DATA COLLECTION WITH ADAPTIVE PERTURBATIONS ENABLES EFFECTIVE POLICY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern robot learning systems feed on high-quality demonstration data, and their performance hinges on coverage. Without good coverage, small errors compound and the policy can quickly go into out-of-distribution states from which it cannot recover. Yet there is no principled way of collecting high-coverage data, and data collection often relies on heuristic instructions to human operators. Purely online reinforcement learning (RL) explores at the cost of safety and sample efficiency, while DAgger-style interventions require the policy to fail by chance before a human can correct it. In this paper, we ask instead whether we can modify how the robot behaves *during data collection* to collect more useful data. We propose Data Inoculation (DI), a shared-control framework in which a human provide closed-loop teleoperation commands while a residual policy perturbs these commands. The residual policy is trained online, on the growing dataset, with the same reward-maximizing objective later used for policy extraction, so its perturbations steer data collection toward the deviations a policy trained on the data would make. These perturbations expose the robot to a wider range of states, while the human's continuous corrections provide recovery demonstrations, "inoculating" the dataset against the learner's future mistakes while the human remains in control. On three challenging real-world robotic tasks, DI consistently improves or matches the success rate of policies trained with both behavioral cloning and offline RL, showing DI as an effective data collection strategy for improving downstream policy performance.

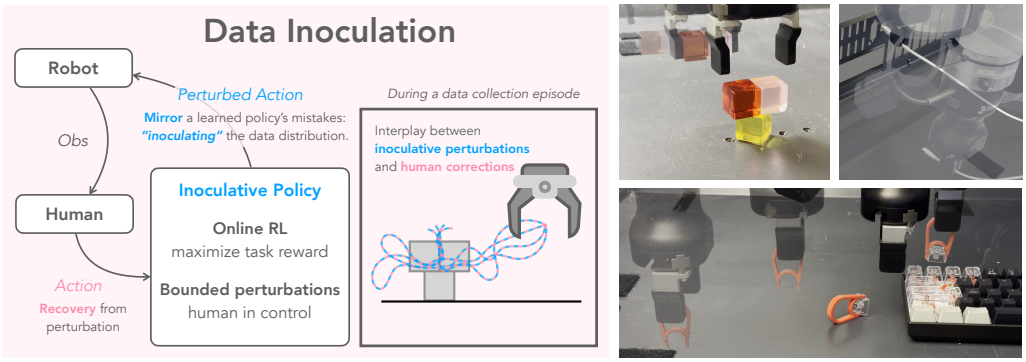

Figure 1: **Data inoculation (DI).** *Left*: During data collection, we train a residual inoculative policy with online RL to perturb the human's closed-loop teleoperation commands. The perturbations steer data collection towards the mistakes the policy trained on the data would make, and elicit human's corrective feedback, "inoculating" the dataset against such mistakes. *Right*: We test DI on three challenging real-world robotic tasks that require precise manipulation, and DI consistently improve success rate of policies trained with behavioral cloning and offline RL.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Modern robot learning systems have shown promise in acquiring complex manipulation skills (*e.g.*, shirt-folding (Intelligence et al., 2025b), tea/juice-making (Wang et al., 2026)) from demonstration data. While there has been extensive recent research on the design of these models, the quality of the data the models are trained on is every bit as critical. Good data should have high coverage: it should contain a wide range of states to maximize the chance that the policy generalizes to the states that it actually visits at test time. In practice, coverage is determined by the scenes where data is collected and the behaviors of the teleoperators, which must strike a balance between consistency and variety. Improving data quality and coverage through task design and operator instruction is largely heuristic and ad hoc. Can we instead directly modify how the robot behaves *during* data collection to provide for more useful data?

Existing solutions largely sit at the two extremes of how the robot is being controlled. Aside from the human-driven teleoperation which exhibits obvious drawbacks above, a learned policy may control the robot autonomously and collect data. While this allows us to run RL online which improves the policy upon its own experience overtime, real-world exploration can be sample inefficient and unsafe. DAgger-style methods partially address this by using learned policy to drive the robot until it makes a mistake, where a human intervenes and corrects it, often through teleoperation. This is reactive: each intervention requires a failure to occur naturally, and rare failure modes may take many rollouts to appear, and for many tasks (*e.g.*, that require working with delicate objects that may break) failures cannot be corrected once they occur.

In this paper, we argue that autonomous policy rollout and human control should *not* be mutually exclusive, and a shared-control framework in which a learned agent and the human drive the robot together allows for better data collection. Our framework is simple: the human retains closed-loop teleoperation of the robot that supplies the demonstration signal as usual in the form of control commands, but these commands are perturbed by a residual policy, which is trained online on the growing dataset, in a way that mimics the mistakes a policy trained on the current data might make. We call this residual policy an *inoculative policy* because it exposes data collection to controlled, human-corrected "doses" of the learner's expected mistakes, so that the policy learned from the collected data is already robust to them. The human can therefore correct policy-like deviations continuously throughout every data collection episode rather than only after a visible failure while ensuring safety because they are always in control of the robot. The resulting data collection strategy, which we call **Data Inoculation (DI)** (Figure 1), produce data that are favourable for downstream policy extraction—it contains near-failure states paired with actions that allow recovery from these states, which is exactly the good coverage data that would help behavior cloning (BC) to learn the recovery behavior directly but also help offline RL to learn better value functions.

Our main contribution is the new data collection strategy (DI). We evaluate DI on three challenging real-world robotic tasks and show that DI consistently improves or matches the success rate of the policies extracted from both behavior cloning (BC) and offline RL. These results demonstrate the benefits of proactively *inoculating* data collection so that the policies become more robust at deployment time.

