SERNF: Sample-Efficient Real-World Dexterous Policy Fine-Tuning via Action-Chunked Critics and Normalizing Flows

Anonymous Authors

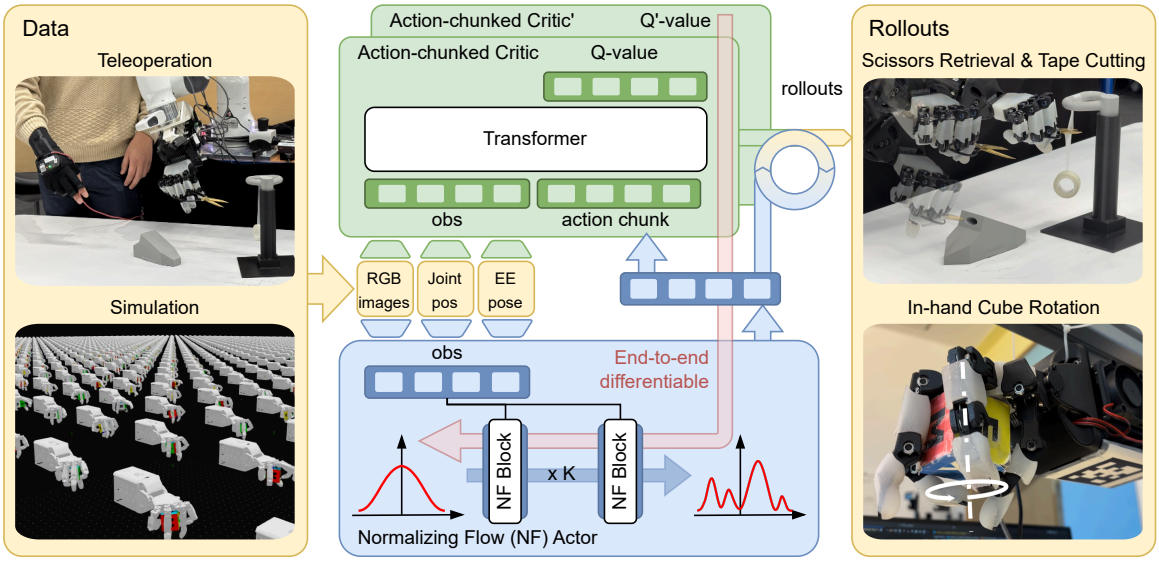

Fig. 1. Overview of SERNF (**S**ample-**E**fficient **R**einforcement Learning with **N**ormalizing **F**lows). SERNF models action chunks using a conditional normalizing flow policy, enabling expressive multimodal action distributions with exact likelihoods and therefore allowing direct, conservative off-policy fine-tuning. An action-chunked critic evaluates entire action sequences, aligning value estimation with chunked execution and improving long-horizon credit assignment. Initialized from real-world teleoperation data or policies trained in simulation, SERNF achieves high real-world performance under a limited interaction budget, demonstrating stable and sample-efficient adaptation on challenging dexterous manipulation tasks.

***Abstract*—Real-world fine-tuning of dexterous manipulation policies remains challenging due to limited real-world interaction budgets and highly multimodal action distributions. Diffusion-based policies, while expressive, do not permit conservative likelihood-based updates during fine-tuning because action probabilities are intractable. In contrast, conventional Gaussian policies collapse under multimodality, particularly when actions are executed in chunks, and standard per-step critics fail to align with chunked execution, leading to poor credit assignment. We present SERNF, a sample-efficient off-policy fine-tuning framework with normalizing flow (NF) to address these challenges. The normalizing flow policy yields exact likelihoods for multimodal action chunks, allowing conservative, stable policy updates through likelihood regularization and thereby improving sample efficiency. An action-chunked critic evaluates entire action sequences, aligning value estimation with the policy's temporal structure and improving long-horizon credit assignment. To our knowledge, this is the first demonstration of a likelihood-based, multimodal generative policy combined with chunk-level value learning on real robotic hardware. We evaluate SERNF on two challenging dexterous manipulation tasks in the real world: cutting tape with scissors retrieved from a case, and in-hand cube rotation with a palm-down grasp – both of which require precise, dexterous control over long horizons. On these tasks, SERNF achieves stable, sample-efficient adaptation where standard methods struggle.**

I. Introduction

Robotic manipulation systems increasingly rely on high-capacity visuomotor policies trained from large offline datasets. While these policies can achieve impressive performance in controlled settings, deploying them reliably in the real world remains challenging, as manipulation policies must operate across a wide range of environments and conditions that inevitably change over time. In practice, real-world deployment is affected by numerous sources of mismatch, such as unmodeled dynamics, hardware errors, camera drift, domain shift, and task-specific issues that are difficult to capture during data collection.

These challenges heavily affect dexterous and precise manipulation tasks, where collecting large amounts of high-quality demonstration data is expensive in terms of both equipment and time. Such tasks are often hard to teleoperate, require expert skill, and are sensitive to small variations in contact and timing, making scalable data collection costly and error-prone. As a result, roboticists frequently deploy policies that "almost" work, showing reasonable behavior but not achieving the reliability and success rates required for real-

world, in-the-wild scenarios.

In contrast to the prevailing paradigm of scaling real-robot datasets, we target settings where only limited task-specific data is available and fine-tuning must extract maximal improvement from each additional rollout.

In this setting, sample efficiency is a critical requirement for policy fine-tuning algorithms. Since real-world robot interaction time is costly, an effective algorithm and architecture should operate under a tight interaction budget and make maximum use of each collected sample. Moreover, to enable RL updates, the policy should be formulated probabilistically, allowing optimization through likelihood-based objectives.

We propose to represent the policy with a *normalizing flow* (NF) and to learn an *action-chunked critic* that evaluates temporally-extended action sequences (chunks). This combination aims to (i) keep the policy expressive yet tractable for likelihood-based regularization and stable updates, and (ii) improve credit assignment and variance reduction when the control interface executes action chunks.

We evaluate on two real-world dexterous manipulation tasks chosen to reflect two dominant deployment pipelines in robotics. (i) Scissors retrieval and tape cutting start from a small set of human teleoperation demonstrations, representing the common imitation-learning workflow for contact-rich tasks where simulation is difficult. (ii) In-hand cube rotation starts from a policy trained in simulation and distilled into our architecture, representing the mainstream sim-to-real reinforcement-learning workflow for dexterous hands. Both tasks are long-horizon, high-dimensional, and sensitive to small contact and timing errors, making them strong stress tests for sample-efficient fine-tuning.

The main contributions of this study can be summarized as follows:

- We introduce an RL fine-tuning method for real-world visuomotor control that combines *normalizing-flow* policies with an *action-chunked critic*.
- We describe a practical training recipe for *limited on-robot data*, including conservative regularization toward the initial policy.
- We empirically evaluate our method on representative *dexterous and precise real-world tasks* and analyze the impact of chunk length and critic design.

II. Related Work

a) Policy optimization for generative policies: Diffusion and flow-matching policies enable expressive, multimodal action distributions for imitation learning [7, 10, 51], but their use in reinforcement learning is limited by intractable action likelihoods, which hinder conservative or likelihood-regularized fine-tuning. Recent work connects diffusion- or flow-based models with policy optimization [2, 23, 34, 41, 52]. In contrast, we focus on *normalizing flows*, which retain comparable expressiveness while providing exact likelihoods, enabling stable and principled RL-based fine-tuning.

b) Normalizing flows for RL and IL: Normalizing flows have recently been shown to be effective policy representations for reinforcement and imitation learning [18, 29, 50], but their application to real robotic systems remains limited. Prior real-world deployments are restricted to on-policy settings [25], while most existing work focuses on simulation [1, 9, 33]. Our work leverages normalizing-flow policies for visuomotor control and focuses on stable, conservative fine-tuning on real hardware. We are first to apply normalizing flows within an *off-policy* reinforcement learning framework for real-world robotic fine-tuning.

c) Temporal abstraction and action chunking: Temporally extended action sequences are now standard in robotic control, enabling real-time execution and reduced feedback frequency [7]. Prior work shows that aligning critics with action chunking improves learning efficiency and credit assignment [28]. We extend these ideas by integrating action-chunked critics into a real-world fine-tuning pipeline.

d) Offline + online RL: Real-world robot learning commonly combines offline initialization with limited online fine-tuning. AWAC [37] and related methods enable off-policy adaptation but lack explicit conservatism in highly multimodal action spaces. Subsequent work studies the offline-to-online transition [3, 38], trading off conservatism and exploration at the cost of increased algorithmic complexity. Systems approaches such as IBRL and SERL emphasize the practical integration of demonstrations, off-policy RL, and tooling for real robots [21, 31], while extreme online-only efficiency has been demonstrated in locomotion from scratch [45]. In contrast, SERNF targets sample-efficient real-world fine-tuning of pretrained visuomotor policies under multimodal action distributions for sparse reward tasks by combining exact-likelihood generative policies with chunk-aligned value learning.

III. Background

a) MDP with observations and action chunks: We consider an episodic, discounted Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, T)$, with state space $s \in \mathcal{S}$, continuous actions $a_t \in \mathcal{A}$, transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, sparse reward function r, discount factor γ, and horizon T. The agent receives observations $o_t \in \mathcal{O}$ and selects an *action chunk* $a_{t:t+H}$, which is executed open-loop over the next H environment steps.

