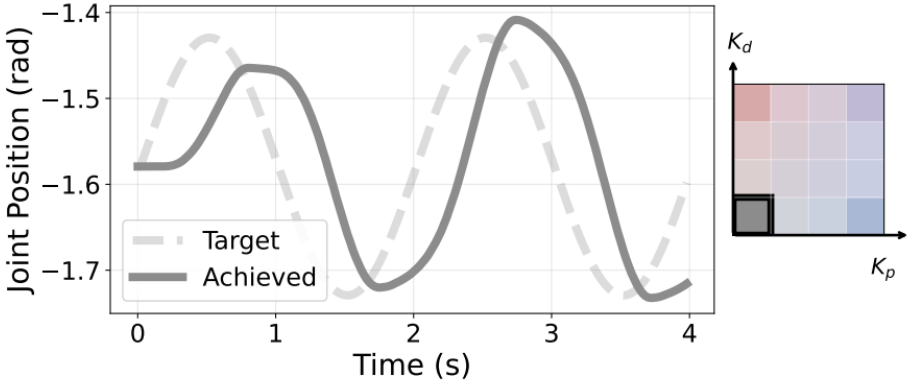
Action Representations: Control Tuning Matters



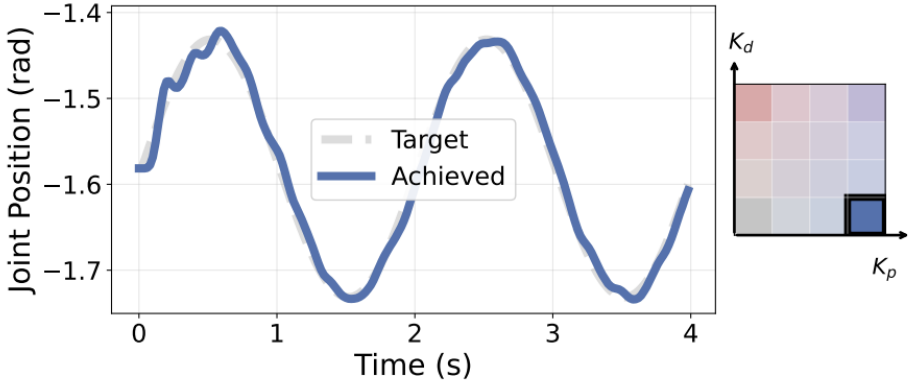
(a) Compliant and Overdamped (CO)



(b) Stiff and Overdamped (SO)



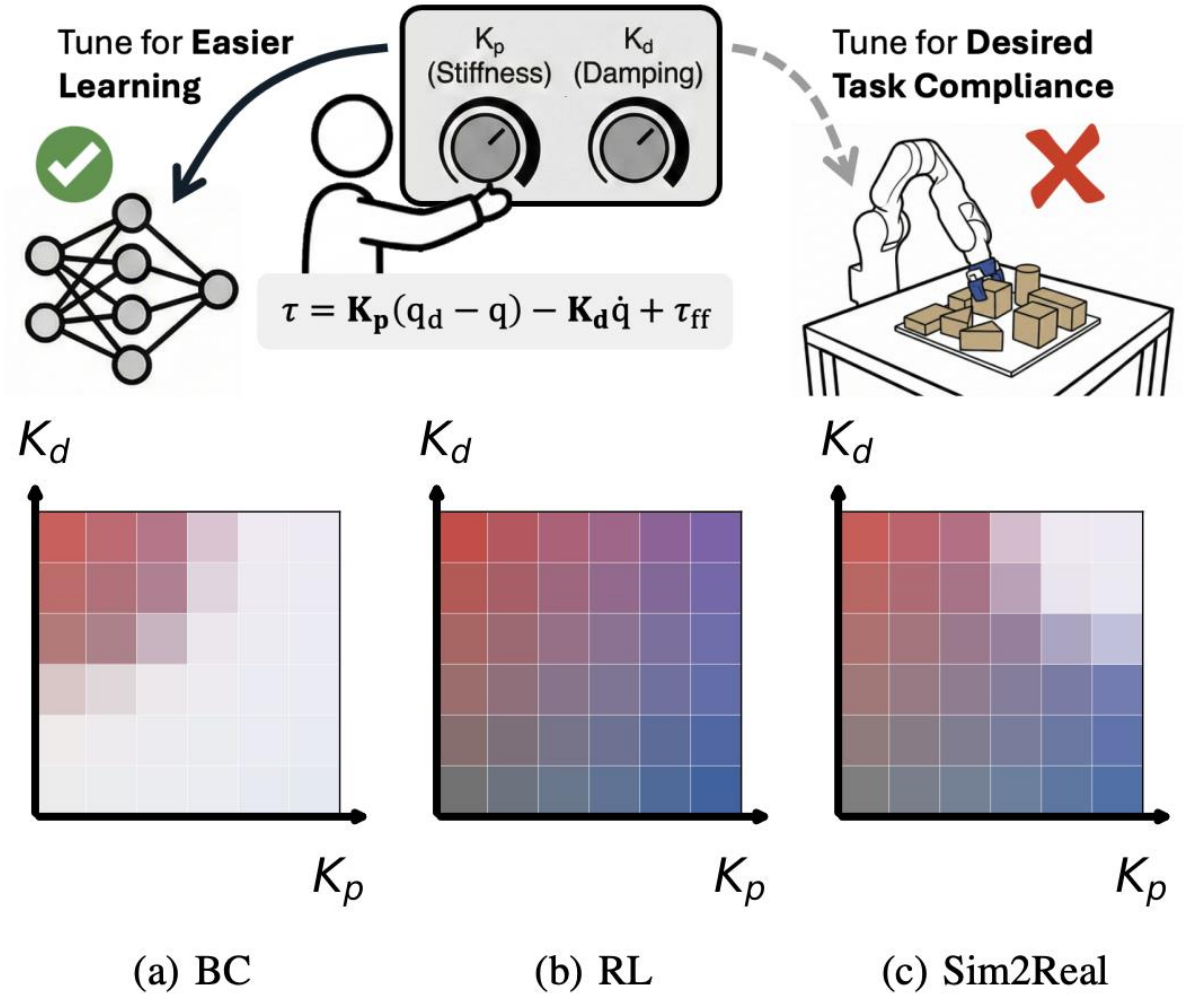
(c) Compliant and Underdamped (CU)



(d) Stiff and Underdamped (SU)

Tune to Learn: How Controller Gains Shape Robot Policy Learning, Bronars et al.

Action Representations: BC vs RL.



Tune to Learn: How Controller Gains Shape Robot Policy Learning, Bronars et al.

Behavioral Cloning: Algorithms

- General optimization problem

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

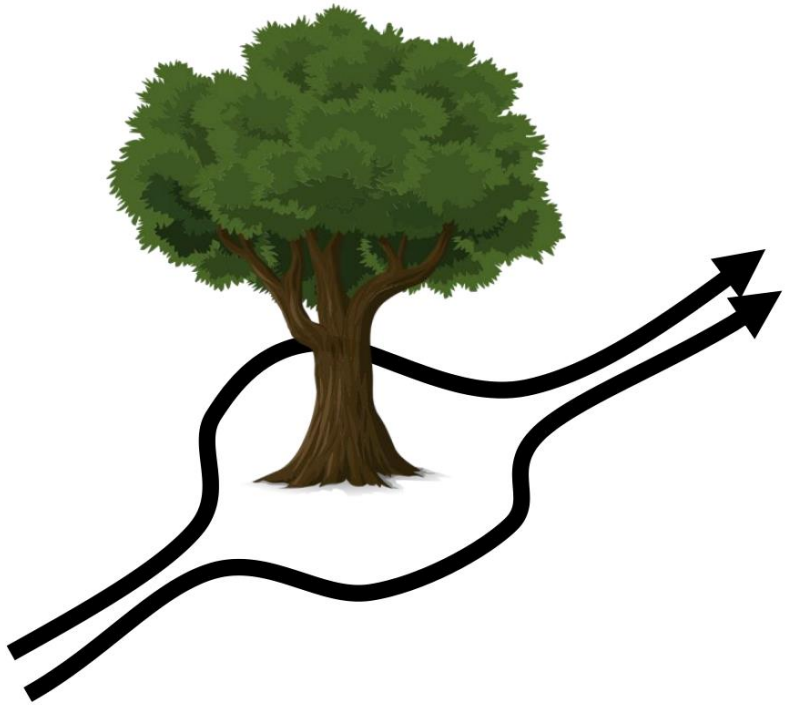
- Things we can decide:
  - How to collect the data?
  - How to represent the input?
  - How to represent the actions?
  - **How to model the training process?**

The Training Process

- $\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i | s_i)$
- $\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \| a_i - NN_\theta(s_i) \|_2$ for a particular action distribution choice. This makes it a traditional regression problem.

- Why isn't this always enough?

- Multi-modality in the demonstrations
- Multi-modality in the optimal policy



The Training Process

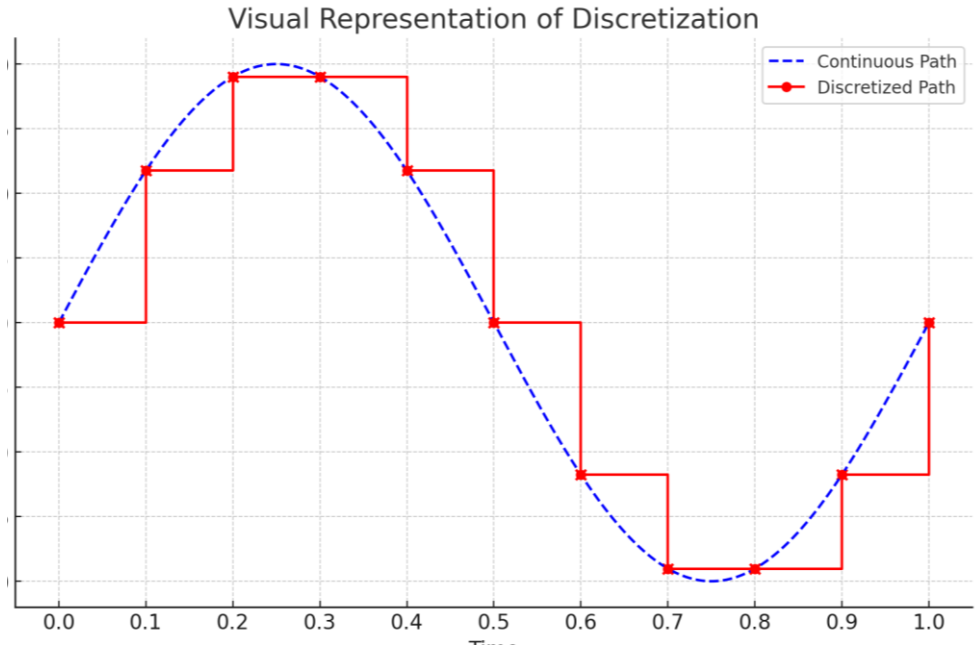
Possible solutions to the problem:

- Discretization of the action space

- Learn a more expressive distribution
  - Mixture of Gaussians
  - Diffusion
  - Latent actions

Action Space Discretization

- Simple idea: divide actions into bins.
- You don’t predict a continuous value, but a probability over bins.
- This automatically accounts for multi-modality.



- Common approach:
  - Uniform discretization with 256 bins **over each coordinate** (using the 1st and 99th quantile of the actions in the training data as the minumum and maximum values, respectively)

Action Space Discretization

- Can I predict action dimensions independently?
  - Assume I do, that's how the distribution will look:

$$\pi(a_t|s_t) = \pi(a_{t,0}|s_t)\pi(a_{t,1}|s_t)\pi(a_{t,2}|s_t) \dots$$

  - This implicitly assumes that action dimensions are independent given the state. Not really a good assumption.

- Let's try to remove the independence assumption:

$$\pi(a_t|s_t) = \pi(a_{t,0}|s_t)\pi(a_{t,1}|s_t, a_{t,1})\pi(a_{t,3}|s_t, a_{t,2}, a_{t,1}) \dots$$

- Does this formulation remind you of something?

