$\pi^*_{0.6}$: a VLA That Learns From Experience

Physical Intelligence

Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, Zhiyuan Zhou

https://pi.website/blog/pistar06

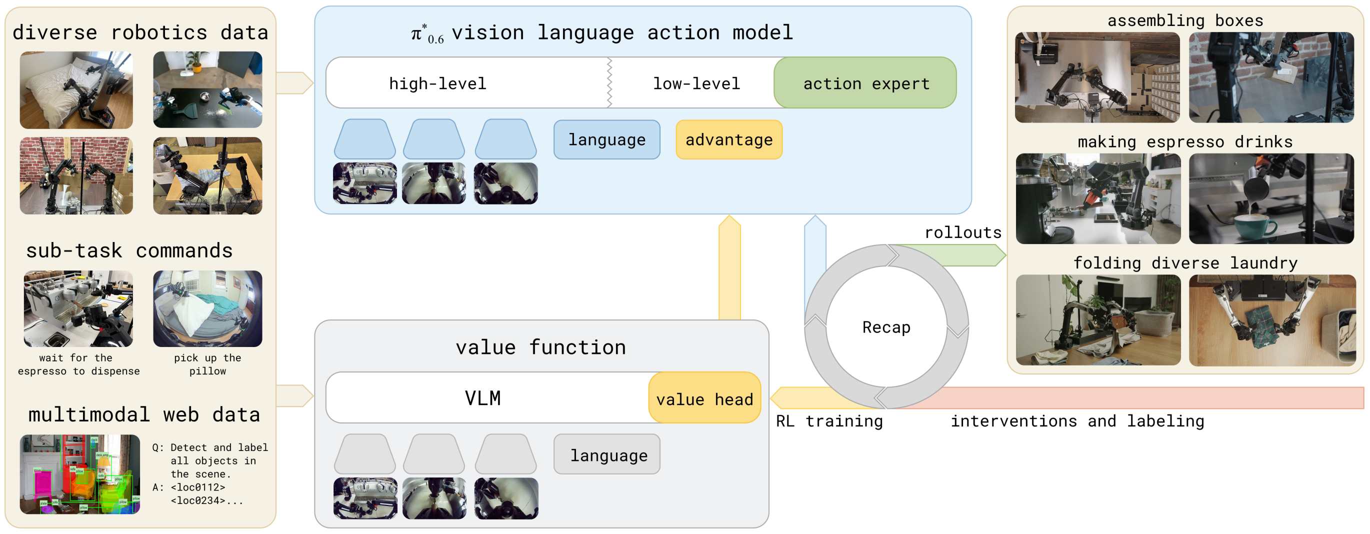

Fig. 1: **RECAP enables training VLAs with reward feedback and interventions.** Our system starts with a pre-trained VLA that incorporates *advantage conditioning*, allowing the model to learn effectively from real-world experience. For each task, we deploy the model and collect both autonomous rollouts and online human corrections. We then fine-tune the value function on this online data, improving its estimates of how actions influence performance. Fine-tuning and conditioning the VLA on these updated advantage estimates in turn improves policy behavior.

***Abstract*—We study how vision-language-action (VLA) models can improve through real-world deployments via reinforcement learning (RL). We present a general-purpose method, RL with Experience and Corrections via Advantage-conditioned Policies (RECAP), that provides for RL training of VLAs via advantage conditioning. Our method incorporates heterogeneous data into the self-improvement process, including demonstrations, data from on-policy collection, and expert teleoperated interventions provided during autonomous execution. RECAP starts by pre-training a generalist VLA with offline RL, which we call $\pi^*_{0.6}$, that can then be specialized to attain high performance on downstream tasks through on-robot data collection. We show that the $\pi^*_{0.6}$ model trained with the full RECAP method can fold laundry in real homes, reliably assemble boxes, and make espresso drinks using a professional espresso machine. On some of the hardest tasks, RECAP more than doubles task throughput and roughly halves the task failure rate.**

I. INTRODUCTION

It's amazing what you can learn if you're not afraid to try.

Robert A. Heinlein, *Have Space Suit–Will Travel*

Practice makes perfect: while people are remarkably flexible in acquiring new skills, mastery invariably requires learning from repeated attempts. With general-purpose robotic foundation models, such as vision-language-action (VLA) models, we can flexibly specify tasks for generalist robots through prompts. But just like people, these models will need to *practice* a skill to achieve mastery. This means leveraging not only on demonstration data, but also autonomously collected experiential data that allows the policy to correct the mistakes that it actually makes in deployment, improve speed and robustness beyond the level of human teleoperation, and adapt to new deployment conditions. The foundations of learning through autonomous practice, as formalized with reinforcement learning (RL) [1], have been known for decades, but instantiating these principles in a general and scalable robotic learning system presents significant challenges: designing scalable and stable RL methods for large models, handling heterogeneous data from different policies, and setting up RL training with reward feedback in the real world, where reward signals might be ambiguous or stochastic.

In this paper, we present RECAP, a method that enables VLA models to incorporate reward feedback in all stages of the training pipeline, from pre-training all the way to training on data from autonomous execution. RECAP aims to address this problem with a general-purpose recipe that combines demonstrations, autonomous experience, and expert interventions. Starting from the training recipe for a general-

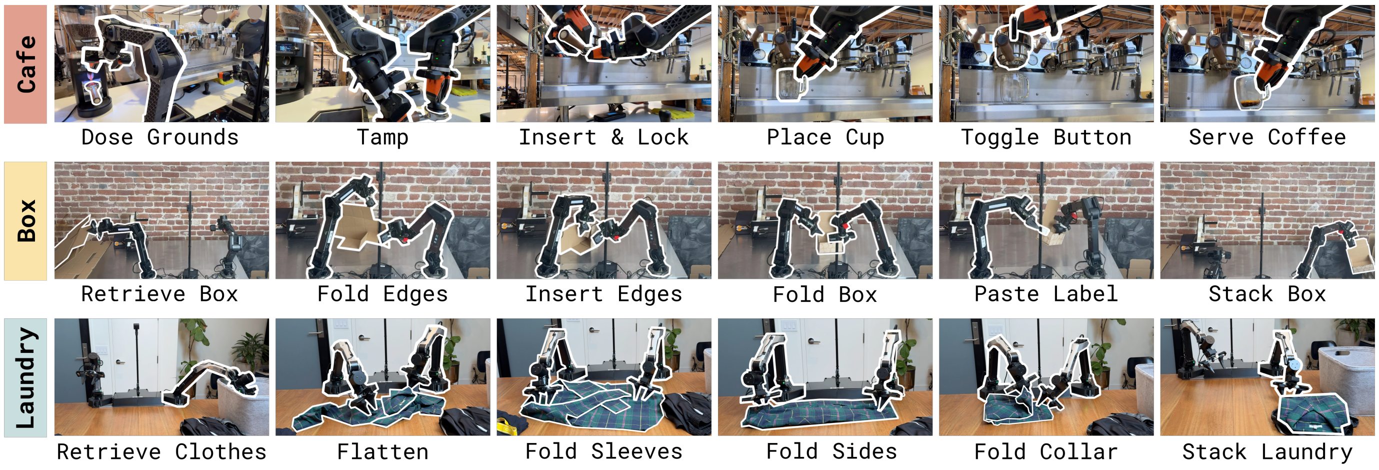

Fig. 2: **Some of the tasks learned by RECAP.** $\pi^*_{0.6}$ trained with RECAP can make espresso drinks, assemble cardboard boxes, and fold diverse and realistic laundry with a high success rate. Each task involves realistic variability – flattened unfolded boxes stick together and bend, making espresso drinks requires pouring liquids, and folding laundry requires generalization to a wide range of clothing items.

purpose VLA and training on diverse data from many different robotic platforms, RECAP first pre-trains the VLA with offline RL, followed by additional training on data collected through deployments. During these deployments, the robot receives (sparse) reward feedback based on the outcome of each trial, and potentially additional expert interventions that correct mistakes. The training process follows an offline RL [2, 3] recipe: we train a value function that evaluates progress toward successful task completion, and then use this value function to estimate the advantage of each action in the dataset. By conditioning the policy on an improvement indicator based on this advantage [4], we can obtain an improved policy. Figure 1 provides a high-level overview of RECAP.

We can use RECAP to train policies for complex tasks, such as folding diverse laundry, assembling boxes, or making espresso drinks. We illustrate some of these tasks in Figure 2. The method starts by pre-training the $\pi^*_{0.6}$ model with offline RL on a diverse multi-task and multi-robot dataset. $\pi^*_{0.6}$ is an adaptation of the $\pi_{0.6}$ model for RL, and $\pi_{0.6}$ is an improvement on $\pi_{0.5}$ [5], adding a larger backbone and more diverse conditioning [6]. $\pi^*_{0.6}$ adds the ability to condition on binarized *advantage* values, which makes it possible to incorporate a value function to improve the policy. After pre-training $\pi^*_{0.6}$ finetunes the $\pi^*_{0.6}$ model to a downstream task with demonstrations, and then performs one or more iterations of on-robot data collection to improve the model with RL. Training $\pi^*_{0.6}$ with RECAP on autonomous experience more than doubles the throughput on some of the hardest tasks, and can decrease failure rates by 2× or more. This enables $\pi^*_{0.6}$ to reach practically useful levels of robustness: we were able to run it to make espresso drinks for 13 hours straight, fold novel laundry items in a new home for over two hours without interruptions, and assemble boxes that are used for real packaging in a factory.

While RECAP is based on individual algorithmic components that have been explored in prior works, the particular combination of these components is novel, and the results show, for the first time, that a general-purpose reinforcement learning recipe with human reward feedback and interventions can significantly improve both the robustness and throughput of VLA models with experience collected through deployment.

II. RELATED WORK

Policies trained with imitation learning are known to suffer from compounding errors [7] and, at best, can only be as performant as the demonstration data. The goal of this work is to improve the reliability and speed of vision-language-action policies by going beyond imitation learning from offline demonstrations. Prior works have used online interventions to improve robotic manipulation policies [8–11]. We adopt a form of such interventions, called human-gated DAgger [10, 12]. In contrast to these works, our method uses both expert interventions and fully autonomous experience, resulting in an RL-based framework that integrates multiple data sources. There is a large body of work on using RL for autonomous improvement of robotic manipulation policies [13–21], including methods using diffusion-based policies [22–24], in multi-task settings [25, 26], and using pre-trained multi-task policies [27–29]. Unlike these works, we study how to scale real-world RL to large VLA policies for long-horizon, fine-grained manipulation tasks.

Many recent works have studied how to improve a base VLA model through RL. Several works directly apply the proximal policy optimization (PPO) algorithm and variations thereof to VLA fine-tuning [30–34], yielding approaches that are difficult to extend to real-world RL in an efficient and scalable fashion. Another line of research has explored RL fine-tuning *on top of* pre-trained VLA models, where RL either trains a residual policy [35, 36], fine-tunes an action head network [37], selects or refines actions proposed by the VLA [38–40], or optimizes a policy acting in the noise space of a diffusion-based VLA [41]. Some of these works have also explored ways to distill the learned behavior back into the VLA for end-to-end iterative improvement [35, 36, 38, 42].

These prior works generally use discrete actions or simple Gaussian continuous action distributions. A critical distinction is that we train an entire VLA end-to-end using (iterated) offline RL, with an expressive flow matching VLA model. This is made possible by a simple and scalable advantage-conditioned policy extraction method, which removes much of the complexity of using policy gradient style objectives with large VLA models. In our comparisons, we show that this significantly outperforms a more traditional policy gradient based extraction scheme.

