

RL HAS A DAGGER IN ITS SLEEVE: SLOPE-BASED EVALUATION OF
VALUE ESTIMATES FOR ROBOT-GATED INTERVENTIONS

by

Karyna Volokhatiuk

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

© Copyright 2026 by Karyna Volokhatiuk

Abstract

RL has a DAgger in its SLEEVE: SLoPE-based Evaluation of Value Estimates for
Robot-Gated Interventions

Karyna Volokhatiuk

Master of Applied Science

Graduate Department of Aerospace Science and Engineering

University of Toronto

2026

Reinforcement learning (RL) can fine-tune robot policies with fewer environment interactions when a human expert intervenes to provide help during training. Human-gated intervention, however, requires continuous supervision. Robot-gated methods could remove that burden, but most train an additional model or were developed for imitation learning. We propose SLEEVE (SLoPE-based Evaluation of Value Estimates), a robot-gated criterion that uses Q-values from a critic trained as part of an actor-critic algorithm. During each rollout, SLEEVE estimates the rate of improvement in recent Q-values and requests help when that rate becomes non-positive, indicating stalled progress. The method requires Q-values that increase along successful trajectories, a property provided by the demonstration-based pretraining used in ConRFT. Across simulated robosuite tasks and real-world assembly tasks, SLEEVE matches or outperforms human-gated fine-tuning on all but one task. In simulation, it also outperforms random robot-gated baselines. We ablate intervention timing, duration, and necessity, showing how each affects learning.

Acknowledgements

I am deeply grateful to my supervisor, Professor Jonathan Kelly, for guiding me through this academic journey. He gave me the opportunity to work on this project, offered valuable advice throughout, and was always responsive to my questions, encouraging and supporting me along the way. His mentorship helped me grow both academically and professionally, and his jokes and San Pellegrino supplies kept my spirits up.

I am also very grateful to Miguel Angel Rogel Garcia, Trevor Ablett, and Philippe Nadeau from our STARS lab for all their advice and for kindly answering my countless questions. I also owe special thanks to Miguel for his assistance with the real robot experiments. I am grateful to everyone who took part in the experiments: Alison Lei, Vlad Surdu, Ben Wei, Phone Thiha Kyaw, and Oliver Petrovic. I owe special thanks to Alison for helping test our method with a new model. That work is still ongoing and therefore not included in this thesis, but I appreciate her help and support. I would also like to thank my lab-mates at STARS lab and at the Robotics Institute for making my stay in Toronto more enjoyable and fun.

I am truly grateful to Nicholas Rhinehart from the LEAF lab for the helpful discussions and insightful advice on my work. From Magna, I am thankful to Ye Zhang, Parag Patre, Mochan Shrestha, Ninad Doshi, and Ruban Maria Arulraj for sharing their expertise and useful suggestions, and for giving me the chance to begin this project in the first place.

I am super grateful to my partner Denys and our dog Olko, who were always close by and made me feel happy and supported. Our dog walks have been a great support for my peace of mind, especially during the last months of writing this thesis. I am also thankful to my family outside of Toronto, who always believed in me and encouraged my studies and interests. Finally, I am grateful to my best friend Kate, who listened to all my audio voices and was always there for me.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	3
2	Background	6
2.1	Reinforcement Learning Preliminaries	6
2.1.1	Markov Decision Processes	7
2.1.2	Value Functions and the Bellman Equations	7
2.1.3	Temporal-Difference Learning and the TD Error	8
2.1.4	On-Policy and Off-Policy Learning	9
2.1.5	Off-Policy Actor-Critic Methods with Experience Replay	9
2.2	Pretraining of Off-Policy RL	10
2.2.1	Why Demonstrations Alone Are Not Enough	11
2.2.2	Conservative Q-Learning	11
2.2.3	Calibration and Cal-QL	12
2.2.4	Alternatives to Conservative Pretraining	13
2.3	ConRFT	14
2.3.1	A Consistency Action Head on a Frozen Backbone	14
2.3.2	Stage I: Offline Fine-Tuning with Cal-ConRFT	15
2.3.3	Stage II: Online Fine-Tuning with HIL-ConRFT	16
2.4	Interactive Imitation Learning	17
2.5	Making Q-Values Track Progress	18
3	Related Work	22
3.1	Who Guards the Gate? Human- vs. Robot-Gated Learning	22
3.2	What Robot-Gated Methods Measure	23
3.2.1	Uncertainty and Novelty	23
3.2.2	Discrepancy from the Expert	24

3.2.3	Risk and Safety	25
3.2.4	Progress Toward the Goal	26
3.3	Positioning	28
4	SLEEVE: SLoPE-based Evaluation of Value Estimates	30
4.1	The Criterion	30
4.2	Hyperparameters	32
4.3	Computing the Slope	33
4.4	Integration with ConRFT	33
5	Experimental Design	36
5.1	Motivation for Simulation Experiments	36
5.2	Adapting robosuite to a Real-World Setting	37
5.3	Simulated Tasks and Scripted Experts	39
5.3.1	Pick-Place	39
5.3.2	Nut Assembly	40
5.3.3	Door	41
5.3.4	Wipe	42
5.4	Real-World Setup	43
5.5	Real-World Tasks	44
5.5.1	USB Insertion	44
5.5.2	Waterproof Connector	46
5.6	Reward Classifier	46
5.7	Baselines	47
5.7.1	Human Baseline	47
5.7.2	Random Baselines	48
5.7.3	Complete-After-Start Intervention Baselines	49
5.7.4	No-Intervention Baseline	49
5.8	Metrics	49
6	Results and Discussion	51
6.1	Hyperparameter Search	51
6.2	Comparison with the Human-Gated Baseline	53
6.2.1	One Human Operator	53
6.2.2	Multiple Human Operators	55
6.2.3	Expert Time	55
6.2.4	Query Accuracy	55

6.3	Comparison with Random Baselines	59
6.4	Ablations	61
6.5	Real-World Experiments	62
7	Conclusion and Future Work	66
7.1	Discussion	66
7.2	Limitations and Potential Improvements	67
	Appendices	70
A	Additional Metrics	71
	Bibliography	78

List of Tables

3.1	Summary of robot- and human-gated methods	29
5.1	Environment setup and training parameters for the Pick-Place task. . . .	39
5.2	Environment setup and training parameters for the Nut Assembly task. .	40
5.3	Environment setup and training parameters for the Door task.	41
5.4	Environment setup and training parameters for the Wipe task.	42
5.5	Environment setup and training parameters for the USB Insertion task. .	44
5.6	Environment setup and training parameters for the Waterproof Connector task.	46
6.1	Summary of metrics plotted in experiments	52
6.2	Expert effort for SLEEVE and each human operator	58
6.3	Query decisions and task outcomes	61

List of Figures

2.1	Whether Q tracks progress depends on where the reward is paid.	20
4.1	SLEEVE on one episode of real-robot fine-tuning	31
4.2	ConRFT with SLEEVE	34
5.1	The two real-world tasks. The top image in each column shows the scene after a reset, and the bottom image shows the goal state.	45
6.1	Hyperparameter search over the slope window	54
6.2	Comparison with Human 1	56
6.3	Comparison with all four operators	57
6.4	Comparison with the random baselines	60
6.5	Ablations on intervention timing and length	63
6.6	Real-world experiments	64
A.1	Hyperparameter search over the slope window, all metrics	72
A.2	Comparison with Human 1, all metrics	73
A.3	Comparison with all four operators, all metrics	74
A.4	Comparison with the random baselines, all metrics	75
A.5	Ablations on intervention timing and length, all metrics	76
A.6	Real-world experiments, all metrics	77

List of Abbreviations

BC	behaviour cloning
Cal-ConRFT	Calibrated ConRFT (offline stage)
Cal-QL	Calibrated Q-learning
CASI	complete-after-start intervention
CASI-B	complete-after-start intervention with a budget
ConRFT	Consistency-based Reinforced Fine-Tuning
CQL	Conservative Q-learning
Dagger	Dataset Aggregation
ERRG	empirical random robot-gated
ERRG-C	empirical random robot-gated with curriculum
HG-Dagger	human-gated Dagger
HIL-ConRFT	Human-in-the-Loop ConRFT (online stage)
HIL-SERL	Human-in-the-Loop Sample-Efficient Robotic Reinforcement Learning
IL	imitation learning
OOD	out-of-distribution
RL	reinforcement learning
SAC	Soft Actor-Critic
SERL	Sample-Efficient Robotic Reinforcement Learning
SLEEVE	SLopE-based Evaluation of Value Estimates
TD	temporal-difference
VLA	vision-language-action
VLM	vision-language model

Chapter 1

Introduction

This chapter motivates the use of robot-gated interventions for reinforcement learning on physical robots and introduces the problem addressed in this thesis. Section 1.1 explains why interventions are useful during online training and why triggering them automatically can reduce the burden on a human operator. Section 1.2 then summarizes our algorithm, SLEEVE (SLoPE-based Evaluation of Value Estimates), the main contributions of the thesis, and the experimental results.

1.1 Motivation

Automating an assembly operation is straightforward when the robot can replay the same programmed path repeatedly [1]. When the environment is uncertain or the task varies significantly, the engineering needed to cope can cost more than paying someone to do the work. This challenge is particularly relevant in high-mix low-volume manufacturing, which motivated our collaboration with Magna International, one of the world’s largest automotive suppliers [2]. Magna’s operations span both component supply and complete-vehicle manufacturing [3], with high product variety across programs of limited duration. These are precisely the conditions under which the cost of automating individual operations can be difficult to justify.

For some operations, the robot cannot simply repeat the same motion because the part or task changes from one attempt to the next. A flexible component, for example, may not lie in exactly the same configuration each time, while a tight insertion may require the robot to respond to contact forces as the parts come together. Custom fixtures, specialized grippers, and carefully tuned force control can address these problems, but they must often be redesigned or retuned for each operation, making this approach difficult to justify at low production volumes.

Learning from demonstrations avoids programming the motion: a human performs the operation and the robot learns from that. Fitting a policy to those demonstrations by behaviour cloning, however, produces a policy that can be no faster and no more efficient than the demonstrator. This is because the objective is to match the demonstrated actions, not to do the task well. Under reinforcement learning (RL) the policy is scored on what its actions achieve rather than on how closely they resemble the demonstrations, so it can improve on the demonstrations instead of just matching them. But training a manipulation policy with RL alone takes far more interaction than is practical to collect on a physical robot. Demonstrations reduce the amount of interaction required. A small set of demonstrations is added to the replay buffer alongside the agent’s own transitions. HIL-SERL (Human-in-the-Loop Sample-Efficient Robotic Reinforcement Learning) [4] and ConRFT (Consistency-based Reinforced Fine-Tuning) [5] are two open-source pipelines that follow this approach, using demonstrations to make reinforcement learning practical on a physical robot. Both learn manipulation tasks, including contact-rich assembly, within a few hours of real-robot training. ConRFT additionally pretrains the policy and critic offline on the demonstrations before online fine-tuning begins.

Initializing from demonstrations shortens fine-tuning but does not remove exploration. The policy still has to try actions and see how they turn out. On a physical robot, much of this exploration is unproductive. For example, the arm may repeatedly press a connector against its port at an angle where it cannot seat until the episode times out. Letting a human take control mid-episode addresses this problem by moving the robot out of a stalled state and giving the policy another chance to finish the task. Both HIL-SERL and ConRFT rely on human takeover. HIL-SERL identifies high-mix low-volume manufacturing as a setting in which such a framework is particularly valuable.

Human-in-the-loop operation is common in deployed commercial robotic systems. Nimble [6] and Kindred [7], for example, automate warehouse picking but rely on remote operators when a robot cannot complete a pick. The operator takes over, and the correction can then be used to improve the system. Telexistence [8] restocks convenience-store shelves automatically and successfully more than 98% of the time and keeps remote operators on standby for the remainder; they take manual control when a bottle is misplaced or topples over. Aktoria Robotics [9] sells this arrangement itself, providing trained teleoperators available at any hour together with the software needed to connect them to a customer’s fleet. A robotics company can buy operator time instead of hiring and training staff of its own. MicroFactory [10] puts interventions at the centre of training rather than deployment. There, an operator first demonstrates the task through a teleoperation kit, then lets the robot run the task itself, correcting it when it misses; the corrections

become training data. The company reports that, on a complex task, collecting demonstrations takes about an hour and correcting the robot’s own attempts about a day [11]. After that the robot works on its own, notifying an operator when something goes wrong.

Interventions can be triggered from either side: *human-gated* when the human decides to step in, *robot-gated* when the robot asks for help. In the examples above, interventions at deployment are robot-gated, whereas the training stages of MicroFactory, HIL-SERL, and ConRFT are all human-gated. In each, an operator watches the whole training and decides when to take over, unable to do anything else. Under a robot-gated criterion the operator would be called only when the robot needs help, leaving them free in between to supervise several runs at once or to do other work [12]. This reduction in supervision matters most at high product variety, because each new operation needs its own fine-tuning run.

Robot-gated criteria for training are not new, but almost all of them were designed for imitation learning (IL). There, intervention answers a specific problem: a policy fitted to demonstrations drifts into states those demonstrations never covered. DAgger (Dataset Aggregation) [13] and its variants correct that drift by training on the expert’s actions from the states the policy itself reaches. Under RL the purpose is different: interventions are an aid to exploration. They are most useful when the policy has stalled and would take a long time to make progress on its own, if at all, or when it is about to enter a state from which it cannot recover. HIL-SERL shows that RL with interventions outperforms IL given the same amount of demonstrations and corrections, because RL explores a wider range of states and optimizes the task reward, which is what allows it to improve on the demonstrations rather than reproduce them [4].

In either setting, most robot-gated criteria require a separate model whose only job is to decide when to intervene. This thesis asks whether that decision can instead be made using quantities that the RL algorithm already computes.

1.2 Contributions

We introduce SLEEVE (SLoPE-based Evaluation of Value Estimates), a robot-gated intervention criterion that fits a least-squares line to the last M Q-value estimates of the current episode and asks for help when the slope of the line is no longer positive. A positive slope means the policy is making progress; a non-positive one means it has stalled. Those estimates come from the critic that an off-policy actor-critic algorithm already trains to evaluate the policy’s actions, so the criterion needs no additional model and no additional data. We identify the condition SLEEVE depends on, that is, the

critic’s Q-values must rise along successful trajectories. We also show that this condition holds whenever the reward is concentrated at the goal. To the best of our knowledge, no prior work gates human takeover on the trend of Q-values from the policy’s own critic during robotic RL.

We place SLEEVE among existing methods, surveying robot-gated criteria in both IL and RL and grouping them by what they measure to decide when to ask for help. The survey highlights two points. First, most criteria involve a quantity produced by a model trained only to make that decision, such as an out-of-distribution detector, discrepancy predictor, safety critic, or progress model. SLEEVE avoids the need for this additional model. Second, a criterion built for IL does not always carry over to RL. Gating on the discrepancy between the robot’s action and the one the expert would take, for instance, treats any deviation as evidence of failure. Under RL, however, the same deviation may instead reflect improvement by the policy.

We evaluate SLEEVE against human-gated fine-tuning across four simulated manipulation environments, each with up to four operators. To keep intervention quality out of the comparison, we develop a scripted expert for each task that can take over from any state the policy reaches. The same expert supplies the actions in both settings, so the two differ only in who triggers the intervention. In all but one environment, SLEEVE matches or outperforms every operator’s success rate, with fewer interventions and less expert time. The exception is an environment where progress is less clearly reflected in the Q-values.

We then test four aspects of an intervention separately: how accurate the decision to intervene is, whether its timing matters, how long it should last, and whether it is needed at all. On the same recorded rollouts, SLEEVE and a human reach comparable accuracy and F1 scores in identifying episodes that the policy cannot finish on its own. The timing matters: random robot-gated baselines that intervene at arbitrary moments learn more slowly, even though they spend a comparable number of expert steps. Returning control before the episode ends is more effective than letting the expert finish the task. Without interventions, the policy in the hardest environment barely improves over its initial performance

Finally, we test SLEEVE on a real robot, using two assembly tasks from the National Institute of Standards and Technology (NIST) Task Board 1. It reaches a success rate comparable to the human-gated baseline, showing that Q-values computed from real, noisier observations still provide a usable trigger. That result comes at a lower cost in supervision, as the operator no longer has to watch the whole training process.

The main contributions of this thesis are summarized below.

- We introduce SLEEVE, which gates on the trend of the critic’s Q-value estimates and needs no additional model or data.
- We group existing robot-gated criteria by what they measure, and show why an IL criterion does not always carry over to RL.
- We write a scripted expert per task and use it in both the human- and robot-gated settings, so intervention quality does not affect the comparison.
- We compare SLEEVE with human-gated fine-tuning across four simulated environments, each with up to four operators.
- We analyze an intervention’s accuracy, timing, length, and necessity.
- We validate SLEEVE on two real-robot NIST Task Board assembly tasks.

The remainder of the thesis is organized as follows. Chapter 2 presents the reinforcement learning machinery this thesis relies on, and the condition that makes Q-values track progress toward the goal. Chapter 3 reviews prior work on human- and robot-gated interventions, and sets out how this thesis differs. Chapter 4 defines SLEEVE. Chapter 5 covers the simulated and real tasks, the experts, and the baselines; Chapter 6 presents the results; and Chapter 7 closes with the limitations and directions for future work.

Chapter 2

Background

This chapter covers the topics from reinforcement learning and interactive imitation learning that the rest of the thesis relies on. Training a manipulation policy on a physical robot allows only a limited number of interactions with the environment. Because of this limit, we make three choices in our setup: learning is off-policy, so that every transition can be reused; the policy and its critic are pretrained on demonstrations before any online interaction; and an expert can intervene during online training.

We provide background on each of these choices. Section 2.1 presents off-policy reinforcement learning, Section 2.2 covers the pretraining that fits a policy and a critic to demonstrations, and Section 2.3 describes ConRFT, which combines the two and is the algorithm this thesis builds on. Section 2.4 explains interactive imitation learning, how the expert’s role changes when the learner is an RL agent, and the intervention terminology we use throughout. Finally, Section 2.5 establishes the condition under which a critic’s values rise as the robot approaches the goal. In the next chapter, we survey how existing methods in imitation learning and reinforcement learning decide when an intervention should begin.

2.1 Reinforcement Learning Preliminaries

This section describes the reinforcement learning concepts used in this thesis. We start by formulating the task as a Markov decision process (MDP), then define the value functions on which our intervention criterion is based, and finally describe the algorithms that learn these value functions from a mixture of demonstration and interaction data. We follow Sutton and Barto [14] for the fundamentals and the deep RL literature for more recent developments. We adopt the conventions of the latter rather than those of the textbook: rewards are indexed by the timestep of the action that produced them, states and actions

are written in lowercase, and value functions in uppercase. A superscript denotes the policy associated with a value function, as in Q^π and V^μ , and a subscript denotes the parameters of a neural network that approximates the function, as in Q_θ .

2.1.1 Markov Decision Processes

We model a robot manipulation task as an MDP, specified by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. At each timestep, the agent is in a state $s \in \mathcal{S}$ and selects an action $a \in \mathcal{A}$. The environment then returns a scalar reward $r(s, a)$ and a successor state drawn from the transition distribution $p(s' \mid s, a)$. The Markov property requires p and r to depend only on the current state and action; once s_t is known, whatever happened before it no longer matters. In this thesis, the policy controls a robot arm carrying out the task, and cameras watch the arm as it acts. In every task, the policy receives camera images together with proprioceptive measurements of the arm, which together form the state s .

Behaviour is described by a policy $\pi(a \mid s)$, a conditional distribution over actions. The quantity to be maximized is the discounted return, the sum of future rewards weighted by how far ahead they occur,

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad (2.1)$$

where the discount factor $\gamma \in [0, 1)$ makes rewards count for less the further ahead they arrive.

The tasks in this thesis are episodic. Interaction is divided into episodes, and an episode ends in one of three ways: the task is completed, the robot reaches a state from which the task can no longer be completed, or a fixed time limit is reached. The manipulator arm is then reset and a new episode begins. Following the usual convention [14], a terminal state is absorbing: no action leaves it, and it yields zero reward from then on. Because everything after the termination of an episode contributes zero to Equation 2.1, the same formula holds for episodic and continuing tasks.

2.1.2 Value Functions and the Bellman Equations

Two value functions describe the return expected under a given policy. The state-value function is the return expected from a state if π is followed thereafter,

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s], \quad (2.2)$$

and the action-value function is the return expected from a state after a given action is taken, with π followed thereafter,

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]. \quad (2.3)$$

The two are related by averaging over the policy's action choice, $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q^\pi(s, a)]$.

If we split the return in Equation 2.1 into its first term and the remainder, we obtain the Bellman equation for Q^π , which says that the value of a state-action pair is the immediate reward plus the discounted value of whatever follows under π :

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a), a' \sim \pi(\cdot|s')} [Q^\pi(s', a')]. \quad (2.4)$$

Substituting an arbitrary function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ on the right-hand side gives a different function on the left. The map from one to the other is the Bellman operator $\mathcal{B}^\pi$,

$$(\mathcal{B}^\pi Q)(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a), a' \sim \pi(\cdot|s')} [Q(s', a')]. \quad (2.5)$$

Equation 2.4 then says that $\mathcal{B}^\pi Q^\pi = Q^\pi$; the operator returns Q^π unchanged, so Q^π is a fixed point of $\mathcal{B}^\pi$. Applying the operator to any other function brings it closer to Q^π . If Q is wrong by at most ε at every state-action pair, then $\mathcal{B}^\pi Q$ is wrong by at most $\gamma\varepsilon$ [15]; in other words, $\mathcal{B}^\pi$ is a contraction with modulus γ , and Q^π is its only fixed point. Repeating the update $Q \leftarrow \mathcal{B}^\pi Q$ therefore drives Q to Q^π from any starting point. In Section 2.1.3, we show how sampled transitions can be used to approximate the expectation in Equation 2.5 and update the value function.

2.1.3 Temporal-Difference Learning and the TD Error

The expectation in Equation 2.5 cannot be computed exactly, so we estimate it from a single observed transition (s_t, a_t, r_t, s_{t+1}) , with the next action drawn from the policy, $a_{t+1} \sim \pi(\cdot \mid s_{t+1})$. For a current estimate Q , the gap between the two sides of the Bellman equation is then

$$\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t), \quad (2.6)$$

which is called the temporal-difference (TD) error. It measures how far the estimate is from being self-consistent. Building the target $r_t + \gamma Q(s_{t+1}, a_{t+1})$ from the same Q we are trying to improve is called bootstrapping.

We train an action-value network Q_θ by minimizing the squared TD error, averaged over a set of stored transitions $\mathcal{D}$,

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} [(\mathcal{B}^\pi Q_\theta(s,a) - Q_\theta(s,a))^2], \quad (2.7)$$

where we hold the target $\mathcal{B}^\pi Q_\theta(s,a)$ fixed when taking the gradient, so no gradient reaches θ through it.

2.1.4 On-Policy and Off-Policy Learning

Algorithms differ in whether the data used for an update must come from the policy that is being improved. On-policy methods use data generated by the policy being improved, so past transitions generally cannot be reused after the policy changes, and fresh data must be collected for subsequent updates. Off-policy methods separate these two roles: the *behaviour policy* generates the transitions, the *target policy* is the one being learned, and the two can be different. The Bellman operator in Equation 2.5 allows this directly, because the expectation over a' is taken with respect to the target policy π , no matter which action was actually executed at the next state.

This thesis builds on off-policy RL because the majority of the data it relies on are produced by something other than the current policy. The demonstrations that initialize training are teleoperated by a human or generated by a scripted controller, and the corrective interventions that are the subject of this thesis are, by construction, actions the policy did not choose. Off-policy learning also allows us to use each transition for many gradient updates rather than one, which makes training on a physical robot feasible within a realistic number of environment steps.

2.1.5 Off-Policy Actor-Critic Methods with Experience Replay

Learning Q does not by itself give a policy. Using Q to choose actions requires maximizing it over the action space at every state, and that maximization is intractable when actions are continuous. Actor-critic methods [16, 17] avoid this search by learning an explicit policy alongside the value estimate. Thus, two networks are trained, an actor and a critic. The critic Q_θ estimates action values as above, and the actor π_ψ , which is the target policy of Section 2.1.4, is trained to produce actions the critic rates highly,

$$\mathcal{L}_\pi^Q(\psi) = -\mathbb{E}_{s \sim \mathcal{D}} [Q_\theta(s, \pi_\psi(s))]. \quad (2.8)$$

Here $\pi_\psi(s)$ denotes an action sampled from the policy at state s . We update the two networks in alternation.

Rather than discarding transitions after use, off-policy methods store them in a replay buffer and draw minibatches from it [18]. Mnih et al. [18] note that drawing minibatches from the buffer also breaks the temporal correlation between consecutive samples, which reduces the variance of the updates.