2 RELATED WORK

Human-in-the-loop learning methods address a common failure mode of imitation learning on passive data, where the learned policy may visit states different from the training distribution and suffers from compounding errors. They typically work by alternating between executing the current policy and collecting human correction data in the form of action re-labeling (Ross et al., 2011) or intervention trajectories (Kelly et al., 2019). Interventions can be triggered automatically by the model (Zhang & Cho, 2016; Hoque et al., 2021; Menda et al., 2019) or initiated by human (Kelly et al., 2019; Mandlekar et al., 2020; Luo et al., 2025; Liu et al., 2025; Hu et al., 2026; Intelligence et al., 2025a). In real-world robotics, intervention-based approaches are often more convenient and natural to human because the corrective trajectories are collected under human's continuous, closed-loop control of the robot. However, this also comes with a limitation: once the control is switched to human, the trajectory may quickly go out of the learned policy's state distribution. As a result, only the first few transitions right after the switching point provide direct supervision to the mistakes made by the

policy. In contrast, rather than using a binary switching from the learned policy's distribution to the expert's distribution after intervention, our method constructs a controlled distribution by combining human demonstrator actions with residual actions from an *inoculation policy*, allowing the human to provide corrections of the policy's mistakes throughout a data collection trajectory. Prior work trains assistive residual policies in simulation, using a model trained on a fixed set of demonstrations to approximate human control, then deploys them with human operators (Sha et al., 2026; Schaff & Walter, 2020). In contrast, our method doesn't depend on simulation and directly train the residual inoculation policy online as data accumulates. This also yields a broader range of corrective demonstrations as updates to the residual policy expose humans to different errors during its training.

Data perturbation approaches broaden the distribution of observations (Tobin et al., 2017; Laskin et al., 2020; Young et al., 2021; Zhou et al., 2023), actions (Zhou et al., 2023; Ke et al., 2024; Hoque et al., 2024; Mandlekar et al., 2023), or even environment and its dynamics (Huang et al., 2025; Wang et al., 2024; Peng et al., 2018; Tan et al., 2018) during training so that the model is robust. These approaches typically directly perturb either a simulator (*e.g.*, domain randomization) or previously collected data (*e.g.*, data augmentation), but do not directly interact with how a human demonstrator collects trajectories, which our method does. The closest line of work injects zero-mean Gaussian noises during data collection, exposing human demonstrator to random mistakes and thereby eliciting recovery behaviors (Laskey et al., 2017; Oh et al., 2023; Tahara et al., 2023). In contrast, our method injects directed residual actions produced by a state-dependent *inoculation policy* trained online to approximate the mistakes that a policy learned from the growing dataset would likely make. This allows the collected trajectories to target policy-specific failure modes like DAgger-style approaches, rather than relying on symmetric random perturbations. Furthermore, while prior approaches mostly focus on improving imitation learning performance, we additionally demonstrate that our technique allows for improved offline RL performance and the quality of learned value functions.

3 PRELIMINARIES

Learning robot policy via reinforcement learning. We consider a Markov decision process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, \gamma, \rho)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $T : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$, reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor $\gamma \in [0, 1)$, and an initial state distribution $\rho \in \Delta_{\mathcal{S}}$. The robot policy receives a partial observation of the state, defined by an observation model, $\Phi : \mathcal{S} \to \mathcal{O}$ (*e.g.*, image observations, proprioceptive states, and the previous actions that are being executed on a robot). The performance of a policy $\pi : \mathcal{O} \to \Delta_{\mathcal{A}}$ is estimated by its Q-function, $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) := \mathbb{E}_\pi[\sum_t \gamma^t r_t]$, averaged over the initial state distribution $\sim \rho$. We use a standard goal-reaching sparse-reward formulation where $r = -1$ until the goal is reached. Optimizing π against $\mathbb{E}_{\boldsymbol{s}\sim\rho}[Q^\pi(\boldsymbol{s}, \boldsymbol{a})]$ corresponds to completing the task as fast as possible. Given a dataset D of transitions $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o}', r)$, we may use the standard temporal-difference (TD) learning procedure where we approximate $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ by $Q_\phi(\boldsymbol{o} = \Phi(\boldsymbol{s}), \boldsymbol{a})$ and train it via the following TD loss:

$$L(\phi) = \mathbb{E}_{\boldsymbol{a}'\sim\pi(\cdot|\boldsymbol{o}')}\left[\left(Q_\phi(\boldsymbol{o}, \boldsymbol{a}) - r - \gamma Q_{\bar{\phi}}(\boldsymbol{o}', \boldsymbol{a}')\right)^2\right]. \tag{1}$$

Meanwhile, we can train the policy to hill-climb on $\mathbb{E}_{\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{o})}\left[Q_\phi(\boldsymbol{o}, \boldsymbol{a})\right]$ to improve its goal-reaching capability.

Vanilla human data collection. A typical human data collection episode goes like this: a human teleoperator drives the robot in closed loop, giving a control command $\boldsymbol{a}_t$ at every step for the robot to execute, while continuously observing the scene to provide any immediate command corrections needed for task completion. The episode terminates on task success, non-recoverable failure (*e.g.*, objects fallen off the table), or a time limit. Throughout the episode, we record a trajectory of control commands and the observations (*e.g.*, by reading off the proprioceptive states provided by the robot and the images from camera(s) attached to the robot) that are being synchronized to the same time steps (*i.e.*, $\{\boldsymbol{a}_t, \boldsymbol{o}_t\}_{t=1}^T$). Since a proficient human teleoperator is often fast and consistent, the states in such data are often narrow and lack recovery. The data may teach the policy how to *proceed* from states an expert visits, but lacks supervision at the states an imperfect learned policy might actually visit at deployment when deviating from the expert trajectories, nor corrective commands that may recover from these deviations.

4 DATA INOCULATION

In this section, we describe in detail how we use a residual policy in a shared-control framework to continuously inject controlled perturbations such that the collected data is favorable for downstream policy training. We start by discussing how a residual policy alongside closed-loop human control can address the failure mode of vanilla human teleop data collection, and then detail how such a residual policy can be learned online during data collection with online RL.

Method overview. As discussed in Section 3, vanilla human data collection can suffer from lack of corrective and recovery demonstrations. DI solves this by intelligently modifying what the robot executes. At each step, an *inoculative policy* π_θ takes the current observation $\boldsymbol{o}_t$ and the teleoperator's live command $\boldsymbol{a}_t$, and generates a unit L_∞ bounded residual perturbation $\Delta_t \sim \pi_\theta(\cdot \mid \boldsymbol{o}_t, \boldsymbol{a}_t)$ so that the executed command $\tilde{\boldsymbol{a}}_t \leftarrow \boldsymbol{a}_t + \alpha\Delta_t$ deviates from the teleoperator's by at most α in L_∞-norm. The teleoperator can then immediately react to the deviation and counter-corrects it through subsequent commands. Every perturbation is then naturally paired with human's follow-up recovery in the trajectory.

