

Multitask policy learning and Vision-Language-Action models

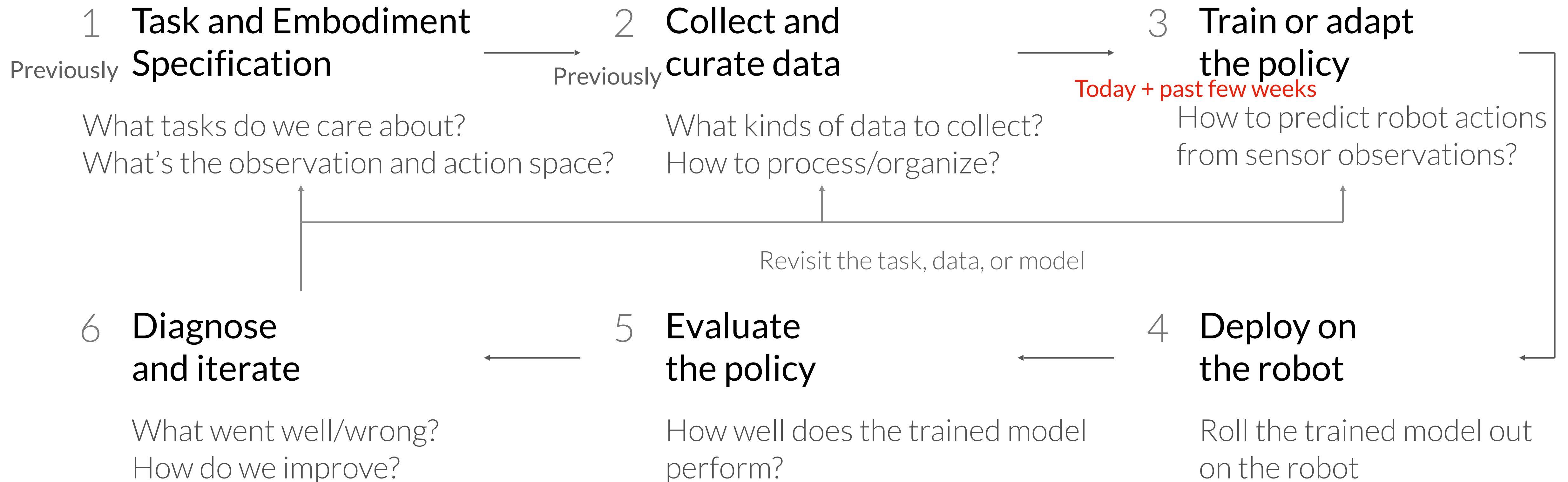
EN.601.[4|6]98 Hands-on Robot Learning

Krishna Murthy Jatavallabhula

Administrivia

- Assignment descriptions for assignments 1, 2, 3 out tomorrow morning
- Assignment 0 evaluated — please look at feedback from course staff and fix hardware issues if prompted
- Assignment 0 statistics:
 - Max/Min/Median team scores: 100%
- Steven's office hours this Friday will be remote (email incoming with link)
- Reminder: Fall break; no class next week
- The week after (i.e., on Oct 29), assignment 1 will be demo-ed in-class (first half of the class will be a shortened lecture, and the second half will be demos)

The *modern* robot learning lifecycle



Generalization in robot learning

- So far, we looked at single-task imitation learning
- In single-task imitation learning, the *task* is *implicitly specified* (by the demonstration distribution)
- To build robots that *generalize*, it is infeasible to collect demonstrations and train a task-specific policy for every new task the robot will ever encounter
- Instead, we would like one policy that can perform many tasks
- *Generalization* can take on multiple meanings — new objects, environments, robot embodiments, tasks; or *all of these*

Multi-task policy learning

- Learn a single policy that enables multiple tasks (interacting with multiple objects, across multiple environments)
- A primary hypothesis is that multi-task learning entails multi-task data gathering: collect data across multiple objects, environments, tasks, and perhaps embodiments
- Train a shared policy; scale data and compute (to the extent that budget allows)
- The bet is that generalization becomes accessible as this experience grows
- This is the bet (variants of which) most frontier robotics labs are making now

Challenges in multitask policy learning

- How do we represent heterogeneous robot data?
- How do we encode all this data into a single model?
- How do we account for heterogeneous data sources (robot data, human data, hybrid/wearable data)?
- How do we build upon useful knowledge available in web-scale pretraining?
- How do we make learning new tasks more efficient?

Aside: Task-conditioned diffusion policies

Large Behavior Models (LBMs)

- A multi-task model for a fixed set of previously seen tasks
- Train a *conditional* diffusion policy on demonstrations from many tasks
- Task is specified as an additional conditioning input



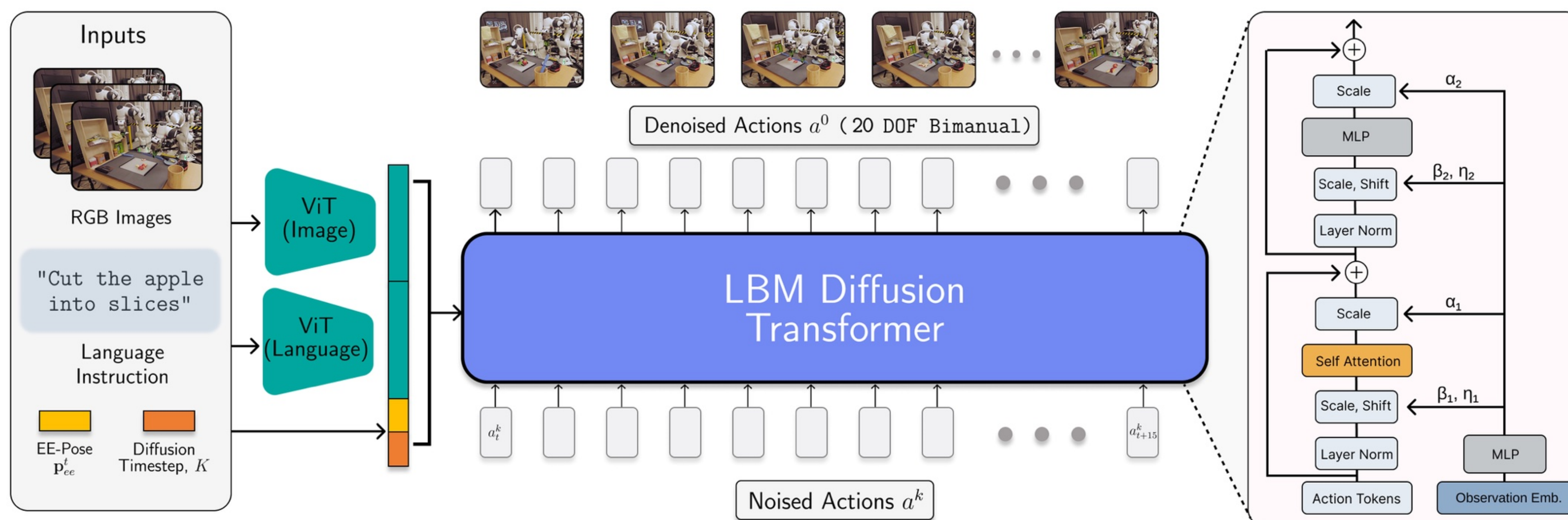
Multitask policy learning: nominal training objective

$$\mathcal{L}(\theta) = \sum_{k=1}^K w_k \mathbb{E}_{(o,a) \sim \mathcal{D}_k} \left[\ell(\pi_{\theta}(o, k), a) \right]$$

- Policy parameters shared across multiple tasks
- Task-sampling weights (w_k) determine how much each dataset contributes
- Here, the per-example loss is the conditional diffusion objective (predict noise at each timestep)

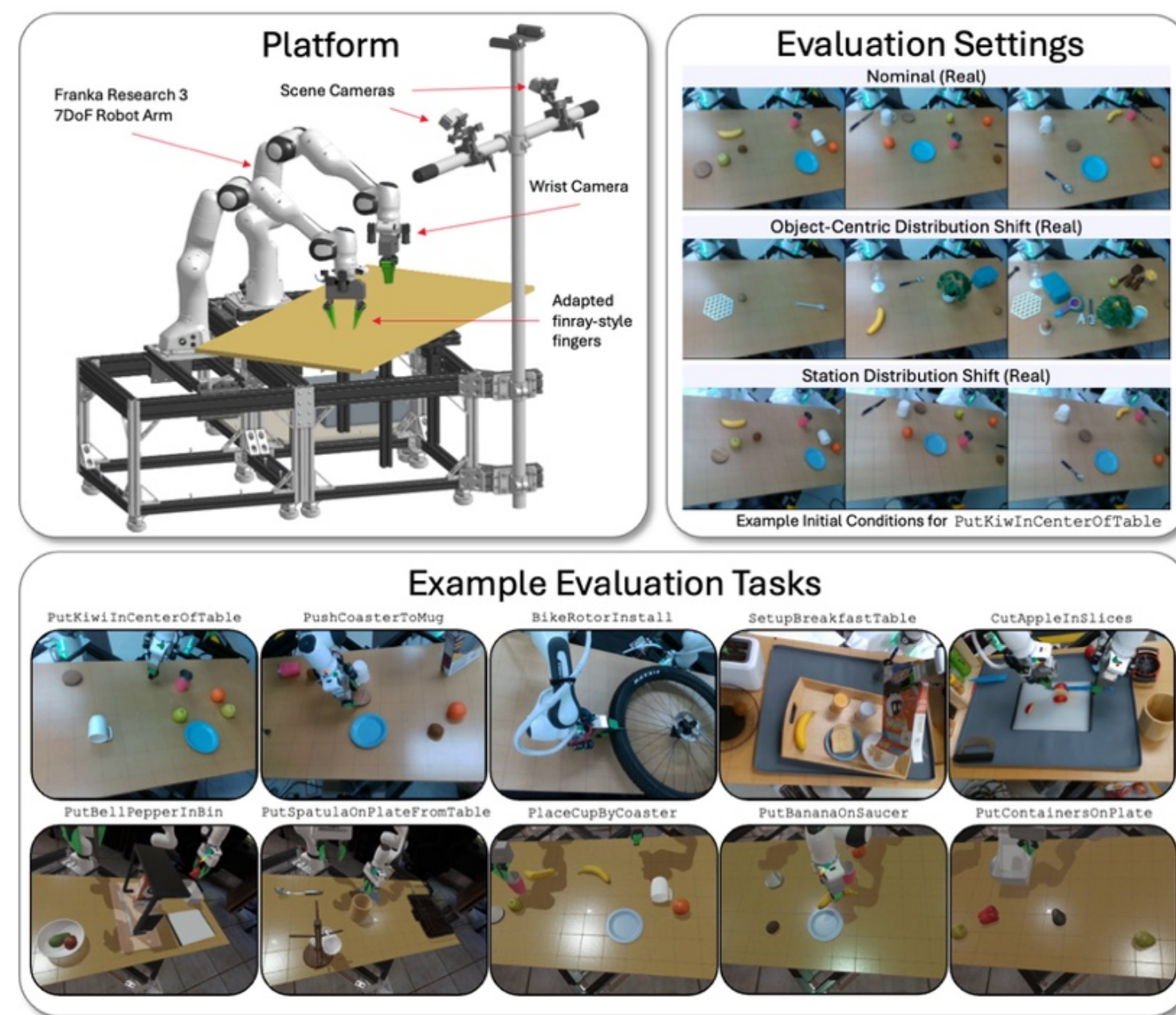
LBM: conditioning and action prediction

- Images, proprioception, and projected CLIP text features are concatenated
- Concatenated vector used as conditioning input in a diffusion transformer
- Action chunk prediction horizon is 16 future steps, and each action is 20-dimensional



LBM: pre-training and fine-tuning

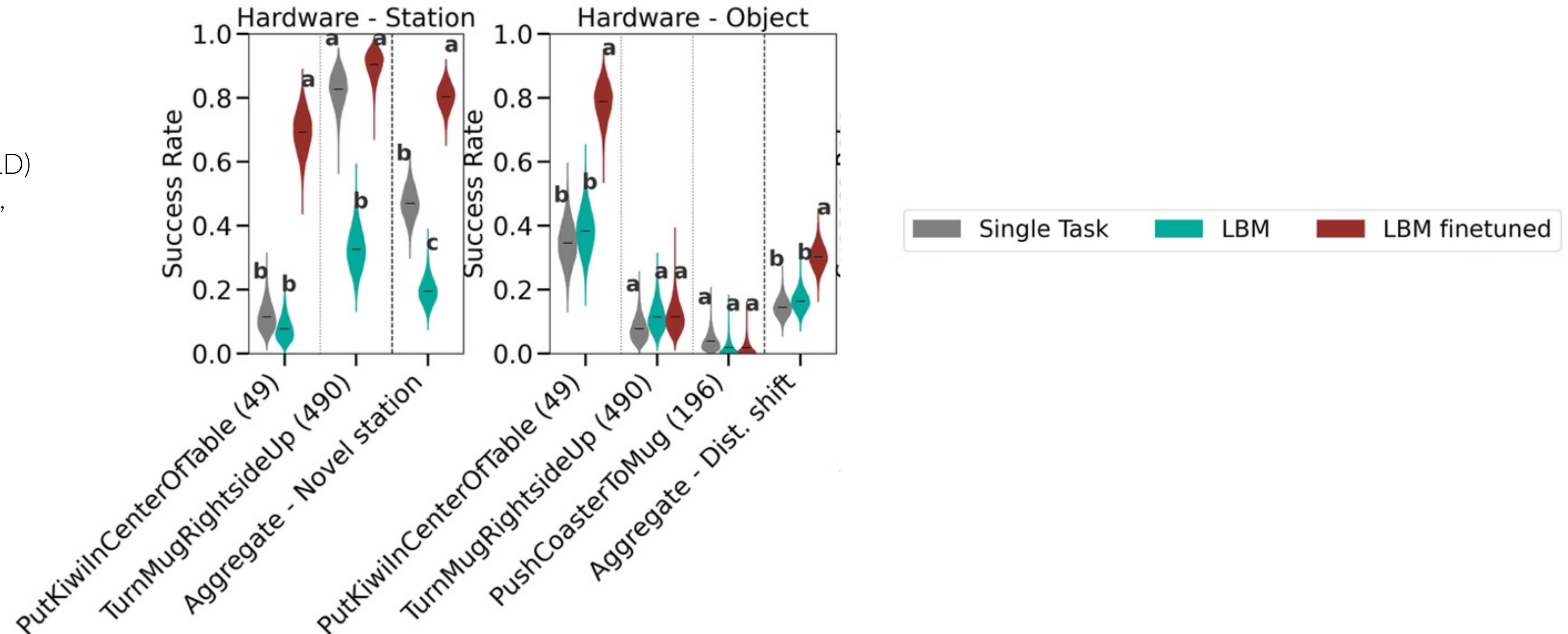
- Pre-training comprises mix of real-robot, simulation, UMI and Open X-Embodiment data
- Pre-trained model is fine-tuned on task-specific data (one model per task)
- Pre-training helps (compared to training from scratch for the target task, pre-trained models require 3-5x fewer demonstrations)



LBM: evaluation (seen tasks, initial condition distribution shifts)

- Evaluate familiar tasks under nominal initial conditions and distribution shift
- The benefit of pretraining depends on whether the policy is adapted to the target setting

Compact Letter Display (CLD) plots. If letters are different, approaches are statistically significantly different (95% confidence)

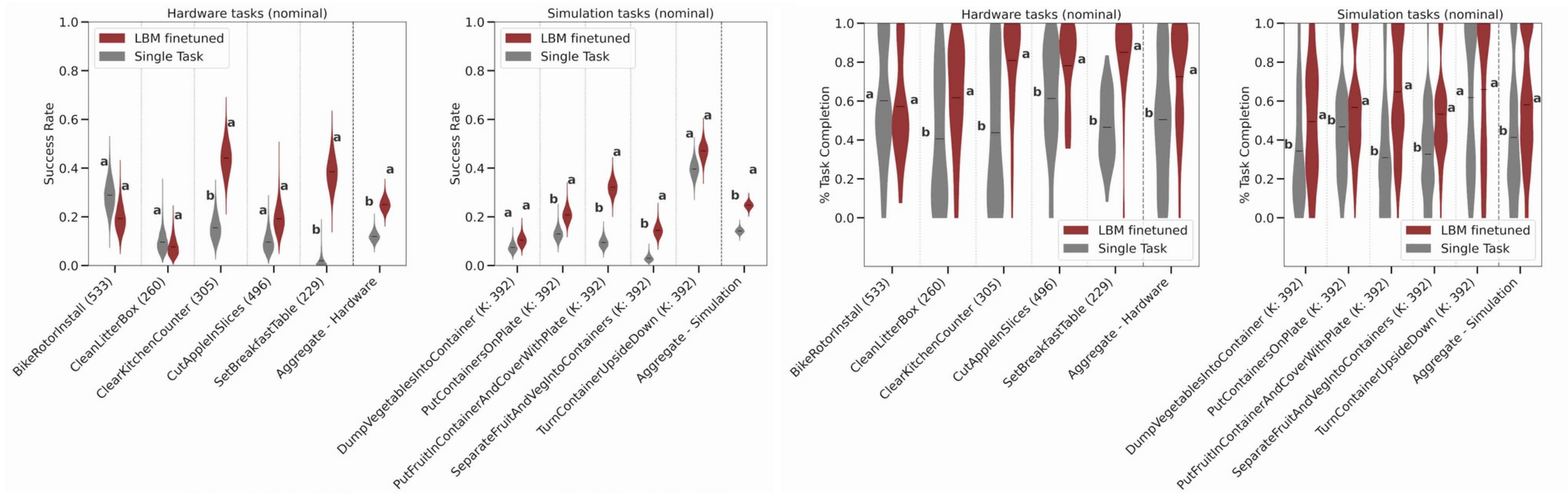


LBM: Single-task policy vs Multi-task pretraining

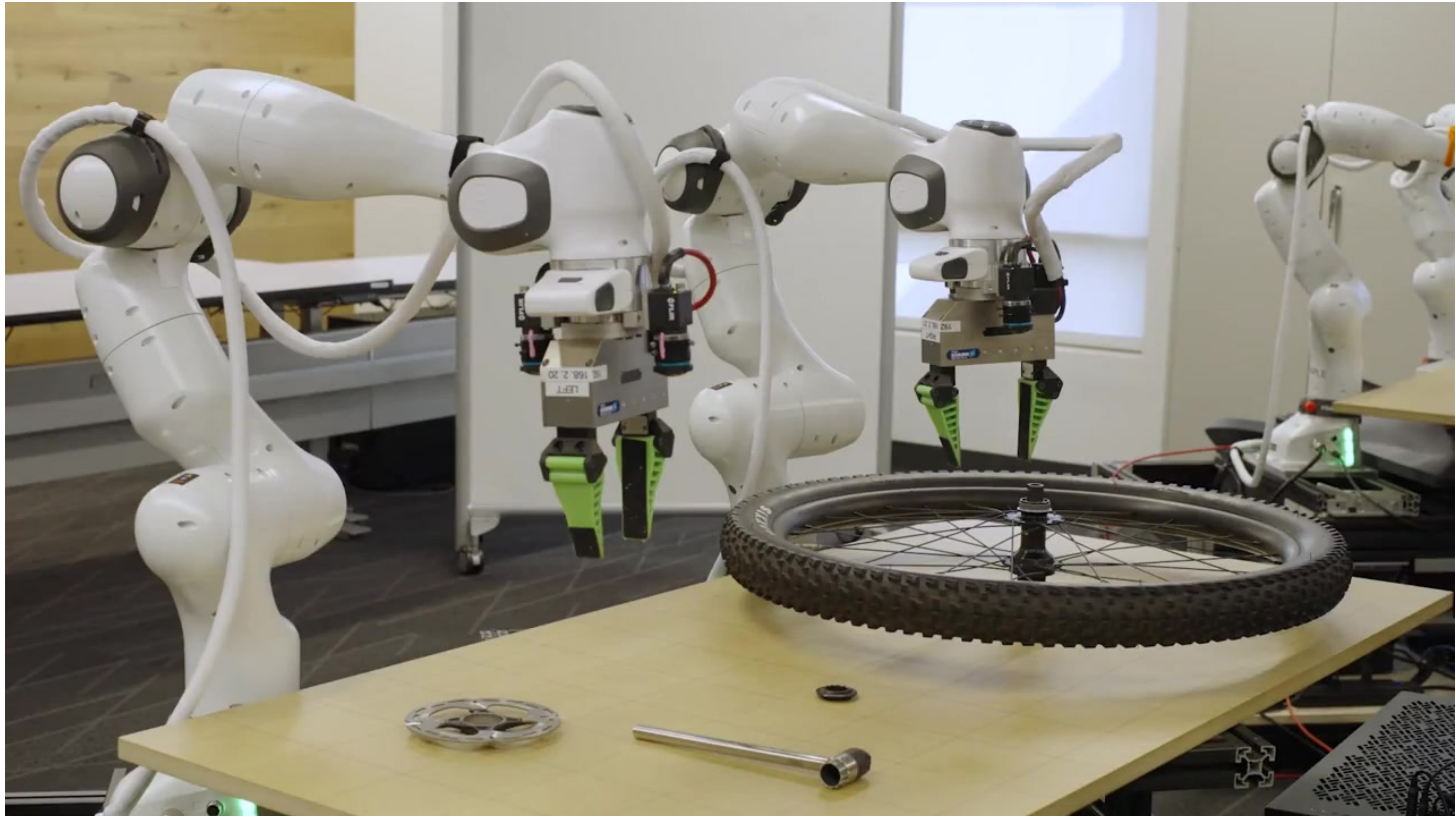
Can you guess which one's which?