One problem with Equation 2.7 is that the target $\mathcal{B}^\pi Q_\theta(s, a)$ is built from Q_θ itself, so its value shifts every time θ is updated. To avoid this, the target is computed with a separate copy of the critic, called the target network $\bar{Q}_\theta$. Its parameters $\bar{\theta}$ track θ slowly and are not trained. Two schemes are common. Either $\bar{\theta}$ is copied from θ every fixed number of steps [18], or it is moved toward θ after every critic update [19],

$$\bar{\theta} \leftarrow \tau \theta + (1 - \tau) \bar{\theta}, \quad (2.9)$$

at a rate $\tau \ll 1$. We use the second scheme, as ConRFT does (Section 2.3). The target therefore stays approximately fixed over many updates to θ , which stabilizes the critic updates that minimize $\mathcal{L}_Q(\theta)$ in Equation 2.7.

A second problem is overestimation bias. Because the actor maximizes Q_θ , it is biased toward actions where Q_θ is too high; bootstrapping on those actions propagates the error, and estimates inflate over training. Fujimoto et al. [20] address this by training two critics, Q_{θ_1} and Q_{θ_2} , from different initializations on the same data, and forming the TD target from the smaller of the two. Taking the minimum biases the target downward wherever the two critics disagree and reduces the overestimation. We write the equations in this thesis for a single critic but train two in practice. When forming the TD target, we use the smaller of the two critic estimates; whenever a single Q-value is needed elsewhere, we use their average. Soft Actor-Critic (SAC) [21] combines these components with an entropy bonus that rewards stochastic policies, and it is a common choice for off-policy continuous control.

2.2 Pretraining of Off-Policy RL

Every algorithm in Section 2.1 assumes that the robot is interacting with the environment. The buffer fills as the robot acts, and every value the critic learns is eventually tested against what the environment returns. Pretraining removes this assumption and uses only demonstrations. Still, this stage must produce a policy and a critic that online training can improve from the first step. The critic, however, is hard to pretrain, because

its loss queries Q_θ at actions the demonstrations do not have. In this section, we explain why training on demonstrations alone breaks the critic and how pretraining methods fix the issue.

2.2.1 Why Demonstrations Alone Are Not Enough

The most obvious way to pretrain a critic is to run the algorithms of Section 2.1.5 on a fixed set of demonstrations instead of live interaction. The critic loss of Equation 2.7 and the actor loss of Equation 2.8 average over whatever transitions the buffer contains, and neither requires on-policy data. Nothing therefore stops us from reusing these algorithms, but doing so does not work in practice.

The Bellman target in Equation 2.5 uses $Q_\theta(s', a')$ with $a' \sim \pi(\cdot | s')$, so it depends on the action chosen by the current policy at the next state rather than on the action taken by the demonstrator. These policy actions may not appear in the demonstration data, so their Q-values must be estimated from limited or no direct experience. As a result, the critic may overestimate or underestimate their values. These inaccurate estimates are then used in subsequent Bellman targets. Because the corresponding actions may never appear in the dataset, there may be no observed transition that corrects the error. The actor selects actions with high estimated Q-values, so overestimated actions are especially likely to be preferred. Bootstrapping can then propagate these overestimates to earlier states.

This is related to the overestimation described in Section 2.1.5, but the source of the problem is different. During online training, an overestimated action can be corrected after it is executed, because the resulting transition provides the reward and next state needed to update its value. In offline training, unseen actions are never executed, so this corrective feedback is unavailable. Using two critics and taking their minimum does not fully solve the problem, since both critics are trained on the same demonstration data and may assign overly high values to the same unseen actions. The minimum can therefore still be an overestimate.

2.2.2 Conservative Q-Learning

Conservative Q-learning (CQL) [22] addresses the overestimation of unseen actions described in Section 2.2.1. It adds a term to the critic loss that pushes down the values of

policy actions and pushes up the values of dataset actions,

$$\begin{aligned} \mathcal{L}_Q^{CQL}(\theta) = & \underbrace{\alpha \left(\mathbb{E}_{s \sim \mathcal{D}, a \sim \pi(\cdot|s)}[Q_\theta(s, a)] - \mathbb{E}_{(s,a) \sim \mathcal{D}}[Q_\theta(s, a)] \right)}_{\mathcal{R}(\theta)} \\ & + \frac{1}{2} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(\mathcal{B}^\pi \overline{Q}_\theta(s, a) - Q_\theta(s, a) \right)^2 \right], \end{aligned} \quad (2.10)$$

where the first term is the conservative regularizer $\mathcal{R}(\theta)$ and the second is the critic loss of Equation 2.7. A larger α makes the critic more conservative.

The two expectations in $\mathcal{R}(\theta)$ are taken over the same states and differ only in where the action comes from. Their difference is therefore how much higher the critic rates the policy’s action than the demonstrator’s. Minimizing the loss makes the critic rate the policy’s actions below the demonstrator’s, as far as the TD term allows. For a large enough α , CQL guarantees that at every state the critic’s value for the policy is a lower bound on its true value, $\mathbb{E}_{a \sim \pi(\cdot|s)}[Q_\theta(s, a)] \leq V^\pi(s)$ [22].

2.2.3 Calibration and Cal-QL

The conservative regularizer of Section 2.2.2 guarantees the direction of the critic’s error but not its size. When the true values are near -3 , a critic whose values sit near -40 satisfies CQL’s lower bound exactly as well as a roughly correct critic. Nakamoto et al. [23] report that the Q-values CQL learns offline end up far below their ground-truth values.

This is not a problem while training stays offline, because the actor learns from the critic’s relative values, not absolute ones. It becomes a problem at the first online step. Once the robot begins interacting, the TD term updates the critic on the actions it executes. Everywhere else, the values remain underestimated. The values updated online are therefore on a different scale from those that remain underestimated. A poor action executed online can therefore be valued above a good action that the policy never executed and whose value is still underestimated. The actor then follows the poor action and unlearns the policy obtained from the demonstrations. Nakamoto et al. show that the interval spent correcting the scale of the Q-values coincides with the interval over which the policy’s performance recovers [23, Fig. 3]. Until that correction is complete, the policy does not improve on its offline initialization.

The critic therefore needs a bound from below as well. We can obtain it from the data. Let μ be a reference policy whose value V^μ we can estimate from the recorded

episodes. We call Q_θ calibrated with respect to μ when

$$\mathbb{E}_{a \sim \pi(\cdot|s)}[Q_\theta(s, a)] \geq V^\mu(s) \quad \text{for all } s \in \mathcal{D}. \quad (2.11)$$

The calibrated critic may still underestimate the policy, but not below the value of the reference policy. When new online data arrives, the critic already sits within a bounded distance of the true value of π , and not arbitrarily far below it. The initial unlearning is therefore reduced, and the policy recovers quickly [23]. The natural choice of μ is the behaviour policy that produced the demonstrations. We can approximate V^μ by the discounted return of the recorded rollouts of μ . This requires no bootstrapping, and therefore avoids the overestimation described in Section 2.2.1. For a demonstration that terminates at step T ,

$$V^\mu(s_t) \approx \sum_{k=0}^{T-t-1} \gamma^k r_{t+k}. \quad (2.12)$$

CQL pushes down the values of all policy actions. In contrast, Calibrated Q-learning (Cal-QL) [23] pushes down only the values above V^μ ,

$$\mathcal{R}^{Cal}(\theta) = \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi(\cdot|s)}[\max(Q_\theta(s, a), V^\mu(s))] - \mathbb{E}_{(s,a) \sim \mathcal{D}}[Q_\theta(s, a)]. \quad (2.13)$$

Where $Q_\theta(s, a)$ is above $V^\mu(s)$, the maximum selects it and the term behaves exactly as in CQL. Where it is below, the maximum selects $V^\mu(s)$ instead, which does not depend on θ , so the expectation over policy actions contributes no gradient.

Together, calibration and conservatism bound the critic,

$$V^\mu(s) \leq \mathbb{E}_{a \sim \pi(\cdot|s)}[Q_\theta(s, a)] \leq V^\pi(s). \quad (2.14)$$

Calibration supplies the lower bound and conservatism the upper one.

2.2.4 Alternatives to Conservative Pretraining

Conservative methods address the problem of Section 2.2.1 at the critic. Behaviour-regularized methods address it at the policy instead. They keep the learned policy close to the behaviour policy, so the critic is rarely queried far from the demonstrated actions. TD3+BC [24] adds a behaviour-cloning (BC) term to the actor loss of Twin Delayed Deep Deterministic policy gradient (TD3). Advantage weighted actor critic (AWAC) [25] drops the value-maximization term entirely and fits the policy to the dataset actions by maximum likelihood, weighting each action by the exponential of its advantage

$Q_\theta(s, a) - \mathbb{E}_{\tilde{a} \sim \pi(\cdot | s)}[Q_\theta(s, \tilde{a})]$. AWAC’s critic is trained with the target of Equation 2.5, which evaluates Q_θ at $a' \sim \pi(\cdot | s')$. Implicit Q-learning (IQL) [26] instead replaces the $Q_\theta(s', a')$ term with $V_\xi(s')$. This state-value network V_ξ is fitted to a high expectile of Q_θ over the dataset’s actions. The policy is extracted as in AWAC, except that $V_\xi(s)$ replaces the expectation over π in the advantage.

We can also use the demonstrations without a pretraining stage at all. Reinforcement learning with prior data (RLPD) [27] starts its policy and critic from random initialization. It keeps two buffers, one containing the demonstrations and one filling with the policy’s own transitions. At each gradient step, it draws half of the training batch from each.

Finally, behaviour regularization can be combined with a conservative critic. ConRFT applies behaviour regularization to its actor, as in TD3+BC, and Cal-QL to its critic. We describe ConRFT in the next section.

2.3 ConRFT

Consistency-based Reinforced Fine-Tuning (ConRFT) [5] combines the components of Sections 2.1 and 2.2 into the algorithm this thesis builds on. Its policy is a pretrained vision-language-action (VLA) backbone followed by a new consistency action head. This head and a critic are trained in two stages. The offline stage, Cal-ConRFT, fits both networks to 20–30 demonstrations, and the online stage, HIL-ConRFT, continues to train them as the robot interacts with the environment. HIL-ConRFT uses human-gated interventions, with an operator monitoring training and intervening when needed. In this section, we describe the objectives used in each stage and define the notation used in Chapter 4.

2.3.1 A Consistency Action Head on a Frozen Backbone

The policy $\pi_{\phi, \psi}$ is a pretrained VLA backbone with parameters ϕ followed by a consistency action head with parameters ψ . The transformer-based backbone encodes the camera images and a natural-language task description into a state encoding $E_\phi(s)$. ConRFT uses Octo-small [28] as the backbone. It is a 27-million-parameter model pretrained on 800k robot trajectories from the Open X-Embodiment dataset [29]. The head maps $E_\phi(s)$ to an action. Only ψ is trained; ϕ stays frozen.

The head is a consistency policy conditioned on $E_\phi(s)$. Consistency policies are part of the diffusion family and produce an action from a sample of Gaussian noise. A diffusion

model is trained to map a noisy sample to a slightly less noisy one, so generation starts from pure noise and repeats this step many times. A consistency model [30] is trained so that a sample at any noise level maps directly to the clean sample. Generating an action is therefore a single forward pass through the consistency head rather than a chain of passes. We train the head by minimizing the following loss

$$\mathcal{L}_\pi^{BC}(\psi) = \mathbb{E}_{(s,a) \sim \mathcal{D}, k, z} [\|f_\psi(a + kz, k \mid E_\phi(s)) - a\|], \quad (2.15)$$

where f_ψ is the head, $z \sim \mathcal{N}(0, I)$ is Gaussian noise, and k is a noise level drawn uniformly from a fixed set of $N = 40$ levels spanning $[0.002, 80]$, with more levels at the low end. At generation time, we run the head on pure noise sampled at the largest of these levels. Its output is the action.

One-step generation matters because ConRFT’s actor loss in both stages (Sections 2.3.2 and 2.3.3) includes $\mathcal{L}_\pi^Q$ of Equation 2.8. That term evaluates Q_θ at the action the head generates, so its gradient must travel back through the head into ψ . This is straightforward for one-step generation and considerably harder for an iterative one. A single forward pass also keeps the latency low enough for the 10 Hz control rate used throughout this thesis.

2.3.2 Stage I: Offline Fine-Tuning with Cal-ConRFT

At the start of the offline stage, both the policy and the critic have a randomly initialized two-layer multilayer perceptron (MLP) on top of a frozen encoder. Each MLP has a hidden size of 256. The policy’s encoder is the VLA backbone, and its MLP is the consistency head, as described in Section 2.3.1. The critic’s encoder is a ResNet-10 [31] pretrained on ImageNet [32], as in HIL-SERL [4]. The policy and the critic are fitted to the demonstration buffer $\mathcal{D}$ alone, and the robot does not interact with the environment. We train the critic by minimizing

$$\begin{aligned} \mathcal{L}_Q^{offline}(\theta) = & \alpha \left(\mathbb{E}_{s \sim \mathcal{D}, a \sim \pi(\cdot|s)} [\max(Q_\theta(s, a), V^\mu(s))] - \mathbb{E}_{(s,a) \sim \mathcal{D}} [Q_\theta(s, a)] \right) \\ & + \frac{1}{2} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[(\mathcal{B}^\pi \overline{Q}_\theta(s, a) - Q_\theta(s, a))^2 \right], \end{aligned} \quad (2.16)$$

which is Equation 2.10 with the calibrated regularizer of Equation 2.13 in place of $\mathcal{R}(\theta)$.

We train the policy by minimizing

$$\mathcal{L}_\pi^{offline}(\psi) = \beta \mathcal{L}_\pi^{BC} + \eta \mathcal{L}_\pi^Q \quad (2.17)$$

whose two terms are the behaviour-cloning loss of Equation 2.15 and the value loss of Equation 2.8. Their weights β and η control the trade-off between imitating the demonstrated actions and maximizing value.

Following ConRFT [5], we set $\alpha = 0.01$, $\beta = 1.0$, and $\eta = 0.1$ in the offline stage. The same values are used for every task. Throughout this thesis, we refer to the offline fine-tuning stage as pretraining, since this stage happens before the robot starts interacting with the environment, and since the critic and consistency head are randomly initialized at the start of this stage. ConRFT itself calls it offline fine-tuning because the underlying VLA backbone is already pretrained before this stage begins.

2.3.3 Stage II: Online Fine-Tuning with HIL-ConRFT

For the online stage, we keep the demonstrations in $\mathcal{D}$ and add a replay buffer $\mathcal{R}$. Every transition collected during interaction with the environment is written to $\mathcal{R}$, whether the policy or the human was in control. Transitions where the human was in control are also added to $\mathcal{D}$. Following the symmetric sampling of RLPD (Section 2.2.4), every gradient step draws half of its batch from each buffer.

The training objectives also change. The critic loss drops the conservative term of Equation 2.16, and the remaining TD term takes its expectation over both buffers, $\mathcal{D} \cup \mathcal{R}$,

$$\mathcal{L}_Q^{online}(\theta) = \mathbb{E}_{(s,a,s') \sim (\mathcal{D} \cup \mathcal{R})} \left[\left(\mathcal{B}^\pi \overline{Q}_{\bar{\theta}}(s, a) - Q_\theta(s, a) \right)^2 \right]. \quad (2.18)$$

Once the robot begins interacting with the environment, the values of the actions it executes are corrected by the returns they earn, so conservatism is no longer needed.

The policy loss keeps both of its terms as in the offline stage (Equation 2.17),

$$\mathcal{L}_\pi^{online}(\psi) = \beta \mathcal{L}_\pi^{BC} + \eta \mathcal{L}_\pi^Q. \quad (2.19)$$

Only two things change. First, both terms now take their expectations over $\mathcal{D} \cup \mathcal{R}$ rather than over $\mathcal{D}$ alone,

$$\begin{aligned} \mathcal{L}_\pi^{BC}(\psi) &= \mathbb{E}_{(s,a) \sim (\mathcal{D} \cup \mathcal{R}), k, z} [\|f_\psi(a + kz, k \mid E_\phi(s)) - a\|], \\ \mathcal{L}_\pi^Q(\psi) &= -\mathbb{E}_{s \sim (\mathcal{D} \cup \mathcal{R})} [Q_\theta(s, \pi_{\phi, \psi}(s))]. \end{aligned} \quad (2.20)$$

Second, β is decreased to 0.5 and η is increased to 1.0, which shifts the balance from imitating the demonstrations toward maximizing value, since online the policy is expected to improve beyond the demonstrations [5]. The behaviour-cloning term is kept because it stops the policy from drifting far enough from the demonstrations to collapse

its performance, and because it stabilizes exploration where an unsafe action can damage hardware [5].

2.4 Interactive Imitation Learning

Allowing a human to take over during an episode, as in ConRFT’s online stage, is an approach that originated in imitation learning rather than reinforcement learning. In this section, we describe how the expert’s role differs in the two settings and introduce the terms the rest of the thesis uses.

Celemin et al. [33] define interactive imitation learning as “a branch of Imitation Learning (IL) where human feedback is provided intermittently during robot execution allowing an online improvement of the robot’s behavior.” Several kinds of feedback fit this definition, and the one this thesis builds on is the Dataset Aggregation (DAgger) family, in which the human supplies corrective actions at the states the learner itself visits. However, DAgger does not require a human. Any expert policy can supply the corrections, including scripted policies, model-based controllers, planners, and policies trained in advance.

The DAgger family extends behaviour cloning, where a policy is fitted to a fixed set of demonstrations by supervised learning. A small error can move a behaviour-cloning policy into states the demonstrations never covered. The policy was not trained on those states, so its next error is larger. The deviation compounds over a rollout, and the policy drifts. DAgger [13] addresses this with an iterative procedure. At each iteration, the current policy is rolled out, the expert labels each visited state with the action they would have taken, these state-action pairs are aggregated into the dataset, and a new policy is trained on the whole dataset. Because the labelled states are the policy’s own, the gap between the data it is fitted on and the data it encounters at deployment shrinks with each iteration.

However, labelling every visited state is costly, and much of this cost comes from states the policy can already handle. Later variants of DAgger reduce the cost by intervening selectively rather than labelling every visited state. Human-gated DAgger (HG-DAgger) [34] does so by letting the human take over during the policy’s rollouts when they judge it necessary. Only those interventions are added to the dataset.

In RL, the expert has a different role. Because every RL transition has a reward, the robot can learn from its own experience without any expert actions. Given enough exploration, it can also learn to recover from the states where a behaviour-cloning policy would drift instead. Intervention has two other purposes. The first one is training speed:

when the robot stalls, the expert can help it progress, so training converges sooner than under autonomous exploration alone [4, 5]. The second one is safety: an unsafe action during autonomous exploration can damage the robot or its surroundings. In contrast to IL, RL keeps every transition, whether the policy or the expert was in control and whether the attempt succeeded or failed. The critic and the actor are trained on all of them. ConRFT’s online stage (Section 2.3.3) works this way, with the human choosing when to intervene as in HG-Dagger.

Three terms recur in the rest of the thesis. The *expert* is the source of corrective actions; an *intervention* is a segment of steps during which the expert holds control; and *gating* is the decision of when this segment begins and ends. Methods are called *human-gated* or *robot-gated* according to who makes this decision, and we compare the two in Section 3.1.

2.5 Making Q-Values Track Progress

The intervention criterion of Chapter 4 is based on a single assumption, that the critic’s values rise as the policy makes progress toward the goal. A flat or falling trend is therefore used as an indication that the policy has stalled. This assumption does not hold for Q-functions in general. By Equation 2.3, Q is the expected return from the current state-action pair onward, and nothing in that definition requires it to increase as the robot approaches the goal. In this section, we give the conditions on the reward under which the assumption holds and those under which it does not.

Consider a sparse reward, a discount $\gamma < 1$, and a state-action pair (s, a) from which the goal is k steps away. The action that completes the task earns $R > 0$, and every other action earns $-c$. Of these k steps, the first $k - 1$ earn $-c$ and the k -th earns R . Because the terminal state is absorbing, we can calculate the value as a function of k rather than of (s, a) ,

$$\begin{aligned} Q(k) &= \sum_{j=0}^{k-2} \gamma^j (-c) + \gamma^{k-1} R \\ &= -\frac{c(1 - \gamma^{k-1})}{1 - \gamma} + R\gamma^{k-1} \\ &= \left(R + \frac{c}{1 - \gamma}\right) \gamma^{k-1} - \frac{c}{1 - \gamma}, \end{aligned} \tag{2.21}$$

As $k \rightarrow \infty$, $\gamma^{k-1} \rightarrow 0$ and $Q(k) \rightarrow -c/(1 - \gamma)$, which is the value of never finishing the task. The function $Q(k)$ is strictly decreasing in k if and only if $R + c/(1 - \gamma) > 0$, and this holds for any penalty $c \geq 0$. The value therefore rises with every step closer to the

goal. Progress appears in the value by construction, without any reward shaping. The same conclusion holds even when the reward at the goal is zero ($R = 0$), but only if every other step gets penalty $c > 0$.

The function $Q(k)$ ranges from $-c/(1 - \gamma)$ for a trajectory that never succeeds to R for a trajectory one step from the goal. That difference of $R + c/(1 - \gamma)$ does not divide evenly across the steps. Differencing Equation 2.21 gives the per-step increment,

$$Q(k) - Q(k + 1) = (1 - \gamma) \left(R + \frac{c}{1 - \gamma} \right) \gamma^{k-1}, \quad (2.22)$$

which shrinks by a factor of γ with every step away from the goal. Thus, the increment is large near the goal and small far from it. With the values ConRFT uses ($R = 10$, $c = 0.05$, and $\gamma = 0.98$ [5]), the two ends of a 100-step episode differ by 10.81 in value, but the increment falls from 0.25 per step at the goal to about 0.03 at the start. A small increment is comparable to the critic’s own noise, so a single step is an unreliable signal. Chapter 4 therefore fits its slope over a window of M steps rather than differencing consecutive values.

Above, we described the reward function under which the critic can be used as a signal of progress. However, not every reward function satisfies this. Figure 2.1 shows the same episode under four reward functions, at the same discount. Panels (a) and (b) satisfy $R + c/(1 - \gamma) > 0$ and behave as Equation 2.21 predicts, with the value climbing to R and steepening as the goal nears. Panel (a) has no per-step penalty; panel (b) has one. In contrast, neither (c) nor (d) has the reward Equation 2.21 assumed. Panel (c) adds a milestone bonus to the reward function in (b), a further +1 paid once, before the goal, and (d) replaces $-c$ with a positive reward that changes from step to step.

At a given step, Q counts the current action’s reward and all those that follow, never one already collected. The step that earns a bonus is the last one whose value includes it, so the value at this step can exceed the following step’s if the bonus is large enough. In (c), the bonus at step 70 does exactly that, and the value dips for a single step before climbing again. In (d), a reward is paid at every step, for closeness to the object and more once it is grasped, so most of Q comes from per-step reward and not from the reward at the goal. The closeness reward is negligible while the gripper is far away, and the value climbs much as in (a) and (b). The closeness reward grows as the gripper nears the object, and from step 64 each step collects enough that what remains ahead is worth less than it was, so the value falls even though the robot continues to make progress and completes the task successfully.

