

To Err is Robotic: Rapid Value-Based Trial-and-Error during Deployment

Maximilian Du¹ **Alexander Khazatsky¹** **Tobias Gerstenberg²** **Chelsea Finn¹**

¹Department of Computer Science ²Department of Psychology
Stanford University

{maxjdu, alexkhaz, gerstenberg, cbfinn}@stanford.edu

Abstract: When faced with a novel scenario, it can be hard to succeed on the first attempt. In these challenging situations, it is important to know how to *retry* quickly and meaningfully. Retrying behavior can emerge naturally in robots trained on diverse data, but such robot policies will typically only exhibit undirected retrying behavior and may not terminate a suboptimal approach before an unrecoverable mistake. We can improve these robot policies by instilling an explicit ability to try, evaluate, and retry a diverse range of strategies. We introduce *Bellman-Guided Retrials*, an algorithm that works on top of a base robot policy by monitoring the robot’s progress, detecting when a change of plan is needed, and adapting the executed strategy until the robot succeeds. We start with a base policy trained on expert demonstrations of a variety of scenarios. Then, using the same expert demonstrations, we train a value function to estimate task completion. During test time, we use the value function to compare our expected rate of progress to our achieved rate of progress. If our current strategy fails to make progress at a reasonable rate, we recover the robot and sample a new strategy from the base policy while skewing it away from behaviors that have recently failed. We evaluate our method on simulated and real-world environments that contain a diverse suite of scenarios. We find that *Bellman-Guided Retrials* increases the average absolute success rates of base policies by more than 20% in simulation and 50% in real-world experiments, demonstrating a promising framework for instilling existing trained policies with explicit trial and error capabilities. Refer to this site for evaluation videos: <https://sites.google.com/view/to-err-robotic/home>

Keywords: Imitation Learning, Adaptation, Value Function

1 Introduction

Real robots see many novel scenarios as they operate in the human world, which highlights the importance of rapid adaptation—and particularly, the ability to recover and retry after an initial mistake. Recent works have demonstrated that robots can learn sophisticated real-world strategies [1, 2, 3, 4]. Many such robot skills have been learned by imitating demonstrations collected by an expert [1, 5]. From these demonstrations, the robot can implicitly learn how to recover and retry from a mistake. However, this ability depends on sufficient data diversity, and novel situations can pose a significant challenge. If the robot does not judiciously terminate an unproductive strategy, it may drift into an even more unfamiliar situation, which will further impair the performance of its policy [6, 7], including any learned ability to recover and retry. Even if the robot is robust to failure states in this novel scenario, there is no guarantee that the robot will try strategies systematically.

Recognizing the shortcomings of learning retrying behavior implicitly from expert demonstrations, we propose *Bellman-Guided Retrials*, a method that endows expert-trained robot policies with *explicit and systematic* strategy retrying behavior, allowing them to adapt quickly to novel scenarios. We design *Bellman-Guided Retrials* around a key insight: while attempting a novel situation, a robot

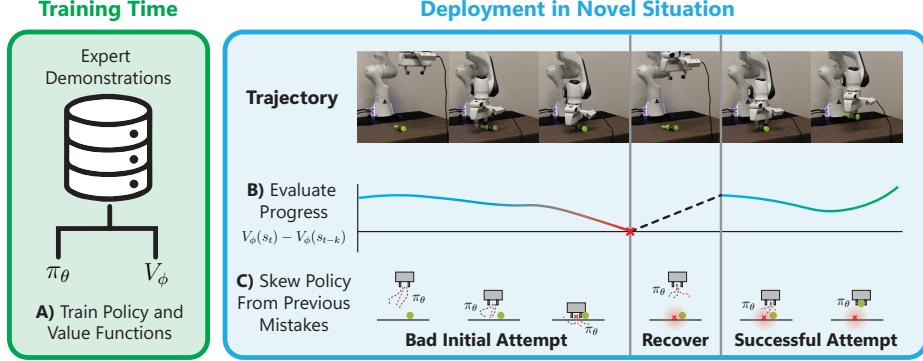


Figure 1: **The Bellman-Guided Retrials Method.** (A) Using expert demonstrations, we train a policy and value function. While solving a novel situation, we evaluate progress by looking at the behavior of the trained value function (B). If we detect suboptimality, we recover the robot and perform (C), which modifies the sampling of the pretrained policy to avoid past mistake states.

needs to monitor its progress. If it is progressing slower than expected, then it should try something different. By keeping track of previous failures, we can avoid retrying these suboptimal strategies.

We design *Bellman-Guided Retrials* to build on top of an existing base robot policy, which means that our method requires no external reward signals, expert interventions, or additional data outside of the pre-collected set of expert demonstrations used to train the base robot policy. We leverage the demonstrations to model good expert progress [8]. During test time, we can compare the robot’s progress with this expert progress to evaluate how optimal the current strategy is.

Concretely, we train a value function V_ϕ to estimate the time it should take the expert demonstrator to solve the scenario from a given state. During task execution, we monitor how consistent the value function’s predictions are. If the value function significantly overestimates the robot’s true performance, then the current strategy is likely suboptimal. By constantly monitoring the robot’s progress, we can stop the robot’s execution of a bad strategy before it reaches an unrecoverable state.

When we detect suboptimality, we execute a recovery policy. In many cases, this policy can be as simple as retracting a robot arm. Then, we sample another strategy from the base policy. We record states associated with suboptimal attempts and skew the sampling process to avoid similar states. The skew modification acts on top of the base policy’s existing sampling process by sampling multiple action proposals and picking the proposal that is least likely to yield a suboptimal state. This skewing method is non-parametric and can work with very little interaction data.

The main contribution of this work is a general framework for enabling explicit trial-and-error capabilities during deployment. Critically, by avoiding fine-tuning and using a separately trained value function, our approach is a drop-in replacement for the base policy. We test our method on two simulated grasping environments with a large collection of realistic objects. We perturb the test-time scenario by introducing novel objects or obstacles. We also test our method on a real robot in a similar grasping task, as well as a longer horizon door-opening task. Our approach boosts success rates by more than 20% in our simulation evaluations and more than 50% on a real robot setup.

2 Related Work

Learning From Demonstrations: In our framework, we assume access to a dataset of expert robot task demonstrations. With this dataset, a common approach is to train a control policy to imitate the demonstrated expert behavior [9, 10, 11, 12, 5]. Extensive work has explored different frameworks of imitation learning. This includes new observational spaces such as those with history [13], a large range of training procedures [6, 14, 15], algorithms that target different operation assumptions [6, 16], and various policy architectures [17, 18, 19]. There have also been significant efforts to collect diverse, multimodal expert demonstrations [17, 20, 21, 22]. Recently, state-of-the-art perfor-

mance has been achieved on models that predict long sequences of actions trained on diverse expert demonstrations [1, 14]. We use one of these models, Diffusion Policy, in our experiments as the base policy that captures a strategy repertoire. However, an expressive policy is not sufficient to attempt a new scenario successfully. If the policy gets into an out-of-distribution error state, it can fail to recover [6, 7]. Our method works on top of the base policy to catch these error states and recover from them before they become irrecoverable.

Adapting Rapidly: Many real-world situations require adapting to distribution shifts under short timeframes. When we have access to an expert, we can use this expert to intervene on mistakes and show the robot how to recover from them [8, 23, 24]. Expert corrections during test time can be effective, but constant expert supervision may limit the practicality of such approaches. Therefore, our method does not rely on expert corrections for adaptation.

