Human-in-the-Loop Reinforcement Learning for Adaptive Assistive Interfaces

Jensen Gao

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-62
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-62.html

May 11, 2022

Human-in-the-Loop Reinforcement Learning for Adaptive Assistive Interfaces

by Jensen Gao

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

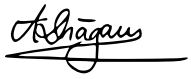
Committee:

Professor Sergey Levine
Research Advisor

May 8, 2022

(Date)

* * * * * * *



Professor Anca Dragan
Second Reader

May 11, 2022

(Date)

Human-in-the-Loop Reinforcement Learning for Adaptive Assistive Interfaces

by

Jensen Gao

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Sergey Levine, Chair

Spring 2022

Human-in-the-Loop Reinforcement Learning for Adaptive Assistive Interfaces

Abstract

Human-in-the-Loop Reinforcement Learning for Adaptive Assistive Interfaces

by

Jensen Gao

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Associate Professor Sergey Levine, Chair

Machine learning has shown great potential to facilitate more effective methods for human-computer interaction. This includes artificial intelligence-based interfaces that can assist users with performing their desired objectives with improved performance. In this technical report, we propose two human-in-the-loop deep reinforcement learning (RL) based methods to infer the intent of a user through only high-dimensional, noisy user inputs, while adapting to the user's inputs and feedback over time, in order to assist the user in performing their desired objectives more effectively. In Chapter 1, we propose a deep RL approach that learns from human feedback for assistive typing interfaces, which we formulate as contextual bandit problems. In Chapter 2, we propose a method that extends this style of approach for robotics tasks, which require sequential decision making. We do this through leveraging autonomous pre-training with deep RL. We demonstrate the effectiveness of these approaches using simulated user inputs, real user studies where participants communicate intent through webcam eye gaze, and a pilot study using brain-computer interfaces.

To my family

Contents

Acknowledgments

I would first like to thank my advisor Sergey Levine for providing me with the invaluable opportunity to work in his lab as an undergraduate and master's student. I've truly appreciated the constant feedback and guidance from him that have been crucial towards steering my research in productive directions, and I will forever cherish the time I've spent at RAIL.

I would also like to thank my graduate student advisor Sid Reddy for being a fantastic mentor and a great person to work with. I've learned so much about the research process from him, and I am incredibly grateful for the immense amount of support he's provided in our projects together.

Thank you to Anca Dragan, for the consistently positive energy during all of our interactions, and for the very helpful advice and feedback on the papers we've written together.

I would also like to thank my other collaborators, Sean Chen, Glen Berseth, Nicholas Hardy, Nikhilesh Natraj, and Karunesh Ganguly. It's been a great privilege to have been able to work on real brain-computer interfaces and shared autonomy with you all.

Thank you to Yahav Avigal and Ken Goldberg, for introducing me to the world of AI research during my time at AUTOLAB.

Finally, thank you to all of the friends I've made during my time at Berkeley, it's made my experience here thoroughly memorable and enjoyable.

Chapter 1

X2T: Training an X-to-Text Typing Interface with Online Learning from User Feedback

We aim to help users communicate their intent to machines using flexible, adaptive interfaces that translate arbitrary user input into desired actions. In this chapter, we focus on assistive typing applications in which a user cannot operate a keyboard, but can instead supply other inputs, such as webcam images that capture eye gaze or neural activity measured by a brain implant. Standard methods train a model on a fixed dataset of user inputs, then deploy a static interface that does not learn from its mistakes; in part, because extracting an error signal from user behavior can be challenging. We investigate a simple idea that would enable such interfaces to improve over time, with minimal additional effort from the user: online learning from user feedback on the accuracy of the interface's actions. In the typing domain, we leverage backspaces as feedback that the interface did not perform the desired action. We propose an algorithm called x-to-text (X2T) that trains a predictive model of this feedback signal, and uses this model to fine-tune any existing, default interface for translating user input into actions that select words or characters. We evaluate X2T through a small-scale online user study with 12 participants who type sentences by gazing at their desired words, a large-scale observational study on handwriting samples from 60 users, and a pilot study with one participant using an electrocorticography-based brain-computer interface. The results show that X2T learns to outperform a non-adaptive default interface, stimulates user co-adaptation to the interface, personalizes the interface to individual users, and can leverage offline data collected from the default interface to improve its initial performance and accelerate online learning.