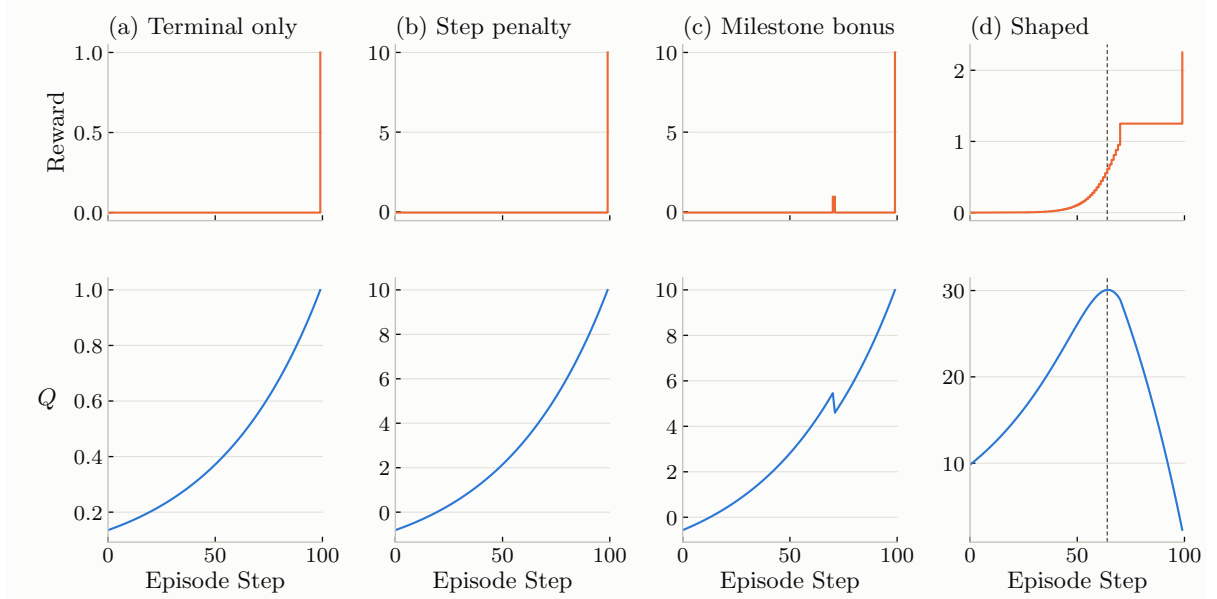


Figure 2.1: Per-step reward (top) and the value it produces (bottom), along a modelled episode that succeeds on its last action, with $\gamma = 0.98$. The reward sequences are constructed rather than logged from rollouts, and Q is computed exactly from those rewards rather than learned by a critic. (a) +1 at success and 0 per step. (b) The reward used in this thesis, +10 at success and -0.05 per step. (c) Same as (b), with a milestone bonus, a further +1 paid once when a subtask is completed, at step 70 in this example. (d) A shaped reward after robosuite’s Lift [35]: a closeness term rising from 0 to 1 as the gripper nears the object, a further +0.25 per step while the object is held, and +2.25 at success. In the modelled episode, the robot neared the object at constant speed. Only the reward values are taken from robosuite; the initial distance to the object and the robot’s speed are chosen for illustration. The dashed line indicates where the value begins to fall.

Even when the reward satisfies $R+c/(1-\gamma) > 0$, Equation 2.21 is only an idealization. The equation computes the return along one fixed trajectory, whereas Q^π averages over every trajectory the policy might actually take. The value is therefore guaranteed to rise only on average, not on the step that actually occurs. This is a second reason for fitting the slope over a window rather than differencing consecutive values.

Chapter 3

Related Work

In Section 2.4 of the previous chapter, we introduced gating as the problem of deciding when an intervention should begin and when control should return to the policy. This chapter reviews how existing methods address this problem. Section 3.1 compares human-gated and robot-gated approaches, while Section 3.2 groups robot-gated methods by the signals they use to trigger and end interventions. Section 3.3 then explains how SLEEVE differs from these methods.

3.1 Who Guards the Gate? Human- vs. Robot-Gated Learning

Human-gated and robot-gated methods mainly differ in how much human supervision they require. With human gating, the operator watches the robot continuously and can intervene immediately when needed. It also means, however, that the human cannot easily supervise several training runs at once, or do other work alongside [12].

Examples of human-gated methods include HG-DAGger (human-gated DAGger) [34] for imitation learning and HIL-SERL [4] for reinforcement learning. In HG-DAGger the human intervenes during rollouts of a frozen policy. Those interventions are added to the dataset, and the policy is fine-tuned on the expanded dataset afterwards. Only the human’s corrective behaviour is added to the dataset, so the policy is trained to recover from failure states without being trained to reproduce the actions that led to them. In HIL-SERL, by contrast, rollouts and training happen in parallel. Both policy actions and human interventions go into the training buffer, because reinforcement learning can learn from both successful and unsuccessful behaviour.

Robot-gated algorithms aim to reduce human burden by signalling when an intervention may be needed. Rather than a human watching for signs that the policy is about to fail, the robot itself decides when it needs help and triggers the intervention. The next section describes the criteria used to decide when to trigger an intervention request.

3.2 What Robot-Gated Methods Measure

This section groups robot-gated methods by the signal they use to decide when help is needed: policy uncertainty, disagreement with the expert, risk, and progress toward the goal. The groups are not exclusive; ThriftyDagger, for instance, combines an uncertainty measure with a risk estimate. In most of these methods the trigger transfers control to an expert, as defined in Section 2.4, which need not be a human. Other methods use the trigger only to request a reset or to flag that the robot has failed.

3.2.1 Uncertainty and Novelty

In pure imitation learning, several approaches evaluate policy uncertainty or measure how far the current state lies outside the training distribution. EnsembleDagger [36], Diff-Dagger [37], and RND-Dagger [38] each do so in a different way.

In EnsembleDagger, an ensemble of policies is used. At each environment step, the disagreement among the policies is calculated. If disagreement is high, human help is requested. Intuitively, this metric measures what the authors call the policy’s ‘doubt’. Higher disagreement indicates that the policy is less reliable. In practice, EnsembleDagger combines the policy’s doubt metric with a discrepancy rule from SafeDagger (Section 3.2.2), and requests help if either signal crosses its threshold. SafeDagger trains a separate classifier to predict whether the policy is likely to deviate from the expert. EnsembleDagger instead queries the expert directly at every step and compares the expert’s action with the policy’s.

Diff-Dagger uses the training loss of a diffusion policy as its uncertainty measure. A high loss means the policy has encountered a state it did not see during training, and help is requested. Unlike EnsembleDagger, Diff-Dagger shows that it can separate out-of-distribution (OOD) states from multimodal in-distribution ones.

RND-Dagger triggers expert intervention using a state novelty measure derived from Random Network Distillation (RND). Two networks are used: a target network with fixed random weights and a predictor network. Both take the current state as input. The predictor learns to match the target’s output on the states the agent has visited,

while the target itself is not updated. The prediction error is therefore low on frequently visited states and high on less-visited ones, giving a state-based out-of-distribution score.

EnsembleDagger, Diff-Dagger, and RND-Dagger are examples of uncertainty and novelty metrics for pure imitation learning. However, there are also works that propose such metrics for reinforcement learning (RL). For instance, Active Reinforcement Learning with Demonstration (ARLD) [39] asks the agent to request a demonstration at the states it is uncertain about, and derives that uncertainty from its own Q-value estimates. ARLD proposes two such measures. The first is the Jensen–Shannon divergence between the softmax policies of the heads of a bootstrapped deep Q-network (DQN). The second is the predictive variance of a noisy DQN. An extension to continuous action spaces [40] replaces these discrete-action measures with the standard deviation output from the actor network of Soft Actor-Critic (SAC). EARLY (Episodic Active Learning from demonstration querY) [41] proposes evaluating uncertainty in a trajectory-based feature space. Specifically, EARLY estimates the uncertainty of a policy rollout by computing the average absolute temporal-difference (TD) error across all state-action pairs along the policy rollout trajectory. EARLY tracks the uncertainty of recent rollouts in a sliding window, and requests help when the uncertainty of a rollout falls in the top fraction of that window. The expert is then asked for a demonstration starting from the most uncertain initial state in that window, since EARLY treats the initial state of the rollout as a feature representing that rollout.

EnsembleDagger, Diff-Dagger, ARLD, and EARLY share a useful property: they do not train a separate model whose only job is to decide when help is needed, and instead reuse something the policy or its critic already produces. This idea inspired the approach described in this thesis.

3.2.2 Discrepancy from the Expert

A second family of methods measures the discrepancy between the robot’s action and the one the expert would take. Help is requested once the discrepancy exceeds a threshold.

SafeDagger [42] introduces a safety policy, which in this case is a simple classifier trained iteratively alongside the main policy. The safety policy learns from the discrepancy between the expert and robot policies. If the L2 distance between the expert’s and robot’s actions exceeds a threshold β_H , the robot’s action is labeled as unsafe. At execution time, the safety policy takes a state as input and outputs the probability that the robot’s action deviates from the expert’s, meaning the state is unsafe. This probability

is then thresholded to produce a binary safe/unsafe decision. If the decision is unsafe, SafeDagger requests an intervention.

LazyDagger [43] extends SafeDagger with a hysteresis band on the discrepancy, using the same upper threshold β_H and a new lower threshold β_R . Control is passed to the expert once the discrepancy exceeds the upper threshold β_H , and is returned to the robot only once the discrepancy falls below the lower threshold β_R . LazyDagger trains a classifier to decide when to switch control from the robot to the expert, as in SafeDagger, and uses the true expert action to decide when control should be returned to the robot. This hysteresis prevents rapid switching near the decision boundary, reducing context switches by roughly 60%. This yields fewer but longer interventions than SafeDagger.

Adaptive Intervention Mechanism (AIM) [44] replaces the binary classifier with a proxy Q-function over state-action pairs. This Q-function is trained with two losses. The first loss is a regression toward -1 on expert actions and toward $+1$ on the agent’s own actions, but only when the agent’s action differs from the expert’s by more than a fixed threshold. The second loss is a bootstrapped temporal-difference loss. Because of this second loss, an action can be flagged not only when it currently deviates from the expert but also when it leads to states where the agent is expected to deviate.

SafeDagger, LazyDagger, and AIM all treat deviation from the expert as evidence of failure. This assumption does not hold once the policy is optimizing return rather than optimizing similarity to the expert’s actions.

3.2.3 Risk and Safety

Other robot-gated approaches trigger help in risky states, or in states that lead to risky ones. Risky states are those from which the robot is unlikely to finish the task, or in which it may damage itself or its surroundings.

ThriftyDagger [12] proposes to combine the metric from EnsembleDagger with risk estimation. For risk estimation, a Q-function is trained on transitions from both the robot and the expert to estimate the discounted probability that the robot reaches the goal under its current policy. The risk is one minus that value, so it is high wherever the policy is unlikely to finish the task.

Recovery Reinforcement Learning (Recovery RL) [45] trains a Q-function that estimates the discounted probability of entering a constraint-violating state in the future, such as touching an obstacle or toppling a block. Rather than asking for human help, it trains an additional recovery policy that returns the robot to a safe state from which the main policy can resume.

Failure-Aware Offline-to-Online Reinforcement Learning (FARL) [46] has the same aim of avoiding risky states. It targets what the authors call ‘intervention-requiring failures,’ such as spilling water or knocking an object out of reach, which the robot is unable to undo on its own. FARL trains a world model that rolls the policy forward and estimates the discounted probability of a failure over the next few steps. If that probability exceeds a threshold, a recovery policy supplies the action instead. As in Recovery RL, the recovery policy is trained in advance and no human provides help.

Leave No Trace (LNT) [47] trains two policies in parallel. A forward policy performs the task, and a reset policy returns the environment to a starting state so the forward policy can begin a new episode. A state the reset policy cannot return from stands in for an irreversible one, and since the reset policy’s Q-value is low at such states, LNT reads that Q-function to keep the forward policy away from them. Each proposed forward action is executed only if its reset Q-value is above a threshold. Otherwise, the action is aborted and the reset policy takes control. The threshold is fixed but the reset policy keeps learning, so the set of permitted actions grows as recovery improves. The authors call this abort mechanism a learned, dynamic safety constraint.

Another example of predicting irreversible states is Proactive Agent Interventions (PAINT) [48]. PAINT trains the agent’s reversibility classifier to predict if the agent has entered an irreversible state. If it does, the human is asked to reset the environment. During the reset, the human also answers binary reversibility queries about the states visited by the robot since the previous human reset. These labels are used to train the classifier online and to penalize irreversible states in the critic update.

3.2.4 Progress Toward the Goal

The last group of robot-gated methods measures how much progress the robot is making toward the goal. This progress signal can be used to decide when to request help or just to detect robot failures. For example, Behavioural Cloning Value Approximation (BCVA) [49] trains a value function for a behaviour cloning policy and uses it to detect failures. BCVA asks for help once the value stays below a fixed threshold for a set number of consecutive frames. Training this value function required around 9,000 episodes of policy rollouts for the door-opening task.

Sentinel [50] detects task progression failures with a vision-language model (VLM). It queries GPT-4o with a prompt describing the task, the model’s role as a runtime monitor, and the images observed up to the current timestep. In their experiments, the monitor was queried only twice per episode, since each query to a closed-source VLM has both

latency and monetary cost. GPT-4o takes about 14s to respond on average. In addition to detecting task progression failures with a VLM, Sentinel uses statistical measures of temporal action consistency to detect erratic failures. The authors call failures erratic when the policy oscillates between modes. For example, the robot might move left and right to avoid an obstacle instead of committing to one direction.

AutoSERL [51] derives automated interventions from a single demonstration trajectory using two geometric triggers. A sliding-window mechanism detects deviation from the reference trajectory and intervenes when the distance exceeds a threshold, using a separate motion-planning module to move the robot back onto the reference. A safety-recovery mechanism detects stagnation. At each time step, AutoSERL computes the minimum point, which is the point on the reference trajectory whose end-effector pose has minimum distance to the robot’s current end-effector pose. If the distance between the current minimum point and the minimum point from l_{stag} steps ago is smaller than a threshold, AutoSERL detects stagnation and replays a predefined demonstration segment instead of executing policy actions. Both triggers are anchored to one pre-collected trajectory. The method assumes a hand-held object and a single target.

UniIntervene [52] is the closest of these methods to the approach that we take in this thesis. It trains a value-risk critic that, from the recent trend in value estimates and a future-conditioned action-value estimate, detects policy stagnation or degradation. When either is detected, a recovery policy takes over and returns the robot to a safe state, drawing on a memory of past intervention episodes. Both the triggering system and the recovery policy are pretrained before the main policy is trained, and both are kept frozen while the main policy trains. UniIntervene is combined with HIL-SERL, so the human still watches the robot and intervenes on unsafe or OOD states, but does so less often because the recovery policy handles some cases automatically.

Our approach differs from UniIntervene in three respects. First, UniIntervene assumes a separate dataset of failures and successes with which to train the value function, the triggering system, and the recovery policy, whereas we require no additional data and reuse quantities that the model already produces. Second, its intervention module is frozen during online training, so it cannot register that the main policy is improving, or that the policy has become better at recovering on its own, whereas ours updates as training proceeds. Third, UniIntervene gives a condition for triggering a recovery policy rather than for asking a human. The human must still watch throughout and step in when the recovery policy is triggered but fails to recover, or when the robot needs help but the recovery policy is not triggered. In our approach, the human intervenes only when the robot asks for help, and there is no separate recovery policy, though one could

be added. In UniIntervene’s reported results, the human intervenes 14.6% of the time and the recovery policy is triggered 34.6% of the time, so the policy is provided with help almost 50% of the time in total. This exceeds the 34.3% intervention rate of their pure human-gated baseline, where a human provides all the help. We instead give the policy time to recover on its own and request help only when it cannot.

3.3 Positioning

Table 3.1 summarizes every method surveyed in this chapter along five axes. It shows who guards the gate (human, robot, or both), whether the method is for imitation or reinforcement learning, what quantity is used to decide when to ask for help, whether a separate model is trained just for that decision, and who or what takes over once help is requested. The last row describes our method, SLEEVE (SLopE-based Evaluation of Value Estimates), which reads the trend of the critic’s own Q-values to decide when a robot needs help.

Most robot-gated methods train an extra model whose only job is to decide when to intervene. That model can be an out-of-distribution detector, a discrepancy predictor, a safety critic, or a progress model. EnsembleDAgger [36], Diff-DAgger [37], ARLD [39, 40], and EARLY [41] instead read a quantity the policy or its critic already produces. To our knowledge, no prior work uses the trend of the policy’s own critic to decide when to ask a human to intervene during robotic reinforcement learning. Two methods come close, on different axes. EARLY uses the temporal-difference error of its own critic to decide when to ask a human for help during reinforcement learning. It does not require training an extra model to make that decision—the idea that inspired our approach. EARLY triggers when a rollout’s error is among the highest of the last twenty rollouts, with the cut-off tuned per task. EARLY relies on an uncertainty metric computed from the TD error, which uses Q-values; we instead rely on a progress metric computed from the Q-values themselves. EARLY decides whether to intervene after an episode has finished. The expert it queries provides a fresh demonstration from a chosen starting state rather than taking control mid-episode, so it does not gate interventions in the sense of Section 2.4. UniIntervene [52] uses the recent trend in value estimates, but trains three additional models (the value function, the triggering system, and the recovery policy), triggers the recovery policy instead of a human, and still relies on human-gated interventions.

Table 3.1: Methods surveyed in this chapter, compared by how each decides when a robot needs help. *Gate* names who decides when to intervene. *Trained to gate* marks whether a separate model is trained just to decide when to intervene. *Who takes over* defines who or what performs the correction once triggered.

Method	Gate	IL/RL	Quantity used to decide when to intervene	Trained to gate?	Who takes over?
HG-DAGger [34]	human	IL	—	—	human
HIL-SERL [4]	human	RL	—	—	human
EnsembleDAGger [36]	robot	IL	ensemble disagreement + expert-robot action discrepancy	no	human
Diff-DAGger [37]	robot	IL	training loss of a diffusion policy	no	human
RND-DAGger [38]	robot	IL	RND prediction error	yes	human
ARLD (discrete) [39]	robot	RL	JS divergence across softmax policies of a bootstrapped DQN’s heads, or noisy-DQN predictive variance	no	human
ARLD (continuous) [40]	robot	RL	action standard deviation of a SAC actor	no	human
EARLY [41]	robot	RL	mean TD error over rollout	no	human
SafeDAGger [42]	robot	IL	expert-robot action discrepancy	yes	human
LazyDAGger [43]	robot	IL	same as SafeDAGger, with hysteresis (two thresholds)	yes	human
AIM [44]	robot	IL	Q-function predicting how much the policy’s future actions will deviate from the expert’s	yes	human
ThriftyDAGger [12]	robot	IL	ensemble disagreement + risk Q-function	yes (only risk Q)	human
Recovery RL [45]	robot	RL	Q-function estimating constraint-violation probability	yes	recovery policy
FARL [46]	robot	RL	world model estimating probability of failure over the next few steps	yes	recovery policy
PAINT [48]	robot	RL	reversibility classifier	yes	human (only to reset)
BCVA [49]	robot	IL	value function for a BC policy	yes	human
Sentinel [50]	robot	IL	VLM’s judgement of task progress + temporal action-consistency statistics	no	stops and reports failure
AutoSERL [51]	robot	RL	deviation from / stagnation along one reference trajectory	no	motion planner (deviation) / demonstration replay (stagnation)
UniIntervene [52]	both	RL	value-risk critic that detects stagnation or degradation	yes	recovery policy (when robot-gated) / human (when human-gated)
LNT [47]	robot	RL	Q-function of a reset policy	no	reset policy
SLEEVE (ours)	robot	RL	slope of a line fit to the critic’s last M Q-values, with Q-values evaluated at the policy’s actions	no	human

Chapter 4

SLEEVE: SLoPE-based Evaluation of Value Estimates

Chapter 3 surveyed how prior work decides when a robot needs help. This chapter proposes a criterion for making that decision during reinforcement learning (RL) fine-tuning. Section 4.1 defines SLEEVE (SLoPE-based Evaluation of Value Estimates), Section 4.2 states its hyperparameters, Section 4.3 gives the slope estimate, and Section 4.4 integrates SLEEVE with ConRFT (Section 2.3).

4.1 The Criterion

SLEEVE measures a policy’s progress from the Q-values of its actions, where the Q-values come from the critic. Thus, SLEEVE can be used for any actor-critic algorithm whose critic is a Q-network. Because the critic is continuously fitted online, its values reflect the current policy, so the signal remains up to date as training proceeds.

To mitigate noise in the Q-value signal, we fit a line to the last M Q-values and take its slope as the progress signal. Our intervention criterion is based on whether the slope is positive, that is, whether the policy is still making progress. The only assumption we make is that the Q-function outputs values with an upward trend for successful trajectories. Section 2.5 shows this holds whenever the goal earns a positive reward and other steps earn a non-positive one, or the goal earns a zero reward and other steps earn a negative one. We obtain such a Q-function with ConRFT (Section 2.3), and the intervention signal is available from the first episode, since ConRFT’s offline stage fits the critic on demonstrations.

Tracking whether the slope is positive rather than the raw Q-value also makes the criterion robust to the task’s value scale. The slope is unchanged when a constant is

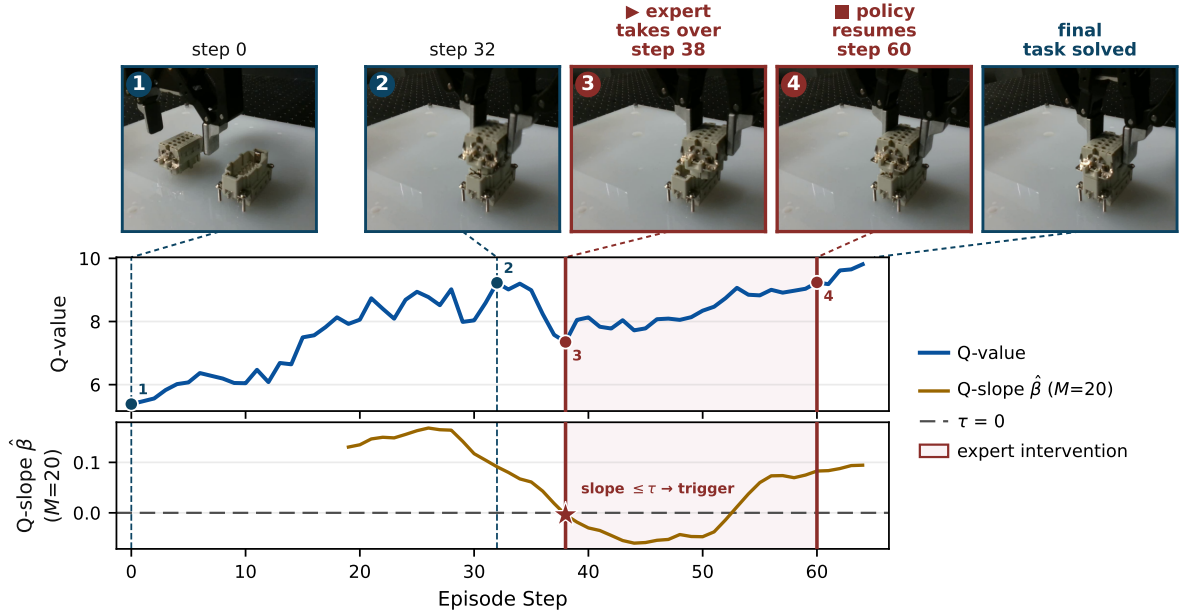


Figure 4.1: SLEEVE during one episode of robot-gated fine-tuning on the Waterproof Connector task. *Top*: frames from the side camera, numbered in the order they occur; the two highlighted frames are the step at which the expert takes over and the step at which the policy resumes, and the last frame is the terminal observation. *Middle*: the critic’s Q-value along the episode, with the numbered dots marking where each frame above falls on the curve. *Bottom*: the least-squares slope $\hat{\beta}$ fitted to the last M Q-values, and the threshold τ ; the star marks the first step at which $\hat{\beta} \leq \tau$, triggering the intervention. The shaded band spans the intervention in both plots.

added to every Q-value, so we never need to know which absolute Q-value to expect for a task, only whether the slope is positive, near zero, or negative (progress rising, flat, or falling). Scaling the Q-values by a positive constant also scales the slope, so slope magnitudes are not comparable across tasks with different reward scales. However, our intervention criterion depends only on the sign of the slope, and positive scaling preserves that sign. Figure 4.1 shows the criterion on one episode of a real fine-tuning run; the quantities it plots are defined in the sections that follow.

4.2 Hyperparameters

Our approach requires four hyperparameters: (1) the sliding window size M , (2) the slope threshold, (3) the minimum intervention duration, and (4) the fraction of Q-value recovery required to end an intervention. In practice, three of these can remain fixed across tasks:

- **Slope threshold** (τ) = 0: this lets the algorithm react to drops and stagnation in the Q-value, both of which indicate a lack of progress.
- **Minimum intervention duration** ($d_{\min}$) = 15 steps in simulation and 10 on the real robot, or 1.5s and 1.0s at our 10 Hz control rate. Both are long enough to be comfortable for a human expert. On the assembly tasks we run on the robot, a correction is usually a small motion, so fewer steps suffice.
- **Q-value recovery** (ρ) = 1: control returns to the policy once the Q-value of its action is at least $\rho Q_{\max}$, where $Q_{\max}$ is the highest Q-value the policy’s actions have reached so far in the episode. $\rho = 1$ demands full recovery to that peak. Setting ρ above 1 would demand recovery past that peak, which we avoid, as we still want the policy to explore rather than rely entirely on the expert.

The only parameter that needs tuning is the sliding window size M . A larger M smooths the slope estimate over more Q-values, so it is less likely to react to per-step noise but slower to detect a genuine stall; a smaller M reacts faster but is more easily triggered by a few noisy estimates. Because the window spans M control steps, M scales with the control rate: a faster controller produces more Q-value samples per second and therefore needs a larger M to look back over the same amount of time. We tune M empirically in Chapter 6.