LBM: performance on new tasks

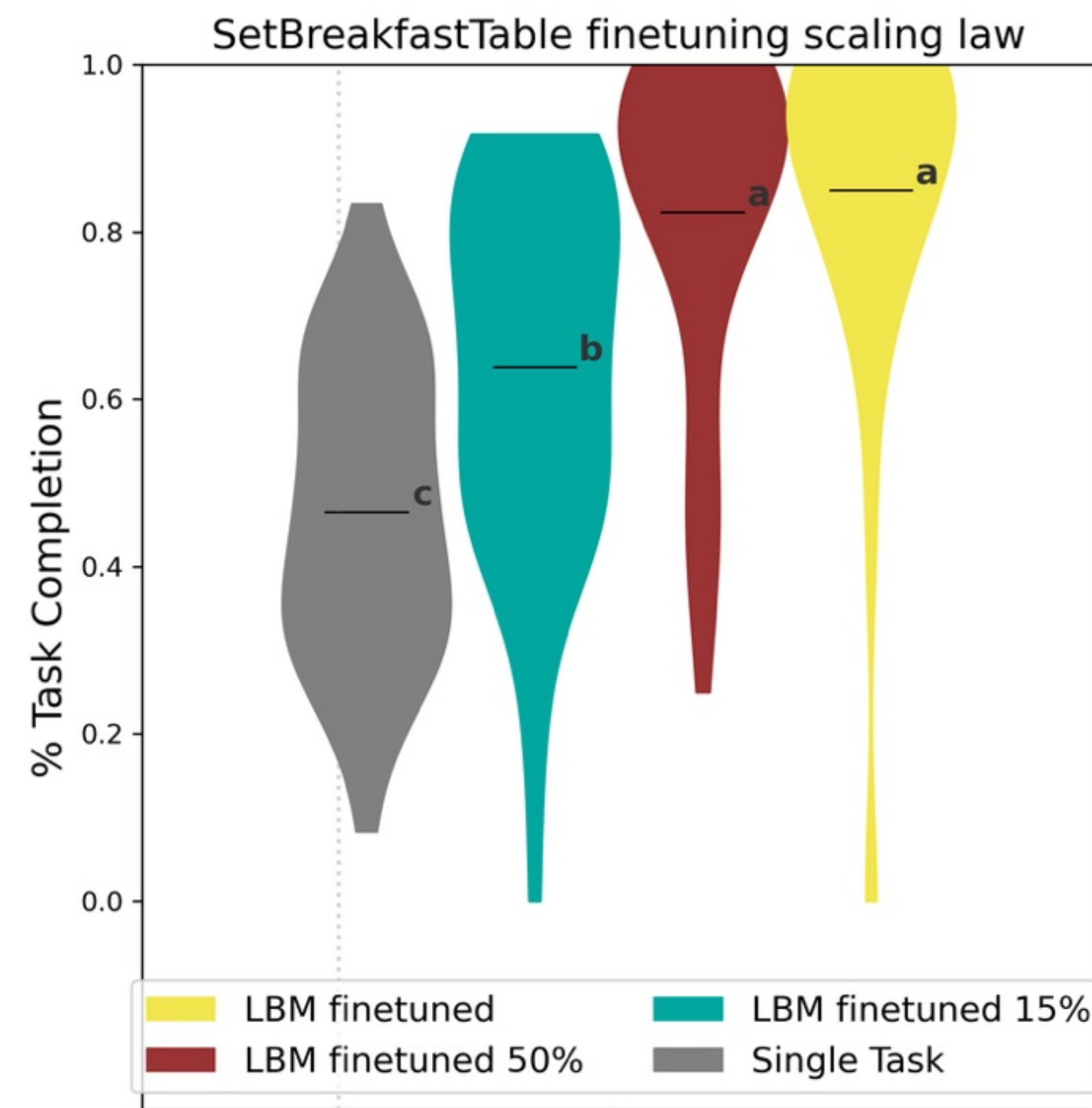


LBM: task execution



LBM: pretraining reduces the volume of data needed for fine-tuning

- Pre-training reduces target-data requirements by roughly 3–5 times
- Assessed by comparing how much data is needed to reach a certain performance level
- Caveat: performance level alone is too soft an indicator (policies with same % success or % task completion could have qualitatively different behaviors)



LBM: Summary

- A simple way to extend diffusion policy to multi-task settings
- Robust evaluation (statistical significance tests) of single- vs multi-task policies
- Multi-task pretraining lets us fine-tune over smaller volumes of data
- Not truly *general* (as we'll see with most other methods in this module, zero-shot deployment in a new environment and/or task is out-of-scope)

Early VLAs (2021-23)

Language-conditioned policy learning

- Task descriptions can specify which behavior a shared policy should execute
- Language-conditioned imitation learning (2020) and BC-Z (CoRL 2021) predate modern VLAs
- The *promise* of language-condition policies is
 - they will likely *generalize* better to new tasks in new environments (including ones that aren't seen before, at least in the robot-specific training dataset)
 - they can build upon strong priors already available in web-scale (VLM) pretraining

Gato (2022): a general sequence model

- Serialize observations and actions from different domains into a common sequence
- Train one transformer with a next-token objective
- Tokenizing all modalities enables shared (multimodal) training

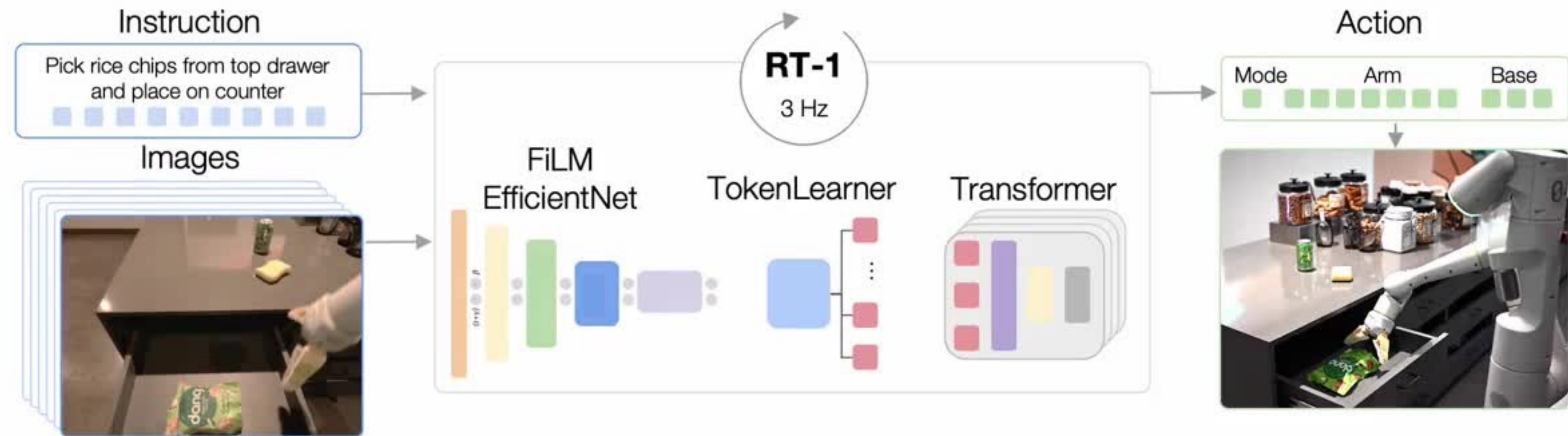


SayCan (2022): selecting executable skills with language and value models

- A language model scores which skill (among a predefined skill library) is useful for the instruction
- A value function scores whether that skill can succeed in the current state
- Score multiple skills; execute the best-scoring one; repeat



RT-1 (2022): images and actions as tokens



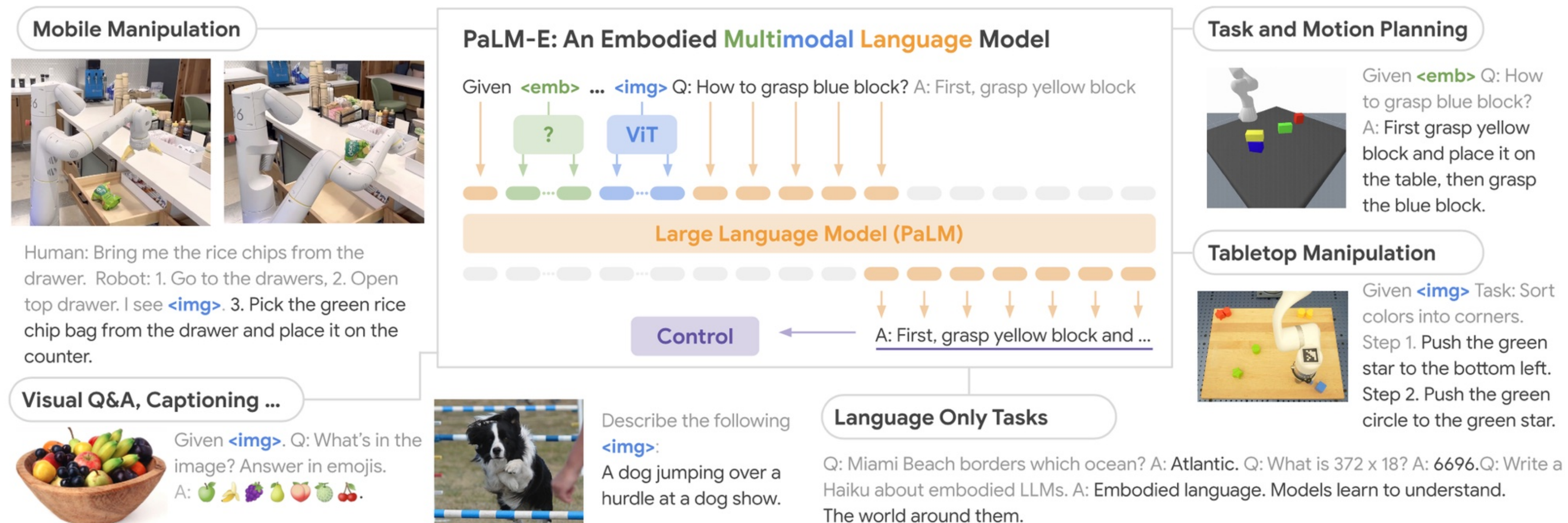
RT-1 (2022): one policy for multiple tasks

- About 130,000 episodes, collected with 13 robots, covering more than 700 tasks
- Actions are tokenized by discretizing into bins, and bins are consistent across all tasks



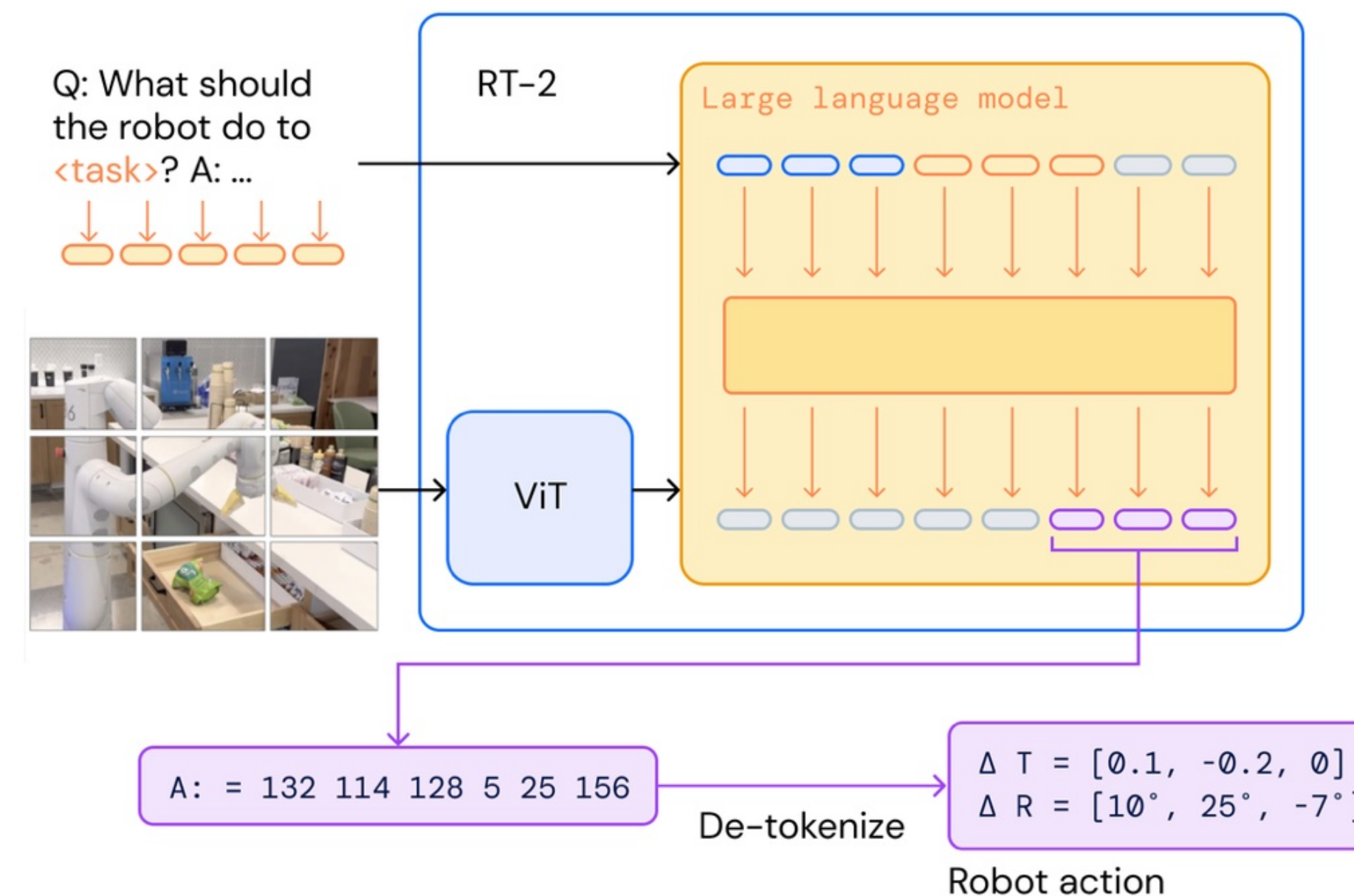
PaLM-E (2023): observations in an LM

- Embed robot observations into the language-model input sequence
- Model produces a text plan, which is decoded into robot actions by another module

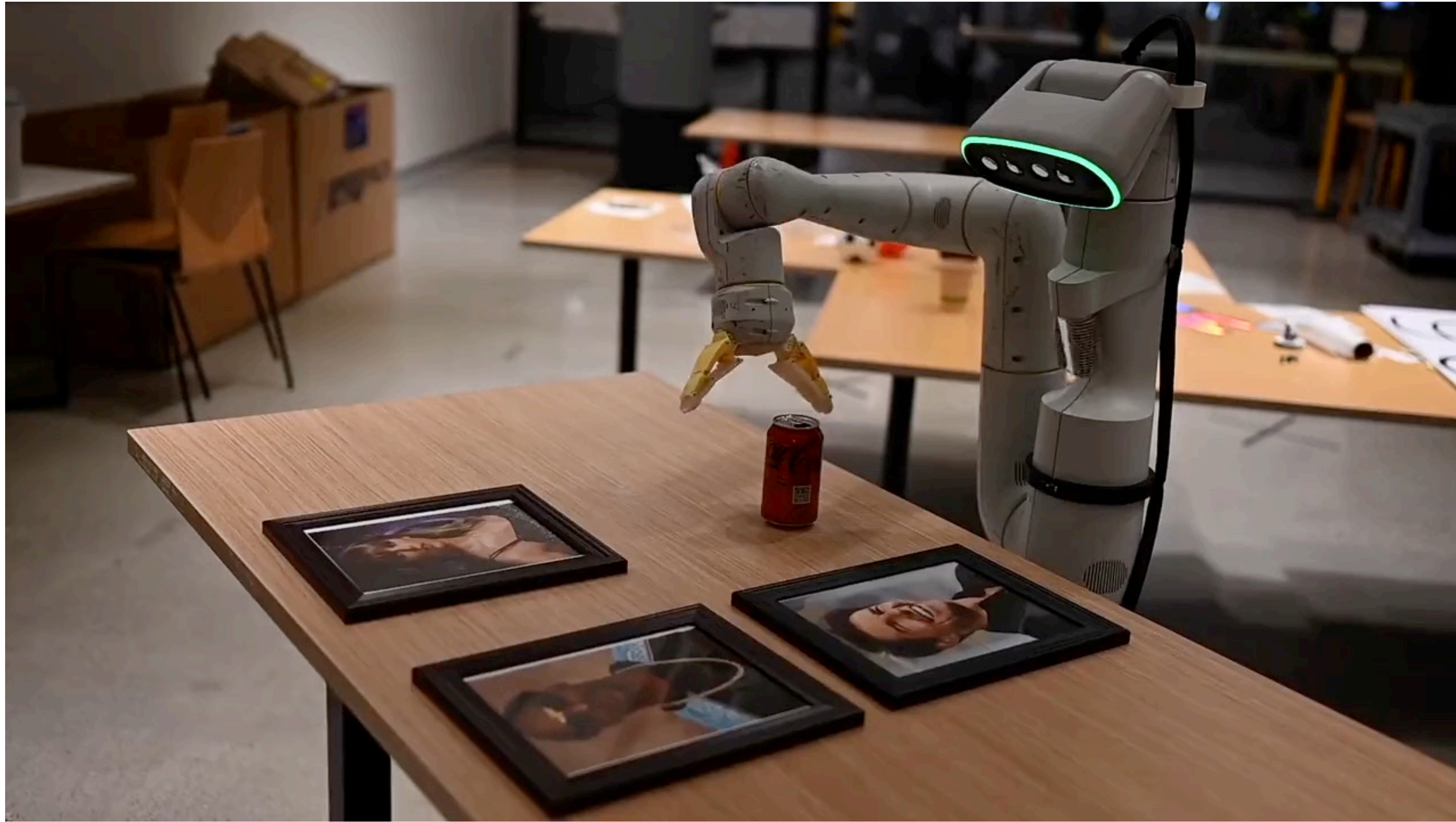


RT-2 (2023): transferring VLM knowledge

- Fine-tune a pretrained VLM on web examples and robot action tokens
- VLM predicts robot action *tokens* that are de-tokenized to get end-effector delta positions
- Semantics learned from web-scale pretraining help with *grounding* (identifying the object needed for the task)

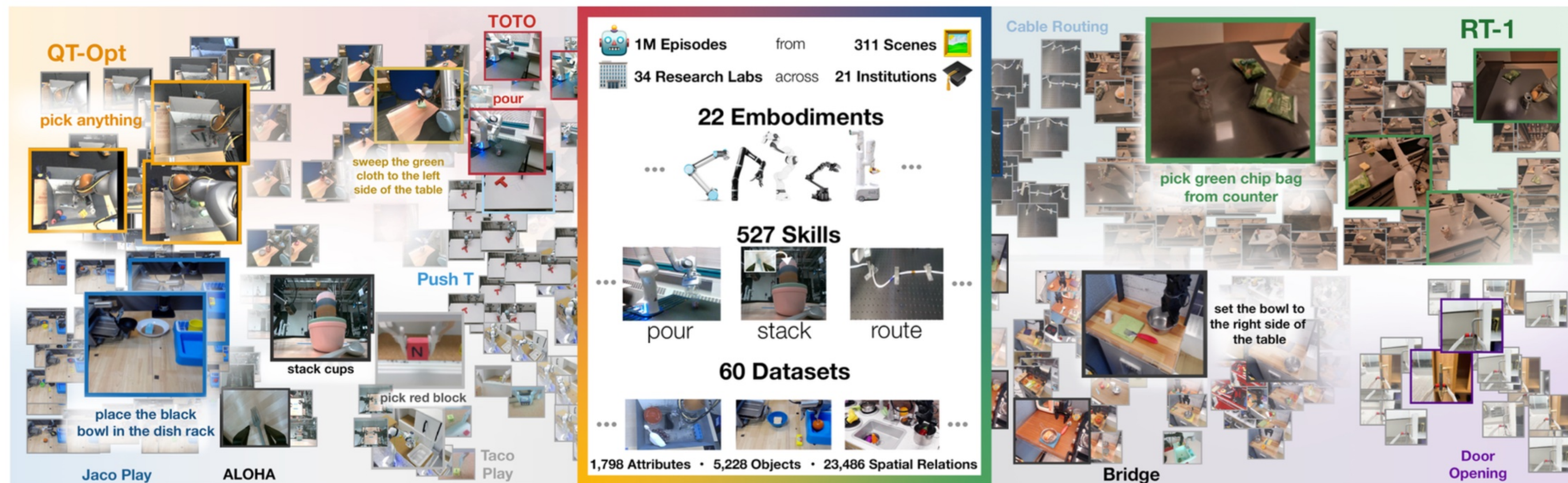


RT-2: leveraging web-scale pretraining for generalization

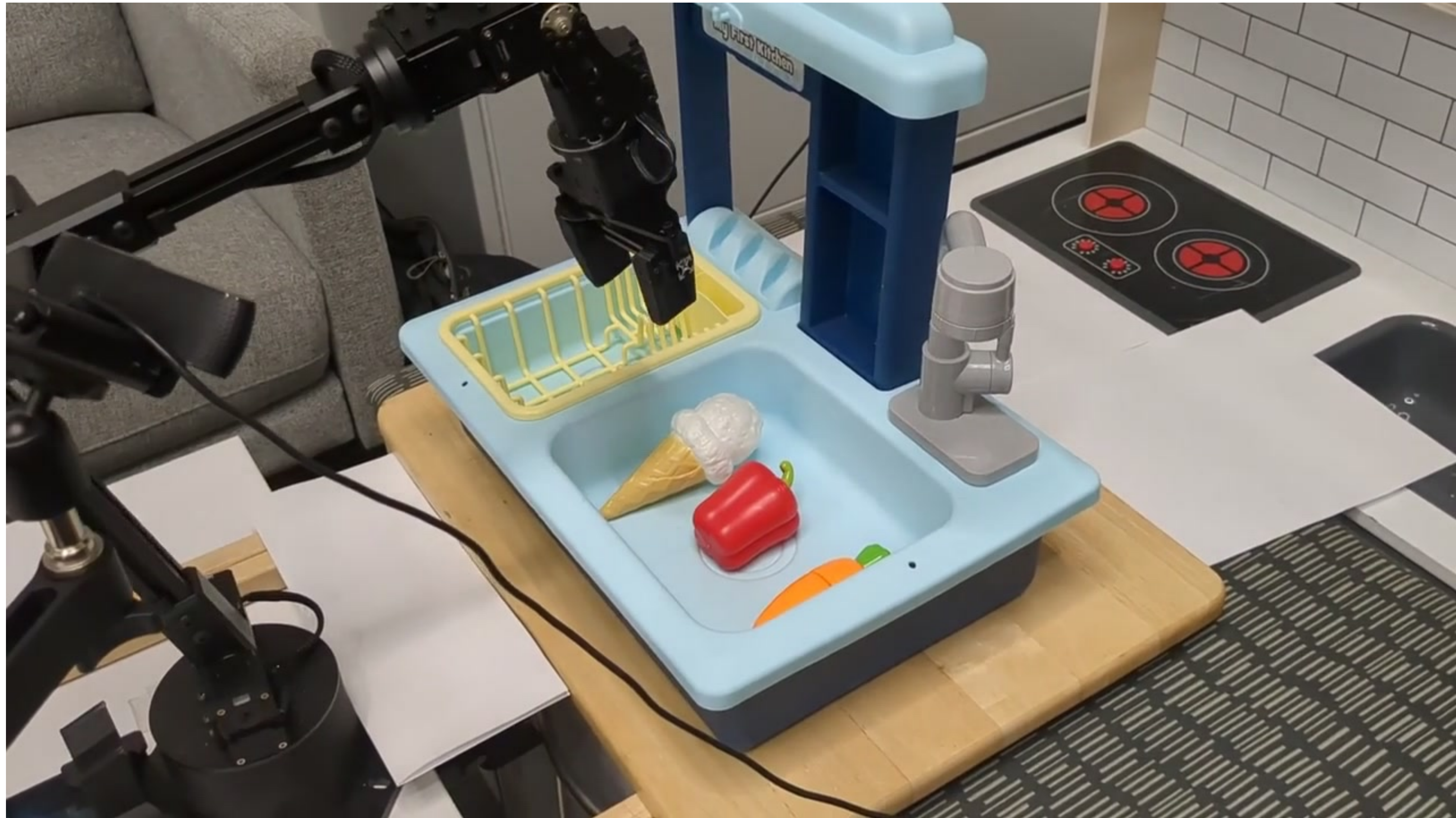


RT-X (2023): training across embodiments

- Pool demonstrations from different robot platforms in Open X-Embodiment
- RT-1-X and RT-2-X test transfer through joint training on compatible representations