Following Li et al. [28], the H-step return is defined as

$$V_{H\text{-step}}(s_t) = \sum_{\tau=t}^{t+H-1} \gamma^{\tau-t} r_\tau + \gamma^H Q_{H\text{-step}}(s_{t+H}, a_{t+H:t+2H}), a_{t+H:t+2H} \sim \pi(s_{t+H}), \quad (1)$$

where $Q_{H\text{-step}}$ is an unbiased action-value function over H-step action chunks. This formulation propagates value information back over H steps at once and better matches modern imitation and visuomotor learning frameworks in which policies generate temporally extended action sequences.

b) Offline RL and multi-step bootstrapping: In offline reinforcement learning [27], we assume access to a fixed dataset $\mathcal{D} = \{(s, a, s', r)\}$ collected from $\mathcal{M}$ by an unknown

behavior policy π_β. We train a critic Q_ϕ by minimizing a temporal-difference loss with multi-step bootstrapping over action chunks of length H:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\substack{s_t,\, a_{t:t+H},\\ s_{t+H},\, r_{t:t+H}\sim\mathcal{D}}}\left[\left(Q_\phi(s_t, a_{t:t+H}) - \sum_{\tau=1}^{H}\gamma^\tau r_{t+\tau} - \gamma^H V_{\bar\phi}(s_{t+H})\right)^2\right], \quad (2)$$

where $\bar\phi$ denotes the parameters of a slowly updated target network with Polyak averaging configured by parameter τ. The value function $V_{\bar\phi}$ estimates the expected return under the current policy,

$$V_{\bar\phi}(s_t) = Q_{\bar\phi}(s_t, a_{t:t+H}), \quad a_{t:t+H} \sim \pi_\theta(s_t), \quad (3)$$

with policy parameters θ. The policy is trained to maximize the critic while remaining close to the behavior distribution:

$$\theta = \arg\max_\theta \mathbb{E}_{s_t\sim\mathcal{D}}\mathbb{E}_{a\sim\pi_\theta}\left[Q_{\bar\phi}(s_t, a)\right], \text{s.t. } D(\pi_\theta, \pi_\beta) < \epsilon, \quad (4)$$

where $D(\cdot,\cdot)$ is a divergence measure and ϵ controls the degree of regularization.

c) Conditional normalizing flows: Normalizing flows (NFs) [42] are expressive models capable of representing multimodal policies over high-dimensional action chunks [50, 29] and can be simply trained with reinforcement learning [18]. An input latent variable $z_0 \in \mathbb{R}^d$ is sampled from a base distribution p_0, and transformed through a sequence of invertible mappings $f_k(\cdot; c)$ conditioned on $c \in \mathbb{R}^m$:

$$z_K = f_\theta(z_0; c) = f_K(\cdots f_2(f_1(z_0; c); c); c), \quad (5)$$

where c typically encodes the current observation. The conditional density follows from the change-of-variables formula:

$$\log p_\theta(z_K \mid c) = \log p_0(z_0) - \sum_{k=1}^{K}\log\left|\det\frac{\partial f_k(\cdot; c)}{\partial z_{k-1}}\right|. \quad (6)$$

Maximizing (6) enables exact likelihood evaluation and efficient sampling. As a policy parameterization, this allows us to sample action chunks $\mathbf{a}_k \sim \pi_\theta(\cdot \mid o_k)$ and compute $\log \pi_\theta(\mathbf{a}_k \mid o_k)$, which is critical for conservative regularization toward the behavior policy π_β during offline and online fine-tuning.

IV. Method

A. Setting

We learn a visuomotor policy $\pi_\theta(\mathbf{a} \mid o)$ and deploy it in a real robotic environment that provides observations o_t and rewards r_t. The robot executes *action chunks* of horizon H: at each decision step k, the policy samples a sequence $\mathbf{a}_k = (a_{k,0}, \ldots, a_{k,H-1})$, which is executed open-loop (or with low-level stabilization). Observations o_t include camera images, robot proprioception, and a small number of actions from the previous chunk that have not yet been executed. This enables real-time chunking (RTC) [6] and provides the policy with context about its recent action history. We found RTC to be helpful not only because of improved inference time but also in terms of improving the performance thanks to the context about previous actions.

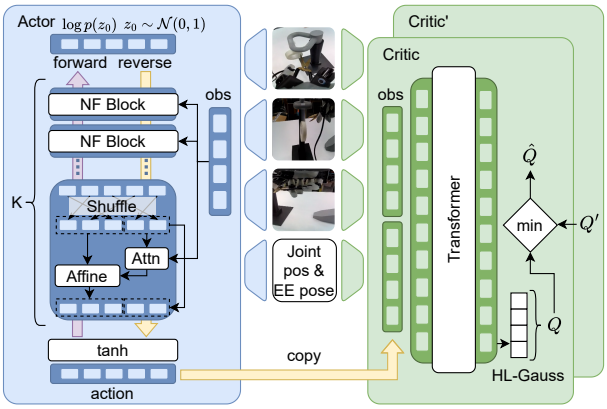

Fig. 2. Actor–critic architecture of SERNF. The actor is a conditional normalizing flow (NF) that models an invertible mapping between action chunks and a base Gaussian distribution. It consists of K stacked NF blocks, each applying affine transformations to a subset of tokens conditioning on observations. In the forward process, sampled actions are mapped to the base distribution, yielding tractable log-likelihoods used for behavior cloning supervision (8). In the reverse direction, latent samples drawn from the Gaussian are transformed into actions through fully differentiable operations, enabling gradient-based policy optimization using the critic. The critic is a transformer-based Q-network that predicts action-chunk values conditioned on observations. Q-values are parameterized using an HL-Gaussian distribution to improve regression stability. To mitigate overestimation bias, the final Q estimate is computed by taking the minimum over multiple critic predictions.

B. Normalizing-flow policy

We parameterize the policy $\pi_\theta(\mathbf{a} \mid o)$ as a conditional normalizing flow, which enables efficient sampling, end-to-end differentiation and exact log-likelihood evaluation. This property is critical for seamless integration into RL pipeline and conservative regularization toward an initial policy during offline and online fine-tuning.

We implement each normalizing-flow transformation $f_k(\cdot; c)$ using a transformer-based architecture [26]. Inspired by NinA [50], we adopt coupling layers similar to RealNVP [13]. During the forward pass, ground-truth actions (normalized to $[-1, 1]$ using dataset statistics) are first perturbed with small Gaussian noise $\mathcal{N}(0, \sigma^2_{\text{noise}})$, which has been shown to be crucial for stable action modeling [18, 50]. To enable bounded action space support, we then apply an element-wise $\tanh^{-1}$ transformation as the next step.

Each flow step operates on latent variables $z_k = [\mathbf{z}_1, \ldots, \mathbf{z}_H]$, where each element corresponds to one action in the chunk. The variables are randomly partitioned into two equally sized subsets x_{k_1} and x_{k_2}. Conditioned on c, x_{k_1} is processed by a nonlinear transformation $g_k(x_{k_1}, c)$ implemented via stacked self-attention over x_{k_1} and cross-attention between x_{k_1} and c. The outputs s and b parameterize an affine transformation applied to x_{k_2}:

$$y_2 = \exp(\tanh(s)) \odot x_{k_2} + b,$$

while $y_1 = x_{k_1}$. Concatenating y_1 and y_2 produces z_{k-1}. This design ensures invertibility and enables efficient computation of the Jacobian determinant. The final latent variable z_0 is used to compute $\log p_0(z)$, allowing optimization of the exact log-likelihood in (6). The inverse pass applies the same transformations in reverse order. A schematic overview is

Algorithm 1: Normalizing-flow policy over action chunks

Input: Observation o, base distribution $p_0(z)$, invertible flow $f_\theta(\cdot;\, o)$
$c \leftarrow \mathrm{Enc}(o)$
Forward pass (likelihood evaluation):
$z_0 \leftarrow f_\theta(a;\, c)$ ▷ `actions → latent`
$\log \pi_\theta(a \mid o) \leftarrow \log p_0(z_0) + \log |\det J_{f_\theta}(a;\, c)|$
Inverse pass (action generation):
$z_0 \sim p_0(z)$
$a \leftarrow f_\theta^{-1}(z_0;\, c)$ ▷ `latent → actions`
$\log \pi_\theta(a \mid o) \leftarrow \log p_0(z_0) - \log \left|\det J_{f_\theta^{-1}}(z_0;\, c)\right|$

Algorithm 2: Action-chunk selection with critic evaluation

Input: Observation o_k, policy π_θ, critic Q_ϕ, number of samples N_π
Output: Selected action chunk a_k
▷ `Sample and score candidate chunks`
for $i = 1$ *to* N_π **do**
  $a^{(i)} \sim \pi_\theta(\cdot \mid o_k)$
  $q^{(i)} \leftarrow \min_j Q_\phi^{(j)}(o_k, a^{(i)})$
$a_k \leftarrow \arg\max_i q^{(i)}$
return a_k

Algorithm 3: SERNF training pipeline

Input: Demo data $\mathcal{D}_{\text{demo}}$, offline data $\mathcal{D}$
Output: Fine-tuned policy π_θ and critic Q_ϕ
▷ **`Stage I: Imitation learning`**
foreach *minibatch* $(o, a) \sim \mathcal{D}_{demo}$ **do**
  $\mathcal{L}_{\text{IL}} \leftarrow -\mathbb{E}[\log \pi_\theta(a \mid o)]$
  $\theta \leftarrow \theta - \eta_\pi \nabla_\theta \mathcal{L}_{\text{IL}}$
▷ **`Stage II: Offline critic warm-up`**
repeat
  $\hat{a}_{k+1} \sim \pi_{\theta_0}(\cdot \mid o_{k+1})$
  $y \leftarrow \sum_{t=0}^{H-1} \gamma^t r_{k,t} + \gamma^H (1-d) Q_{\bar{\phi}}(o_{k+1}, \hat{a}_{k+1})$
  $\mathcal{L}_Q \leftarrow \mathrm{CrossEntropy}(Q_\phi(o_k, a_k), y)$
  $\phi \leftarrow \phi - \eta_Q \nabla_\phi \mathcal{L}_Q$
until *convergence on* $\mathcal{D}$
▷ **`Stage III: Full offline RL`**
repeat
  Update critic as above
  $a_{\pi_\theta} \leftarrow$ Alg. 2
  Compute $\log \pi_\theta(a_d \mid o)$ using Alg. 1
  $\theta \leftarrow \theta - \eta_\pi(\mathbb{E}[Q_\phi(o, a_{\pi_\theta})] - \lambda\, \mathbb{E}[\log \pi_\theta(a_d \mid o)])$
until *convergence on* $\mathcal{D}$
▷ **`Stage IV: Online fine-tuning`**
while *interaction budget remains* **do**
  Collect rollouts using Alg. 2
  Add data to replay buffer
  Update actor and critic with mixed (with ratio ρ) offline and online data

shown in Fig. 2, and pseudocode is provided in Alg. 1. When using $\mathcal{N}(0, I)$ as $p_0(z)$ for training we can sample from $\mathcal{N}(0, \sigma^2_{sample}), \sigma_{sample} \leq 1$ during inference to obtain more likely actions which can help to improve performance [29].