4.3 Computing the Slope

At step t , let $W_t = (Q_{t-M+1}, \dots, Q_t)$ be the window of the M most recent Q-values, relabelled $Q_1, \dots, Q_M$ from oldest ($i = 1$) to newest ($i = M$). For the first $M - 1$ steps of an episode, this window is not yet full, so the criterion does not trigger an intervention; once M measurements are available, we fit a least-squares line to W_t and take its slope $\hat{\beta}(W_t)$ as the progress signal:

$$\hat{\beta}(W_t) = \frac{\sum_{i=1}^M (i - \bar{i}) (Q_i - \bar{Q})}{\sum_{i=1}^M (i - \bar{i})^2}, \quad \bar{i} = \frac{M+1}{2}, \quad \bar{Q} = \frac{1}{M} \sum_{i=1}^M Q_i, \quad (4.1)$$

where

- $i \in \{1, \dots, M\}$ indexes the window from oldest ($i = 1$) to newest ($i = M$);
- $\hat{\beta}(W_t)$ is the slope of the least-squares line fit to the window W_t . This is our progress signal, where a positive slope means the policy is making progress and a non-positive one means it has stalled;
- M is the sliding window size: the number of most recent Q-values the line is fit to;
- Q_i is the Q-value of the policy's action at the i -th step of the window, averaged over ConRFT's two critics;
- $\bar{i} = \frac{M+1}{2}$ is the mean index; and
- $\bar{Q} = \frac{1}{M} \sum_{i=1}^M Q_i$ is the mean Q-value over the window.

Because the steps are evenly spaced, Equation 4.1 simplifies: the $\bar{Q}$ term cancels (since $\sum_{i=1}^M (i - \bar{i}) = 0$), and the denominator is the constant $\sum_{i=1}^M (i - \bar{i})^2 = \frac{M(M^2-1)}{12}$, leaving

$$\hat{\beta}(W_t) = \frac{12}{M(M^2-1)} \sum_{i=1}^M \left(i - \frac{M+1}{2} \right) Q_i. \quad (4.2)$$

The slope is therefore a fixed weighted sum over the window, that is, a single dot product per step, with the weights $i - \frac{M+1}{2}$ and the normalizer computed once in advance.

4.4 Integration with ConRFT

We now fold this criterion into ConRFT; Figure 4.2 shows the ConRFT pipeline with SLEEVE integrated, and Algorithm 1 gives the full procedure. The offline stage (Cal-

ConRFT) is unchanged. During it, the actor and critic are pretrained on the demonstration dataset as described in Section 2.3.2. SLEEVE acts only in the online stage (HIL-ConRFT), making its interventions robot-gated rather than human-gated, while the expert still supplies the corrective action. An intervention starts when the slope is at or below τ , and ends once three conditions hold: the Q-value has returned to at least ρ times the highest value seen so far in the episode, the intervention has run for at least $d_{\min}$ steps, and the slope has climbed back above τ . Without that last condition, help would be requested again on the next step. The critic evaluates the policy’s action at every step, even while the expert is in control. Transitions collected autonomously go to the replay buffer $\mathcal{R}$, and those collected under intervention go to both the demonstration buffer $\mathcal{D}$ and $\mathcal{R}$, as in HIL-ConRFT.

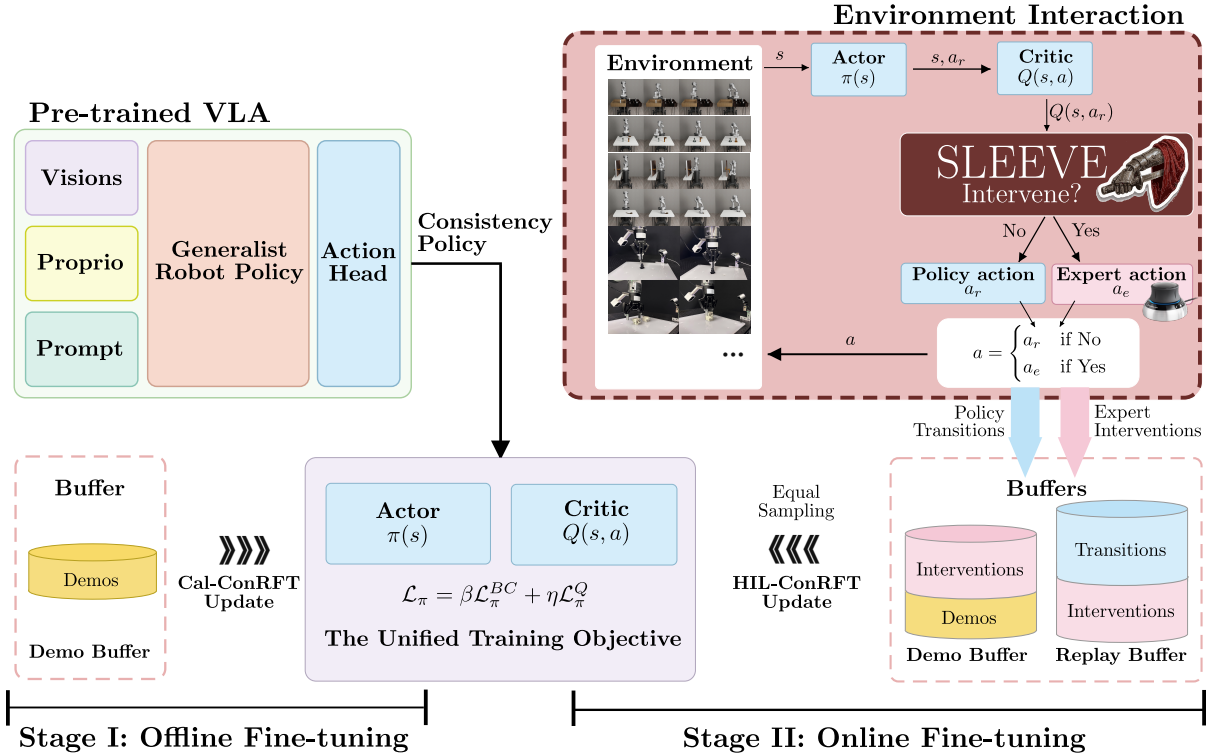


Figure 4.2: ConRFT’s two-stage pipeline with SLEEVE. The offline stage, Cal-ConRFT, fine-tunes the action head and the critic on pre-collected demonstrations alone. The online stage, HIL-ConRFT, trains on those demonstrations together with the robot’s own transitions and the expert’s corrections; SLEEVE decides when to request those corrections, so the stage runs robot-gated rather than human-gated. The layout follows ConRFT’s own overview figure [5].

Algorithm 1 Procedure of ConRFT [5] with SLEEVE robot-gated interventions. ConRFT’s two stages are offline (Cal-ConRFT) and online (HIL-ConRFT); both are defined in Section 2.3.

Require: A pretrained VLA model $\pi_{\phi,\psi}$ with VLA parameters ϕ and consistency head parameters ψ . A critic model Q with parameters θ . A pre-collected dataset $\mathcal{D}$ including 20–30 demonstrations. Initialize batch size B .

Randomly initialize the action head ψ and the critic model θ

Stage I: Offline fine-tuning with Cal-ConRFT

for each offline training step **do**

Sample (s_t, a_t, r_t, s_{t+1}) of size B from $\mathcal{D}$

Update the action head ψ and the critic model θ by Equations 2.16 and 2.17.

end for

Stage II: Online fine-tuning with HIL-ConRFT with SLEEVE

Start Policy Learning Thread:

Wait until the number of transitions in $\mathcal{R}$ is at least 100

for each online training step **do**

Sample (s_t, a_t, r_t, s_{t+1}) of size $\frac{B}{2}$ from $\mathcal{D}$ and $\mathcal{R}$

Combine both minibatches to form batch of size B

Update the action head ψ and the critic model θ by Equations 2.18 and 2.19.

end for

Start Interaction Thread

for each episode **do**

$W_t \leftarrow []$; $Q_{\max} \leftarrow -\infty$; $intervening \leftarrow \mathbf{false}$; $d \leftarrow 0 \triangleright W_t$ holds at most M values

for each interaction step **do**

$a_t \sim \pi_{\psi}(\cdot | s_t)$

$\triangleright$ sample the policy action

$q \leftarrow Q(s_t, a_t)$; append q to W_t (keep the last M)

$\triangleright$ evaluate the policy action

$recovered \leftarrow (q \geq \rho Q_{\max})$; $Q_{\max} \leftarrow \max(Q_{\max}, q)$

if $intervening$ **then**

if $d \geq d_{\min}$ **and** $recovered$ **and** $\hat{\beta}(W_t) > \tau$ **then**

$intervening \leftarrow \mathbf{false}$

$\triangleright$ stop intervening if all conditions are True

else

$d \leftarrow d + 1$

$\triangleright$ intervention continues

end if

else if $\text{len}(W_t) = M$ **and** $\hat{\beta}(W_t) \leq \tau$ **then**

$intervening \leftarrow \mathbf{true}$; $d \leftarrow 1$

$\triangleright$ trigger intervention: progress has stalled

end if

if $intervening$ **then**

Take action a_{intv}

Store transition $(s_t, a_{intv}, r_t, s_{t+1})$ in $\mathcal{D}$ and $\mathcal{R}$

else

Take action a_t

Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$

end if

end for

end for

Chapter 5

Experimental Design

In the previous chapter, we introduced SLEEVE and explained the method in detail. This chapter describes the experiments used to evaluate it. We consider two settings: four robosuite tasks in simulation, where a scripted expert provides the demonstrations and interventions, and two assembly tasks on a real robot, where a human provides them. Sections 5.1–5.3 describe the simulation setup, Sections 5.4–5.6 the real-world setup, and Sections 5.7–5.8 the baselines and metrics used in both settings. The next chapter presents and discusses the results of these experiments.

5.1 Motivation for Simulation Experiments

We build primarily on HIL-SERL [4] and ConRFT [5], both of which use human-gated interventions (with a human acting as the expert) and report results from real-world experiments only. The quality of human interventions is likely to vary based on factors such as fatigue, which makes a fair comparison between human-gated and robot-gated approaches difficult. Moreover, we want to run experiments across multiple tasks and seeds without requiring months of human operator time. We therefore set up four scripted experts in simulation, one per task, that provide both demonstrations and interventions for the trained policy. The same experts are used in the robot-gated and human-gated experiments, which removes intervention quality as a confounding factor between the two settings. The only difference is who triggers the intervention. In the human-gated setting, the human decides when to start and stop each intervention. In the robot-gated setting, the system runs fully autonomously: the robot-gated criterion is used to trigger interventions, and the scripted expert supplies the actions.

By a scripted expert, we mean a reactive, rule-based controller with privileged access to ground-truth state that the image-based policy cannot observe directly, such as the

poses of objects. At each step it selects the appropriate sub-goal directly from the current ground-truth state and outputs an action from a feedback controller, so it can take over from any state the policy reaches. For the Pick-Place task, described later in this chapter, the sub-goals are to approach the can, grasp it, lift it, move it over the target quadrant of the destination bin, lower it, and release the gripper. Diff-DAGger [37] also uses non-human experts in simulation, either an RL policy or a motion planner, but we chose not to use either for two reasons:

1. Because RL explores more widely than IL, the expert must be able to recover from a broad range of states. Training an RL expert would not guarantee coverage of all the necessary recovery behaviours, and such an expert may handle similar failure states in quite different ways. For our experiments, we instead want the expert to be as consistent as possible.
2. In our setting, a single episode may contain multiple interventions because our stopping criterion can end an intervention before the task is complete. In Diff-DAGger, by contrast, the expert finishes the entire task once an intervention begins. A motion planner would therefore have to be invoked many times per episode, increasing the running time of each experiment; querying a feedback controller, by contrast, is essentially instantaneous.

5.2 Adapting robosuite to a Real-World Setting

For our simulation experiments, we use the robosuite [35] simulation framework, specifically four of its single-arm tasks: Pick-Place, Nut Assembly, Door, and Wipe. robosuite was not originally designed for interactive real-world training, so we make several changes to better match the real-world setups used in HIL-SERL [4] and ConRFT [5]:

1. We change the camera placements (the side cameras in all tasks and the wrist camera in Wipe) and crop the images to remove uninformative regions from the observations.
2. We add a safety bounding box so that the robot does not explore regions irrelevant to the task.
3. We change the randomization of the initial robot pose; robosuite adds noise to each joint, whereas we add uniform per-axis noise to the initial end-effector reset pose, as in HIL-SERL and ConRFT.

4. We mostly keep the robosuite defaults for scene randomization. When we deviate, we note this in the corresponding task description.
5. We keep robosuite’s operational space control (OSC) pose controller and change only the action scale for certain environments.
6. We use a different maximum episode length and terminate episodes early in unrecoverable cases, including when a joint saturates, when the robot can no longer reach the object, or when the gripper has broken due to high contact forces. HIL-SERL [4] and ConRFT [5] allow the operator to terminate such episodes manually; we automate this instead. Each of these conditions could also be detected on real hardware: joint saturation and contact force could be measured directly, and an object that is out of reach could be detected with a pose tracker such as FoundationPose [53]. In practice, the last case may not require a tracker at all. In our real-world experiments, an out-of-reach object usually causes the system to request help anyway.
7. We add delays to enforce a 10 Hz control frequency, with an additional 0.6 s pause when closing the gripper. This matches the timing of our real-robot setup and provides a realistic reaction time for the human operator in the human-gated experiments.

The main goal of these changes is to keep each experiment short enough to run with human operators. Training a single policy requires only 20,000 to 30,000 environment steps. Each simulation experiment runs on two NVIDIA RTX A6000 GPUs, with the actor and learner¹ processes running on separate GPUs. The actor process runs the current policy (ConRFT’s actor model) to gather new trajectory rollouts. The learner process computes gradient updates for both ConRFT’s actor and critic models (Section 2.3).

The robot is the Franka Emika Panda arm with a Robotiq 2F-85 gripper. Every task uses the same control rate of 10 Hz and the same sparse reward, with +10 when an action completes the task successfully and -0.05 otherwise, following ConRFT [5].

The default action space consists of 6D delta end-effector actions plus a 1D gripper action. The default observation space consists of two RGB images (a 128×128 wrist-camera image and a 256×256 side-camera image), together with proprioceptive observations: end-effector pose (6D, with orientation as Euler angles), end-effector velocity (6D), gripper pose (1D), wrist force (3D), and wrist torque (3D). For the Pick-Place

¹The learner is not the critic. The critic is one of the networks used by the actor-critic algorithm, while the learner is the process that updates both the actor and critic networks.

and Wipe environments, these action and observation spaces are slightly modified, as described in the corresponding task sections.

5.3 Simulated Tasks and Scripted Experts

This section describes the four simulated tasks and their scripted experts. For each task, we describe the environment setup, the training parameters, and the sub-goals the expert follows. For some tasks, the expert also includes recovery behaviours for failure states encountered by the policy.

5.3.1 Pick-Place

In the Pick-Place task, the robot has to pick up an object from a source bin and place it in one of the four quadrants of the destination bin. We use the ‘Can’ version, so the robot must place the can in the quadrant assigned to it, while avoiding collisions. In this task we constrain rotation, so the action space is only 4D, and we remove end-effector orientation and angular velocity from the observations. Because rotation is constrained, we terminate the episode early when the gripper cannot recover without rotating, specifically when the can ends up too close to a bin wall to fit a finger between them or when it falls onto its side. The initial can location is randomized using the default robosuite [35] procedure. The language instruction given to the policy is “pick up the red can and place it in the far left container”. For the environment setup and training parameters, see Table 5.1.

Parameter	Value
Observation space	Default, with translation-only pose and velocity
Action space	4D (3D delta position and 1D gripper action)
Initial offline demonstrations	30
Maximum episode length	150
End-effector reset randomization	5 cm in x and y
Action scale	0.05 for translation
Total environment steps	30,000
Discount factor	0.98

Table 5.1: Environment setup and training parameters for the Pick-Place task.

The Pick-Place expert first picks up a can from the source bin, then places it in its assigned quadrant of the destination bin. To pick, the expert aligns the gripper over the can in xy with a P-controller, descends, and closes the gripper. When the can sits close

to a bin wall, grasping directly over the can’s centre would drive the open fingers into the wall, so it shifts the grasp slightly inward and takes the can a little off-centre. To place, the expert lifts the can to a safe height so that it clears the bin walls and the dividers between quadrants, moves sideways until it is over the target spot, and then lowers it at constant speed, releasing it once the can falls below a height threshold.

The expert also includes several recovery behaviours for common manipulation failures. If the gripper contacts a bin wall, it moves to break the contact. If the gripper closes without grasping the can, or closes against a bin wall, the expert re-opens it and retries.

5.3.2 Nut Assembly

In the Nut Assembly task, the robot has to pick up the nut, move it to the peg, and fit it onto the peg. We use the square-nut variant of this task. The initial nut location is randomized using the default robosuite [35] procedure, except that we reduce the nut’s yaw randomization from 360° to 180° . The language instruction given to the policy is “pick up the brown square nut and insert it onto the square peg.” For the environment setup and training parameters, see Table 5.2.

Parameter	Value
Observation space	Default
Action space	Default
Initial offline demonstrations	30
Maximum episode length	150
End-effector reset randomization	5 cm in x and y , and 30° in yaw
Action scale	0.05 for translation and 0.2 for rotation
Total environment steps	30,000
Discount factor	0.98

Table 5.2: Environment setup and training parameters for the Nut Assembly task.

The Nut Assembly expert first picks up the nut by its handle, then places it on the square peg. To pick, the expert aligns the gripper over the handle in xy and yaw with a P-controller so that the fingers can close across the flat handle. Once aligned, the expert descends under the same P-control and closes the gripper. To place, the expert lifts the nut straight up, without sideways motion, to a transport height above the top of the peg. The expert then moves sideways over the peg and rotates the nut under PID control until the square hole is centred above the square peg and aligned with it within about a degree. Once centred and aligned, the expert descends at constant speed while

correcting in xy and yaw, and releases the nut deep onto the peg, letting gravity seat it the rest of the way.

The expert also includes several recovery behaviours for common grasping failures. If the gripper holds the nut somewhere other than the handle, it drops the nut and re-approaches the handle. If the nut is near the workspace edge, the expert pulls it back inward. If the gripper closes without grasping the nut, the expert first lifts when needed to avoid knocking the nut, then re-opens the gripper and re-approaches the handle.

5.3.3 Door

In the Door task, the robot has to open a latched door by pressing down its spring-loaded horizontal handle and then pulling the door open. The handle springs back to rest if released, so the grip must be held throughout. The initial door location is randomized using the default robosuite [35] procedure. The language instruction given to the policy is “open door.” For the environment setup and training parameters, see Table 5.3.

Parameter	Value
Observation space	Default
Action space	Default
Initial offline demonstrations	20
Maximum episode length	150
End-effector reset randomization	4 cm in x and z , 5.7° in roll, and 14.3° in yaw
Action scale	0.05 for translation and 0.1 for rotation
Total environment steps	20,000
Discount factor	0.98

Table 5.3: Environment setup and training parameters for the Door task.

The Door expert first aligns the gripper over the handle with a P-controller, staying at a safe height so that it does not scrape the handle while aligning. Once the open fingers are aligned across the handle, the expert descends and closes the gripper. The expert then pushes the handle down along the grasp axis to rotate it and release the latch. At the same time, the wrist tracks the handle’s rotation with a P-controller on the tilt axis, up to a limit of 0.4 rad (about 23°). Once this limit is reached, the tilt command is set to zero and the wrist holds its angle while the downward push continues to rotate the handle under the closed grip. Once the latch is released, tilt tracking stops and the expert commands a pulling motion perpendicular to the handle to swing the door open.

The expert also includes several recovery behaviours. If the gripper makes unintended contact with the handle, door, or frame, the expert moves it to break contact. If the

wrist is rotated far from the orientation the task needs, the expert holds position while rotating the wrist back to the needed orientation, then continues the task. If the gripper closes without grasping the handle, the expert reopens it and re-approaches the handle. If the grasp is too close to the latch end of the handle to turn it effectively, the expert releases the handle and re-approaches it closer to the free end.

5.3.4 Wipe

In the Wipe task, the robot has to remove dirt markers from a table surface using a whiteboard eraser. At the start of each episode, 100 dirt markers are randomly distributed over the central 40% of the table’s width and length. We reduce this region from the default 60% used by robosuite [35] because, with only a single side camera, showing the full region would make the individual markers too small in the image. The markers are placed by sampling a start point uniformly and then following a correlated random walk. Any markers that fall outside the 40% region are resampled. In this task, we constrain end-effector rotation, and because there is no gripper to open or close, the action space becomes 3D. We retain end-effector orientation and angular velocity in the observations because they may still provide useful information during contact, but remove the gripper pose because a whiteboard eraser is attached in place of the gripper. The language instruction given to the policy is “wipe table surface with eraser.” For the environment setup and training parameters, see Table 5.4.

Parameter	Value
Observation space	Default, without gripper pose
Action space	3D (delta position only)
Initial offline demonstrations	30
Maximum episode length	150
End-effector reset randomization	2 cm in x and y
Action scale	0.05 for translation
Total environment steps	30,000
Discount factor	0.97

Table 5.4: Environment setup and training parameters for the Wipe task.

The Wipe expert removes the dirt markers one row at a time, following a lawnmower pattern across the table. In each cycle, the expert selects the rightmost row containing remaining dirt markers, moves the eraser over the near end of the row with a P-controller, then descends and presses it onto the table. Once contact is detected, the expert drags the eraser in a single straight pass along the row. The expert then lifts the eraser and

repeats the process until no dirt markers remain. To hold the eraser against the table during a pass, the expert uses the downward contact force from the wrist force sensor in an integral controller. At each step, the controller adjusts the downward command by an amount proportional to the difference between the measured force and a target of 45 N. The expert adjusts this command only while the eraser is in contact with the table.

5.4 Real-World Setup

Repeating real-robot experiments across multiple seeds is costly, and human actions cannot be reproduced exactly. Such experiments are nonetheless important because they show that SLEEVE works outside of simulation, where the signals it relies on are generally more noisy.

Our real-world setup is based on the setups used in HIL-SERL [4] and ConRFT [5]. The robot is a Franka Research 3 (FR3) arm with a Robotiq 2F-85 gripper. We use the impedance controller and safety bounding box from Sample-Efficient Robotic Reinforcement Learning (SERL) [54]. The controller reduces contact forces, while the box keeps the arm out of regions irrelevant to the task. The observation space matches our default simulation setup and consists of two images and a 19D proprioceptive vector containing end-effector pose, end-effector velocity, gripper pose, wrist force, and wrist torque. The images come from a wrist-mounted Intel RealSense D405 and a side-mounted Intel RealSense D435. We use only their colour streams. The default action space is a 6D delta end-effector pose plus a 1D gripper command (7D). For the USB Insertion task, which does not require grasping, we drop the gripper command, giving a 6D action space. The control rate is 10 Hz, and the reward is +10 when an action completes the task successfully and -0.05 otherwise. The actor and the learner both run on a single NVIDIA Quadro RTX 8000 GPU.

A human teleoperates the arm with a 3Dconnexion SpaceMouse to provide both the demonstrations before training and the corrective interventions during it.

We do not automate early termination after an unrecoverable failure. The only unrecoverable failure that occurred in our real-world experiments was an object being dropped out of reach. In the robot-gated setting, the system usually requested help in that case anyway, so the operator could end the episode at that point without watching between requests. In the human-gated setting, the operator already watches throughout training, so early termination requires no additional attention.

The scripted expert in simulation responds instantly, but a human needs time to react to each help request. To keep the comparison between the human-gated and robot-gated

settings fair, we pause both the actor and learner processes whenever the system requests help, so that training does not advance while waiting for the human. No gradient steps are taken while the system waits, so the update-to-data (UTD) ratio is not inflated by the human’s reaction time.

5.5 Real-World Tasks

In the real world, we test SLEEVE on two tasks: (1) inserting a pre-grasped USB connector into a port, and (2) picking up a waterproof connector and inserting it into a receptacle. Both tasks come from the National Institute of Standards and Technology (NIST) Task Board 1 setup [55]. For the USB task, we use a different physical USB port, but keep it fixed to the board in the same vertical orientation as in the NIST setup. Figure 5.1 shows each task after a reset and in its goal state. For each task, we describe the environment setup, the training parameters, and the scripted motions used to reset it between episodes.