More closely related to RECAP in terms of methodology, a number of prior works have integrated value functions and end-to-end RL training of VLAs on real robots [43–46]. For example, Huang et al. [43] apply calibrated Q-learning to an offline demonstration dataset for grasping tasks, without an online improvement phase. Zhang et al. [44] use direct preference optimization (DPO) to optimize pick-and-place skills from human preferences, using online rollouts from a VLA. Finally, Zhai et al. [45], Ghasemipour et al. [46] use PPO and REINFORCE respectively with time-to-completion value functions to train VLAs for tasks like moving a bowl, unfolding a mat, and pushing objects on a table. In contrast to these prior works, we describe an iterated offline RL framework for VLAs with multiple advantages. First, our method supports high-capacity diffusion and flow-based VLAs, unlike the discrete-action models studied in prior works. Second, we avoid the need for on-policy PPO or REINFORCE by using an advantage conditioning strategy for policy extraction, which can utilize all prior (off-policy or offline) data. Lastly, our evaluation consists of complex, dexterous, and temporally extended tasks, where our method increases throughput by about 2× while handling deformable objects, liquids, and multi-stage tasks.

Prior works have explored the idea of conditioning the policy on rewards, values, and advantages [47–56], including methods that use classifier-free guidance [4]. We extend this approach to pre-train and fine-tune a large-scale generalist VLA policy [5], incorporating a variety of data sources (including demonstrations, interventions, and autonomous policy roll-outs) to learn real robotic manipulation tasks. Recent research has also studied how to effectively train multi-task, language-conditioned reward functions [57–63] and value functions [45, 64, 65]. Building on these works, we also train a language-conditioned distributional value function, which allows us to estimate state-action advantages for our advantage-conditioned VLA training framework.

III. PRELIMINARIES

Reinforcement learning. We consider the standard RL setting in which an agent, given by a policy $\pi(\mathbf{a}_t|\mathbf{o}_t)$, selects actions $\mathbf{a}_t$ given an observation $\mathbf{o}_t \in \mathcal{O}$. We define a trajectory as $\tau = (\mathbf{o}_0, \mathbf{a}_0, \cdots, \mathbf{o}_T) \in \mathcal{O} \times \mathcal{A} \cdots \mathcal{O}$. A distribution over trajectories $\rho_\pi(\tau)$ is induced by the policy $\pi(\mathbf{a}_t|\mathbf{o}_t)$ and the stochastic dynamics $p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t)$: $\rho_\pi(\tau) = p(\mathbf{o}_0) \prod_{t=0}^{T-1} \pi(\mathbf{a}_t|\mathbf{o}_t) p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t)$.[1] The reward function is given by $r(\mathbf{o}_t, \mathbf{a}_t)$, and we abbreviate it to r_t to shorten notation, where r_T is the terminal reward. We can define the discounted cumulative reward, or return, as $R(\tau) = \sum_{t=0}^{T} r_t$ (we do not use a discount factor, though one could easily be added). The goal of RL is to maximize the cumulative reward (or return), learning a policy that maximizes $\mathcal{J}(\pi) = \mathbb{E}_{\tau \sim \rho_\pi}[R(\tau)] = \mathbb{E}_{\tau \sim \rho_\pi}[\sum_{t=0}^{T} r_t]$. The value function for a policy π is then defined as $V^\pi(\mathbf{o}_t) = \mathbb{E}_{\tau_{t+1:T}}[\sum_{t=t}^{T} r_t]$. We can then calculate an advantage value for an action $\mathbf{a}_t$ as $A^\pi(\mathbf{o}_t, \mathbf{a}_t) = \mathbb{E}_{\rho_\pi(\tau)}[\sum_{t'=t}^{t+N-1} r_{t'} + V^\pi(\mathbf{o}_{t+N})] - V^\pi(\mathbf{o}_t)$, corresponding to an n-step estimate.

Regularized reinforcement learning. Instead of maximizing $\mathcal{J}(\pi)$, it is common to use regularization in RL, optimizing for a policy that maximizes reward while remaining close to some reference policy π_{ref} [66–70]. This is important, for example, when we want to train for many gradient steps on the same data, in which case π_{ref} typically corresponds to the behavior policy that collected the training data. This can be formalized via the objective $\mathcal{J}(\pi, \pi_{\text{ref}}) = \mathbb{E}_{\tau \sim \rho_{\pi_\theta}}[\sum_{t=0}^{T} \gamma^t r_t] - \beta \mathbb{E}_{\mathbf{o} \sim \rho_{\pi_\theta}}[D(\pi(\cdot|\mathbf{o}) \| \pi_{\text{ref}}(\cdot|\mathbf{o}))]$, where D denotes some divergence metric. For the case where D is the KL divergence, we have the well-known result that $\hat{\pi}(\mathbf{a}|\mathbf{o}) \propto \pi_{\text{ref}}(\mathbf{a}|\mathbf{o}) \exp(A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a})/\beta)$ is the solution to $\max_\pi J(\pi, \pi_{\text{ref}})$, with Lagrange multiplier β [67–70]. Our advantage-conditioned policy extraction method is based on a closely related but less well-known result: if we define the policy $\hat{\pi}(\mathbf{a}|\mathbf{o}) \propto \pi_{\text{ref}}(\mathbf{a}|\mathbf{o}) p(I|A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a}))^\beta$, where $p(I|A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a})) = g(A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a})) / \int g(A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a}')) \mathrm{d}\mathbf{a}'$ is the probability of any action $\mathbf{a}$ improving over π_{ref} as measured by a monotonically increasing function g, then $\hat{\pi}$ is guaranteed to improve over π_{ref}, i.e., $\mathcal{J}(\hat{\pi}) \geq \mathcal{J}(\pi_{\text{ref}})$ [4, 71]. We will use this property in deriving our policy extraction method in Section IV-B. Using this definition we can then obtain a parametric policy from the closed form definition of $\hat{\pi}$ by solving the following minimization problem: $\min_\theta \mathbb{E}_{s \sim \rho_{\pi_{\text{ref}}}}[KL(\hat{\pi}, \pi_\theta)]$.

IV. RL WITH EXPERIENCE AND CORRECTIONS VIA ADVANTAGE-CONDITIONED POLICIES (RECAP)

Our method consists of the follow steps, which can be repeated one or more times to improve a base VLA model:

1) **Data collection.** We run the VLA on the task, labeling each episode with task outcome labels (which determine the reward), and optionally providing human interventions to provide examples of corrections for mistakes in the earlier iterations.
2) **Value function training.** We use all of the data collected so far to train a large, multi-task value function, which we refer to as $V^{\pi_{\text{ref}}}$, that can detect failures and judge the expected time to task completion.
3) **Advantage conditioned training.** To improve the VLA policy with this value function, we include an optimality indicator based on advantage values derived from this

[1]For simplicity, we assume the observation $\mathbf{o}_t$ constitutes a valid Markovian state. While not true in general, it is a common simplification in robotic RL.

value function in the VLA prefix. This "advantage conditioned" recipe provides a simple and effective way to extract a more optimal policy from our value function with suboptimal data.

Figure 1 illustrates the overall structure of the training process, while Figure 3 provides more detailed specifics of the value function and policy architectures. Our pre-training phase consists of performing steps (2) and (3) above on our entire pre-training dataset, which consists of tens of thousands of hours of demonstrations from numerous tasks and a variety of different robots. Then, we perform steps (1), (2), and (3) one or more times to further improve the VLA with autonomously collected data. We describe the value function training and policy training steps below, and then present our specific instantiation of this approach for training $\pi^*_{0.6}$ in Section V.

A. Distributional value function training

To train a value function that can act as a reliable critic for any task in our pre-training or post-training stages, we represent $V^{\pi_{\text{ref}}}$ with a multi-task distributional value function $p_\phi(V|\mathbf{o}_t, \ell) \in \Delta_B$ [72], mapping the observations $\mathbf{o}_t$ and language command ℓ to a distribution over B discretized value bins. In our implementation, this value function uses the same architecture as the VLA policy, but with a smaller VLM backbone. Using $R_t(\tau) = \sum_{t'=t}^{T} r_{t'}$ to denote the empirical return of a trajectory τ from time step t until the end, we train $p_\phi(V|\mathbf{o}_t, \ell)$ by first discretizing the empirical return value $R_t(\tau)$ into $B = 201$ bins (using R_t^B to denote the discretized returns), and then minimizing the cross-entropy H over the trajectories in the current dataset $\mathcal{D}$:

$$\min_\phi \mathbb{E}_{\tau \in \mathcal{D}} \left[\sum_{\mathbf{o}_t \in \tau} H(R_t^B(\tau), p_\phi(V|\mathbf{o}_t, \ell)) \right]. \quad (1)$$

This is a Monte Carlo estimator for the value function of the policy represented by the dataset $\mathcal{D}$ (i.e., the behavior policy π_{ref}). We can extract a continuous value function (and thus an advantage) from the learned value distribution using $V^{\pi_{\text{ref}}}(o_t, \ell) = \sum_{b \in [0,B]} p_\phi(V = b|\mathbf{o}_t) v(b)$, where $v(b)$ denotes the value corresponding to bin b. During the pre-training phase, the dataset $\mathcal{D}$ corresponds to the human demonstrations, and the value function captures the expected return for the task and metadata we condition on, while on subsequent iterations, it skews toward a weighted combination of the return of the demonstrations and the learned policy.

While this on-policy estimator is less optimal than a more classic off-policy Q-function estimator, we found it to be simple and highly reliable, while still allowing for substantial improvement over imitation learning. Our method could be extended to accommodate off-policy estimators in future work.

B. Policy extraction via advantage conditioning

Once we have the value function $V^{\pi_{\text{ref}}}$, we need a way to train an improved policy using this value function. This is called *policy extraction*. An effective policy extraction method in our setting needs to satisfy several criteria. First,

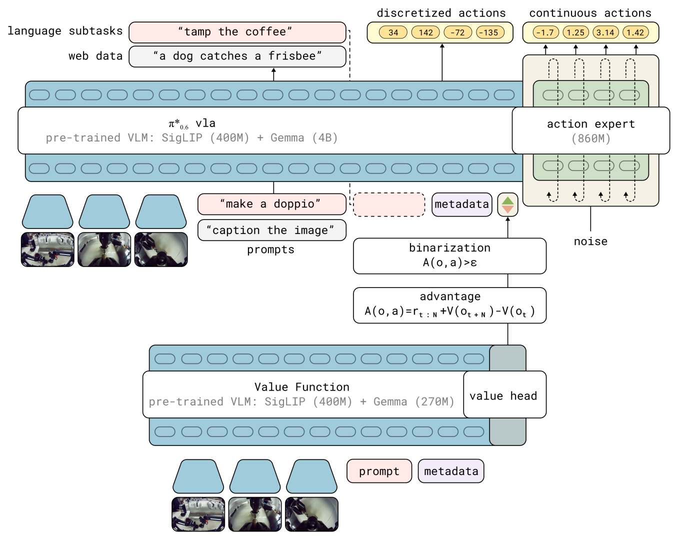

Fig. 3: **Interaction between the $\pi^*_{0.6}$ VLA and value function during RECAP training.** The $\pi^*_{0.6}$ VLA uses a pre-trained VLM backbone. Training follows the KI recipe [73], with next-token prediction on many data sources in pre-training, and an flow-matching action-expert with stop gradient. The VLA is conditioned on a binarized advantage indicator, obtained from a separate value function initialized from a pre-trained but smaller VLM model.

it needs to effectively utilize diverse off-policy data, comprising the initial demonstrations, the expert interventions, and autonomous episodes from both the latest policy and older policies. This is closely related to the challenge faced by offline RL methods [2, 3]. Second, it needs to be scalable and easily to apply to large VLA models, including models that use flow matching or diffusion to generate actions. Third, it needs to effectively utilize both good (near-optimal) and bad (suboptimal) data, which is important if we want to improve the policy using autonomous experience.