Action Space Discretization: Autoregressive Prediction

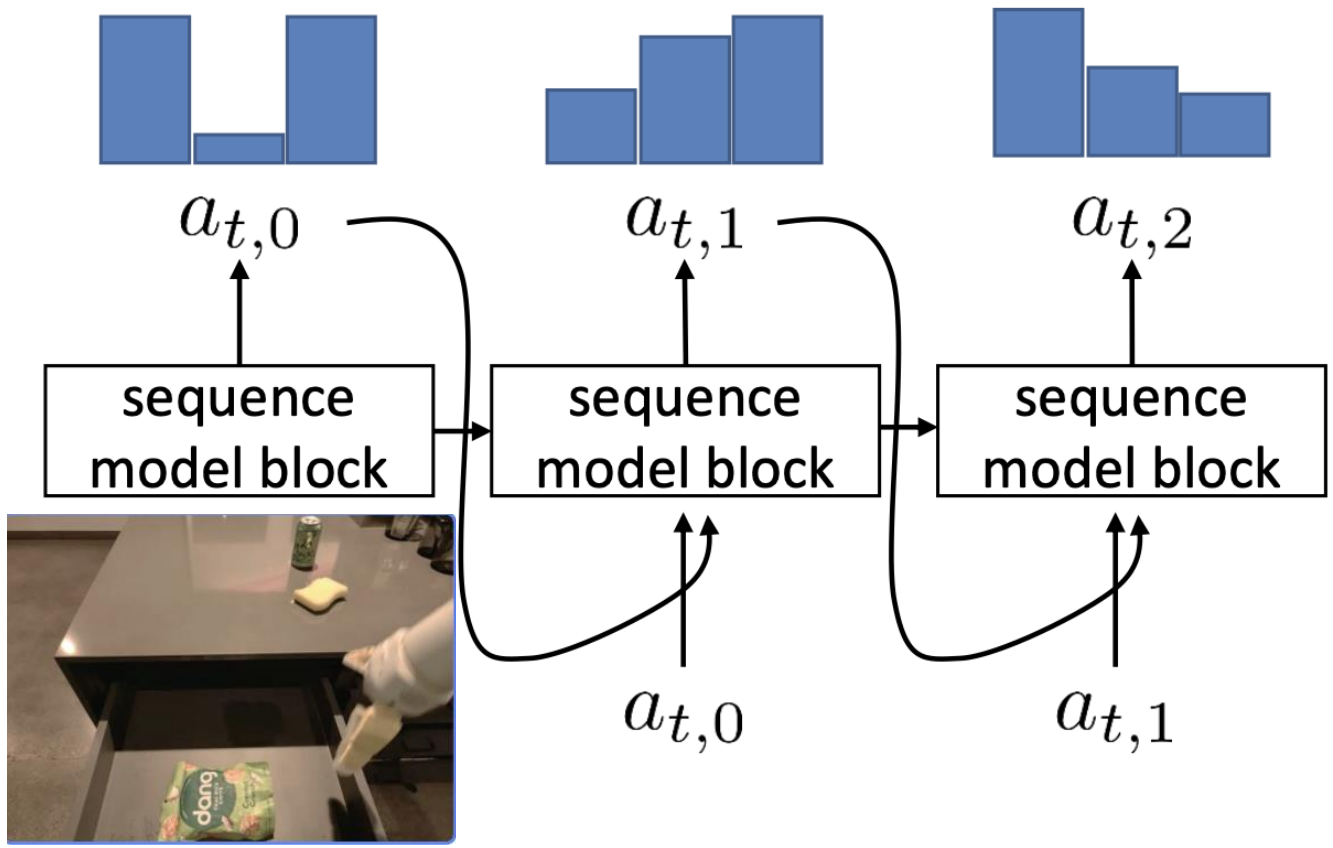
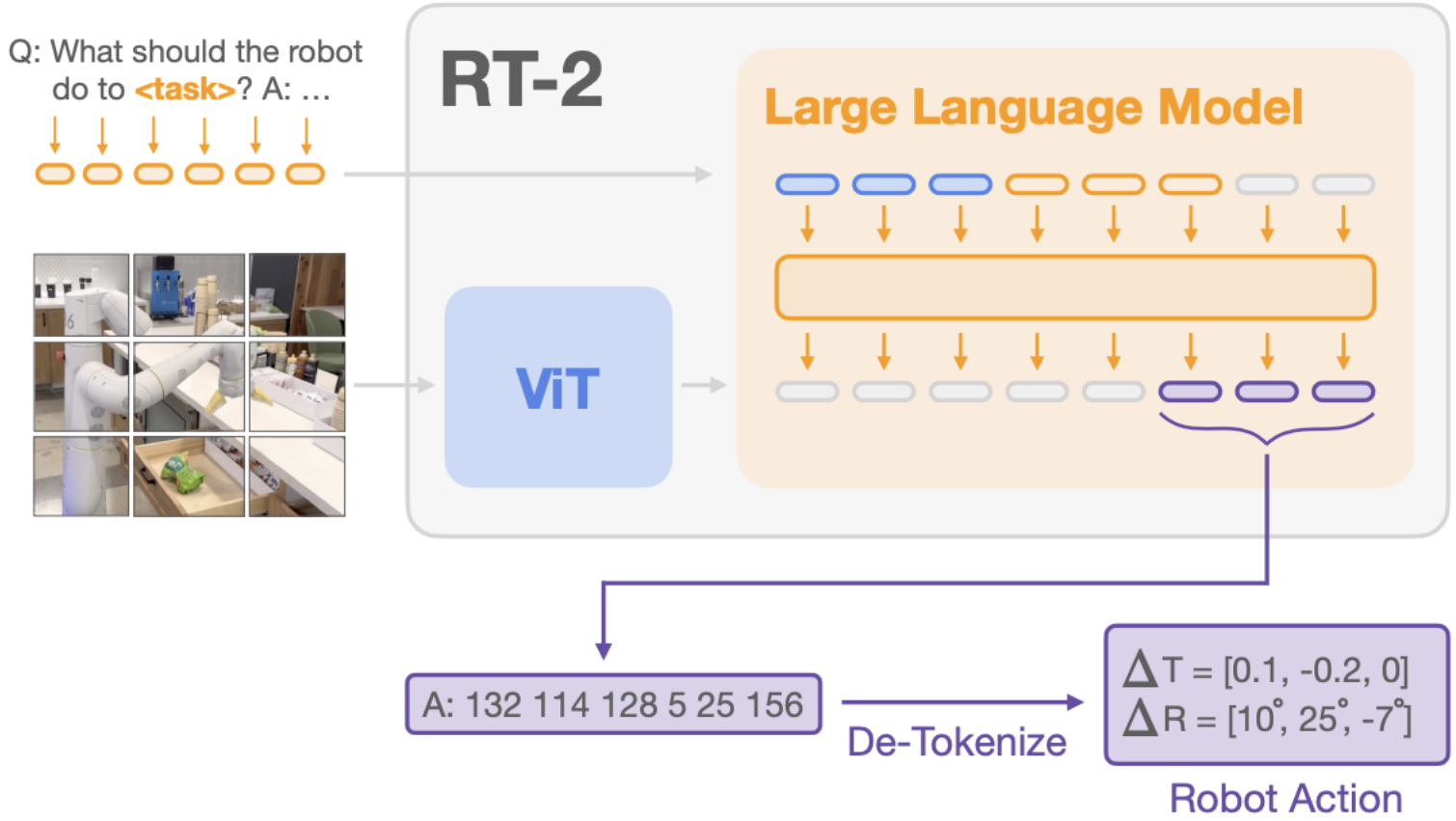


Image adapted from S. Levine

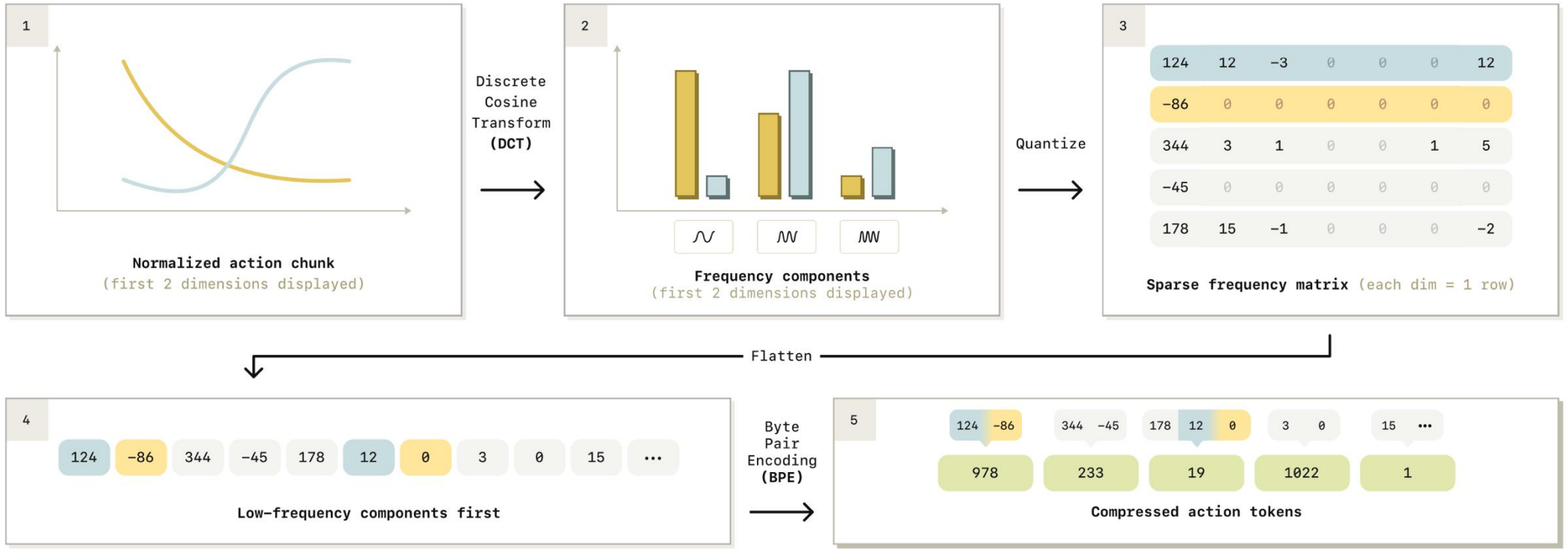
Action Space Discretization: An Example



- The model outputs a set of strings (tokens), each of which is mapped to an action (delta translation and rotation of end effector).

RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control

Action Space Discretization: FAST

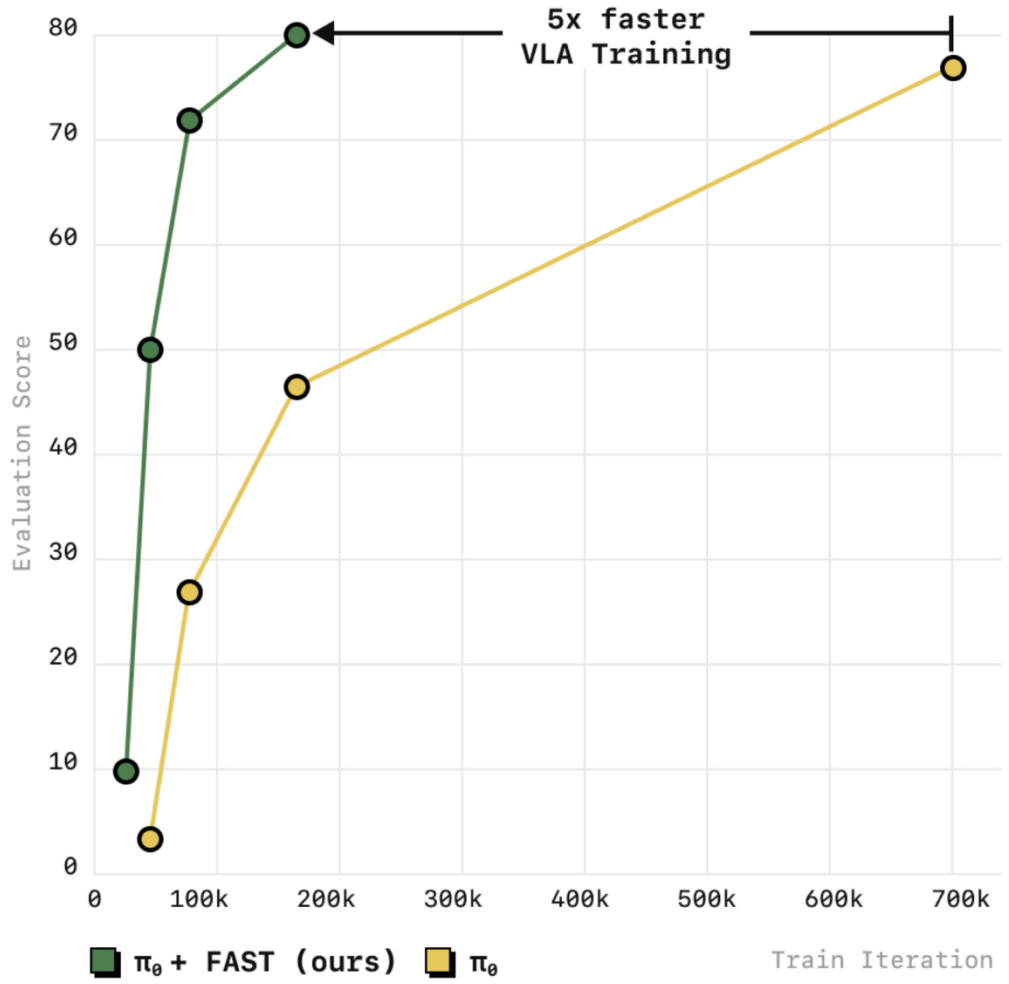


- Apply DCT to convert actions to the frequency domain. Then, quantize the DCT coefficients and use byte-pair encoding (BPE) to compress the flattened sequence of per-dimension DCT coefficients into the final action token sequence.