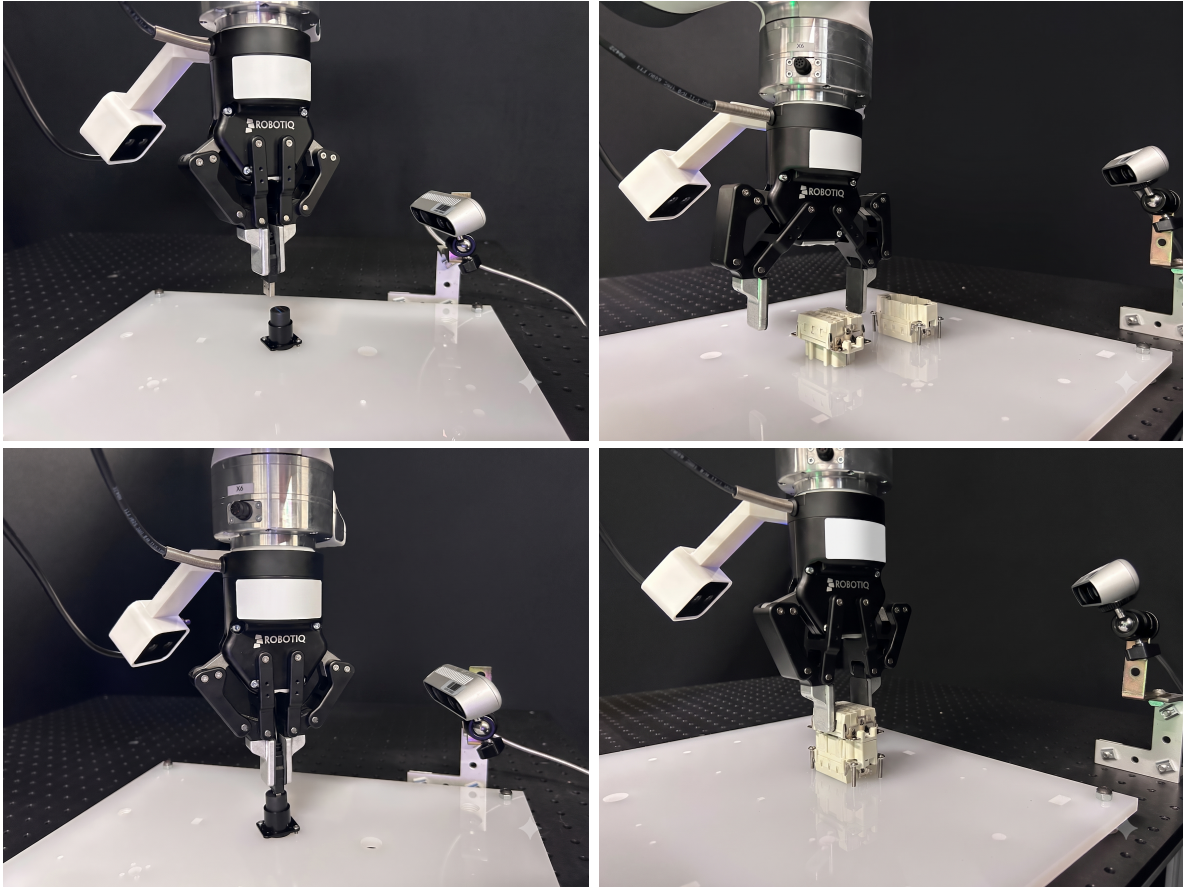
5.5.1 USB Insertion

In the USB Insertion task, the robot has to insert a pre-grasped USB connector into a port fixed to the board. The robot completes the task successfully when the connector is seated in the port. The language instruction given to the policy is “insert the usb into the receptacle”. For the environment setup and training parameters, see Table 5.5.

Parameter	Value
Observation space	Default
Action space	6D (no gripper action)
Initial offline demonstrations	20
Maximum episode length	100
End-effector reset randomization	2 cm in x and y , and 2.9° in yaw
Action scale	0.015 for translation and 0.1 for rotation
Total environment steps	8,000
Discount factor	0.97

Table 5.5: Environment setup and training parameters for the USB Insertion task.

To reset the USB Insertion environment, the robot first moves upward and then moves to a randomized end-effector pose.



(a) USB Insertion

(b) Waterproof Connector

Figure 5.1: The two real-world tasks. The top image in each column shows the scene after a reset, and the bottom image shows the goal state.

5.5.2 Waterproof Connector

In the Waterproof Connector task, the robot has to grasp the connector, move it to the receptacle, and complete the insertion. The task is completed successfully when the connector is seated in the receptacle. The language instruction given to the policy is “pick up the waterproof connector and insert it into the receptacle.” For the environment setup and training parameters, see Table 5.6.

Parameter	Value
Observation space	Default
Action space	Default
Initial offline demonstrations	30
Maximum episode length	200
End-effector reset randomization	2 cm in x and y , and 2.9° in yaw
Action scale	0.02 for translation and 0.2 for rotation
Total environment steps	8,000
Discount factor	0.98

Table 5.6: Environment setup and training parameters for the Waterproof Connector task.

To reset the Waterproof Connector task, the robot first opens the gripper without moving the arm. If the connector is already in the receptacle, it stays there. If the gripper is still holding the connector, opening the gripper releases it. The robot then moves directly upwards, clear of the receptacle. If the episode failed, the operator inserts the connector into the receptacle by hand before the robot re-grasps it. The robot then grasps the connector at the fixed receptacle pose, lifts it out vertically, and puts it back at a fixed location on the table for the policy to pick up. The policy does not always grasp the connector at its centre, so regrasping it at this fixed pose ensures that it starts each episode at the same location. The reset conditions are therefore the same for every method, allowing a fair comparison between them. The arm finally moves to a randomized end-effector pose, where the next episode begins.

5.6 Reward Classifier

Sections 5.4 and 5.5 described the real-world setup and tasks. This section explains how task success is determined. Unlike in simulation, where success can be read from the ground-truth state, the real-world reward has to be inferred from sensor data. Following SERL [54] and HIL-SERL [4], we train a binary classifier that predicts task success from

the camera images. The classifier uses the same camera images as the policy. The policy receives a reward of +10 when the classifier predicts successful task completion and -0.05 otherwise. The classifier consists of a pretrained ResNet-10 [31] feature extractor followed by a two-layer MLP. We train it using binary cross-entropy and Adam with a learning rate of 1×10^{-4} for 150 gradient steps.

We collect each classifier’s training set by teleoperating the arm through the task with the SpaceMouse and pressing the keyboard’s Space bar when the task is completed successfully. The frames captured at that point are labelled positive and the rest negative. This gives roughly 200 positive and 1,000 negative images per task. If a classifier exhibits systematic false positives or false negatives, we collect additional examples of those cases and retrain it. All of this happens before we collect the demonstrations, so the reward function remains fixed during training.

5.7 Baselines

Sections 5.1–5.6 covered the simulation and real-world settings. This section describes the baselines used to evaluate SLEEVE. We compare SLEEVE with a human-gated baseline, two random robot-gated baselines, two baselines that use the same trigger but never return control before the episode ends, and a baseline with no interventions. In simulation, all methods use the same scripted expert to provide intervention actions. The main difference between them is therefore when an intervention is triggered. We run three seeds for each robot-gated method and for each human operator in the human-gated baseline. In the real world, we compare SLEEVE with the human-gated baseline using a single seed per experiment. On the Waterproof Connector task, we also test SLEEVE + human release, where the operator may end an intervention early. If the operator does not, control returns when SLEEVE’s usual release rule triggers.

5.7.1 Human Baseline

The first baseline we compare with uses human-gated interventions. For this baseline, a wrapper around the environment checks whether the keyboard’s Space bar is pressed and switches control between the policy and the expert on each press. The human therefore does not provide the intervention actions, but only decides when each intervention starts and stops. This human-gated baseline corresponds to the HIL-ConRFT procedure [5] that SLEEVE builds on (Chapter 4).

Each operator is instructed to start an intervention when they think the policy will not be able to finish the task on its own, and to stop once they think the policy can finish from the current state. Before the recorded experiments, each operator is given time to become familiar with the environment and practice triggering several interventions.

We run the human-gated baseline with four operators, none of whom has seen how SLEEVE works. We first ask Human 1 to trigger interventions, with the scripted expert providing the actions, in all four environments. We then select the two environments with the largest performance difference between Human 1 and SLEEVE and ask Humans 2–4 to run experiments in those environments. This lets us measure how much the human-gated results depend on the operator.

5.7.2 Random Baselines

Both random baselines match the amount of expert help used by SLEEVE, including its intervention frequency and the distribution of intervention lengths, but choose when to intervene at random. These baselines test whether the timing of SLEEVE’s interventions matters. If SLEEVE outperforms a baseline that uses the same amount of expert help at random times, the difference comes from when the interventions occur rather than how much help is given. Both baselines use statistics measured from SLEEVE’s training. The first estimates them from the entire training history, whereas the second re-estimates them at regular intervals during training.

Empirical random robot-gated (ERRG): We count the interventions triggered during SLEEVE training and record the length of each one. The trigger probability is $\lambda = N/T$, where N is the number of interventions and T is the number of steps where the policy, not the expert, was in control. During ERRG training, an intervention starts at each policy-controlled step with probability λ . The length of each intervention is sampled uniformly with replacement from the recorded intervention lengths, and the intervention ends after that many steps or at the end of the episode, whichever comes first.

Empirical random robot-gated with curriculum (ERRG-C): ERRG-C is identical to ERRG, except that λ and the set of recorded intervention lengths are recomputed every 2,000 environment steps instead of using statistics from the full training history. This allows the amount of random intervention to decrease during training as it does for SLEEVE, rather than using a single average that gives too little help early and too much help later.

5.7.3 Complete-After-Start Intervention Baselines

The next two baselines use SLEEVE’s trigger rule but change when control returns to the policy. SLEEVE hands control back mid-episode once the Q-value has recovered to a fraction ρ of its episode peak, the intervention has run for at least $d_{\min}$ steps, and the slope has climbed back above τ (Chapter 4). The two baselines below never hand control back within an episode.

Complete-after-start intervention (CASI): CASI uses the same trigger as SLEEVE, but once an intervention starts, the expert completes the episode without handing control back. Each episode therefore contains at most one intervention. This matches Diff-Dagger [37], where the expert intervenes from the current state and completes the task, so control never returns to the policy within an episode. Because CASI and SLEEVE use the same trigger, any difference between them comes from when control returns to the policy. This lets us test whether returning control before the end of the episode improves learning.

Complete-after-start intervention with a budget (CASI-B): CASI-B is identical to CASI, except that the total number of intervention steps is capped at the number of expert steps SLEEVE used on the same task. We report these caps in the Intervention Steps column of Table 6.2. Once the budget is exhausted, the expert is no longer available and the policy trains on its own for the remainder of training. Because each CASI intervention runs to the end of the episode, CASI generally uses more expert steps than SLEEVE. CASI-B eliminates this difference by limiting both methods to the same number of expert steps.

5.7.4 No-Intervention Baseline

The last baseline receives no interventions at all. After the offline Cal-ConRFT stage, the policy fine-tunes with reinforcement learning on its own. This is HIL-ConRFT [5] with the interventions removed. Nothing else changes. We compare SLEEVE and the other baselines with this condition to determine whether expert interventions improve performance.

5.8 Metrics

During training, we compare SLEEVE and the baselines using success rate, trajectory length, intervention rate, cumulative intervention steps, and cumulative intervention count. Trajectory length is the number of environment steps in an episode, and the

intervention rate is the fraction of an episode’s steps on which the expert is in control. Cumulative intervention steps is the total number of steps taken by the expert so far, while cumulative intervention count is the number of interventions that have started. The two differ because a single intervention can last many steps. Following ConRFT [5] and HIL-SERL [4], we show success rate, intervention rate, and trajectory length as running averages over 20 episodes. We include all completed episodes in these averages, including failed episodes and those containing interventions. In simulation, each curve shows the mean over three seeds with a shaded standard-deviation band. The real-world plots show a single seed. We also report expert time, which measures how much of the total training time the expert spends controlling the robot.

After training, we evaluate the policy without interventions at checkpoints saved every 2,000 environment steps. In simulation, we roll out each checkpoint for 100 episodes per seed, and in the real world for 20 episodes. We evaluate every method from the same initial states. We report the success rate and, over the successful episodes only, the trajectory length.

Table 6.1 summarizes these metrics, and Chapter 6 uses them to compare the different query heuristics. Our main comparison is the trade-off between evaluation success rate and the number of intervention steps used during training. Intervention steps and intervention count together show whether a method uses many short interventions or fewer long ones. Trajectory length measures how efficiently the trained policy completes the task. Finally, the standard deviation across seeds measures how much performance varies between runs.

Chapter 6

Results and Discussion

Our simulation experiments first select the SLEEVE hyperparameters used in the remaining experiments, and then compare SLEEVE extensively with the human-gated baseline and test how intervention timing and length affect learning. We then compare SLEEVE with the human-gated baseline on the real robot. Chapter 5 described the tasks, baselines, and evaluation protocol. This chapter presents the results.

Section 5.8 defined the metrics we plot and how they are computed. Table 6.1 lists the metric abbreviations used in this chapter and the three additional metrics reported in Appendix A. The figures use the same layout throughout, with tasks in columns and metrics in rows, plotted against environment steps. When we compare SLEEVE with other baselines, the window size M used for each task is shown next to the task name.

6.1 Hyperparameter Search

In the hyperparameter search, we run SLEEVE with four window sizes $M \in \{5, 10, 20, 30\}$ in each simulation environment. The other three hyperparameters are held fixed at $\tau = 0$, $\rho = 1$, and $d_{\min} = 15$ steps, as described in Section 4.2. Figure 6.1 shows the results.

We observe that the larger windows work best: $M = 20$ for Pick-Place and Door, and $M = 30$ for Wipe and Nut Assembly. We also observe that $M = 5$ is too small. With such a short window, SLEEVE is more sensitive to noise in the Q-values, which results in a high intervention rate without improving the success rate. This problem is close to the noise sensitivity that motivated fitting a slope over a window rather than reading the per-step increment between consecutive Q-values as a progress signal (Section 2.5). The next larger window size, $M = 10$, is better than $M = 5$, but still requires more interventions than $M = 20$ and $M = 30$ while achieving the same or a lower evaluation success rate.

	Abbreviation	Description	How calculated
Reported in Chapter 6	Eval. Success (%)	Success rate at evaluation.	Measured every 2,000 environment steps over 100 (simulation) or 20 (real) episodes. Interventions are off during evaluation, and the same initial states are used when comparing different methods.
	Cumul. Interv. Steps	Cumulative intervention steps, which is also the number of expert data points used for training.	Running total of expert-controlled steps during training.
	Interv. Rate	Intervention rate.	Running average of the fraction of steps under expert control.
	Train Success (%)	Success rate during training, including episodes in which the expert intervened.	Running average of episode success.
Reported in Appendix A	Eval. Traj. Length	Trajectory length at evaluation, averaged over successful episodes only.	Same protocol as Eval. Success.
	Cumul. Interv. Count	Cumulative intervention count. Together with intervention steps, this indicates the average length of an intervention.	Running total of interventions started during training.
	Train Traj. Length	Trajectory length during training, including failed episodes and those with interventions.	Running average of trajectory length (steps per episode).

Table 6.1: Summary of the metrics of Section 5.8 and their abbreviations used in the figures of Chapter 6 and Appendix A. Interv. Rate, Train Success, and Train Traj. Length are running averages over a window of 20 episodes.

Comparing $M = 10$, $M = 20$, and $M = 30$ suggests that it is not always necessary to react immediately when the policy appears to have stalled. Using a larger window gives the policy more time to recover on its own, which is why SLEEVE needs fewer interventions for the same or a better evaluation success rate. However, this does not mean an arbitrarily large window is better. With a larger M , SLEEVE is also slower to detect a genuine stall (Section 4.2), so past some point a longer wait for help would outweigh the benefit of smoothing out noise and giving the policy a chance to recover.

In all subsequent experiments, each task uses the window size that performed best in the hyperparameter search. Section 7.2 additionally discusses adjusting M during training instead of fixing it for the whole run.

6.2 Comparison with the Human-Gated Baseline

We compare SLEEVE with the human-gated baseline, where the human operator intervenes at their own discretion. We first compare SLEEVE with one operator, Human 1, on all four simulation environments. We then compare against four operators on the two environments where the performance difference between SLEEVE and Human 1 is largest. We also report the expert time required by each gating method and measure how accurately each identifies when help is needed.

6.2.1 One Human Operator

We present the results of the comparison with Human 1 in Figure 6.2. SLEEVE outperforms Human 1 in three of the four environments, achieving higher or comparable success rates with fewer interventions. The exception is Wipe, where Human 1 achieves a higher success rate with fewer interventions.

We hypothesize that SLEEVE performs worse on Wipe because progress is less clearly reflected in the Q-values for this task. The scripted expert used for demonstrations and interventions first moves the end-effector to the rightmost uncleared marker closest to the robot base. It then wipes a row by moving the end-effector forward, lifts the eraser, and re-approaches the next set of uncleared markers. When the policy learns this strategy, the Q-values do not always indicate progress during the re-approach. The eraser may move upward and toward the next row for several steps without wiping any markers, making it difficult to determine from a single state-action pair how close the task is to completion. Pick-Place, Door, and Nut Assembly instead progress through a sequence of sub-goals that each occur once per episode. Their Q-values therefore provide a clearer

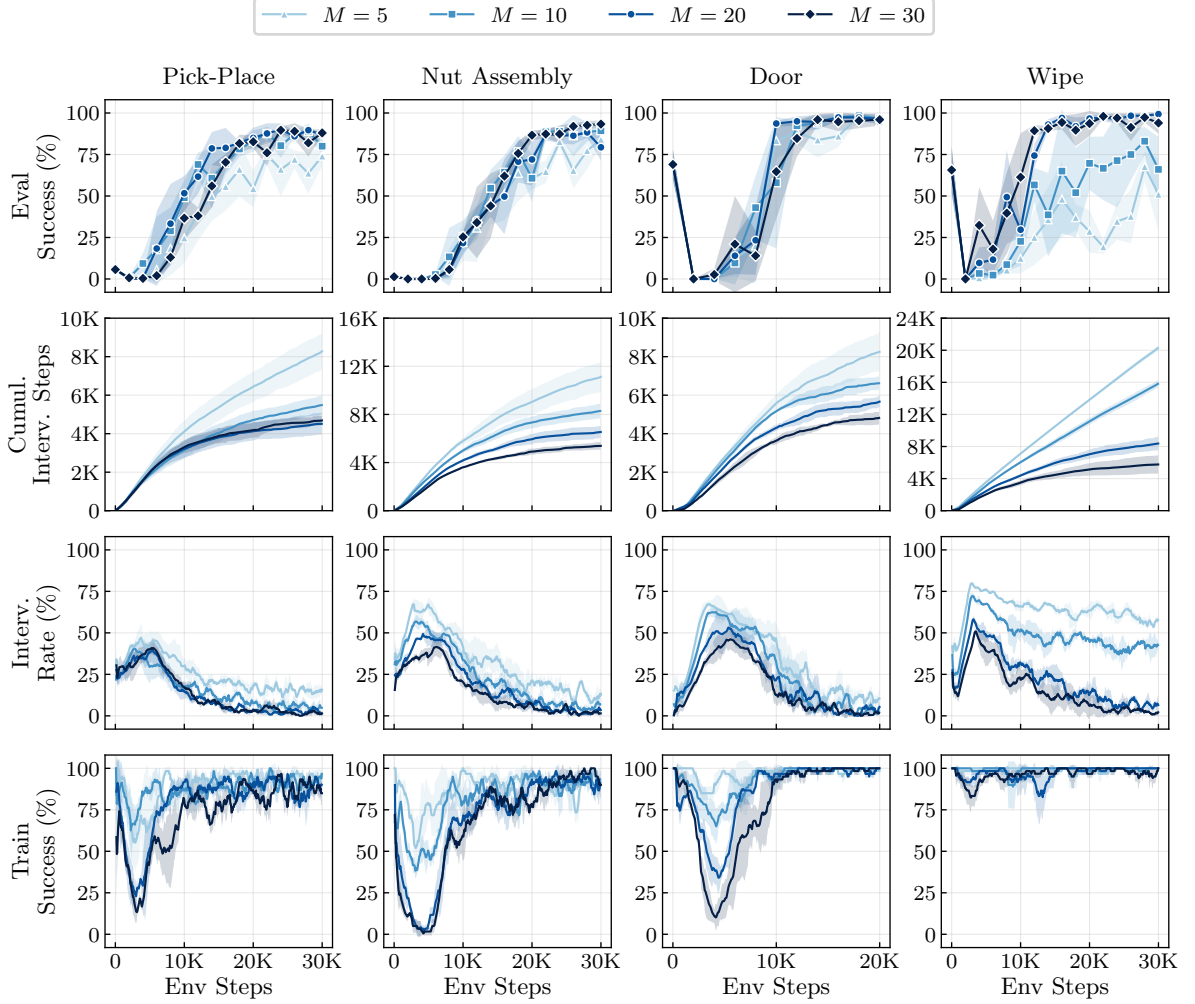


Figure 6.1: Hyperparameter search over the slope window M . Columns are simulation tasks and rows are metrics, all plotted against environment steps. Each curve is the mean over three seeds and the shaded band is one standard deviation. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training.

indication of how far the robot has progressed through the task. Wipe repeatedly returns to the same sub-goals, so the current state alone may not indicate how much of the table has already been cleared. In addition, we use only one side camera and one wrist camera, so some markers may be occluded, further affecting the Q-value estimates.

Addressing these limitations may require estimating progress from a history of observations rather than from the Q-value at a single step. Doing so would likely require an additional model, whereas SLEEVE uses only the critic already trained by the reinforcement learning algorithm.

6.2.2 Multiple Human Operators

We repeat the experiment with three more operators (Humans 2–4) on Pick-Place and Nut Assembly, and present the results in Figure 6.3. We observe that SLEEVE outperforms all four operators in both environments, achieving higher success rates with fewer interventions. Human operators adopt different intervention strategies, with some intervening earlier and some later, and some using longer interventions while others give only corrective nudges and let the policy explore. The same operator may also change strategy with experience, and the new strategy is not always better. Variation both between operators and across runs by the same operator produces a wider spread in performance than we see with SLEEVE.

6.2.3 Expert Time

Table 6.2 reports how much expert time SLEEVE and each human operator required, together with the duration of the training run. SLEEVE spends less expert time than any operator on Pick-Place and Nut Assembly, and less than Human 1 on Door; Wipe is again the exception.

The duration column shows the amount (in terms of time) of supervision required by the human-gated baseline. The operator must watch the entire training run, so the required supervision is the full run duration rather than only the time spent intervening. With robot gating, the operator is needed only during interventions, so the required supervision is the expert time.

6.2.4 Query Accuracy

In addition to comparing training cost and success rate, we measure how accurately SLEEVE and a human operator identify episodes in which intervention is needed, fol-

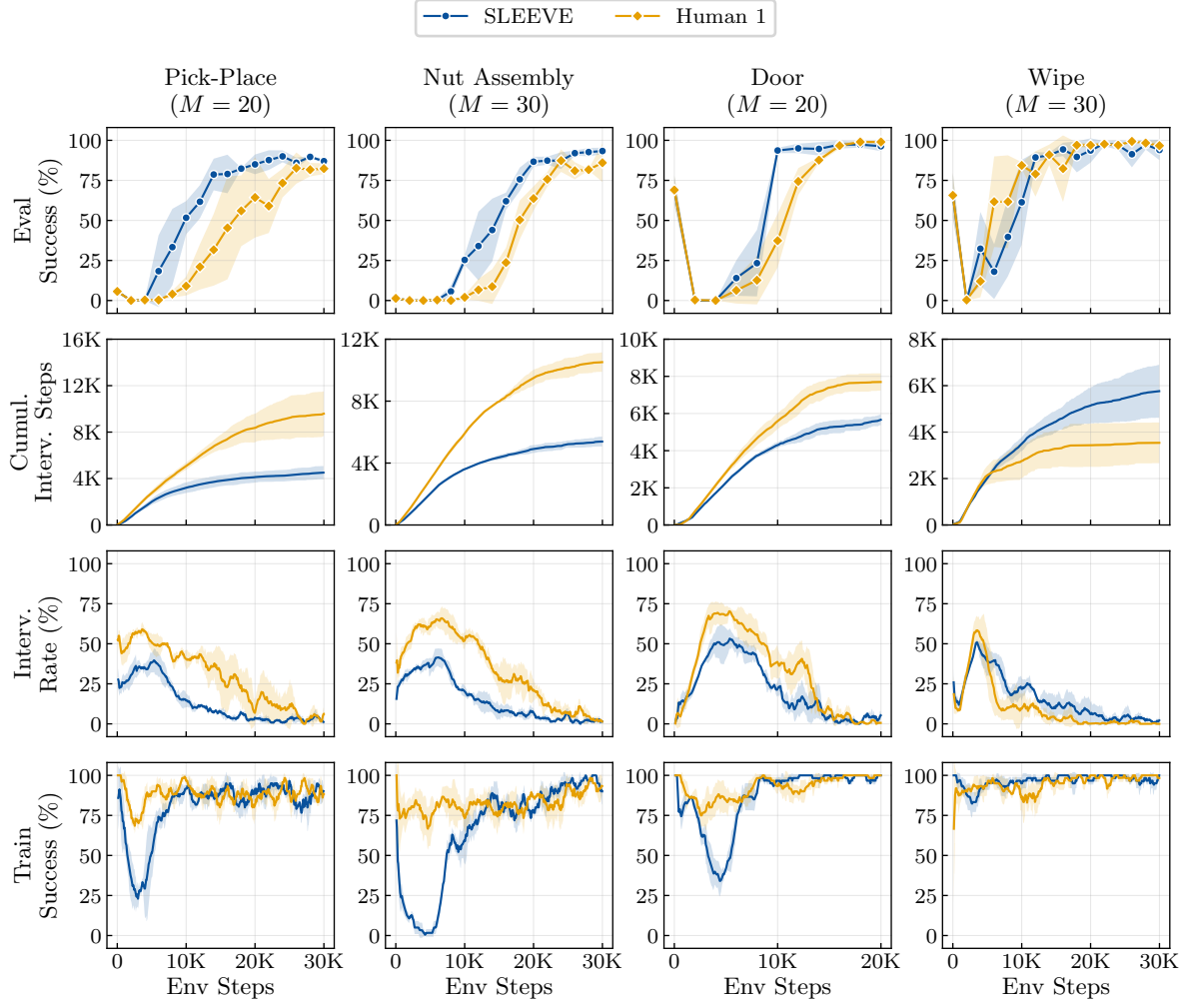


Figure 6.2: SLEEVE compared with Human 1. Columns are simulation tasks and rows are metrics, all plotted against environment steps. Each curve is the mean over three seeds and the shaded band is one standard deviation. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training. Each column title gives the window size M that SLEEVE uses for that task.