Injecting purely random noises without any guidance inefficient, because perturbations along axes that barely influence task success (*e.g.*, yaw rotation in a simple cube pick-and-place task) are uninformative to the downstream learner. Intuitively, the data collection budget is best spent on the critical perturbation axes that influence the task success the most, and elicit recovery behavior as a consequence of these perturbations. However, finding informative perturbations is not simple as they are highly task-dependent and non-stationary, requiring a form of fast adaptation that vanilla zero-mean random noises (as often used in prior work (Laskey et al., 2017)) cannot provide. How should we best implement an algorithm that can quickly adapt and find the perturbation axes that are most informative? It turns out that we can achieve this altogether by simply optimizing the inoculative policy along with human data collection using standard online RL to *maximize task success*.

In particular, we train our inoculative policy using standard TD-based off-policy RL, concurrently with collection, on the growing dataset from the data collection itself. While the data collection loop runs on the robot at the normal control frequency, an asynchronous RL training loop updates π_θ on the incoming data stream to maximize task success rate—or in other words, to maximize its ability to *assist* the teleoperator in completing task. This may seem counter-intuitive: how can a *cooperative* policy produce informative perturbations and elicit recovery behaviors?

This answer is two-fold. First of all, when the inoculative policy is optimized using the same reward-maximizing objective as a downstream offline RL algorithm later uses for policy extraction, its deviations are expected to closely mirror those a downstream learned policy would produce. Furthermore, early in collection, the critic is barely trained and inaccurate which causes the residual policy to exploit the critic's errors, steering the robot into suboptimal states that require explicit recovery behaviors. Aside from eliciting recovery behaviors, our inoculative policy can sometimes also improve data collection efficiency (as we will show in Figure 3) because it is optimized to cooperate with human to improve task success rates.

We have thus far described how training our residual inoculative policy with online RL leads to more informative data collection with the teleoperator maintaining closed-loop control of the robot. Next, we will describe important implementation details that allow efficient learning of our inoculative policy.

Learning the inoculative residual policy online. We train a critic $Q_\phi(\boldsymbol{o}, \tilde{\boldsymbol{a}})$ that estimates the value of the perturbed action directly rather than the value of the residual action Δ. This allows us to update Q_ϕ on any off-policy data regardless whether the action has been perturbed (in practice we collection some portion of our data un-perturbed). To establish the training loss for Q_ϕ, we start from the standard temporal-difference (TD) loss constructed from a transition $(\boldsymbol{o}, \boldsymbol{a}, \boldsymbol{o}')$, where $\tilde{\boldsymbol{a}}'$ is the command perturbed by the current π_θ on top of the human teleoperation command that would have been given at observation $\boldsymbol{o}'$:

$$L(\phi) = \left(Q_\phi(\boldsymbol{o}, \tilde{\boldsymbol{a}}) - r - \gamma Q_{\bar{\phi}}(\boldsymbol{o}', \tilde{\boldsymbol{a}}')\right)^2. \tag{2}$$

However, this requires $\tilde{\boldsymbol{a}}'$ which cannot be obtained from the residual policy π_θ alone because it needs a clean action $\boldsymbol{a}'$ as input that the transition does not have. To approximate $\boldsymbol{a}'$, we additionally train a behavior-cloning (BC) flow model π_β that imitates the human teleoperation commands, trained on

the same growing dataset from the data collection itself. [1] With such approximation, our modified TD loss is therefore

$$L(\phi) = \left(Q_\phi(\boldsymbol{o}, \tilde{\boldsymbol{a}}) - r - \gamma Q_{\bar{\phi}}(\boldsymbol{o}', \boldsymbol{a}' + \alpha\Delta')\right)^2, \qquad \boldsymbol{a}' \sim \pi_\beta(\cdot \mid \boldsymbol{o}') \quad \Delta' \sim \pi_\theta(\cdot \mid \boldsymbol{o}', \boldsymbol{a}'). \tag{3}$$

We can then simultaneously use Q_ϕ to train our inoculative policy π_θ via the following policy loss:

$$L(\theta) = -\mathbb{E}_{\Delta\sim\pi_\theta(\cdot|\boldsymbol{o},\boldsymbol{a})}\left[Q_\phi(\boldsymbol{o}, \boldsymbol{a} + \alpha\Delta) - \lambda\|\Delta\|_2^2\right], \tag{4}$$

where λ is a tunable parameter that additionally penalizes the magnitude of the perturbation which we find to be helpful for environments that require very precise commands (*e.g.*, `t-cube`).

Delay-aware action chunking policy learning. Since the inoculative policy takes the human command as input and adds perturbation on top of it, the executed command needs to be delayed at least the latency of policy inference. To make this work in practice, we adopt a delay-aware learning scheme where we always execute commands a *fixed* interval δ after the observation and human command. We keep a constant δ and evaluate the learned policy with the same fixed delay. To make sure $\boldsymbol{o}_t$ contains enough information about $\boldsymbol{s}_{t+\delta}$ ($t+\delta$ is when the action $\boldsymbol{a}_t$ is actually executed), we include both the observation at t, and the action being executed from t to $t+\delta$:

$$\boldsymbol{o}_t = (I_t, p_t, \boldsymbol{a}_{t-\Delta t}), \tag{5}$$

where I_t is the encoding (using frozen pre-trained ResNet-18 following Luo et al. (2024)) of image observations at t, p_t is the proprioceptive state at t, and $\boldsymbol{a}_{t-\Delta t}$ being the action executed at the previous time step.

Other practical considerations. In our early experiments, we find that the changing perturbation direction too frequently can make the human hard to react to the perturbation effectively at all. In practice, we only predict the perturbation (the action residual) once every 0.5 seconds (5 control time steps with a frequency of 10 Hz). For all 5 time steps, we use the same perturbation added on top of $\boldsymbol{a}_t$ so that the teleoperator has a chance to correct any mistake that the perturbation induces. For our policy, we train an action chunking policy with a horizon of 5, where it predicts a sequence of 5 actions at a time and then execute at the control frequency with a fixed delay δ. To extract this action chunking policy from dataset, we follow the BFN variant of Q-chunking (Li et al., 2025) with a best-of-N of $N = 4$.

