Abstract: Building an interface that enables users to control robots through high-dimensional, noisy inputs like images of eye gaze can be challenging. The main problem is inferring the user’s desired action from the user’s raw input, without prior knowledge of how to do so. We approach the problem by eliciting user feedback on the system’s online performance, and training the interface through reinforcement learning (RL). This enables the interface to become more effective and personalized over time, but can be inefficient when feedback is sparse. To learn efficiently from sparse feedback, we divide and conquer: we first acquire a latent embedding space of useful high-level behaviors by autonomously pre-training the robot to do a variety of tasks, then quickly learn to map user input to the desired high-level behavior through human-in-the-loop RL. Our key insight is that access to a pre-trained policy enables the interface to learn more from the user’s sparse feedback during RL: when the interface successfully completes a task, we compute an optimal policy for that task in hindsight using the pre-trained policy, then train the interface to imitate that optimal policy in states from the successful trajectory as well as previous failures on the same task. We evaluate our method through a user study with 8 participants who perform tasks in two simulated robotic manipulation domains using a webcam and their eye gaze: flipping light switches, and opening a shelf door to reach objects inside. The results show that our method successfully learns to map 128-dimensional gaze features to 7-dimensional joint torques from sparse rewards, and seamlessly helps users who employ different gaze strategies, while adapting to distributional shift in webcam inputs, tasks, and environments.

1 Introduction

Shared-control teleoperation interfaces can help users control systems like robotic arms and wheelchairs more effectively, by augmenting the user’s commands with automated assistance (Kim et al., 2006, McMullen et al., 2013, Carlson and Demiris, 2012, Argall, 2016, Javdani, 2017). For example, they can help users perform dexterous robotic manipulation tasks by automatically maximizing contact area with grasped objects (Zhuang et al., 2019), or enable control via complex, noisy user input streams like eye gaze (Bien et al., 2004) and brain-computer interfaces (Muelling et al., 2015). In this paper, we focus on the problem of efficiently training an interface to infer the user’s desired action (e.g., robot arm motion) from a high dimensional control input (e.g., gaze). One class of prior methods accomplishes this through Bayesian goal inference, but typically requires knowledge of the user’s set of possible goals, the user’s control policy, or the environment dynamics (Hauser, 2013, Javdani et al., 2015, Pérez-D’Arpino and Shah, 2015, Koppula and Saxena, 2016, Muelling et al., 2017). Another class of prior methods calibrates the interface by performing supervised learning on paired examples of inputs and actions (Gilja et al., 2012, Dangi et al., 2013, 2014, Merel et al., 2015, Wang et al., 2016, Anumanchipalli et al., 2019, Karamcheti et al., 2020, Gaddy and Klein, 2020). This approach can also be limiting, in that it does not learn from the user’s online interactions with the system during deployment, and as a result, does not improve with use or adapt to distributional shift in the user’s inputs, tasks, and environments.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.
We overcome these limitations by eliciting user feedback on the system’s online performance, and training the interface through reinforcement learning (RL; Sutton and Barto, 2018). Our adaptive interface observes the user’s input, takes an action, receives a sparse, binary reward signal from the user at the end of each episode that indicates task success or failure, and learns to optimize this feedback. This approach is appealing because it scales with regular use: the more the user uses the interface to perform the activities of daily living (Ray et al., 2008, Mast et al., 2012, Petrich et al., 2021), the more competent and personalized the interface becomes. The main challenge is that, due to the sparsity of rewards, it can require extensive human-in-the-loop training, which may be impractical for an individual user operating a physical robot.