RT-X: demo



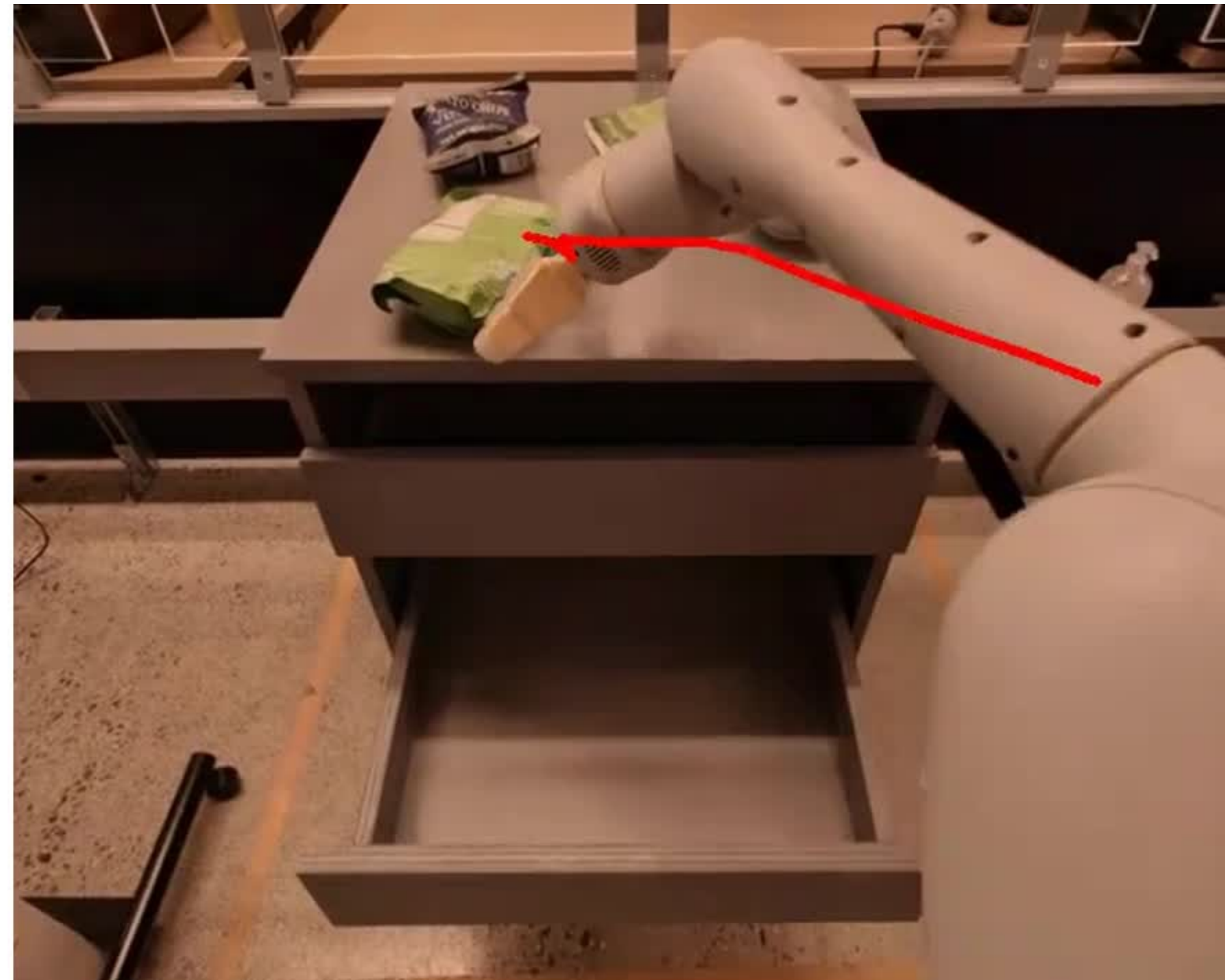
RT-H: leveraging action hierarchies

- Predict an intermediate language motion before generating robot commands
- The language motion makes the next action more explicit and can be corrected if unsafe



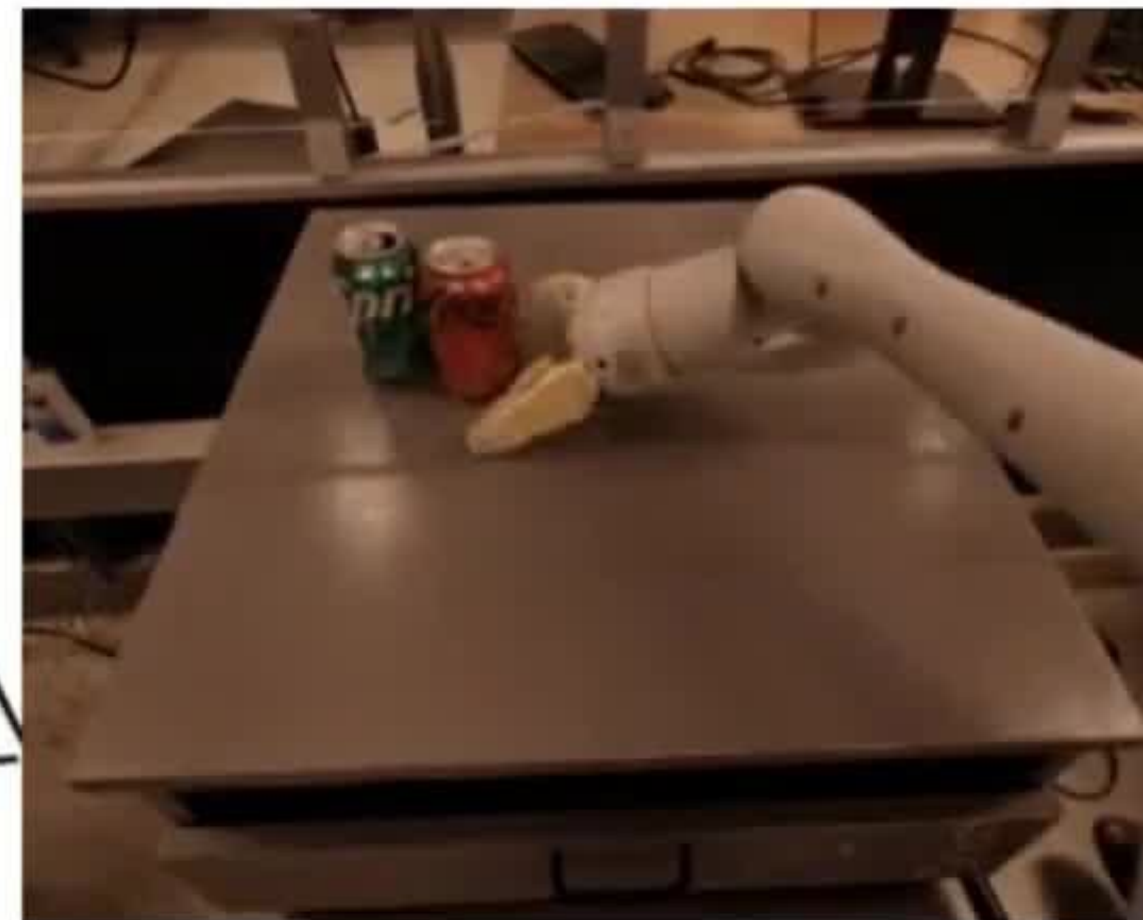
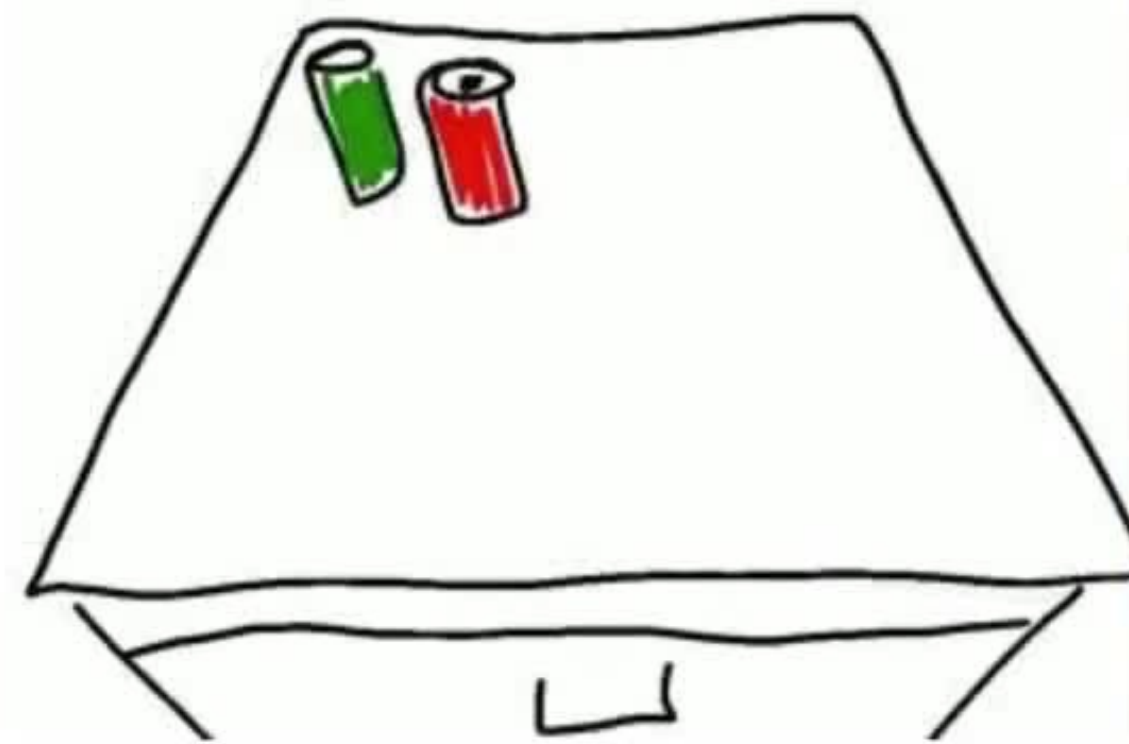
RT-Trajectory: specifying motion

- A trajectory sketch supplies intermediate spatial information
- Condition the action-generation policy on the image, instruction and desired trajectory



RT-Sketch; RT-Affordance

- RT-Sketch conditions a policy on a hand-drawn goal sketch
- RT-Affordance uses affordance annotations as intermediate task information



Early VLAs: Recap

- Early VLAs relied largely on the capabilities of a base VLM policy for generality
- Most of the demos (pick-place tasks) focus on generalization at the semantic level (i.e., new objects in new settings; new language instructions)
- Limited complexity in robot actions (which necessitated skill selection, hierarchies, sketches, two-level action decoding, etc.)
- The next generation of VLA models that followed the RT series (and which we'll see next) go the opposite way (emphasize on large-scale imitation learning to generate more complex robot behaviors; often at the expense of semantic generalization)

Vision-Language-Action (VLA) models since then

Key hurdles/problems* for generalist^ policies

- Problem 1: How do we effectively represent robot data for policy learning?
 - Tokenization methods (we will see FAST, VQ-BeT, OAT), pi0
- Problem 2: How do we efficiently learn from multiple data sources?
 - NVIDIA GROOT N1 / N1.5 models (latent actions unify data sources)
- Problem 3: How do we retain web-scale pretrained knowledge?
 - pi0.5, knowledge insulation
- Problem 4: How do we learn new tasks efficiently?
 - Few-shot and In-context learning

* Non-exhaustive list

^ Not the company *Generalist AI*

Problem 1: Representing robot data for generalist policies

Problem 1: representing robot data

- How to *tokenize* robot pose and action data
- Several cameras, robot state and commands may arrive at different rates
- Across embodiments, sensor configuration, action spaces, teleop controllers, etc. will inevitably vary
- Training general-purpose policies that operate on any robot require accounting for all this
- Nominal solutions to this problem include time-synchronization of observations and actions across embodiments, and normalization/standardization of data
- Joint-space action representations are robot-specific, but unambiguous. End-effector space action representations tend to be robot-agnostic (esp. in cases where the same gripper, e.g., UMI, is used across embodiments), but are ambiguous

Tokens, embeddings, latent actions

- Embedding -> continuous feature vector
- Token -> (usually) discrete feature that indexes a vocabulary or codebook
- Robot actions can be discretized or predicted continuously
- In a later subsection, we will briefly discuss *latent actions*, which are learned action spaces that align multiple robot datasets/embodiments (and non-robot data) and are useful when command labels are unavailable

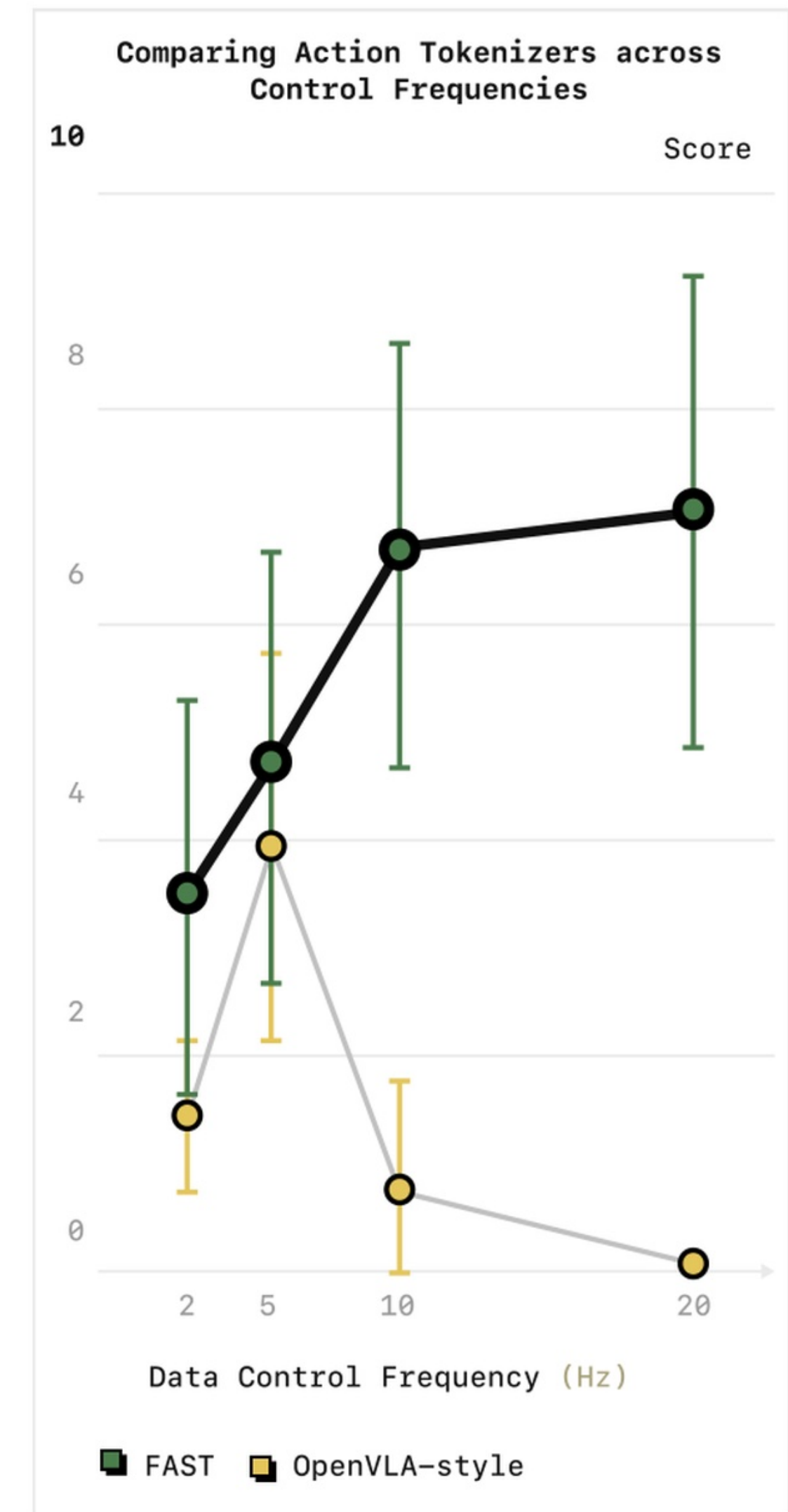
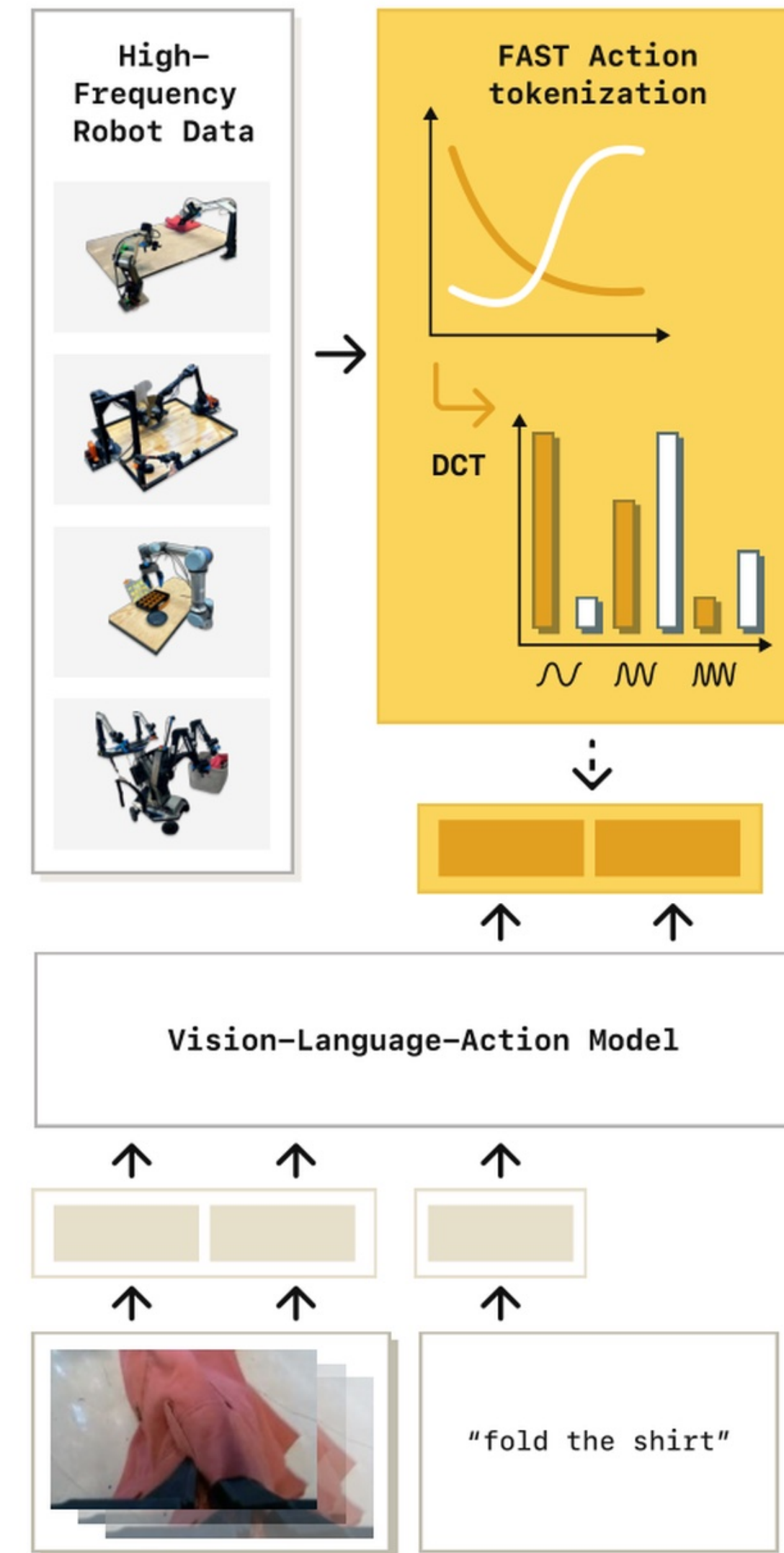
Early VLA tokenization: Uniform scalar quantization

- Normalize each coordinate, assign a bin, then serialize the values
- A 50-step chunk with six coordinates requires 300 scalar symbols
- Temporal and cross-coordinate structure are left for the policy to learn
- Clip the command to the chosen physical range before assigning a bin
- Each coordinate receives one index in a finite vocabulary
- Decoding a bin recovers a representative value within that interval
- Non-uniform quantization is possible too, and used (e.g., OpenVLA)

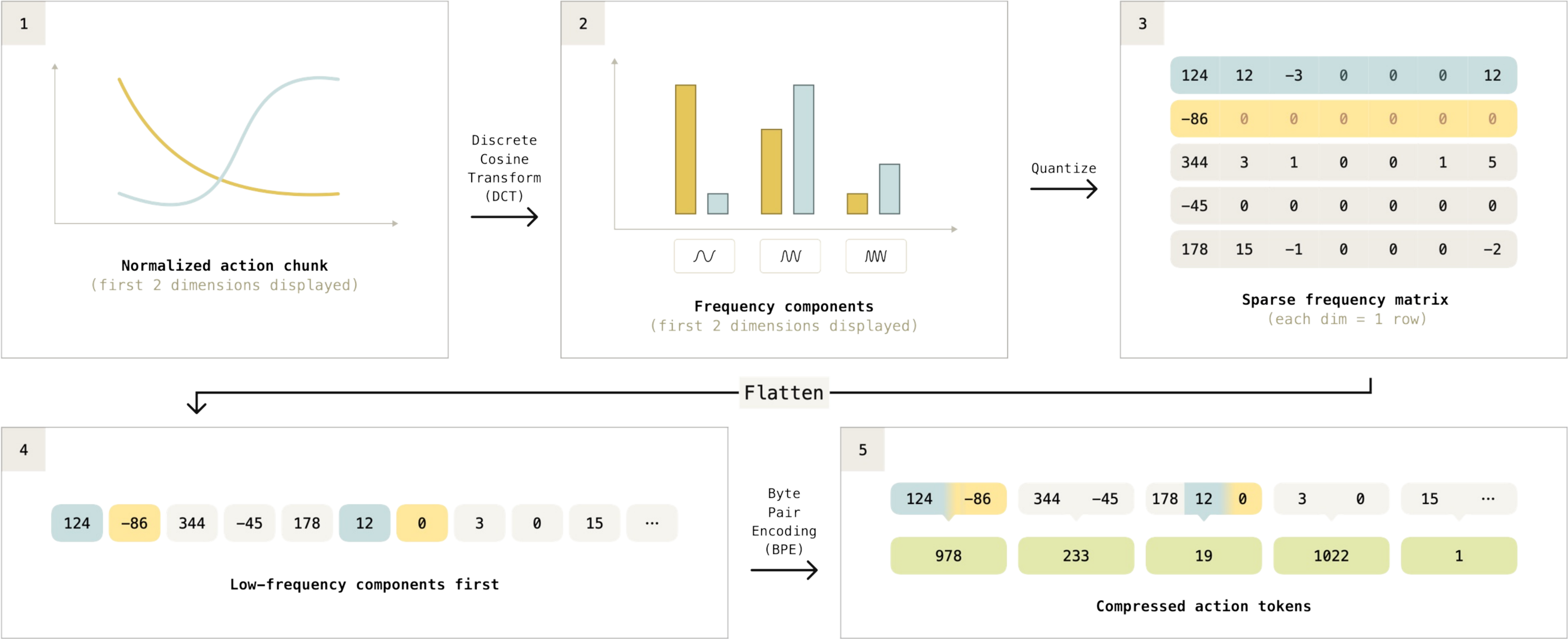
$$z = \min\left(K - 1, \left\lfloor K \frac{a - a_{\min}}{a_{\max} - a_{\min}} \right\rfloor\right)$$