Algorithm 1: Data Inoculation

Input: Residual scale α; residual penalty λ
Output: Inoculated demonstration dataset D
$D \leftarrow \emptyset$; Initialize residual policy π_θ, critic Q_ϕ, and BC flow policy π_β

Async Data Collection Loop:

- $\boldsymbol{o}_0 \leftarrow$ initial observation after reset
- **while** $t \in \{0, 1, \cdots\}$ **do**
  - Observe $\boldsymbol{a}_t$ from human teleop
  - $\Delta_t \sim \pi_\theta(\cdot \mid \boldsymbol{o}_t, \boldsymbol{a}_t)$
  - Execute $\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_t + \alpha\Delta_t$
  - Observe $\boldsymbol{o}_{t+1}$ and r_t
  - $D \leftarrow D \cup \{(\boldsymbol{o}_t, \boldsymbol{a}_t, \tilde{\boldsymbol{a}}_t, \boldsymbol{o}_{t+1}, r_t)\}$

Async Training Loop:

- Sample a training batch $(\boldsymbol{o}, \boldsymbol{a}, \tilde{\boldsymbol{a}}, \boldsymbol{o}', r) \sim D$
- Update Q_ϕ with the following TD loss:

$$L(\phi) = \left(Q_\phi(\boldsymbol{o}, \tilde{\boldsymbol{a}}) - r - \gamma Q_{\bar{\phi}}(\boldsymbol{o}', \boldsymbol{a}' + \alpha\Delta')\right)^2,$$

  where $\boldsymbol{a}' \sim \pi_\beta(\cdot \mid \boldsymbol{o}')$, $\Delta' \sim \pi_\theta(\cdot \mid \boldsymbol{o}', \boldsymbol{a}')$

- Update π_θ with the following policy loss:

$$L(\theta) = -\mathbb{E}_{\Delta\sim\pi_\theta(\cdot|\boldsymbol{o},\boldsymbol{a})}\left[Q_\phi(\boldsymbol{o}, \boldsymbol{a} + \alpha\Delta) - \lambda\|\Delta\|_2^2\right],$$

- Update π_β with flow-matching loss using $(\boldsymbol{o}, \boldsymbol{a})$

5 Experimental Setup

We empirically evaluate our method across three different challenging precise manipulation tasks in the real world. We first describe the experimental setup, then the results in Section 6.

[1] We could use $\boldsymbol{a}'$ recorded in the trajectory instead of the flow BC approximation, but this creates an unnecessary non-Markovian dependency on $\boldsymbol{o}, \boldsymbol{a}$ which may hurt learning in theory, so we did not pursue in our experiments.

Robot setup and tasks. We evaluate our method on three real-robot tasks that require precise manipulation. We focus on these tasks because successful executions require precise control of the robot, making it a challenging testbed for data collection and policy learning. We use a Franka robot arm with a Robotiq gripper and two wrist cameras attached near the end effector. We use a spacemouse to control the robot's end effector position and rotation during data collection at a frequency of 10Hz. Illustrations of the three tasks are shown in Figure 1, and we describe them in detail below:

1. `t-cube` requires the robot to pick up two colored cubes and carefully balance them on top of another yellow cube, resulting in a T-shape (Ghugare et al., 2025). While this seems like a simple pick and place task, this in fact requires very precise manipulation skills and is arguably the most demanding task of the three: the robot needs to stack the two top cubes diagonally on top of the yellow cube with the centers of the three cubes roughly aligned, and any misalignment would cause one or two of the top cubes to fall off.
2. `cable` is a precision insertion task where the robot needs to pick up a USB-C cable and insert it into a port hole on a desktop machine that is barely bigger than the cable itself. Without changing the difficulty of the task, we only use the side plate of the desktop machine for ease of task setup.
3. `keycap` tasks the robot to use a keycap puller to remove the Q keycap from a keyboard. The robot must pick up the tool, locate the Q key, position the puller over the keycap, and lock its two hooks onto both sides of the keycap. Successful removal requires precise, directionally controlled force rather than a large brute force pull, which could cause the keycap to fly off. The robot must therefore learn a sequence of small lateral wiggling motions to loosen the keycap before pulling it out and placing it on the table.

Data collection procedure. Since data collection through tele-operation is unavoidably a subjective process and can vary across different operators and collection trials, we conduct experiments on all tasks and data collection strategies with multiple operators and trials, which we refer to as "seeds". For all experiments, three different operators conduct data collection for a total of 9 seeds for the `t-cube` task and 5 seeds for the other two tasks. We collect more seeds for `t-cube` because it is the most difficult task. Before data collection, each operator spent time to practice on the task to achieve a sufficient level of expertise before collecting any data.

Policy learning and evaluation. The observations for the policy consist of the two wrist RGB images, as well as proprioceptive state consisting of the end-effector position, rotation, and gripper position. The policy outputs a $5D$ action as the command, consisting of the delta end effector position, the delta yaw, and the absolute gripper position. For policy training, including both the residual policy used in DI and the downstream policy trained on the collected dataset, we use action chunking (Li et al., 2025) with a chunk size of $h = 5$. In this setting, the policy learns to output a sequence of h atomic actions and execute them in open loop. To evaluate policy performance, we train separate policies for each seed of data collection, and measure success rate over 20 rollouts.

Comparisons. We compare our data collection framework DI against two baseline data collection strategies. The first, ***clean***, refers to collecting data with pure teleoperation, where the control commands from the operator is passed "cleanly" without modification to the robot for closed-loop execution. The second, ***random***, adds uniform random noise to the control command from the teleoperator before executing on the robot. The noise is sampled uniformly from $[-c, c]$ ($c = 0.05$ for `t-cube/cable`, $c = 0.1$ for `keycap`), where 0.1 in the action space translates to 3 mm in end effector position and 0.01 rad in rotation. Similar to DI, we do not add any random noise to the gripper dimension of the action because random opening and closing of the gripper is usually harmful for task progress. In addition, we apply the same random noise for a sequence of $h = 5$ actions, which is the same horizon we use for action chunk learning for the residual policy in DI.

We evaluate the quality of the collected data through downstream policy performance under both imitation learning and offline RL. For imitation learning, we train a flow-matching policy using behavior cloning (BC). For offline RL, we use BFN (Li et al., 2025), which uses the learned Q-function to perform rejection sampling with the BC policy: the BC policy proposes $N = 4$ candidate actions, and the action with the highest predicted Q-value is selected.