To address this challenge, we split up the problem into two phases: (1) acquiring a latent embedding space of high-level robot behaviors, and (2) mapping user commands to these high-level behaviors (see Figure 1). In phase 1, we pre-train a task-conditioned policy to perform a wide variety of tasks without the user in the loop, and automatically discover useful, high-level robot behaviors in the process. Then, in phase 2, we bring the user into the loop, and use RL with sparse, user-provided rewards to learn how to interpret the user’s commands as desired high-level behaviors. We contribute two novel insights into leveraging the pre-trained models from phase 1 to extract more information from the sparse rewards in phase 2 than standard RL algorithms: (a) when the user successfully completes a task, we observe information (e.g., the final state) that enables us to compute an optimal policy for that task in hindsight using our pre-trained task-conditioned policy, then train the interface to imitate that optimal policy; and (b) assuming that when the user fails, they attempt the same task again until they succeed, we can also relabel actions from failed trajectories with an optimal policy calculated in hindsight after an eventual success. We instantiate these insights in an algorithm that we call ASsistive teleoperation via HumAn-in-the-loop reinforcement learning (ASHA).

Our primary contribution is the ASHA algorithm for efficient learning of an adaptive interface from user feedback. We primarily evaluate ASHA through an online user study with 8 participants who use a webcam and their eye gaze to perform tasks in two simulated manipulation domains: flipping light switches, and opening a shelf door to reach objects inside (see Figure 2 for screenshots).
The results show that our method successfully learns to map 128-dimensional gaze features to 7-dimensional joint torques from sparse rewards, while adapting to distributional shift in the user’s webcam input caused by changes in ambient lighting and head position (Section 3.1); changes in the user’s set of desired tasks, like the addition of a new light switch (Section 3.2); and changes in environmental conditions, like whether a shelf door is initially open or closed (Section 3.3). In both domains, ASHA increased success rates for the majority of users, compared to a non-adaptive baseline interface. These users employed a variety of strategies to operate the interface – e.g., looking directly at the target, looking at distant parts of the screen to indicate different targets, exaggerating their gaze to correct the robot, and dynamically guiding the robot to subgoals (see Appendix C.2) – illustrating ASHA’s ability to seamlessly adapt to different communication styles.

2 Training an Assistive User Interface

In our setting, the user cannot directly operate the robot. Instead, the user relies on an assistive interface that infers the user’s intended action from available inputs, such as webcam images of eye gaze, or signals recorded by a brain implant. The user’s desired task is typically not directly observable to the robot, and this desired task may change between episodes. As such, we formulate the assistance problem as a partially-observable Markov decision process (POMDP; Kaelbling et al., 1998). The state consists of the state of the environment \( s_t \) (e.g., the position and orientation of the robot) and the user’s desired task \( T \) (e.g., flipping a particular light switch). The observation consists of the state of the environment \( s_t \) and the user’s control input \( x_t \) (e.g., an image of their eyes that captures gaze direction), but not the task \( T \). The user’s control input \( x_t \) communicates their intent to the robot. We do not assume access to the user’s desired task \( T \), since this can be difficult for the user to specify. Furthermore, we do not assume access to some ground-truth reward function \( R(s_t, a_t; T) \) that characterizes task performance, since that is typically internal to the user. Instead, we elicit a sparse, binary reward signal \( r_t \in \{0, 1\} \) from the user, in the form of a button press that indicates task success or failure at the end of each episode. We aim to learn an interface \( \pi^H(a_t|s_{0:t}, x_{0:t}) \) that optimizes this user-provided reward feedback. We also aim to minimize the number of human interactions required to learn this interface.

Our approach to this problem is outlined in Figure 1. Training an assistive interface through human-in-the-loop RL with sparse rewards typically requires many hours of interactions with users, in part due to the difficulty of simultaneously learning to control the environment and infer the user’s intent (Reddy et al., 2018). However, in typical teleoperation tasks, there are aspects of controlling the environment that can be learned separately from the user. Hence, we decompose the problem into two phases: (1) pre-training a ‘decoder’ \( g(a_t|s_t, z) \) that parameterizes the robot’s policy using a high-level latent variable \( z \); and (2) learning an ‘input encoder’ \( f(z|s_{0:t}, x_{0:t}) \) that infers the user’s desired high-level behavior \( z \) given the user’s control inputs \( x \).