FAST: frequency-space tokenization

- Discretization/binning starts to fall-apart at higher action prediction horizons / frequencies (e.g., 25-50 Hz)
- Sequence length increases (i.e., “number of bins x length of sequence” actions need to be predicted)
- At higher frequencies, consecutive actions are near-identical, which causes learnability problems for autoregressive models
- FAST -> Frequency-space Action Sequences Tokenization



FAST: frequency-space tokenization



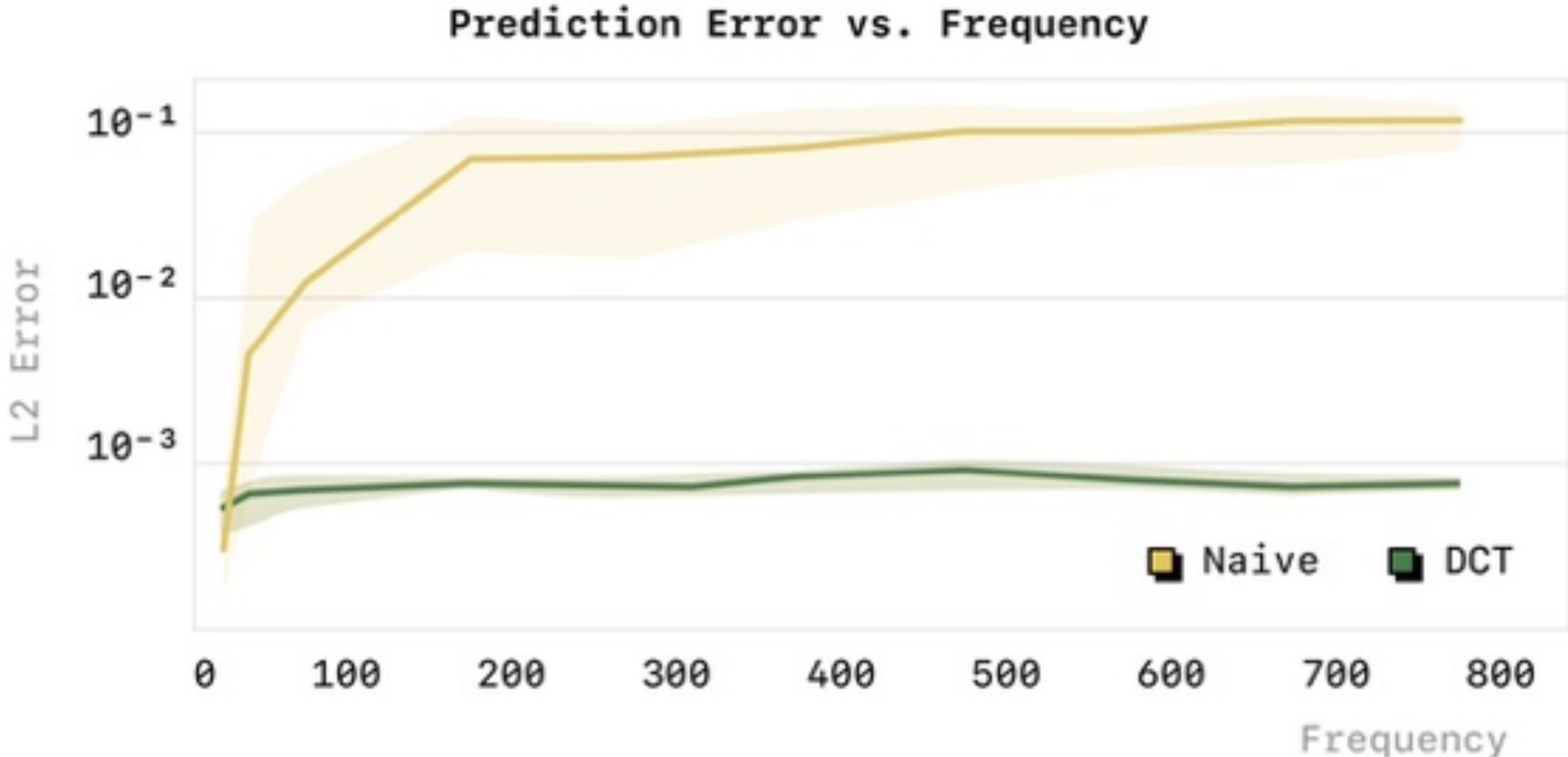
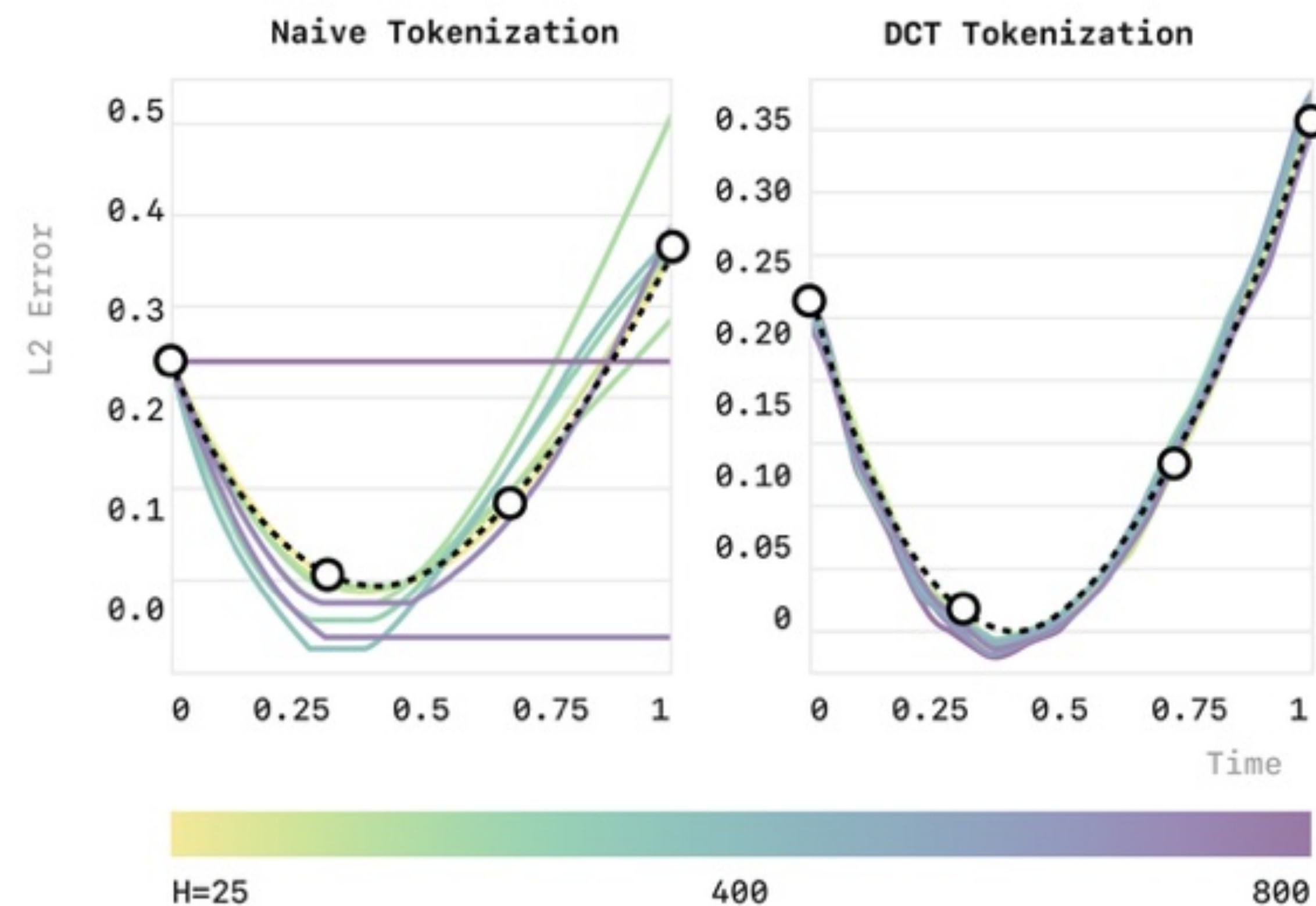
FAST: overview



$$c_k = \alpha_k \sum_{h=0}^{H-1} a_h \cos \left[\frac{\pi}{H} \left(h + \frac{1}{2} \right) k \right]$$

- Normalize action chunks (use 1st and 99th percentile; map linearly to [-1, 1])
- Apply discrete cosine transform (DCT, type-II — widely used in JPEG) to get H scalar coefficients
- Intuitively, first coeff is mean, second coeff captures best cosine fit, higher order coeffs capture higher-freq components
- Quantize the resulting coeffs (multiply by γ and round to nearest integer). This is *lossy*. Larger γ => higher precision
- Flatten the coeff matrix and sort lowest-freq first (this ensures zeros / low-values cluster at the end of the sequence — helpful for next steps; also decoding becomes coarse-to-fine by default)
- Train a standard byte-pair encoding (like in LLM tokenization) over these sequences
- Results in a variable-length sequence of (discrete) tokens. Usually much smaller than the input sequence
- Decoder works in the reverse!

FAST: results



FAST: Results

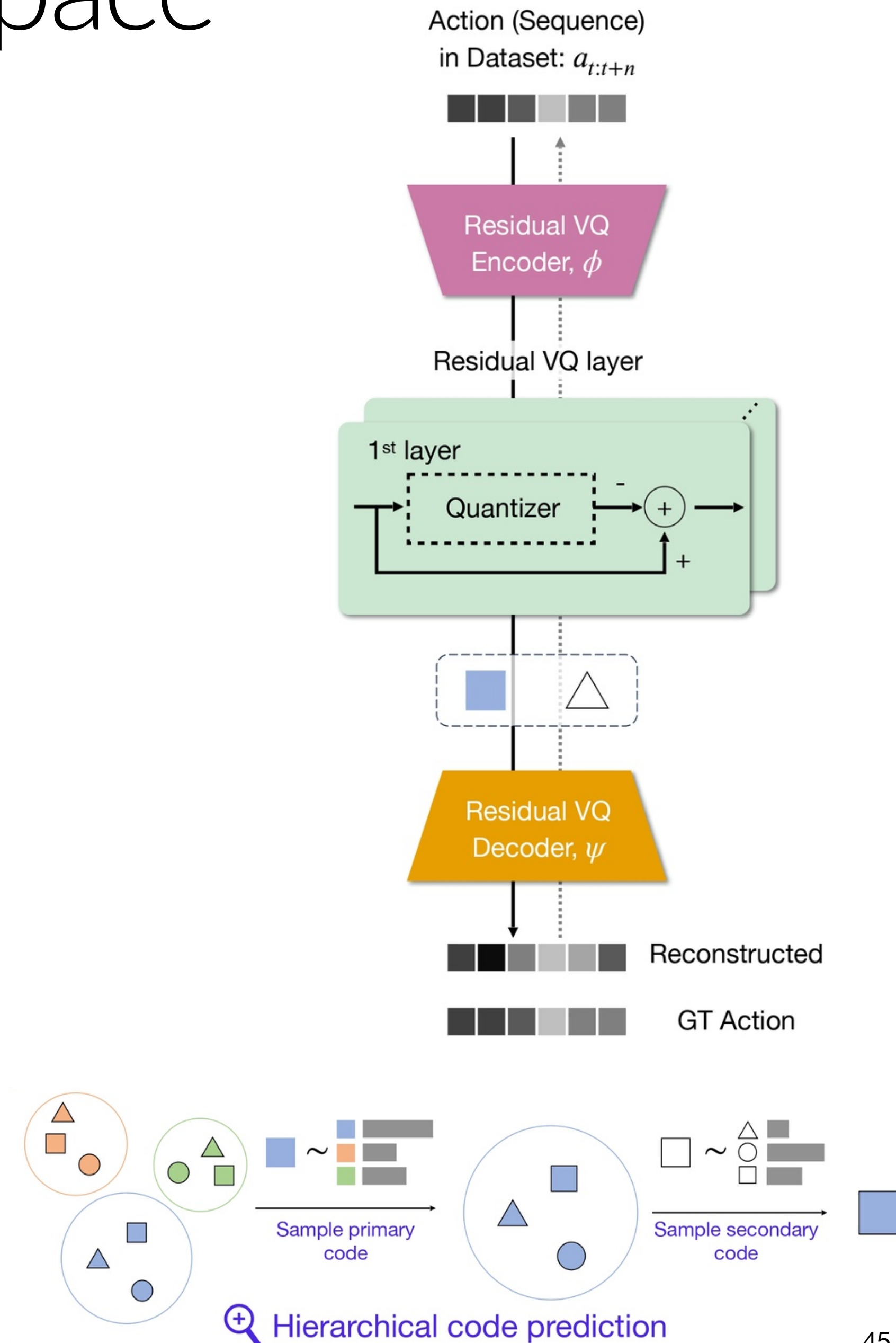


FAST: Caveats

- DCT-basis is fixed (i.e., not the most efficient representation tailored/optimized) to a particular downstream dataset
- Token validity and variance: FAST is variable length. Token sequences are non-unique. Invalid token sequence can fail to decode (in such cases, FAST resamples tokens)
- Ordering is coarse-to-fine (not necessarily by importance)
- Uniform precision everywhere, determined by γ
- Entire chunk needs to be decoded before even a single action is executed

VQ-BeT: learning a discrete action space

- Action tokenizer is a VQ-VAE (a vector-quantized variational autoencoder)
- Note: VQ-BeT uses a shorter action chunksize (3-10)
- A small MLP maps the chunk to a continuous latent vector, which is quantized using 2 *codebooks* in sequence
- Codebook 1 has K learnable vectors, and transforms z to a code e_1 (residual $r = z - e_1$). Codebook 2 quantizes the residual r to a code e_2
- Quantized latent is $e_1 + e_2$ and the token is (k_1, k_2) — indices of e_1 and e_2 in the 2 codebooks respectively
- Decoder maps $e_1 + e_2$ (quantized latent) back to the action chunk



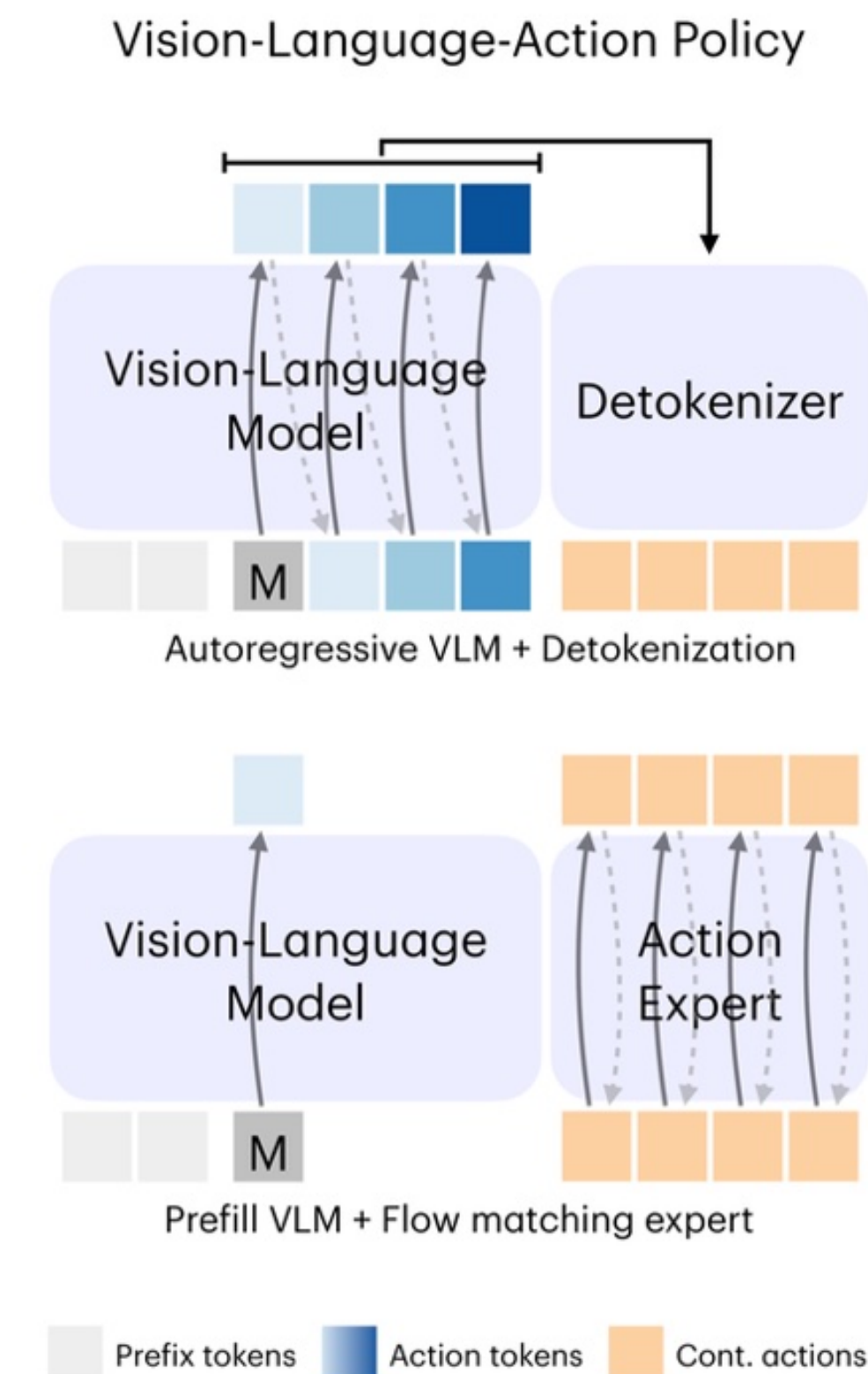
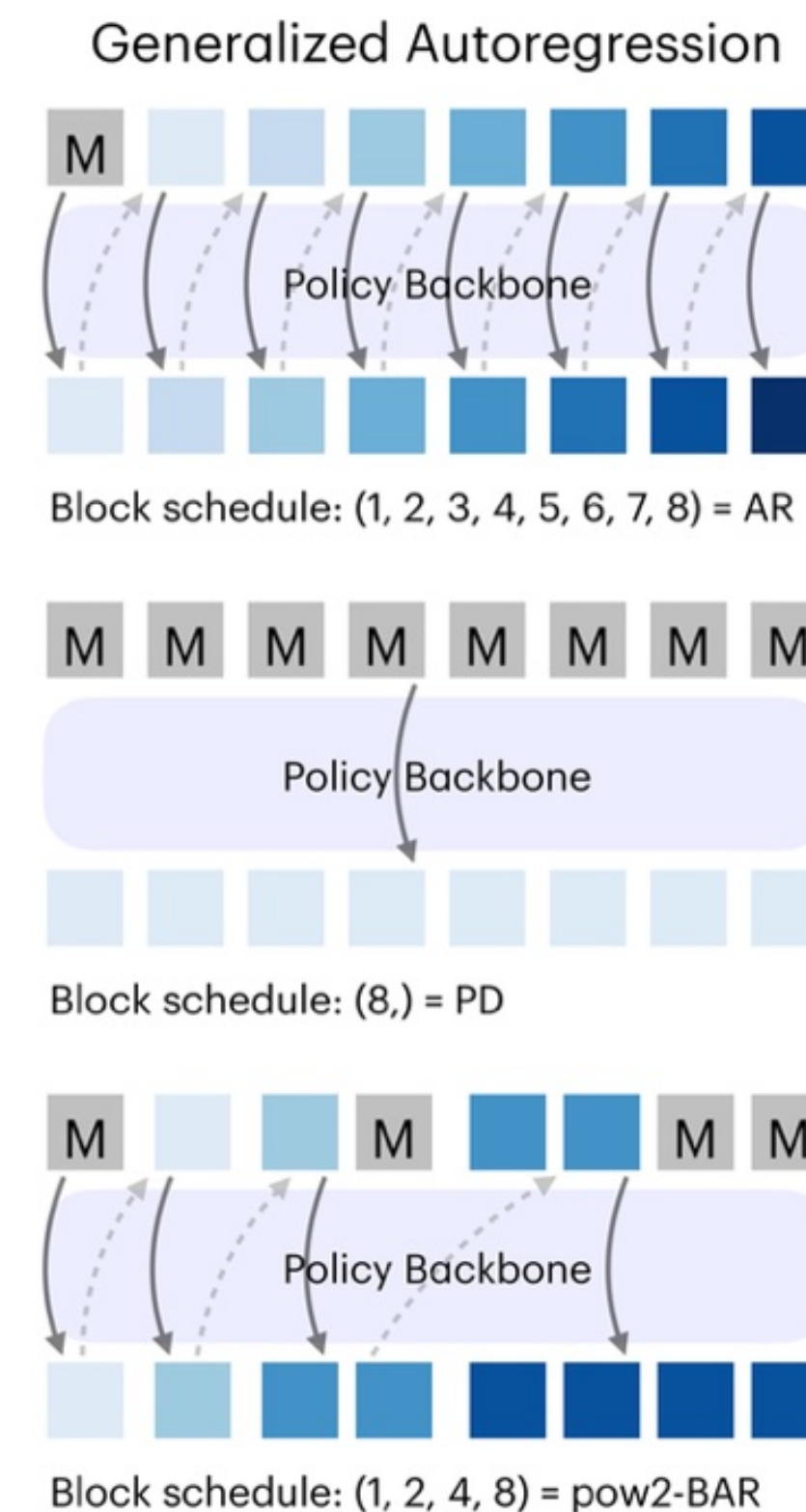
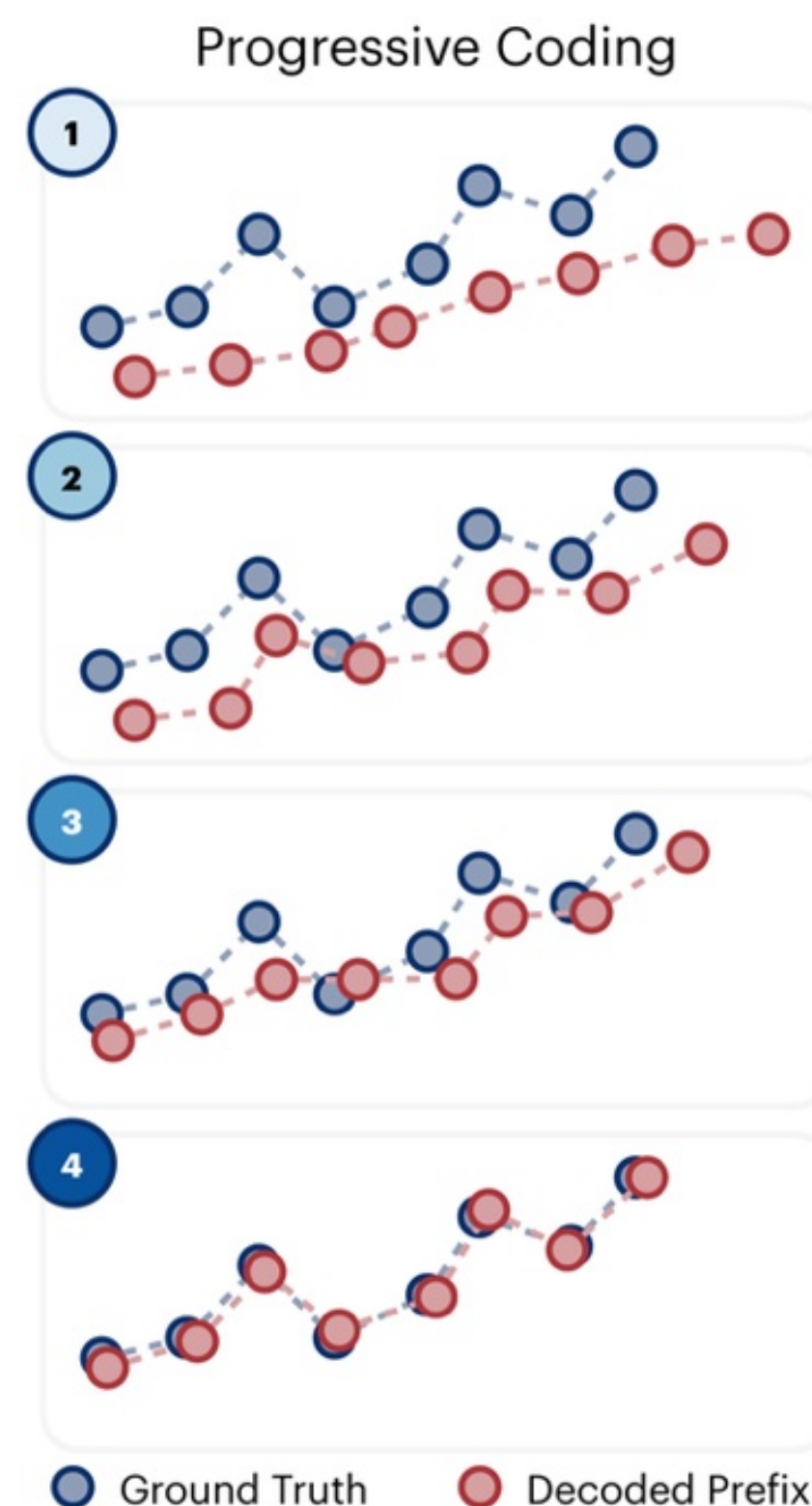
VQ-BeT: Summary and Caveats