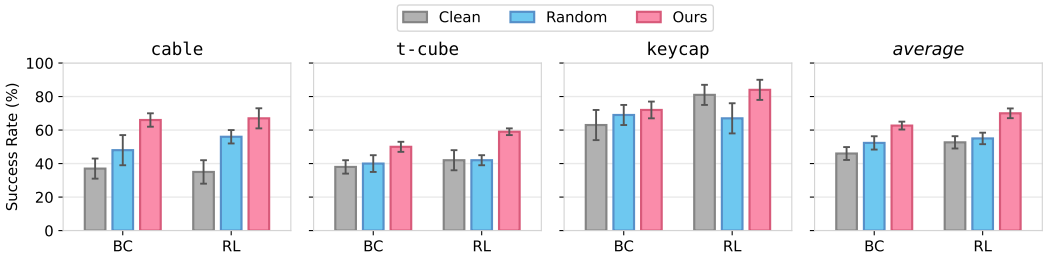

Figure 2: **Task success rates.** The results for `{t-cube/cable/keycap}` are averaged over `{9/5/5}` seeds with 20 evaluation trials each. Error bars indicate the standard errors. Full table in Tables 1 to 4.

6 RESULTS

In this section, we conduct real-robot experiments to study the effectiveness of DI for data collection and its impact on downstream policy learning. We study the following research questions:

Q1. Does DI improve policy performance for BC and offline RL?

First, we investigate whether data collection with DI can improve the performance of the resulting policies, as compared to *clean* and *random* data collection. Since different policy collection methods have different degrees of optimality, we evaluate policies trained with both behavioral cloning and offline RL, as described in Section 5. Figure 2 shows the success rates of both BC and RL policies trained with each data collection method on each of the three real-robot tasks. Overall, our DI data collection method leads to the highest success rate policies, regardless of whether we train with BC or offline RL. This shows that the residual inoculative policy is able to help collect more useful data for downstream learning. Compared to *clean* human demonstrations, data collection with DI significantly improves policy success rate on `cable` and `t-cube`, and is similar or slightly better on `keycap`. The `keycap` task shows similar performance across all three data collection methods, perhaps because the task is less demanding in terms of precise manipulation skills and recovery behavior (supported by high success rate of clean human demonstration in Table 5), and so plain human demonstration is mostly sufficient. Compared to data collection with injected random noise, which only leads to policy performance improvement on `cable` and can hurt performance on `keycap`, DI consistently improves policy performance for both RL and BC policies. This demonstrates that the residual policy learns meaningful state-dependent noise to inject into data collection.

Q2. How does DI affect human demonstrator's success rate?

Given that DI artificially injects a perturbation into the human demonstration, it is natural to ask how DI affects the human demonstrator's success rate during data collection. In Figure 3, we measure the success rate over the course of data collection for the three methods. For both *clean* and *random* data collection, the success rate remains mostly constant. For DI, the success rate starts lower than its counterparts. This is expected since the residual inoculative policy outputs near random actions at the beginning of data collection, and this could negatively impact the success rate initially. However, the success rate of DI quickly rises through data collection, matching the success rates of *clean* and *random* roughly a quarter way into the whole collection session and eventually achieves high success rate than its counterparts. This demonstrates that the residual inoculative policy can quickly learn useful behavior, and can *assist* the human demonstrator to collect higher success rate data for precise manipulation tasks. In contrast, collecting with a random noise almost always decrease success rate over the course of data collection. We also measure the magnitude of the residual action in Figure 3, and found that the magitude of DI residual action consistently decreases over the course of collection, showing that the residual policy shift from outputting random noise to only useful small deviations from human demonstration.

Q3. Does DI Induce Useful Corrective Demonstrations from Human Operator?

Since we observe above that DI improves downstream policy performance, we next seek to understand more mechanistically how DI leads to such improvement. Intuitively, the residual inoculation policy could perturb the robot into an unfamiliar state and elicit corrective demonstration from the

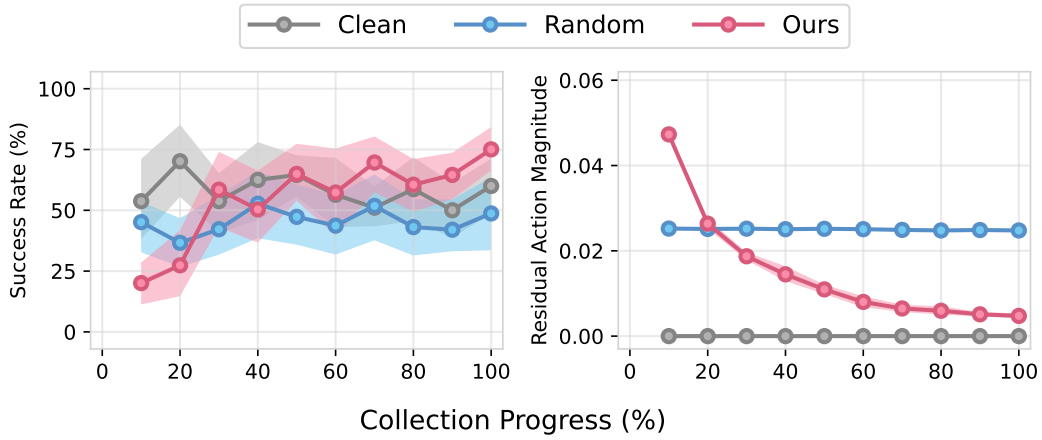

Figure 3: **Success rate over the course of data collection.** DI starts at a lower success rate when the residual policy is untrained, but eventually reaches a higher success rate than clean and random collection.