2.1 Phase 1: Autonomous Pre-Training of a Task-Conditioned Policy

In many teleoperation domains, we can conservatively define a task distribution that covers a wide variety of behaviors that the user may potentially want to execute in the future – e.g., opening and closing cupboards in a kitchen, or flipping light switches on a wall – and then pre-train the robot to perform those tasks, without the user in the loop. Our final system is not necessarily limited to selecting from among these pre-training tasks. Rather, the space of skills acquired in phase 1 is meant to act as a kind of ‘basis’ for the tasks that the user might want to perform in phase 2.

During phase 1, we assume access to a discrete set of tasks \( i \in \{1, ..., n\} \), a specification \( \tau_i^{\text{spec}} \) of each task, and a reward function \( R_i(s_t, a_t) \) for each task. Note that the specification does not have to be a full trajectory, but merely a representation of the task that can be extracted from a successful trajectory – in our experiments, we define a set of goal-reaching tasks, set each specification \( \tau_i^{\text{spec}} \in \mathbb{R}^d \) to be the 3D position of the target light switch \( i \) or bottle \( i \), and define a hand-crafted, shaped reward function \( R_i \) for each task (see Appendices B.3 and B.6). To ensure that we learn a basis of skills, rather than a separate policy for each of the pre-training tasks, we follow prior work (Hausman et al., 2018) and represent the robot’s policy \( \pi^H \) as a latent variable model,

\[
\pi^H_{\phi}(a_t|s_t; i) \triangleq E_{g_{\phi}(x|z)} \{ g_{\phi}(a_t|s_t, x) \},
\]

where \( f_\phi \) is the ‘specification encoder’, \( g_\phi \) is the ‘decoder’, \( z \in \mathbb{R}^d \) is a latent variable that characterizes the task (we set \( d = 3 \) in our experiments), the prior distribution of \( z \) is the standard
normal distribution $\mathcal{N}(0, I_d)$, and the action $a_i$ is conditionally independent of the task $i$ given the state $s_i$ and latent embedding $z$. At the beginning of each pre-training episode, we sample a task $i$ uniformly at random. We then jointly pre-train the decoder $g_\phi$ and specification encoder $f_\psi$ to optimize the task rewards $R_i$ using RL — in our implementation, we use the soft actor-critic algorithm (SAC; Haarnoja et al., 2018). An important consequence of the latent variable model in Equation 1 is that, in addition to optimizing the task rewards $R_i$, we regularize the latent embedding $z$ to its prior distribution $\mathcal{N}(0, I_d)$ using a variational information bottleneck (VIB; Alemi et al., 2016, Achille and Soatto, 2018). The VIB encourages the model to learn a smooth, compressed latent space that shares information across tasks, and encourages the specification encoder $f_\psi$ to discard task-irrelevant information about the specification $\tau^\text{spec}$ from the embedding $z$ — this is critical to phase 2 of our method, because it prevents the interface from attempting to infer these task-irrelevant details from the user’s control inputs (see Q2 in Section 3.4). Appendices B.5 and B.6 describe our implementation of pre-training in more detail.

2.2 Phase 2: Human-in-the-Loop Reinforcement Learning of a User Interface

Now that we have acquired a latent embedding space of high-level robot behaviors (the left half of Figure 1), we turn to the problem of learning an interface that maps user inputs to desired high-level behaviors (the right half of Figure 1). We represent the interface as a latent variable model that reuses the pre-trained decoder $g_\phi$,

$$
\pi^R_{\theta, \phi}(a_t | s_{0:t}, x_{0:t}) \triangleq \mathbb{E}_{z \sim f_\psi(s_{0:t}, x_{0:t})} [g_\phi(s_t, z)],
$$

where $f_\psi$ is the ‘input encoder’, and the prior distribution of $z$ is the standard normal distribution $\mathcal{N}(0, I_d)$. Note that the input encoder $f_\psi$ differs from the specification encoder $f_\psi$ learned in the pre-training phase: $f_\psi$ reads in the user’s control input $x$ (e.g., gaze), while $f_\psi$ takes a specification $\tau^\text{spec}$ (e.g., goal state) as input instead. Since the user’s inputs $x$ are only partial observations of the state variable $T$ that defines the task, the interface $\pi^R_{\theta, \phi}$ is generally conditioned on the full sequence of states $s_{0:t}$ and inputs $x_{0:t}$. However, in our experiments, we find that conditioning on only the most recent state $s_t$ and input $x_t$ works well empirically.