Among the existing methods for policy extraction, policy gradient methods (including regularized policy gradients and reparameterized gradients) are perhaps the most widely used [66, 74], but these methods are difficult to apply to flow matching models, which do not readily provide a tractable log-likelihood, making them hard to scale up to modern VLA architectures (see comparisons in Section VI). An alternative is to use weighted regression methods, such as AWR [68, 75, 76], which implicitly provide for regularization to the behavior policy and use a simple (importance-weighted) supervised learning objective. However, these methods discard or significantly downweight a significant portion of the data, effectively implementing a kind of filtered imitation technique. Instead, we use a variant of *advantage conditioning* [48], where the policy is trained on all of the data with supervised learning, but with an additional input indicating how *optimal* the action is based on the advantage. This is closely related to a variety of methods in the literature that propose to condition the policy on some function of the resulting trajectory [47, 50].

The specific formulation in our method is most closely related to CFGRL [4]. Building on the formulation in Section III, we can apply Bayes rule to rewrite the probability of policy

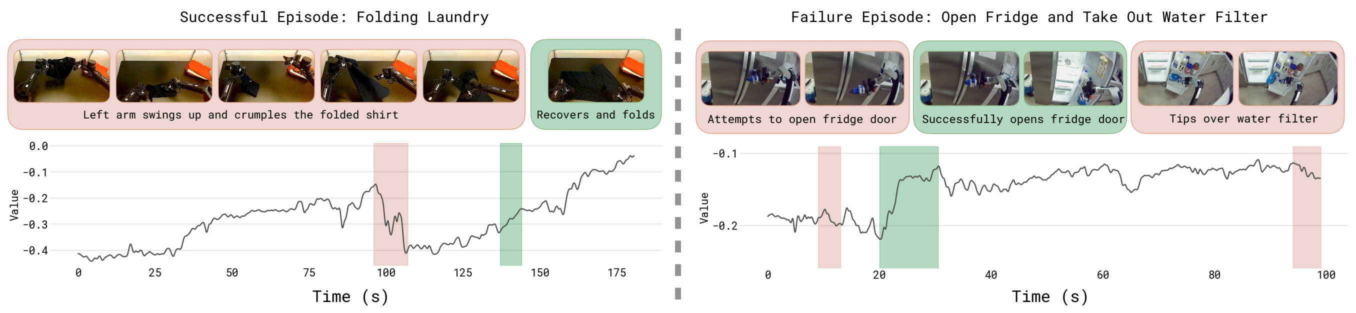

Fig. 4: **Visualization of the value functions.** We train a multi-task value function to predict the number of steps to success, normalized by maximum task length to $(-1, 0)$, where 0 corresponds to successful completion. We visualize the value function output on a folding task that finished successfully (left), and an unsuccessful example of a manipulation task from the pre-training dataset (right). The red parts highlight a drop in value, and green parts highlight increases; images on top show the corresponding frames of the episode. The visualization shows that the VF correctly identifies mistakes in the episode, as well as the speed of progress.

improvement as $p(I|A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a})) = \pi_{\text{ref}}(\mathbf{a}|I, \mathbf{o})/\pi_{\text{ref}}(\mathbf{a}|\mathbf{o})$. Applying this to our setting and including language conditioning, we can obtain an alternative closed form for the improved regularized policy described in Section III as

$$\hat{\pi}(\mathbf{a}, |\mathbf{o}, \ell) \propto \pi_{\text{ref}}(\mathbf{a}|\mathbf{o}, \ell) \left(\frac{\pi_{\text{ref}}(\mathbf{a}|I, \mathbf{o}, \ell)}{\pi_{\text{ref}}(\mathbf{a}|\mathbf{o}, \ell)}\right)^{\beta}. \tag{2}$$

For the special case $\beta = 1$, $\hat{\pi}(\mathbf{a}, |\mathbf{o}, \ell) = \pi_{\text{ref}}(\mathbf{a}|I, \mathbf{o}, \ell)$.

We can therefore represent $\hat{\pi}$ without needing to explicitly represent the improvement probability $p(I|A^{\pi_{\text{ref}}}(\mathbf{o}, \mathbf{a}))$, if we train the policy so that it can represent both $\pi_{\text{ref}}(\mathbf{a}|\mathbf{o}, \ell)$ and $\pi_{\text{ref}}(\mathbf{a}|I, \mathbf{o}, \ell)$. This principle is similar to the approach in classifier-free guidance, where a diffusion model is trained to model the data both with and without a conditioning variable [4]. We assume the improvement indicator I follows a delta distribution

$$p(I|A^{\pi_{\text{ref}}}(o, a, \ell) = \delta(A^{\pi_{\text{ref}}}(o, a, \ell) > \epsilon_{\ell}),$$

with a task dependent improvement threshold ϵ_{ℓ}. This threshold allows us to control the optimality indicator, and minimizes the need for finding an attenuation factor β to sharpen the improvement conditioned distribution after training.[2] The policy objective then corresponds to minimizing the following negative log-likelihood:

$$\min_{\theta} \mathbb{E}_{\mathcal{D}_{\pi_{\text{ref}}}} \Big[- \log \pi_{\theta}(\mathbf{a}_t|\mathbf{o}_t, \ell) - \alpha \log \pi_{\theta}(\mathbf{a}_t|I_t, \mathbf{o}_t, \ell) \Big], \quad \text{where } I_t = \mathbb{1}\left(A^{\pi_{\text{ref}}}(\mathbf{o}_t, \mathbf{a}_t, \ell) > \epsilon_{\ell}\right). \tag{3}$$

The advantage values $A^{\pi_{\text{ref}}}(\mathbf{o}_t, \mathbf{a}_t, \ell)$ are obtained from the value function in the previous section, and α is a trade-off hyperparameter. In practice, the dataset $\mathcal{D}_{\pi_{\text{ref}}}$ consists of all of the data collected so far, including all demonstrations and autonomous task attempts, and the reference policy π_{ref} is therefore a mixture of human behavior and previously deployed policies. To include human corrections, we found it useful to force I_t = True (i.e., positive) for actions provided as human corrections during autonomous rollouts. This choice is reasonable if we assume that human experts always provide good corrective actions. As we will discuss in Section V, in practice our VLA model produces both discrete and continuous outputs, with the continuous distribution represented via flow matching. Therefore, the real training objective combines likelihoods for the discrete values with the flow matching objective for the continuous values.

In practice, we pre-train one model to represent $\pi_{\theta}(\mathbf{a}_t|I_t, \mathbf{o}_t, \ell)$ on our entire pre-training dataset, and then perform one or more iterations of our method with on-policy rollouts (and, optionally, expert corrective interventions) for each task.

C. *Method summary*

We provide an overview of our full method in Algorithm 1. As summarized at the beginning of this section, the method can be fully defined through application of three subroutines: collecting data through autonomous rollouts (with optional corrective interventions from an expert), training a value function according to Equation 1, and training a policy according to Equation 3. The only thing that changes between different steps of the method is the data provided to each subroutine: the pre-training stage uses all prior demonstration data, and the training process for the specialists for each skill $\ell^{(i)}$ uses additional autonomous data. In practice, the specialists are fine-tuned from the pre-trained model, while the final generalist is trained from scratch. Additional details on the method are provided in Appendix F.

V. Implementation, Model, and System Details

We instantiate Recap with a VLA that we call $\pi^{*}_{0.6}$. $\pi^{*}_{0.6}$ is based on the $\pi_{0.6}$ VLA, which is an evolution of the $\pi_{0.5}$ VLA [5] with a few improvements that we detail in the accompanying model card [6]. $\pi^{*}_{0.6}$ additionally adds the ability condition on the binarized advantage indicator I_t, making it suitable for RL training with Recap. The model architecture is illustrated in Figure 3. We train a value function alongside

[2]Prior work [4] instead uniformly chose $\epsilon = 0$ and tuned β at test time, as in classifier-free guidance (CFG). However, high CFG weights can drive the action distribution to the corners of its support (leading to aggressive behavior) and would not affect the autoregressive part of the model. We found it easier to obtain good results by instead using the threshold ϵ_{ℓ} to trade off regularization and optimality.

Algorithm 1 RL with Experience and Corrections via Advantage-conditioned Policies (RECAP)

```
Require: multi-task demonstration dataset $\mathcal{D}_{\text{demo}}$
 1: Train $V_{\text{pre}}$ on $\mathcal{D}_{\text{demo}}$ using Eq. 1
 2: Train $\pi_{\text{pre}}$ on $\mathcal{D}_{\text{demo}}$ using Eq. 3 and $V_{\text{pre}}$
 3: Initialize $\mathcal{D}_\ell$ with demonstrations for $\ell$
 4: Train $V_\ell^0$ from $V_{\text{pre}}$ on $\mathcal{D}_\ell$ using Eq. 1
 5: Train $\pi_\ell^0$ from $\pi_{\text{pre}}$ on $\mathcal{D}_\ell$ using Eq. 3 and $V_\ell^0$
 6: for $k = 1$ to $K$ do
 7:     Collect data with $\pi_\ell^{k-1}$, add it to $\mathcal{D}_\ell$
 8:     Train $V_\ell^k$ from $V_{\text{pre}}$ on $\mathcal{D}_\ell$ using Eq. 1
 9:     Train $\pi_\ell^k$ from $\pi_{\text{pre}}$ on $\mathcal{D}_\ell$ using Eq. 3 and $V_\ell^k$
10: end for
```

the VLA, following the method described in Section IV-A. This value function is also initialized from a VLM. Training this value function and VLA with RECAP results in our final model, which we call $\pi^*_{0.6}$. In this section, we first elaborate on the design of our model and how it can be extended to use advantage values from the value function, then describe the reward function and value function, and then elaborate on the training and data collection process in our implementation.

A. The $\pi_{0.6}$ model

The $\pi_{0.6}$ model [6] is derived from the $\pi_{0.5}$ model, which can flexibly represent chunked action distributions via flow matching and produce intermediate text for high-level policy reasoning. It uses the Knowledge Insulation (KI) training procedure [73], which trains the entire model end-to-end on continuous actions and discretized tokens (including actions discretized via FAST [77]), while using a stop gradient to prevent the flow-matching action expert from impacting the rest of the model. Pre-training uses both robot data and vision-language co-training data from the web.

$\pi_{0.6}$ improves on $\pi_{0.5}$ in several ways: (i) The pre-training dataset is augmented with additional data from multiple robot platforms. (ii) The base VLM is Gemma 3 [78] 4B model. (iii) The size of the action expert is increased to 860M parameters.