human operator. To test this intuition, we compare where the corrective demonstrations occur along trajectories collected with DI versus randomly injected noise. We focus our analysis on the `t-cube` task in the following because it is the most precise task, and therefore the most revealing on corrective behavior. We divide the task into 5 phases using rough heuristics: approach (start to close the gripper), grasp (from closing the gripper until $\geq$ 1cm lift), transport (lift to final descent), place/release (final descent through place and release), and retreat (after release). Visualizations fo the five phases are in Figure 4. Then, we quantify the amount of corrective demonstration per phase by weighting the DI residual intervention in this phase by the strength of corrective demonstration. Specifically, if we denote the residual action as Δ_t, the human teleoperation command $\boldsymbol{a}_t$, the chunked corrective direction is $\Delta \boldsymbol{a} = \boldsymbol{a}_{t+H} - \boldsymbol{a}_t$. We define the *strength* of corrective demonstration to be $c_s = \max\left(0, -\frac{\sum_t \Delta_t^\top \Delta \boldsymbol{a}_t}{\sum_t \|\Delta_t\|_2^2}\right)$. Intuitively, the numerator measures whether the human command changes in the direction opposite to the residual over the next $H = 5$ control steps. Opposing changes produce a positive gain; aligned changes produce a negative value that is clipped to zero. The denominator serves as a normalizing constant. We then use c_s to weight the total residual magnitude in this phase, quantifying the *amount* of corrective demonstration: $c_a = c_s \sum_t ||\Delta_t||_1$. Figure 4 shows amount of corrective demonstration (c_a) in each phase as a percentage of the total amount. The result shows that most of the corrective behavior during DI collection happens during the place/release phase, the most difficult and central phase of `t-cube`. Failures in `t-cube` happen almost exclusively in this phase because any imprecise placement or gripper release can knock over the cubes. As such, this is the most important phase from which to collect corrective behavior. In contrast, many corrective demonstration for random data collection happen during the retreat phase, which is relatively straight forward and does not need much corrections. This shows that by learning a residual policy online that adapts to the task, DI is able to collect more useful corrective demonstrations.

7 Discussion

In this paper we introduce Data Inoculation, a data collection strategy that trains a residual RL policy online to aid human teleoperated data collection in a shared-control framework. The residual inoculative policy perturbs the human's control of the robot to a wider range of states and can elicit recovery demonstrations in critical phases of the task, and eventually lead to higher data collection success rate as the residual policy is trained. Data collected with DI also improves the success rate of downstream policy learning. Our work serves as a step towards a new paradigm of robot learning where data collection is shared between humans and autonomous policies, each helping the other. Several challenges remain and is promising for future work. First, our approach trains the residual policy from scratch using online RL, which offers near-random perturbations at the beginning of training. A natural next step is to develop mechanisms for directed exploration that can more quickly provide useful perturbations, such as boostrapping training from an existing offline dataset. Second,

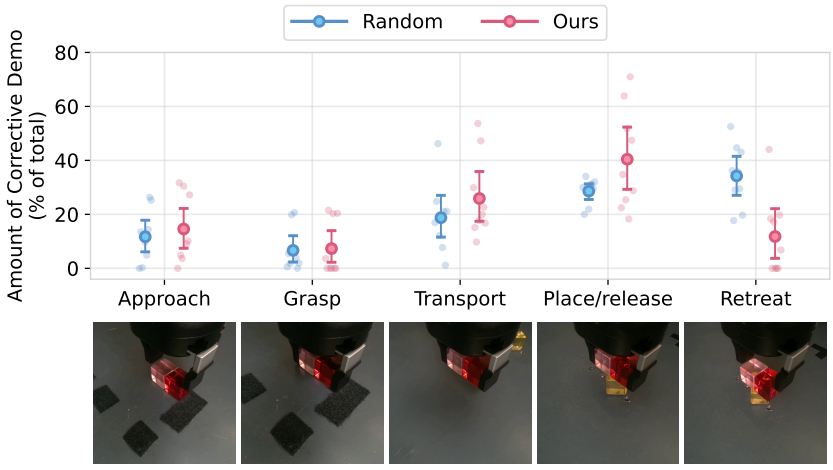

Figure 4: **Amount of human corrective demonstrations over different phases**. DI elicits the most corrections at the most critical phase of the `t-cube` task.

our approach assume the human operator to be fully in-the-loop during data collection, which is expensive in human time. One interesting direction is to explore whether we can automate parts of data collection to be autonomous gradually during training and leave only important correction for human operators.

8 AI USE STATEMENT

In this work, we used AI tools to edit the real-world teaser figures for each task in Figure 1. In particular, we used ChatGPT to predict cropping masks and then used Claude to use the masks to combine multiple real-world camera frames together. We also used Claude to generate matplotlib plots, provide writing feedback, implement methods / support qualitative and thematic data analysis. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of AI.

REFERENCES

Raj Ghugare, Roger Creus Castanyer, Catherine Ji, Kathryn Wantlin, Jin Schofield, Karthik Narasimhan, and Benjamin Eysenbach. Builderbench: The building blocks of intelligent agents. *arXiv preprint arXiv:2510.06288*, 2025.

Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. *arXiv preprint arXiv:2109.08273*, 2021.

Ryan Hoque, Ajay Mandlekar, Caelan Garrett, Ken Goldberg, and Dieter Fox. Intervengen: Interventional data generation for robust and data-efficient robot imitation learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2840–2846. IEEE, 2024.

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. *IEEE Transactions on Robotics*, 2026.

Siyuan Huang, Yue Liao, Siyuan Feng, Shu Jiang, Si Liu, Hongsheng Li, Maoqing Yao, and Guanghui Ren. Adversarial data collection: Human-collaborative perturbations for efficient and robust robotic imitation learning. *arXiv preprint arXiv:2503.11646*, 2025.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin 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*, 2025a.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $pi_0.5$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025b.

Liyiming Ke, Yunchu Zhang, Abhay Deshpande, Siddhartha Srinivasa, and Abhishek Gupta. Ccil: Continuity-based data augmentation for corrective imitation learning. In *International Conference on Learning Representations*, volume 2024, pp. 35813–35831, 2024.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. DART: Noise injection for robust imitation learning. In *Conference on robot learning*, pp. 143–156. PMLR, 2017.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33: 19884–19895, 2020.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=XUks1Y96NR.

Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot learning on the job: Human-in-the-loop autonomy and learning during deployment. *The International Journal of Robotics Research*, 44(10-11):1727–1742, 2025.

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 *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16961–16969. IEEE, 2024.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Human-in-the-loop imitation learning using remote teleoperation. *arXiv preprint arXiv:2012.06733*, 2020.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.

Kunal Menda, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Ensembledagger: A bayesian approach to safe imitation learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5041–5048. IEEE, 2019.