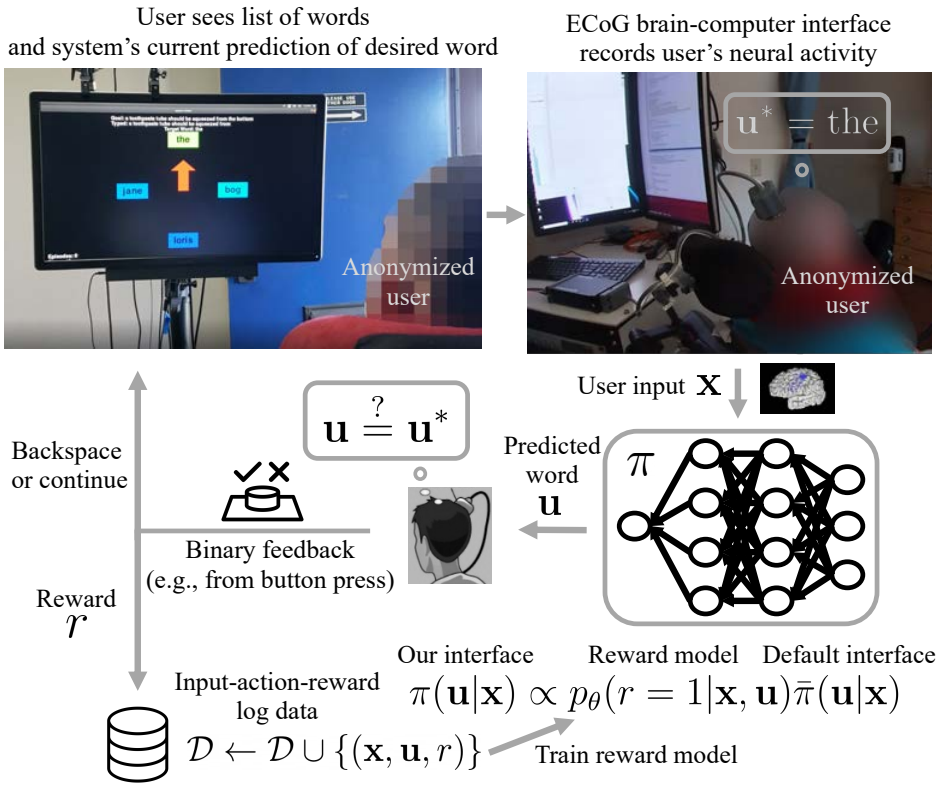


Figure 1.1: We formulate assistive typing as a human-in-the-loop decision-making problem, in which the interface observes user inputs (e.g., neural activity measured by a brain implant) and performs actions (e.g., word selections) on behalf of the user. We treat a backspace as feedback from the user that the interface performed the wrong action. By training a model online to predict backspaces, we continually improve the interface.

1.1 Introduction

Recent advances in user interfaces have enabled people with sensorimotor impairments to more effectively communicate their intent to machines. For example, [94] enable users to type characters using an eye gaze tracker instead of a keyboard, and [96] enable a paralyzed human patient to type using a brain implant that records neural activity. The main challenge in building such interfaces is translating high-dimensional, continuous user input into desired actions. Standard methods typically calibrate the interface on predefined training tasks for which expert demonstrations are available, then deploy the trained interface. Unfortunately, this does not enable the interface to improve with use or adapt to distributional shift in the user inputs.

In this chapter, we focus on the problem of assistive typing: helping a user select words or characters without access to a keyboard, using eye gaze inputs [94]; handwriting inputs (see Figure 4.2 in the appendix), which can be easier to provide than direct keystrokes [96];

or inputs from an electrocorticography-based brain implant [51, 89]. To enable any existing, default interface to continually adapt to the user, we train a model using online learning from user feedback. The key insight is that the user provides feedback on the interface's actions via *backspaces*, which indicate that the interface did not perform the desired action in response to a given input. By learning from this naturally-occurring feedback signal instead of an explicit label, we do not require any additional effort from the user to improve the interface. Furthermore, because our method is applied on top of the user's default interface, our approach is complementary to other work that develops state-of-the-art, domain-specific methods for problems like gaze tracking and handwriting recognition. Figure 2.1 describes our algorithm: we initialize our model using offline data generated by the default interface, deploy our interface as an augmentation to the default interface, collect online feedback, and update our model.

We formulate assistive typing as an online decision-making problem, in which the interface receives observations of user inputs, performs actions that select words or characters, and receives a reward signal that is automatically constructed from the user's backspaces. To improve the default interface's actions, we fit a neural network reward model that predicts the reward signal given the user's input and the interface's action. Upon observing a user input, our interface uses the trained reward model to update the prior policy given by the default interface to a posterior policy conditioned on optimality, then samples an action from this posterior (see Figure 2.1). We call this method *x-to-text* (X2T), where x refers to the arbitrary type of user input; e.g., eye gaze or brain activity.

Our primary contribution is the X2T algorithm for continual learning of a communication interface from user feedback. We primarily evaluate X2T through an online user study with 12 participants who use a webcam-based gaze tracking system to select words from a display. To run ablation experiments that would be impractical in the online study, we also conduct an observational study with 60 users who use a tablet and stylus to draw pictures of individual characters. The results show that X2T quickly learns to map input images of eye gaze or character drawings to discrete word or character selections. By learning from online feedback, X2T improves upon a default interface that is only trained once using supervised learning and, as a result, suffers from distribution shift (e.g., caused by changes in the user's head position and lighting over time in the gaze experiment). X2T automatically overcomes calibration problems with the gaze tracker by adapting to the mis-calibrations over time, without the need for explicit re-calibration. Furthermore, X2T leverages offline data generated by the default interface to accelerate online learning, stimulates co-adaptation from the user in the online study, and personalizes the interface to the handwriting style of each user in the observational study.