C. Action-chunked critic

Following Li et al. [28], we learn a critic that matches the chunked control interface:

$$Q_\phi(o_k, \mathbf{a}_k) \approx \mathbb{E}\Big[\sum_{i=0}^{H-1} \gamma^i r_{k,i} + \gamma^H V_\phi(o_{k+1})\Big], \tag{7}$$

where o_{k+1} denotes the observation at the next decision boundary.

Inspired by recent findings on value learning stability [15, 48], we parameterize the critic as a categorical distribution trained using a cross-entropy objective. We employ HL-Gauss [22] as the distributional parameterization, which improves stability and scalability in practice.

D. Algorithm

Our approach, **S**ample-**E**fficient **R**einforcement learning with **N**ormalizing **F**low (**SERNF**), consists of four training stages (Alg. 3).

a) I. Policy Initialization: We first train the normalizing-flow policy using imitation learning on a filtered dataset of successful trajectories $\mathcal{D}_{\text{demo}}$, a standard and effective baseline for robot policy learning:

$$\mathcal{L}_{\text{IL}} = \mathbb{E}_{(o_t, \mathbf{a}^*_{t:t+H}) \sim \mathcal{D}_{\text{demo}}}\Big[-\log \pi_\theta(\mathbf{a}^*_{t:t+H} \mid o_t)\Big] \tag{8}$$

The resulting policy is denoted π_{θ_0}. At inference time, we can sample multiple candidate action chunks from π_{θ_0}, evaluate their likelihoods exactly, and select the most likely one. This strategy, which is difficult to implement for diffusion-based policies, has been shown to improve performance [29] and is also effective in our experiments.

b) II. Offline RL critic warm-up: Although imitation learning yields strong initial policies, it typically relies on high-quality demonstrations and cannot fully exploit suboptimal data. Reinforcement learning, by contrast, can leverage broader state–action coverage and optimize temporal efficiency through discounting. Because the critic is randomly initialized, we first warm up the critic by learning Q-values under π_{θ_0} before updating the policy. The target value is

$$\sum_{t=0}^{H-1} \gamma^t r_{k,t} + \gamma^H (1-d) Q_{\bar{\phi}}(o_{k+1}, \hat{a}_{k+1}), \quad \hat{a}_{k+1} \sim \pi_{\theta_0}.$$

This warm-started critic can already be used to improve online behavior by sampling multiple actions from π_{θ_0} and selecting the one with the highest estimated value, see Alg. 2.

c) III. Full offline RL: We then fine-tune π_θ using a full offline actor–critic pipeline. Directly maximizing $Q(o, a)$ can lead to exploitation of value estimation errors [27], so we regularize the actor toward the data distribution, following TD3+BC [17]:

$$\max_{\theta} \mathbb{E}[Q(o, a)] - \lambda \mathcal{L}_{\text{IL}}.$$

This simple formulation with appropriate design choices has been shown to outperform more complex alternatives [46, 47, 18] and adapts naturally to offline-to-online fine-tuning by reducing λ during online interaction [4].

d) IV. Online RL fine-tuning: Finally, we perform online fine-tuning on the real robot. Following Ball et al. [3], we mix offline data with collected replay data at a fixed ratio ρ and apply imitation regularization only to the offline data. All other training components remain unchanged.

E. Implementation Details

Most hyperparameters are selected using simulation experiments (subsection A) and transferred to real-world tasks without modification, unless otherwise noted.

a) Policy architecture and normalizing flow: The policy is parameterized as a conditional normalizing flow with 16 coupling attention-based blocks, each with hidden dimensionality 256. We use an action chunk length of $H = 10$ across all experiments. During imitation learning, ground-truth actions are perturbed with Gaussian noise $\mathcal{N}(0, \sigma^2_{\text{noise}})$, with $\sigma_{\text{noise}} = 0.05$ for simulation and $\sigma_{\text{noise}} = 0.01$ for real-world tasks; this magnitude is consistent with prior work [50]. At inference time, we sample latent variables with standard deviation $\sigma_{\text{sample}} = 0.7$, which provides a good trade-off between action diversity and distribution support.

b) Reinforcement learning and critic configuration: For offline and online RL, we use two critics, following standard practice. The imitation regularization coefficient is fixed to $\lambda = 0.1$ across all tasks. Distributional critic hyperparameters (including HL-Gauss configuration) are taken directly from Tarasov et al. [48]. Additional training parameters can be found in Appendix G2.

V. Results

Appendix sections A-C provide detailed information on the experimental setups we use, i.e. scissors extraction with tape cutting (*imitation learning* $\rightarrow$ *offline RL* $\rightarrow$ *online RL*) and palm-down in-hand cube rotation (*RL in simulation* $\rightarrow$ *real-world policy distillation* $\rightarrow$ *online RL*).

A. Scissors Retrieval & Tape Cutting

Fig. 3 summarizes the performance evolution on the scissors task as additional demonstrations and online interaction data are incorporated. The initialized imitation policy presents a 50% success rate in the grasping phase but exhibits very limited cutting performance. Offline RL substantially increases the grasp success rate and exhibits limited pose adjustment behavior. However, it acquired counterproductive backward motion during cutting, preventing successful cuts. As the iterative online training progresses, the policy learns to actively adjust the scissors' pose and execute effective cutting motions. Once the robot successfully picks up the scissors, it is generally able to complete the cut, leading to a steadily increasing cut success rate that reaches 70% at the end of training (see Fig. 3).

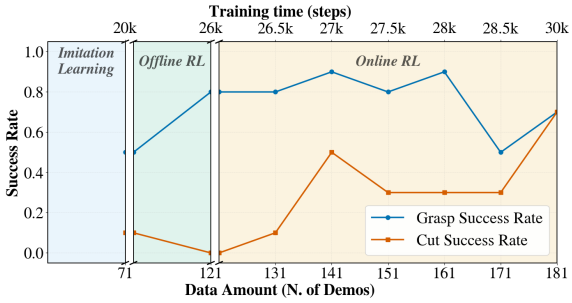

Fig. 3. Performance evolution on the scissors task with respect to the number of collected demonstrations. We collect 121 teleoperated demonstrations, of which 71 are successful. The policy is first initialized via imitation learning on successful trajectories, followed by offline RL fine-tuning using the full dataset. Subsequently, we iteratively collect 10 online rollouts and fine-tune the policy in an online RL setting. We report the success rates of grasping the scissors (blue) and cutting the tape (orange), measured as the number of successful executions out of 10 rollouts from a fixed test configuration. Offline RL substantially improves the grasp success rate, while online RL gradually increases the overall task success.

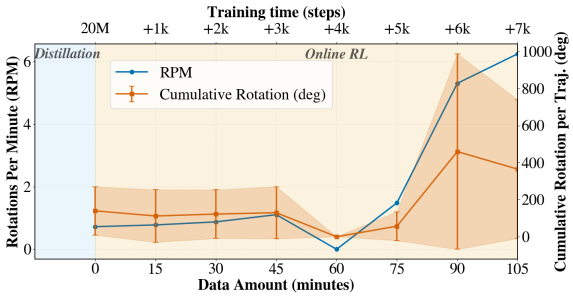

Fig. 4. Performance evolution of the cube rotation task with regards to real-world data collected during online fine-tuning of SERNF. We report rotations per minute (RPM, blue) and cumulative rotation per trajectory (orange). The policy is initialized via teacher–student distillation in massive simulation, trained for 20M steps, and subsequently deployed and fine-tuned online in the real world using SERNF. For every 1,000 gradient updates, approximately 15 minutes of real-world interaction data are collected. The first 3,000 updates are used for critic warm-up, after which actor fine-tuning begins. A temporary performance drop is observed at the onset of actor fine-tuning due to increased policy exploration, followed by significant performance improvements as training progresses.

Notably, we observe a slight decrease in the grasp success rate in the later stage of online RL. We attribute this behavior to two factors. First, as more rollouts are collected in which the robot smoothly grasps the scissors and repeatedly moves back and forth until the timeout (2 minutes), the relative proportion of grasping behavior in the replay buffer decreases, leading to increased errors in grasp execution. Second, by intentionally avoiding data collection from the test configurations, the model has fewer opportunities to correct grasping errors specific to those positions, which reduces grasp performance during evaluation. Despite this trade-off, online RL significantly improves

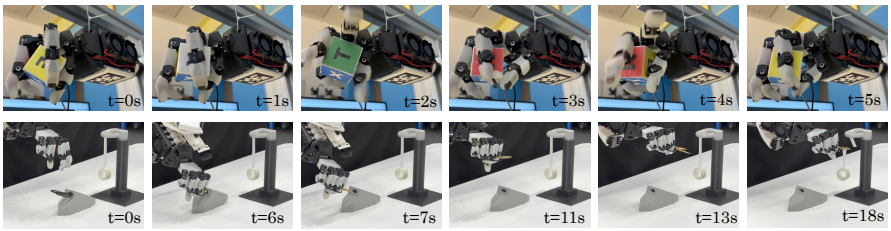

Fig. 5. Qualitative rollouts of SERNF on real hardware. Top: successful scissors retrieval and tape cutting task. Bottom: in-hand cube rotation task.

overall task success by enabling more robust and consistent cutting behavior.