Hanbit Oh, Hikaru Sasaki, Brendan Michael, and Takamitsu Matsubara. Bayesian disturbance injection: Robust imitation learning of flexible policies for robot manipulation. *Neural Networks*, 158:42–58, 2023.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Charles Schaff and Matthew R Walter. Residual policy learning for shared autonomy. *arXiv preprint arXiv:2004.05097*, 2020.

Shuo Sha, Yixuan Wang, Binghao Huang, Antonio Loquercio, and Yunzhu Li. Efficient and reliable teleoperation through real-to-sim-to-real shared autonomy. *arXiv preprint arXiv:2603.17016*, 2026.

Hirotaka Tahara, Hikaru Sasaki, Hanbit Oh, Edgar Anarossi, and Takamitsu Matsubara. Disturbance injection under partial automation: Robust imitation learning for long-horizon tasks. *IEEE Robotics and Automation Letters*, 8(5):2724–2731, 2023.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Jun Wang, Yuzhe Qin, Kaiming Kuang, Yigit Korkmaz, Akhilan Gurumoorthy, Hao Su, and Xiaolong Wang. Cyberdemo: Augmenting simulated human demonstration for real-world dexterous manipulation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17952–17963. IEEE, 2024.

Yi Wang, Xinchen Li, Pengwei Xie, Pu Yang, Buqing Nie, Yunuo Cai, Qinglin Zhang, Chendi Qu, Jeffrey Wu, Jianheng Song, et al. Learning while deploying: Fleet-scale reinforcement learning for generalist robot policies. *arXiv preprint arXiv:2605.00416*, 2026.

Sarah Young, Dhiraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. Visual imitation made easy. In *Conference on Robot learning*, pp. 1992–2005. PMLR, 2021.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.

Allan Zhou, Moo Jin Kim, Lirui Wang, Pete Florence, and Chelsea Finn. Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17907–17917. IEEE, 2023.

A Compute Resource and Robot Hardware

We run all our experiments on a local workstation with two GPUs (RTX-4090). The first GPU trains the models ($\pi_\beta, \pi_\theta, Q_\phi$) and periodically save the model weights to disk, and the second GPU loads the model weights periodically. We use Franka Emika Panda robot arm with the impedance controller from the SERL (Luo et al., 2024). To teleoperate the robot, we use 3Dconnexion spacemouse.

B Task Details

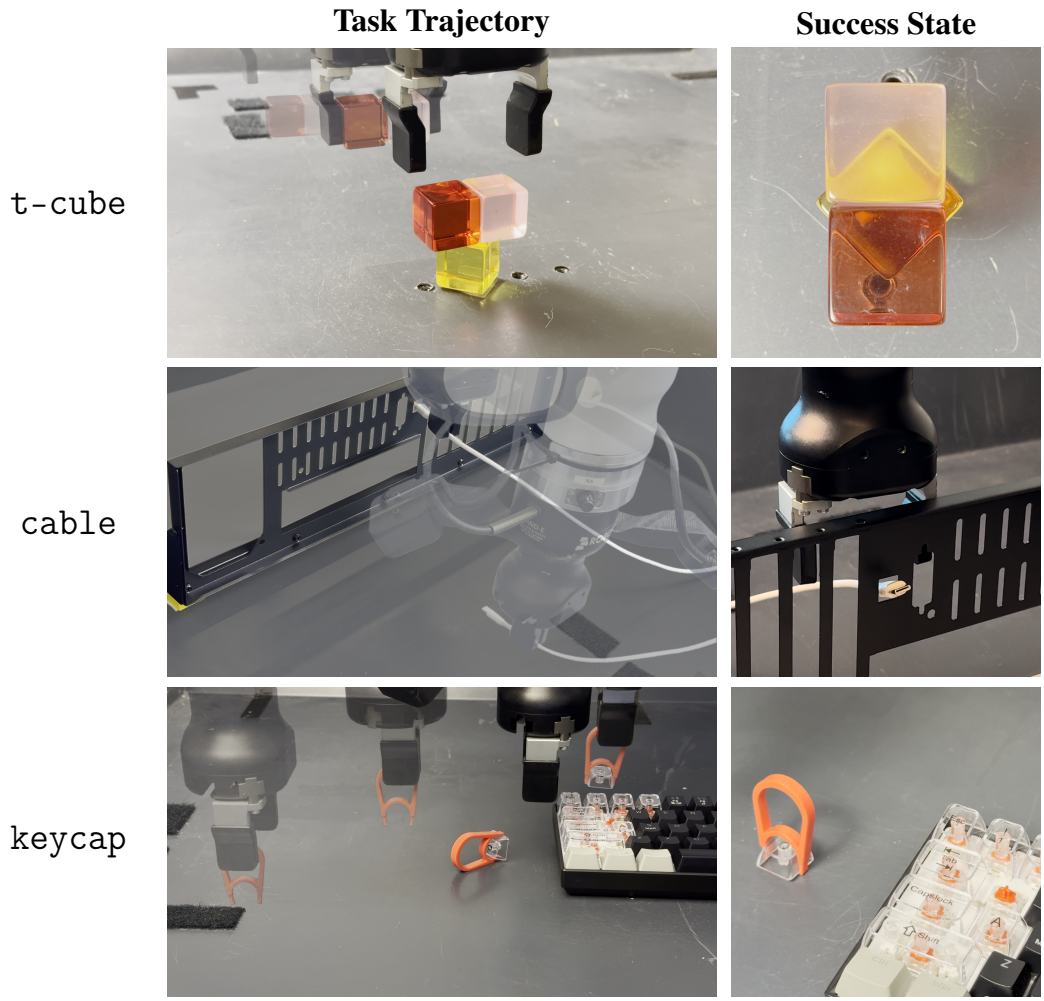

Figure 5: **Three real-robot tasks.** *Top:* `t-cube`. The robot needs to move two colored cubes and carefully balance them on top of the yellow cube. *Middle:* `cable`. The robot needs to feed a USB-C cable through a desktop machine case port hole that is barely bigger than the cable itself. *Bottom:* `keycap`. The robot needs to use a plastic tool to extract a transparent keycap from a keyboard.

Task `t-cube`. The scene contains three acrylic cubes (red, pink, and yellow). The robot must grasp the two cubes (red and pink) together and carefully balance it on top of an equal-sized yellow cube. This task is inspired by the T-Block task from Ghugare et al. (2025). Balancing the two cubes on top of the yellow cube requires a lot of precision. A misalignment of a few millimeters would fail the balance. Nearly all failures of this task occur at placement, making this the most precision-demanding task. An episode succeeds when both cubes rest on top of the yellow cube without falling off.