FAST: Efficient Action Tokenization for Vision-Language-Action Models, 2025

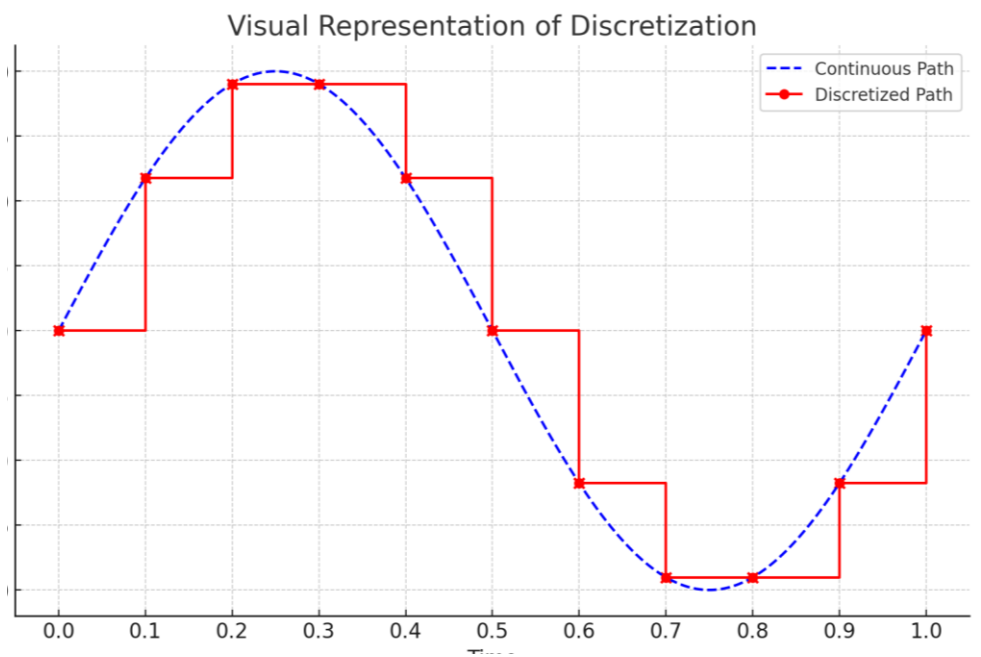
Action Space Discretization: FAST

- The way you discretize actions has a large impact on performance and training speed!



Action Space Discretization

- Advantages:



- Simple
- Effective

- Disadvantages:

- Resolution of action space
- You lose the inductive bias of continuous values (0.1 is closer to 0.2 than 0.5, but has equal “distance” to them in discrete space).

The Training Process

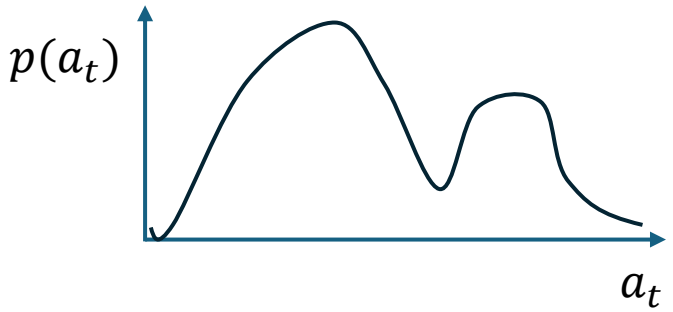
Possible solutions to the problem:

- Discretization of the action space

- **Learn a more expressive distribution**
  - Mixture of Gaussians
  - Diffusion
  - Latent actions

Expressive Continuous Action Distributions

Fancy NN

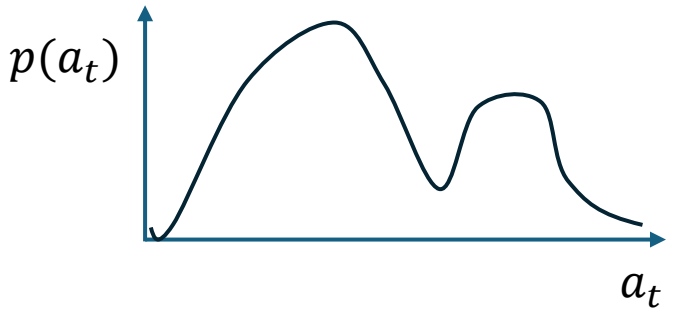


- Predict a (parametric) continuous distribution directly.
  - Example: parameters of a mixture of gaussians (e.g, Implicit Behavioral Cloning, 2021)
- Directly optimize the BC problem:

$$\theta^* = \text{argmax}_\theta \sum_{a_i, s_i \sim \rho_{\pi^*}} \log \pi_\theta(a_i|s_i)$$

Expressive Continuous Action Distributions

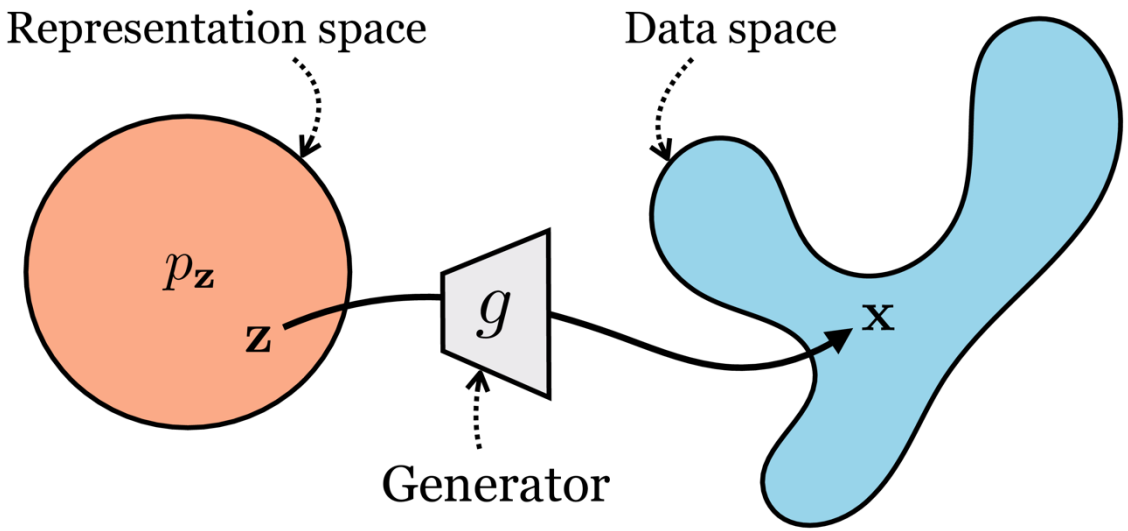
Fancy NN



- Do we need the whole distribution at test time?

- Generally, no (exception: we want to be aware of uncertainty)
- What we need is a sample from that distribution.
- Does this problem remind you of something?

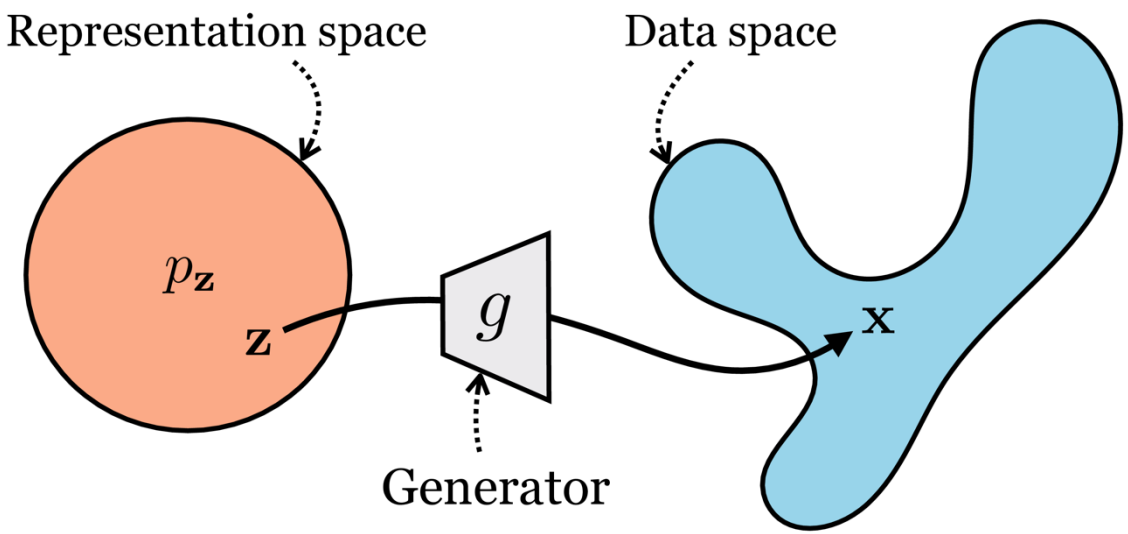
Generative Models



- High-level Objective: Transform a “simple” distribution, e.g., a unit Gaussian, into a complex distribution.
- Why is it possible? The Manifold Hypothesis

Image from “Foundations of Computer Vision”

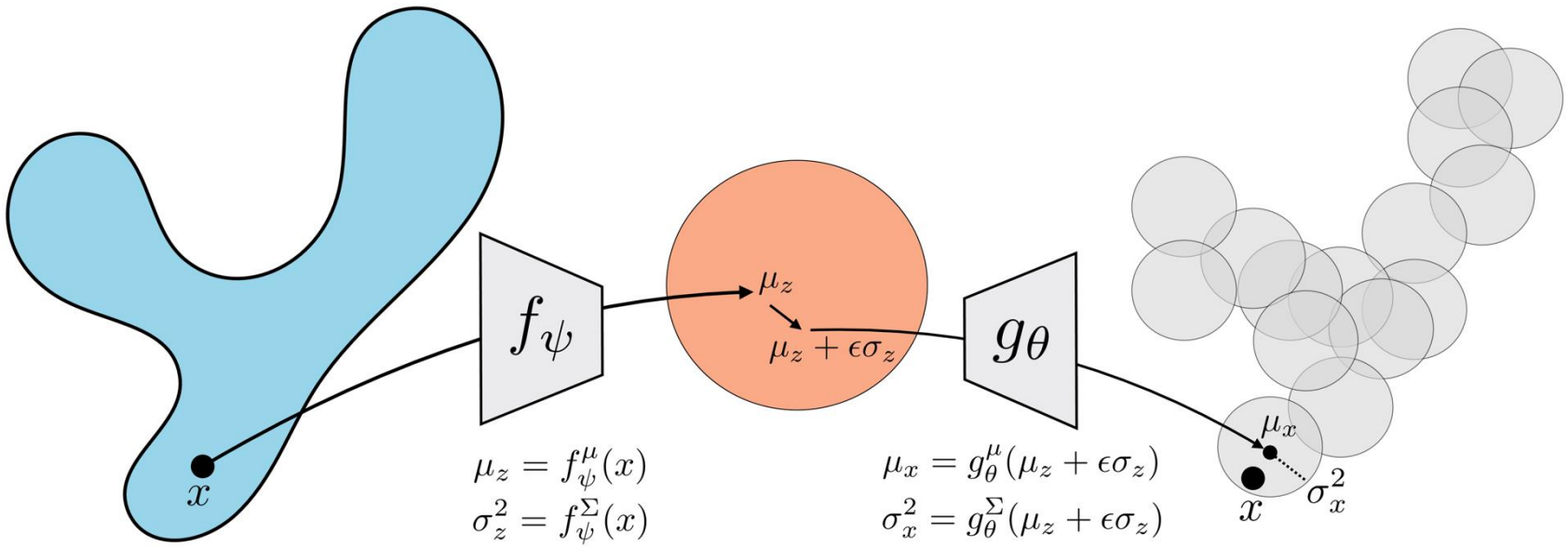
Generative Models



- Two types of generative modeling:
  - **Direct**: Predict the whole distribution (e.g., mixture of gaussians, as before)
  - **Indirect**: Transform a sample from the starting distribution into a sample from the goal distribution (recently popularized by genAI).
- Like genAI, indirect methods are the most popular in robot learning.