1) Baselines Comparison: Table I shows that the normalizing-flow imitation policy used in SERNF performs on par with flow-matching and ACT baselines when trained with imitation learning, indicating comparable expressiveness of the architecture for dexterous manipulation.

Moreover, simply adding more data does not resolve the problem: augmenting imitation learning with additional teleoperated demonstrations yields only marginal improvements and fails to enable cutting. Naively incorporating trajectories collected during online interaction further degrades performance, likely due to a distribution mismatch between expert teleoperation and on-policy data. In contrast, SERNF effectively leverages both offline and online data through reinforcement learning, achieving substantially higher grasping performance and enabling successful cutting only after online fine-tuning.

TABLE I
SCISSORS TASK SUCCESS RATES FOR SERNF AND BASELINES.
$+$ SUPERSCRIPT DENOTES SUCCESSFUL TRAJECTORIES.

Policy	**N trajectories**	**Grasping**	**Cutting**
ACT IL	71^+	0.5	0.0
Flow-Matching IL	71^+	0.5	0.1
NF IL with original data	71^+	0.5	0.1
NF IL + online RL data	83^+	0.1	0.0
NF IL + more teleoperation	97^+	0.6	0.0
SERNF (Offline only)	121 (71^+)	**0.8**	0.0
SERNF (Full)	181 (83^+)	0.7	**0.7**

B. In-hand Cube Rotation

Figure 4 shows the performance evolution of SERNF during real-world online reinforcement learning fine-tuning. During the first 3,000 gradient updates after the teacher–student distillation initialization, the critic is warmed up while the actor remains frozen, resulting in relatively stable but limited performance. Early online performance achieves modest rotation speeds, with most trajectories failing before completing a full rotation.

Once actor fine-tuning begins, a brief performance drop is observed due to increased exploration. Subsequently, performance improves rapidly as the policy learns to both retain the cube and execute faster rotations. By 5k steps after distillation, the policy reaches 1.49 RPM, and finally peaks at 6.25 RPM over 58 trajectories with a training time of 7k steps after distillation and 105 minutes of real-world data. At this stage, the policy consistently achieves continuous rotations, with an average cumulative rotation of 1.01 full turns per trajectory (see Fig. 5, bottom).

These results demonstrate that real-world RL fine-tuning substantially refines the distilled policy, transitioning from fragile, short-horizon behavior to stable grasp maintenance with rapid, continuous in-hand manipulation.

VI. CONCLUSION, LIMITATIONS, AND FUTURE WORK

a) Conclusion: We presented SERNF, a framework for sample-efficient real-world visuomotor policy fine-tuning that combines expressive normalizing-flow policies with action-chunked critics. By aligning policy representation, likelihood-based regularization, and value estimation with temporally extended action execution, SERNF enables stable off-policy reinforcement learning under limited real-world interaction budgets. Crucially, aside from pretrained visual encoders, SERNF requires only a small amount of task-specific real-world data, yet succeeds on two long-horizon dexterous tasks that are difficult to solve with IL or RL and sim-to-real alone. Our results in these challenging dexterous manipulation tasks demonstrate improved update stability, credit assignment, and learning efficiency. We believe SERNF provides a practical foundation for scalable and reliable real-world fine-tuning of high-capacity visuomotor policies.

b) Limitations: Despite strong empirical results, several limitations remain. First, *real-world scalability* is constrained by the need for on-robot interaction and careful safety measures during fine-tuning. We focus on two dexterous manipulation tasks, and scaling to many tasks or continual multi-task fine-tuning on a single robot remains an open challenge. Second, *reward design* plays a critical role in practice. Our experiments rely on sparse, manually annotated rewards, which limit scalability and may hinder learning when rewards are poorly aligned. Finally, *computation and implementation complexity* is higher than for simple Gaussian policies: normalizing flows introduce additional architectural and training overhead, increasing engineering and computational costs.

c) Future work: Several directions are promising. Automating reward labeling with VLMs could substantially reduce manual effort and enable scaling to larger datasets and task collections. Applying SERNF to large-scale VLA models is another natural extension, where multimodal action distributions and chunked control are already standard [23]. In addition, evaluating SERNF on standard offline and offline-to-online RL benchmarks, such as D4RL [16] or OGBench [40], would help characterize its behavior across a broader range of environments.

References

[1] Dmitriy Akimov, Vladislav Kurenkov, Alexander Nikulin, Denis Tarasov, and Sergey Kolesnikov. Let offline rl flow: Training conservative agents in the latent space of normalizing flows. *arXiv preprint arXiv:2211.11096*, 2022.

[2] Anonymous. Policyflow: Policy optimization with continuous normalizing flow in reinforcement learning. In *Submitted to The Fourteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YETCQLcKtn. under review.

[3] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[4] Alex Beeson and Giovanni Montana. Improving td3-bc: Relaxed policy constraint for offline learning and stable online fine-tuning. *arXiv preprint arXiv:2211.11802*, 2022.

[5] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

[6] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*.

[7] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[8] Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[9] Chen-Hao Chao, Chien Feng, Wei-Fang Sun, Cheng-Kuang Lee, Simon See, and Chun-Yi Lee. Maximum entropy reinforcement learning via energy-based normalizing flow. *Advances in Neural Information Processing Systems*, 37:56136–56165, 2024.

[10] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *CoRR*, abs/2303.04137, 2023. doi: 10.48550/ARXIV.2303.04137. URL https://doi.org/10.48550/arXiv.2303.04137.

[11] Clemens C. Christoph, Maximilian Eberlein, Filippos Katsimalis, Arturo Roberti, Aristotelis Sympetheros, Michel R. Vogt, Davide Liconti, Chenyu Yang, Barnabas Gavin Cangan, Ronan J. Hinchet, and Robert K. Katzschmann. Orca: An open-source, reliable, cost-effective, anthropomorphic robotic hand for uninterrupted dexterous task learning, 2025. URL https://arxiv.org/abs/2504.04259.

[12] Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15617–15625. IEEE, 2025.

[13] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[14] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pages 9104–9149. PMLR, 2023.

[15] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

[16] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[17] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[18] Raj Ghugare and Benjamin Eysenbach. Normalizing flows are capable models for RL. *CoRR*, abs/2505.23527, 2025. doi: 10.48550/ARXIV.2505.23527. URL https://doi.org/10.48550/arXiv.2505.23527.

[19] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[21] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

[22] Ehsan Imani and Martha White. Improving regression performance with distributional losses. In *International conference on machine learning*, pages 2157–2166. PMLR, 2018.

[23] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ash-

win Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, et al. $\pi^*_{0.6}$: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[24] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

[25] Shahbaz Abdul Khader, Hang Yin, Pietro Falco, and Danica Kragic. Learning stable normalizing-flow control for robotic manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1644–1650. IEEE, 2021.

[26] Alexander Kolesnikov, André Susano Pinto, and Michael Tschannen. Jet: A modern transformer-based normalizing flow. *arXiv preprint arXiv:2412.15129*, 2024.

[27] 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.

[28] Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. *CoRR*, abs/2507.07969, 2025. doi: 10.48550/ARXIV.2507.07969. URL https://doi.org/10.48550/arXiv.2507.07969.

[29] Simon Kristoffersson Lind, Jialong Li, Maj Stenmark, and Volker Krüger. Normalizing flows are capable visuomotor policy learning models. *CoRR*, abs/2509.21073, 2025. doi: 10.48550/ARXIV.2509.21073. URL https://doi.org/10.48550/arXiv.2509.21073.

[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[31] 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. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, pages 16961–16969. IEEE, 2024. doi: 10.1109/ICRA57147.2024.10610040. URL https://doi.org/10.1109/ICRA57147.2024.10610040.

[32] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[33] Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pages 430–444. PMLR, 2020.

[34] David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. *CoRR*, abs/2507.21053, 2025. doi: 10.48550/ARXIV.2507.21053. URL https://doi.org/10.48550/arXiv.2507.21053.

[35] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science robotics*, 7(62):eabk2822, 2022.

[36] Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, Milad Rakhsha, Alain Denzler, Eric Heiden, Ales Borovicka, Ossama Ahmed, Iretiayo Akinola, Abrar Anwar, Mark T. Carlson, Ji Yuan Feng, Animesh Garg, Renato Gasoto, Lionel Gulich, Yijie Guo, M. Gussert, Alex Hansen, Mihir Kulkarni, Chenran Li, Wei Liu, Viktor Makoviychuk, Grzegorz Malczyk, Hammad Mazhar, Masoud Moghani, Adithyavairavan Murali, Michael Noseworthy, Alexander Poddubny, Nathan Ratliff, Welf Rehberg, Clemens Schwarke, Ritvik Singh, James Latham Smith, Bingjie Tang, Ruchik Thaker, Matthew Trepte, Karl Van Wyk, Fangzhou Yu, Alex Millane, Vikram Ramasamy, Remo Steiner, Sangeeta Subramanian, Clemens Volk, CY Chen, Neel Jawale, Ashwin Varghese Kuruttukulam, Michael A. Lin, Ajay Mandlekar, Karsten Patzwaldt, John Welsh, Huihua Zhao, Fatima Anes, Jean-Francois Lafleche, Nicolas Moënne-Loccoz, Soowan Park, Rob Stepinski, Dirk Van Gelder, Chris Amevor, Jan Carius, Jumyung Chang, Anka He Chen, Pablo de Heras Ciechomski, Gilles Daviet, Mohammad Mohajerani, Julia von Muralt, Viktor Reutskyy, Michael Sauter, Simon Schirm, Eric L. Shi, Pierre Terdiman, Kenny Vilella, Tobias Widmer, Gordon Yeoman, Tiffany Chen, Sergey Grizan, Cathy Li, Lotus Li, Connor Smith, Rafael Wiltz, Kostas Alexis, Yan Chang, David Chu, Linxi "Jim" Fan, Farbod Farshidian, Ankur Handa, Spencer Huang, Marco Hutter, Yashraj Narang, Soha Pouya, Shiwei Sheng, Yuke Zhu, Miles Macklin, Adam Moravanszky, Philipp Reist, Yunrong Guo, David Hoeller, and Gavriel State. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning, 2025. URL https://arxiv.org/abs/2511.04831.