For fast adaptation, agents can also leverage prior experiences in the environment. It can be beneficial to collect experiences that cover many diverse behaviors, which helps aggregate a robust suite of approaches for the test-time scenario [25, 26, 27]. With sufficient diversity, adaptation becomes a matter of selecting a working strategy rather than making a new one. Our method provides a way of selecting strategies during adaptation.

If adaptive behaviors are present in the training data, it is possible to distill adaptive behaviors naturally through offline meta-learning [28, 29, 30, 31]. However, assuming such a well-crafted data distribution remains constrictive. It can be hard to show a diverse range of adaptive behaviors, especially for a scenario that is not seen during training. Our method has a non-parametric skewing component that can accomplish rapid behavior adaptation without requiring adaptive behavior in the training data.

Adapting Without Supervision: As discussed above, most adaptation algorithms require some sort of supervision during adaptation. However, it can be impractical to assume access to experts constantly, particularly in a real-world environment. In our problem setup, we do not assume access to such supervision, including full environment resets. There have been works in the Single-Life RL domain with similar problem setups [32]. These works have focused on self-supervised adaptation by learning an internal objective [33] or searching for the best-pretrained policy [34] for deployment. An important component of Single-Life RL approaches is the ability to detect and recover from failures. A common approach is to train a failure detection model that looks at the probability of success [35], other internal properties [36, 37], or an explicitly trained Q function for risky states [38]. Failure detection models can detect mistake states effectively, but the same out-of-distribution observations that degrade performance for the policy can also degrade performance for the failure detection models. We also create a failure detection model for our method, but critically, we rely on the model’s self-consistency through a long history of past predictions. In the steps leading up to an out-of-distribution mistake, the predictions are still in-distribution. Therefore, the reliance on self-consistency is robust to distribution shifts in the robot’s current behavior, allowing unexpected mistakes to be detected rapidly.

Single-Life RL approaches often use failure detection models in conjunction with a recovery policy that bring the robot back to an in-distribution state [38, 39]. Our method also assumes access to a recovery policy and is compatible with such policies of any complexity. For our experiment setups, we find that a simple withdrawal of the robot arm is sufficient.

3 Background

3.1 Imitation Learning and Diffusion Policy

We adopt an imitation learning framework, where we train a policy $\pi_\theta(a|s)$ to imitate expert behavior from a dataset. Specifically, we use a diffusion-based policy π_θ [14]. Diffusion is a paradigm for training generative models to predict a sequence of denoising steps $\epsilon_{1:n}$ that refines a noisy sample a until it resembles a sample from the target distribution $p(a)$ [40]. During test-time generation, we can sample $a \sim p(a)$ by starting from noise and predicting a sequence of denoising steps. To

construct a diffusion *policy*, the noise prediction model ϵ_θ can be modified by conditioning it on the current state s . The final $\epsilon_\theta((a_1, \dots, a_k)|s)$ defines a policy and it allows us to sample a sequence of actions $\mathbf{a} = a_1, \dots, a_k$ to execute open-loop in the environment. Predicting sequences of actions at one time allows for higher control frequencies and leads to higher performance [14, 1].

3.2 Value Functions From Expert Demonstrations

Many expert-collected trajectories may not contain reward annotations, but it is still possible to learn a value function if we add a sparse reward label. In our case, we make the common assumption that the last state s_T in an expert demonstration is a success state [41, 42, 43]. We therefore assign reward r_+ to the success state at time T and r_- to all other states. With this, we can train a value function V_ϕ on the offline data by using Monte Carlo policy evaluation objective:

$$\mathcal{L}_{\text{MC}}(\phi) = E_{(t, T, s_t) \sim D} [(V_\phi(s_t) - (\gamma^{T-t} r_+ + \sum_{i=t}^T \gamma^{i-t} r_-))^2] \quad (1)$$

When trained on demonstration data, the estimated value function should be a good approximation of both V^{π^*} and V^{π_θ} , where π^* is the expert policy that collected the data and π_θ is a policy trained to imitate the demonstrations.

4 Rapid Trial and Error With Value Functions

4.1 Problem Setup

In this work, we consider the problem setting of adapting to a novel scenario, where partial observability may necessitate interaction with the environment before it can be solved successfully. Formally, we consider a Hidden-Parameter Markov Decision Process (HiP-MDP) [44], defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, p_s(s_0), p_z(z), p(s'|s, a, z), r(s, a, z))$. The hidden variable z influences the dynamics and the reward of the HiP-MDP, and the z is sampled from $p(z)$ at the start of every episode. During training, we are given a dataset \mathcal{D} that contains expert demonstrations in the HiP-MDPs on samples $z_1, \dots, z_k \sim p_z(\cdot)$ of the hidden variables. These experts may already have privileged knowledge of the hidden variables, so they may not reliably show how to discover them. They do, however, reliably show how to solve the HiP-MDP once these parameters are known. During test-time, we sample $z' \sim p_z(\cdot)$ and we try to solve this HiP-MDP. To solve the z' scenario, we need to uncover the hidden variable by interacting in the HiP-MDP. After we gain enough knowledge about the hidden parameter, we can then solve the z' scenario by using the strategies learned from \mathcal{D} .

4.2 Overview

In *Bellman-Guided Retrials*, we propose an approach that allows robots to iterate quickly and systematically across different strategies to solve the HiP-MDP with a new $z' \sim p_z(\cdot)$. We first train a diverse base policy π_θ via imitation learning on an expert dataset (Section 3.1). During a novel scenario, we sample strategies from π_θ with special oversight. Our key insight is that we can estimate how efficiently our π_θ should solve this scenario, and therefore we can see if the policy's current strategy is suboptimal. We estimate this performance expectation using a value function V_ϕ trained on the same expert demonstrations used to make π_θ (Section 4.3). When we detect significant suboptimality with the current strategy, we recover and try a new strategy from π_θ while avoiding past mistakes through a skewed sampling approach (Section 4.4). This trial-and-error process implicitly discovers parts of the hidden parameter z' , allowing us to solve the MDP quickly. See Algorithm 1 for a summary of our method.

4.3 Strategy Evaluation With Value Functions

In the first component of our method, we are interested in leveraging a trained value function V_ϕ to persistently evaluate the strategy that the base policy π_θ is currently executing. To evaluate progress, let us consider the Bellman target of $V_\phi(s_{t-1})$ computed with the trajectory of the current strategy:

$$y = r + \gamma E_{s_t}[V_\phi(s_t)] \approx r + \gamma V_\phi(s_t) \quad (2)$$

If V_ϕ were perfectly accurate and π_θ perfectly following expert behavior, then $V_\phi(s_{t-1}) = E[r + V_\phi(s_t)]$. But novel z' scenarios can create new and unexpected mistakes during a trajectory. Therefore, the Bellman error $V_\phi(s) - y$ may be non-zero for the current trajectory.

Specifically, let us consider the case where $V_\phi(s_{t-1}) - y > 0$. Here, $V_\phi(s_{t-1})$ is overestimating the true value of s_{t-1} in the novel z' scenario. For example, suppose that the robot is attempting to grasp an object and sees a handle-like protrusion in s_{t-1} . We sample a strategy from π_θ that tries to pinch the handle. Unknown to π_θ at the time, this affordance in the novel z' scenario is actually very slippery. After executing action $a \sim \pi_\theta$, the robot finds itself in a bad state s_t without any grasp. Therefore, the value associated with s_{t-1} should have been lower under this strategy, meaning the original handle grasping strategy was suboptimal.