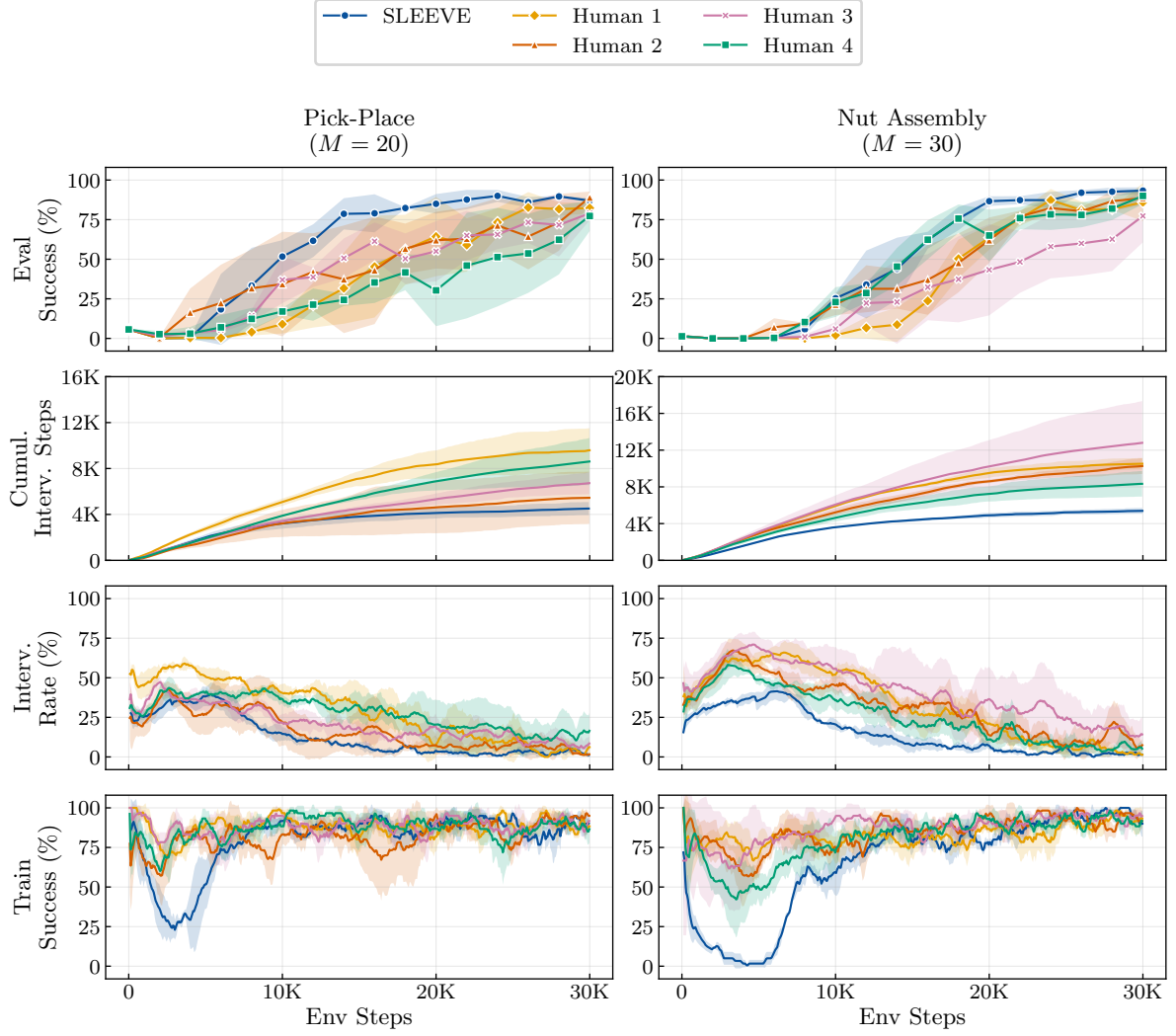


Figure 6.3: SLEEVE compared with all four operators, on the two environments with the largest performance difference between SLEEVE and Human 1. Columns are simulation tasks and rows are metrics, all plotted against environment steps. Each curve is the mean over three seeds and the shaded band is one standard deviation. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training. Each column title gives the window size M that SLEEVE uses for that task.

Task	Method	Intervention Steps $\downarrow$	Expert Time (min) $\downarrow$	Duration (min)
Pick-Place	SLEEVE ($M=20$)	4516 $\pm$ 568	12.8 $\pm$ 1.6	84.6 $\pm$ 0.3
	Human 1	9587 $\pm$ 1917	27.9 $\pm$ 5.8	87.3 $\pm$ 0.6
	Human 2	5447 $\pm$ 2274	17.1 $\pm$ 8.2	91.1 $\pm$ 6.8
	Human 3	6730 $\pm$ 943	19.0 $\pm$ 2.7	84.7 $\pm$ 0.2
	Human 4	8620 $\pm$ 2057	24.9 $\pm$ 6.2	86.2 $\pm$ 1.3
Nut Assembly	SLEEVE ($M=30$)	5387 $\pm$ 318	15.2 $\pm$ 0.9	84.6 $\pm$ 1.0
	Human 1	10518 $\pm$ 628	29.5 $\pm$ 1.5	84.2 $\pm$ 0.7
	Human 2	10269 $\pm$ 867	31.1 $\pm$ 2.8	90.8 $\pm$ 0.5
	Human 3	12803 $\pm$ 4537	36.6 $\pm$ 13.2	85.3 $\pm$ 1.0
	Human 4	8333 $\pm$ 1390	24.0 $\pm$ 4.1	86.4 $\pm$ 0.7
Door	SLEEVE ($M=20$)	5664 $\pm$ 287	16.1 $\pm$ 0.9	56.7 $\pm$ 0.4
	Human 1	7697 $\pm$ 466	21.2 $\pm$ 1.3	55.1 $\pm$ 0.8
Wipe	SLEEVE ($M=30$)	5764 $\pm$ 1152	13.5 $\pm$ 2.7	70.4 $\pm$ 0.0
	Human 1	3538 $\pm$ 876	8.4 $\pm$ 2.1	70.9 $\pm$ 0.0

Table 6.2: Expert effort for SLEEVE and each human operator. Duration is the wall-clock time from the start of the run to the end of its last recorded episode; expert time is that duration scaled by the fraction of steps on which the expert was in control. Each entry is the mean $\pm$ standard deviation over three seeds, and the lowest intervention steps and expert time in each task are shown in bold. SLEEVE is shown at the window size chosen for each task in Section 6.1. Humans 2–4 ran only Pick-Place and Nut Assembly.

lowing an evaluation similar to Diff-DAGger [37]. For each task and seed, we choose the checkpoint with an evaluation success rate closest to $\sim 50\%$. We select these checkpoints from the experiments with Human 1. We ask a human operator to watch 100 evaluation rollouts from each checkpoint and indicate whether they would intervene. The operator does not know the episode outcome or its final trajectory length. We compare these decisions with the actual episode outcomes and calculate the true positive rate (TPR), true negative rate (TNR), F1 score, and accuracy. We repeat the same evaluation for four SLEEVE window sizes and report the results in Table 6.3.

For this experiment, we use a different operator who did not participate in the training experiments. The operator receives the same instruction as the others: to step in when they think the policy will not be able to finish the task on its own. In this experiment, however, the operator does not take control and only indicates whether they would have intervened. The operator achieves accuracy and F1 scores comparable to SLEEVE at the larger window sizes.

The human operator does not achieve 100% accuracy or an F1 score of 100%. One source of error is an unrecoverable failure that occurs after the operator has already decided that no intervention is needed, such as an object being dropped out of the robot’s reach. Another source is slow progress. The policy may continue moving towards the goal but too slowly to finish within the episode limit, giving the operator little reason to intervene. False positives occur when an awkward or repeated attempt looks unlikely to succeed, but the policy later recovers without help.

6.3 Comparison with Random Baselines

Details on the random baselines are provided in Section 5.7.2. Figure 6.4 compares SLEEVE with the empirical random robot-gated (ERRG) and empirical random robot-gated with curriculum (ERRG-C) baselines. The random baselines use about 90% as many intervention steps as SLEEVE. An intervention may be cut short when an episode ends but is never extended, so the random baselines produce slightly shorter interventions on average than the recorded distribution.

Compared with SLEEVE, the random baselines improve more slowly through most of training. On some tasks, however, they match SLEEVE’s final success rate. Because the random baselines use a similar number of expert steps at randomly chosen times, the difference comes from intervention timing rather than the amount of expert help. Between the two baselines, ERRG-C is not uniformly better than ERRG. ERRG-C performs better at most checkpoints on Door, Pick-Place, and Wipe, but worse on Nut Assembly.

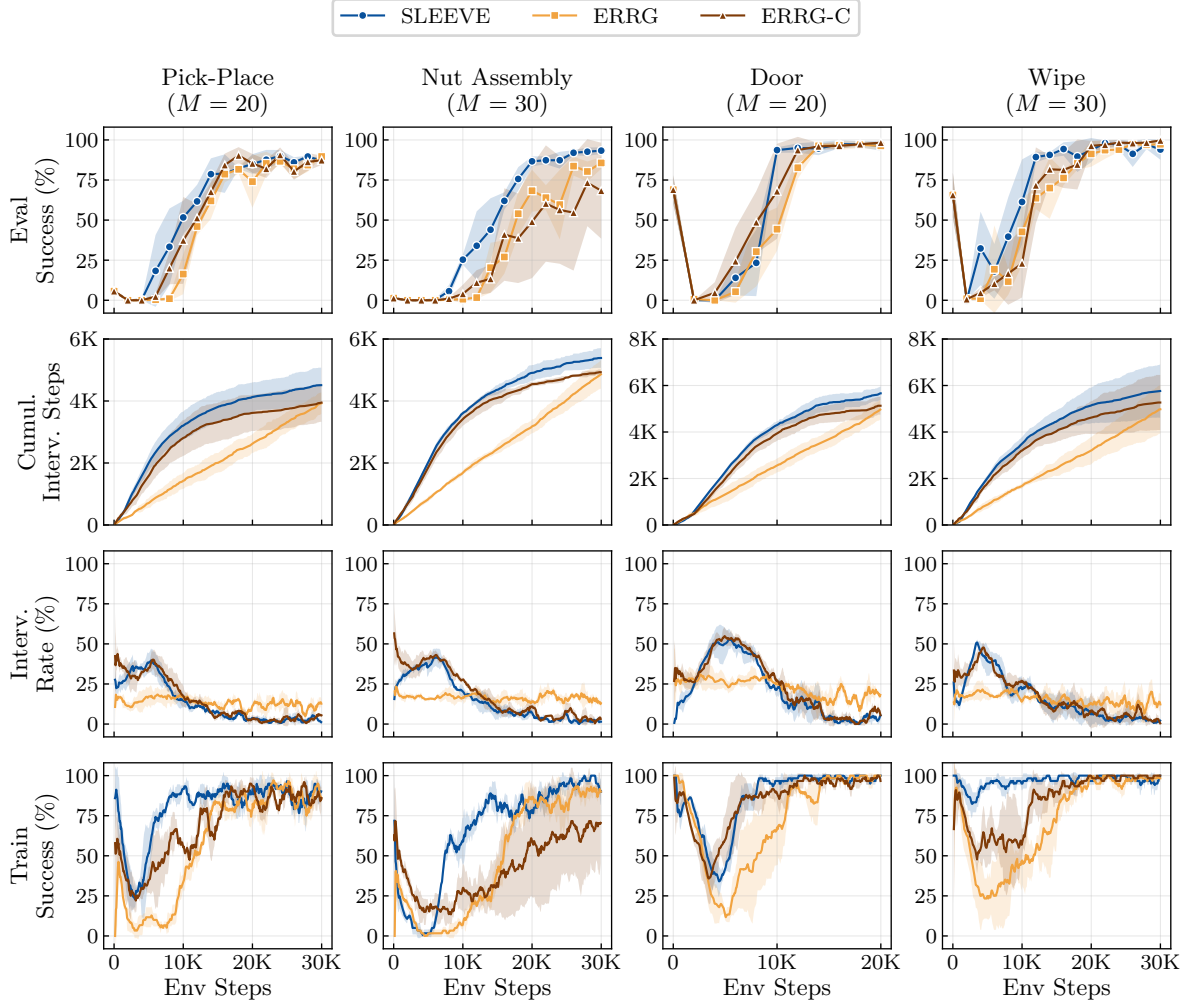


Figure 6.4: SLEEVE compared with the two random baselines, ERRG and ERRG-C, which reproduce the amount of expert help SLEEVE requests but choose when to intervene at random. Columns are simulation tasks and rows are metrics, all plotted against environment steps. Each curve is the mean over three seeds and the shaded band is one standard deviation. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training. Each column title gives the window size M that SLEEVE uses for that task.

Table 6.3: Q-slope query decisions vs. task outcomes on each task’s human reference policy at its $\sim 50\%$ -success checkpoint, sweeping the slope window $w \in \{5, 10, 20, 30\}$ over the same 100 rollouts per seed (mean $\pm$ std over 3 seeds). TPR/TNR are the fractions of failing episodes queried / successful episodes not queried; row H is the human baseline.

Task	Succ. (%)	w	TPR	TNR	F1 $\uparrow$	Accuracy $\uparrow$
PICKPLACE	45.7 ± 3.2	5	0.950 ± 0.024	0.322 ± 0.020	0.754 ± 0.032	0.663 ± 0.032
		10	0.863 ± 0.115	0.572 ± 0.063	0.773 ± 0.089	0.730 ± 0.089
		20	0.776 ± 0.091	0.766 ± 0.141	0.784 ± 0.103	0.770 ± 0.101
		30	0.705 ± 0.108	0.887 ± 0.084	0.779 ± 0.071	0.787 ± 0.049
		H	0.679 ± 0.142	0.908 ± 0.063	0.766 ± 0.107	0.783 ± 0.072
NUTASSEMBLY	50.0 ± 14.7	5	0.979 ± 0.037	0.154 ± 0.151	0.688 ± 0.106	0.577 ± 0.097
		10	0.952 ± 0.070	0.349 ± 0.210	0.730 ± 0.088	0.663 ± 0.072
		20	0.863 ± 0.066	0.611 ± 0.159	0.762 ± 0.088	0.743 ± 0.064
		30	0.791 ± 0.077	0.802 ± 0.079	0.782 ± 0.075	0.793 ± 0.035
		H	0.663 ± 0.118	0.840 ± 0.041	0.718 ± 0.104	0.760 ± 0.036
DOOR	44.0 ± 9.5	5	1.000 ± 0.000	0.062 ± 0.076	0.726 ± 0.087	0.583 ± 0.114
		10	1.000 ± 0.000	0.220 ± 0.195	0.760 ± 0.095	0.650 ± 0.131
		20	0.993 ± 0.012	0.602 ± 0.257	0.851 ± 0.103	0.810 ± 0.131
		30	0.929 ± 0.123	0.738 ± 0.153	0.859 ± 0.123	0.843 ± 0.121
		H	0.879 ± 0.209	0.931 ± 0.038	0.900 ± 0.123	0.913 ± 0.091
WIPE	47.0 ± 18.7	5	1.000 ± 0.000	0.006 ± 0.011	0.682 ± 0.153	0.533 ± 0.186
		10	0.965 ± 0.061	0.072 ± 0.076	0.682 ± 0.149	0.553 ± 0.167
		20	0.894 ± 0.184	0.238 ± 0.174	0.689 ± 0.152	0.613 ± 0.131
		30	0.894 ± 0.184	0.507 ± 0.174	0.764 ± 0.129	0.740 ± 0.098
		H	0.832 ± 0.136	0.915 ± 0.027	0.868 ± 0.091	0.883 ± 0.064

6.4 Ablations

We run two ablations on the Pick-Place and Nut Assembly environments. The first ablation keeps SLEEVE’s trigger rule and changes only when control returns to the policy. Instead of giving control back to the policy mid-episode, CASI and CASI-B let the expert finish the episode. CASI-B additionally caps the total number of expert steps. The second ablation, No Interventions, removes interventions altogether and fine-tunes the same pretrained checkpoint without the expert. This tests whether interventions are needed at all. Implementation details are given in Sections 5.7.3 and 5.7.4, and the results are shown in Figure 6.5.

SLEEVE reaches a higher success rate than CASI and CASI-B. These baselines improve more slowly than SLEEVE on both environments and have a lower success rate on Nut Assembly by the end of training. CASI uses 2.3 times as many expert steps as SLEEVE on Pick-Place and 1.7 times as many on Nut Assembly, while CASI-B uses

exactly the same number as SLEEVE on both environments. In the more complex Nut Assembly environment, learning largely stalls once CASI-B exhausts its intervention budget. In two of three seeds its evaluation success rate stays close to 0 until the end of training.

The No Interventions ablation shows that the effect of interventions depends on the environment. In Pick-Place, the No Interventions baseline converges to a success rate similar to SLEEVE’s, but its intermediate performance is significantly worse. In Nut Assembly, it barely improves on the pretrained policy, indicating that interventions are important for learning this task.

Together, the two ablations show that interventions can speed up training and that letting the expert finish the episode is less effective than returning control to the policy earlier. Both effects are larger in the more complex Nut Assembly environment.

6.5 Real-World Experiments

The simulation experiments evaluate SLEEVE systematically across multiple tasks, baselines, and random seeds, while the real-world experiments test whether Q-values computed from noisier real-world observations still provide a reliable signal for triggering help. In this section, we compare SLEEVE with the human-gated baseline on the USB Insertion and Waterproof Connector tasks. On the Waterproof Connector task, we also report SLEEVE + human release, where the operator may end an intervention early rather than waiting for SLEEVE’s release rule. Figure 6.6 shows the results.

SLEEVE reaches a success rate comparable to the human-gated baseline on both tasks, but differs in the expert effort required. On USB Insertion, SLEEVE uses fewer expert steps, 776 compared with 1,118. On Waterproof Connector, SLEEVE uses slightly more, 1,518 compared with 1,377, because one failure mode leads to long interventions. When the policy grasps the connector poorly and only discovers this while trying to insert it, the arm is already close to the receptacle, and the operator has to pull back, regrasp, transport the connector, and insert it. By this point, the episode has already reached a high Q-value, and the Q-value remains below that peak throughout the recovery. Control therefore does not return to the policy until the expert has almost completed the insertion. This failure mode also happens when a connector shifts or rotates inside the gripper due to contact. SLEEVE + human release lets the operator end the intervention once the connector has been grasped correctly. This reduces the number of expert steps to 1,302, slightly below the human-gated baseline. The expert-step counts are therefore similar, but the human-gated baseline requires much more continuous attention. Its operator

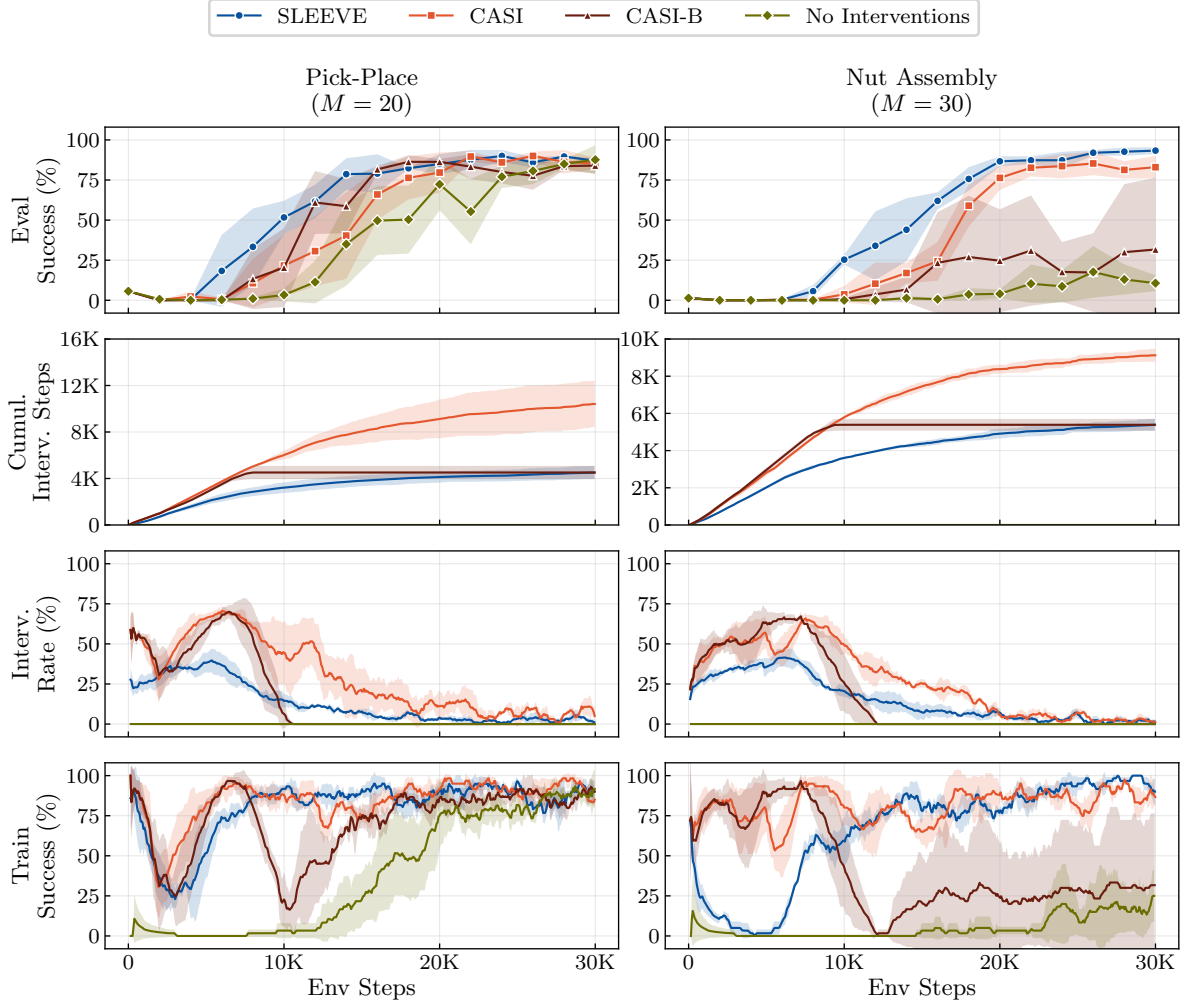


Figure 6.5: SLEEVE compared with CASI, CASI-B and a run with no interventions at all. Columns are simulation tasks and rows are metrics, all plotted against environment steps. Each curve is the mean over three seeds and the shaded band is one standard deviation. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training. Each column title gives the window size M that SLEEVE uses for that task. The no-intervention line is flat at zero on the intervention rows by construction, since its expert is never invoked.