[37] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[38] Mitsuhiko Nakamoto, Simon Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural*

Information Processing Systems, 36:62244–62269, 2023.

[39] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[40] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. *arXiv preprint arXiv:2410.20092*, 2024.

[41] Allen Z. Ren, Justin Lidard, Anthony Simeonov, Lars Lien Ankile, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=mEpqHvbD2h.

[42] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[43] Clemens Schwarke, Mayank Mittal, Nikita Rudin, David Hoeller, and Marco Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.

[44] Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube. *arXiv preprint arXiv:2202.10448*, 2022.

[45] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

[46] Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36:11592–11620, 2023.

[47] Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. Corl: Research-oriented deep offline reinforcement learning library. *Advances in Neural Information Processing Systems*, 36:30997–31020, 2023.

[48] Denis Tarasov, Kirill Brilliantov, and Dmitrii Kharlapenko. Is value functions estimation with classification plug-and-play for offline reinforcement learning? *arXiv preprint arXiv:2406.06309*, 2024.

[49] Denis Tarasov, Anja Surina, and Caglar Gulcehre. The role of deep learning regularizations on actors in offline rl. *arXiv preprint arXiv:2409.07606*, 2024.

[50] Denis Tarasov, Alexander Nikulin, Ilya Zisman, Albina Klepach, Nikita Lyubaykin, Andrei Polubarov, Alexander Derevyagin, and Vladislav Kurenkov. Nina: Normalizing flows in action. training vla models with normalizing flows. *arXiv preprint arXiv:2508.16845*, 2025.

[51] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy. *CoRR*, abs/2403.03954, 2024. doi: 10.48550/ARXIV.2403.03954. URL https://doi.org/10.48550/arXiv.2403.03954.

[52] Tonghe Zhang, Chao Yu, Sichang Su, and Yu Wang. Reinflow: Fine-tuning flow matching policy with online reinforcement learning. *arXiv preprint arXiv:2505.22094*, 2025.

[53] Tony Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[54] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[55] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

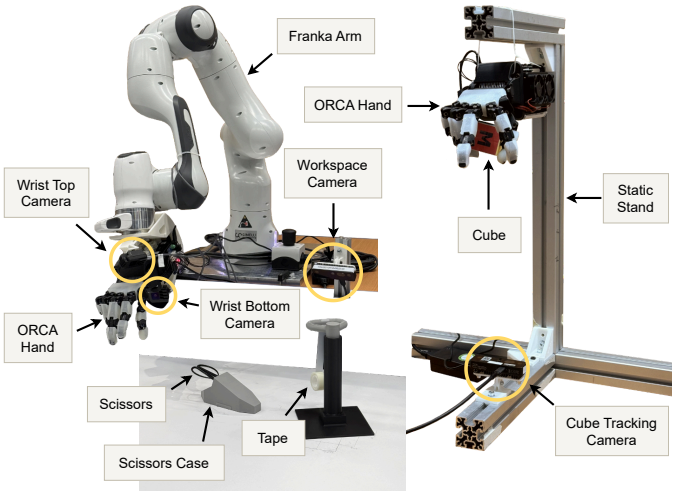

Fig. 6. Real-world experimental setup. Left: scissors retrieval and tape cutting with a 7-DoF Franka Panda arm and ORCA hand, using two wrist-mounted RGB cameras and one external workspace camera. Right: in-hand cube reorientation with the ORCA hand, performing continuous palm-down rotations, using single-camera vision-based pose estimation.

TABLE II
SUCCESS RATES (OVER 4 SEEDS) ON ROBOMIMIC MH DATASETS.

Environment	Imitation Learning	Offline RL
Lift	0.79 ± 0.00	0.91 ± 0.04
Can	0.96 ± 0.00	0.96 ± 0.02
Square	0.61 ± 0.04	0.68 ± 0.05

APPENDIX

A. Simulation Tasks

We first validate our approach in simulation to identify reasonable hyperparameters before deploying on real hardware. We use the RoboMimic benchmark [32] and consider the Lift, Can, and Square manipulation tasks. For all tasks, we use the Mixed Human (MH) datasets, which contain a mixture of successful and suboptimal demonstrations and are therefore more challenging than Expert Human datasets. Policies operate on image-based observations with resolution 84×84, encoded using a ResNet-18 backbone [20]. We evaluate performance using success rate over 100 rollouts.

Table II reports success rates (mean $\pm$ standard deviation over 4 random seeds) for imitation learning (IL) and offline RL fine-tuning. All policy architecture and offline RL hyperparameters are tuned exclusively on the Square task and then reused without modification for Lift and Can.

Offline RL fine-tuning consistently improves performance when the initial policy does not already solve the task. On the Square task, offline RL yields an absolute improvement of approximately 7%, while the Lift task shows an even larger gain (from 0.79 to 0.91). In contrast, performance on the Can task remains unchanged, as the imitation policy already achieves near-perfect success.

Due to constrained computational resources, we limit training to approximately 270k gradient updates in total, whereas prior works in offline RL commonly use 1–2 million updates. Despite this reduced budget, offline RL already provides clear gains, and we observe that performance continues to improve with additional training. The primary goal of this simulation experiment was to identify a set of stable hyperparameters and verify the benefits of reinforcement learning over pure imitation.

Notably, the selected hyperparameters transfer well across tasks, including for the offline RL stage, which is often highly sensitive to tuning choices [14]. Most of these hyperparameters are reused unchanged in our real-world experiments.

B. Scissors Retrieval & Tape Cutting

This task features highly multimodal action distributions and is hard to simulate due to complex interactions. We test whether SERNF can successfully fine-tune an imitation-initialized policy under limited real-world data.

1) Task Definition and Experimental Setup: The robot is required to grasp a pair of scissors from a case and cut a tape suspended from a support. All experiments are conducted on a 7-DoF Franka Emika Panda arm equipped with a dexterous ORCA hand [11] Fig. 6, left. The arm is controlled in end-effector pose space, parameterized by position and quaternion orientation. The overall action space comprises 24 degrees of freedom. This task requires a high degree of precision, as the scissor finger holes are only minimally larger than the robot's fingers. Moreover, the cutting motion is performed while the scissors are manipulated in mid-air, and even small errors can result in dropping them. The experimental setup includes three RGB cameras: two egocentric cameras mounted on the hand and one external camera mounted on the table. Additional details regarding real-time inference and the experimental setup are provided in Appendix E1.

2) Policy Initialization: The policy for this task is initialized using imitation learning from 121 collected demonstrations. The dataset was collected by teleoperating the robotic system with motion capture gloves. Among these, only 71 demonstrations are fully successful and exhibit high-quality execution of both the grasping and cutting motions, highlighting the high precision required by the task. These successful demonstrations are used to train the initial imitation learning policy, while the remaining unsuccessful demonstrations are incorporated later during the offline RL fine-tuning stage. Additional details on the teleoperation and data collection can be found in Appendix E2.

3) Implementation Details: To process the image stream from the 3 views we use a pre-trained frozen DINOv2[39] image encoder. The images are processed independently with the same encoder, and then the image features are concatenated. After the initial policy is trained via imitation learning, we augment the dataset with all available data points, including both successful and unsuccessful demonstrations. The reward is defined as sparse: a reward of 1 is assigned for successfully retrieving the scissors from the case, and an additional reward of 1 is assigned for successfully cutting the tape. Reward labels are manually assigned by a human annotator. After labeling the dataset, we perform an offline critic warm-up phase for 5,000 gradient steps following Section IV-D0b,

followed by 1,000 steps of full offline reinforcement learning as described in Section IV-D0c. We then perform 5 iterations of online reinforcement learning fine-tuning (Section IV-D0d), where, at each iteration, 10 policy rollouts are collected and labeled using the defined reward at randomly selected locations distinct from the test configurations. For each online RL iteration, the policy is trained for 500 steps.

4) Metrics and Evaluation: Performance is quantified using success rates for the two sequential stages of the task: (i) grasping the scissors from the case and (ii) cutting the tape. Since these subtasks are executed in sequence, successful tape cutting requires successful completion of the grasping stage. Success rates are computed over 10 runs, with each run conducted at a different test configuration of the scissors case and tape holder. The test configurations are illustrated in Appendix E4. The episode terminates when the tape is successfully cut, when the scissors are dropped, when the robot arm reaches unsafe or infeasible configurations, or when a maximum time limit of 120 s is reached.

C. In-hand Cube Rotation

This task represents a common sim-to-real pipeline in dexterous manipulation: it features highly dexterous movements and is impossible to teleoperate reliably due to latency, retargeting errors and lack of haptic feedback to the operator. We test if SERNF can successfully bridge the sim-to-real gap with limited real-world fine-tuning.

1) Task Definition and Experimental Setup: The second task is an *in-hand cube orientation* task, where the ORCA hand palm faces downward and the cube is manipulated without any external support (Fig. 6, right). The states are the hand joints and the cube pose, while the actions correspond to desired joint commands. The total action space dimension is 17 DoF. This task fully exploits the dexterity of the ORCA hand and represents a highly dynamic and precise scenario, as even minimal control errors can lead to catastrophic failure, resulting in the cube being dropped. For real-world deployment, an RGB camera observes the cube from below. Cube pose estimation is obtained by detecting its corners using a custom neural network initialized from a pretrained Mask R-CNN model, followed by a PnP-based pose reconstruction similar to the pipeline in [19]. Additional details on cube pose estimation and real-time inference are provided in the Appendix F2.

2) Policy Initialization: The starting policy is first trained via PPO in simulation with domain randomization (DR) for sim-to-real generalization. Training is performed in a parallelized IsaacLab simulation environment [36]. The teacher policy is then distilled to our initial policy. The details of the distillation and the simulation reward definition are provided in Appendix F5.