From the analysis above, we argue that we can detect strategy suboptimality by comparing a value estimate $V_\phi(s_{t-1})$ with its bellman target $y = r + \gamma V_\phi(s_t)$. **If $V_\phi(s_{t-1}) - y > 0$, the executed strategy was suboptimal and the robot should recover and try a different strategy.**

In practice, at timestep t , we have access to transitions from $0 \rightarrow t$. We can take advantage of this access by computing the lower-bias k -step returns for the Bellman target of $V_\phi(s_{t-k})$. Computing the multi-step returns also improves the robustness of our method against out-of-distribution mistake states. Even if s_t is out of distribution, it is likely that s_{t-k} was still in distribution for large enough k . Therefore, we are comparing the $V_\phi(s_t)$ to a reliable, in-distribution value estimation. Empirically, we observe that out-of-distribution mistakes cause drastic inconsistencies in $V_\phi(s_t)$ that are easily detectable through our method (Figure 3).

The above framework should be functional for any value function on any reward structure, including sparse rewards. The cleanest formulation happens when we use the reward structure described in Section 3.2 and assign $r_- = -1$ to non-success states and $r_+ = 0$ to the final success state. If we also set $\gamma = 1$, our strategy evaluation framework with k -step returns becomes the following:

$$V_\phi(s_{t-k}) \stackrel{?}{>} V_\phi(s_t) - k \quad (3)$$

The plot in Figure 3 shows the evolution $V_\phi(s_t) - k - V_\phi(s_{t-k})$ as the robot attempts a scenario multiple times. Each of the times that the expression crosses zero corresponds closely to a mistake in the environment.

4.4 Non-Parametric Behavior Skewing

After detecting a suboptimal strategy using our strategy evaluation framework (Section 4.3), we can recover and try again. For manipulation tasks, recovery can be as simple as lifting the end-effector away from the object. Our proposed method does not set restrictions on recovery behavior, and the recovery can easily be a more complicated behavior.

After recovery, we must replan using π_θ . When we detect that a strategy is suboptimal, we also know the state s^- that triggered this detection. Therefore, in future runs of π_θ , we want to avoid s^- . To continue the previous example, if the z' scenario was a grasping task with a somewhat slippery object, s^- might correspond to being close to a slippery handle. Future samples from π_θ should avoid trying this slippery handle.

One explicit way to avoid s^- is adding a bias to the sampling from π_θ . If we had access to a transition model $p(s'|s, a)$ and multiple samples $a_1, \dots, a_k \sim \pi_\theta(\cdot|s)$, we could pick the sample a^* such that $p(s^-|s, a^*)$ is lowest. This biased sampling method is non-parametric, meaning that we can modify

Algorithm 1: *Bellman-Guided Retrials*
(Novel Scenario Deployment)

-
- ```

1: $S^{avoid} \leftarrow \emptyset$
2: while not successful do
3: Sample $\mathbf{a}_1, \dots, \mathbf{a}_k \sim \pi_\theta(\cdot|s_{t-1})$
4: Select \mathbf{a}^* according to Eq 4
5: Execute \mathbf{a}^* in environment, get s_t
6: Compute k -step Bellman target
 $y \leftarrow V_\phi(s_t) + \sum_{m=0}^{k-1} \gamma^m r_{t-k+m}$
7: if $V_\phi(s_{t-k}) > y$ then
8: Recover robot to neutral state
9: $S^{avoid} \leftarrow S^{avoid} \cup s_{t-1}$

```
-

$\pi_\theta$  with as few as a single  $s^-$  avoidance point. Biased sampling is less expressive than a gradient-based update on  $\pi_\theta(a|s)$ , but as long as  $\pi_\theta(a|s)$  has a sufficient variance, this biased sampling method can modify the action distribution significantly, achieving strategy adaptation.

Since we do not have access to a transition model, we approximate the above approach. We use a diffusion policy as  $\pi_\theta$ , which outputs a sequence of actions  $\mathbf{a}$  in the space of robot proprioception. We record  $s^-$  in the space of proprioception ( $s_{prop}^-$ ) and skew  $\pi_\theta$  based on the distance of a sampled action from a set  $S^{avoid}$  of avoidance points  $s_{prop}^-$ . Formally, we sample  $k$  action sequences  $\mathbf{a}_1, \dots, \mathbf{a}_k$  from  $\pi_\theta$  and then select skewed action sequences according to the following:

$$a^* = \arg \max_{\mathbf{a}_i \in \{1, \dots, k\}} \min_{t,j} \|S_j^{avoid} - (\mathbf{a}_i)_t\|_2^2 \quad (4)$$

The expression of states as proprioception has some drawbacks, including reduced efficacy in dynamic scenes. However, it allows us to avoid training  $p(s'|s, a)$ , which is advantageous.

In summary, our method takes a base robot policy and monitors its progress as it attempts a novel scenario (Section 4.3). If we detect suboptimality, we recover and replan by skewing the base policy away from states that have caused suboptimality in the past. Algorithm 1 presents a summary of our novel scenario adaptation.

## 5 Experiments

We conduct a series of experiments designed to evaluate whether *Bellman-Guided Retrials* can boost the performance of a trained base policy. First, we qualitatively assess if our strategy evaluation method is making reasonable judgments about strategy optimality (Section 5.2). Next, we look at how *Bellman-Guided Retrials* boosts performance in simulated and real tasks (Section 5.3). Finally, we perform ablations of our method (Section 5.3).

We train  $\pi_\theta$  and  $V_\phi$  on sets of human-teleoperated demonstrations. We make the design decision to represent  $V_\phi$  as a categorical distribution, inspired by accuracy improvements seen in Distributional RL [45, 46]. We also adopt the reward structure of  $r = -1$  for non-success states, which allows us to evaluate the current strategy by the simple formulation in Equation 3. The distributional nature of  $V_\phi$  allows us to turn the inequality of Equation 3 into a statistical comparison, which allows us to account for uncertainty in the estimated value. See the Appendix for more details.

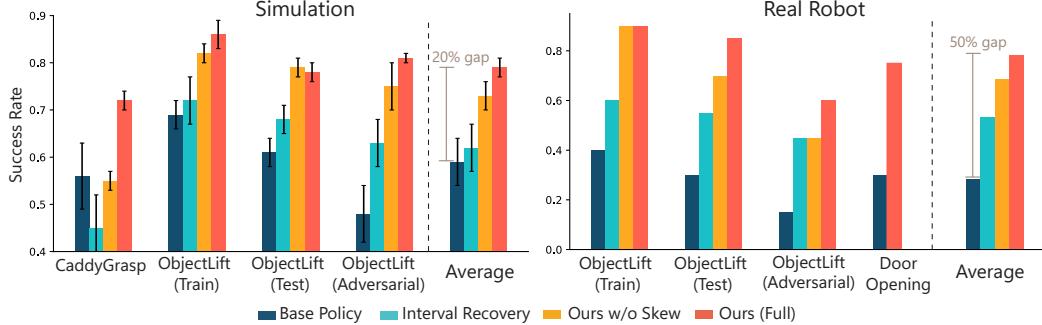
### 5.1 Experimental Domains

We construct three main experimental domains that allow us to collect expert data on a set of training scenarios and test on a novel scenario. We also add a fourth domain to demonstrate that *Bellman-Guided Retrials* is robust to different types of tasks.