Task `cable`. The scene contains a side panel of a machine case and a USB-C cable. The robot must pick up the cable lying on the table and then insert it through a port hole on the side panel that is barely

bigger than the cable head. The insertion is a contact-rich process where the robot must search locally and find the right motion that allows the USB-C to go through the port hole. An episode succeeds when the USB-C passes through the port hole and rests there without falling (with the arm away).

Task `keycap`. The scene contains a mechanical keyboard and a keycap-puller tool. The robot must pickup the keycap tool and remove a transparent "Q" keycap from the keyboard. This requires the arm to seat the tool's two hooks on the opposite sides of the keycap, loosen the keycap, and then pull it off the switch. An episode succeeds when the keycap is detached and placed on the table.

C Full Results

Table 1: Task success rates (mean ± standard error). The results for `{t-cube/cable/keycap}` are averaged over `{9/5/5}` seeds with 20 evaluation trials each. Best result per column in bold.

	`cable`		`t-cube`		`keycap`	
	BC	**RL**	**BC**	**RL**	**BC**	**RL**
Clean	0.37 ± 0.06	0.35 ± 0.07	0.38 ± 0.04	0.42 ± 0.06	$\mathbf{0.63 \pm 0.09}$	$\mathbf{0.81 \pm 0.06}$
Random	0.48 ± 0.09	0.56 ± 0.04	0.40 ± 0.05	0.42 ± 0.03	$\mathbf{0.69 \pm 0.06}$	0.67 ± 0.09
Ours	$\mathbf{0.66 \pm 0.04}$	$\mathbf{0.67 \pm 0.06}$	$\mathbf{0.50 \pm 0.03}$	$\mathbf{0.59 \pm 0.02}$	$\mathbf{0.72 \pm 0.05}$	$\mathbf{0.84 \pm 0.06}$

Table 2: Per-seed successes on `cable` (successes out of 20 trials per entry). The last row reports the mean success rate ± standard error over the five seeds.

	Clean		*Random*		*Ours*	
Seed	**BC**	**RL**	**BC**	**RL**	**BC**	**RL**
0	4/20	2/20	6/20	9/20	14/20	14/20
1	11/20	9/20	13/20	13/20	12/20	17/20
2	7/20	7/20	7/20	12/20	12/20	10/20
3	9/20	9/20	7/20	10/20	12/20	13/20
4	6/20	8/20	15/20	12/20	16/20	13/20
Mean	0.37 ± 0.06	0.35 ± 0.07	0.48 ± 0.09	0.56 ± 0.04	$\mathbf{0.66 \pm 0.04}$	$\mathbf{0.67 \pm 0.06}$

Table 3: Per-seed successes on `t-cube` (successes out of 20 trials per entry). The last row reports the mean success rate ± standard error over the nine seeds.

	Clean		*Random*		*Ours*	
Seed	**BC**	**RL**	**BC**	**RL**	**BC**	**RL**
0	9/20	10/20	4/20	9/20	9/20	12/20
1	3/20	0/20	8/20	7/20	8/20	9/20
2	10/20	12/20	11/20	12/20	11/20	12/20
3	11/20	9/20	6/20	7/20	11/20	12/20
4	8/20	9/20	7/20	7/20	10/20	11/20
5	8/20	8/20	4/20	8/20	10/20	12/20
6	6/20	9/20	9/20	8/20	10/20	13/20
7	5/20	6/20	14/20	9/20	8/20	12/20
8	8/20	12/20	9/20	8/20	13/20	14/20
Mean	0.38 ± 0.04	0.42 ± 0.06	0.40 ± 0.05	0.42 ± 0.03	$\mathbf{0.50 \pm 0.03}$	$\mathbf{0.59 \pm 0.02}$

Table 4: Per-seed successes on `keycap` (successes out of 20 trials per entry). The last row reports the mean success rate ± standard error over the five seeds.

	Clean		*Random*		*Ours*	
Seed	**BC**	**RL**	**BC**	**RL**	**BC**	**RL**
0	14/20	19/20	17/20	19/20	17/20	20/20
1	14/20	15/20	15/20	13/20	17/20	18/20
2	6/20	13/20	15/20	15/20	13/20	17/20
3	13/20	15/20	11/20	9/20	12/20	16/20
4	16/20	19/20	11/20	11/20	13/20	13/20
Mean	0.63 ± 0.09	0.81 ± 0.06	0.69 ± 0.06	0.67 ± 0.09	$\mathbf{0.72 \pm 0.05}$	$\mathbf{0.84 \pm 0.06}$

Table 5: Success rates during data collection, computed from the recorded episodes. Each entry reports successful episodes over all collected episodes and the corresponding percentage.

	`cable`	`t-cube`	`keycap`
Clean	252/260 (96.9%)	432/749 (57.7%)	113/118 (95.8%)
Random	233/244 (95.5%)	311/686 (45.3%)	122/123 (99.2%)
Ours	178/189 (94.2%)	361/663 (54.4%)	110/117 (94.0%)

D HYPERPARAMETERS

Parameter	**Value**
Batch size	256
Discount factor (γ)	0.97
Action chunk size (h)	5
Best-of-N sampling (N)	4
Optimizer	Adam
Learning rate	3×10^{-4}
Target network update rate (λ)	5×10^{-3}
Target entropy for residual Gaussian policy	$-5 \times \lvert\mathcal{A}\rvert$
Zero residual action probability	0.1
Maximum residual action magnitude	$\lVert\Delta a\rVert_\infty \leq 0.1$
Network width	512
Network depth	4 hidden layers
Critic pessimistic backup (ρ)	2
Critic ensemble size	10
Policy flow steps (T)	10
Action delay (δ)	0.05 s
Control frequency (F)	10 Hz

Table 6: **Shared hyperparameters.**

Tasks	Random Noise c	Ours β	Training Steps	Data Size
`t-cube`	0.05	10	1×10^5	10000
`cable`	0.05	0.03	0.5×10^5	4000
`keycap`	0.1	0.01	0.5×10^5	3000

Table 7: **Task-specific hyperparameters.**