- Learned representation (codebook is tailored / optimized to fit the demonstration)
- Fixed-length (opposed to FAST, which is variable-length)
- Coarse-to-fine
- One-shot decoding (unlike fast)
- However, only a 2-level hierarchy is explored, and codes can't quite represent fine detail or high-frequency motion (unlike DCT, where high-frequency details can be captured)
- VQ-BeT is fragile to train (codebook collapse, dead codes, sensitivity to hyperparams)
- Our next tokenizer fixes these (and other) issues

OAT: ordered action tokens

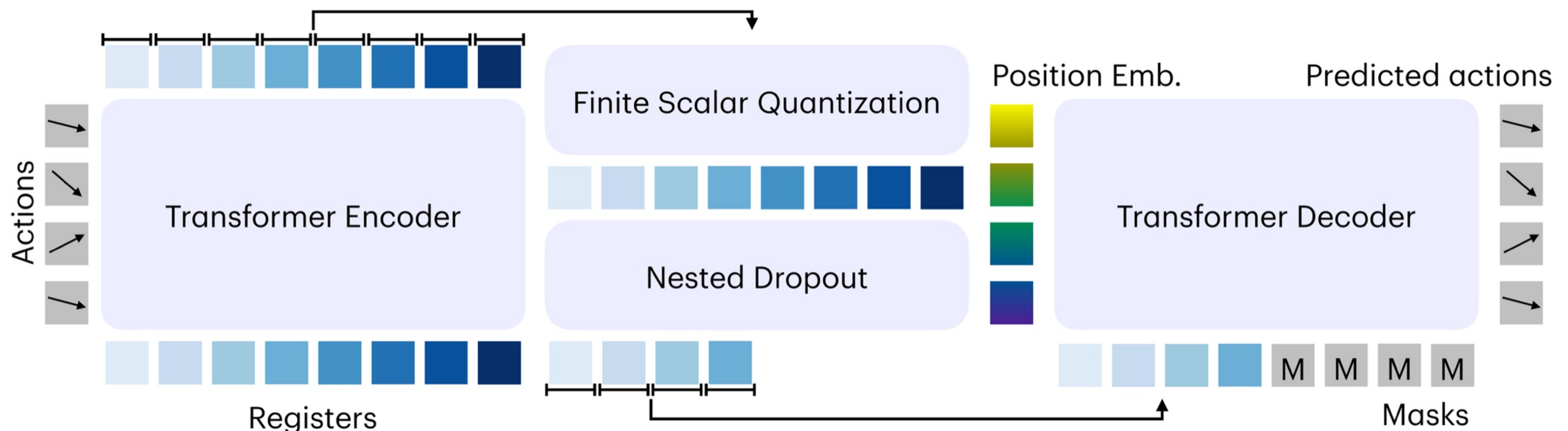
- Motivations:

- A short token prefix should reconstruct the entire action chunk (progressive coding)
- Later tokens refine this reconstruction
- Tokens need not be decoded in time-order



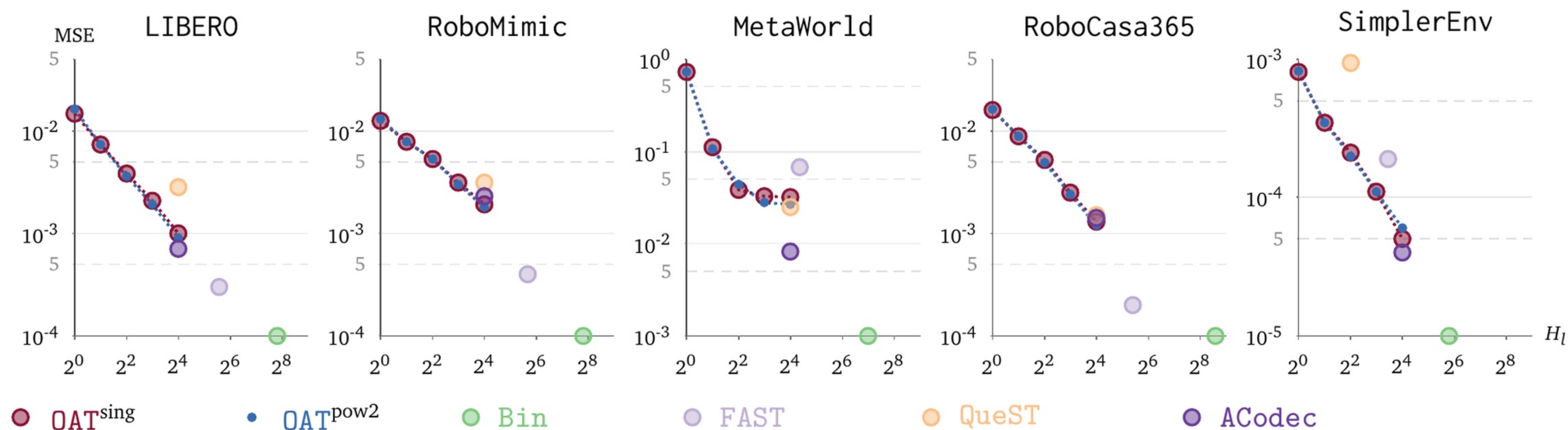
OAT: Approach

- OAT tokenizer has a transformer with learnable *queries* (input to the encoder is learnable queries + action chunk to be encoded)
- Learnable queries cross-attend to the action chunk, and result in a token embedding each
- Each token embedding is discretized (using an FSQ-based quantization scheme)
- Prefix / nested dropout is used during training. For each chunk, sample random prefix length and throw away the token that follow, and get the decoder to reconstruct the full chunk from just the prefix
- Trained with reconstruction loss



OAT: choosing a token budget

- More tokens can improve reconstruction, at the cost of additional generation steps
- Can also use token supervision to train context for a continuous action expert



Tokenization: Notes and summary

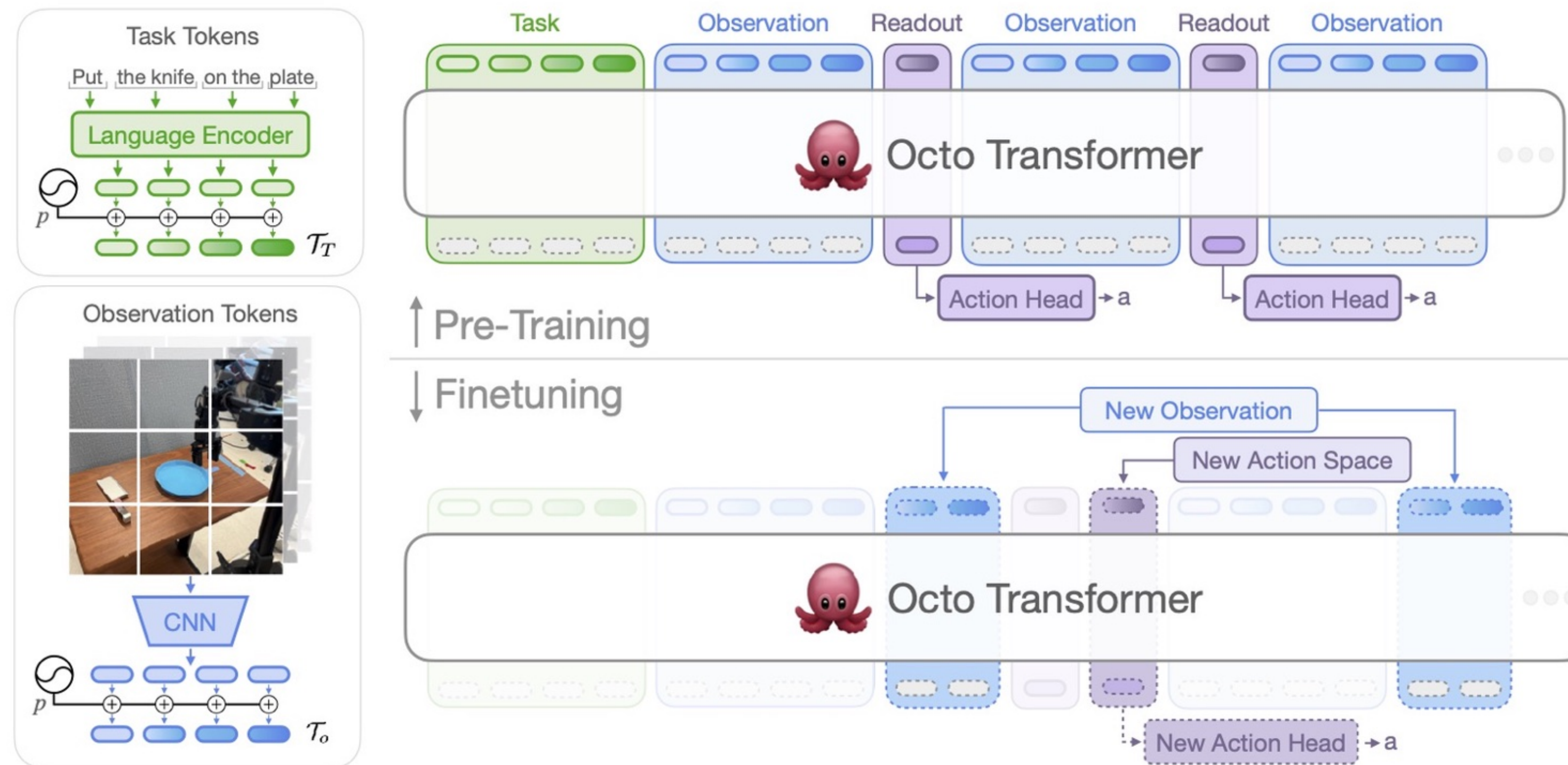
- Tokenization methods (FAST, VQ, OAT) matter primarily for autoregressive models
- They enable training recipes optimized for discrete tokens (e.g., LLM/VLM training) to be easily adapted to robot policies
- An increasingly large class of learned policies have since moved away from autoregressive prediction, in favor of directly representing actions as continuous vectors (e.g., ACT, diffusion policies)
- VLA models that use such continuous parameterizations add a diffusion/flow matching head (a.k.a. action expert / action decoder), getting around the issue of tokenization entirely

Continuous actions and how to represent them for multi-task policies

- Modern multi-task policies often directly decode continuous (normalized) action chunks
- Some choices / tradeoffs to consider:
 - Absolute vs Delta actions
 - End-effector space vs joint-space
 - EEF deltas are easier to train across embodiments, but are ambiguous (IK solvers and underlying controllers can result in OOD states / singularities)
 - Joint deltas are unambiguous, but each embodiment has a different number of joints / limits (plus there's higher cross-task variance in joint configurations)
- Relative-to-previous vs relative-to-first action in chunk
 - Relative-to-previous action in chunk => error accumulates over time;
 - Relative-to-first action in chunk => error at every timestep is independent; potentially allows to skip through some actions

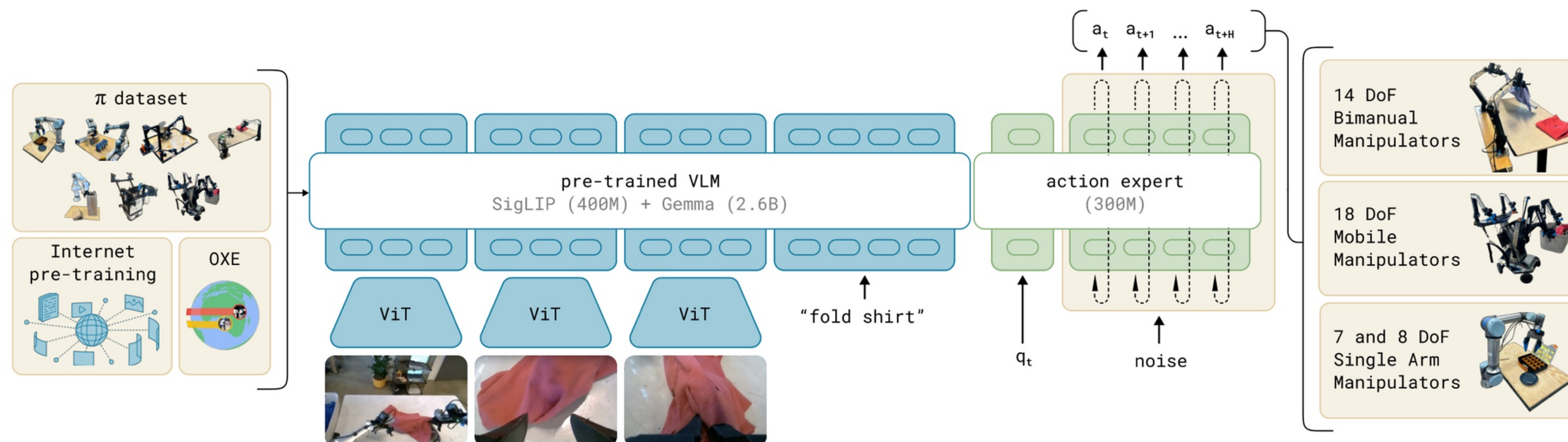
Octo: modular policy interfaces

- One of the earliest VLA models that had a modular architecture and predicted continuous actions
- A shared transformer processes observation tokens and language or goal-image conditioning
- A diffusion action head generates continuous commands



pi0: cross-embodiment vision-language generalist policy

- Combines a retrained vision-language model with a flow-matching (or autoregressive) action generation head (action expert)
- A ~3B PaliGemma backbone used for vision-language context
- A smaller ~300M action expert generates continuous action chunks using flow matching
- Discrete variants exist too (autoregressive)
- Output action space denotes predicted actions for every embodiment in the training data — but is supervised only against the embodiment that corresponds to each demonstration



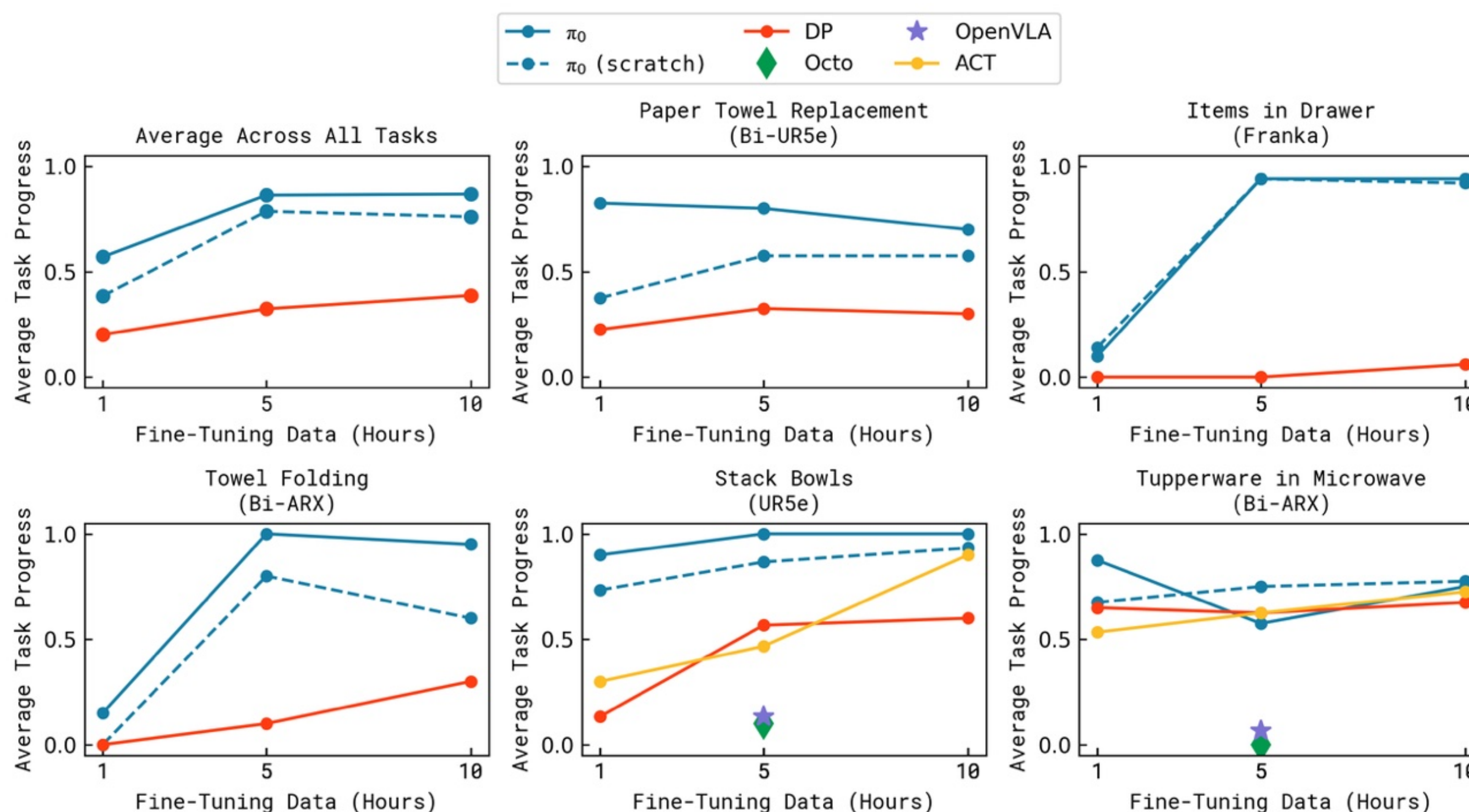
pi0: computation during a rollout

- Encode the context, integrate the action expert, and execute a prefix of the chunk
- Acquire new observations and repeat
- Recall discussion from previous lecture: prediction and execution horizons can be different; real-time action chunking

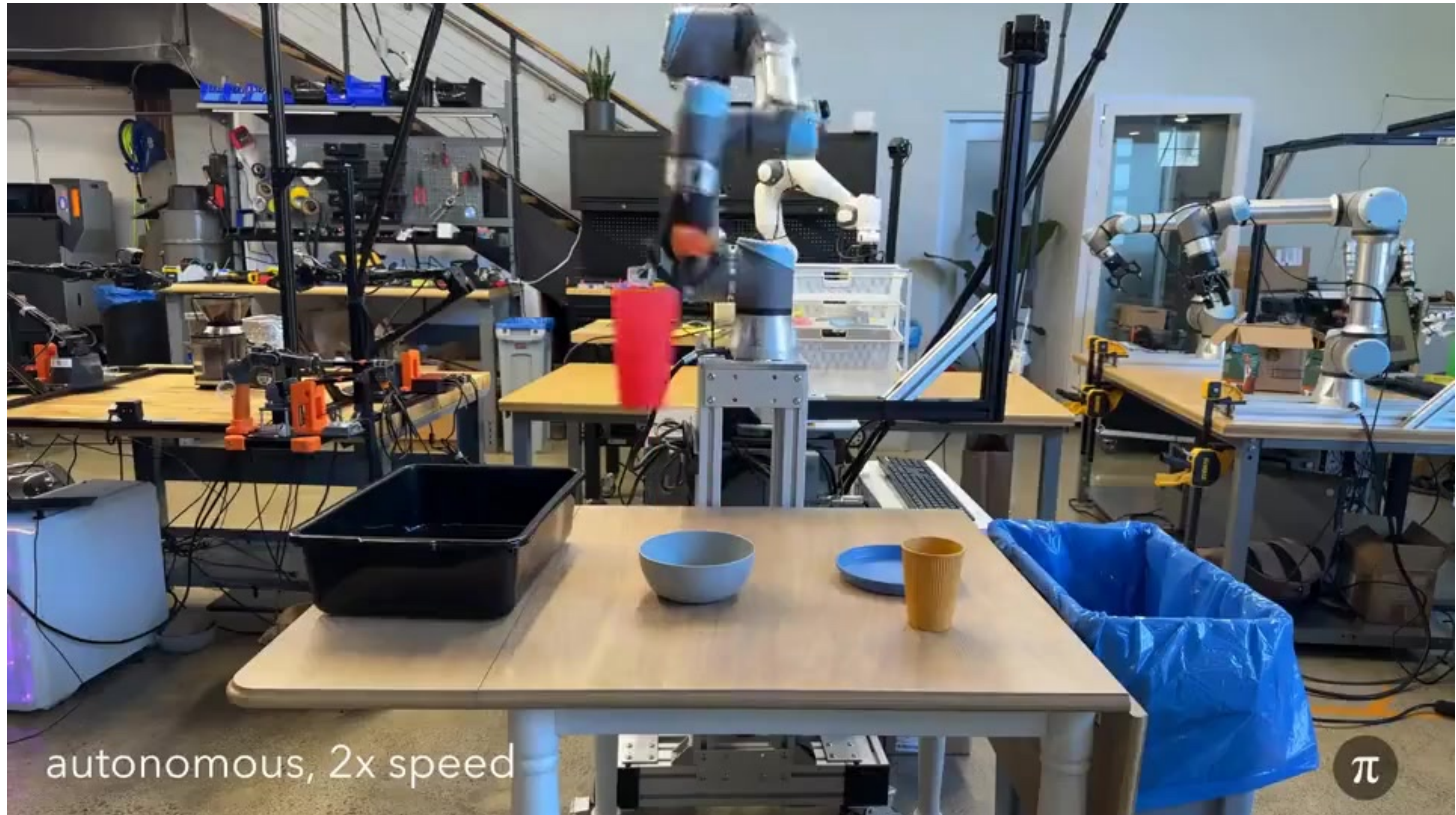


pi0: results

- The base policy transfers across tasks in its pretraining distribution
- Post-training is needed for more complex behaviors not captured in pretraining