In simulation, we introduce **CaddyGrasp** and **SimObjectLift**. The **CaddyGrasp** is a modified Robosuite [47] environment that requires the robot to lift a large shower caddy. The training set shows different grasps on all affordances of the caddy. During testing, an invisible obstacle (made visible in Figure 2) introduces a novel situation by blocking some of the affordances on the caddy, reducing the set of valid strategies. The **SimObjectLift** environment requires the robot to grab and lift a variety of objects from the Gazebo and ShapeNet datasets [48]. The training set shows grasps on a collection of objects. During testing, we provide novel situations by introducing held-out objects to the robot. We also test on an *adversarial* set of difficult-to-grasp objects.



Figure 2: **Experiment Environments.** We consider four sets of experiment domains spanning simulated and real tasks of varying difficulty.



**Figure 4: Comparing *Bellman-Guided Retrials* To Relevant Baselines.** In both simulation and real robots, our method boosts the performance of the base policy and outperforms an interval recovery baseline. Note that the **DoorOpening** results are limited because we lost the environment prematurely (Appendix C.5).

On a real Franka Robot arm, we introduce **RealObjectLift** and **DoorOpening**. The expert data comes from human teleoperation with hand-held Oculus VR controllers. The **RealObjectLift** is the analogous task to **SimObjectLift** with real play kitchen objects that require a variety of strategies to lift. For **RealObjectLift**, we also test on an *adversarial* setup with low-friction film taped to the objects and the robot grippers, which reduces the number of viable grasp strategies. The additional long-horizon **DoorOpening** task requires the robot to grab, twist, and pull open a tool cabinet. This task shows the ability of *Bellman-Guided Retrials* to work beyond object-picking applications.

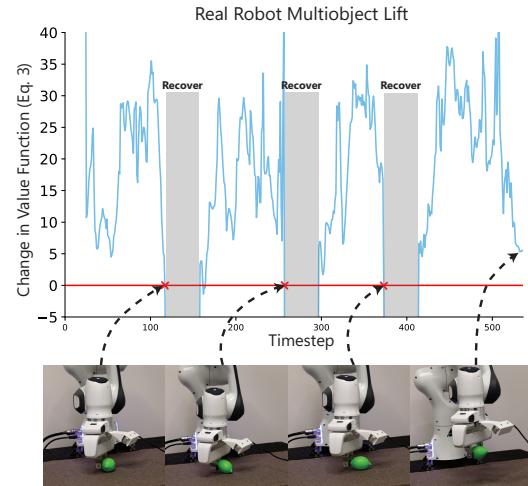
## 5.2 When does *Bellman-Guided Retrials* Detect Suboptimality?

For our first experiment, we qualitatively analyze how *Bellman-Guided Retrials* detects suboptimality by querying the value function. In Figure 3, we plot  $V_\phi(s_t) - V_\phi(s_{t-k}) - k$  over a trajectory. As detailed in Section 4.3, if this expression drops below zero, it indicates a suboptimality. In the figure, we see that the expression crosses the threshold three times, each happening a few frames after the plastic lime slips out of the robot’s grasp. These results are representative of a general property of our method: we can detect mistakes quickly, allowing the robot to save time during adaptation and also prevent it from reaching unrecoverable error states.

## 5.3 Does *Bellman-Guided Retrials* Improve Performance Of Expert-Trained Policies?

For our main set of quantitative experiments, we want to see how our method performs against relevant baselines. We test the expert-trained  $\pi_\theta$  without our method, and then we use the same  $\pi_\theta$  with our method by adding the suboptimality detection and skewed sampling. We test our method without the skewed sampling as an ablation, and we also test our method against an *interval recovery* baseline, where we trigger recoveries regularly based on the average number of steps it takes an expert to finish the task. For simulation, we compute success rates and standard deviations over 100 trials across three versions of  $\pi_\theta$ ,  $V_\phi$  trained with different random seeds. For the real robot experiment, we conduct 20 trials for each result. In all setups, we use a matched-pair design that ensures all algorithms get the same set of objects and starting orientations.

As seen in Figure 4, *Bellman-Guided Retrials* improves performance significantly over baselines in the three main real and simulated domains. The **CaddyGrasp** environment shows an especially large jump between the base policy and our method, as the base policy often gets stuck trying to grab a



**Figure 3: Visualizing Suboptimality Detection.** The change in value function (Eq 3) drops below zero immediately as the object slips out of the robot’s grasp.

part of the caddy that has been blocked. **CaddyGrasp** also shows a large jump in performance when we add skewed sampling. Without skewed sampling, the robot often tries the blocked regions over and over. The interval reset baseline also exhibits this failure mode. This **CaddyGrasp** environment is somewhat contrived, but it demonstrates clearly and intuitively the benefits of both components of our method. Appendix C.1 shows an additional **CaddyGrasp** visualization of skewed sampling.

The **SimObjectLift** environment also shows a significant difference between our method and baselines. We test performance on the train objects, held-out test objects, and adversarially-chosen objects. The more difficult situations (adversarial) created more opportunities for mistakes and recoveries, which increased the performance gap between the bare base policy and our method. Qualitatively, the main failure mode of the base policy is a failed grasp. Without our method, the base policy will hover over the dropped object with its grippers closed. In contrast, our method can quickly detect this mistake, recover, and retry. We also observe instances of early recovery, where our method will trigger a recovery before the robot attempts to lift on a bad grasp, which reduces the time per trial and the risk of irrecoverable changing the environment (e.g. knocking the object off the table). Our method still improved performance on the non-novel train set objects, which demonstrates that mistake detection is broadly useful. On average, our full method improves the base policy performance by 20% (Figure 4). See Appendix C.3 for more details.

The **RealObjectLift** environment shows similar trends as its analogous simulated task. The common failure mode of the base policy is a misgrasp, where the object shifts out of the grippers as they close. The base policy will attempt to lift and end up in an error state. We exacerbate this failure mode in the *adversarial* setup with low-friction film. This failure mode is readily detected by our method, and the skewed sampling encourages the robot to try different grasp point after recovery. On average, our full method improves base policy performance by 50% (Figure 4). We introduce an additional, difficult **DoorOpening** task and evaluate our full method against the bare base policy. Figure 4 shows that our full method boosts base policy performance by more than two times. Commonly, the base policy pushes the handle of the door too close to the pivot, which gets the robot stuck. Our method will detect this mistake, withdraw the arm and try again. In theory, the *Bellman-Guided Retrials* framework will work for any type of task, and **DoorOpening** supports this notion. See the Appendix C.4, C.5 for more details about **RealObjectLift** and **DoorOpening** results, respectively.

The main experiment results also include two critical ablations. First, the interval recovery baseline ablates the strategy evaluation afforded by our method. The reduction in performance shows that the statistics of an expert (average completion time) is not sufficient to determine when to recover from a mistake. Second, **Ours w/o Skew** ablation shows that the skewed resampling can boost performance by avoiding bad grasps, although it contributes less to overall performance than strategy evaluation. See the Appendix C.6 for more details.