The model can be written as $\pi_\theta(\mathbf{a}_{t:t+H}, \hat{\ell}|\mathbf{o}_t, \ell)$, where $\mathbf{o}_t = [\mathbf{X}_t^1, ..., \mathbf{X}_t^n, \mathbf{q}_t]$ contains camera images $\mathbf{X}$, the robot's configuration $\mathbf{q}$, and $\ell = \ell_t + s$ is the language input consisting of the overall task prompt ℓ_t (e.g., "make me an espresso"), as well as additional language inputs s providing metadata that further modulates how the task is performed. The model produces action chunks $\mathbf{a}_{t:t+H}$, which consists of joint angles and gripper commands at 50 Hz, using a separate "action expert" — a dedicated set of weights (860M parameters) that are trained with flow matching specifically for action generation, but can attend to the activations in the rest of the model. The model also produces tokenized discrete outputs $\hat{\ell}$, which includes a textual representation of the next predicted sub-task (such as "pick up the coffee cup") used for high-level decision-making. Since the actions are generated after $\hat{\ell}$, action generation is effectively conditioned on this predicted sub-task, providing high-level guidance. At inference time, the sub-task prediction runs at a lower frequency than action generation. During training, the model also predicts a tokenized representation of the action chunk $\mathbf{a}_{t:t+H}$, using the FAST tokenizer [77], as part of the KI recipe [73]. We denote these discretized actions $a^\ell_{t:t+H}$. The action expert does not receive these as input, such that discrete and continuous actions are predicted independently. This results in the final training log-likelihood $\log \pi_\theta(\mathbf{a}_{t:t+H}, a^\ell_{t:t+H}, \hat{\ell}|\mathbf{o}_t, \ell)$. Since we predict $\hat{\ell}$ first, we can factorize this log-likelihood according to:

$$\log \pi_\theta\big(\mathbf{a}_{t:t+H}, a^\ell_{t:t+H}, \hat{\ell}|\mathbf{o}_t, \ell\big) = \log \pi_\theta\big(\hat{\ell}|\mathbf{o}_t, \ell\big) + \log \pi_\theta\big(a^\ell_{t:t+H}|\mathbf{o}_t, \ell, \hat{\ell}\big) + \log \pi_\theta\big(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell, \hat{\ell}\big).$$

B. From $\pi_{0.6}$ to $\pi^*_{0.6}$ with advantage conditioning

To incorporate information about the advantage into the policy, we expand the model inputs to contain an additional improvement indicator as an additional text input, inputting "Advantage: positive" when I_t = True, and "Advantage: negative" otherwise. The VLA model is otherwise the same as described in Section V-A. The advantage indicator appears in the training sequence after $\hat{\ell}$ but before the (discretized and continuous) actions, such that only the action log-likelihoods are affected. The continuous part of the log-likelihood cannot be evaluated exactly, and instead is trained via the flow matching loss [79]. It is possible to draw a close parallel between flow matching and diffusion (under some assumptions), and the latter in turn can be interpreted as a lower bound on the log-likelihood [80], so we can roughly motivate the sum of the log-likelihood of the discrete actions and the flow matching loss on the continuous actions as a lower bound on the overall action likelihood:

$$\begin{aligned} &\log \pi_\theta(\mathbf{a}_{t:t+H}, a^\ell_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell}) \geq \\ &\mathbb{E}_{\eta,\omega}\Big[\log p_\theta(a^\ell_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell}) - \\ &\quad \alpha_\eta \left\|\omega - \mathbf{a}_{t:t+H} - f_\theta(\mathbf{a}^{\eta,\omega}_{t:t+H}, I_t, \mathbf{o}_t, \ell, \hat{\ell})\right\|^2\Big] \end{aligned}, \quad (4)$$

with $\mathbf{a}^{\eta,\omega}_{t:t+H} = \eta \mathbf{a}_{t:t+H} + (1-\eta)\omega$, $\omega \sim \mathcal{N}(0, \mathbf{I})$ denoting the noised action, where $\eta \in [0, 1]$ is the flow matching time index and f_θ denotes the continuous outputs of the diffusion expert. α_η is a loss weighting term (which can optionally be noise dependent). Full details for the loss are provided in Appendix C.

During training, we randomly omit the indicator I_t instead of tuning the loss multiplier α to allow us to either directly sample from the policy with I_t = True (which corresponds to setting $\beta = 1$ in Equation (2)), or to use both a conditional and unconditional model to implement classifier-free guidance (CFG), which enables inference with $\beta > 1$. See Appendix E for details.

C. Reward definition and value function training

Since our aim is to develop a general and broadly applicable method for training VLAs from experience, we use a general sparse reward definition that can be applied to essentially any task. For each episode, we obtain a label indicating whether that episode was successful. We derive the reward from this episode-level success label such that the value function corresponds to the (negative) number of steps until successful completion of the episode. This is equivalent to the following reward function, where T corresponds to the last step in the episode, and C_{fail} is a large constant that is chosen so as to ensure that failed episodes have low values:

$$r_t = \begin{cases} 0 & \text{if t = T and success} \\ -C_{\text{fail}} & \text{if t = T and failure} \\ -1 & \text{otherwise.} \end{cases} \quad (5)$$

With this reward function, we train the value function to predict the (negative of the) number of remaining steps until success for successful episodes, and a large negative value for failed episodes. In practice, we normalize the values predicted to be between $(-1, 0)$. Since we train on diverse tasks that have very different typical lengths, we normalize the values per task based on the maximum episode length of the task.

The value function takes as input the same language inputs as the $\pi^*_{0.6}$ VLA, and uses the same architecture design, with a smaller 670M parameter VLM backbone that is also initialized from Gemma 3 (see Figure 3). To prevent overfitting, we also co-train the value function on a small mixture of multi-modal web data. Figure 4 show visualizations of the value function on some examples of successful and failure episodes, with additional visualizations in Figure 13 in Appendix B.

D. Pre-training, data collection, and learning from experience

The data mixture used in the pre-training phase of our model largely follows the recipe used by $\pi_{0.5}$ [5], with vision-language data from the web, prediction of subtasks $\hat{\ell}$, and prediction of low-level actions on a variety of tasks from many different robots. We note that, after pre-training, $\pi^*_{0.6}$ can perform many more tasks than the ones used in evaluation in Section VI. During pre-training, we first train the value function on the same dataset, predicting (the negative of) the number of steps to successful completion of each task. Then we estimate the per-task improvement threshold, ϵ_ℓ, used in determining the advantage-based improvement indicator I_t. We set ϵ_ℓ to the 30% percentile of values predicted by the value function for the task ℓ. We then run the value function on-the-fly during VLA training to estimate $A^{\pi_{\text{ref}}}(\mathbf{o}_t, \mathbf{a}_t, \ell)$ for each example, and then use it to compute I_t based on ϵ_ℓ. I_t is included as an input to $\pi^*_{0.6}$ as described in Section V-A. As we use a relatively small VLM backbone (670M) for the value function, on-the-fly inference of the value function incurs minimal additional cost during VLA training.

After pre-training we start a policy improvement loop for the target task. We first finetune $\pi^*_{0.6}$ with demonstration data $\mathcal{D}_\ell$ for the target task ℓ. We fix the indicator I_t to True in this stage, which we found to lead to slightly better results, such that this stage corresponds to supervised finetuning (SFT). This results in the initial policy π^0_ℓ, which is then used to collect additional data that is added to $\mathcal{D}_\ell$. While some of the episodes are collected fully autonomously, some are monitored by an expert teleoperator who can intervene to provide corrections. These corrections can show the policy how to avoid catastrophic failures or how to recover from mistakes. Note, however, that the corrections alone are unlikely to fix all issues: intervening during autonomous execution is a disruptive event, and even expert human operators cannot guarantee a consistent quality of interventions nor improve subtle aspects of the behavior, such as overall speed. Thus, the corrections serve more to fix large mistakes and overcome challenges with exploration, and do not by themselves provide for optimal supervision, in contrast to theory [7]. Recall from Section IV-B that we force I_t = True for all corrections, but otherwise the entire episode (both the autonomous parts and the corrections) are optionally added to the dataset $\mathcal{D}_\ell$ regardless of whether or not a correction was provided.

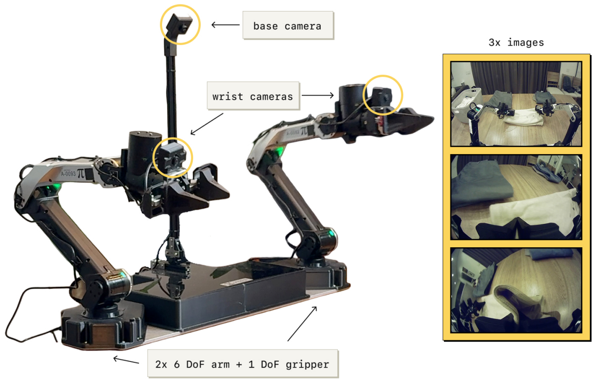

Fig. 5: **The robot setup used in our experiments.** $\pi^*_{0.6}$ is trained on data from many different robots in pre-training. For the iterative improvement experiments, we use a static bimanual system with two 6 DoF arms with parallel jaw grippers. The arms are controlled at 50 Hz with joint positions. Observations consist of joint and gripper positions, as well as images from three cameras: a base camera mounted between the arms, and a wrist-mounted camera on each arm. The setup can be mounted flexibly, e.g. on a table.

After data collection, we finetune the value function on all of the data collected for the task so far, and then use it to finetune the policy with updated indicators I_t, using the same procedure as in pre-training. Both the value function and policy are finetuned from the pre-trained checkpoint, rather than the policy and value function from the last iteration. We found this to be useful for avoiding drift over multiple iterations, though it may be possible to also obtain good results by consistently finetuning from the last model.

We can repeat this process for several iterations as needed, though in practice we found that even one iteration often leads to significantly improved results.

VI. Experimental Evaluation

In our experimental evaluation, we use Recap to train the $\pi_{0.6}$ model on a set of realistic tasks: making espresso drinks, folding diverse laundry, and assembling boxes. Each task requires multiple steps, ranging from 5 to 15 minutes in duration, complex manipulation behaviors (constrained forceful manipulation, pouring liquids, manipulating cloth and cardboard, etc.), and fast execution to provide for high throughput. We illustrate the robotic platform used in our experiments in Figure 5. We

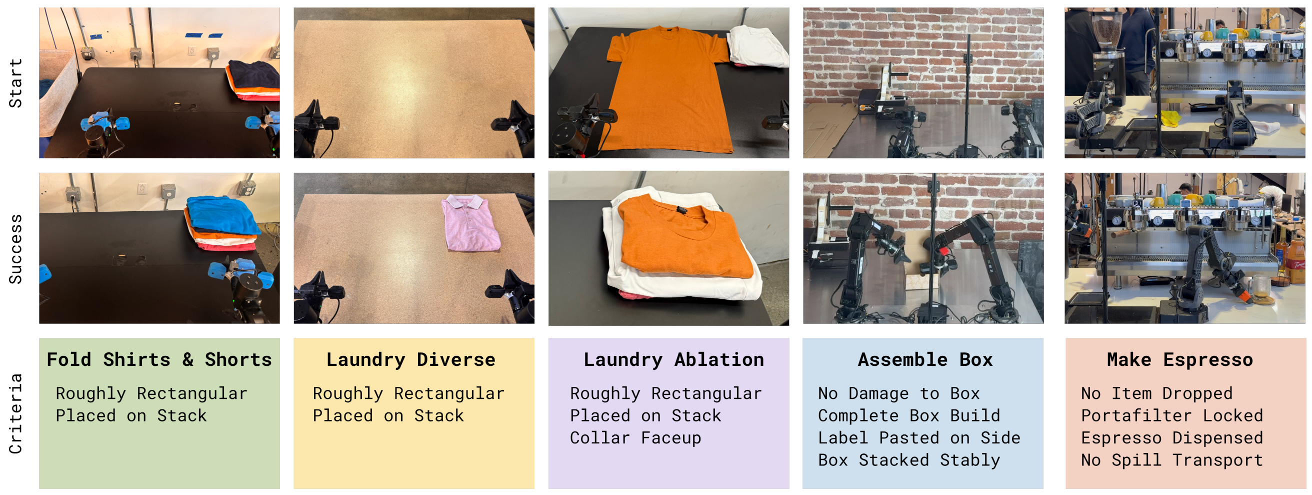

Fig. 6: **Illustrations of the tasks used in our experiments.** Tasks include three different laundry variants, assembling boxes, and making coffee drinks with an espresso machine.

give details on the tasks and baselines below, followed by quantitative experiments.

A. *Evaluation Tasks*

Our quantitative evaluations and comparisons use three broad task categories each with individual task variants: laundry folding, coffee making, and box assembly. We summarize the tasks below, with illustrations in Figure 6:

Laundry (t-shirts and shorts). This is the standard laundry folding task in the π_0 paper [81]. This task entails retrieving either a T-shirt or shorts from a basket with variable initial conditions, flattening, folding. Success requires one clothing item to be folded and stacked in the top right corner of the table within 200 seconds.