1.2 Learning to Infer Intent from User Input

In our problem setting, the user cannot directly perform actions; e.g., due to a sensorimotor impairment. Instead, the user relies on an assistive typing interface, where the user's intended

action is inferred from available inputs such as webcam images of eye gaze or handwritten character drawings. As such, we formulate assistive typing as a contextual bandit problem [49, 100, 56, 48, 30]. At each timestep, the user provides the interface with a context $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the set of possible user inputs (e.g., webcam images). The interface then performs an action $\mathbf{u} \in \mathcal{U}$, where $\mathcal{U}$ is the set of possible actions (e.g., word selections). We assume the true reward function is unknown, since the user cannot directly specify their desired task (e.g., writing an email or filling out a form). Instead of eliciting a reward function or explicit reward signal from the user, we *automatically construct a reward signal from the user's backspaces*. The key idea is to treat backspaces as feedback on the accuracy of the interface's actions.

Our approach to this problem is outlined in Figure 2.1. We aim to minimize expected regret, which, in our setting, is characterized by the total number of backspaces throughout the lifetime of the interface. While a number of contextual bandit algorithms with lower regret bounds have been proposed in prior work [50], we use a simple strategy that works well in our experiments: train a neural network reward model to predict the reward given the user's input and the interface's action, and select actions with probability proportional to their predicted optimality. Our approach is similar to prior work on deep contextual multi-armed bandits [17] and NeuralUCB [101], except that instead of using Thompson sampling or UCB to balance exploration and exploitation, we use a simple, stochastic policy.

Modeling User Behavior and Feedback

Unlike in the standard multi-armed bandit framework, we do not get to observe an extrinsic reward signal that captures the underlying task that the user aims to perform. To address this issue, we infer rewards from naturally-occurring user behavior. In particular, in the assistive typing setting, we take advantage of the fact that we can observe when the user *backspaces*; i.e., when they delete the most recent word or character typed by the interface. To infer rewards from backspaces, we make two assumptions about user behavior: (1) the user can perform a backspace action independently of our interface (e.g., by pressing a button); (2) the user tends to backspace incorrect actions; and (3) the user does not tend to backspace correct actions. Hence, we assign a positive reward to actions that were not backspaced, and assign zero reward to backspaced actions. Formally, let $r \in \{0, 1\}$ denote this reward signal, where $r = 0$ indicates an incorrect action and $r = 1$ indicates a correct action.

Training the Reward Model to Predict Feedback

In order to perform actions that minimize expected regret – i.e., the total number of backspaces over time – we need to learn a model that predicts whether or not the user will backspace a given action in a given context. To do so, we learn a reward model $p_\theta(r|\mathbf{x}, \mathbf{u})$, where p_θ is a neural network and θ are the weights. Since the reward $r \in \{0, 1\}$ can only take on one of two values, p_θ is a binary classifier. We train this binary classifier on a dataset $\mathcal{D}$

Algorithm 1 X-to-Text (X2T)

Require $\bar{\pi}, \theta_{\text{init}}$ ▷ default interface, pretrained reward model parameters
while true **do**
  $\mathbf{x} \sim p_{\text{user}}(\mathbf{x})$ ▷ user gives input
  $\mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x}) \propto p_\theta(r = 1|\mathbf{x}, \mathbf{u})\bar{\pi}(\mathbf{u}|\mathbf{x})$ ▷ interface performs action
  $r \leftarrow 0$ if user backspaces else 1 ▷ infer reward from user feedback
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}, \mathbf{u}, r)\}$ ▷ store online input-action-reward data
  $\theta \leftarrow \theta + \nabla_\theta \sum_{(\mathbf{x},\mathbf{u},r)\sim\mathcal{D}} \log\left(p_\theta(r|\mathbf{x}, \mathbf{u})\right)$ ▷ update reward model w/SGD

of input-action-reward triples $(\mathbf{x}, \mathbf{u}, r)$. In particular, we fit the model p_θ by optimizing the maximum-likelihood objective; i.e., the binary cross-entropy loss (see Equation 4.1 in the appendix).

Since X2T learns from human-in-the-loop feedback, the amount of training data is limited by how frequently the user operates the interface. To reduce the amount of online interaction data needed to train the reward model, we use offline pretraining. We assume that the user already has access to some default interface for typing. We also assume access to an offline dataset of input-action pairs generated by the user and this default interface. We assign zero rewards to the backspaced actions and positive rewards to the non-backspaced actions in this offline dataset, and initially train our reward model to predict these rewards given the user's inputs and the default interface's actions. Thus, when X2T is initially deployed, the reward model has already been trained on the offline data, and requires less online interaction data to reach peak accuracy.

Using the Reward Model to Select Actions

Even with offline pretraining, the initial reward model may not be accurate enough for practical use. To further improve the initial performance of our interface at the onset of online training, we combine our reward model $p_\theta(r|\mathbf{x}, \mathbf{u})$ with the default interface $\bar{\pi}(\mathbf{u}|\mathbf{x})$. We assume that $\bar{\pi}$ is a stochastic policy and that we can evaluate it on specific inputs, but do not require access to its implementation. We set our policy $\pi(\mathbf{u}|\mathbf{x}) = p(\mathbf{u}|\mathbf{x}, r = 1)$ to be the probability of an action conditional on optimality, following the control-as-inference framework [52]. Applying Bayes' theorem, we get $p(\mathbf{u}|\mathbf{x}, r = 1) \propto p(r = 1|\mathbf{x}, \mathbf{u})p(\mathbf{u}|\mathbf{x})$. The first term is given by our reward model p_θ, and the second term is given by the default interface. Combining these, we get the policy

$$\pi(\mathbf{u}|\mathbf{x}) \propto p_\theta(r = 1|\mathbf{x}, \mathbf{u})\bar{\pi}(\mathbf{u}|\mathbf{x}). \tag{1.1}$$

This decomposition of the policy improves the initial performance of our interface at the onset of online training, and guides exploration for training the reward model. It also provides a framework for incorporating a language model into our interface, as described in Section 1.4.