3) Implementation Details: After the initial policy is trained via distillation, we directly perform online reinforcement learning refinement (Section IV-D0d). This choice is motivated by the large sim-to-real gap in the simulated data, which makes the inclusion of real-world data necessary. This new setting highlights the flexibility of the proposed method, as the training stages can be adapted or modified while still improving policy performance.

The reward is again defined as sparse: a reward of 1 is assigned for every 90° rotation of the cube. The rewards are assigned manually, but can be easily automated, for example by using the trained cube pose estimator.

We perform 7 iterations of online reinforcement learning fine-tuning. In each iteration, the policy is executed for 15 minutes, and each successful 90° rotation is labeled with a positive reward. When the cube is dropped, the policy execution is restarted. Each online fine-tuning iteration is trained for 1000 gradient steps.

4) Metrics and Evaluation: Performance is measured as (i) the average cumulative cube rotation achieved before the cube is dropped and (ii) the average number of rotations per minute during the evaluation period. Testing consists of 15 minutes of consecutive runs. Each run terminates when the cube is dropped.

D. Baselines

With the baseline definition, we want to answer two key questions:

1) Can the SERNF architecture effectively learn dexterous robotic manipulation policies?

We compare SERNF against state-of-the-art robotic manipulation policies based on *flow-matching* architecture. Flow matching is commonly adopted as a faster-inference alternative to diffusion models while retaining the ability to model multimodal action distributions. This class of architectures is widely used in the action heads of modern vision–language–action (VLA) models [7, 5, 24]. Note, that RL policy finetuning with flow-matching is not as straightforward as for NFs. We also use ACT [53] as an additional baseline, that proved to be effective in highly dexterous and precise tasks.

2) Are the robot trajectories leveraged effectively?

Both the offline and online components of the algorithm either reuse previously collected data or acquire new data, and we aim to evaluate whether this data is being leveraged effectively compared to naive alternatives. To this end, we compare against two additional baselines: (i) an imitation learning policy trained on all successful trajectories collected throughout training, including both the original teleoperated demonstrations and those obtained during online RL refinement; and (ii) an imitation learning policy trained on the original expert demonstrations augmented with additional expert data to match the total number of demonstrations used by SERNF. In both cases, the SERNF architecture is trained using only the imitation learning loss (8), with the RL part disabled.

E. Scissor Retrieval & Tape Cutting Task

1) Experimental Setup Details: We mount the Orca Hand [11] on a Franka Emika Panda robot using a custom 3D-printed mount with a 60-degree tilt. Two OAK-1 Lite cameras are attached to the mount: one positioned underneath the hand to enable accurate finger placement, and the other facing left to provide visual guidance during scissor manipulation. The third camera is an OAK-D Lite that provides a front view. We

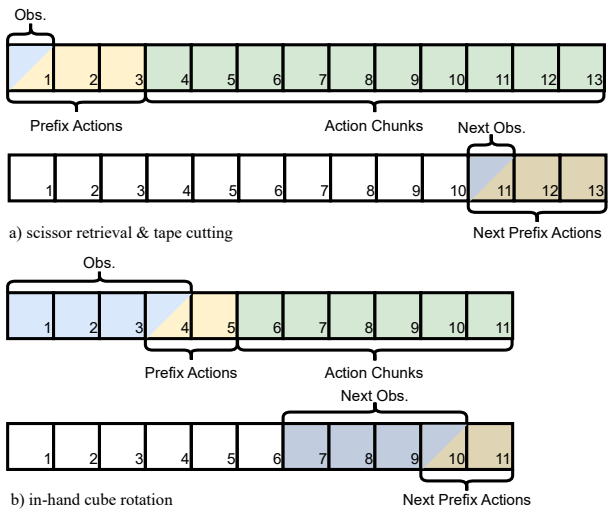

Fig. 7. Observation and real-time action-chunking structure for both tasks. At each decision step, the policy takes as input the current observation together with a sequence of prefix actions, which are previously predicted actions that are queued and ready to be executed, and predicts a temporally extended action chunk. The figure also illustrates how the next observation and next prefix actions, used for reinforcement learning updates, are defined based on the executed portion of the action chunk. (a) Scissor retrieval and tape cutting: the policy uses 1 step of observation and 3 steps of prefix actions, and predicts a subsequent chunk of 10 actions. (b) In-hand cube rotation: the policy uses 4 steps of position observations and 2 steps of prefix actions, and predicts a chunk of 6 future actions.

use a single pair of craft scissors with a length of 144 mm and a width of 65 mm, to cut 3M Scotch Magic Tape, which is suspended freely and allowed to hang naturally. The same pair of scissors is used throughout all experiments.

The Franka low-level controller runs on an Intel NUC at 1K Hz, receiving end-effector pose commands and executing pose impedance control. A separate PC equipped with Intel i9-11900K CPU and Nvidia RTX3090 runs the policy inference at 10Hz and a hand controller that commands the finger motors at 40 Hz. Communication between cameras, controllers, and the policy inference node is handled using ROS 2. The PC, NUC, and Franka controller are connected via Ethernet through a network switch. A ROS 1–ROS 2 bridge is used to interface the Franka controller with the policy inference node. All camera images are cropped and resized to 224 × 224 before being passed to the policy. The inference policy runs at 10 Hz.

2) Teleoperation and Data Collection: Expert demonstrations are collected using Rokoko Smart Gloves in combination with the Rokoko Coil Pro, which together provide finger motion capture and wrist pose estimation in 3D space. Human finger postures are retargeted to the robotic hand joint angles using an energy-based retargeting method [44].

All demonstrations attempt to complete the full cutting task from a randomized initial configuration. Operators teleoperate the system with direct visual observation of the scene and make their best efforts to execute the task smoothly. Failed trajectories are not intentionally collected; however, the overall demonstration success rate is approximately 59%, primarily due to scissor drops, activation of the Franka protective mode, or unintended tilting of the scissor stand. The demonstrations are collected by two operators, each exhibiting distinct teleoperation styles. The duration of the demonstrations ranges from 10 to 60 seconds. The recorded rosbag is sampled to a 10 Hz sequence with linear interpolation.

Rewards are labeled manually after demonstrations are collected. A sparse reward of 1 is assigned in the following cases:

- Successful scissor retrieval: the hand securely grasps the scissors and the scissors do not have contact with any objects other than the hand.
- Successful tape cutting: the tape is fully cut into two separate pieces.

3) Action Chunking and Real-Time Inference Implementation: As shown in Fig. 7, the policy uses 1 step of observation and 3 steps of prefix actions following the observation and predicts a subsequent chunk of 10 steps of actions. The observations contain the RGB images, the joint positions of the hand, and the end effector pose of Franka. The actions are represented as relative positions of the end effector within the current pose frame.

4) Test Configurations: We evaluate all policies under a fixed set of predefined test configurations. Specifically, we define 2 distinct test positions for the scissors and 5 distinct test positions for the tape holder. A total of 10 combinations can be seen in Fig. 8.

For each evaluation episode, a test configuration is sampled from the predefined set. The length of the tape is randomized across trials while ensuring that it remains freely hanging and always reachable and cuttable by the robot. During on-policy reinforcement learning rollouts, the initial configurations are randomized to increase the diversity of collected experience. All other experimental conditions, including the scissors, tape type, and hardware setup, are held constant across evaluations.

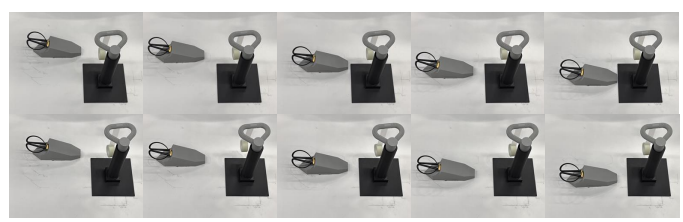

Fig. 8. Test configurations for the scissor retrieval and tape cutting task. The figure shows the 10 predefined test positions used for evaluation, covering variations in the relative placement of the scissors and the tape holder. The tape length is randomized across trials while ensuring that the tape remains freely hanging and can be successfully cut in all configurations.

F. In-hand Cube Rotation Task

1) Experimental Setup Details: We mount the Orca Hand [11] horizontally with an OAK-D Lite camera providing the view from the bottom. Both the training and the inference runs on a desktop with NVIDIA RTX 4090 GPU.

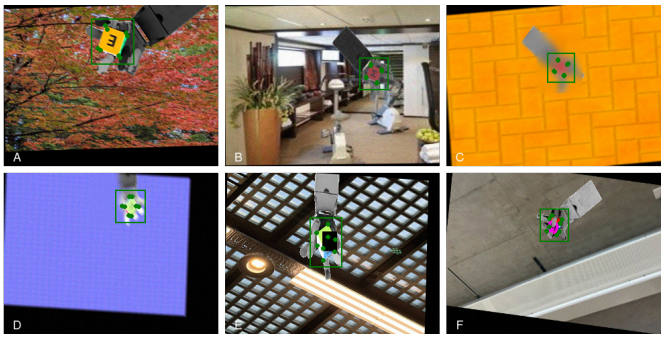

Fig. 9. Examples of synthetic training samples. Images are rendered during parallelized IsaacLab training with randomized camera poses, lighting conditions, and hand textures. Additional data augmentations are applied offline, including random cropping, color jitter, random rotation, motion blur, and synthetic occlusions. Backgrounds are randomly sampled from the Places365 dataset [55] (A, B), IsaacSim textures and materials (C, D), and custom in-house images of ceilings and walls (E, F).

2) Cube Pose Estimation: We use an OAK-D Lite camera mounted approximately 30 cm below the hand holding the cube. To estimate the cube pose, which is provided as input to the policy, we follow the same approach as in [19]. We also use the same cube, as its colors and surface features are beneficial for tracking.

A large synthetic dataset is generated using parallel IsaacLab environments. During policy rollouts, images are rendered from a bottom-up viewpoint across 200 parallel environments, and cube corner keypoints are automatically labeled by projecting the 3D cube corners into the image. Using an RTX 5090 GPU, we collected approximately 4 M images in 2 hours. During data generation, the camera pose, lighting conditions, and object textures are randomized.