Image from “Foundations of Computer Vision”

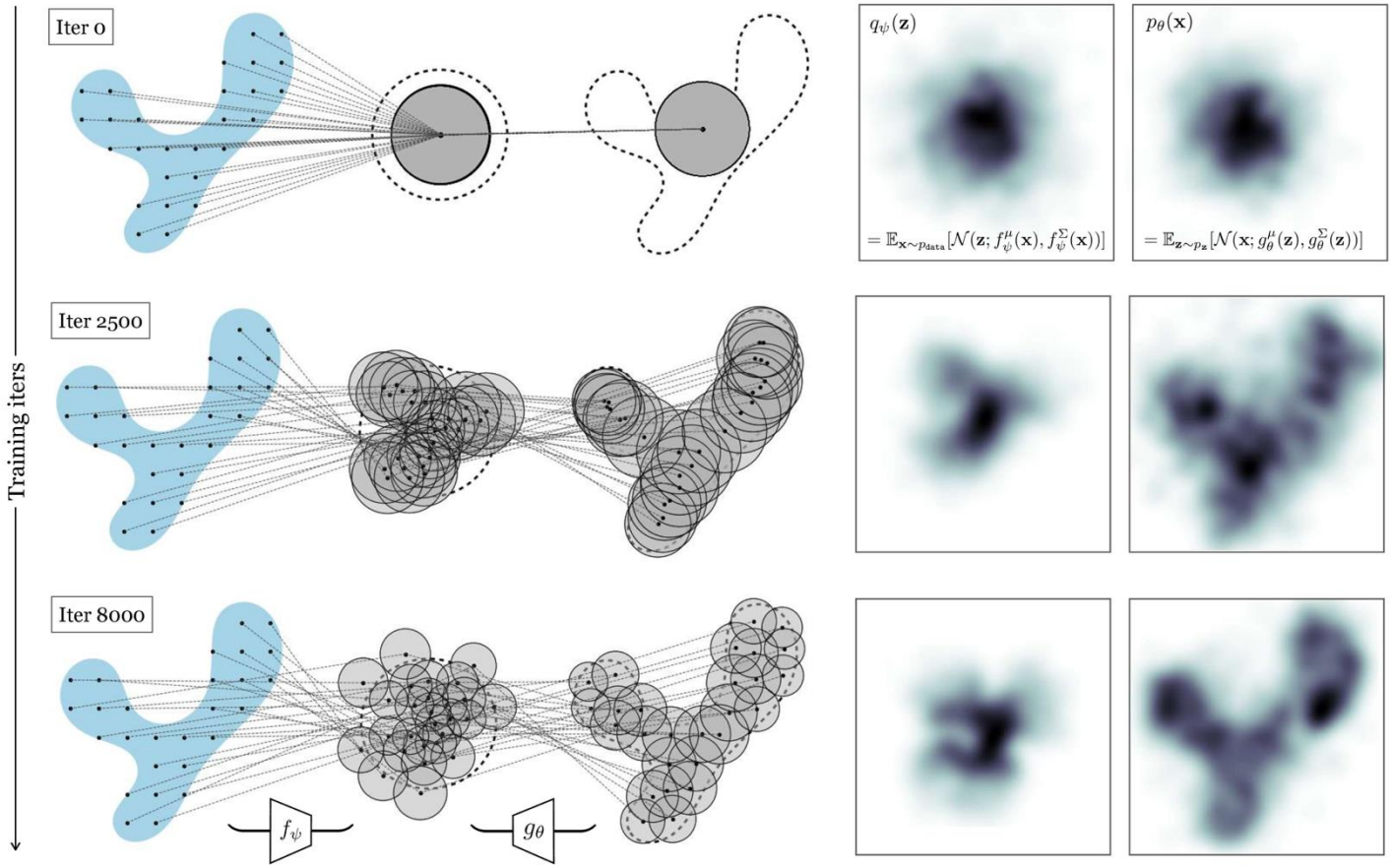
Generative Models: Variational Auto-Encoders



- Train an encoder to map a point into the mean and std of a distribution, sample from it, and decode the output.

Image from “Foundations of Computer Vision”

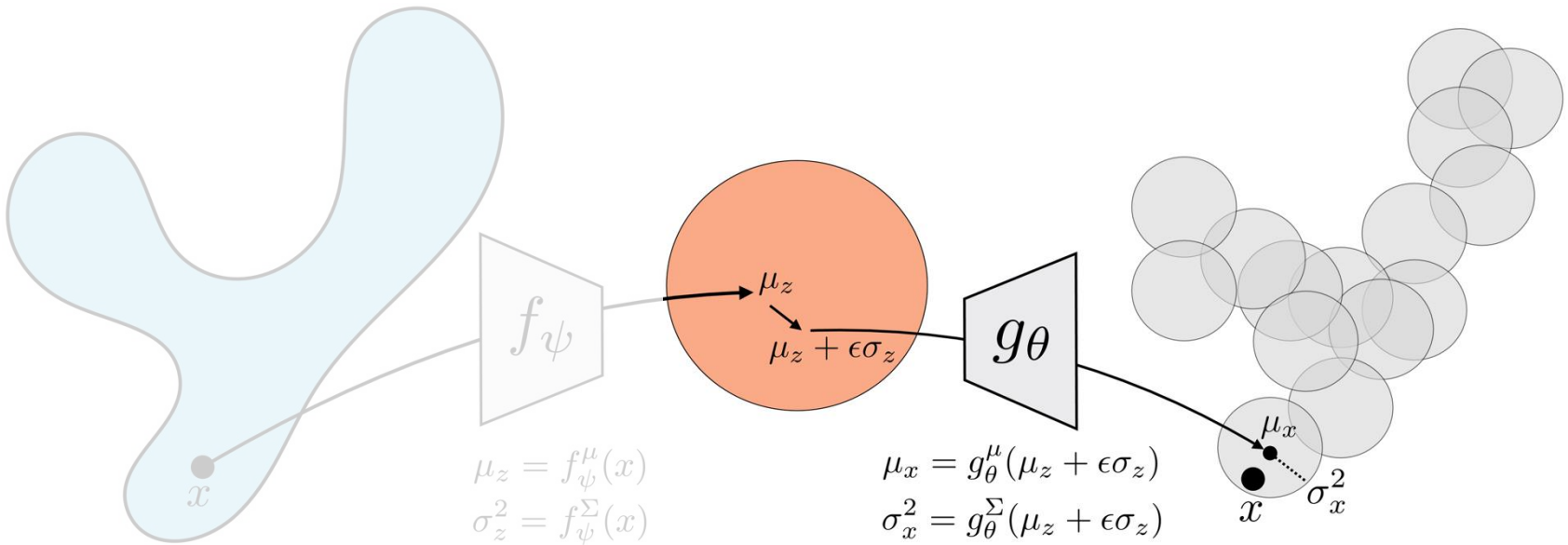
Generative Models: Variational Auto-Encoders



- Trained with reconstruction loss.

Image from “Foundations of Computer Vision”

Generative Models: Variational Auto-Encoders



- At test time, you can throw the encoder and sample from the decoder.

Image from “Foundations of Computer Vision”

Generative Models: Conditional Variational Auto-Encoders

- Providing additional information to the generator is a process called “conditioning”.
- This enables us to model a **conditional** distribution.
- What type of conditioning is very common in genAI?

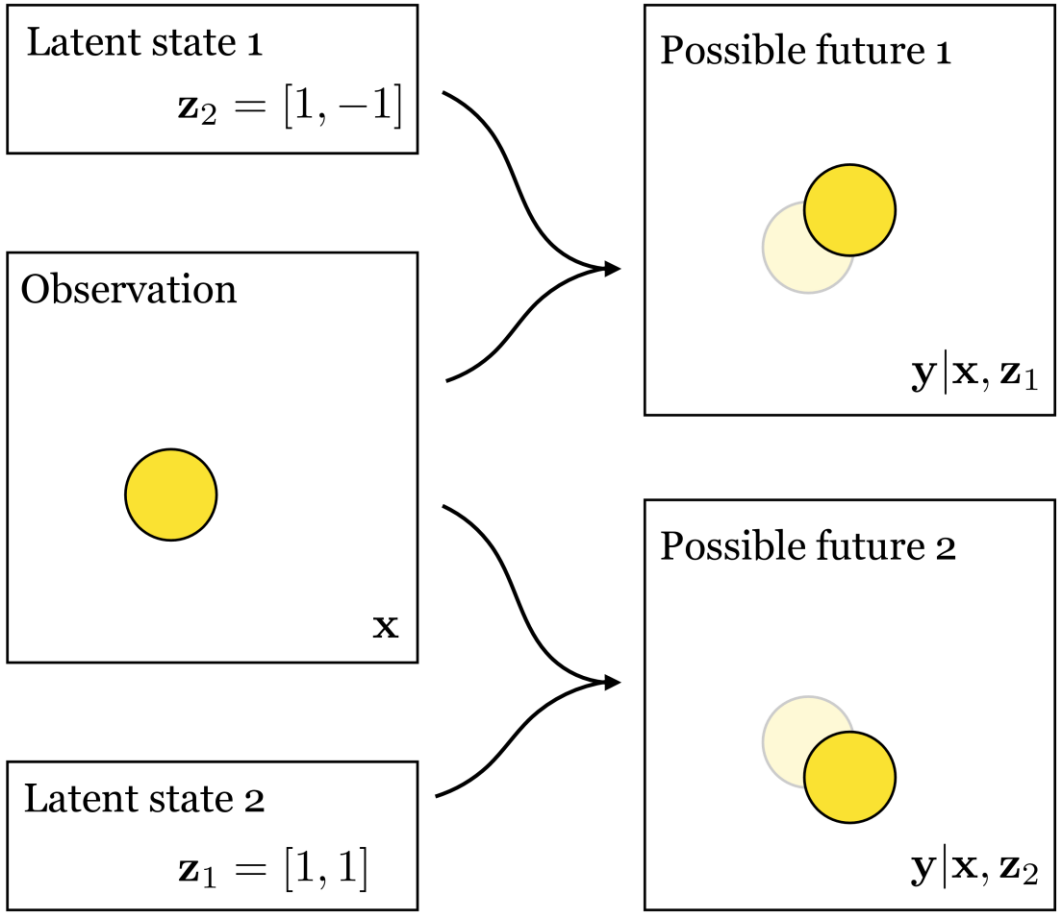
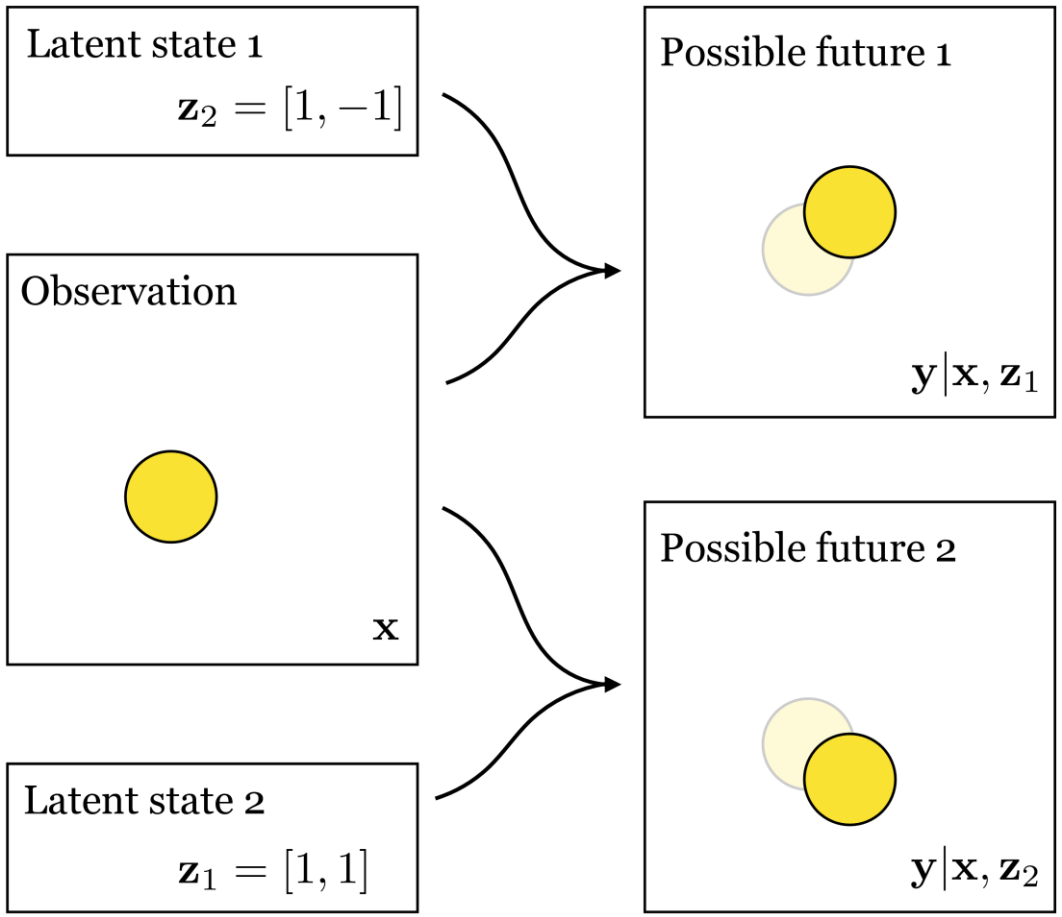