Laundry (diverse items). The diverse laundry task requires folding a much larger variety of items, considering 11 item types, including towels, button-up shirts, sweaters, jeans, T-shirts, shorts, polos, skirts, long sleeve shirts, socks, and underwear. To obtain a low-variance metric in our experiments, we measure performance on one of the most challenging items – the button-up shirt. However, the policy is trained on all items, and the accompanying videos show results for a variety of clothing. Success is defined as having the target item correctly folded and placed on a stack on the table within 500 seconds.

Laundry (targeted failure removal). The final version of the laundry folding task considers a much more structured setup for use in our ablation experiments, in which the task involves folding a single orange T-shirt from a fixed flattened initial condition. We place the highest emphasis on success, with a strict success criteria that requires the shirt to be folded correctly with the collar always facing up within 200 seconds. We found this task to be useful for assessing whether RECAP can remove specific undesirable behaviors via RL (in this case, placing the collar facing down rather than up).

Cafe (double shot espresso). We evaluate our policies on the challenging long-horizon task of making coffee with a commercial espresso machine. While our cafe policy can make many drinks (lattes, iced Americanos, espresso, etc), and even clean the espresso machine with a towel, for the purposes of our quantitative experiments we focus on the double espresso shot task. This entails picking up the portafilter, placing it on the grinder and grinding beans into it, tamping the ground coffee beans, locking the portafilter into the espresso machine, bringing over the cup, extracting the full shot of espresso, then serving. Success is measured as completing all steps within 200 seconds without critical mistakes (such as dropping the portafilter or spilling the coffee).

Box assembly. We evaluate our policy on the problem of assembling packaging boxes in a real-world factory deployment scenario. Box assembly involves folding a cardboard box starting from a flattened cardboard sheet, attaching a label onto it and placing the box in the appropriate spot in a crate. For the purposes of the quantitative experiments, we focus on all portions of the task and count overall success as going from a flattened to an assembled and stacked box in under 600 seconds.

B. *Comparisons and Ablations*

We compare RECAP to several baselines:

Pre-trained $\pi_{0.5}$ [5]. This baseline does not use RL and does not leverage RECAP.

Pre-trained $\pi_{0.6}$ [6]. It does not include the advantage indicator I_t, and is pre-trained with supervised learning.

RL pre-trained $\pi^*_{0.6}$. It is pre-trained with RL alongside its value function, and includes an advantage indicator I_t as described in Section V-D.

$\pi^*_{0.6}$ **offline RL + SFT**. This model is trained by finetuning the base $\pi^*_{0.6}$ pre-trained checkpoint with demonstration data for the target task. We refer to this finetuning as "SFT" because the advantage values are fixed to True for all demonstrations. We find that this combination of the offline RL pre-trained $\pi^*_{0.6}$ model with high-quality SFT outperforms standard SFT

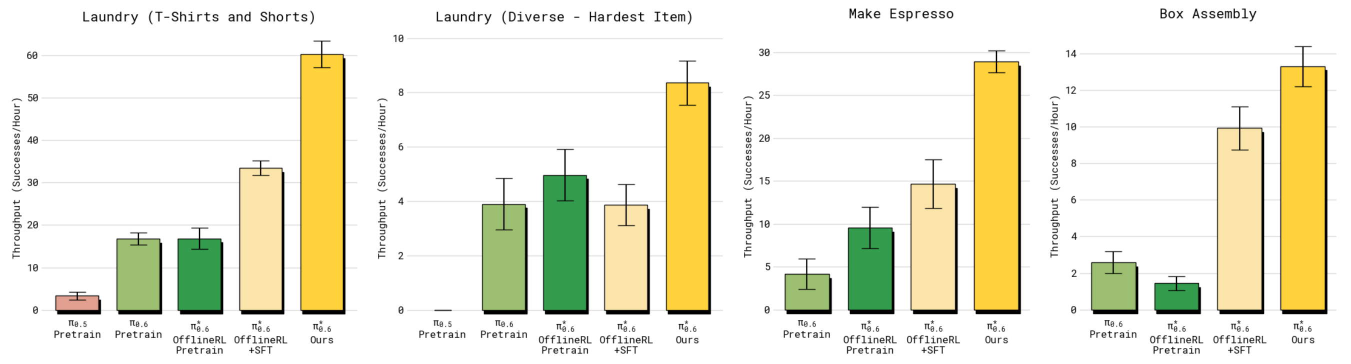

Fig. 7: **Throughput.** We show the number of successfully completed tasks *per hour* for laundry (simple and diverse), espresso making, and box assembly. Error bars show standard error. This metric measures both success and speed. In all cases, RECAP applied to $\pi^*_{0.6}$ (Ours) leads to substantial improvements in throughput. RECAP has the highest impact on throughput for diverse laundry and espresso tasks, more than doubling successful completions per hour.

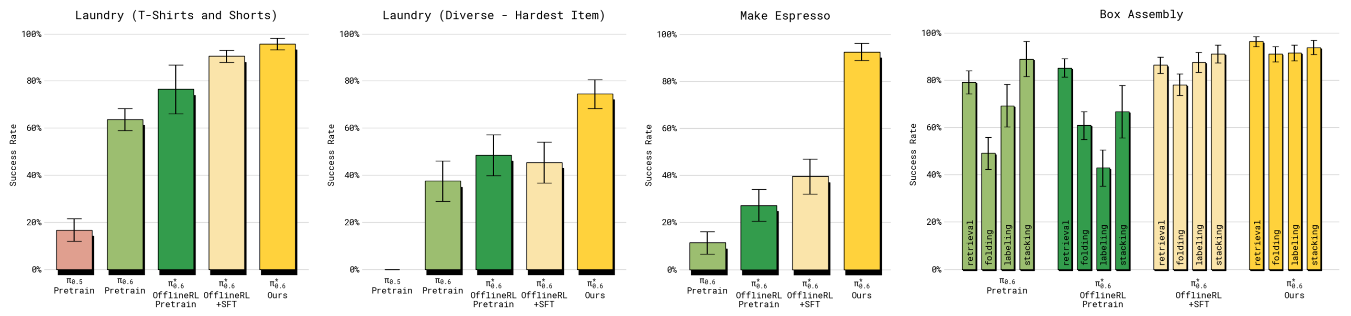

Fig. 8: **Success rates.** We show the absolute success rates with standard error. Each stage of RECAP improves performance across the tasks, with the challenging diverse laundry and espresso tasks seeing the largest gains success rate, corresponding to more than 2× reduction in failure rates. For the box assembly task we show the success rate for the different subtasks. RECAP leads to the most consistent (and highest) success across all subtasks.

(without offline RL pre-training), and provides a good starting point for RL with on-robot data.

$\pi^*_{0.6}$ **(ours)**. This is the final model trained with RECAP on the target task, including both autonomous rollouts and expert corrections. By default we evaluate with $\beta = 1$. In some experiments we also consider inference with CFG, which corresponds to $\beta > 1$.

We also consider two alternative policy extraction methods in the literature as comparisons for our advantage-conditioned approach, both of which use the same on-robot data as RECAP but a different policy learning method:

AWR. Starting from the same pre-trained model $\pi_{0.6}$ (without advantage conditioning) we fine-tune using advantage weighted regression [68], based on advantages extracted from our value-function.

PPO. We implement a variant of DPPO/FPO [23, 82] in which we calculate likelihoods based on the single step diffusion objective and use an alternative definition of the PPO constraint following SPO [83] (see Appendix D for details).

C. Quantitative results

We use two metrics in our evaluation: throughput and success rate. Throughput measures the number of successful task executions per hour, thus capturing both speed and success rate into one practically relevant quantity. Success rate measures the proportion of episodes that succeed, and is derived from human-provided annotations. Raters are asked to judge the episode with respect to multiple quality metrics, and we aggregate these quality indicators into a success label.

1) How much does RECAP *improve the policy?:* To answer this question, we present the main quantitative results in Figures 7 and 8. Across all tasks, the final $\pi^*_{0.6}$ significantly improves over the base (supervised) $\pi_{0.6}$ model, the RL pre-trained $\pi^*_{0.6}$ model, and the **offline RL + SFT** $\pi^*_{0.6}$ model. Throughput more than doubles on the diverse laundry folding and espresso tasks from including on-robot data (the improvement from **offline RL + SFT** to the final $\pi^*_{0.6}$ model), and the rate of failure reduces by about a factor of two. On the easier laundry task (t-shirts and shorts), the success rate is already close to the maximum after the SFT phase, but throughput still increases by a significant margin with the final model.

On all of the tasks except diverse laundry, the success rate of the final $\pi^*_{0.6}$ model is in the 90%+ range. This makes it feasible to use in practical settings, such as making espresso drinks at the office or assembling boxes in a factory, as shown in the accompanying videos. For the box assembly task, Figure 8 (right) contains a breakdown of the task success over its four stages: picking up a box sheet, building the box, labeling the box, and placing it at an available spot in a crate. $\pi^*_{0.6}$ attains higher success rates for all of the stages compared to the other models. The majority of failures on these stages happen because the policy runs out of time. The accompanying videos present time lapses where each of the tasks is run for multiple hours.

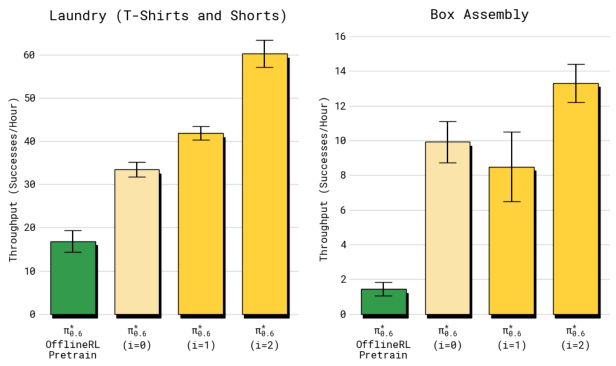
Fig. 9: **Improvement in throughput over multiple iterations.** Both tasks improve significantly in throughput as we take more iterations of RECAP, with box assembling first dropping and then improving significantly.

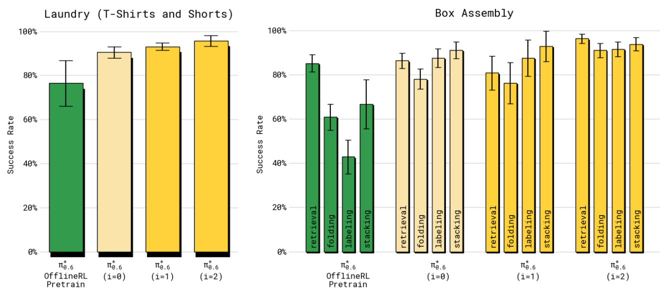
Fig. 10: **Improvement in success rate over multiple iterations.** The laundry task quickly reaches the maximum success rate (but continues to improve in throughput as shown in Figure 9, while box assembly continues to improve.

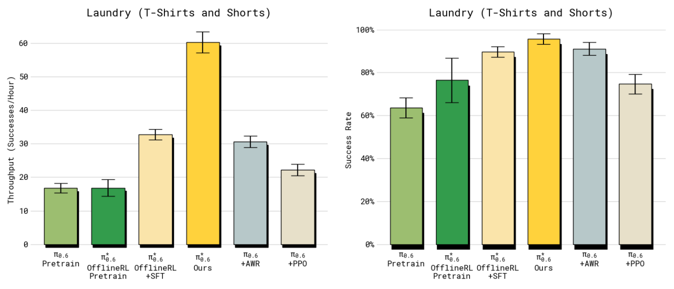
Fig. 11: **Comparison of different policy extraction methods.** RECAP applied to $\pi^*_{0.6}$ achieves by far the highest throughput for the laundry task compared to AWR and PPO.