Our x-to-text (X2T) method is summarized in Algorithm 1. In the beginning, we assume the user has already been operating the default interface $\bar{\pi}$ for some time. In doing so, they

generate an 'offline' dataset that we use to train the initial reward model parameters θ_{init}. When the user starts using X2T, our interface π already improves upon the default interface $\bar{\pi}$ by combining the default interface with the initial reward model via Equation 1.1. As the user continues operating our interface, the resulting online data is used to maintain or improve the accuracy of the reward model. At each timestep, the user provides the interface with input $\mathbf{x} \sim p_{\text{user}}(\mathbf{x})$. Although standard contextual bandit methods assume that the inputs $\mathbf{x}$ are i.i.d., we find that X2T performs well even when the inputs are correlated due to user adaptation (see Section 1.4) or the constraints of natural language (see Section 1.4). The interface then uses the policy in Equation 1.1 to select an action $\mathbf{u}$. We then update the reward model p_θ, by taking one step of stochastic gradient descent to optimize the maximum-likelihood objective in Equation 4.1. Appendix 4.2 discusses the implementation details.

1.3 Related Work

Prior methods for training interfaces with supervised learning typically collect a dataset of input-action pairs, then fit a model that predicts actions given inputs. These methods tend to either assume access to ground-truth action labels from the user [3, 93, 42], or assume the user always intends to take the optimal action [29, 23, 22, 66]. Unfortunately, the user may not always be able to provide ground-truth action labels. Furthermore, in order for the system to compute optimal actions, the user must be instructed to perform specific calibration tasks for which the optimal policy is already known. These calibration tasks may not be representative of the tasks that the user intends to perform. This can lead to a distribution mismatch between the inputs that the model is trained on during the calibration phase, and the inputs that the model is evaluated on at test time when the user performs their desired tasks. Standard methods address this problem by periodically repeating the calibration process [96, 79], which can be time-consuming, disruptive, and requires assumptions about when and how frequently to re-calibrate. X2T overcomes these issues by continually learning from user feedback on tasks that naturally arise as the user types, rather than imposing separate training and test phases.

Extensive prior work on text entry systems [60] enables users to type using eye gaze [94], Braille [74], gestures [40], and palm keyboards for wearable displays [92]. X2T differs in that it enables the user to type using arbitrary inputs like eye gaze or handwriting, rather than a fixed type of input that restricts the flexibility of the system and must be anticipated in advance by the system designer.

X2T trains a typing interface through reinforcement learning (RL) with human-in-the-loop feedback instead of an explicit reward function. COACH [59, 6], TAMER [45, 95], and preference learning [85, 15] also train an RL agent without access to extrinsic rewards, but require explicit user feedback on the agent's behavior. X2T differs in that it learns from naturally-occurring feedback, which requires no additional effort from the user to train the agent. Other prior work trains RL agents from implicit signals, such as electroencephalog-

raphy [99], peripheral pulse measurements [63], facial expressions [36, 19], and clicks in web search [80]. X2T differs in that it trains an interface that always conditions on the user's input when selecting an action, rather than an autonomous agent that ignores user input after the training phase. Furthermore, X2T focuses on the assistive typing domain, where, to the best of our knowledge, backspaces have not yet been used to train an interface through RL. Related work on assistive robotic teleoperation interfaces proposes human-in-the-loop RL methods that minimally modify user actions [78, 10, 83, 86, 24, 39]. X2T differs in that it learns to operate on arbitrary types of user inputs, instead of assuming the user provides suboptimal actions.

1.4 Experimental Evaluation

We seek to answer the following questions: **Q1** (Sec. 1.4): Does X2T improve with use and learn to outperform a non-adaptive interface? **Q2** (Sec. 1.4): Does the user adapt to the interface while the interface adapts to the user? **Q3** (Sec. 1.4): Does X2T personalize the interface to different input styles? **Q4** (Sec. 1.4): Do offline pretraining and an informative prior policy accelerate online learning? **Q5** (Sec. 1.4): Does online learning improve the interface beyond the initialization provided by offline pretraining? **Q6** (Sec. 1.4): Can X2T improve the accuracy of a brain-computer interface for typing? To answer **Q1-2**, we run a user study with 12 participants who use webcam images of their eyes to type via gaze (see Figure 4.3). To answer **Q3-5**, we conduct an observational study with prerecorded images of handwritten characters drawn by 60 users with a tablet and stylus (see Figure 4.2). To answer **Q6**, we conduct a pilot study with one participant using an electrocorticography-based brain-computer interface. In our experiments, we use default interfaces that are not necessarily the state of the art in gaze tracking or handwriting recognition, but are instead chosen to test the hypotheses in **Q1-6**. Appendix 4.2 describes the experiment design in detail.