## 6 Conclusion

We have presented *Bellman-Guided Retrials*, a method for adapting quickly to a novel scenario by enhancing the performance of a base policy. We monitor the viability of a strategy by using the self-consistency of a trained value function. We trigger a robot recovery when we detect suboptimal progress, and we also skew the robot policy away from past suboptimal states. Our experiments show that *Bellman-Guided Retrials* can boost success rates by more than 20% in simulation and 50% on a real robot.

**Limitations and Future Work.** We are excited by the possibilities of *Bellman-Guided Retrials*, but some limitations remain. Because we do not modify the expert-trained policy, our method will not imbue the robot with any strategies that it does not already have. Also, the framework of *Bellman-Guided Retrials* assumes access to a robust recovery policy, which may not be trivial for more involved tasks. Finally, we used distance in proprioceptive space for skewing, but such a metric would not work well if the environment were very dynamic. In future work, it is worth exploring a more expressive skewing method that operates in the robot’s *strategy space*, much like how a human may explicitly try conceptually different strategies.

## References

- [1] T. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *Robotics: Science and Systems*, 2023. URL <https://www.roboticsproceedings.org/rss19/p016.pdf>.
- [2] J. Grannen, Y. Wu, B. Vu, and D. Sadigh. Stabilize to Act: Learning to Coordinate for Bimanual Manipulation. URL <http://arxiv.org/abs/2309.01087>.
- [3] F. Xie, A. Chowdhury, M. C. De Paolis Kaluza, L. Zhao, L. Wong, and R. Yu. Deep Imitation Learning for Bimanual Robotic Manipulation. In *Advances in Neural Information Processing Systems*, volume 33, pages 2327–2337. Curran Associates, Inc. URL <https://proceedings.neurips.cc/paper/2020/hash/18a010d2a9813e91907ce88cd9143fdf-Abstract.html>.
- [4] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic. Dual arm manipulation—A survey. 60(10):1340–1353. ISSN 0921-8890. doi:10.1016/j.robot.2012.07.005. URL <https://www.sciencedirect.com/science/article/pii/S092188901200108X>.
- [5] S. Schaal. Is imitation learning the route to humanoid robots? 3(6):233–242. ISSN 1364-6613. doi:10.1016/S1364-6613(99)01327-3. URL <https://www.sciencedirect.com/science/article/pii/S1364661399013273>.
- [6] S. Ross, G. Gordon, and D. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635. JMLR Workshop and Conference Proceedings. URL <https://proceedings.mlr.press/v15/ross11a.html>.
- [7] S. Tu, A. Robey, T. Zhang, and N. Matni. On the Sample Complexity of Stability Constrained Imitation Learning. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, pages 180–191. PMLR. URL <https://proceedings.mlr.press/v168/tu22a.html>.
- [8] B. Mazoure, J. Bruce, D. Precup, R. Fergus, and A. Anand. Accelerating exploration and representation learning with offline pre-training. URL <http://arxiv.org/abs/2304.00046>.
- [9] D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 305 – 313. Morgan Kaufmann, December 1989.
- [10] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. 57(5):469–483. ISSN 09218890. doi:10.1016/j.robot.2008.10.024. URL <https://linkinghub.elsevier.com/retrieve/pii/S0921889008001772>.
- [11] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot Programming by Demonstration. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 1371–1394. Springer. ISBN 978-3-540-30301-5. doi:10.1007/978-3-540-30301-5\_60. URL [https://doi.org/10.1007/978-3-540-30301-5\\_60](https://doi.org/10.1007/978-3-540-30301-5_60).
- [12] S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. 358(1431):537–547. ISSN 0962-8436. doi:10.1098/rstb.2002.1258. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1693137/>.
- [13] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *Proceedings of the 5th Conference on Robot Learning*, pages 1678–1690. PMLR. URL <https://proceedings.mlr.press/v164/mandlekar22a.html>.

- [14] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation. ISBN 978-0-9923747-9-2. doi:10.15607/RSS.2023.XIX.026. URL <http://www.roboticsproceedings.org/rss19/p026.pdf>.
- [15] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit Behavioral Cloning. In *Proceedings of the 5th Conference on Robot Learning*, pages 158–168. PMLR. URL <https://proceedings.mlr.press/v164/florence22a.html>.
- [16] H. Liu, S. Dass, R. Martín-Martín, and Y. Zhu. Model-based runtime monitoring with interactive imitation learning, 2023.
- [17] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation. ISBN 978-0-9923747-9-2. doi:10.15607/RSS.2023.XIX.025. URL <http://www.roboticsproceedings.org/rss19/p025.pdf>.
- [18] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning. In *Proceedings of the 5th Conference on Robot Learning*, pages 991–1002. PMLR. URL <https://proceedings.mlr.press/v164/jang22a.html>.
- [19] N. M. Shafullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior Transformers: Cloning \$k\$ modes with one stone. 35:22955–22968. URL [https://proceedings.neurips.cc/paper\\_files/paper/2022/hash/90d17e882adbdda42349db6f50123817-Abstract-Conference.html](https://proceedings.neurips.cc/paper_files/paper/2022/hash/90d17e882adbdda42349db6f50123817-Abstract-Conference.html).
- [20] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine. Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets. In *Robotics: Science and Systems XVIII*. Robotics: Science and Systems Foundation. ISBN 978-0-9923747-8-5. doi:10.15607/RSS.2022.XVIII.063. URL <http://www.roboticsproceedings.org/rss18/p063.pdf>.
- [21] H. Walke, K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, C. Finn, and S. Levine. BridgeData V2: A Dataset for Robot Learning at Scale. URL <http://arxiv.org/abs/2308.12952>.
- [22] O. X.-E. Collaboration. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. URL <http://arxiv.org/abs/2310.08864>.
- [23] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. DART: Noise Injection for Robust Imitation Learning. URL <http://arxiv.org/abs/1703.09327>.
- [24] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer. HG-DAgger: Interactive Imitation Learning with Human Experts. URL <http://arxiv.org/abs/1810.02890>.
- [25] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is All You Need: Learning Skills without a Reward Function. URL <http://arxiv.org/abs/1802.06070>.