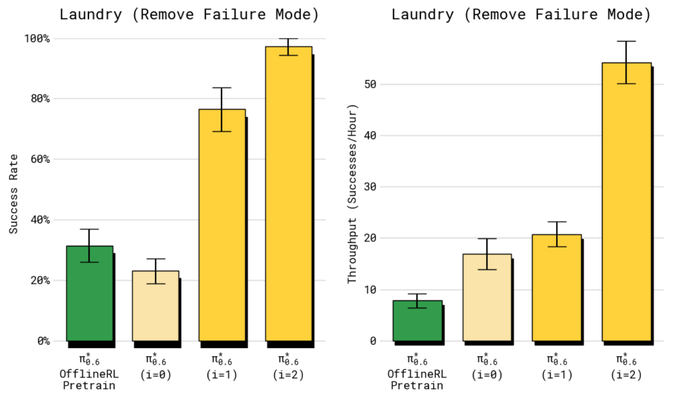
Fig. 12: **Failure mode removal.** Here we apply RECAP on a variant of the laundry task with one item but a very strict success criteria. RECAP is particularly effective at removing failure modes that would be considered non successful under the strict criteria. Therefore, our method can also be used to alter a policy's behavior with relatively little data effectively.

2) How much does RECAP *improve $\pi^*_{0.6}$ over multiple iterations?:* We next elucidate how training with RECAP improves policies through multiple iterations of data collection and training. We study the T-shirt and shorts folding task and the box assembly task. For the T-shirt folding task, only data collected with autonomous evaluation (without human corrections) is used to perform policy improvement over two iterations, in order to evaluate how well our method can improve the policy via RL alone. We collect 300 trajectories on four robots in each iteration. Box assembly uses both autonomous trials and trials with expert teleoperator interventions, with 600 autonomous trials and 360 trials with interventions in each iteration.

We plot the throughput over iterations in Figure 9, comparing two iterations of RECAP, denoted by $i = 1$, $i = 2$ respectively. The final iteration, labeled (Ours), corresponds to the overall best result for these tasks presented in the previous section. We also compare the initial data collection policy, which uses the offline RL pre-trained $\pi^*_{0.6}$ model with SFT finetuning. For both tasks, $\pi^*_{0.6}$ improves over the two iterations. In the laundry task we can see steady improvement yielding an overall 50% improvement in throughput. For the long-horizon box assembly task, more data is needed to yield a significant improvement, but after the second iteration we see a $2\times$ improvement in throughput.

We also show the success rate over the iterations in Figure 10. For the laundry task, the first iteration already raises the success rate to over 90%, while the second iteration mainly improves throughput. For the box assembly task, we see clear improvements in the success rate over both iterations. While there are still some failures (especially when placing the box on the stack at the end), the final policy achieves a success rate of about 90% both for folding the box and labeling it in the allocated time limit of 600 seconds.

3) How does the advantage-conditioned policy extraction method in RECAP *compare to other methods?:* We compare our advantage conditioned policy extraction method from Section IV-B to other methods in the literature: AWR and PPO. We use the T-shirts and Shorts task for this comparison. To ensure a controlled comparison, we use the same data for these comparisons that was used to train our final model. This provides a slight advantage to the baselines, since they have access to better data that was collected while running RECAP. The results are shown in Figure 11. While both AWR and PPO can attain reasonable results, they both fall far short of our method, and struggle to improve over the offline RL + SFT $\pi^*_{0.6}$ model. For PPO, we had to use a small trust-region constraint ($\eta = 0.01$) to stabilize training in this off-policy setting, and while this makes training stable, the method does not achieve good performance. AWR can achieve a reasonable success rate, but leads to much slower polies with lower throughput.

4) Can Recap *significantly alter policy behavior with relatively little data and remove a failure mode?:* While the preceding experiments have focused on holistic end-to-end evaluations of policy performance, we can also zoom in on a specific failure mode to examine whether RL training with Recap can remove a specific mistake from the policy. To answer this question, we use a version of the laundry task with a strict success criterion, which requires the policy to fold a t-shirt with the collar centered and facing up. Each episode is initialized with a specific adversarial condition in which the shirt is placed flat on the table in such a way that the baseline offline RL + SFT policy often fails to fold it correctly. As shown in Figure 12, applying Recap in this setting for two iterations (collecting 600 trajectories in each iteration) results in a policy that succeeds 97% of the time, and with high speed. Thus we conclude that Recap can be effective at removing specific failure modes, even when learning entirely via RL without any intervention data or additional demonstrations.

VII. Discussion and Future Work

Training policies that can achieve the same robustness, speed, and fluency on real-world tasks as people presents a major challenge in robotic learning. In this paper, we discussed how learning from experience, through a combination of DAgger-style coaching and RL, can begin to address this challenge. We describe Recap, a method for training VLAs with autonomous trials, reward feedback, and human interventions, and present results for a model trained with Recap, $\pi^*_{0.6}$, on a set of realistic tasks: making espresso drinks, folding diverse laundry, and assembling boxes. At the core of Recap is an RL method that is well-suited for scalable training of VLA policies, using advantage conditioning for policy extraction with value functions. The data for this RL method is collected with a combination of autonomous rollouts and human interventions, correcting mistakes with interventions while finetuning the details of the behavior on autonomous data. Our experiments show that Recap can improve both the success rate and throughput of the VLA, more than doubling the throughput on some of the harder tasks, and decreasing the number of failures by roughly $2\times$.

There are several directions for improvement with Recap. First, our system is not fully autonomous: it relies on human labeling and effort for reward feedback, interventions, and episode resets. A number of prior works have explored ways to automate these components [84, 85], and VLAs offer new ways to provide for more automated data collection, for example by using high-level policies [86] to reason through resetting the scene. Second, our system is relatively naïve in how it approaches exploration: exploration is largely greedy, relying on stochasticity in the policy and human interventions to explore new solutions. This is reasonable when the initial imitation learning policy already takes reasonable actions, but there is plenty of room for improvement with more sophisticated exploration methods. Lastly, Recap performs iterated "offline" updates (i.e., it collects a batch of data, retrains the model, and repeats), rather than running a fully online RL loop where the policy and value function are updated in real time as data is collected. We make this decision out of convenience, but extending our approach into a fully concurrent online RL framework is a promising direction for future work.

More broadly, training VLAs with RL is perhaps the most direct path to get to performance levels that are adequate for real-world use cases. RL with VLAs presents a number of challenges, from the difficulty of large-scale RL training of high capacity models to sample complexity, autonomy, and delayed feedback. While existing RL frameworks designed for smaller-scale systems or "virtual" domains such as LLMs can provide a good starting point, more research will be needed to make RL a practical tool for VLA training. We hope that our work represents a meaningful step in this direction.

Acknowledgements

We thank our robot operators for data collection, evaluations, logistics, and video recording, and our technicians for robot maintenance and repair. See Appendix A for a full contributions statement.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018. 1

[2] Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In Marco A. Wiering and Martijn van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 45–73. Springer, 2012. doi: 10.1007/978-3-642-27645-3_2. 2, 4

[3] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020. 2, 4

[4] Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Diffusion guidance is a controllable policy improvement operator. *arXiv preprint*, arXiv:2505.23458, 2025. 2, 3, 4, 5, 17

[5] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025. 2, 3, 5, 7, 8

[6] Physical Intelligence Team. $\pi_{0.6}$ model card. 2025. 2, 5, 6, 8

[7] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, pages 627–635, 2011. 2, 7

[8] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Proceedings of the 2016 IEEE International Conference*

on Robotics and Automation (ICRA), pages 462–469, 2016. doi: 10.1109/ICRA.2016.7487175. 2

[9] Michael Laskey, Jonathan Lee, Roy Fox, Anca D. Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1989–1998. PMLR, 2017.

[10] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022. 2

[11] Zheyuan Hu, Robyn Wu, Naveen Enock, Jasmine Li, Riya Kadakia, Zackory Erickson, and Aviral Kumar. Rac: Robot learning for long-horizon tasks by scaling recovery and correction. *arXiv preprint*, arXiv:2509.07953, 2025. 2

[12] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *ICRA*, 2019. 2

[13] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 2

[14] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[15] Ajay Mandlekar, Fabio Ramos, Byron Boots, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *ICRA*, 2020.

[16] Archit Sharma, M. Ahmed Ahmed Rehaan Ahmad, and Chelsea Finn. Self-improving robots: End-to-end autonomous visuomotor reinforcement learning. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, volume 229, pages 3292–3308. PMLR, 2023.

[17] Russell Mendonca, Shikhar Bahl, and Deepak Pathak. Alan: Autonomously exploring robotic agents in the real world. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3044–3050, 2023. doi: 10.1109/ICRA48891.2023.10013321.

[18] Russell Mendonca, Emmanuel Panov, Bernadette Bucher, Jiuguang Wang, and Deepak Pathak. Continuously improving mobile manipulation with autonomous real-world rl. In *Proceedings of the 8th Conference on Robot Learning (CoRL)*, pages 5204–5219, 2024.

[19] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning, 2024.

[20] Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual off-policy rl for finetuning behavior cloning policies. *arXiv preprint arXiv:2509.19301*, 2025.

[21] Thomas Lampe, Abbas Abdolmaleki, Sarah Bechtle, Sandy H. Huang, Jost Tobias Springenberg, Michael Bloesch, Oliver Groth, Roland Hafner, Tim Hertweck, Michael Neunert, Markus Wulfmeier, Jingwei Zhang, Francesco Nori, Nicolas Heess, and Martin Riedmiller. Mastering stacking of diverse shapes with large-scale iterative reinforcement learning on real robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7772–7779, 2024. doi: 10.1109/ICRA57147.2024.10610297. 2

[22] Perry Dong, Suvir Mirchandani, Dorsa Sadigh, and Chelsea Finn. What matters for batch online reinforcement learning in robotics? *arXiv preprint*, arXiv:2505.08078, 2025. 2

[23] Allen Z. Ren, Justin Lidard, Lars Lien Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion Policy Policy Optimization. In *Proceedings of the 2025 International Conference on Learning Representations (ICLR)*, 2025. 9, 17

[24] Kun Lei, Huanyu Li, Dongjie Yu, Zhenyu Wei, Lingxiao Guo, Zhennan Jiang, Ziyu Wang, Shiyu Liang, and Huazhe Xu. Rl-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint*, arXiv:2510.14830, 2025. 2

[25] Dmitry Kalashnkov, Jake Varley, Yevgen Chebotar, Ben Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv*, 2021. 2

[26] Abhishek Gupta, Justin Yu, Tony Z. Zhao, Vikash Kumar, Aaron Rovinsky, Kelvin Xu, Thomas Devlin, and Sergey Levine. Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention. In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6664–6671, 2021. 2

[27] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023. 2

[28] Aviral Kumar, Anikait Singh, Frederik Ebert, Mitsuhiko Nakamoto, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline reinforcement learning enables learning new tasks from a handful of trials. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023. doi: 10.15607/RSS.2023.XIX.019.

[29] Jingyun Yang, Max Sobol Mark, Brandon Vu, Archit Sharma, Jeannette Bohg, and Chelsea Finn. Robot

fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024. doi: 10.1109/ICRA57147.2024.10610421. 2

[30] Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive post-training for vision-language-action models. *arXiv preprint*, arXiv:2505.17016, 2025. 2

[31] Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. Vla-rl: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint*, arXiv:2505.18719, 2025.

[32] Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can rl bring to vla generalization? an empirical study. *arXiv preprint*, arXiv:2505.19789, 2025.