Adapting the Interface to the User

In this experiment, we aim to test X2T's ability to improve over time, relative to a non-adaptive, default interface. To that end, we formulate a gaze-based word selection task in which we display a list of words to the user (see Figure 4.3), ask them to look at their desired word, record an image from their webcam that captures their eye gaze, and predict their desired word. To measure objective performance, we ask the user to type specific goal sentences. To simplify the experiment, we restrict the action space $\mathcal{U} = \{1, 2, 3, ..., 8\}$ to the eight buttons on the screen, and always assign the next word in the sentence to one of those eight buttons (see Figure 4.3).

We evaluate (1) a default interface that uses iTracker [47] to estimate the user's 2D gaze position on the screen and select the nearest button, and (2) X2T. We calibrate the default interface once at the beginning of each experimental condition for each user, by asking the

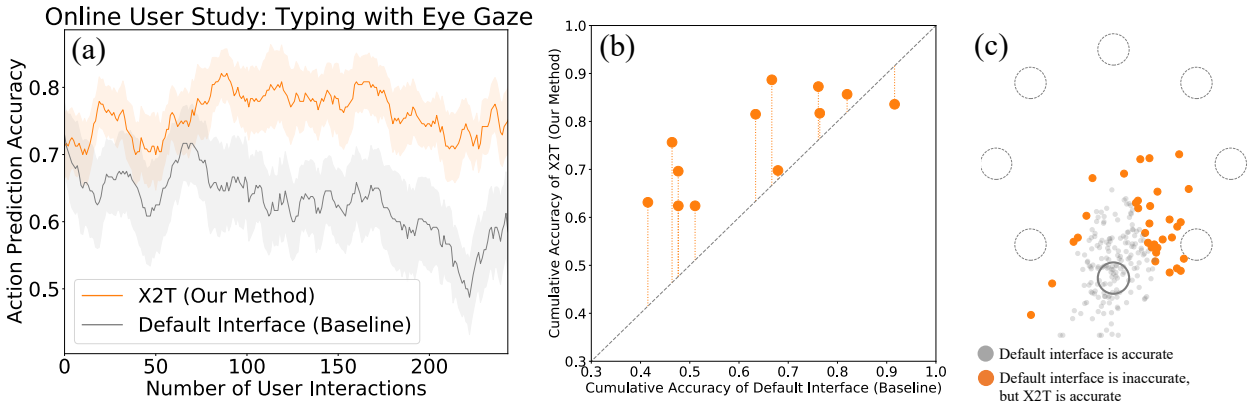


Figure 1.2: An online user study with 12 participants in the gaze tracking domain that addresses **Q1**: does X2T improve with use and learn to outperform a non-adaptive interface? **(a)** X2T predicts the user's intended action more accurately than the default interface, and the gap between the two methods grows over time. We smooth the curves using a moving average with a window size of 20 interactions, and measure standard error across the 12 users. **(b)** X2T improves the performance of 10 out of the 12 users, and the improvement from X2T is smaller when the default interface already performs well. Each orange circle represents one of the 12 users. The dashed gray line shows default-equivalent performance, and the dotted orange lines show the difference between X2T and default performance. Per-user accuracy is averaged across 250 interactions. **(c)** As shown in the screenshot in Figure 4.3, the user is shown a display of eight words arranged in a circle. Here, we plot the default interface's 2D gaze position estimates given user inputs intended to select the bottom-most button. The widely-scattered 2D estimates show that action prediction is particularly hard for this button, perhaps because the user's eyes tend to be more obscured when they are looking down. By training a reward model on user feedback, X2T helps the interface recover from incorrect gaze estimates.

user to look at each of the eight buttons one by one, recording 20 eye image samples for each button in 2 cycles, and training a 2D gaze position estimator using the iTracker method. After calibration, the default interface stays fixed, and does not adapt to the user within the session; in part, because we do not know in advance which word the user intends to select, so we cannot automatically harvest the necessary paired data of gaze images and targets to re-calibrate the interface. Instead of periodically interrupting the user to gather new paired data and re-calibrate, X2T continually adapts to the user throughout the session, following Algorithm 1. Appendix 4.2 describes the implementation of the default interface and X2T in further detail. We measure the performance of each method using the ground-truth accuracy of action selections. To access ground-truth actions for calculating accuracy, we instruct the user to try to select a specified word from the list displayed on their screen. Note that this instruction is not an essential component of X2T, and is used purely to evaluate objective performance in this experiment.

The results in panel (a) of Figure 1.2 show that at the onset of online learning, both X2T and the default interface predict the user's intended action with the same accuracy, but quickly diverge. The default interface's performance degrades over time, potentially due to distribution shift in the user's inputs caused by changes in head position, background

lighting, and other visual conditions. In contrast, X2T maintains the interface's strong initial performance throughout the experiment, by continually updating the reward model to reflect changes in the user's inputs. Panel (b) shows that X2T significantly improves the performance of 10 out of 12 participants, and that there are diminishing returns to X2T when the default interface already performs well. We ran a one-way repeated measures ANOVA on the action prediction accuracy dependent measure from the default and X2T conditions, with the presence of X2T as a factor, and found that $f(1, 11) = 17.23, p < 0.01$. The subjective evaluations in Table 4.1 in the appendix corroborate these results: users reported feeling that X2T selected the words they wanted and improved over time more than the default interface. Panel (c) qualitatively illustrates how X2T helps the interface recover from incorrect 2D gaze position estimates: each green 'x' shows that even when the default interface estimates a gaze position far from the intended button, which would normally cause an incorrect action prediction, the reward model can adjust the overall action prediction back to the correct button via Equation 1.1. We also find that X2T performs well even when the user's feedback is slightly noisy: the user backspaces mistakes, and does not backspace correct actions, in 98.6% of their interactions.