- [26] S. Kumar, A. Kumar, S. Levine, and C. Finn. One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL. URL <http://arxiv.org/abs/2010.14484>.
- [27] K. Derek and P. Isola. Adaptable Agent Populations via a Generative Model of Policies. In *Advances in Neural Information Processing Systems*, volume 34, pages 3902–3913. Curran Associates, Inc. URL <https://proceedings.neurips.cc/paper/2021/hash/1fc8c3d03b0021478a8c9ebcd457c67-Abstract.html>.
- [28] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL $\hat{2}$ \$: Fast Reinforcement Learning via Slow Reinforcement Learning. URL <http://arxiv.org/abs/1611.02779>.
- [29] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A Simple Neural Attentive Meta-Learner. URL <http://arxiv.org/abs/1707.03141>.
- [30] V. H. Pong, A. Nair, L. Smith, C. Huang, and S. Levine. Offline Meta-Reinforcement Learning with Online Self-Supervision. URL <http://arxiv.org/abs/2107.03974>.
- [31] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables. URL <http://arxiv.org/abs/1903.08254>.
- [32] A. S. Chen, A. Sharma, S. Levine, and C. Finn. You Only Live Once: Single-Life Reinforcement Learning. URL <http://arxiv.org/abs/2210.08863>.
- [33] N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-Supervised Policy Adaptation during Deployment. URL <http://arxiv.org/abs/2007.04309>.
- [34] A. S. Chen, G. Chada, L. Smith, A. Sharma, Z. Fu, S. Levine, and C. Finn. Adapt On-the-Go: Behavior Modulation for Single-Life Robot Deployment. URL <http://arxiv.org/abs/2311.01059>.
- [35] A. Rodriguez, M. T. Mason, S. Srinivasa, M. Bernstein, and A. Zirbel. Abort and Retry in Grasping.
- [36] L. Y. Ku, D. Ruiken, E. Learned-Miller, and R. Grupen. Error detection and surprise in stochastic robot actions. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1096–1101. doi:10.1109/HUMANOIDS.2015.7363505. URL <https://ieeexplore.ieee.org/document/7363505>.
- [37] J. Hejna, J. Gao, and D. Sadigh. Distance Weighted Supervised Learning for Offline Interaction Data. In *Proceedings of the 40th International Conference on Machine Learning*, pages 12882–12906. PMLR. URL <https://proceedings.mlr.press/v202/hejna23a.html>.
- [38] 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. URL <http://arxiv.org/abs/2010.15920>.
- [39] A. Sharma, R. Ahmad, and C. Finn. A State-Distribution Matching Approach to Non-Episodic Reinforcement Learning. URL <http://arxiv.org/abs/2205.05212>.
- [40] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc. URL <https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html>.

- [41] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, and S. Levine. Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills. URL <http://arxiv.org/abs/2104.07749>.
- [42] B. Eysenbach, T. Zhang, R. Salakhutdinov, and S. Levine. Contrastive Learning as Goal-Conditioned Reinforcement Learning. URL <http://arxiv.org/abs/2206.07568>.
- [43] A. Kumar, A. Singh, F. Ebert, M. Nakamoto, Y. Yang, C. Finn, and S. Levine. Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials. URL <http://arxiv.org/abs/2210.05178>.
- [44] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. *Advances in neural information processing systems*, 30, 2017.
- [45] M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning. URL <http://arxiv.org/abs/1707.06887>.
- [46] B. Eysenbach, R. Salakhutdinov, and S. Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. URL <http://arxiv.org/abs/1906.05253>.
- [47] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. robo-suite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.
- [48] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.
- [49] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

# Appendices

## A Model Details

### A.1 Data Modalities

In simulation, we include an eye-in-hand ( $224 \times 224 \times 3$ ) and a third-person ( $224 \times 224 \times 3$ ) camera perspective. We also include low-dimensional proprioceptive data, including the robot’s end-effector position, orientation, and gripper width.

For the real robot experiments, we include an eye-in-hand ( $256 \times 256 \times 3$ ) image from a camera mounted to the wrist of the Franka-Emika robot. We also include proprioceptive data about the end-effector position, orientation, and gripper width. See Figure 6 for an example of the camera angle on the real robot.

In both real and simulated robot experiments, we use an absolute position/orientation action space, meaning that the policy outputs the target position and orientation. Empirically, we found that absolute position/orientation actions are easier to learn on the diffusion policy.

### A.2 Diffusion Policy

We use the U-Net implementation of Diffusion Policy provided through Robomimic [14, 49]. The policy takes in images and feeds them through a Resnet-18 encoder (separate encoders for each camera) before concatenating the features with the low-dimensional proprioceptive state and encoding them through another MLP. The final feature is used to condition the noise model  $\epsilon_\theta$ . We sample action chunks of size 16 and execute 16 steps in an open-loop fashion before querying the diffusion policy for the next 16 steps. The action is in cartesian absolute space with one element of the action to control the robot gripper.

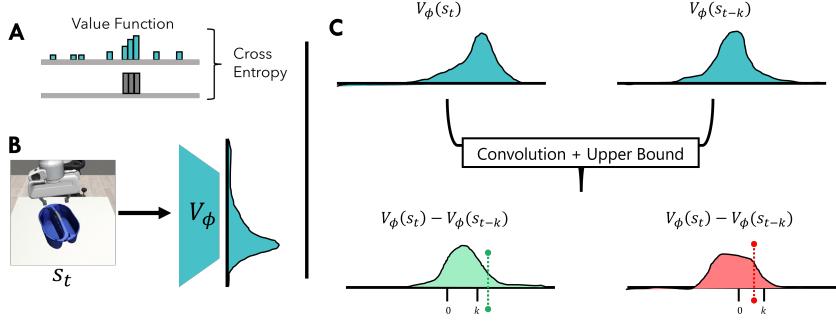
During training, we fit the prediction of 24 action steps to the expert data. During testing, we sample the chunk of 24 actions and execute 16 in the environment before resampling. Following empirical results, we also chose to remove history from the Diffusion policy model. For most other hyperparameters, we used the default provided by the codebase. We use the same architecture for the real and simulated experiments.

### A.3 Value Function

The Value Function is a visual encoder + MLP stack. It takes in the image(s), encodes them, and extracts a value distribution as a categorial distribution by feeding the image features through an MLP and softmax. The categorical distribution contains 50 elements, each representing 2% of progress.

During training, we sample  $s_t, t, T$  from the expert dataset. We can compute the relative progress as  $t/T$ , and we can use this to define the target distribution. We could round  $t/T$  to the nearest 2% and get bin  $b$ . Then, we could make the target distribution a one-hot vector at  $b$ . In practice, we found more success with a soft target. We find  $b$  and fill  $b - 1, b, b + 1$  with  $1/3$  of the density each. This helps the value function output a broader distribution, which is helpful for our statistical methods.

We do not feed proprioceptive data into the Value function to force it to leverage visual features (rather than simple end-effector locations) to measure progress. See Figure 6 for a visualization of the trained value function behavior on a successful trajectory.



**Figure 5: Details on training and using the value function.** We train  $V_\phi$  to output a categorical distribution (B) by regressing the logit outputs to a softened one-hot vector (A). During test-time, we compute  $V(s_t) - V(s_{t-k})$  by convolving the distributions and computing an upper bound on the difference (C). We show an example of acceptable progress (green, C) and not acceptable progress (red, C).

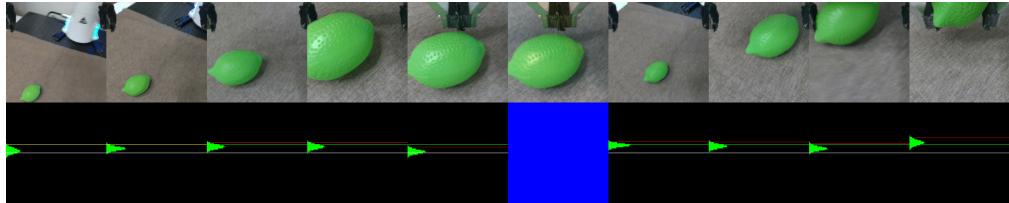


**Figure 6: Demonstration of our trained value function on a successful trajectory.** In the distribution (second row), we visualize the outputs of the value function through a successful trajectory. The value function distribution increases as the robot gets closer to the milk jug. The value function learns a distance to task completion by images alone.