We train a Mask R-CNN model, also adopted from [19], consisting of a bounding-box ROI head, a mask head, and a keypoint head on top of the ROI features. The total number of parameters is around 60M. Input images are resized to 240 × 320. To improve robustness to real-world deployment and severe occlusions, we apply heavy data augmentation, including random cropping (scale 0.7–1.0), small random rotations (±10°) with anisotropic stretching, Gaussian blur and noise, random occlusions, and background replacement. Examples of augmented samples are shown in Fig. 9.

The network is trained for 200,000 steps with a batch size of 16 and a learning rate of 1 x 10^{-8}. At inference time, the network predicts the cube keypoints, which are then used in a PnP-RANSAC pipeline to estimate the 6-DoF cube pose. The resulting pose is further smoothed using a low-pass Kalman filter for improved stability. The pipeline runs real-time at 16 Hz.

Despite these measures, performance remains challenging due to severe finger occlusions during in-hand rotation and suboptimal lighting conditions. To mitigate pose estimation errors, we apply extensive domain randomization to the cube pose during RL policy initialization, improving robustness to inaccuracies in real-world pose estimates.

3) Action Chunking and Real-Time Inference Implementation: As shown in Fig. 7, at each decision step, the policy receives as input a history of four timesteps of joint positions, cube position, and cube orientation (represented as a quaternion), as well as one timestep of the previous action and the previous joint position command. The goal command is also included as part of the observation. Conditioned on these observations and a sequence of two prefix actions, the policy predicts an action chunk consisting of six timesteps of target joint positions. The predicted joint positions are smoothed using an exponential moving average with a smoothing coefficient of 0.5.

4) Teacher Policy Training: We train the teacher policy fully in simulation using IsaacLab, in an environment that models single-axis spinning of a 45 mm cube with the Orca hand.

a) Simulation setup.: Each environment instance contains an Orca hand and a rigid cube object placed above a small kinematic platform. The platform provides support for the first 6 seconds in each episode. After the support phase, the platform moves out of the way, allowing the cube to fall freely. We run 8192 parallel environments. The simulation runs at 120 Hz with control frequency at 20 Hz and a finite-horizon episode length of 32 s.

b) Observations, Commands, and actions.: The teacher uses low-dimensional state observations (no images): normalized hand joint positions and relative joint velocities, cube position and orientation (quaternion), cube linear and angular velocities, the commanded target angular velocity, the previous action, the previous joint-position command, and a counting down of the support of the platform. No noise is added to the observations for teacher policy training. The action is a target joint-position command (scaled to joint limits) filtered with an exponential moving average (EMA) with coefficient $\alpha = 0.5$. During each episode, we sample a target cube rotation command around the vertical axis with angular velocity up to 1.5 rad/s. The command is held for 8–12 s before resampling.

c) Reward shaping.: The teacher is trained with a dense reward designed to stabilize grasping while tracking the commanded spin. Table III summarizes the reward terms and weights used in simulation. Episodes terminate on timeout or if the cube drops below a height threshold.

d) Domain randomization.: At startup, we scale the robot link friction uniformly from the range $[0.5, 1.3]$ and scale robot link masses by $[0.95, 1.05]$. On reset we randomize: actuator stiffness and damping (log-uniform scaling, 0.75–1.5 and 0.3–3.0), small perturbations to joint limits, cube friction (static in $[0.5, 1.3]$), cube mass scaling (uniform in $[0.2, 1.0]$), cube initial pose/velocity perturbations, and gravity perturbations (Gaussian additive noise with standard deviation up to 0.5 m/s^2 along z).

e) Teacher PPO training.: We train the teacher with PPO as implemented in RSL-RL [43]. The policy and value networks are MLPs with hidden sizes [512, 512, 256, 128] and ELU activations. We use observation normalization for both actor and critic, and a learned log standard deviation

TABLE III
REWARD SHAPING TERMS USED FOR TEACHER TRAINING IN SIMULATION.

Term	Explanation	Weight
Object position tracking	Exponential tracking to the origin.	$+10$
Rotation magnitude tracking	Exponential tracking to the command.	$+10$
Rotation direction alignment	Alignment between cube angular velocity and commanded axis.	$+30$
Angular acceleration	Penalty on cube angular acceleration.	$+1$
Reaching object	Encourage finger tips to get close to the object.	$+15$
Pose closure	Encourage the vectors from the finger tips to the object's center of mass to be opposite to each other.	$+2$
Joint velocity penalty	ℓ_2 penalty on joint velocities.	-10^{-4}
Action penalty	ℓ_2 penalty on actions.	-5×10^{-4}
Action rate penalty	ℓ_2 penalty on action differences.	-10^{-2}
DOF acceleration penalty	ℓ_2 penalty on joint accelerations.	-10^{-7}

for the Gaussian policy (initial std 1.0). The PPO training hyperparameters are summarized in Table IV.

TABLE IV
TEACHER PPO TRAINING HYPERPARAMETERS.

Hyperparameter	Value
Rollout length (steps per env)	24
Max iterations	10,000
Learning epochs per iteration	5
Minibatches per epoch	4
Learning rate	10^{-4}
LR schedule	adaptive (target KL)
Desired KL	0.01
Discount factor γ	0.99
GAE parameter λ	0.95
PPO clip ϵ	0.2
Entropy coefficient	10^{-4}
Value loss coefficient	1.0
Clipped value loss	enabled
Gradient norm clip	1.0

5) Policy Distillation Procedure: We distill a PPO-trained teacher policy into the SERNF model using IsaacLab. We follow the principle as in [35], with adaptation to chunked actions. During distillation, we applied strong observation noise to the student to improve the robustness. This noise consists of additive Gaussian noise and a random offset that is reset at the beginning of each episode. Noise is applied to both joint positions and velocities as well as to the object state.

We simulate 1,024 parallel environments during distillation. At each time step, the teacher policy produces an action that is recorded as a supervisory signal for the student. The executed action is selected stochastically: with probability p_{teacher}, the teacher's action is executed, and with probability $1 - p_{\text{teacher}}$, the student's predicted action is executed. A data buffer of student observations and the corresponding teacher actions is populated for 128 time steps. Observation–action pairs are then extracted using a sliding window following the structure illustrated in Fig. 7.

The student policy is trained using a behavior cloning loss to match the teacher's actions. The probability p_{teacher} starts at 1.0 and decays by a factor of 0.999 at each iteration to gradually shift the control from the teacher to the student. We initialize the policy by running this distillation process for 200 iterations, corresponding to approximately 20 million gradient steps.

G. Additional Implementation Details

1) Network Architectures:

Visual Encoders

- **ResNet-18 (Robomimic).** Two separate ResNet-18 backbones are used for `img0` and `img1`. Each encoder follows the standard torchvision ResNet-18 up to the last convolutional stage (no global average pooling and no FC classifier). Activation: ReLU. Normalization: BatchNorm2d. Image normalization uses ImageNet mean/std inside the forward pass. For 224×224 inputs, the output is a 7×7 grid of 512-dim tokens (49 tokens).
- **DINOv2 ViT-L/14 (Scissor).** The encoder is `dinov2-vit-l` from timm (ViT-L/14 with 4 register tokens). Images are resized to 224×224 with `resize-naive`. The backbone outputs patch tokens (16×16 grid, 256 patches) taken from the penultimate block; a linear projector maps 1024-dim patch features to 512-dim tokens. Activation: GELU. Normalization: LayerNorm. Finetuning mode: `norm` (only LayerNorm parameters are trainable). A single backbone is shared across the three camera views via `backbone_instance_name`.
- **No visual encoder (Orca in-hand).** Only low-dimensional proprioceptive and command inputs are used.

Normalizing Flow Actor

- **Model.** RealNVP with 16 affine coupling layers and an initial invertible ArcTanh.
- **Coupling network.** Each coupling layer uses a causal TransformerBlock with hidden dim 256, 8 heads, and block_depth 1. The block consists of FlowBlocks with self-attention, cross-attention, RMSNorm, RoPE positional embeddings, and SwiGLU MLP (SiLU gating). Final scale/shift heads use GELU.

Critic Network (Q Chunking)

- **Architecture.** Each critic is a transformer encoder (d_model=512, nhead=8, 3 encoder layers, FF dim=256, activation=GELU, dropout=0.1) operating over observation tokens plus an appended action token.
- **Action token.** Actions are projected via Linear + ELU to 512-dim, with a learned action positional bias.
- **Output.** Distributional Q with 101 bins.

- **Ensemble.** Two critics are used in all NFQ-chunking configs.
- **Normalization.** LayerNorm in each TransformerEncoder layer.

Parameter Counts

- **ResNet-18 (per encoder).** 11,176,512 parameters (no final FC).
- **DINOv2 ViT-L/14 (per backbone + projector).** 303,711,744 parameters.
- **Flow actor (per policy).** 16 coupling layers; per-layer parameters are $3{,}283{,}904 + 1{,}282 \times \text{action_dim}$, giving total $16 \times (3{,}283{,}904 + 1{,}282 \times \text{action_dim})$.
- **Critic (per critic).** $3{,}999{,}077 + 512 \times (\text{action_dim} + T_a + 1)$.

2) Additional hyperparameters values: We apply dropout for π_θ regularization, using a rate of 0.5 during policy initialization (reduced to 0.2 for real-world experiments to accelerate convergence under limited compute). During reinforcement learning, the dropout rate is reduced to 0.1, as higher values were found to degrade offline RL performance, while a small amount of dropout remains beneficial [21, 49].

The number of inverse samples used for action selection is set to 128 in simulation, 64 in real-world inference after policy initialization, and 24 during RL fine-tuning due to GPU memory constraints on the real robot system.

For the scissors task, the batch size is 256 for imitation learning and 48 for all RL stages. For the cube rotation task, the batch size is 1024 for distillation and 512 for the RL stage.