User Adaptation to the Interface

In the previous experiment, we tested X2T's ability to adapt the interface to the user. Prior work on human-machine co-adaptation [91, 12, 88, 73] and the evolution of communication protocols between humans [35] suggests that an adaptive interface may not only learn to assist a static user, but even stimulate the user to adapt their input style to the interface. In this experiment, we investigate whether the user adapts to the gaze-based interface described in Section 1.4. To do so, we perform a counterfactual experiment: instead of training and evaluating X2T on inputs $\mathbf{x}$ generated by the user while they were typing with X2T, we train and evaluate X2T on inputs $\mathbf{x}$ generated by the user while they were typing with the default interface. During the counterfactual experiment, instead of asking the user for new inputs, we simply replay the old inputs that were intended for the default interface, and automate backspaces. This enables us to test if the user adapted their inputs to X2T, or if the user provided the same distribution of inputs to both the default interface and X2T.

The results in Figure 1.3 suggest that the user does indeed adapt their input style to the interface, and that this user adaptation improves performance. Comparing X2T's actual performance (orange curve) to X2T's counterfactual performance on replayed user inputs that the user originally intended for the default interface (teal curve), we see that X2T is able to perform much better with inputs intended for X2T compared to inputs intended for the default interface. From this result alone, one might infer that the user provided X2T with inputs that were more generically predictable than the inputs they provided to the default interface. However, by comparing the default interface's performance on replayed user inputs that the user originally intended for X2T (gray curve) to X2T's performance on the same inputs (orange curve), we see that X2T performs better than the default interface on the same inputs. This suggests that the user's inputs to X2T are not merely easier to predict,

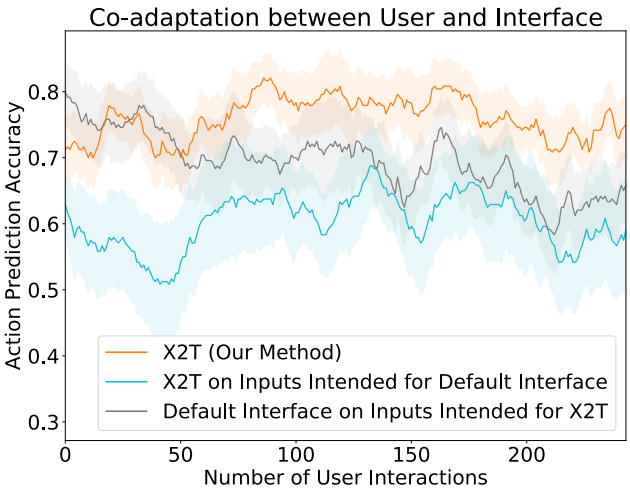


Figure 1.3: A counterfactual experiment with the online user study data that addresses **Q2**: does the user adapt to the interface while the interface adapts to the user? Training and evaluating X2T on user inputs originally intended for the default interface leads to worse performance than doing so on user inputs intended for X2T (orange vs. teal). Evaluating the default interface on inputs intended for X2T leads to worse performance than evaluating X2T on the same inputs (orange vs. gray). These results suggest that the user adapts their input style specifically to X2T, and that this user co-adaptation improves performance. We smooth the curves using a moving average with a window size of 20 interactions, and measure standard error across the 12 users.

but in fact adapted specifically to X2T. In other words, X2T stimulates user co-adaptation to the interface.

Personalizing the Interface for Different Users

In this experiment, we demonstrate X2T's ability to operate on a different type of user input: drawings of characters (see Figure 4.2 in the appendix); which, for some users, can be easier to provide than direct keystrokes [96]. We also investigate to what extent X2T learns a personalized interface that is uniquely adapted to the individual user. To that end, we analyze handwriting samples from 60 users, collected through a tablet and stylus, from the UJI Pen Characters Database [57]. Each sample consists of a sequence of 2D positions that traces the trajectory of the user's pen tip as they draw a known character. We conduct an observational study with this data by sampling goal sentences, replaying a randomly-selected handwriting sample of the user's desired next character from the goal sentence, and treating each drawing as the user input $\mathbf{x}$; akin to the replay experiment in Section 1.4. This observational study is equivalent to an online study, except that it does not permit user co-adaptation to the interface since it replays logged user inputs, and assumes that the feedback signal is not noisy when automating backspaces. To test X2T's robustness to noise and distributional shift in the user's input, we perturb the replayed pen tip positions by

	Evaluation			
Training	User 1	User 2	User 3	User 4
User 1	**.995**	.033	.025	.017
User 2	.018	**.463**	.009	.061
User 3	.000	.002	**.975**	.039
User 4	.018	.002	.025	**.993**

Table 1.1: An observational study with 60 users in the handwriting recognition domain that addresses **Q3**: does X2T personalize the interface to different input styles? We measure action prediction accuracy across 1000 interactions, and randomly sample users 1-4 from the pool of 60 users. The interface trained on user i is substantially more accurate when evaluated on inputs from user i than on inputs from user j, suggesting that the learned interface is personalized to each individual user.

adding Brownian noise (see Figure 4.2 in the appendix).