#### A.4 Tuning and Training

We train both the diffusion policy and value function on the same datasets of expert demonstrations. We train the models until convergence, as evaluated on a set of validation trajectories. Concretely, we train diffusion policies for 200k gradient steps on the SimObjectLift task, 100k gradient steps on the CaddyGrasp task, and 400k on the RealObjectLift task. We train the value function for 400k gradient steps on the SimObjectLift task, 10k gradient steps on the CaddyLift task, and 120k steps on the RealObjectLift task. When evaluating on the whole system, we did a very light search on a set of value function checkpoints, and we picked the checkpoint that yielded the most stable results. In practice, the training of the diffusion policy and value functions are relatively stable and do not require anything more than a sanity check.

#### A.5 Implementing Progress Evaluation



**Figure 7: Demonstration of the progress evaluation on an initial failure.** In the distribution (second row), we visualize the delta distribution. The red horizontal line in the second row denotes the probabilistic upper bound, and the green horizontal line denotes the expected progress. The blue square indicates a recovery behavior.

We trained the value function to output a categorical distribution, which we can use to our advantage when we implement the progress evaluation part of our method. The expression  $V(s_t) - V(s_{t-k})$  now becomes a convolution of two distributions. The new delta distribution  $V(s_t) - V(s_{t-k})$  is a distribution of how the value function has changed in the last  $t$  steps. A recovery and retrial should not be done unless we are quite sure that the robot has made a mistake, so we take a probabilistic upper bound of the delta distribution. We compare this upper confidence interval to the expected progress  $k$ . If the upper bound is less than  $k$ , then we can say with a high probability that the robot has made a mistake. See Figure 5 for a diagram of the distributional value function computations, and see Figure 7 for a visualization of this progress evaluation on a real robot task.

In practice, we derive the upper bound by computing the standard deviation of the delta distribution and adding two standard deviations to the mean of the delta distribution.

We lightly tune the value of  $k$ , although we generally find success at  $k = 20$ . With our 20 Hz control setup, this corresponds to a one-second lookback time. Therefore, it is possible to find mistakes very quickly, down to a second of its occurrence.

### A.6 Implementing Skewing

To skew the actions away from past mistake states, we keep track of these mistake states in the robot’s proprioception space. Then, during execution, we sample  $n$  different action chunks. For each action chunk, we compute the pairwise L2 distance between actions and avoidance points, and we find the minimum distance. This tells us how close the proposed action chunk takes us to the set of states to avoid. We pick the action chunk that maximizes this distance. Intuitively, the action chunk chosen from this algorithm is an action that avoids the past mistake states as much as possible while following the samples of the base policy.

In simulation, we sample 10 chunks and pick the best according to the above method. On a real robot, we sample 3 times to reduce the time needed to query the diffusion policy.

## B Environment Details

### B.1 Generating / Collecting Data

For the CaddyLift task, we collected 500 demonstrations using human teleoperation through hand-held Oculus VR controllers. We used a random number generator to direct the demonstrator towards one part of the caddy to grasp, allowing us to approximate a uniform distribution of grasps on the caddy.

For the SimObjectLift task, we collected 3600 demonstrations by using a scripted policy that had access to privileged state information. The scripted policy attempt to pick up the object by finding an affordance and grasping it. We reject suboptimal demonstrations, which include slow grasps and trajectories that show grasp mistakes. The training set included grasps from 100 different objects sampled randomly from the Gazebo and Shapenet datasets.

For the RealObjectLift task, we use an internal dataset of object grasps on the Franka-Emika robot. We used a subset of the internal dataset that contains roughly 600 demonstrations of grasps on a brown background. The training set included grasps from >10 different kitchen toy objects.

In all tasks, we collect expert-only behavior in the datasets, which include very few demonstrations of mistake recovery. The scarcity of recovery demonstrations highlights the importance of *Bellman-Guided Retrials* in explicitly creating such capacities.

### B.2 Simulation Evaluations

In CaddyLift, we introduced an invisible barrier during test-time that made two affordances invalid. The invisible barrier acts like another wall, preventing the grippers from reaching around the affor-

dance. A robot that tries to grab a blocked affordance will find itself unable to maneuver its grippers around that affordance, but there are no visual deviations from the train data.

In SimObjectLift, we introduce ten novel objects sampled from the same training object distribution. These objects have never been seen in the dataset. For the adversarial task, we run a search over a set of novel objects and pick the objects that yielded the lowest success rates on the base policy.

For SimObjectLift and CaddyLift, we evaluated *Bellman-Guided Retrials* and baselines in simulation. With SimObjectLift, we used a horizon of 400 steps (not including recovery time) and a cap of 20 resets per episode, although nearly all episodes used less than 4 resets. For CaddyLift, we used a horizon of 600 steps (not including recovery time) and a cap of 20 resets per episode. In all simulation tasks, we provided all tested variants with the exact same set of starting objects and orientations. This matched pair comparison ensures that the differences in performance are purely explained by our method. To compute each success statistic, we collected 100 trials on three separate model seeds, allowing us to find the mean and sample variance of success rates.

### B.3 Real Robot Setup

We use a Franka-Emika Panda arm running on absolute positional control at 20 Hz. The images come from an Intel Realsense camera attached to the gripper using a custom 3D-printed mount.

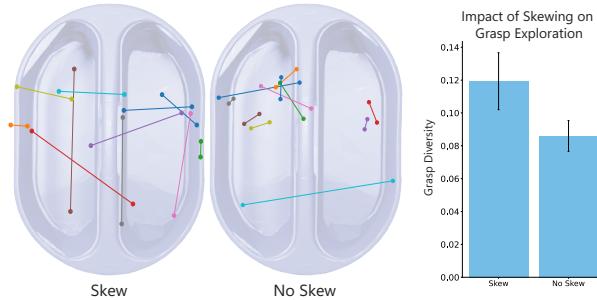
In the RealObjectLift-Train and RealObjectLift-Test, we evaluate the robot on a set of train objects and held-out test objects, respectively. These objects are placed randomly within a 10cm × 10cm table region, at random orientations. The test objects are similar kitchen toy objects but they have never been seen in the dataset.

To showcase our method’s ability to improve the robustness of a base policy, we changed the dynamics in an adversarial task by wrapping a polyethylene sheet (originally from packing material) around the robot grippers and three difficult-to-grasp objects. This modification caused many existing grasping strategies to become less effective. For example, it is no longer optimal to grab the cheese by the wedge part, as the cheese can slip out as the grippers are closing.

We strived to reduce the between-method variances as much as possible. Concretely, we evaluated all methods using the same set of start states. The object type, position, and orientation were made as similar as possible between trials across methods. To compute each success statistic, we collected 20 trials on the robot.

## C Additional Experiments and Discussion

### C.1 How does *Bellman-Guided Retrials* Skew the Base Policy?



**Figure 8: Impact of Skewing on Strategy Diversity.** In this visualization, we look at the different grasp locations attempted by a robot with and without skewing from past mistakes (**left**). For a random sample of trajectories, we plot two consecutive grasp points (denoted by the pairs of points connected by a line segment). With skewing, the points are further apart, indicating greater diversity between attempts. These qualitative results are supported by a larger average distance between grasps (**right**).

For our second experiment, we are interested in looking at how we can improve the diversity of attempted strategies by skewing the policy away from past mistakes states (Section 4.4). In Figure 8, we plot a progression of grasp attempts in the **CaddyGrasp** environment. Consecutive grasps are often in similar locations of the caddy when we do not skew away from past (suboptimal) attempts. In contrast, the grasp locations for the skewed policy are further away and often span different handles.