Given the latent variable model in Equation 2, we train $f_\theta$ through RL from user feedback. Recent work in this area (Gao et al., 2021) suggests a straightforward method: assign a reward of 1 to successes, 0 to failures, and run a standard RL algorithm that essentially imitates the successful trajectories while down-weighting the failed trajectories. Due to the sparsity of the rewards, this approach would typically require a prohibitive amount of human interaction (see Q3 in Section 3.4). However, we contribute a novel insight that makes the method practical: we can extract more information from successful trajectories, by not simply imitating the actions that were actually taken (since some of them may be suboptimal), but instead imitating an optimal policy for the task that was completed. We can also extract more information from failed trajectories in the same manner, if we assume that when the user fails to perform a task, they reset the robot to its initial state (e.g., retract the robotic arm on their wheelchair back to its mount), and try to perform the same task again and again until they succeed. We now operationalize these two ideas, then arrive at our final method.

Learning efficiently from successes in hindsight. Instead of simply imitating a successful trajectory, we imitate an optimal policy conditioned on task information extracted from the successful trajectory. Let $D$ denote the set of successful trajectories. From each of these successes, we extract a specification $\tau^\text{spec}$ of the user’s desired task at the time — in our experiments, we set $\tau^\text{spec}$ to be the final 3D position of the object manipulated in the successful trajectory, analogous to the 3D target position specification used during the pre-training phase in Section 2.1. We then combine this task specification $\tau^\text{spec}$ with the pre-trained policy $\pi^R_{\psi, \phi}$ to represent the optimal policy $\pi^R_{\theta, \phi}(a_t | s_t, \tau^\text{spec})$ via Equation 1. The key idea is to match the interface $\pi^R_{\theta, \phi}$ with the optimal policy $\pi^R_{\psi, \phi}$, by optimizing the loss $\ell(\theta)$,

$$
\sum_{\tau^\text{spec} \in D, t} D_{\text{KL}}(\pi^R_{\psi, \phi}(\cdot | s_t, \tau^\text{spec}) \parallel \pi^R_{\theta, \phi}(\cdot | s_{0:t}, x_{0:t})) + \beta D_{\text{KL}}(f_\theta(\cdot | s_{0:t}, x_{0:t}) \parallel \mathcal{N}(0, I_d)),
$$

where the second term is the VIB for the latent variable model in Equation 2, and $\beta$ is a regularization constant. By minimizing the divergence between the policies induced by the input encoder $f_\theta$ and the pre-trained specification encoder $f_\psi$, we force $f_\theta$ to infer a latent embedding that induces the same low-level action distribution as the embedding inferred by $f_\psi$. This helps to reduce the amount
Algorithm 1 Asistive Teleoperation via Human-in-the-Loop Reinforcement Learning (ASHA)

1: \( g_\theta, f_\phi \leftarrow \text{RL}(\{\tau_i^{\text{spec}}, R_i\}_i) \) \quad \triangleright \text{pre-train the decoder and specification encoder autonomously}
2: \textbf{while} true \textbf{do} \\
3: \quad \tau \sim p(\tau) \quad \triangleright \text{user samples task from unknown task distribution}
4: \quad \mathcal{D} \leftarrow [] \quad \triangleright \text{initialize empty list of trajectories for current task}
5: \textbf{while} user indicates robot has not succeeded at the task \( \mathcal{T} \) yet \textbf{do} \\
6: \quad \tau \leftarrow [] \quad \triangleright \text{initialize empty trajectory}
7: \quad s_0 \sim p(s_0) \quad \triangleright \text{reset environment}
8: \quad \textbf{for} t \in \{0, 1, 2, \ldots, T - 1\} \textbf{do} \\
9: \quad \quad x_t \leftarrow \text{user's control input} \\
10: \quad \quad s_t \sim \pi^R_{\theta, \phi}(a_t | s_t, x_t) \quad \triangleright \text{robot performs action}
11: \quad \tau, \mathcal{D} \leftarrow \tau, \mathcal{D} + \text{store trajectory (regardless of whether it succeeds or fails)} \\
12: \quad \theta \leftarrow \theta - \nabla_\theta \sum_{\tau \in \mathcal{D}} D_{\text{KL}}(\pi^R_{\theta, \phi}(\cdot | s_{\tau}, \tau^{\text{spec}}) \parallel \pi^R_{\theta, \phi}(\cdot | s_{0:T}, x_{0:T})) + \text{VIB}(\theta) \quad \triangleright \text{update input encoder}