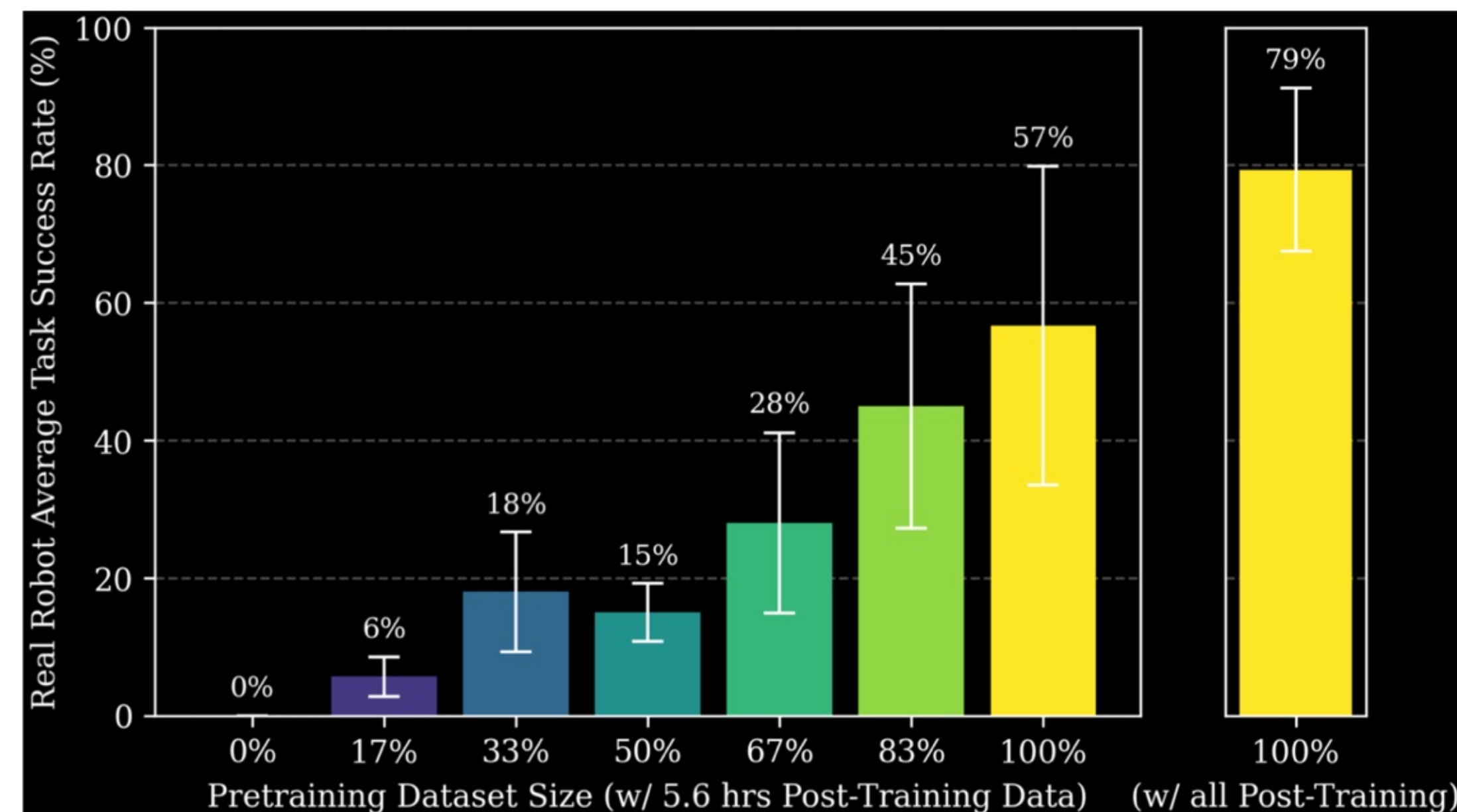


pi0: post-trained manipulation



GEN-0: Modeling asynchronous data streams

- Caveat: Not too many details disclosed in the GEN-0 blog post
- Generalist describes multirate sensing and action streams in its Harmonic Reasoning framework
- Shared pretraining can scale while each stream retains different output rates
- Network architecture, input encoding, output representation, etc. not disclosed



Problem 2: Learning from multiple data sources

Main data paradigms for robot learning today*



Robot data

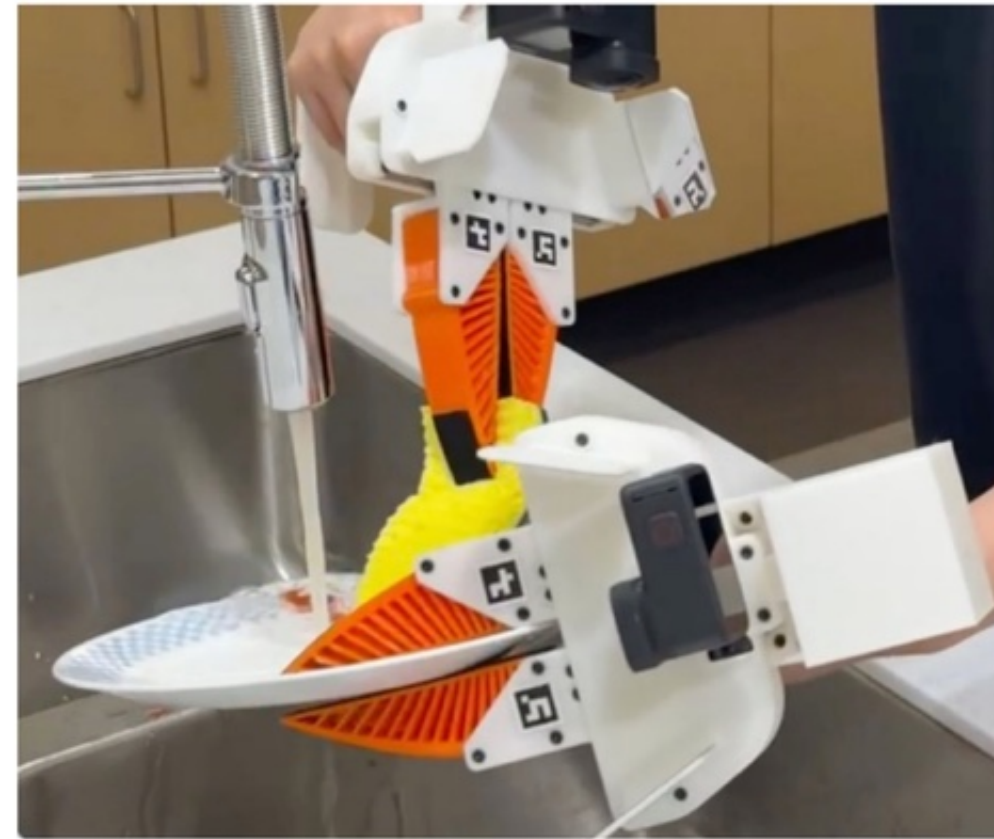
(Joint-level teleoperation)

Pros:

- + Kinematic feasibility
- + Fully-specified trajectories

Cons:

- Need kinematic twin of robot (at least a scaled-down one)
- Needs specialized infrastructure (harder to scale)



Hybrid data

(Wearable interfaces)

Pros:

- + Easy to collect in-the-wild
- + Better scalability

Cons:

- Data collection needs careful management by operator
- Restrictions on robot end-effectors



Human data

(Hand / body-tracking)

Pros:

- + Easy to collect in-the-wild
- + Highly scalable

Cons:

- Data needs to be retargeted to robot body
- Data quality is noisier than other paradigms

GR00T N1: learning from multiple sources

- Leverage all data sources (web data, simulation, real robot data, human data)
- However, each data source comes with a different set of assumptions and input/output signature (e.g., web data / human data doesn't have robot actions)
- Simulation can be a useful tool to synthesize / augment existing trajectories (e.g., variations of objects / motion that aren't captured in real-world demonstrations)
- Learned video-generation models can also be used for data augmentation — however, this data may not contain robot state and/or action labels
- How do we build one model that ingests all sources of data?



GR00T N1: learning from multiple sources

- Data sources used
 - Robot data (teleop) ~88 hours
 - Human egocentric videos
 - Neural data ~740 hours
 - Simulation data (DexMimicGen) ~6500 hours
- Neural trajectories: use video get models to generate variations of captured trajectories. These are annotated with actions using an ‘inverse dynamics model’
- How do we build one model that ingests all sources of data?

Prompt: use the right hand to pick up cucumber to basket



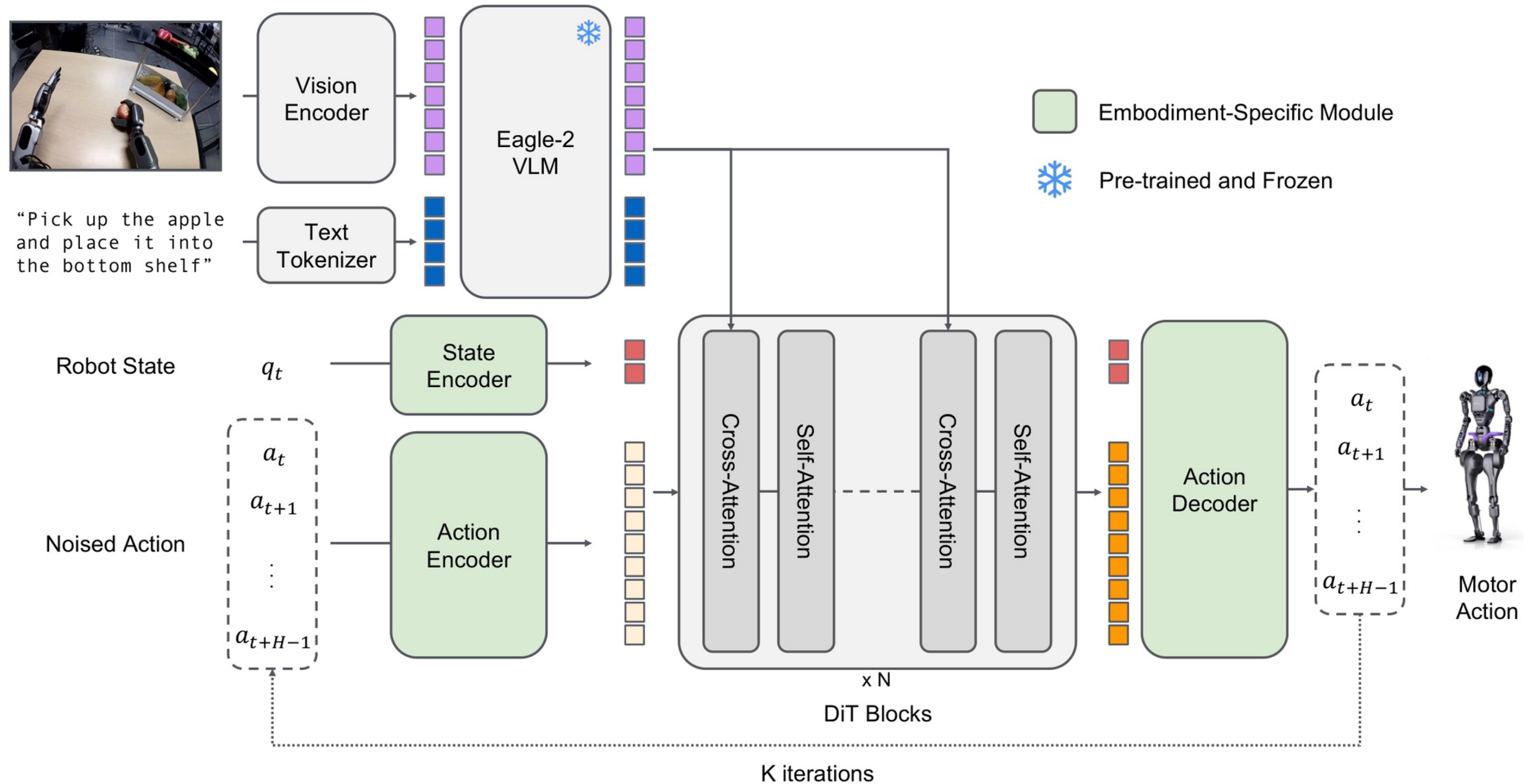
Prompt: use the left hand to pick up spray bottle to basket



Prompt: use the left hand to pick up spray bottle to beige bowl

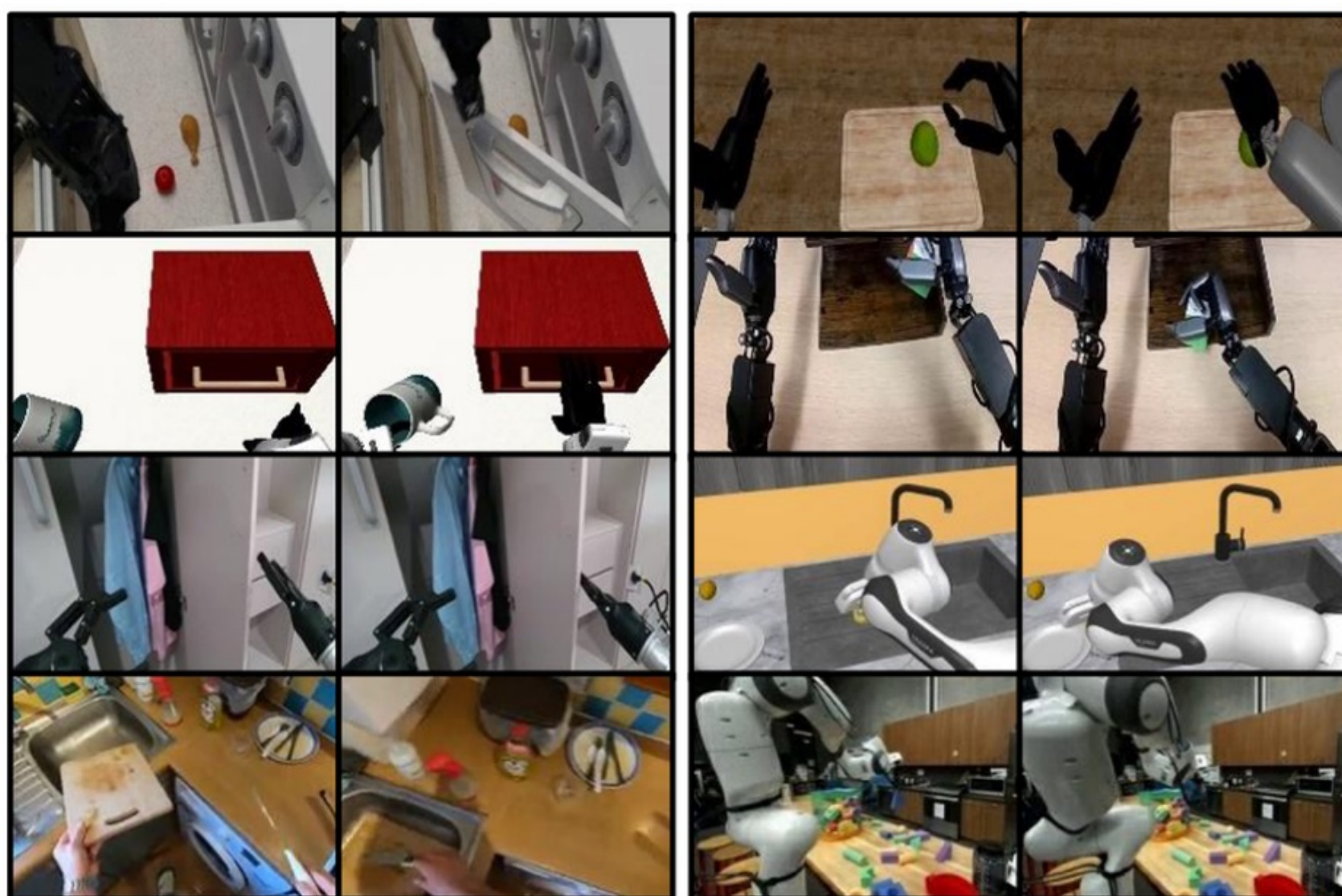


GR00T N1: model architecture

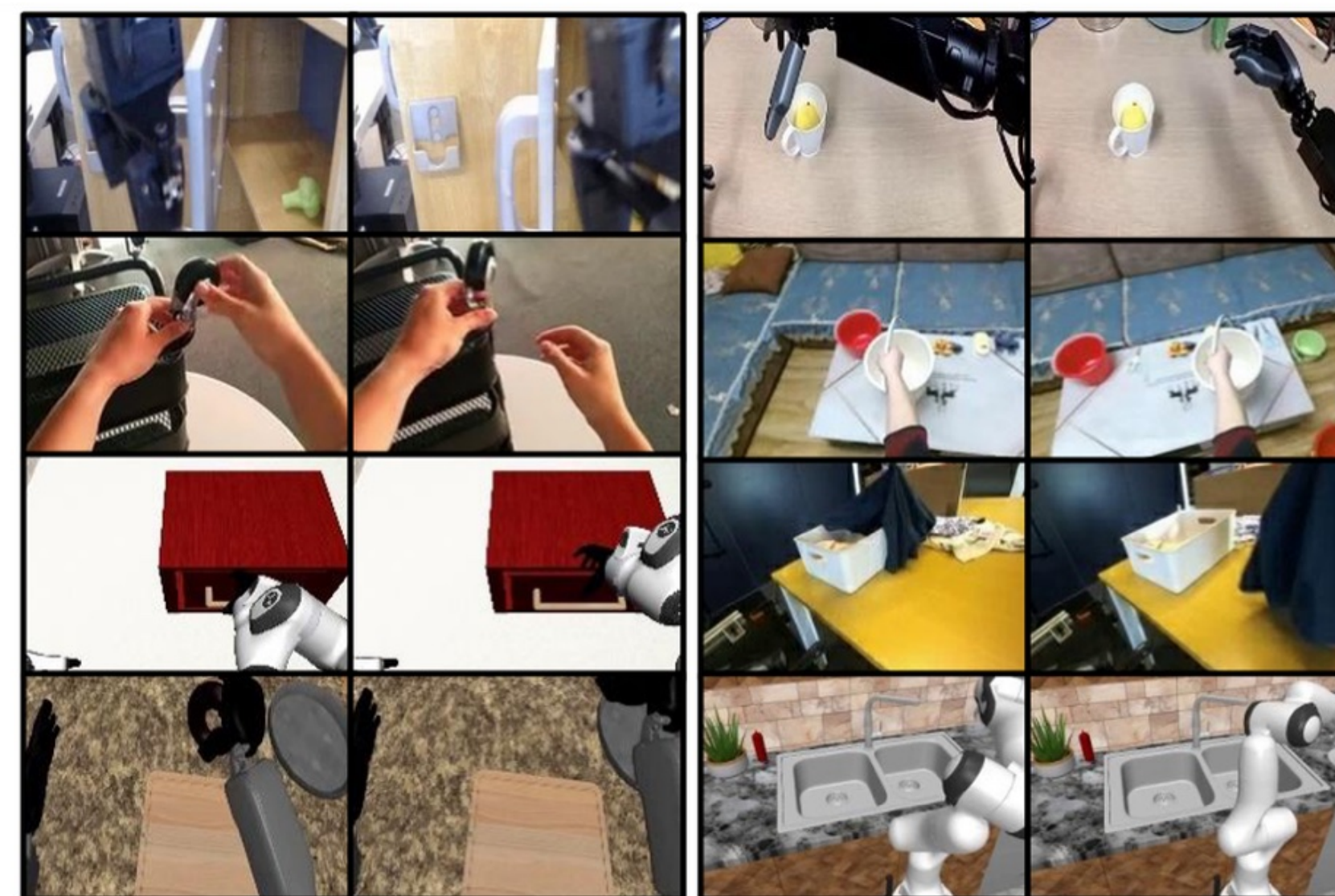


GR00T N1: latent actions

- For egocentric human videos, actions aren't available
- Latent action models represent actions as latent code that can be used to decode future observations
- Latent actions are treated as a separate embodiment in GR00T N1