We use a high discount factor $\gamma = 0.997$, which is well suited for the sparse-reward, long-horizon manipulation tasks considered in this work. Target networks are updated using Polyak averaging with $\tau = 0.05$. During the online RL stage of the simulated tasks, offline and online data are mixed with $\rho = 0.5$. In the real world tasks, during the online reinforcement learning stage, training batches are constructed from a mixture of offline and online data sampled in proportion to their sizes, with offline data assigned a mixing weight of 0.5.

All models are trained using the AdamW optimizer [30] with weight decay set to 10^{-4} throughout. The learning rate is 10^{-4} during policy initialization and increased to 2×10^{-4} during reinforcement learning to accelerate adaptation. We use default Adam values $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.

For scissors task, we first train imitation learning policy with 200,000 update steps, then the critic warm-up stage consists of 5,000 gradient updates, full offline RL of 1,000 steps, and online RL of 4,000 steps.

For the cube rotation task, we first distill policy using 20 million updates, then 3,000 steps of critic warmup, 1,000 steps of full offline RL and 3,000 steps of online RL.

For simulation tasks, we train imitation learning policies for 30,000 steps.

3) Baseline Implementation Details: We compare SERNF against strong baseline methods to evaluate the expressiveness and accuracy of our approach in the imitation learning setting. To ensure a fair comparison, the dataset, optimizer settings, data augmentation strategies, and image encoders are kept identical across all methods.

a) Action Chunking Transformer: We implement the Action Chunking Transformer (ACT) [54] using a Transformer-based architecture. A chunk size of 20 is used, as larger chunk sizes are typically required to achieve strong performance with ACT [54]. The style variable has a dimensionality of 32, and the KL divergence loss is weighted by 10.

Conditioned only on low-dimensional state inputs, the action encoder is a transformer with a model dimension of 512, 8 attention heads, 3 layers, a feedforward dimension of 2048, and a dropout rate of 0.1. The action decoder uses the same configuration and is conditioned on both low-dimensional inputs and visual observations. Causal masking is applied to both the encoder and decoder.

Temporal aggregation is used to ensure smooth and accurate action execution, with an exponential weighting parameter of $k = 0.01$.

b) Flow matching: The flow matching baseline is implemented with DiT block policy backbone [12]. The model has a dimension of 512, 8 attention heads, 4 DiT blocks, a feedforward dimension of 2048, a dropout rate of 0.1, and uses 8 flow steps. Causal masking is applied throughout the model. Both training and inference follow the procedure described in [7]. Real time chunking [8] is applied, with the same configuration as our SERNF, as shown in Fig. 7

H. Additional Experimental Results

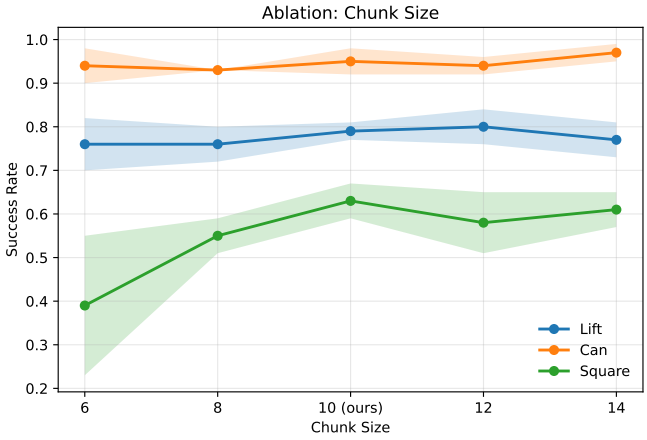

Fig. 10. Effect of action chunk length H on imitation learning performance across RoboMimic Lift, Can, and Square tasks with 4 random seeds.

1) Ablation Studies: We conduct ablation studies to analyze the sensitivity of SERNF to key architectural and algorithmic design choices. The ablations in this subsection are performed on simulated RoboMimic environments (Lift, Can, and Square).

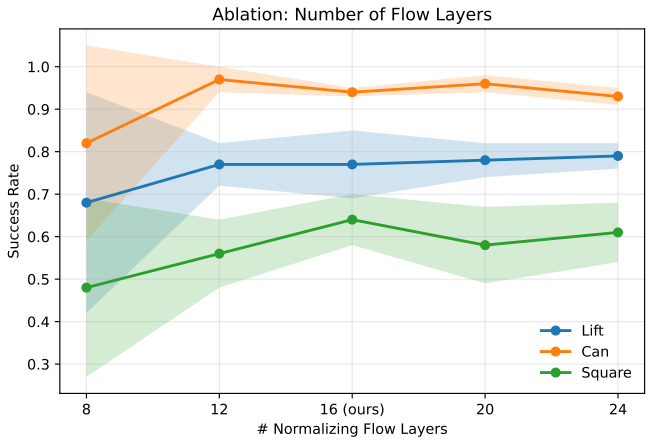

Fig. 11. Effect of normalizing-flow depth (number of coupling blocks) on imitation learning performance across RoboMimic tasks with 4 random seeds.

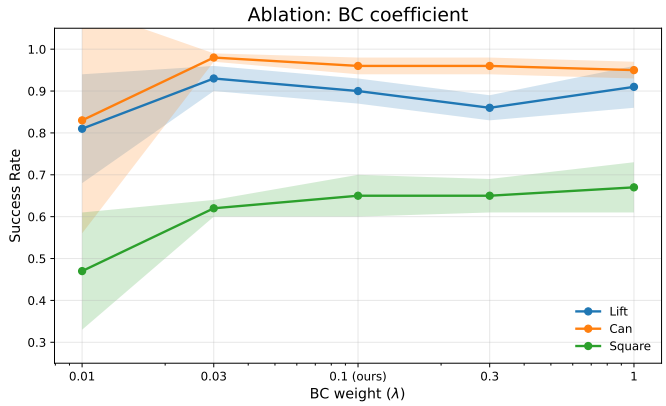

Fig. 12. Effect of λ on offline RL performance across RoboMimic tasks with 4 random seeds.

a) Effect of action chunk length.: Fig. 10 shows the effect of varying the action chunk length $H \in \{6, 8, 10, 12, 14\}$. Across all environments, intermediate chunk sizes yield the most consistent performance. Our choice of $H = 10$ performs robustly across Lift (0.79 ± 0.02), Can (0.95 ± 0.03), and Square (0.63 ± 0.04), balancing temporal abstraction and controllability. Shorter chunks ($H = 6$) lead to a clear degradation on the Square task (0.39 ± 0.16), while larger chunks do not consistently improve performance and can slightly reduce stability due to increased open-loop execution.

b) Effect of normalizing-flow capacity.: Fig. 11 ablates the number of coupling blocks in the normalizing-flow policy, varying the depth from 8 to 24 layers. Performance improves with increasing capacity up to a point and then saturates. A model with 16 coupling blocks achieves strong and stable performance across all tasks (Lift: 0.77 ± 0.08, Can: 0.94 ± 0.01, Square: 0.64 ± 0.06), while shallower models underfit and deeper models provide only marginal gains at higher computational cost. Based on this trade-off, we adopt 16 coupling blocks in all experiments.

c) Effect of BC coefficient.: Fig. 12 studies the impact of the behavior cloning weight $\lambda \in \{0.01, 0.03, 0.1, 0.3, 1.0\}$ on offline RL performance. We observe that performance is sensitive to this hyperparameter, with very small values leading to unstable learning (especially on Can), while overly large values bias the policy toward imitation and can reduce returns on Lift. Overall, $\lambda = 0.1$ provides a good balance between stabilizing policy optimization and allowing improvement beyond the dataset, and we therefore use $\lambda = 0.1$ in all experiments.

2) Failure Mode Analysis: The primary failure modes observed in the scissors task include grasp failure, scissor dropping, and task timeout. Achieving a stable grasp is particularly challenging due to the absence of tactile sensing and frequent occlusion of the index finger by the thumb in wrist-mounted camera views. In both teleoperation and autonomous execution, the fingers must be inserted deeply into the scissor handles to achieve a secure grasp, which further increases sensitivity to perception and control errors. Task timeouts are typically caused by accumulated positioning errors while navigating toward the tape. Additionally, the tape's semi-transparent appearance makes it difficult for the front-facing camera to accurately estimate the relative depth between the scissors and the tape, leading to imprecise cutting motions.

For the cube reorientation task, the dominant failure modes are cube dropping and lack of motion. Cube localization is highly noisy due to frequent occlusion by the fingers, resulting in a significant sim-to-real gap that degrades policy performance.

I. Computational Requirements and Software Details

Training of real-world policies was performed on NVIDIA H200 and A100 GPUs (single GPU job), while policy inference and cube-task distillation were run on an NVIDIA RTX 4090. Simulation experiments were carried out on NVIDIA TITAN RTX GPUs. GPUs were utilized at full capacity to accelerate training and inference; however, all stages can be executed on less powerful hardware by reducing batch sizes or the number of sampled candidate actions during policy evaluation.

For the scissors task, imitation learning pretraining required approximately 35 hours, followed by 21 hours of offline reinforcement learning. For the cube task, simulation-to-real distillation required approximately 4 days on a consumer-grade GPU, while the subsequent real-world offline RL stage completed in about 2 hours.

At inference time, the policy executes one action chunk of length $H = 10$ in approximately 0.24 seconds for the most compute-intensive scissors task on an NVIDIA RTX 3090 Ti. For the cube task, inference latency is below 0.1 seconds per action chunk. These runtimes are compatible with real-time control in all evaluated tasks.

During on-policy training of the scissors task, gradient updates are performed on an NVIDIA H200 GPU, while policy inference and data collection run on a separate desktop equipped with an NVIDIA RTX 3090. The training and inference processes operate asynchronously and coordinate through file transferring and human commands: model checkpoints

and experience buffers are stored in the disk and transferred between machines. The inference process loads the latest available checkpoint for policy execution, while newly collected experience is saved to disk and made available to the training process. The model is implemented in PyTorch, and experience buffers are stored on disk using Zarr.