Image from “Foundations of Computer Vision”

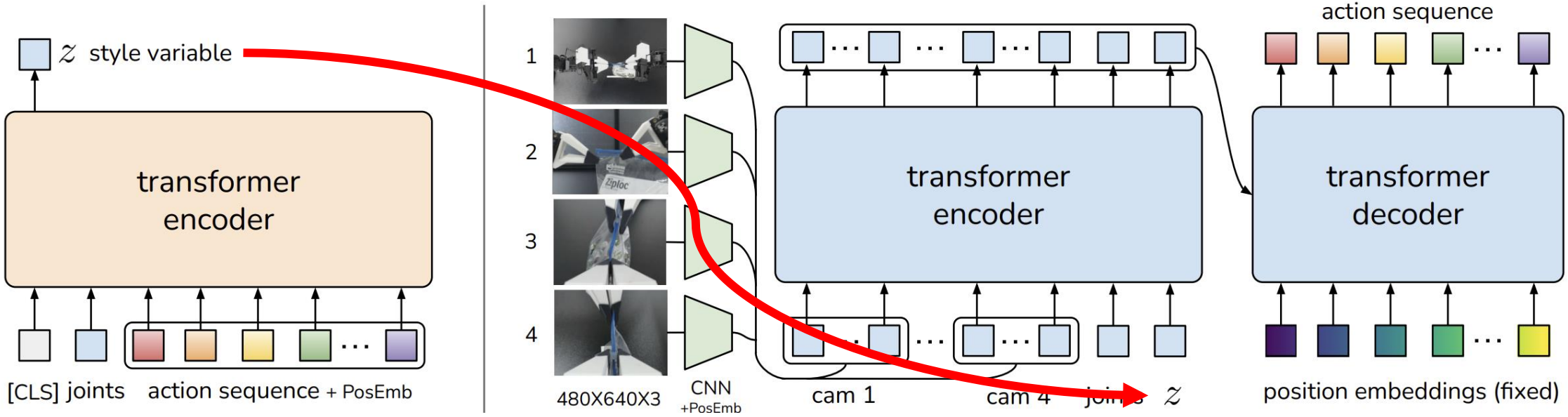
Generative Models: Conditional Variational Auto-Encoders

- In robotics, where we are trying to model $\pi_\theta(a_i|s_i)$, we should naturally condition on the "state" (e.g., sensor observations)
- Additional conditioning, e.g. text, are common in multi-task settings.



Conditional Variational AE in Robotics: ACT

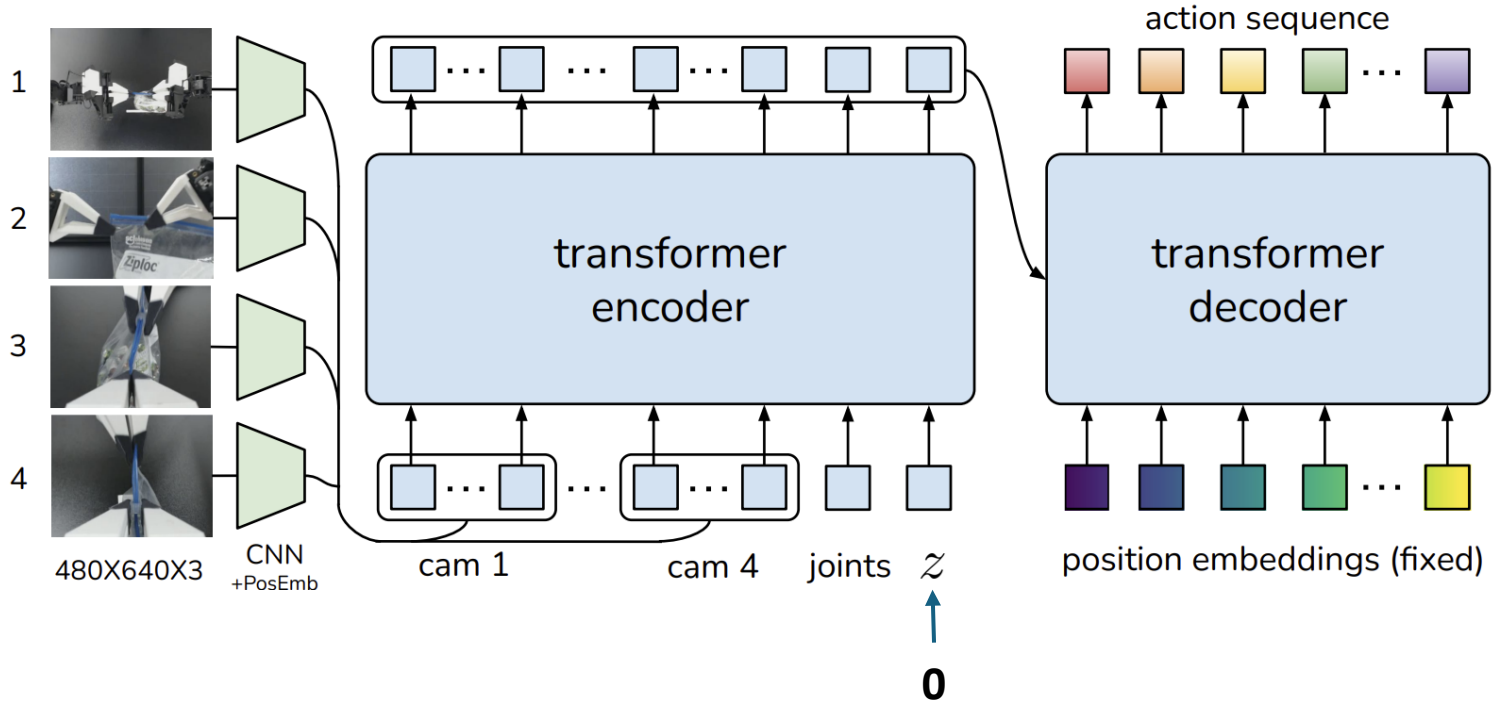
- Action-Chunking Transformer (ACT).
- Training Time: Action sequence reconstruction loss + z regularization.



Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023

Conditional Variational AE in Robotics: ACT

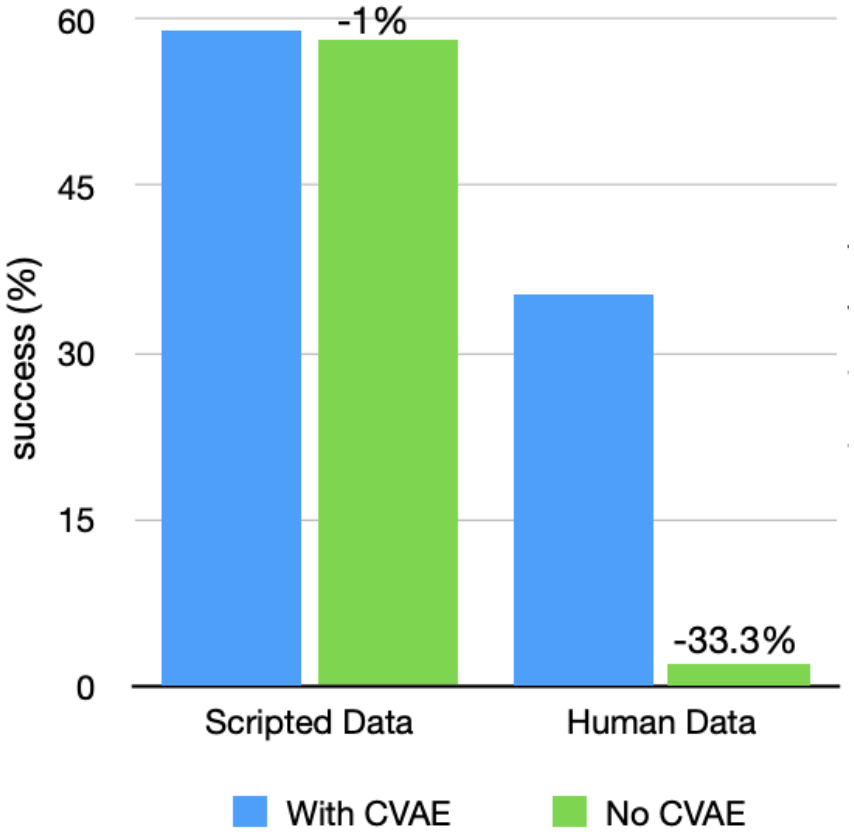
- Action-Chunking Transformer (ACT).
- Test Time: Put z to zero (mean of distribution)



Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023

Conditional Variational AE in Robotics: ACT

- Interesting find: The conditional VAE is particularly helpful with human data, but unimportant with scripted data.



Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, 2023

Generative Models: Diffusion

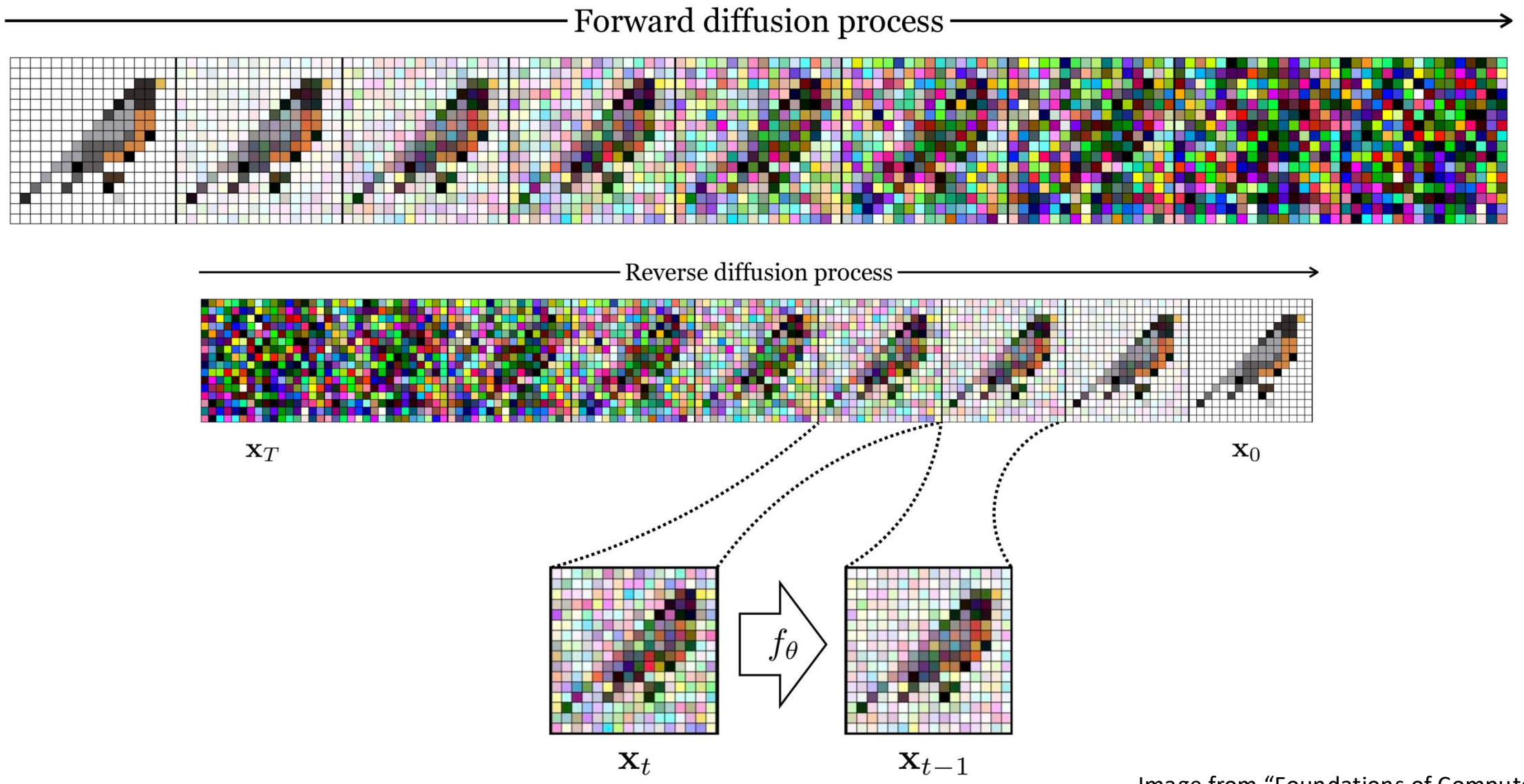
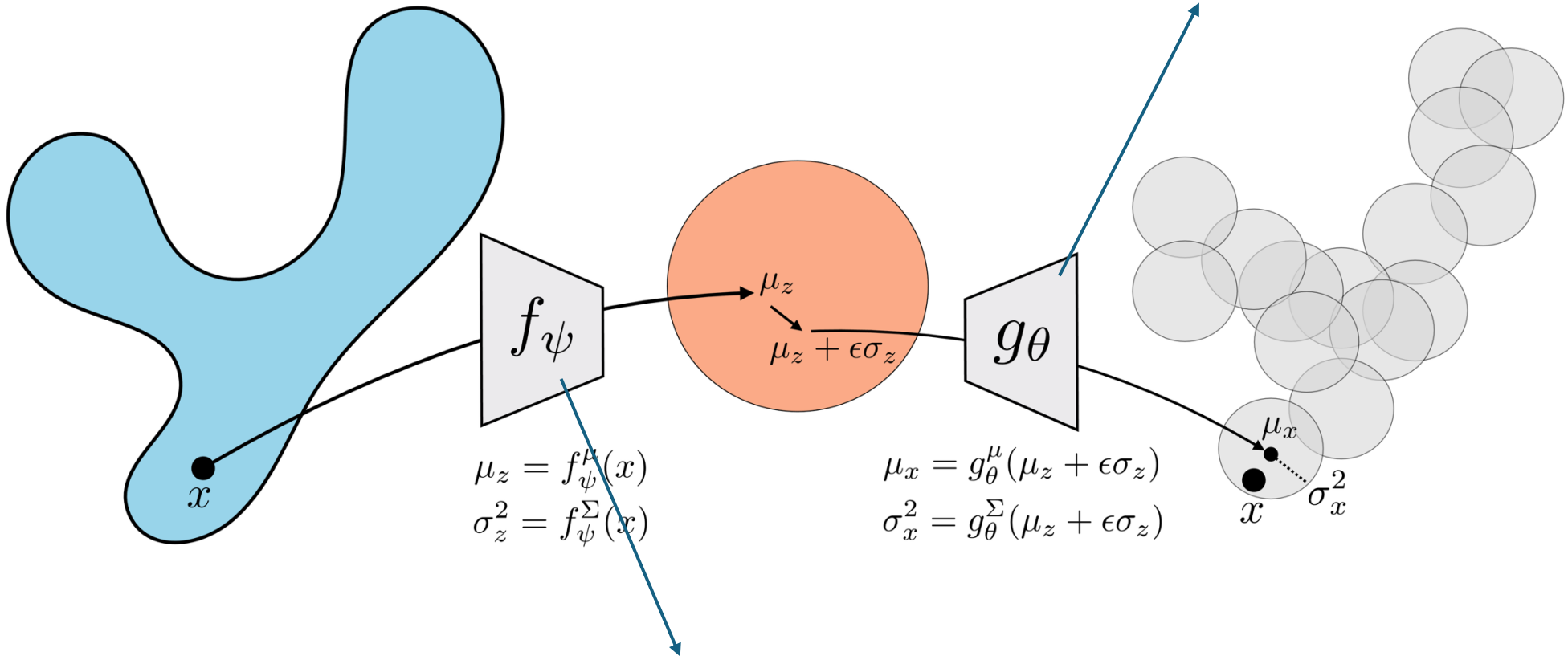


Image from “Foundations of Computer Vision”

Connection between Diffusion and VAE

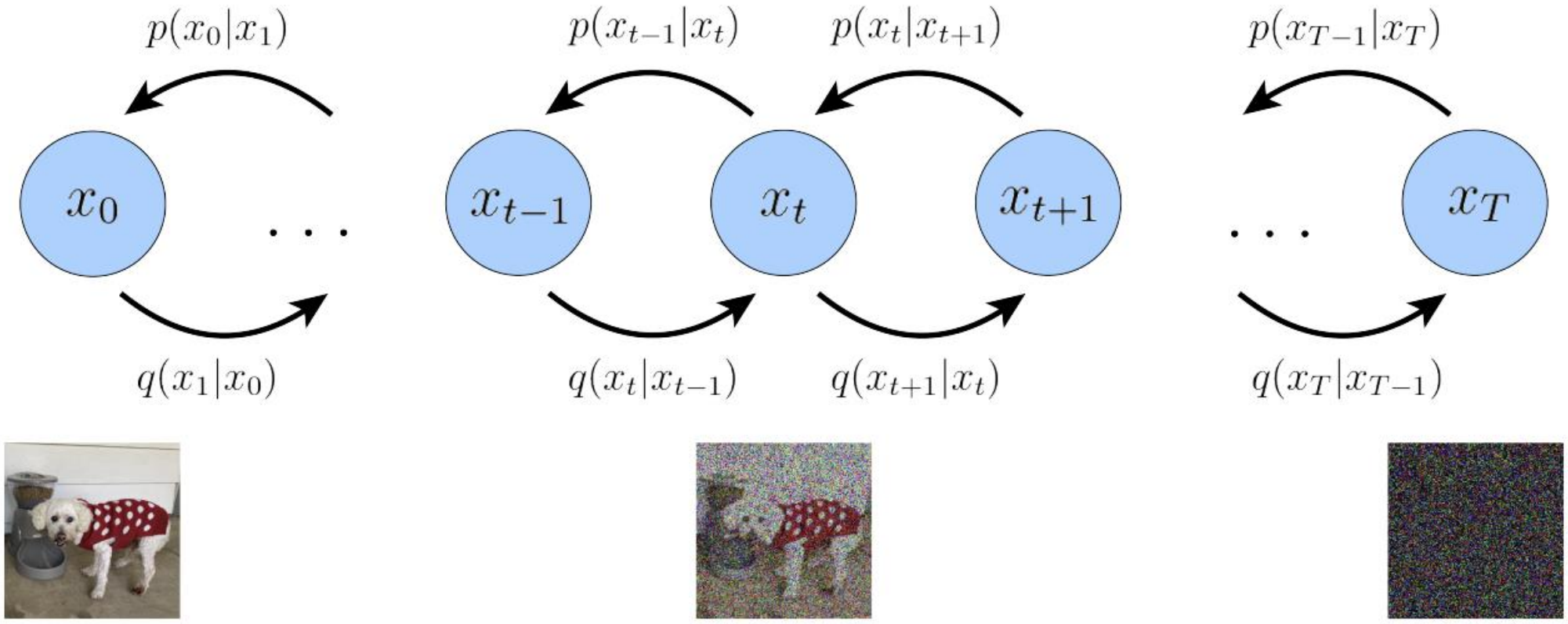
Let’s do multiple passes of the decoder



Let’s make this a fixed function: adding Gaussian noise with “randomized” standard deviations

Connection between Diffusion and VAE

A diffusion model is equivalent to a hierarchical VAE with a noising encoder!



Understanding Diffusion Models: A Unified Perspective, 2022

Conditional Diffusion

- Like conditional VAEs, additional inputs are given to the decoder to model conditional distributions.

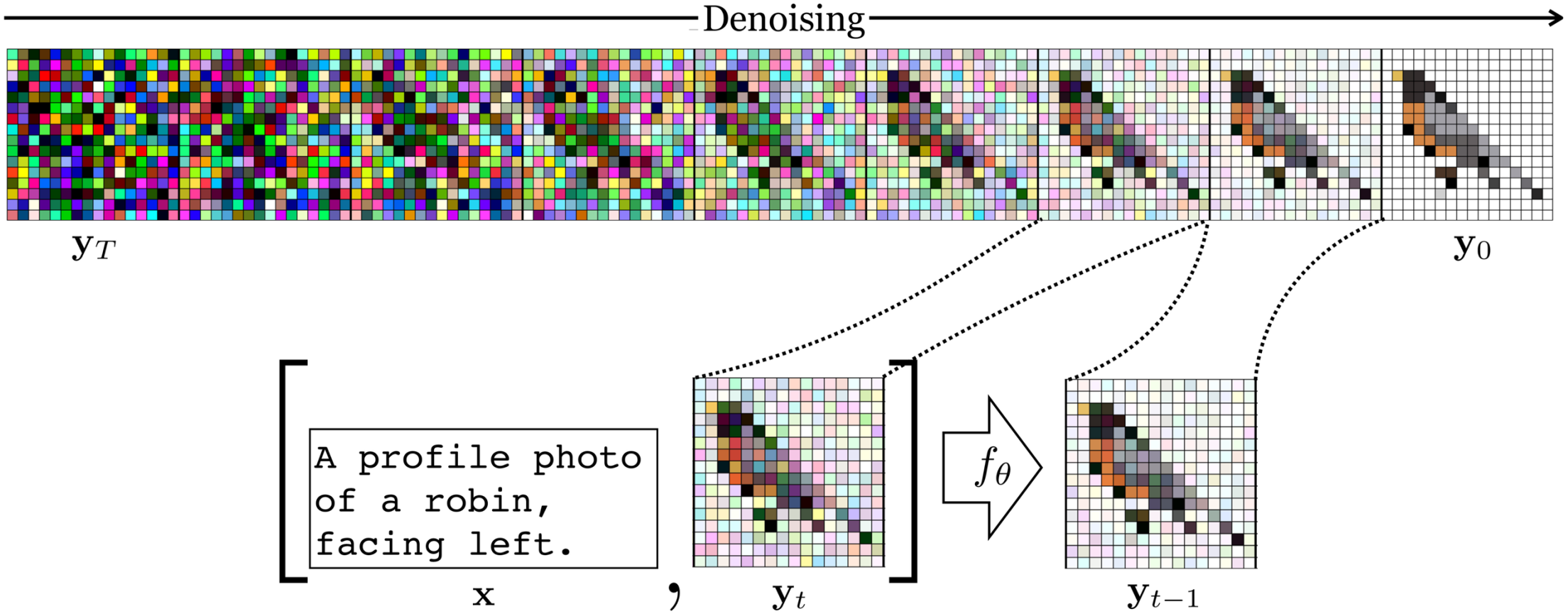
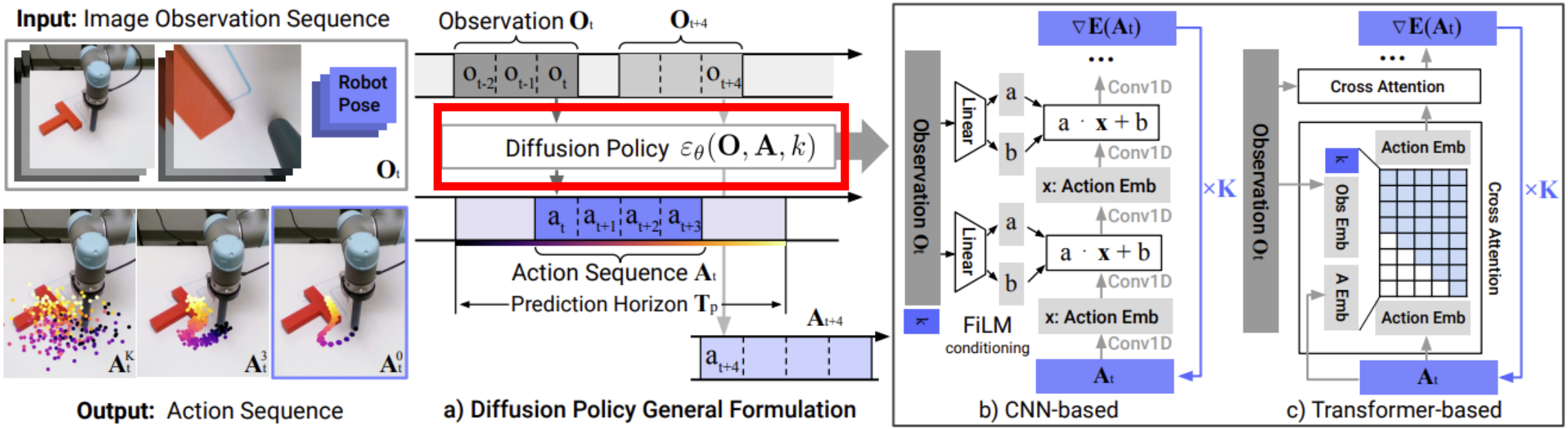