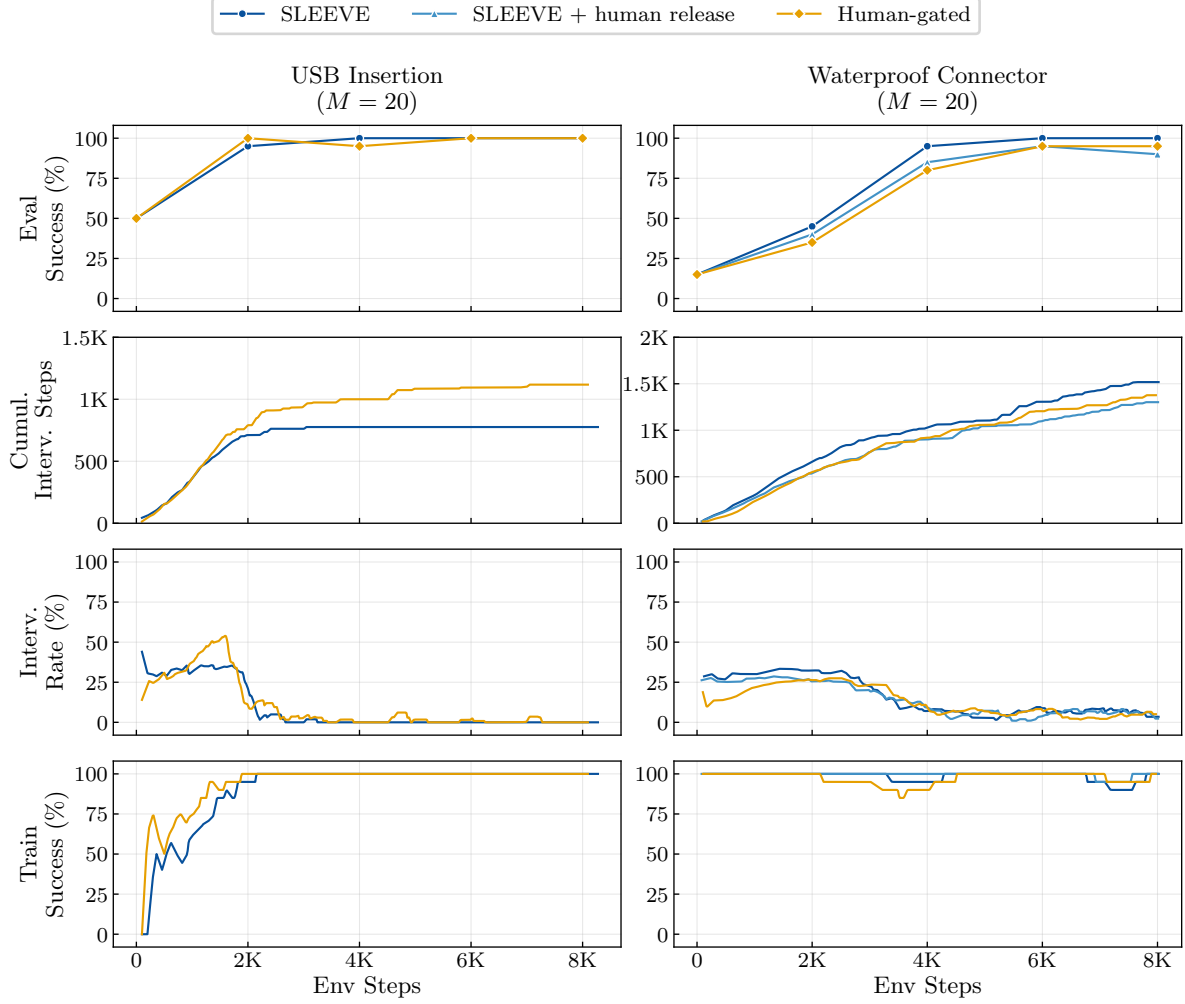


Figure 6.6: SLEEVE compared with the human-gated baseline on the two real-world tasks, with SLEEVE + human release shown on the Waterproof Connector task. Rows are metrics, plotted against environment steps, and each column title gives the window size M that SLEEVE uses for that task. Each line is a single run, so no variance band is shown. Eval Success is measured every 2,000 environment steps with interventions disabled, starting from the pretrained checkpoint at step 0; Interv. Rate and Train Success are running averages over 20 training episodes, and Cumul. Interv. Steps accumulates over training.

must watch every episode and decide when to intervene, whereas SLEEVE requires the operator only after the system requests help.

Resets account for much of the wall-clock time in the real-world experiments. The control loop runs at 5–5.5 Hz rather than the configured 10 Hz, so 8,000 environment steps take about 25 minutes on either task. On Waterproof Connector, the 0.6 s pause for each gripper closure adds another 1.5 to 2 minutes. A reset takes about 3 seconds on USB Insertion, where the arm only lifts and moves to a new pose, and about 16 seconds on Waterproof Connector, where the robot must also release the connector, regrasp it in the receptacle, and return it to the table. The Waterproof Connector runs longer episodes and therefore needs fewer resets, but each one takes much longer. A full training run takes about one hour on USB Insertion and about an hour and ten minutes on the Waterproof Connector.

Only a small part of that time is spent under expert control. Interventions take two to four minutes on USB Insertion and four to five minutes on the Waterproof Connector. These times include only the steps during which the expert controls the robot. The operator is present throughout in both settings, but only the human-gated baseline requires continuous attention. In that setting, the operator must watch the robot and decide when help is needed. These timing estimates are approximate because we logged only when each episode ended, not when it started. Each recorded interval therefore includes both the episode and the reset that preceded it. We estimate the two components by fitting the recorded durations against episode length.

Chapter 7

Conclusion and Future Work

Some operations cannot be automated by simply repeating a fixed motion and instead require engineering specific to each operation. At the high product variety described in Section 1.1, this engineering cost can make manual operation more practical than automation. Fine-tuning a policy with reinforcement learning (RL) from a small set of demonstrations can reduce the need for task-specific programming, but the policy must still explore on real hardware. To reduce that exploration, a human operator commonly intervenes to correct the robot. With human-gated intervention, the operator must watch the entire training run to decide when help is needed. Robot gating instead uses an automatic criterion to trigger help, allowing an operator to supervise several runs or do other work between requests.

7.1 Discussion

The objective of this research was to develop a robot-gated criterion for RL fine-tuning that uses quantities already computed by the learning algorithm. Our survey in Chapter 3 showed that most robot-gated criteria were developed for imitation learning and that few avoid training a separate model. We therefore proposed SLEEVE (SLopE-based Evaluation of Value Estimates), described in Chapter 4. SLEEVE estimates the rate of change of the most recent M Q-values and requests help when that rate becomes non-positive. Using the change in Q-values rather than their absolute level makes the criterion independent of the task’s value scale. The sign of a slope survives any shift or positive rescaling, so the slope threshold is always zero and only the window size M needs tuning. SLEEVE requires the critic’s Q-values to increase as the policy progresses toward the goal. That holds whenever a task’s reward is concentrated at the goal (Section 2.5). We build on ConRFT (Consistency-based Reinforced Fine-Tuning) [5], which pretrains such

a critic in its offline stage. We change only its online stage, making the interventions robot-gated rather than human-gated. The expert still provides the corrective actions, while SLEEVE determines when each intervention starts and stops. It requests help when the slope becomes non-positive and returns control once the Q-values recover and the slope becomes positive again, usually before the episode ends (Section 4.4).

To test SLEEVE, we ran experiments in four robosuite simulation environments while controlling for the quality of the intervention actions (Chapter 5). Because human corrections can vary between operators and across runs, we built a scripted expert for each task using ground-truth object poses. We used the same expert in both the human-gated and robot-gated settings, so that the two differed only in who decided when to intervene. Using scripted experts also made it practical to run three seeds for each setting. Compared with a human-gated baseline run by up to four different operators (Chapter 6), SLEEVE matched or outperformed every operator on three of the four tasks while using fewer interventions and less expert time. Wipe was the exception, for the reasons discussed in Section 7.2. In a separate experiment, an operator viewing recorded rollouts identified episodes that required intervention with accuracy comparable to SLEEVE at the larger window sizes (Section 6.2.4). The remaining baselines all learned more slowly than SLEEVE. These included two random baselines using a similar number of expert steps, two baselines that used the same trigger but did not return control before the end of the episode, and a baseline with no interventions. These results show that intervention timing matters, and that returning control to the policy before the end of the episode is more effective than having the expert finish the task.

On hardware we ran two National Institute of Standards and Technology (NIST) Task Board 1 assembly tasks with a Franka Research 3 arm. We compared SLEEVE with a human-gated baseline, and a human provided the interventions in both settings. SLEEVE achieved a comparable success rate on both tasks and used a similar number of expert steps, showing that Q-values computed from noisier real-world observations can still be used to trigger help. The main difference is the amount of supervision required. Human gating requires the operator to watch the entire training run, whereas SLEEVE requires the operator only during interventions.

7.2 Limitations and Potential Improvements

Finally, we discuss the limitations of SLEEVE and possible directions for improving it. The primary limitation is that SLEEVE depends on the Q-values produced by the underlying actor-critic algorithm. In our experiments, a useful signal is available from

the first episode because ConRFT’s offline stage trains the critic on demonstrations, producing Q-values that already increase along successful trajectories. Whether SLEEVE transfers to other algorithms depends on which part of the offline objective produces this trend: the calibrated conservative term, the consistency term that keeps the actions used in the Bellman backup in distribution, or both (Section 2.3.2). It is also unclear how quickly Q-values that increase along successful trajectories would emerge from online training without an offline stage. Before the Q-values provide a useful signal, the operator could trigger human-gated interventions. Early in training, an untrained policy is likely to require frequent help, so robot gating would provide little reduction in supervision. An ablation of the offline stage could address both questions. When the policy succeeds often enough for accuracy and F1 scores to be meaningful, the resulting intervention rule could be evaluated as in Section 6.2.4. Otherwise, the Q-value trajectories could be examined directly.

A second limitation is that SLEEVE can only detect progress that is reflected in the Q-values computed from individual state-action pairs. This limitation appears in the Wipe task. Wipe repeatedly returns to the same sub-goals rather than progressing through a fixed sequence toward a single goal. The policy clears one row of markers and then moves back to approach the next row. The critic does not always assign increasing Q-values during this re-approach, and occluded markers make the problem more difficult (Section 6.2.1). For tasks like Wipe, progress could instead be estimated from a sequence of observations, using either a recent window or the observations collected over the episode so far. This would normally require a separate model trained specifically to estimate progress. A vision-language model (VLM) could avoid the need to collect additional task-specific training data for this purpose. Sentinel [50], for example, uses a VLM to detect failures in task progression. Its reported mean query latency of 14s is much too slow for our 10 Hz control rate, but this approach may become more practical as inference speeds improve.

A third limitation is that SLEEVE does not include safety in its triggering criterion, so it may request help only after a constraint has already been violated. This is not a problem in our experiments because separate mechanisms already limit the robot’s motion. The impedance controller from SERL (Sample-Efficient Robotic Reinforcement Learning) [54] limits contact forces, while a safety bounding box prevents the arm from entering regions irrelevant to the task. More advanced safety requirements, such as detecting that the robot is approaching an unrecoverable state, could be handled by combining SLEEVE with a separate robot-gated risk measure. The system could request help when either SLEEVE detects stalled progress or the risk measure indicates that

a constraint is likely to be violated. An intervention would then begin when either condition is met. However, every safety criterion we surveyed requires a separate model, and some also require training data beyond the demonstrations used for the main policy (Section 3.2.3). Adding such a criterion would therefore require another model and, in some cases, additional training data.

The final limitation is that the window size M remains fixed throughout training. The choice of M trades reaction speed against robustness to noise. With a larger window, SLEEVE is less sensitive to noise but takes longer to detect that the policy has stalled (Section 4.2). In our experiments, M remains at 20 or 30 throughout training. Early in training, using a larger window reduces sensitivity to noise in the Q-values, while later it gives the policy more time to recover before SLEEVE requests help. In the real-world experiments, however, the operator sometimes observed that SLEEVE requested help slightly earlier or later than they would have. The window size could therefore be adjusted during training. For example, M could be changed directly by the operator, fitted using examples of when the operator would have intervened, or adjusted automatically using episodes that failed without an intervention and interventions that the operator judged to be premature.

Deciding when a learning policy needs help remains a difficult and relatively unexplored problem in RL. Even a human watching the robot cannot always determine whether the policy will recover on its own. We have shown that SLEEVE matches or outperforms human gating in all but one of the four simulated environments and in both real-world tasks. SLEEVE uses only Q-values from the critic already trained by the actor-critic algorithm, while requiring less supervision and a comparable or smaller number of interventions. We hope that SLEEVE and the evaluation procedure used in this thesis will be useful for future work on interactive robot learning.

Appendices

Appendix A

Additional Metrics

Chapter 6 plots four of the seven metrics from Section 5.8 for each experiment to keep the figures readable. This appendix reports all seven metrics for the same runs, adding evaluation trajectory length, cumulative intervention count, and training trajectory length.

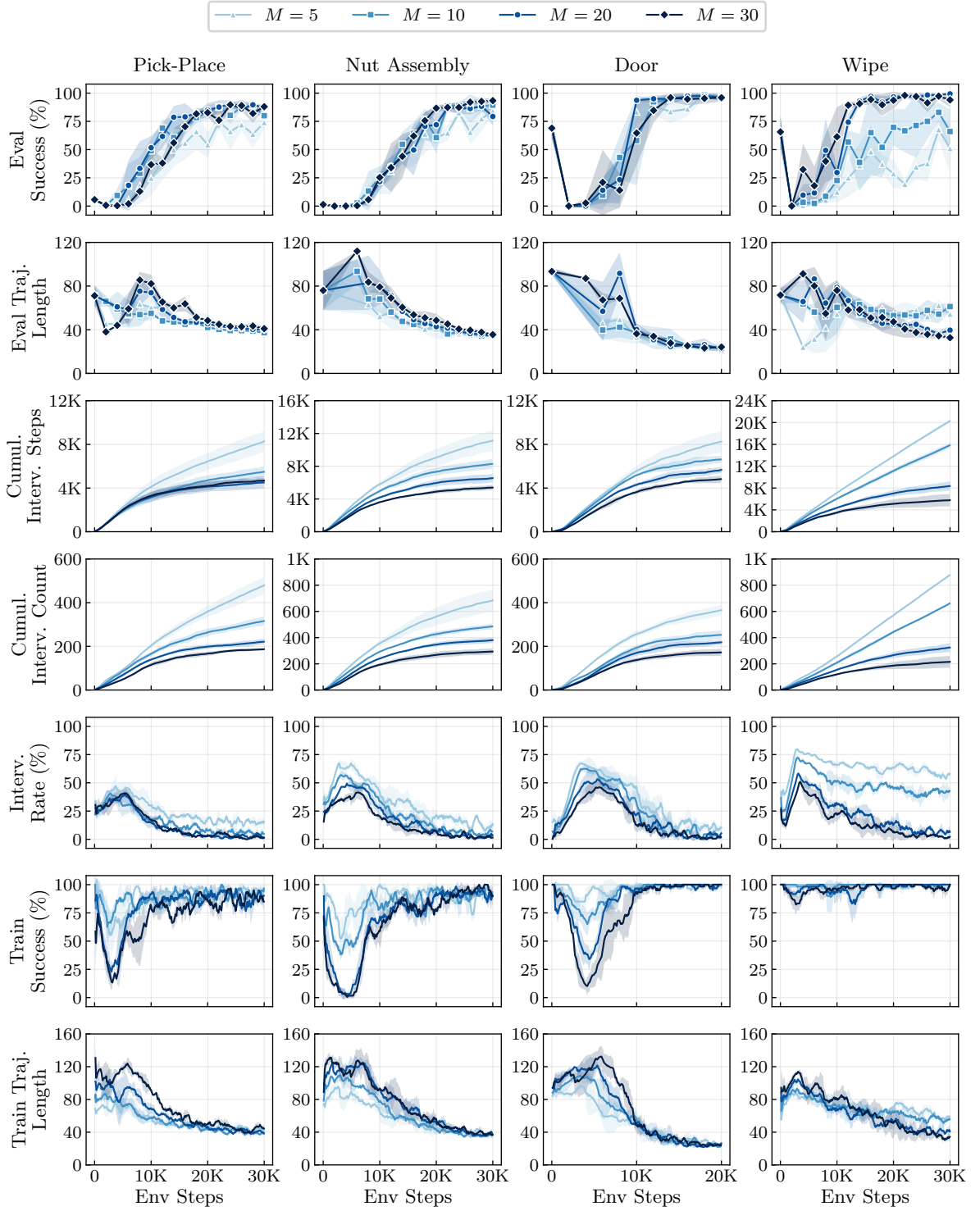


Figure A.1: Hyperparameter search over the slope window M , expanded from Figure 6.1 to all seven metrics.

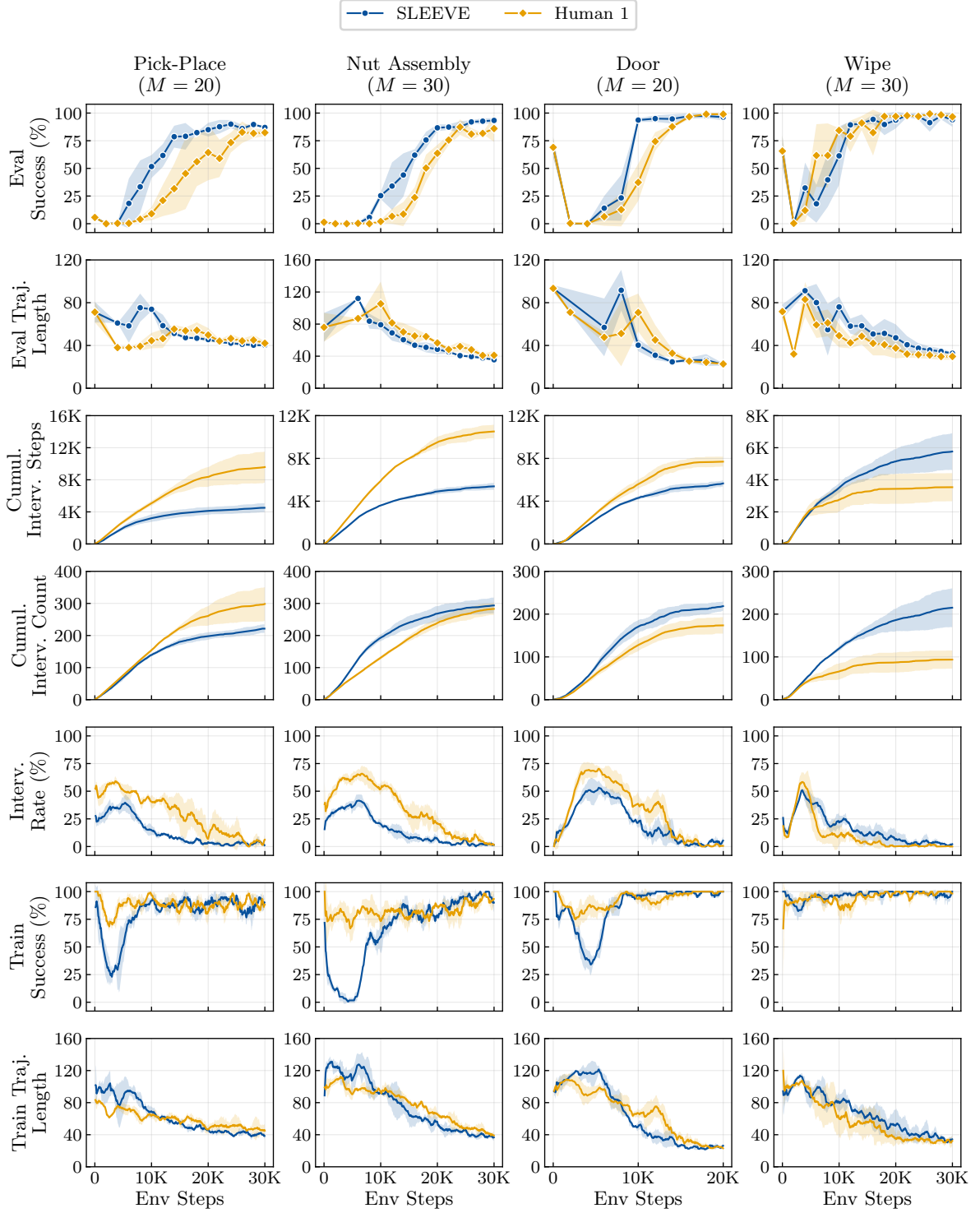


Figure A.2: SLEEVE compared with Human 1, expanded from Figure 6.2 to all seven metrics.

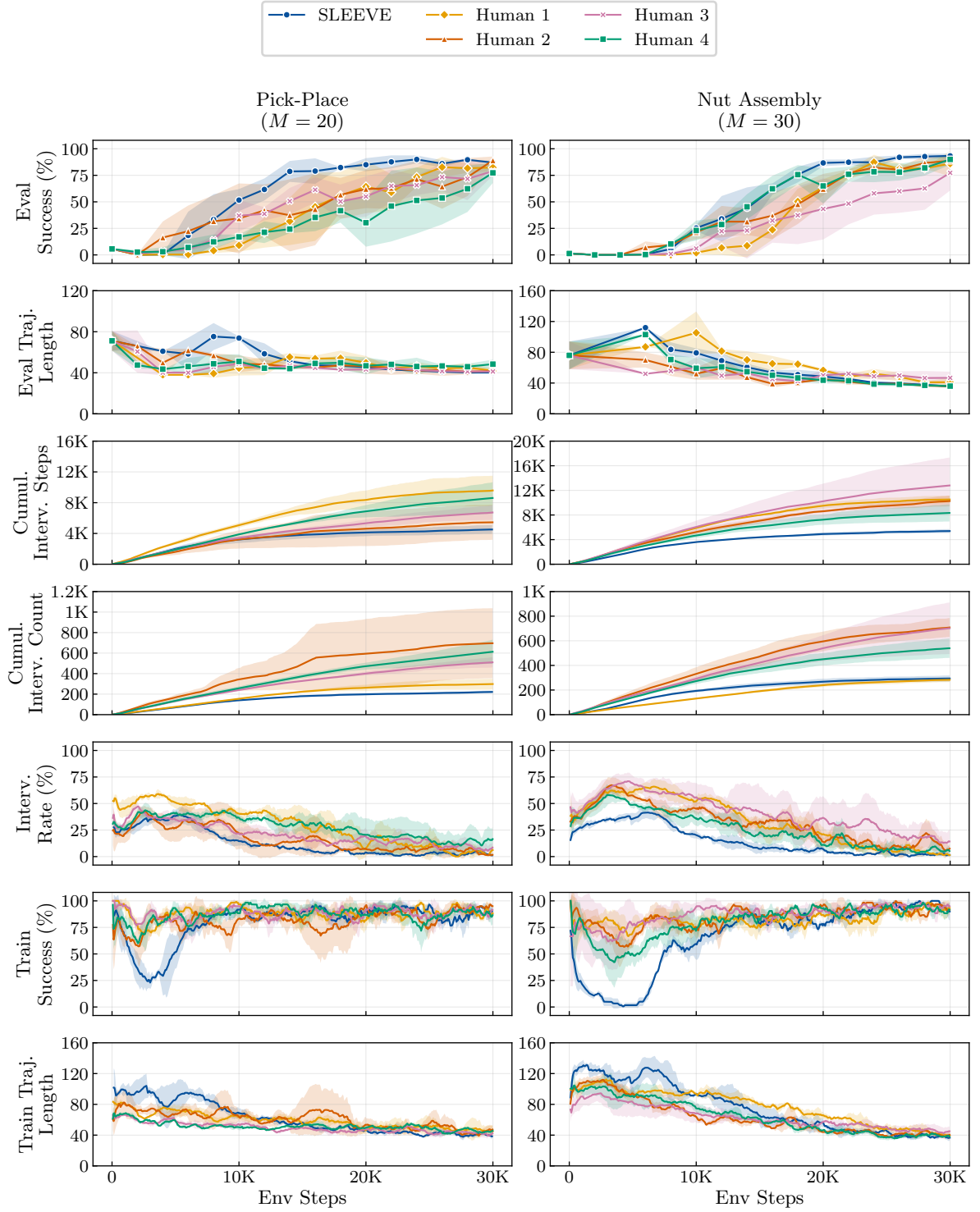


Figure A.3: SLEEVE compared with all four operators, expanded from Figure 6.3 to all seven metrics.