Latent action 1: Move the right arm to the left



Latent action 2: Move the right arm to the right

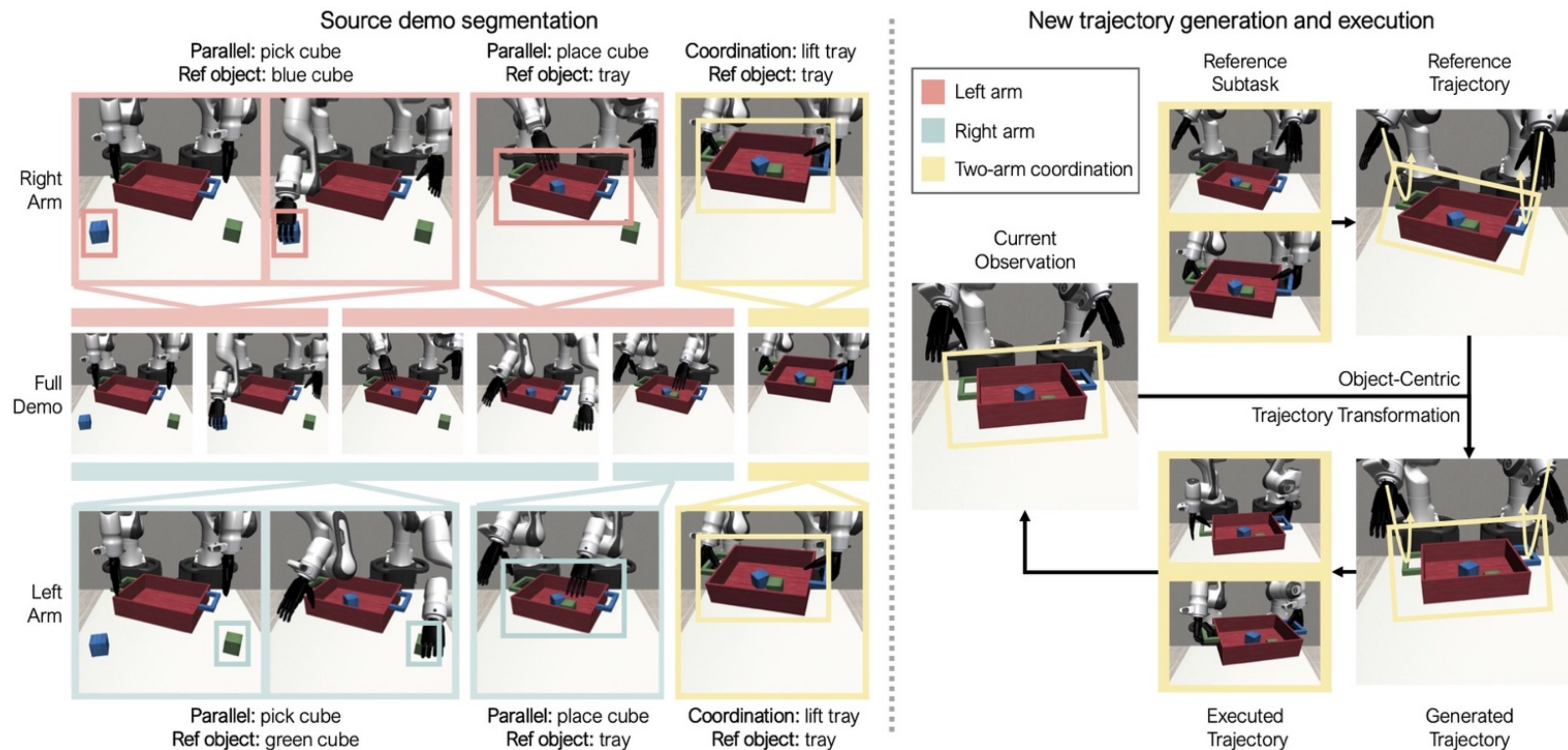
Latent action representations from video

$$z = E(I_t, I_{t+\Delta}), \quad \hat{I}_{t+\Delta} = D(I_t, z)$$

- Given a pair of frames, the latent action encoder predicts a latent code z
- Given the first frame in the pair, and the latent code, the latent action decoder predicts the second image frame in the pair
- A representation *bottleneck* encourages the latent to describe the change between frames
- Without this bottleneck, encoder can (and likely will) encode a compact representation of (i.e., memorize) the second image pair directly into the latent code
- Latent actions can't be executed on robots (in this formulation). But they're still useful in learning relevant priors from action less data

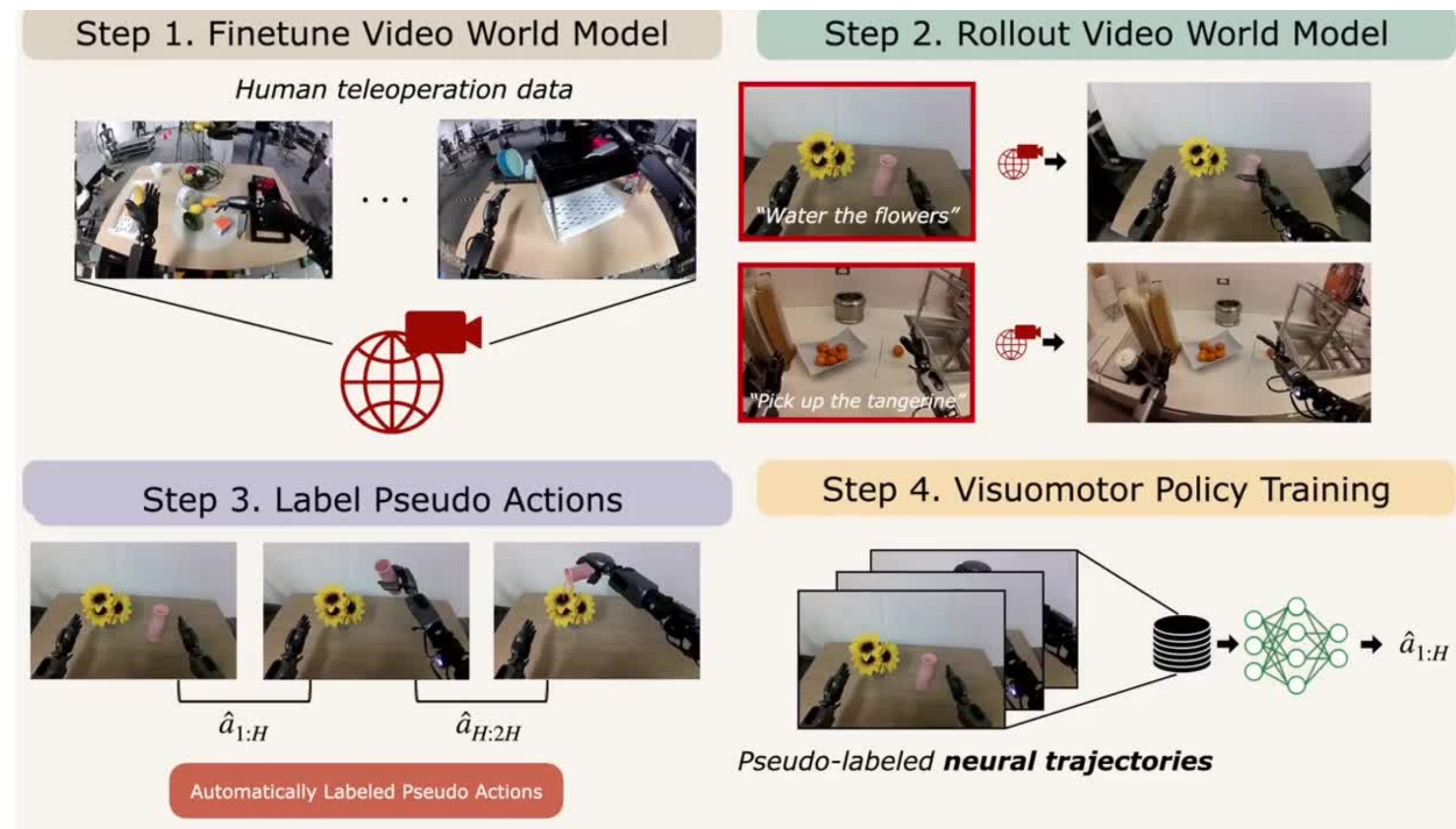
Aside (DexMimicGen): simulation-based trajectory augmentation

- Transform object-relative segments of demonstrations and re-execute them in simulation
- Successful trajectories provide additional observations with known action labels



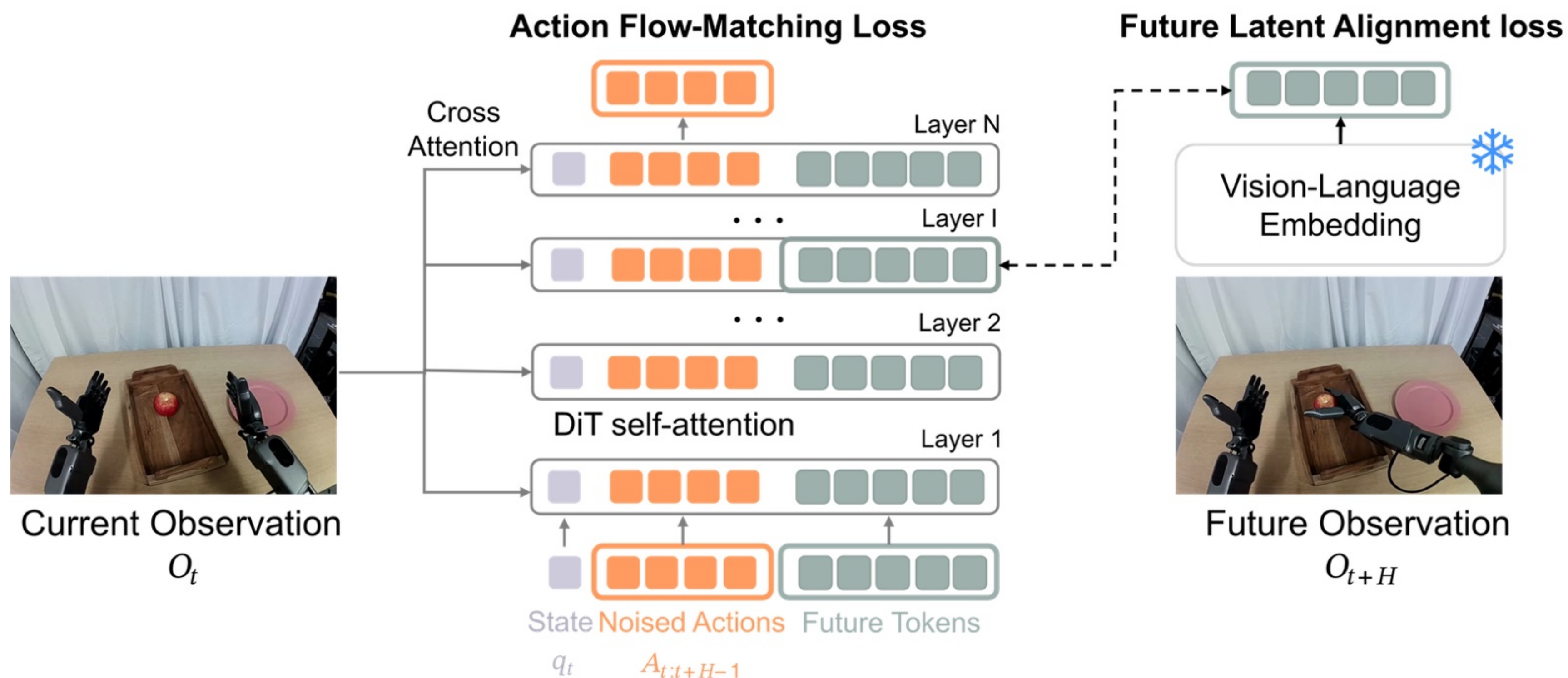
Generated videos and inferred actions

- A video model generates additional visual trajectories for specified behaviors
- An inverse-dynamics model supplies action labels for compatible generated trajectories



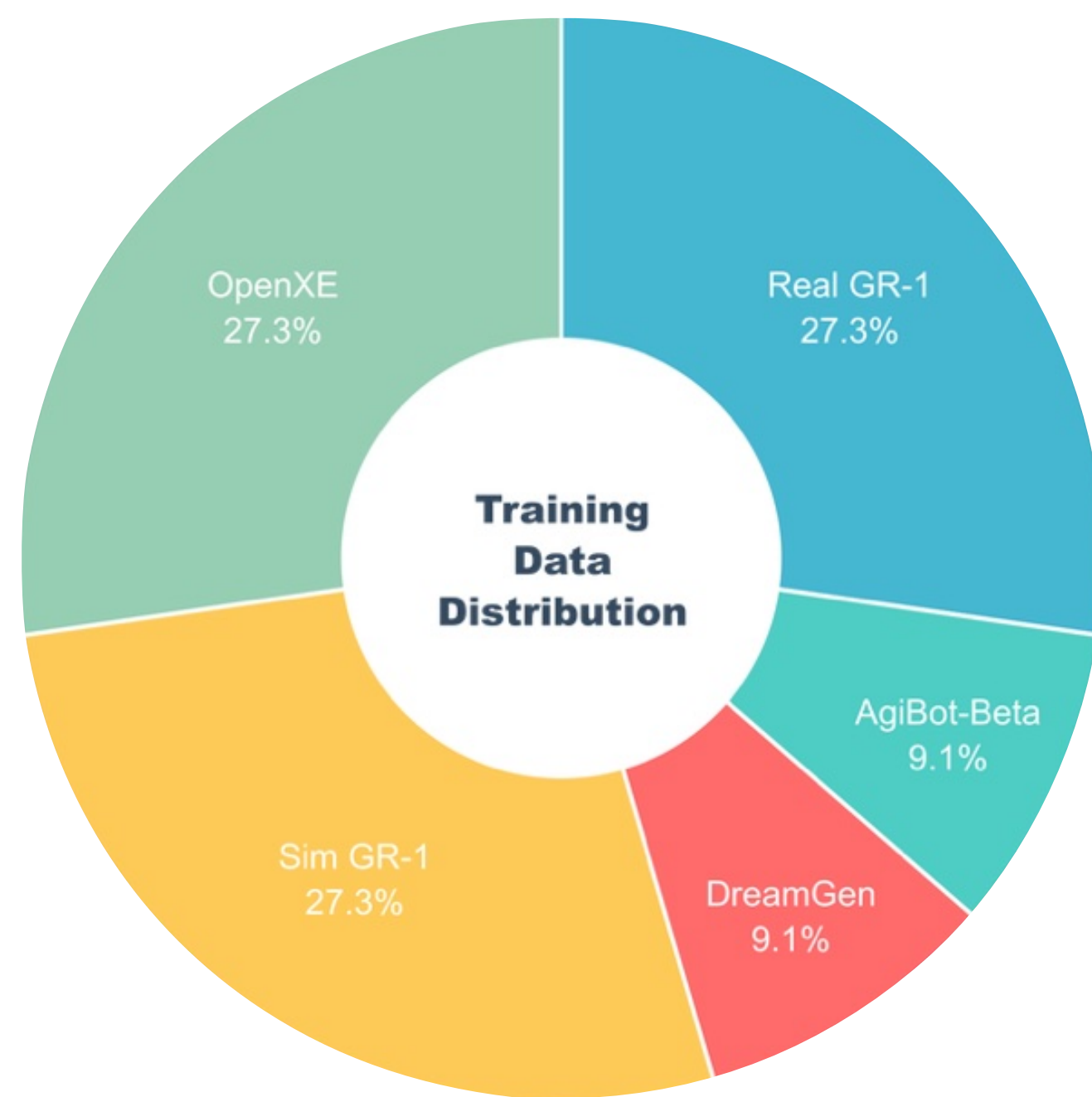
FLARE: predicting future features

- Add a prediction target for latent features of future observations
- Actionless human video contributes to improved performance
- This objective forms the basis for modern world action models (WAMs), which we'll see in a later lecture



GR00T N1.5: data and supervision

- Combine robot trajectories, simulation augmentation, generated examples and human video
- DreamGen adds neural trajectories
- FLARE incorporates a future observation encoding prediction objective



NVIDIA. GR00T N1.5. Technical release. 2025.

Jang et al. DreamGen: Unlocking Generalization in Robot Learning through Video World Models. CoRL. 2025.

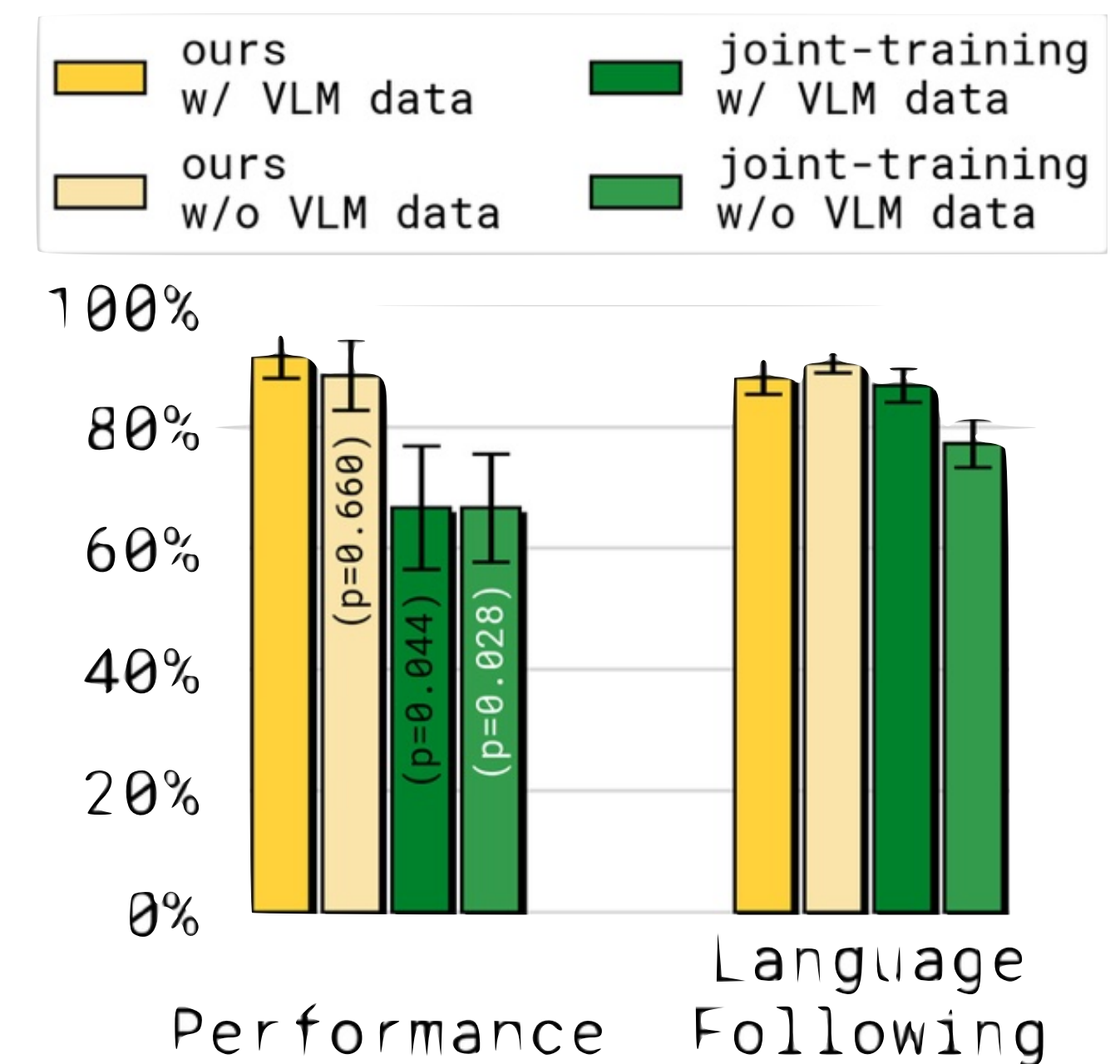
Zheng et al. FLARE: Robot Learning with Implicit World Modeling. CoRL. 2025.

Problem 3: how do we retain web-scale pretrained knowledge?

- Web-scale (VLM) pretraining provides broad visual, semantic, and linguistic priors
- VLA models should ideally leverage all these capabilities to generalize to newer objects, environments, and tasks
- But, naive finetuning / re-training of a VLM to obtain a VLA leads to *catastrophic forgetting* (i.e., underlying VLM modules *forget* much of what was learned before)
- One could keep the VLM frozen (every downstream module conditions on the frozen VLM embeddings)
- However, this (naive) approach doesn't work (representations need to evolve to be predictive of robot actions)
- pi0.5 addresses it with what they term *knowledge insulation*

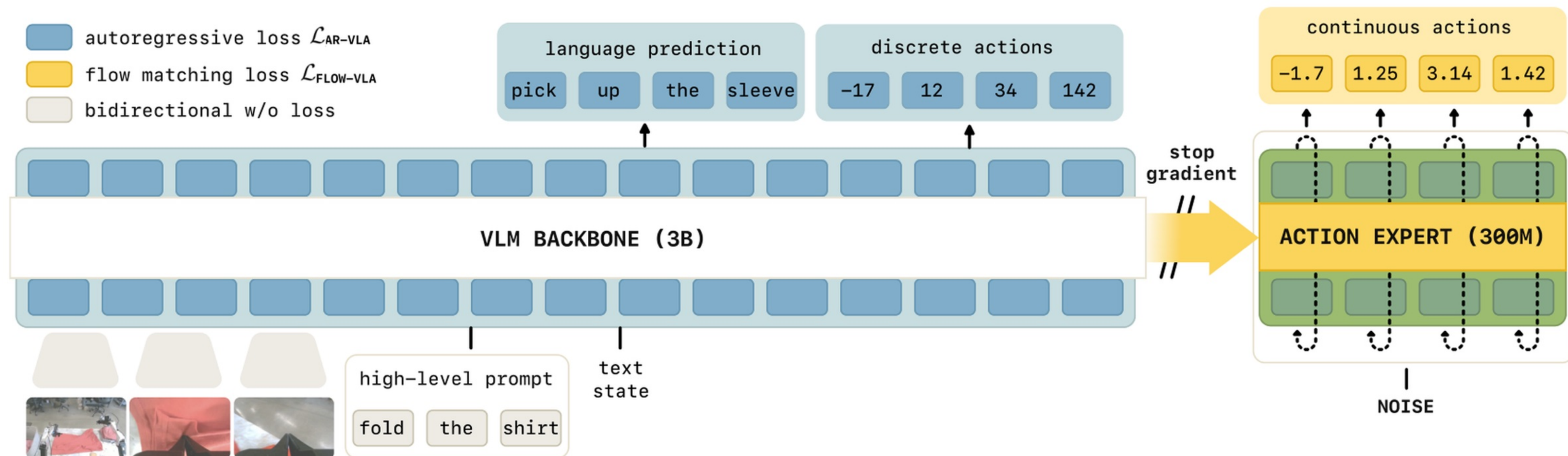
It is a bad idea to *just initialize* an action expert on top of a VLM and train it

- Consider a pristine, frontier VLM that just cost a billion \$\$ \$ to train
- It is tempting to strap a *randomly initialized* action expert (a flow matching head) on top and train it with robot data
- A *very bad* idea!
- The action expert is initially (inevitably) going to produce poor actions predictions, which means gradients through it will be large
- We do not want these flowing back through the VLM and resulting in catastrophic forgetting
- At the same time, we also don't want to just freeze the VLM