\begin{align*}
\text{Learning efficiently from failures in hindsight.} & \quad \text{Instead of simply treating failed trajectories as} \\
\text{examples of behavior that achieved zero reward, we take the final, successful trajectory at the end} & \quad \text{of a string of failed trajectories that attempted to perform the same task, extract a task specification} \\
\tau^{\text{spec}} & \quad \text{from this successful trajectory, compute the optimal policy} \pi^R_{\phi, \theta}(a_t | s_t, \tau^{\text{spec}}) \quad \text{for all the states} \\
\text{in the success and the failures, and optimize the loss in Equation 3. The key idea is that successful} & \quad \text{episodes enable us to compute the optimal policy for the most recent failure episodes, because a} \\
\text{string of failures and eventual success are all attempts to perform the same task. This helps to} & \quad \text{minimize the amount of human interaction required to train the system (see Q4 in Section 3.4).} \\
\text{Exploration in the latent space.} & \quad \text{Typical RL algorithms explore by randomizing the low-level} \\
\text{actions executed by the policy, but this can lead to sample complexity that grows exponentially with} & \quad \text{the episode length. The latent variable model in Equation 2 addresses this issue: by sampling a} \\
\text{laten embedding} & \quad \text{from the stochastic encoder} f_\phi, \text{we explore new high-level behaviors instead,} \\
\text{improving sample efficiency (see Q1 in Section 3.4) (Korf, 1987, Nachum et al., 2019).} \\
\end{align*}

2.3 Algorithm Summary

Our assistive teleoperation method is summarized in Algorithm 1. We initially pre-train the decoder \( g_\theta \) and specification encoder \( f_\phi \) with a set of task specifications and reward functions \( \{\tau_i^{\text{spec}}, R_i\}_i \) using a standard RL algorithm with a VIB – our implementation constructs several goal-reaching tasks and pre-trains on them with SAC (see Appendices B.5 and B.6). We then begin training the input encoder \( f_\theta \) with the user in the loop. First, the user decides on a task \( \mathcal{T} \). At each timestep \( t \), the environment generates the next state \( s_t \), and the user provides the system with input \( x_t \). After seeing the input \( x_t \), the robot takes an action \( a_t \) sampled from the interface \( \pi^R_{\phi, \theta} \) defined by the input encoder \( f_\phi \) and the pre-trained decoder \( g_\theta \) via Equation 2. At the end of each trajectory, we ask the user whether the robot succeeded or failed. If the robot fails, we reset and assume the user attempts to perform the same task again. If the robot succeeds, we take the successful trajectory, extract a task specification \( \tau^{\text{spec}} \) from it, use the pre-trained specification encoder \( f_\phi \) and decoder \( g_\theta \) to define an optimal policy for the task via Equation 1, and train the input encoder \( f_\theta \) to induce actions that match that optimal policy, by optimizing the loss in Equation 3. We find that training \( f_\theta \) to convergence in line 15 using mini-batch stochastic gradient descent on all past data, including data \( \mathcal{D} \) from previous tasks \( \mathcal{T} \), works well empirically. The user then decides on a new task, and we repeat with the updated input encoder \( f_\theta \). Appendix B.5 describes our network architectures and optimization procedures in detail.