We evaluate (1) a default interface trained to classify handwritten EMNIST characters [16], and (2) X2T. We intentionally train the default interface on EMNIST images instead of UJI Pen Characters drawings, to model real-world applications with a distribution mismatch between the training data and test data, as discussed in Section 1.3. To address the challenge of selecting from 27 possible character actions, we use a language model [21] to predict the prior likelihood of an action $p_{\text{LM}}(\mathbf{u}_t|\mathbf{u}_{0:t-1})$ given the preceding characters $\mathbf{u}_{0:t-1}$ in the user's text field. We use this language model in both the default interface and X2T. In particular, we set $\bar{\pi}(\mathbf{u}|\mathbf{x}) \propto p_\phi(\mathbf{u}|\mathbf{x})p_{\text{LM}}(\mathbf{u}|\mathbf{u}_{0:t-1})$, where p_ϕ is the EMNIST image classifier. As in Section 1.4, the default interface stays fixed throughout the experiment and does not adapt to the user, since re-training the default interface would require interrupting the user to collect paired data.

The results in Figure 1.4 show that X2T significantly outperforms the default interface (orange vs. gray). We ran a one-way repeated measures ANOVA on the action prediction accuracy dependent measure from the default and X2T conditions, with the presence of X2T as a factor, and found that $f(1, 59) = 37.46, p < 0.0001$. Furthermore, Table 1.1 shows that X2T learns an interface that is particularly suited to the user whose data it was trained on: when an interface trained on user i's data is evaluated on data from user $j \neq i$ instead of user i, performance degrades substantially.

Ablation Experiments

In this experiment, we aim to test the importance of three components of X2T: offline pretraining, an informative prior policy, and online learning. As discussed in Sections 1.2 and 1.2, to improve the initial performance of X2T and accelerate online learning, we pretrain on offline data collected using the default interface, and incorporate prior knowledge from the default interface into the prior policy $\bar{\pi}$ in Equation 1.1. Using the handwriting recognition task from Section 1.4, we conduct ablation experiments in which we drop out each of the

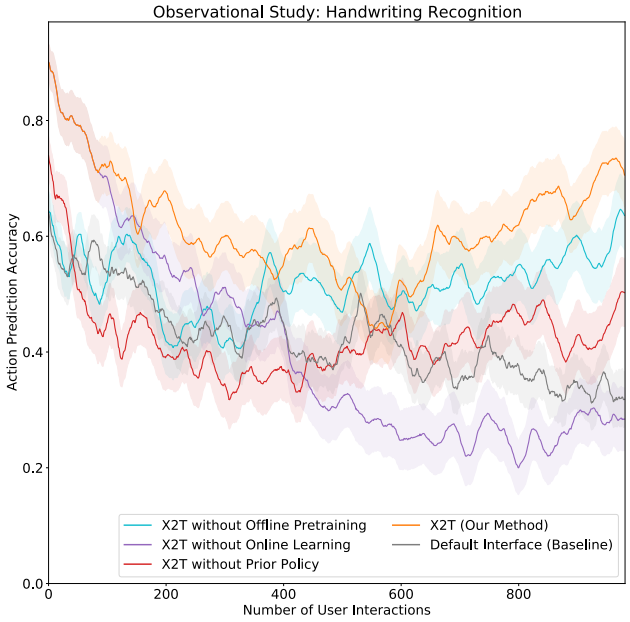


Figure 1.4: An observational study with 60 users in the handwriting recognition domain that addresses **Q4-5**: do offline pretraining and an informative prior policy accelerate online learning, and does online learning improve the interface beyond the initialization provided by offline pretraining? In this ablation experiment, we remove each of the three components – offline pretraining, an informative prior policy, and online learning – one by one, and find that each component is critical for maintaining high action prediction accuracy at different stages of the experiment.

three components, one by one, and measure any resulting changes in performance. In the first condition, we test the effect of not performing offline pretraining, by initializing X2T with random weights θ_{init}. In the second, we test the effect of not incorporating prior knowledge into our policy, by setting the prior policy $\bar{\pi}$ to be a uniform distribution over actions. In the third, we test the effect of not learning online and instead relying solely on offline pretraining, by freezing the reward model parameters after offline pretraining and not storing online data.

The results in Figure 1.4 show that offline pretraining is helpful at the onset of online learning (orange vs. teal), but does not have a substantial effect on performance after some online data has been collected. This is unsurprising, since leveraging offline data is only necessary when insufficient online data has been collected. Using the default interface as a prior policy in Equation 1.1 is critical for X2T's performance throughout the experiment (orange vs. red). This is also unsurprising, given that the default interface contains useful prior knowledge about how to interpret user inputs. Online learning is not particularly helpful at the onset of the experiment (orange vs. purple), but has a substantial effect on performance over time. This result shows that learning from on-policy data is critical for X2T. In other words, X2T learns best when it learns from its own mistakes, rather than the mistakes of the default interface.