[33] Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Quanlu Zhang, Zhaofei Yu, Guoliang Fan, Tiejun Huang, Yu Wang, and Chao Yu. $\pi_{\texttt{rl}}$: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint*, arXiv:2510.25889, 2025.

[34] Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. Simplevla-rl: Scaling vla training via reinforcement learning. *arXiv preprint*, arXiv:2509.09674, 2025. 2

[35] Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. Improving vision-language-action model with online reinforcement learning. *arXiv preprint*, arXiv:2501.16664, 2025. 2

[36] Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Yuqi Xie, Fengyuan Hu, Jimmy Wu, Zhengyi Luo, Linxi "Jim" Fan, Guanya Shi, and Yuke Zhu. Self-improving vision-language-action models with data generation via residual rl, 2025. 2

[37] Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. Conrft: A reinforced fine-tuning method for vla models via consistency policy. *arXiv preprint arXiv:2502.05450*, 2025. 2

[38] Max Sobol Mark, Tian Gao, Georgia Gabriela Sampaio, Mohan Kumar Srirama, Archit Sharma, Chelsea Finn, and Aviral Kumar. Policy-agnostic rl: Offline rl and online rl fine-tuning of any class and backbone. *arXiv preprint*, arXiv:2412.06685, 2024. 2

[39] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. In *Conference on Robot Learning*, pages 4996–5013. PMLR, 2025.

[40] Yang Zhang, Chenwei Wang, Ouyang Lu, Yuan Zhao, Yunfei Ge, Zhenglong Sun, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. Align-then-steer: Adapting the vision-language action models through unified latent guidance. *arXiv preprint arXiv:2509.02055*, 2025. 2

[41] Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Proceedings of the 9th Conference on Robot Learning (CoRL)*, 2025. 2

[42] Charles Xu, Qiyang Li, Jianlan Luo, and Sergey Levine. Rldg: Robotic generalist policy distillation via reinforcement learning. *arXiv preprint arXiv:2412.09858*, 2024. 2

[43] Dongchi Huang, Zhirui Fang, Tianle Zhang, Yihang Li, Lin Zhao, and Chunhe Xia. Co-rft: Efficient fine-tuning of vision-language-action models through chunked offline reinforcement learning. *arXiv preprint*, arXiv:2508.02219, 2025. 3

[44] Zijian Zhang, Kaiyuan Zheng, Zhaorun Chen, Joel Jang, Yi Li, Siwei Han, Chaoqi Wang, Mingyu Ding, Dieter Fox, and Huaxiu Yao. Grape: Generalizing robot policy via preference alignment. *arXiv preprint*, arXiv:2411.19309, 2024. 3

[45] Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Fuxian Huang, Haoran Zhang, Ming Zhou, Shengzhe Zhang, Litao Liu, Sixu Lin, and Jiangmiao Pang. A vision-language-action-critic model for robotic real-world reinforcement learning. *arXiv preprint*, arXiv:2509.15937, 2025. 3

[46] Seyed Kamyar Ghasemipour, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, and Igor Mordatch. Self-improving embodied foundation models. *arXiv preprint*, arXiv:2509.15155, 2025. 3

[47] Jürgen Schmidhuber. Reinforcement learning upside down: Don't predict rewards — just map them to actions. *arXiv preprint*, arXiv:1912.02875, 2019. 3, 4

[48] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *CoRR*, abs/1912.13465, 2019. 4

[49] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems (NeurIPS) 34*, 2021.

[50] David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? In *Advances in Neural Information Processing Systems (NeurIPS) 35*, 2022. 4

[51] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[52] Hiroki Furuta, Yusuke Matsuo, and Shixiang Shane Gu.

Generalized decision transformer for offline hindsight information matching. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[53] Taku Yamagata, Ahmed Khalil, and Raúl Santos-Rodríguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pages 38989–39007. PMLR, 2023.

[54] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 27042–27059. PMLR, 2022.

[55] Jakub Grudzien Kuba, Pieter Abbeel, and Sergey Levine. Advantage-conditioned diffusion: Offline rl via generalization. 2023.

[56] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023. doi: 10.5555/3666122.3666936. 3

[57] Lin Shao, Toki Migimatsu, Qiang Zhang, Kaiyuan Yang, and Jeannette Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. In *Proceedings of Robotics: Science & Systems (RSS)*, 2020. doi: 10.15607/RSS.2020.XVI.082. 3

[58] Annie S. Chen, Suraj Nair, and Chelsea Finn. Learning generalizable robotic reward functions from "in-the-wild" human videos. In *Proceedings of Robotics: Science & Systems (RSS) 2021*, 2021.

[59] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, volume 164 of *Proceedings of Machine Learning Research*, pages 1303–1315. PMLR, 2022.

[60] Sumedh A. Sontakke, Jesse Zhang, Sébastien M.R. Arnold, Karl Pertsch, Erdem Bıyık, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. Roboclip: One demonstration is enough to learn robot policies. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[61] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to rewards for robotic skill synthesis. In *Proceedings of the 7th Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pages 374–404. PMLR, 2023.

[62] Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J. Lim, Jesse Thomason, Erdem Bıyık, and Jesse Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. In *Proceedings of the 9th Conference on Robot Learning (CoRL)*, 2025.

[63] Minttu Alakuijala, Reginald McLean, Isaac Woungang, Nariman Farsad, Samuel Kaski, Pekka Marttinen, and Kai Yuan. Video-language critic: Transferable reward functions for language-conditioned robotics. *Transactions on Machine Learning Research*, 2025:1–22, 2025. 3

[64] Yecheng Jason Ma, William Liang, Vaidehi Som, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. 3

[65] Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, Osbert Bastani, Dinesh Jayaraman, Wenhao Yu, Tingnan Zhang, Dorsa Sadigh, and Fei Xia. Vision language models are in-context value learners. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. 3

[66] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3, 4, 17

[67] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018. 3

[68] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019. 4, 9

[69] Peter Dayan and Geoffrey E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997. doi: 10.1162/neco.1997.9.2.271.

[70] Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1607–1612. AAAI Press, 2010. 3

[71] Qing Wang, Jiechao Xiong, Lei Han, peng sun, Han Liu, and Tong Zhang. Exponentially weighted imitation learning for batched historical data. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018. 3

[72] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages

449–458. PMLR, 2017. 4

[73] Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z Ren, Homer Walke, Quan Vuong, Lucy Xiaoyang Shi, et al. Knowledge insulating vision-language-action models: Train fast, run fast, generalize better. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2025. 4, 6

[74] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ICML*, 2018. 4

[75] Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7768–7778, 2020. 4

[76] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. 4

[77] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient action tokenization for vision-language-action models. *Robotics: Science and Systems*, 2025. 6

[78] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. 6

[79] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022. 6, 16

[80] Diederik Kingma and Ruiqi Gao. Understanding diffusion objectives as the elbo with simple data augmentation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 65484–65516, 2023. 6, 16

[81] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 8

[82] David McAllister, Songwei Ge, Brent Yi, Chung Min

Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients, 2025. 9, 16, 17

[83] Zhengpeng Xie, Qiang Zhang, Fan Yang, Marco Hutter, and Renjing Xu. Simple policy optimization. In *Forty-second International Conference on Machine Learning (ICML)*, 2025. 9, 17

[84] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020. 11

[85] Archit Sharma, Kelvin Xu, Nikhil Sardana, Abhishek Gupta, Karol Hausman, Sergey Levine, and Chelsea Finn. Autonomous reinforcement learning: Formalism and benchmarking. *arXiv preprint arXiv:2112.09605*, 2021. 11

[86] Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025. 11

APPENDIX

A. Contributions

Data collection and operations. Michael Equi, Chelsea Finn, Lachy Groom, Hunter Hancock, Karol Hausman, Rowan Jen, Liyiming Ke, Marinda Lamb, Vishnu Mano, Suraj Nair, Charvi Sharma, Laura Smith, Will Stoeckle, Anna Walling, Blake Williams.

Annotation and supplemental data. Chelsea Finn, Catherine Glossop, Hunter Hancock, Brian Ichter, Rowan Jen, Liyiming Ke, Chandra Kuchi, Karl Pertsch, Laura Smith, Will Stoeckle, Quan Vuong, Anna Walling.

Policy training and research. Ashwin Balakrishna, Kevin Black, Danny Driess, Michael Equi, Yunhao Fang, Chelsea Finn, Catherine Glossop, Karol Hausman, Gashon Hussein, Brian Ichter, Liyiming Ke, Sergey Levine, Yao Lu, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Alex Swerdlow, Marcel Torne, Quan Vuong, Lili Yu, Zhiyuan Zhou.

Policy infrastructure. Kevin Black, Karan Dhabalia, Danny Driess, Michael Equi, Liyiming Ke, Adrian Li, Suraj Nair, Allen Z. Ren, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Alex Swerdlow, Haohuan Wang, Ury Zhilinsky, Zhiyuan Zhou.

Robot hardware. Ali Amin, Raichelle Aniceto, Grace Connors, Adnan Esmail, Thomas Godden, Ivan Goryachev, Tim Jones, Ben Katz, Devin LeBlanc, Mohith Mothukuri, Sukwon Yoo.

Robot infrastructure. Ken Conley, James Darpinian, Jared DiCarlo, Karol Hausman, Szymon Jakubczak, James Tanner.

Writing and illustration. Kevin Black, Danny Driess, Michael Equi, Chelsea Finn, Hunter Hancock, Karol Hausman, Brian Ichter, Liyiming Ke, Sergey Levine, Suraj Nair, Allen Z. Ren, Laura Smith, Jost Tobias Springenberg, Zhiyuan Zhou

B. Additional Value Function Visualization

Figure 13 shows additional visualizations of our trained value function on five different tasks, including tasks on which we evaluate our policies (espresso making, box assembly) and also broader tasks (hang towel, attach hook). The parts with the most prominent changes are highlighted: red corresponds to where value function drops, green corresponds to where value function increases, and yellow corresponds to oscillating values. Images show the corresponding frames and description of the episode.

C. Computing the log-likelihood for policy improvement

To derive the log-likelihood from Equation (4) we can first observe that we can decompose the full model likelihood into autoregressive and diffusion terms

$$\pi_\theta(\mathbf{a}_{t:t+H}, \mathbf{a}^\ell_{t:t+H}, \hat{\ell}|I_t, \mathbf{o}_t, \ell) = \pi_\theta(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell})\pi_\theta(\mathbf{a}^\ell_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell})\pi_\theta(\hat{\ell}|I_t, \mathbf{o}_t, \ell), \tag{6}$$

where the first term is modeled with flow matching, the second term is the autoregressive likelihood of the discretized actions $\mathbf{a}^\ell_{t:t+H}$, and the third term corresponds to the autoregressive text likelihood. The autoregressive likelihoods can be estimated in the usual way, using the cross-entropy loss evaluated on ground truth tokens. For the continuous likelihood over $\mathbf{a}_{t:t+H}$, a closed form likelihood is not available [79]. We can, however follow prior work [82], and consider the one-step diffusion process as a Gaussian distribution with likelihood

$$\log\pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{a}^{\eta,\omega}_{1:H}, I_t, \mathbf{o}_t, \ell, \hat{\ell}) = \log\mathcal{N}\Big(\omega - f_\theta(\mathbf{a}^{\eta,\omega}_{1:H}, I_t, \mathbf{o}_t, \ell, \hat{\ell}), \mathbf{I}\Big), \tag{7}$$

with $\mathbf{a}^{\eta,\omega}_{t:t+H} = \eta\mathbf{a}_{t:t+H} + (1-\eta)\omega$ and $\omega = \mathcal{N}(0, \mathbf{I})$. From this we can form an evidence lower bound to the likelihood following [80, 82] (effectively marginalizing over η and ω) which yields

$$\log\pi_\theta(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell}) \geq \frac{1}{2}\mathbb{E}_{\eta,\omega}\Big[-w(\eta)\Big\|\omega - \mathbf{a}_{1:H} - f_\theta(\mathbf{a}^{\eta,\omega}_{1:H}, I_t, \mathbf{o}_t, \ell, \hat{\ell})\Big\|^2\Big] + c, \tag{8}$$

where $w(\eta) = e^{-\eta/2}$ is a noise dependent weighting term, and c is a constant independent of f_θ. For the derivation, see [80], which also derives the relationship between flow matching and diffusion in Appendix D.3 for this choice of weighting term. Finally putting the lower bound together with the autoregressive likelihood for the discretized action part of the text output $\hat{\ell}$, and subsuming the weighting terms in α, gives

$$\log\pi_\theta(\mathbf{a}_{t:t+H}, \mathbf{a}^\ell_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell}) \geq \mathbb{E}_{\eta,\omega}\Big[\log p_\theta(\mathbf{a}^\ell_{t:t+H}|I_t, \mathbf{o}_t, \ell, \hat{\ell}) - \alpha_\eta\Big\|\omega - \mathbf{a}_{1:H} - f_\theta(\mathbf{a}^{\eta,\omega}_{1:H}, I_t, \mathbf{o}_t, \ell, \hat{\ell})\Big\|^2\Big], \tag{9}$$

which is the bound given in the main part of the paper.