Related work. During phase 2, ASHA learns from user feedback instead of an engineered reward function, similar to prior human-in-the-loop RL methods like COACH (MacGlashan et al., 2017,
Figure 2: In our experimental setup, the user sees the simulated environment through the point of view of someone sitting in the wheelchair, and directs their gaze to communicate what they want the robot to do. Their webcam records an image of their eyes, and our system treats this image as the user’s control input \( x \) (see Appendix B.4). In the switch domain (a), the switches are levers, and the robot must push the desired switch (indicated with a blue sphere) down to successfully complete the task. In the bottle domain (b), the shelf on top of the table has two compartments, and a sliding glass door that only makes one compartment accessible to the arm at any given time. To successfully complete the task, the user must reach for the desired bottle inside its compartment – if the sliding door covers the desired compartment, then the arm must open the door first.

Arumugam et al., 2019, TAMER (Knox and Stone, 2009, Warnell et al., 2017), and preference learning (Sadigh et al., 2017, Christiano et al., 2017). ASHA differs primarily in that it trains an interface that enables the user to control the robot at test time and perform any desired task, instead of training the robot to autonomously perform a single task. The goal of ASHA is to interpret a user’s commands when they are provided through a complex modality, such as gaze, which must be decoded into robot actions. This stands in contrast to prior methods that assume the user already has a viable interface for direct teleoperation, and aim to improve the user’s performance by minimally intervening in the user’s actions, e.g., to avoid collisions and preserve the reachability of potential goal states (Broad et al., 2017, Reddy et al., 2018, Schaff and Walter, 2020, Du et al., 2020, Jeon et al., 2020). See Appendix A for additional related work.

3 User Studies

In our experiments, we evaluate to what extent ASHA can adapt to the user’s inputs (Section 3.1), tasks (Section 3.2), and environment (Section 3.3). We conduct a user study with 8 participants who control a simulated, wheelchair-mounted, 7-DoF Jaco robotic arm using their eye gaze (see Figure 2). The interface receives 128-dimensional feature vectors that represent the user’s webcam image inputs \( x_t \), and outputs 7-dimensional joint torques as actions \( a_t \) (see Appendices B.3 and B.4). The users perform tasks in two simulated manipulation domains implemented with the PyBullet real-time physics simulator (Coumans and Bai, 2016) using assets from Assistive Gym (Erickson et al., 2020): flipping light switches, and opening a shelf to reach objects inside. Appendix B discusses the implementation details, and Appendix C.1 contains the experimental protocol.

3.1 Adapting to Distributional Shift in Gaze Inputs

In this experiment, we aim to test ASHA’s ability to improve over time by learning from online user feedback. To that end, we compare to a non-adaptive baseline interface that is initially calibrated via supervised learning, but does not adapt during deployment (analogous to the prior work discussed in Section 1).

To train this baseline interface, we collect paired data by showing a small number of pre-recorded videos of the robot autonomously performing tasks to the user, and recording the user’s passive gaze inputs as they watch the videos. We show 2 videos per task in each domain, totalling 6 videos in the
ASHA improves performance for these users. ASHA performs worse than the baseline for these users.

**Figure 3:** (a) Each circle represents the success rate for a single participant, averaged over 50 online episodes (11 minutes of wall-clock time). (b) Curves are smoothed using a moving average with a window of 5 episodes, and error bars show standard error across the 8 participants. The baseline is run over 50 episodes like with ASHA because changes in lighting and user behaviour over the course of the experiment may effect the accuracy of the gaze capture model. See Figures 6 and 8 and Table 3 in the appendix for more detailed plots.

To improve the initial performance and sample efficiency of our method, we initialize ASHA’s input encoder $\beta_0$ with the calibrated baseline parameters $\beta_0$, and initialize ASHA’s replay buffer with the same paired data that was used to calibrate the baseline. We measure the online performance of both methods by asking the user to complete particular tasks (e.g., flipping the switch indicated in blue), and computing the success rate of the user’s first attempt at each task (including subsequent attempts would introduce selection effects for difficult tasks; see Figure 8 in the appendix). We calibrate and evaluate on the same distribution of tasks: in the switch domain, a uniform distribution over flipping one of the three switches in the middle; and in the bottle domain, a uniform distribution over reaching one of the two bottles. To establish a lower bound on performance, we also compare to a baseline that randomly samples a latent $z$ and rolls out the policy $g_\phi(\alpha_t|s_t, z)$, without taking any user input.