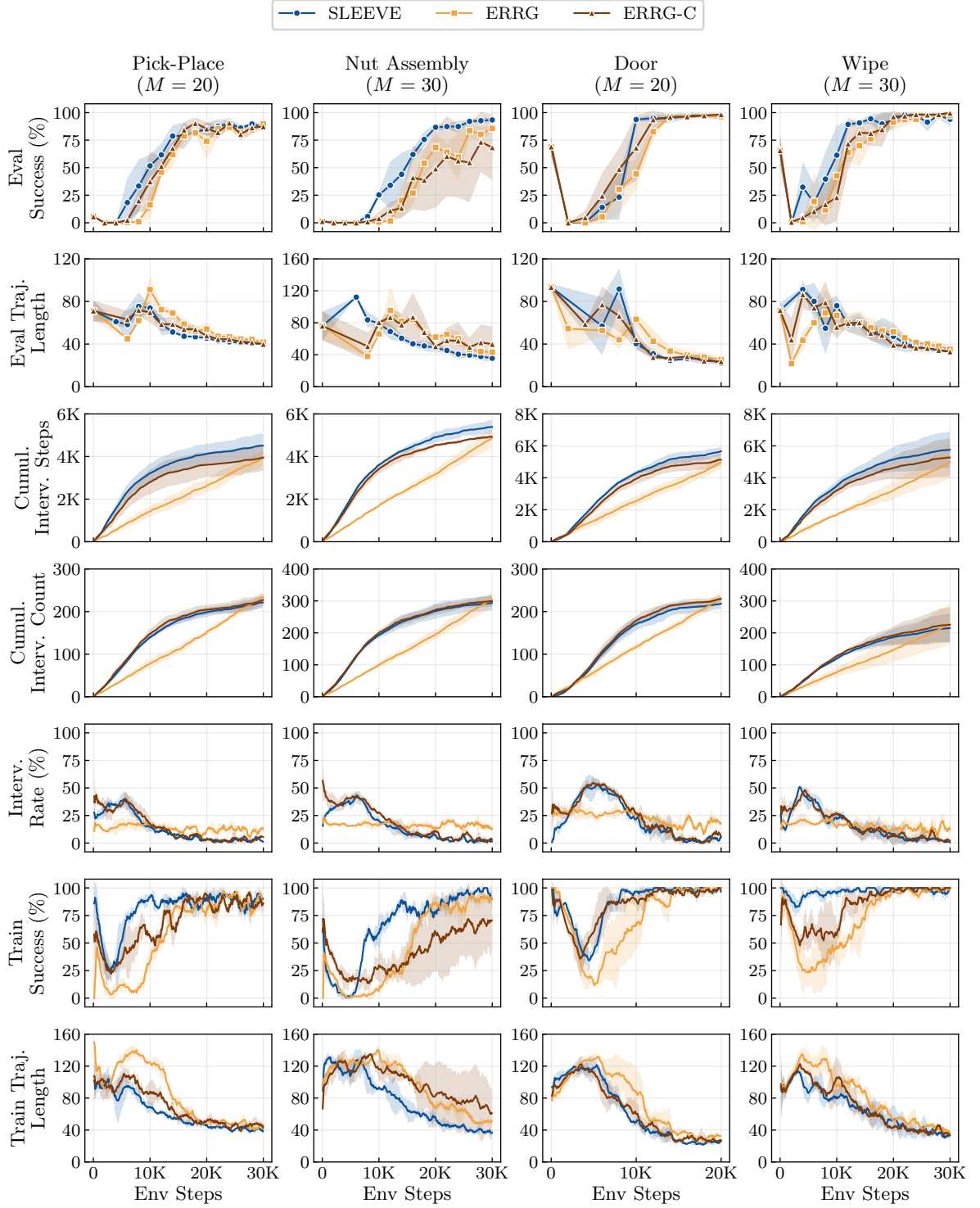


Figure A.4: SLEEVE compared with the random baselines ERRG and ERRG-C, expanded from Figure 6.4 to all seven metrics.

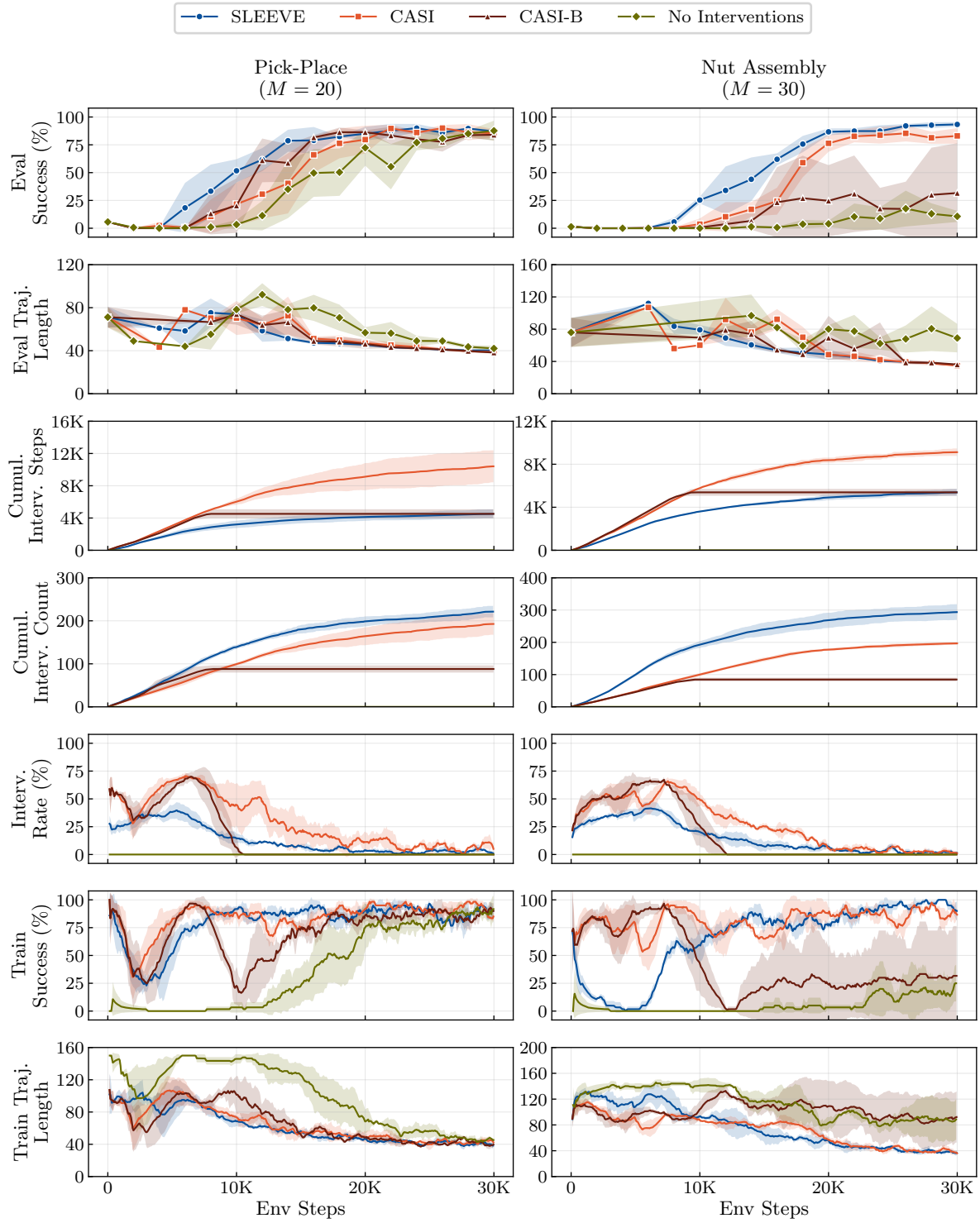


Figure A.5: SLEEVE compared with CASI, CASI-B and a run with no interventions, expanded from Figure 6.5 to all seven metrics.

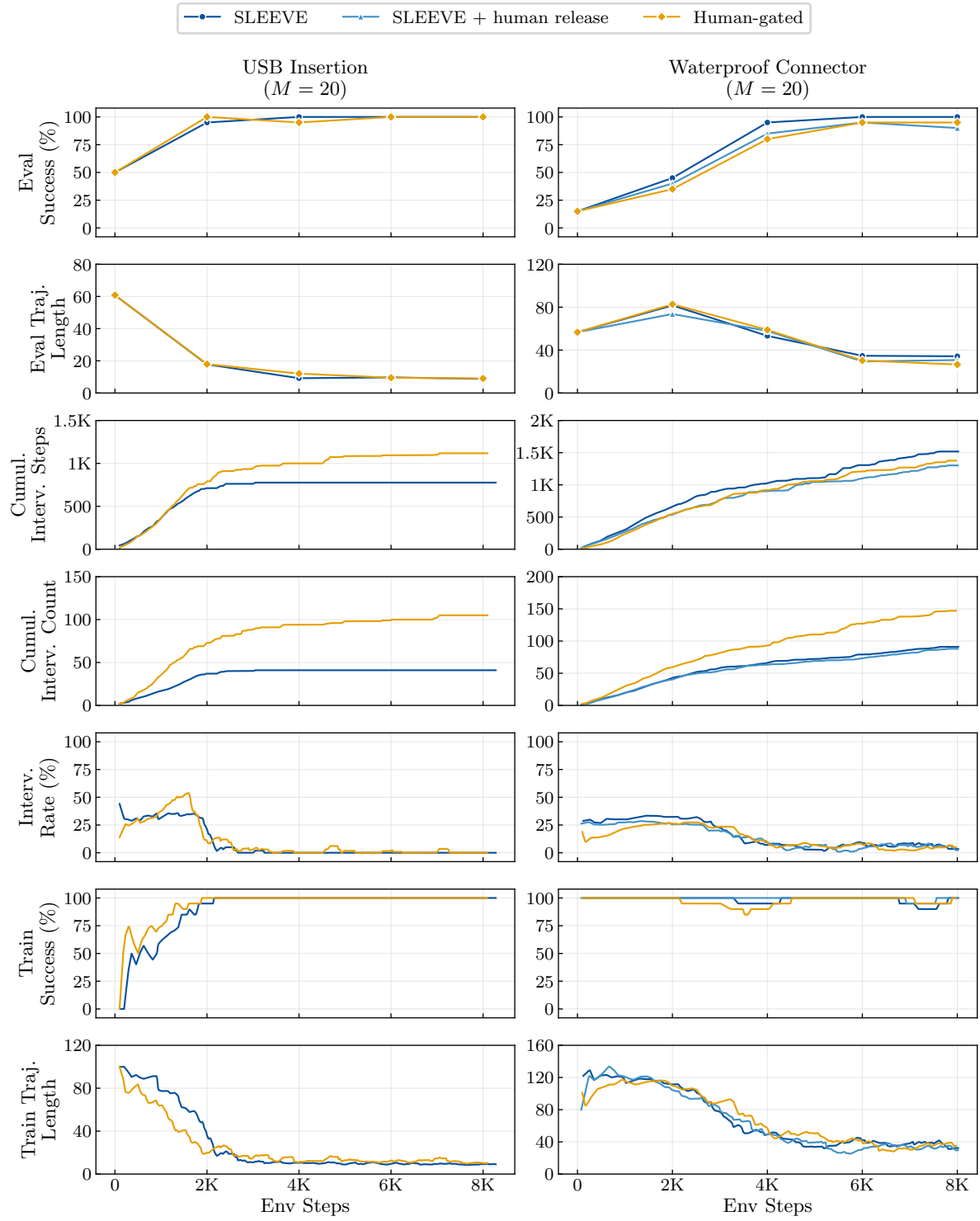


Figure A.6: The real-world experiments, expanded from Figure 6.6 to all seven metrics.

Bibliography

- [1] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, “Industrial robotics,” in *Springer Handbook of Robotics*, 2nd ed., B. Siciliano and O. Khatib, Eds. Cham, Switzerland: Springer, 2016, pp. 1385–1422. [Online]. Available: <https://doi.org/10.1007/978-3-319-32552-1%5F54>
- [2] Automotive News, “Top 100 global suppliers,” 2026, accessed: August 2026. [Online]. Available: <https://www.autonews.com/manufacturing/suppliers/ranking/2026/>
- [3] Magna International, “Products,” 2026, accessed: August 2026. [Online]. Available: <https://www.magna.com/products>
- [4] J. Luo, C. Xu, J. Wu, and S. Levine, “Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning,” *Science Robotics*, vol. 10, no. 105, p. eads5033, 2025. [Online]. Available: <https://doi.org/10.1126/scirobotics.ads5033>
- [5] Y. Chen, S. Tian, S. Liu, Y. Zhou, H. Li, and D. Zhao, “ConRFT: A reinforced fine-tuning method for VLA models via consistency policy,” in *Robotics: Science and Systems XXI (RSS)*, Los Angeles, CA, USA, 2025. [Online]. Available: <https://doi.org/10.15607/RSS.2025.XXI.019>
- [6] S. Kalouche and K. Kruempelstaedter, “Robotic system having shuttle,” U.S. Patent 11 318 499 B2, May 3, 2022. [Online]. Available: <https://patents.google.com/patent/US11318499B2/en>
- [7] M. Braga, “When your robot learns from humans, who should train it?” *CBC News*, Oct. 2017. [Online]. Available: <https://www.cbc.ca/news/science/robots-humans-ai-kindred-suzanne-gildert-singularity-1.4350330>
- [8] I. Salian, “AI flying off the shelves: Restocking robot rolls out to hundreds of Japanese convenience stores,” Aug. 2022, NVIDIA Blog. Accessed: August 2026.

- [Online]. Available: <https://blogs.nvidia.com/blog/telexistence-convenience-store-robotics/>
- [9] Y Combinator, “Aktoria Robotics: Human as API for robots,” 2026, company profile, Summer 2026 batch. Accessed: August 2026. [Online]. Available: <https://www.ycombinator.com/companies/aktoria-robotics>
- [10] MicroFactory, “MicroFactory — general purpose robot for automating manual work,” 2026, accessed: August 2026. [Online]. Available: <https://microfactory.com/>
- [11] I. Kulakov, “Introducing MicroFactory DevKit,” Nov. 2025, LinkedIn post. Accessed: August 2026. [Online]. Available: https://www.linkedin.com/posts/igor-kulakov-364a1553_introducing-microfactory-devkit-the-first-activity-7393723958405087232-z4iy
- [12] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg, “ThriftyDagger: Budget-aware novelty and risk gating for interactive imitation learning,” in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 164, London, United Kingdom, 2021, pp. 598–608. [Online]. Available: <https://proceedings.mlr.press/v164/hoque22a.html>
- [13] S. Ross, G. J. Gordon, and J. A. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA, 2011, pp. 627–635. [Online]. Available: <https://proceedings.mlr.press/v15/ross11a.html>
- [14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: <http://incompleteideas.net/book/the-book-2nd.html>
- [15] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, 1994. [Online]. Available: <https://doi.org/10.1002/9780470316887>
- [16] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983. [Online]. Available: <https://doi.org/10.1109/TSMC.1983.6313077>

- [17] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems 12 (NIPS)*, Denver, CO, USA, 1999, pp. 1008–1014. [Online]. Available: <https://proceedings.neurips.cc/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html>
- [18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: <https://doi.org/10.1038/nature14236>
- [19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016. [Online]. Available: <https://arxiv.org/abs/1509.02971>
- [20] S. Fujimoto, H. van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 80, Stockholm, Sweden, 2018, pp. 1587–1596. [Online]. Available: <https://proceedings.mlr.press/v80/fujimoto18a.html>
- [21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 80, Stockholm, Sweden, 2018, pp. 1861–1870. [Online]. Available: <https://proceedings.mlr.press/v80/haarnoja18b.html>
- [22] A. Kumar, A. Zhou, G. Tucker, and S. Levine, “Conservative Q-learning for offline reinforcement learning,” in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020, pp. 1179–1191. [Online]. Available: <https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html>
- [23] M. Nakamoto, Y. Zhai, A. Singh, M. Sobol Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine, “Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning,” in *Advances in Neural Information Processing Systems 36 (NeurIPS)*,

- New Orleans, LA, USA, 2023. [Online]. Available: <https://proceedings.neurips.cc/paper/2023/hash/c44a04289beaf0a7d968a94066a1d696-Abstract-Conference.html>
- [24] S. Fujimoto and S. S. Gu, “A minimalist approach to offline reinforcement learning,” in *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021, pp. 20 132–20 145. [Online]. Available: <https://proceedings.neurips.cc/paper/2021/hash/a8166da05c5a094f7dc03724b41886e5-Abstract.html>
- [25] A. Nair, A. Gupta, M. Dalal, and S. Levine, “AWAC: Accelerating online reinforcement learning with offline datasets,” 2020, arXiv preprint arXiv:2006.09359. [Online]. Available: <https://arxiv.org/abs/2006.09359>
- [26] I. Kostrikov, A. Nair, and S. Levine, “Offline reinforcement learning with implicit Q-learning,” in *10th International Conference on Learning Representations (ICLR)*, 2022. [Online]. Available: <https://openreview.net/forum?id=68n2s9ZJWF8>
- [27] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine, “Efficient online reinforcement learning with offline data,” in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 202, Honolulu, HI, USA, 2023, pp. 1577–1594. [Online]. Available: <https://proceedings.mlr.press/v202/ball23a.html>
- [28] D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, J. Luo, Y. L. Tan, L. Y. Chen, P. R. Sanketi, Q. Vuong, T. Xiao, D. Sadigh, C. Finn, and S. Levine, “Octo: An open-source generalist robot policy,” in *Robotics: Science and Systems XX (RSS)*, Delft, The Netherlands, 2024. [Online]. Available: <https://doi.org/10.15607/RSS.2024.XX.090>
- [29] Open X-Embodiment Collaboration *et al.*, “Open X-Embodiment: Robotic learning datasets and RT-X models,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, 2024, pp. 6892–6903. [Online]. Available: <https://doi.org/10.1109/ICRA57147.2024.10611477>
- [30] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever, “Consistency models,” in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 202, Honolulu, HI, USA, 2023, pp. 32 211–32 252. [Online]. Available: <https://proceedings.mlr.press/v202/song23a.html>

- [31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778. [Online]. Available: <https://doi.org/10.1109/CVPR.2016.90>
- [32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009, pp. 248–255. [Online]. Available: <https://doi.org/10.1109/CVPR.2009.5206848>
- [33] C. Celemin, R. Pérez-Dattari, E. Chisari, G. Franzese, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, and J. Kober, “Interactive imitation learning in robotics: A survey,” *Foundations and Trends in Robotics*, vol. 10, no. 1–2, pp. 1–197, 2022. [Online]. Available: <https://doi.org/10.1561/23000000072>
- [34] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, “HG-Dagger: Interactive imitation learning with human experts,” in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, 2019, pp. 8077–8083. [Online]. Available: <https://doi.org/10.1109/ICRA.2019.8793698>
- [35] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, K. Lin, A. Maddukuri, S. Nasiriany, and Y. Zhu, “robosuite: A modular simulation framework and benchmark for robot learning,” 2020, arXiv preprint arXiv:2009.12293. [Online]. Available: <https://arxiv.org/abs/2009.12293>
- [36] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, “EnsembleDagger: A Bayesian approach to safe imitation learning,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, 2019, pp. 5041–5048. [Online]. Available: <https://doi.org/10.1109/IROS40897.2019.8968287>
- [37] S.-W. Lee, X. Kang, and Y.-L. Kuo, “Diff-Dagger: Uncertainty estimation with diffusion policy for robotic manipulation,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, Atlanta, GA, USA, 2025, pp. 4845–4852. [Online]. Available: <https://doi.org/10.1109/ICRA55743.2025.11127730>
- [38] E. Biré, A. Kobanda, L. Denoyer, and R. Portelas, “Efficient active imitation learning with random network distillation,” in *13th International Conference on Learning Representations (ICLR)*, Singapore, 2025. [Online]. Available: <https://openreview.net/forum?id=GFgn2LprFR>

- [39] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama, “Active deep Q-learning with demonstration,” *Machine Learning*, vol. 109, no. 9–10, pp. 1699–1725, 2020. [Online]. Available: <https://doi.org/10.1007/s10994-019-05849-4>
- [40] M.-H. Chen, S.-A. Chen, and H.-T. Lin, “Active reinforcement learning from demonstration in continuous action spaces,” in *AI & HCI Workshop at the 40th International Conference on Machine Learning (ICML)*, Honolulu, HI, USA, 2023. [Online]. Available: <https://www.csie.ntu.edu.tw/%7Ehtlin/paper/doc/wsaihci23actreinforce.pdf>
- [41] M. Hou, K. Hindriks, A. E. Eiben, and K. Baraka, “‘Give Me an Example Like This’: Episodic active reinforcement learning from demonstrations,” in *Proceedings of the 12th International Conference on Human-Agent Interaction (HAI '24)*, Swansea, United Kingdom, 2024. [Online]. Available: <https://doi.org/10.1145/3687272.3688298>
- [42] J. Zhang and K. Cho, “Query-efficient imitation learning for end-to-end simulated driving,” in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, CA, USA, 2017, pp. 2891–2897. [Online]. Available: <https://doi.org/10.1609/aaai.v31i1.10857>
- [43] R. Hoque, A. Balakrishna, C. Putterman, M. Luo, D. S. Brown, D. Seita, B. Thananjeyan, E. Novoseller, and K. Goldberg, “LazyDagger: Reducing context switching in interactive imitation learning,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, Lyon, France, 2021, pp. 502–509. [Online]. Available: <https://doi.org/10.1109/CASE49439.2021.9551469>
- [44] H. Cai, Z. Peng, and B. Zhou, “Robot-gated interactive imitation learning with adaptive intervention mechanism,” in *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 267, Vancouver, BC, Canada, 2025, pp. 6243–6256. [Online]. Available: <https://proceedings.mlr.press/v267/cai25e.html>
- [45] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, “Recovery RL: Safe reinforcement learning with learned recovery zones,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021. [Online]. Available: <https://doi.org/10.1109/LRA.2021.3070252>

- [46] H. Li, K. Lei, S. Zang, K. Hu, Y. Liang, B. An, X. Li, and H. Xu, “Failure-aware RL: Reliable offline-to-online reinforcement learning with self-recovery for real-world manipulation,” 2026, arXiv preprint arXiv:2601.07821. [Online]. Available: <https://arxiv.org/abs/2601.07821>
- [47] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, “Leave no trace: Learning to reset for safe and autonomous reinforcement learning,” in *6th International Conference on Learning Representations (ICLR)*, Vancouver, BC, Canada, 2018. [Online]. Available: <https://openreview.net/forum?id=S1vuO-bCW>
- [48] A. Xie, F. Tajwar, A. Sharma, and C. Finn, “When to ask for help: Proactive interventions in autonomous reinforcement learning,” in *Advances in Neural Information Processing Systems 35 (NeurIPS)*, New Orleans, LA, USA, 2022, pp. 16 918–16 930. [Online]. Available: <https://proceedings.neurips.cc/paper/2022/hash/6bf82cc56a5fa0287c438baa8be65a70-Abstract-Conference.html>
- [49] C. Gokmen, D. Ho, and M. Khansari, “Asking for help: Failure prediction in behavioral cloning through value approximation,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, United Kingdom, 2023, pp. 5821–5828. [Online]. Available: <https://doi.org/10.1109/ICRA48891.2023.10161004>
- [50] C. Agia, R. Sinha, J. Yang, Z.-a. Cao, R. Antonova, M. Pavone, and J. Bohg, “Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress,” in *Proceedings of The 8th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 270, Munich, Germany, 2024, pp. 689–723. [Online]. Available: <https://proceedings.mlr.press/v270/agia25a.html>
- [51] Y. Liu, H. Yu, S. Liu, Y. Wang, J. Zhang, Y. Yang, Y. Chen, and C. Zhang, “One demonstration is enough for real-world robotic reinforcement learning,” in *European Conference on Computer Vision (ECCV)*, 2026. [Online]. Available: <https://arxiv.org/abs/2607.01651>
- [52] H. Deng, Y. Gao, Y. Lin, H. Liu, Z. Wu, and Z. Wang, “UniIntervene: Agentic intervention for efficient real-world reinforcement learning,” 2026, arXiv preprint arXiv:2606.12372. [Online]. Available: <https://arxiv.org/abs/2606.12372>
- [53] B. Wen, W. Yang, J. Kautz, and S. Birchfield, “FoundationPose: Unified 6D pose estimation and tracking of novel objects,” in *2024 IEEE/CVF Conference on*

- Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2024, pp. 17 868–17 879. [Online]. Available: <https://doi.org/10.1109/CVPR52733.2024.01692>
- [54] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine, “SERL: A software suite for sample-efficient robotic reinforcement learning,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, 2024, pp. 16 961–16 969. [Online]. Available: <https://doi.org/10.1109/ICRA57147.2024.10610040>
- [55] K. Kimble, K. Van Wyk, J. Falco, E. Messina, Y. Sun, M. Shibata, W. Uemura, and Y. Yokokohji, “Benchmarking protocols for evaluating small parts robotic assembly systems,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 883–889, 2020. [Online]. Available: <https://doi.org/10.1109/LRA.2020.2965869>