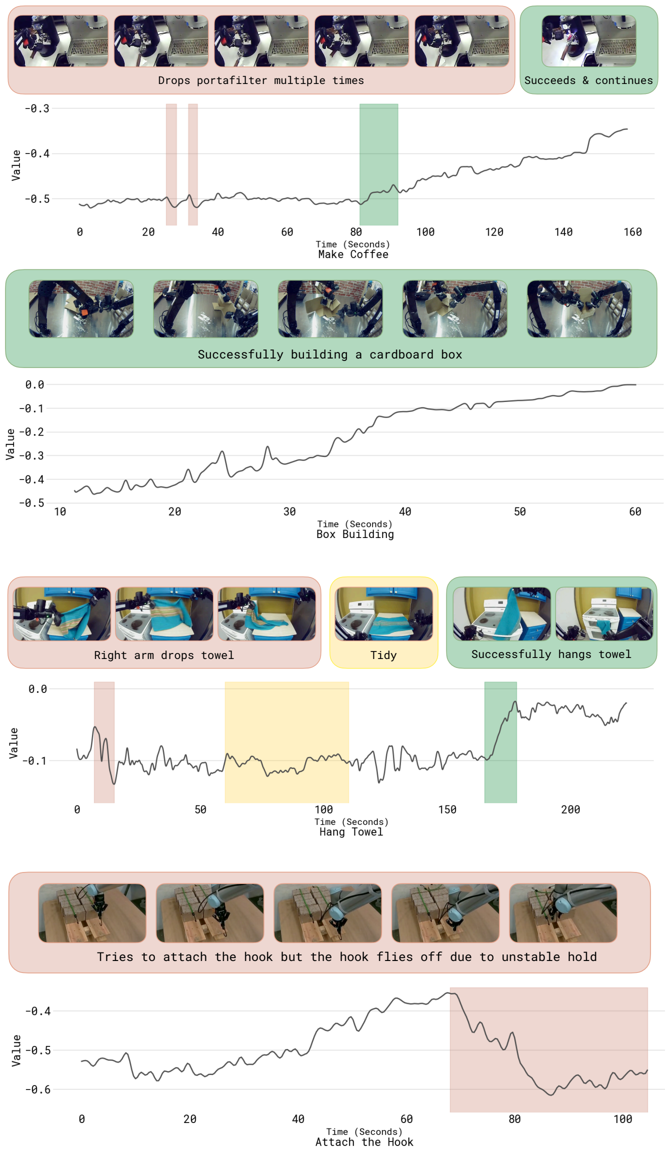

Fig. 13: **Additional visualization of value function on five different tasks.** Red parts highlight places where value drops, green parts highlight places where value increases, and yellow parts highlight oscillating value regions. Images show the corresponding frames and descriptions of the episode.

D. PPO implementation

We implement a variant of PPO [66] related to DPPO and FPO [23, 82] and use it as an additional baseline. To allow for training both the autoregressive part of the model as well as the diffusion based action expert in a compute effective manner we calculate likelihoods based on the single step diffusion objective alone.

In particular, we use a likelihood bound analogous to Eq. (9) (previous section) but without the improvement indicator. Decomposing into autoregressive and flow-matching terms this can be written as

$$\begin{aligned}\log \pi_\theta(\mathbf{a}_{t:t+H}, \mathbf{a}^{\ell}_{t:t+H}|\mathbf{o}_t, \ell, \hat{\ell}) \geq\\ \mathbb{E}_{\eta,\omega}\Big[\log p_\theta(\mathbf{a}^{\ell}_{t:t+H}|\mathbf{o}_t, \ell, \hat{\ell})\\ - \alpha_\eta \left\|\omega - \mathbf{a}_{1:H} - f_\theta(\mathbf{a}^{\eta,\omega}_{1:H}, \mathbf{o}_t, \ell, \hat{\ell})\right\|^2\Big],\end{aligned} \tag{10}$$

which is analogous to the diffusion likelihood bound used in FPO [82]. And we combine it with a PPO style loss separated into diffusion and autoregressive terms. In preliminary experiments we found that for our setting it was difficult to enforce a trust region constraint on the action expert (which models actions with an unbounded diffusion head) when using the standard PPO clipping objective. Presumably, this is partially due to the "offline" nature of our algorithm setting, where we cannot afford to collect new data from real robots every few gradient steps. To stabilize training we found using an alternative definition of the PPO constraint following SPO [83] to be effective. The resulting loss is given as:

$$\begin{aligned}&\mathcal{L}_{SPO+CoVLA}(\theta) =\\ &\Bigg\{\frac{\pi_\theta(a_{\hat{\ell}} \in \hat{\ell}|\mathbf{o}_t, \ell)}{\pi_{\text{ref}}(a_{\hat{\ell}} \in \hat{\ell}|\mathbf{o}_t, \ell)} A^{\pi_{\text{ref}}}(o_t, a_t, \ell)\\ &\quad - \frac{|A^{\pi_{\text{ref}}}(o_t, a_t, \ell)|}{2\epsilon_{\text{ar}}}\left[\frac{\pi_\theta(a_{\hat{\ell}} \in \hat{\ell}|\mathbf{o}_t, \ell)}{\pi_{\text{ref}}(a_{\hat{\ell}} \in \hat{\ell}|\mathbf{o}_t, \ell)} - 1\right]\Bigg\}\\ &+\alpha\Bigg\{\frac{\pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)}{\pi_{\text{ref}}(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)} A^{\pi_{\text{ref}}}(o_t, a_t, \ell)\\ &\quad - \frac{|A^{\pi_{\text{ref}}}(o_t, a_t, \ell)|}{2\epsilon_{\text{flow}}}\left[\frac{\pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)}{\pi_{\text{ref}}(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)} - 1\right]\Bigg\},\end{aligned} \tag{11}$$

where α is a trade-off parameter and ϵ_{ar}, ϵ_{flow} are trust-region parameters for autoregressive and flow-matching model parts respectively. We use this variant to perform training on eval data starting from the $\pi_{0.6}$ checkpoint.

E. Using CFG for test-time policy improvement with $\beta > 1$

After training we can choose to further sharpen the policy used for evaluation by setting $\beta > 1$ in Eq. (2). As shown in prior work [4] we can recover this sharpened policy without additional training since it is implicitly defined by the learned policies $\pi_\theta(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell)$ and $\pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)$. Specifically, after training we can form the approximation

$$\hat{\pi}(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell) \propto \pi_{\text{ref}}(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)\left(\frac{\pi_{\text{ref}}(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell)}{\pi_{\text{ref}}(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)}\right)^\beta. \tag{12}$$

One can now realize that the diffusion model effectively learns the gradient of the likelihoods, i.e. it represents $\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell)$ and $\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)$ respectively. From this, following Frans et al. [4], we can see that if we run flow-matching inference following the gradient

$$\begin{aligned}&\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)+\\ &\quad\beta(\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|I_t, \mathbf{o}_t, \ell) - \nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \ell)),\end{aligned} \tag{13}$$

we are effectively sampling from the desired attenuated distribution. We note that, as mentioned in the main paper, the parameter β is loosely connected to the advantage threshold ϵ_ℓ that we introduce during training (in the sense that both sharpen the distribution, one at inference and one at training time). We find that sharpening the distribution after training with high settings for β can lead to pushing the action distribution towards the boundaries of its learned support (which can lead to overly aggressive motions) and thus primarily rely on ϵ_ℓ for obtaining a good conditioned policy directly after training and combine it with moderate settings (e.g. $\beta \in [1.5, 2.5]$) where useful.

F. Additional algorithm details

We describe details for setting the task specific parameters used in Algorithm 1.

Advantage Estimation: During post-training, we estimate the advantage function using $A^\pi(\mathbf{o}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+N-1} r'_t + V^\pi(\mathbf{o}_{t+N}) - V^\pi(\mathbf{o}_t)$, where $\mathbf{o}_{t+N}$ is an observation sampled from N steps ahead from the same trajectory. We use $N = 50$ lookahead to calculate this advantage. During pre-training, we calculate the advantage estimate as $A^\pi(\mathbf{o}_t, \mathbf{a}_t) = \sum_{t'=0}^{T} r'_t - V^\pi(\mathbf{o}_t)$, setting $N = T$ for each episode, which is a higher variance estimate of the advantage. We use this advantage calculation since it allows us to calculate the advantage values on-the-fly during pre-training using a single inference call to the value function. We find empirically that this advantage estimate works well when the policy is trained on large amounts of data from diverse tasks during pre-training.

Advantage conditioning dropout: During training, we randomly drop out the conditioning on the advantage indicator 30% of the time. We employ this dropout so that we can directly sample directly from either the conditional or unconditional policy during inference time and use CFG for test-time policy improvement (see Section E for details); and it effectively replaces the loss multiplier α.

Advantage threshold: The per task advantage threshold ϵ_ℓ is set as follows. During pre-training we select the threshold for each task such that approximately 30% of the demonstration data has positive advantage (as calculatedd on a random sample of 10k datapoints). During fine-tuning we generally set the threshold such that approximately 40% of the evaluation rollouts in each iteration have positive advantage. For the T-shirt and shorts laundry folding task (in which training on high-quality demonstration data yields slow policies but with high success rate) we increase the threshold such that only approximately 10% of the data has positive advantage.

Dataset composition: We use the dataset aggregation strategy described in Algorithm 1 for all tasks. However each of our task has distinct nature: the episode lengths vary, the performances of Iteration 0 model on each task are different, and one task (Assemble Box) is performed offsite in a deployment scenario. Therefore, we have different amount of demonstration data to begin with and collect different amounts of experience data for iterative improvement. For laundry (T-shirt and shorts), we use autonomous evaluation data only without expert corrections. As we push model performance to closely resemble the expert data collector in terms of speed, it becomes hard to provide corrections. For this task, We collect 300 episodes across 4 robot stations for reporting eval performance. For the diverse laundry folding task we collect 450 evaluation episodes and 287 correction episodes. For the failure mode removal ablation we collect both autonomous and policy correction data. In total we collect ~ 1000 autonomous and $280 + 378$ correction episodes spread over 3 robots. For box assembly we collect data in the deployment scenario directly, collecting 600 demonstrations and 360 correction episodes in each iteration, using 3 robots in total. For cafe we perform a single iteration and collect 429 correction episodes as well as 414 autonomous episodes.