The results in Figure 3a show that ASHA improves the success rates of the majority of users, relative to the non-adaptive baseline. From Figures 3b and 3c, we see that ASHA initially performs the same as the non-adaptive baseline, executing coherent but undesirable behaviors like moving toward the wrong target, then begins to outperform the baseline after 20 online episodes of RL. One potential explanation for the gap between ASHA and the non-adaptive baseline is that most users have a substantial distribution mismatch between passive and active inputs, and that ASHA helps those users by fine-tuning on active inputs instead of only initially calibrating on passive inputs. Another possibility is that ASHA adapts to changes in ambient lighting or head position over time, while the non-adaptive baseline performs increasingly worse over time due to these changes. We ran a one-way repeated measures ANOVA on the success rate dependent measure from the baseline and ASHA conditions, with the presence of ASHA as a factor, and found that $f(1, 7) = 3.14, p > .1$ in the switch domain, and $f(1, 7) = 50.64, p < .001$ in the bottle domain. The subjective evaluations in Table 2 in Appendix C.2 corroborate these results: users reported feeling more in control of the robot with ASHA compared to the baseline, although the difference is not statistically significant.

**3.2 Learning to Perform the User’s Desired Tasks**

The previous experiment showed that ASHA can adapt to distributional shift in the user’s gaze input. In this next experiment, we show that ASHA can also adapt to individual differences in the user’s desired task distribution. In the switch domain in particular, we calibrate on paired data generated from one distribution of tasks – a uniform distribution over the second and third switches from the left – then evaluate online on a different distribution of tasks – a uniform distribution over the second, third, and fourth switches from the left. This is challenging, since examples of the fourth switch being pressed are not included in the calibration data. RL offers a natural solution to this problem by fine-tuning the model on the user’s online attempts to perform new tasks. The results
in Figure 4a show that ASHA can indeed adapt to the new task distribution, substantially improving upon its initial success rate by the end of the online training period.

3.3 Adapting to a Changing Environment

Adaptation is useful, not only because the user’s input might drift or their desired tasks might be individualized, but also because the environment may have changed since the interface was previously calibrated. RL again offers a natural solution to this problem by incorporating new experiences into its replay memory as the user interacts with their changing environment. To illustrate this idea, we run an experiment in the bottle domain in which we calibrate on paired data where the sliding door never covers the desired bottle, then evaluate online in scenarios where the sliding door may randomly cover the desired bottle. The results in Figure 4b show that ASHA adapts to the new environmental conditions, increasing the success rate over time.

3.4 Ablation Experiments

To run ablation experiments at a scale that would be impractical in a user study, we simulate user input $x \in \mathbb{R}^3$ as the 3D position of the target switch or bottle with i.i.d. isotropic Gaussian noise added at each timestep (see Appendix B.9). We seek to answer the following questions: Q1: Does sampling from a stochastic input encoder $f_\theta$ improve exploration, relative to a deterministic encoder? Q2: Does pre-training with a VIB improve downstream performance during human-in-the-loop learning, relative to pre-training without a VIB? Q3: Does pre-training the decoder $g_\phi$ speed up human-in-the-loop learning, relative to end-to-end training the interface from scratch online? Q4: Does relabeling failures speed up human-in-the-loop learning, relative to ignoring failures and only training on successes? Q5: Does regressing onto the optimal policy in Equation 3 perform better than regressing onto sampled actions that led to a success? The results in Table 1 show that all the ablated variants of ASHA perform worse than the full ASHA method, suggesting that sampling from a stochastic input encoder $f_\theta$ improves exploration (Q1), pre-training with a VIB and reusing the pre-trained decoder $g_\phi$ speed up downstream learning (Q2, Q3), relabeling failures makes human-in-the-loop learning more efficient (Q4), and regressing onto an optimal policy is more effective than regressing onto sampled actions (Q5).