We added the skewing component to our method because we wanted to give the policy the ability to adapt quickly to failures. Our suboptimal detection can give the robot multiple attempts to solve a scenario, but without some form of adaptation, the policy could potentially make the same mistake over and over. The qualitative and quantitative results shown in Figure 8 demonstrate the strategy diversity imparted by the skewing, which increases the likelihood of finding the correct strategy.

## C.2 Does *Bellman-Guided Retrials* Improve Timesteps to Success?

Success rate is an easy-to-calculate and directly-relevant metric. Therefore, in the main paper results, we chose to use success rate. There are also other possible metrics to compare *Bellman-Guided Retrials*, including the number of steps the robot takes until a success. If the robot fails, we assign the number of steps as the horizon maximum. See Table 1 for results on the **SimObjectLift** task. Generally, our method reduces the number of steps needed until success. This is expected as our method prevents the robot from becoming stuck while also judiciously applying the recovery. The **Interval Recovery** tends to apply the recovery too frequently, which is reflected in the higher number of steps needed until success.

Table 1: Comparing *Bellman-Guided Retrials* Timesteps to Success To Relevant Baselines.

|                     | Base Policy<br>No Recovery | Interval<br>Recovery | Ours<br>w/o Skew | Ours<br>(Full) |
|---------------------|----------------------------|----------------------|------------------|----------------|
| SimObjectLift Train | 171 ± 10                   | 184 ± 12             | 160 ± 10         | <b>136 ± 7</b> |
| SimObjectLift Test  | 198 ± 9                    | 202 ± 9              | <b>172 ± 4</b>   | <b>172 ± 8</b> |
| SimObjectLift Advr  | 246 ± 17                   | 232 ± 16             | 192 ± 20         | <b>177 ± 7</b> |
| <b>Average</b>      | 205 ± 12                   | 206 ± 12             | 174 ± 11         | <b>162 ± 7</b> |

## C.3 More Details on SimObjectLift Experiment

The base policy may try to make a grasp, fail, and then hover above the object with its grippers closed. In contrast, our method generally triggers a recovery immediately when it becomes apparent that an object has not been grasped successfully (e.g. grippers are closed with no object between them). This gives the robot many more attempts to grasp the object. We also observe many instances of early recovery (Figure 9), where our algorithm finds suboptimality in the trajectory before the robot attempts lift with a bad grasp, reducing the time per trial and the risk of irrecoverably changing the environment (e.g., knocking the object off the table). We detail all simulated results in Figure 4.

## C.4 More Details on RealObjectLift Experiment

A common failure mode is a misgrasp, but another failure mode is an incorrect reach. The robot operates on wrist camera only, which means that an initial mistake during the reaching process could move the entire object out of frame. In these situations, the base policy fails by reaching toward a part of the table that does not contain the object. This mistake is quite out of distribution, and the robot rarely recovers. In contrast, the strategy evaluation component of our method is quick to detect when the object has left the frame and triggers a recovery that brings the object back into the frame

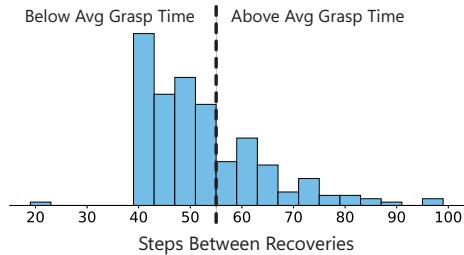


Figure 9: *Bellman-Guided Retrials* Allows for Early Termination of Suboptimal Strategies. Most recoveries of *Bellman-Guided Retrials* are tripped before the average time needed for an expert to grab and lift an object, showing that *Bellman-Guided Retrials* can detect mistakes quickly.

for another attempt. Similar to the simulation results, this early termination allows our method to outperform an Interval Recovery baseline by stopping suboptimal attempts much faster.

In the **test** and **adversarial** scenarios, we evaluate the robot on objects that are more difficult to grasp, like a smooth breadstick or a wedge of cheese. These objects introduce a new failure mode: object slippage during grasping. We exacerbate this failure mode in our **adversarial** variant when we reduce the friction of the robot grippers and objects with plastic film (Figure 2). When faced with these dynamic failures, our strategy evaluation component is quick to trigger a recovery—often within a second of an observed slippage (Figure 3). The **adversarial** variant also demonstrates the importance of skewing. With very different dynamics, a failure often happens because a previously feasible strategy is now suboptimal (e.g., trying to pick up a slippery carrot by the tapered end). Skewing the policy away from past mistakes helps our method improve over the interval recovery baseline. We detail all real robot results in Figure 4.

### C.5 More Details on DoorOpening Experiment

To demonstrate *Bellman-Guided Retrials*, the many of the main experiments used grasping tasks. However, *Bellman-Guided Retrials* is not limited to such tasks. To demonstrate the versatility of *Bellman-Guided Retrials*, we conducted an additional real robot experiment on an articulated **DoorOpening** task that requires the robot to rotate a handle and then pull the handle outwards to open the door (Figure 2). We collected expert demonstrations that showed a variety of successful strategies. The handle was rotated using different contact strategies and the unlocked handle was also grasped in different locations. We used these demonstrations to train the Value function and the base policy. We evaluated the robot on the same setup, and as can be seen in Figure 4, *Bellman-Guided Retrials* can improve performance by more than double the baseline success rate. Unfortunately, due to unforeseen circumstances, the real-world environment was lost before we could conduct our full sweep of experiments, but the existing two results show promising evidence that *Bellman-Guided Retrials* can boost performance of complicated, long-horizon task policies.

Qualitatively, the base robot policy frequently became stuck after trying to rotate the door handle by contacting the handle too close to the pivot, which required excessive force. Our method could detect the slowdown of progress and perform a recovery (pulling the arm back). When the arm came back to the handle, the robot often switched strategies and used a different contact that allowed for greater leverage on the handle. Through its mistake-recovery-retry sequence, *Bellman-Guided Retrials* is able to get the robot unstuck.

### C.6 More Details on Ablations

We need to disentangle the role of *strategy evaluation* from the role of *recoveries*. We make this comparison by looking at **Ours w/o Skew** with **Interval Recovery** in Figures 4. The interval recovery baseline triggers a recovery after a certain number of timesteps. We lightly tune these timesteps by using the average number of steps of an expert trajectory and adding some buffer steps to account for slightly slower progress. Critically, this tuning of the baseline must be done for every policy and environment, while *Bellman-Guided Retrials* is automatic. Under a fixed horizon time limit, our method outperforms the interval reset baseline in all but one variant. We hypothesize that our method has a benefit because it can detect mistakes quickly. Indeed, as seen in Figure 9, a distribution of strategy trial lengths shows many trials that were terminated before the robot attempted to lift an object. From these results, we conclude that a strategy evaluation-based recovery can benefit success rates if we operate under a fixed horizon.

We discussed the qualitative role of skewing in Section C.1. By comparing between **Ours w/o Skew** and **Ours (Full)** in Figures 4, we also see quantitative differences. These differences are most apparent on difficult scenarios like the **adversarial** object variants. We hypothesize that this trend is due to the increased need for diverse strategy exploration when the test-time scenarios differ more from the training scenarios.

From the three pairwise comparisons described above, we can conclude that both components of our method are important to the final performance improvement, with benefits being more pronounced in harder scenarios. Such a trend indicates that our method improves the robustness of the base policy.