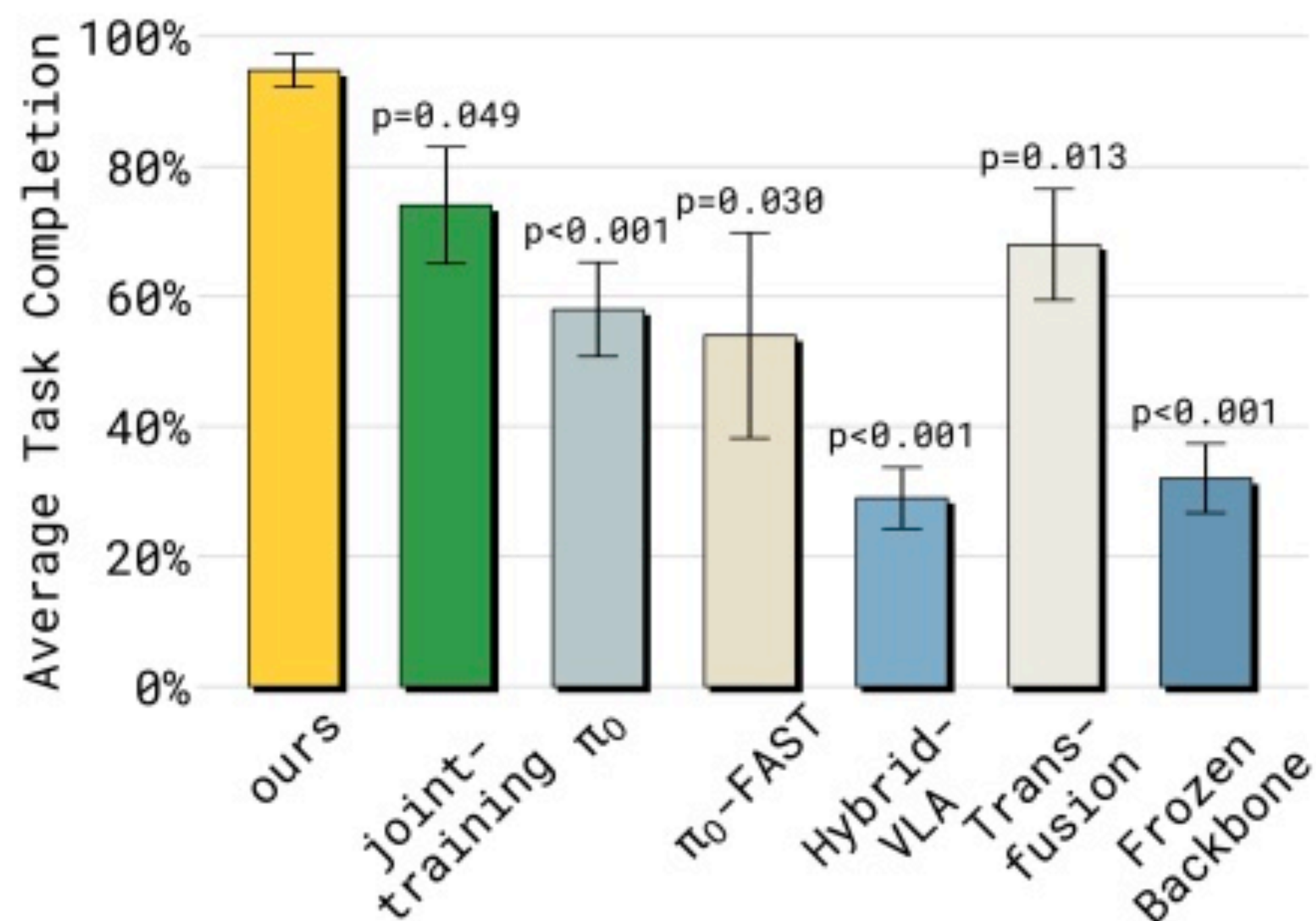
Knowledge insulation

- Key idea 1: Co-train on generally robotics relevant vision-language tasks
- Use those gradients to keep updating the VLM backbone
- Use (continuous) action predictions to only update only the action expert
- Key idea 2: Add discrete action prediction (FAST tokenization) as auxiliary objective to update VLM backbone

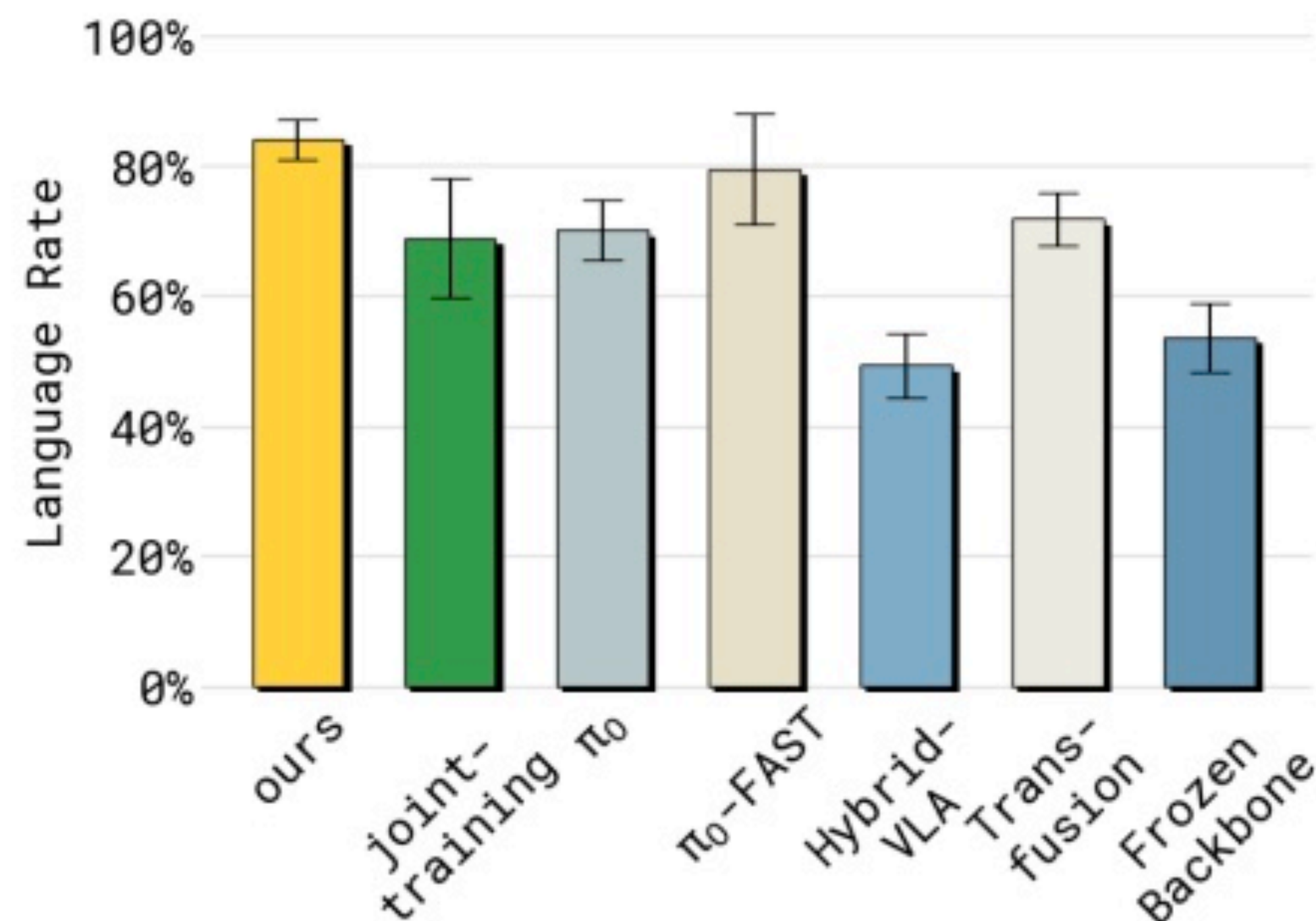


Knowledge insulation: results

- Compare joint-training, frozen backbone, and knowledge insulation
- Metrics: Instruction following, task performance and (not shown) training speed



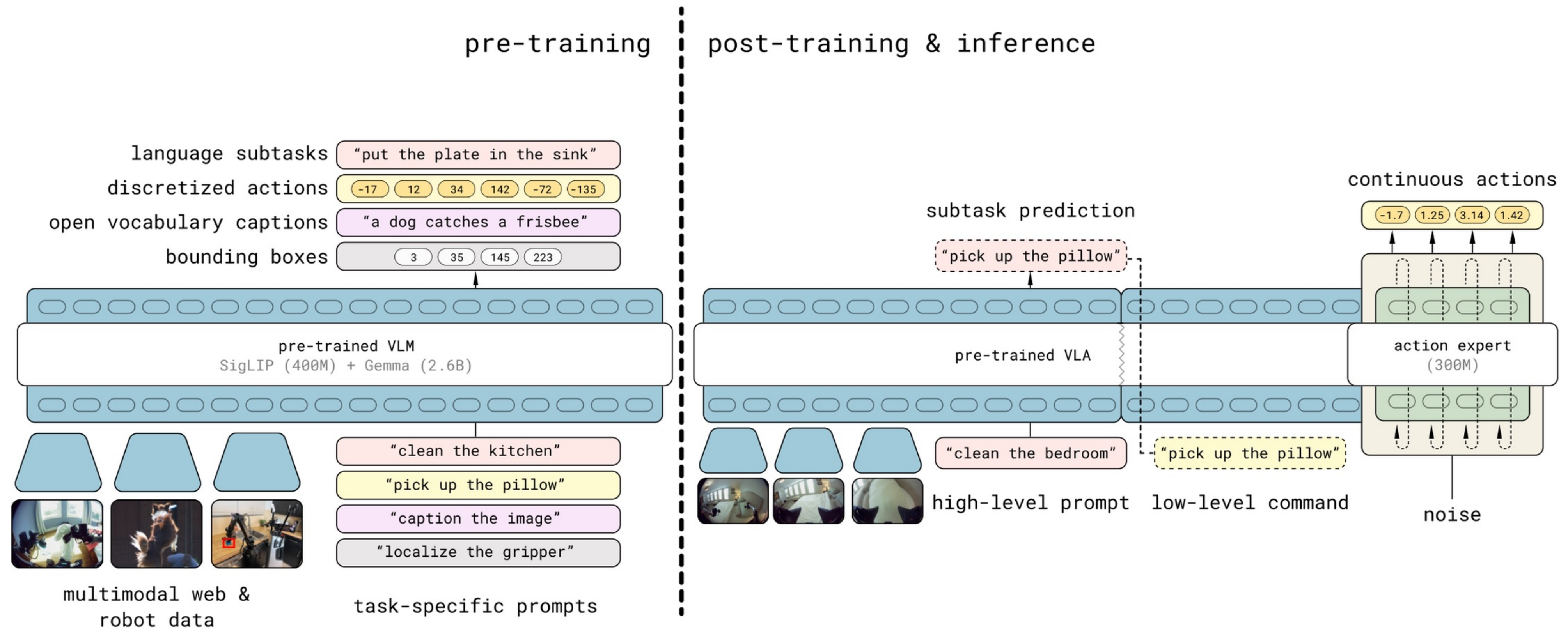
(a) Performance



(b) Language following

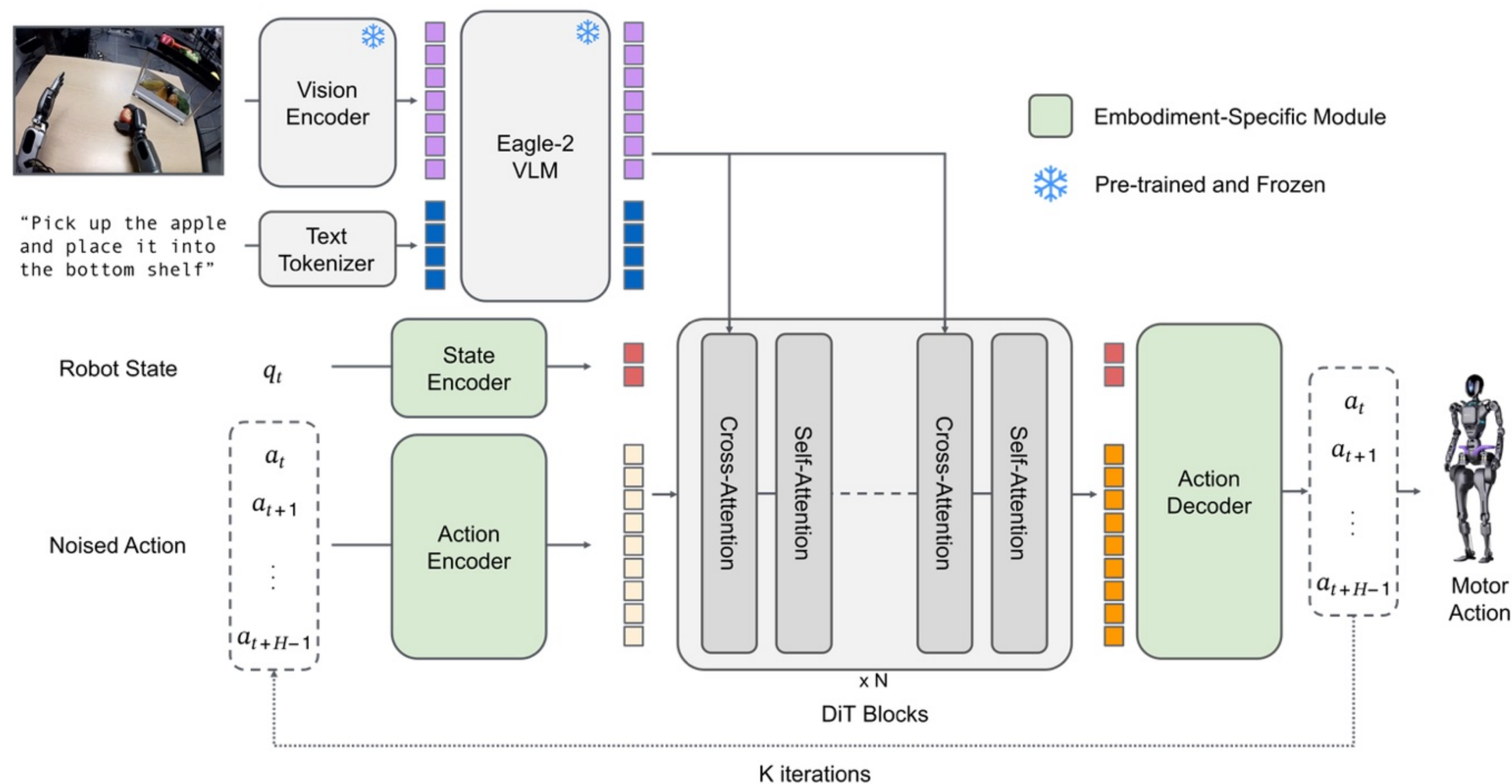
pi0.5: architecture and training

- Predict a subtask, e.g., “put the plate in the sink,” from the overall instruction and scene
- Condition low-level action generation on this subtask and current observations



GR00T N1.5: freezing the VLM

- Freeze the VLM during policy pretraining and fine-tuning
- Blocks action gradients completely, unlike knowledge insulation
- However, the action expert is now trained with a FLARE (future representation prediction) objective



Impact of fine-tuning on previously-learned skills

- Fine-tuning can also change/remove skills acquired during robot pretraining
- E.g., a useful handover behavior in the base policy is lost after fine-tuning

Task: Pick up red apple and place it into the basket

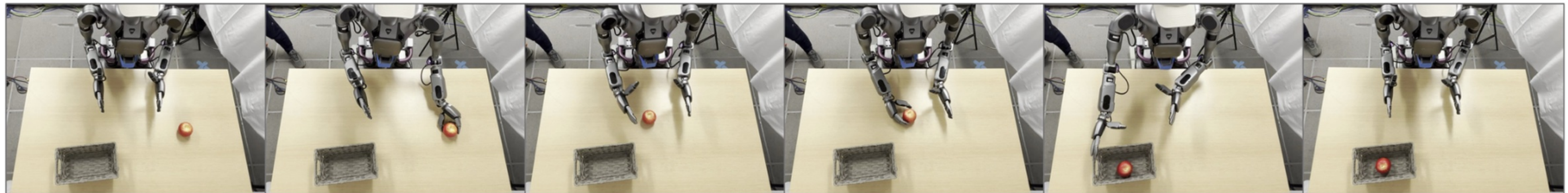
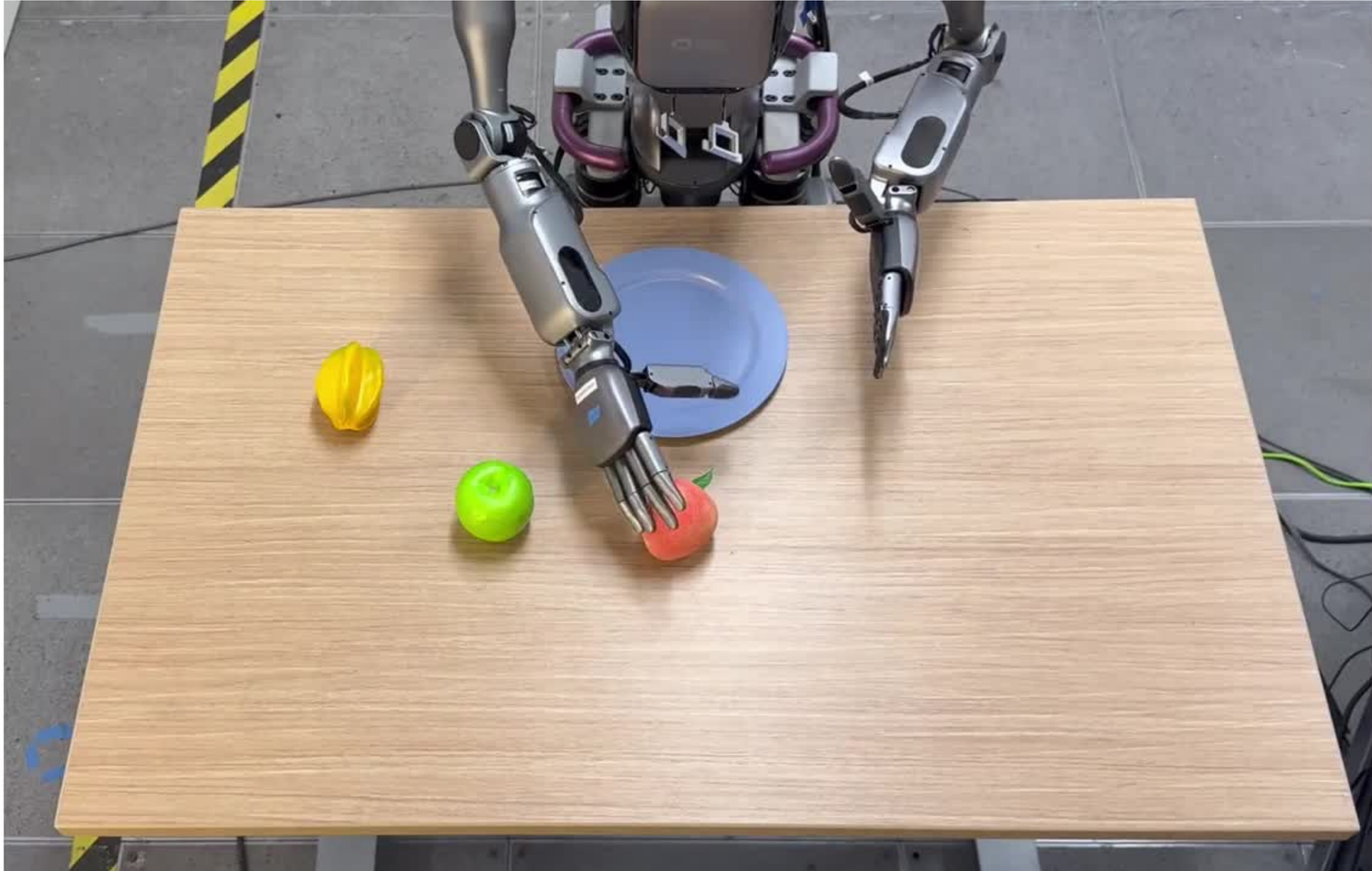


Figure 11: Pre-training Qualitative Example. While prompting the pretrained GR00T-N1-2B model with a post-training task instruction, we even increase the difficulty by placing the apple to the left of both hands. Despite not having encountered this setup during training, the model successfully places the red apple into the basket via a two-handed handover, albeit with jerkier motion.

GR00T N1.5: language-conditioned execution



GR00T N1.5: target-object human video



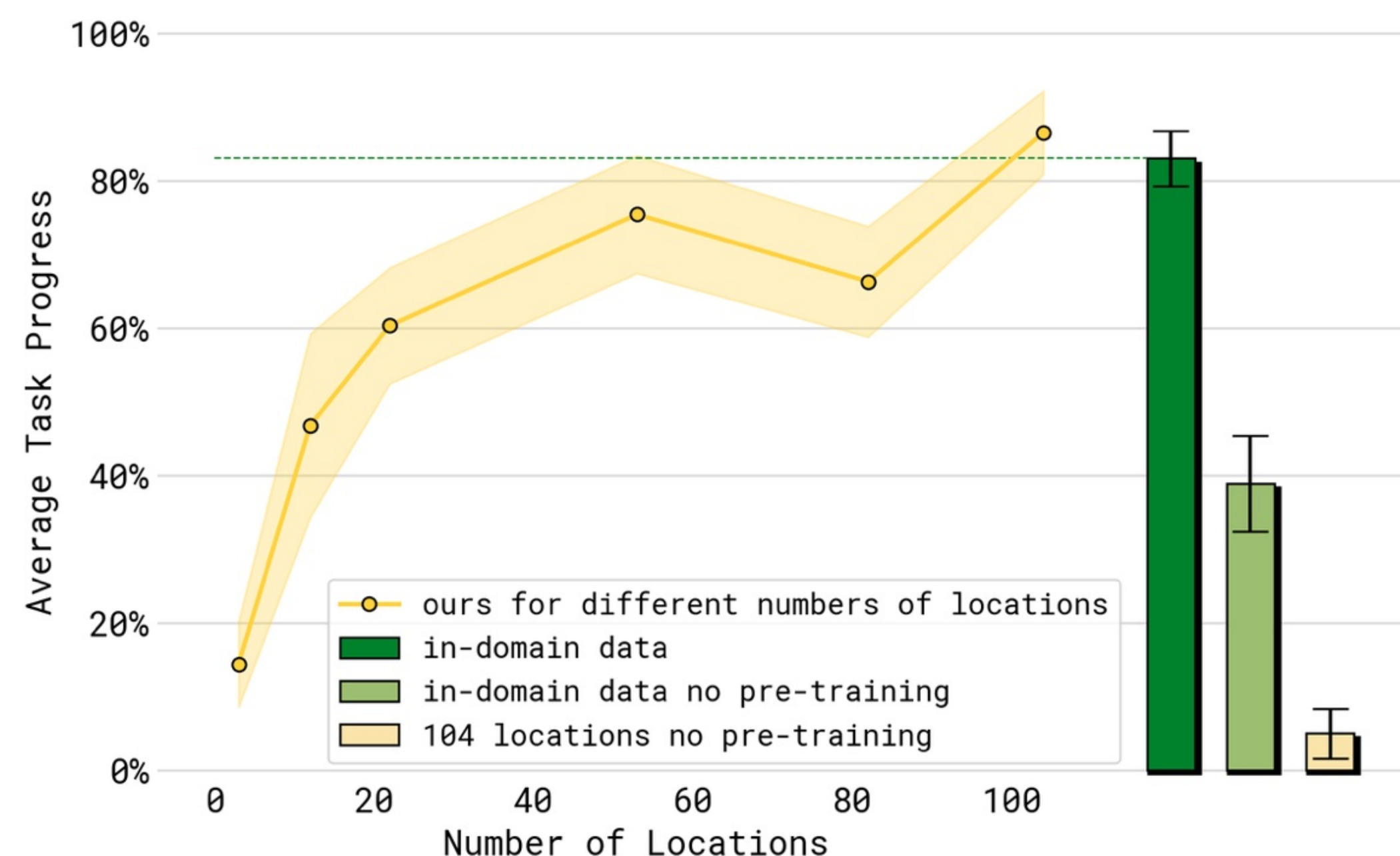
GR00T N1.5: generated behaviors

- DreamGen provides training trajectories for twelve new behaviors
- No teleop data; behaviors solely exist in generated data



pi0.5: deployment in new homes

- Evaluate household tasks in homes absent from the training data
- Training-location diversity improves transfer to held-out environments



pi0.5: execution in a held-out home



Next lecture

- VLAs and multi-task policies continued
- Reminder: No class next week
- Reminder: assignment 1 demo-ed in-class the week after (during the second half of the lecture)