<table>
<thead>
<tr>
<th>Table 1: Ablation Experiments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>Random Latent (Baseline)</td>
</tr>
<tr>
<td>Non-Adaptive (Baseline)</td>
</tr>
<tr>
<td>ASHA (Ours)</td>
</tr>
<tr>
<td>ASHA w/ Det. Input Enc. (Q1)</td>
</tr>
<tr>
<td>ASHA w/ Det. Pre-train Enc. (Q2)</td>
</tr>
<tr>
<td>SAC from Scratch (Q3)</td>
</tr>
<tr>
<td>ASHA w/o Failure Relabeling (Q4)</td>
</tr>
<tr>
<td>ASHA w/ Latent Regression (Q5)</td>
</tr>
</tbody>
</table>

Success rates and standard errors measured across 100 episodes and 10 random seeds. See Figure 9 and Table 4 in the appendix for more detailed plots.

4 Discussion

We presented a proof of concept system that efficiently trains an adaptive interface through RL from sparse user feedback. Our experiments in two simulated robotic manipulation domains show that, in under 10 minutes of online learning, our method can adapt to distributional shift in webcam inputs, tasks, and environments. One limitation of our method is that it assumes access to a set of pre-training tasks and accompanying reward functions (see Section 2.1). Future work could use a self-supervised RL algorithm to discover a latent skill space without a pre-determined set of tasks (Eysenbach et al., 2018, Nair et al., 2020). Despite this limitation, ASHA illustrates how RL can provide a general mechanism for efficiently adapting user interfaces to individual needs; not only for assistive robotic teleoperation, but also potentially for other domains, such as brain-computer interfaces for speech decoding (Guenther et al., 2009, Bocquelet et al., 2016).
References

H. K. Kim, J. Biggs, W. Schloerb, M. Carmena, M. A. Lebedev, M. A. Nicolelis, and M. A. Srini-

vasan. Continuous shared control for stabilizing reaching and grasping with brain-machine inter-


D. P. McMullen, G. Hotson, K. D. Katyal, B. A. Wester, M. S. Fifer, T. G. McGee, A. Harris, M. S.

Johannes, R. J. Vogelstein, A. D. Ravitz, et al. Demonstration of a semi-autonomous hybrid

brain–machine interface using human intracranial EEG, eye tracking, and computer vision to

control a robotic upper limb prosthesis. IEEE Transactions on Neural Systems and Rehabilitation

Engineering, 2013.

T. Carlson and Y. Demiris. Collaborative control for a robotic wheelchair: evaluation of perform-


K. Z. Zhuang, N. Sommer, V. Mendez, S. Aryan, E. Formento, E. D’Anna, F. Artoni, F. Petrini,

G. Granata, G. Cannaviello, et al. Shared human–robot proportional control of a dexterous myo-


Park, S.-H. Kang, et al. Integration of a rehabilitation robotic system (KARES II) with human-


Schwartz, J. L. Collinger, and J. A. Bagnell. Autonomy infused teleoperation with application


K. Hauser. Recognition, prediction, and planning for assisted teleoperation of freeform tasks. Au-

tonomous Robots, 2013.


C. Pérez-D’Arpino and J. A. Shah. Fast target prediction of human reaching motion for cooperative

human-robot manipulation tasks using time series classification. In IEEE International Confer-

ence on Robotics and Automation, 2015.

H. S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive


Schwartz, J. L. Collinger, and J. A. Bagnell. Autonomy infused teleoperation with application to


V. Gilja, P. Nuyujukian, C. A. Chestek, J. P. Cunningham, M. Y. Byron, J. M. Fan, M. M. Church-


tinuous closed-loop decoder adaptation with a recursive maximum likelihood algorithm allows


J. Merel, D. Carlson, L. Paninski, and J. P. Cunningham. Neuroprosthetic decoder training as imita-