Image from “Foundations of Computer Vision”

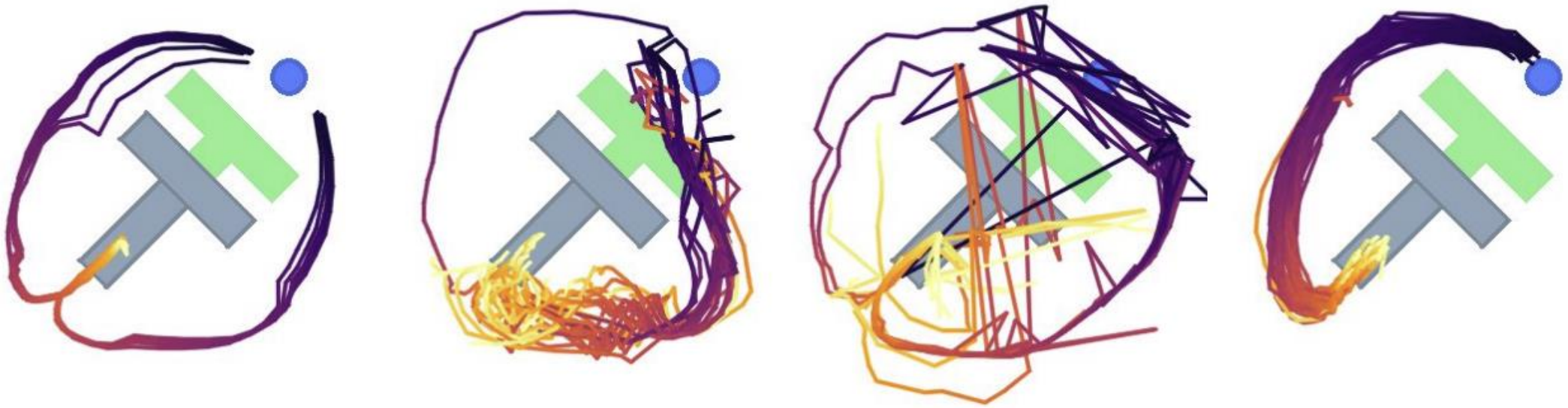
Conditional Diffusion in Robotics: Diffusion Policies

- The denoiser ϵ_θ is conditioned on the robot end effector pose and image sequence O_t, and the previous step denoised output A_t .
- Run the denoiser ϵ_θ for K steps.



Diffusion Policies, 2023

Diffusion Policies: Capturing Multi-Modality



Diffusion or C-VAE for BC in Robotics?

- The advantage of C-VAE is that it is simpler to implement and requires less computation at test time.

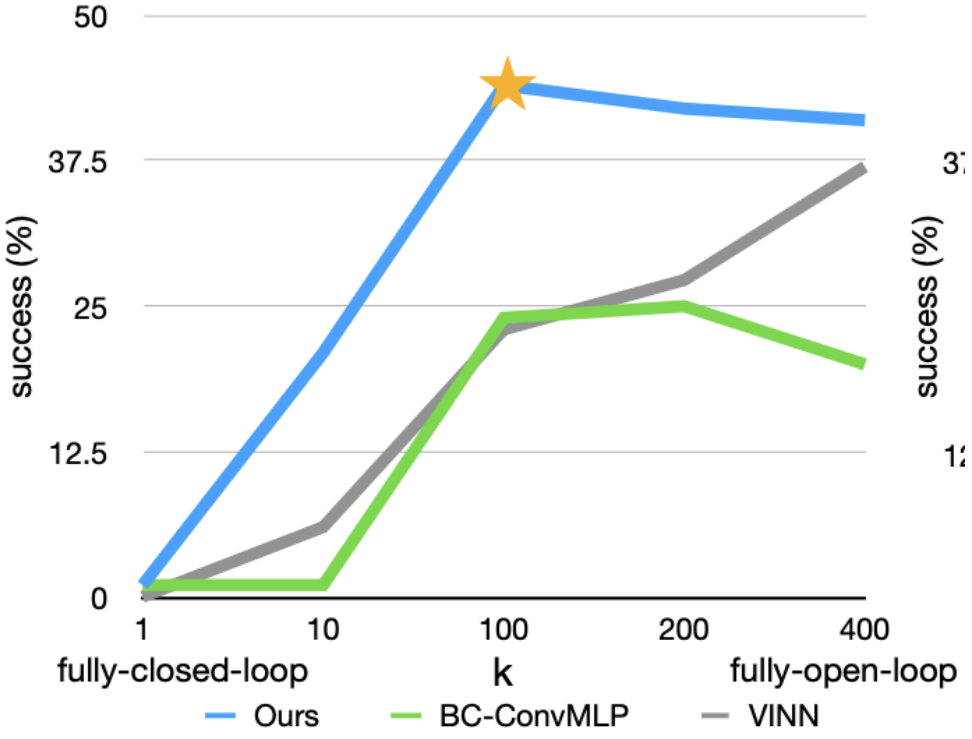
- The advantage of diffusion lies in its widespread adoption in GenAI, allowing us to leverage the latest advancements (e.g., classifier-free guidance). However, it requires more computation at test time.

- Diffusion is much more popular today in robot learning papers. It appears to be empirically superior to C-VAE.

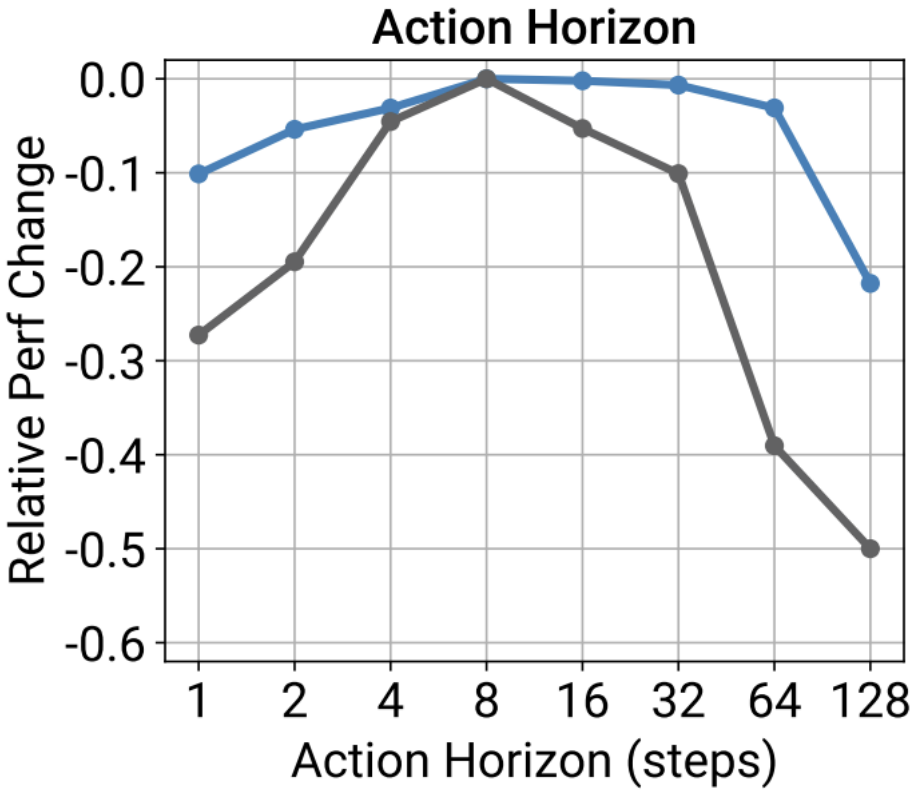
- Probably not so much of a difference in the low-data regime at which we generally work in academia.

A Common Ingredient in Diffusion Policies and ACT

- Multi-step action prediction (i.e., a chunk) is key to performance!



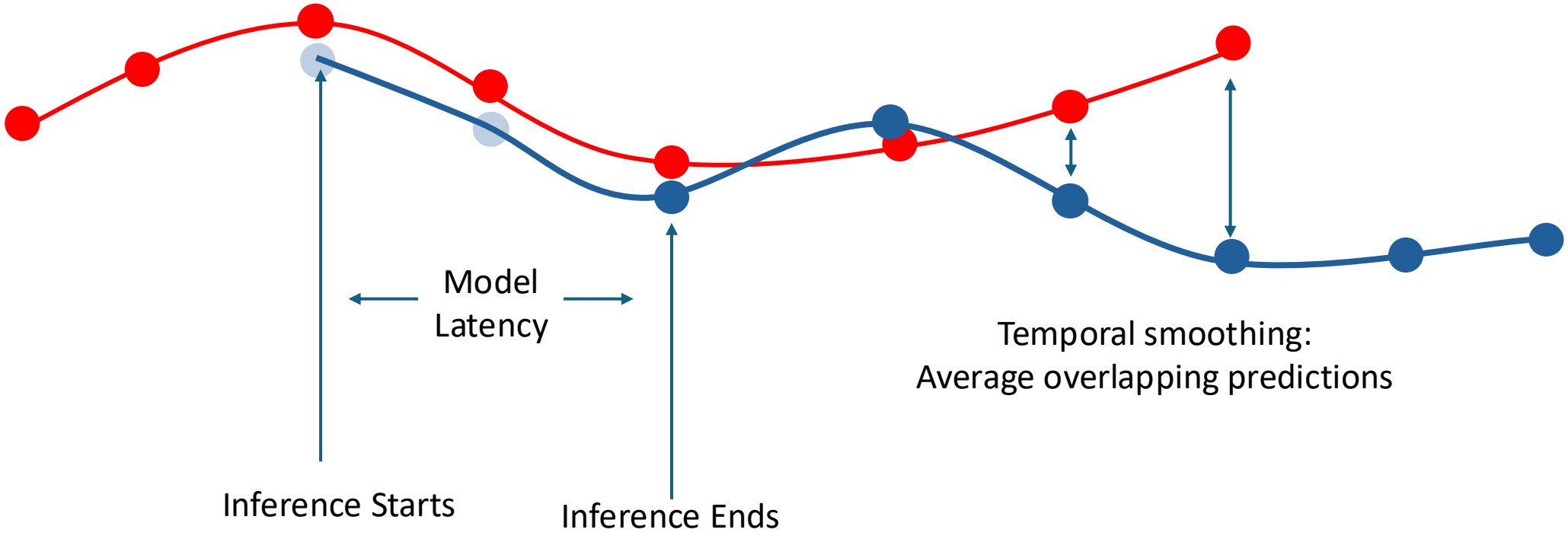
ACT



Diffusion Policies

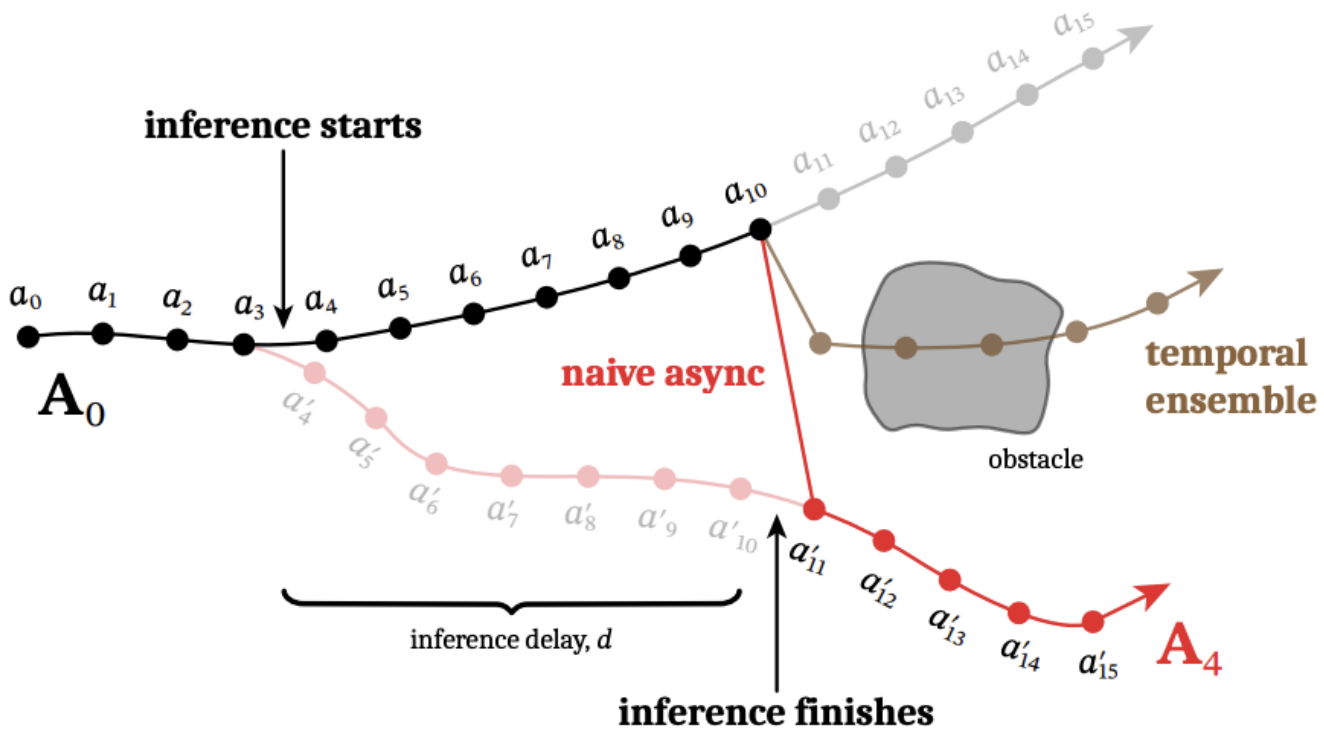
A Common Ingredient in Diffusion Policies and ACT

- Key advantage of chunking: decreasing the “effective” network’s latency while increasing smoothness.



A Common Ingredient in Diffusion Policies and ACT

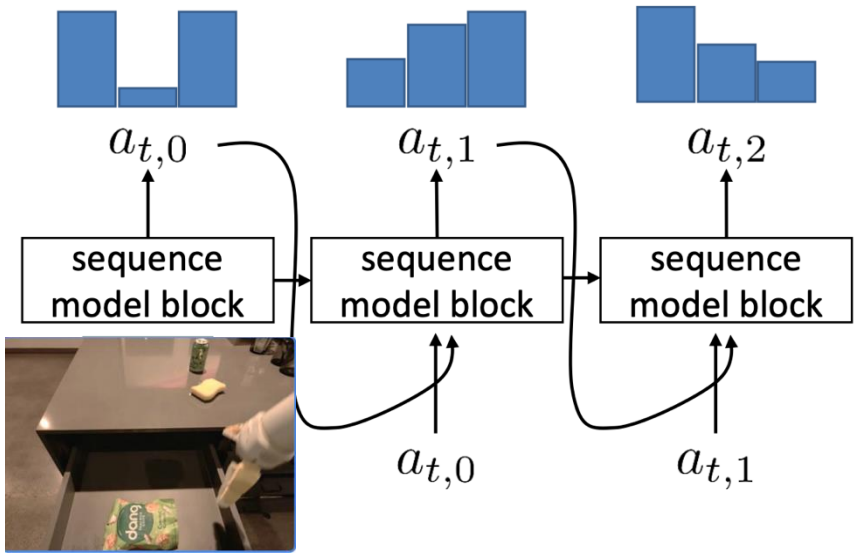
- Key advantage of chunking: decreasing the “effective” network’s latency while increasing smoothness.
- However, you must be careful!



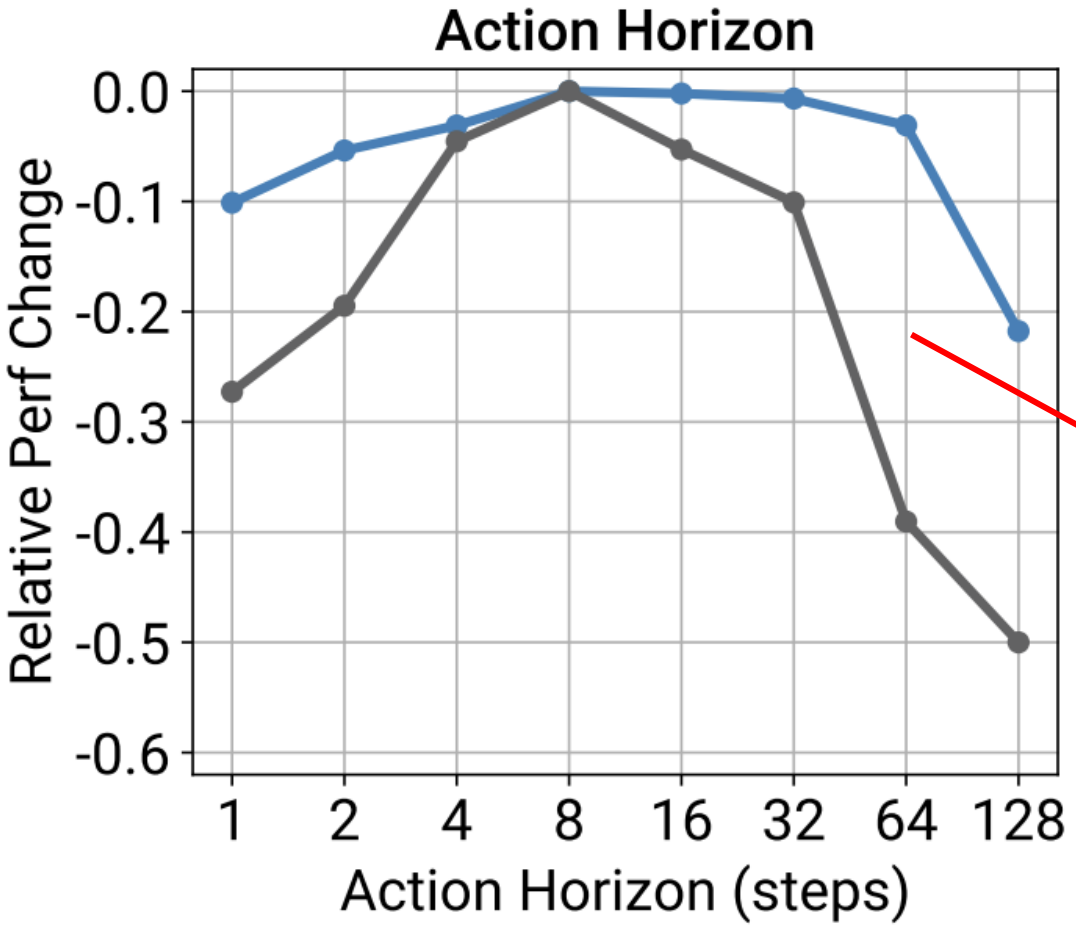
Real-Time Execution of Action Chunking Flow Policies, 2025

A Common Ingredient in Diffusion Policies and ACT

- Key advantage of chunking: decreasing the “effective” network’s latency while increasing smoothness.
- However, you must be careful!
- This is a significant advantage compared to Autoregressive policies, which must wait for the prediction of one action to be completed before starting the next one.



A Common Ingredient in Diffusion Policies and ACT



Why does performance decrease for longer prediction horizons?

I have not discussed many important details…

- Flow-matching instead of diffusion?
- How to cope with potentially bad demonstrations (e.g., the operator stops to see how to proceed)?
- What architecture to use for prediction (transformer, resnet, …) and how to parametrize the conditioning process?
- How to estimate inference delays in real time?
- How to parametrize actions (joint position, end-effector position, etc.)?

How a well-designed BC algorithm looks like

Algorithm 1 Real-Time Chunking

Require: flow policy π with prediction horizon H, minimum execution horizon s_{min}, mutex $\mathcal{M}$, condition variable $\mathcal{C}$ associated with $\mathcal{M}$, initial chunk $\mathbf{A}_{\text{init}}$, initial delay estimate d_{init}, delay buffer size b, number of denoising steps n, maximum guidance weight β

procedure INITIALIZESHAREDSTATE ▷ Initialize mutex-protected shared variables
  $t = 0$; $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{init}}$, $\mathbf{o}_{\text{cur}} = \text{null}$

function GETACTION($\mathbf{o}_{\text{next}}$) ▷ Called at an interval of Δt by controller
  with $\mathcal{M}$ acquired **do**
    $t = t + 1$
    $\mathbf{o}_{\text{cur}} = \mathbf{o}_{\text{next}}$
    notify $\mathcal{C}$
    return $\mathbf{A}_{\text{cur}}[t-1]$

procedure INFERENCELOOP ▷ Run inference in a looping background thread
  acquire $\mathcal{M}$
  $\mathcal{Q} =$ new Queue([d_{init}], maxlen=b) ▷ Holds a limited buffer of past inference delays
  loop
    wait on $\mathcal{C}$ until $t \geq s_{\text{min}}$
    $s = t$ ▷ s is the number of actions executed since last inference started
    $\mathbf{A}_{\text{prev}} = \mathbf{A}_{\text{cur}}[s, s+1, \ldots, H-1]$ ▷ Remove the s actions that have already been executed
    $\mathbf{o} = \mathbf{o}_{\text{cur}}$
    $d = \max(\mathcal{Q})$ ▷ Estimate the next inference delay conservatively
    with $\mathcal{M}$ released **do**
      $\mathbf{A}_{\text{new}} = \text{GUIDEDINFERENCE}(\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s)$
    $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{new}}$ ▷ Swap to the new chunk as soon as it is available
    $t = t - s$ ▷ Reset t so that it indexes into $\mathbf{A}_{\text{new}}$
    enqueue t onto $\mathcal{Q}$ ▷ Record the observed delay

function GUIDEDINFERENCE($\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s$)
  compute $\mathbf{W}$ using Eq. 5; right-pad $\mathbf{A}_{\text{prev}}$ to length H; initialize $\mathbf{A}^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $\tau = 0$ to 1 with step size $1/n$ **do**
    $f_{\widehat{\mathbf{A}^1}} = \mathbf{A}' \mapsto \mathbf{A}' + (1-\tau)\mathbf{v}_\pi(\mathbf{A}', \mathbf{o}, \tau)$ ▷ Define denoising function (Eq. 3)
    $\mathbf{e} = \left(\mathbf{A}_{\text{prev}} - f_{\widehat{\mathbf{A}^1}}(\mathbf{A}^\tau)\right)^\top \text{diag}(\mathbf{W})$ ▷ Weighted error term from Eq. 2
    $\mathbf{g} = \mathbf{e} \cdot \frac{\partial f_{\widehat{\mathbf{A}^1}}}{\partial \mathbf{A}'}\Big|_{\mathbf{A}'=\mathbf{A}^\tau}$ ▷ Compute vector-Jacobian product from Eq. 2 via autodiff
    $\mathbf{A}^{\tau+\frac{1}{n}} = \mathbf{A}^\tau + \frac{1}{n}\left(\mathbf{v}_\pi(\mathbf{A}^\tau, \mathbf{o}, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\mathbf{g}\right)$ ▷ Integration step (Eq. 1)
  return $\mathbf{A}^1$

- A lot of complexity is often overlooked in conversations about BC algorithms.
- Like in RL methods, effective system engineering is crucial to achieving good performance.

Real-Time Execution of Action Chunking Flow Policies, 2025

Recap: The Training Process of a BC Policy

We have seen three main types of training processes for BC policies:

- **Explicit Action Prediction**:
  - Directly predict an action given an observation. Either continuous or categorical.
- **Direct Density Prediction** (Implicit BC):
  - Predict a full distribution over actions, e.g., a mixture of Gaussians.
- **Indirect density Prediction** (Diffusion, C-VAE):
  - Predict a sample from the target (conditional) distribution.

- The latter is right now the most popular in robot learning papers.

Aside: My opinion on generative action prediction

- I am hesitant to believe this story about the multi-modality of the conditional action distribution being the reason why we need generative models.

- What's the probability of being in a purely multi-modal state? Having a history of observations potentially decreases this probability…

Aside: My opinion on generative action prediction

- I am hesitant to believe this story about the multi-modality of the conditional action distribution being the reason why we need generative models.

- What's the probability of being in a purely multi-modal state? Having a history of observations potentially decreases this probability…

- I believe that the real reason why generative models are so successful in robot learning is that collecting data is challenging, and particularly at scale, humans make numerous mistakes.

- **Generative models are better able to "absorb" these mistakes than explicit action prediction methods.**

Behavioral Cloning: Agenda

- Theoretical Foundations
- Tools for Data Collection
- Algorithms
- **Leveraging foundation models**
- Challenges