Pilot Study with a Brain-Computer Interface

In this experiment, we demonstrate X2T's ability to improve the typing accuracy of a brain-computer interface (BCI) that uses a 128-channel chronic electrocorticography (ECoG) implant to measure neural activity [51, 89]. Each user input has 128 features, which are obtained by pre-processing the raw ECoG signals (Appendix 4.1 discusses the details). The results in Figure 1.5 show that, after offline pretraining on a dataset of 2913 input-action-reward tuples, X2T achieves 60.7% action prediction accuracy during a new session of 300 steps. We train a default interface on paired data of inputs and ground-truth actions collected before the offline data. To evaluate the default interface, we conduct a counterfactual experiment (similar to those in previous sections) on the 300 steps of data from the X2T evaluation session. The default interface only achieves 46.3% accuracy. Interestingly, the default interface never predicts the top-most button action, leading to a 0% recall rate for that action. This is most likely due to changes in the feature processing pipeline or other kinds of distributional shift in the user inputs over time. In contrast, X2T achieves 70.2% recall on the top-most button action, suggesting that it can overcome these challenges.

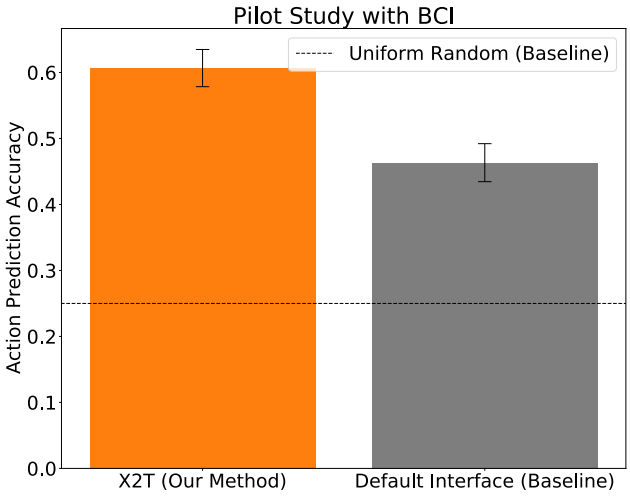


Figure 1.5: A pilot study with one participant that addresses **Q6**: can X2T improve the accuracy of a brain-computer interface (BCI) for typing? We find that X2T adapts to recent input-action-reward data and outperforms a default interface that is trained via supervised learning on older, paired input-action data.

1.5 Discussion

In our online user study on gaze-based word selection, we show that X2T learns to outperform a non-adaptive interface in under 30 minutes (including offline data collection), and that the user simultaneously adapts their input style to the interface. The observational study on handwritten character recognition shows that X2T can also personalize the interface to individual users.Additionally, our pilot study with a brain-computer interface user shows that X2T improves the recall rate of one particular action from 0% to 70.2%. These experiments broadly illustrate how online learning from user feedback can be used to train a human-machine interface.

One limitation of this work is that we assume backspaces can be generated independently of X2T. This assumption restricts X2T to settings where the user can communicate a binary signal; e.g., through a button press or a sip-and-puff interface. One direction for future work

is to relax this assumption by training a binary classifier to generate feedback signals from, e.g., facial expressions or brain activity. X2T is also limited in that the benefit from using X2T to fine-tune the default interface may decrease as the default interface is improved, as suggested by the diminishing returns in Figure 1.2. One direction for future empirical work is to test X2T with state-of-the-art default interfaces. In spite of these limitations, methods like X2T that learn from their mistakes and stimulate user co-adaptation provide a general mechanism for improving user interfaces; not only for assistive typing, but also for other domains, such as brain-computer interfaces for prosthetic limb control and augmented reality devices for visually-impaired users.

1.6 Acknowledgments

Thanks to Sarah Seko, Reza Abiri, Yasmin Graham, and members of the Neural Engineering and Plasticity lab at UC San Francisco for helping to conduct the brain-computer interface experiments in Section 1.4. Thanks to members of the InterACT and RAIL labs at UC Berkeley for feedback on this project. This work was supported in part by an NVIDIA Graduate Fellowship, AFOSR FA9550-17-1-0308, NSF NRI 1734633, NIH New Innovator Award [1 DP2 HD087955], and UCSF Weill Institute for Neurosciences. This work was originally published as the conference paper "X2T: Training an X-to-Text Typing Interface with Online Learning from User Feedback" at the 2021 International Conference on Learning Representations (ICLR) [28].

Chapter 2

ASHA: Assistive Teleoperation via Human-in-the-Loop Reinforcement Learning

Building assistive interfaces for controlling robots through arbitrary, high-dimensional, noisy inputs (e.g., webcam images of eye gaze) can be challenging, especially when it involves inferring the user's desired action in the absence of a natural 'default' interface. Reinforcement learning from online user feedback on the system's performance presents a natural solution to this problem, and enables the interface to adapt to individual users. However, this approach tends to require a large amount of human-in-the-loop training data, especially when feedback is sparse. In this chapter, we propose a hierarchical solution that learns efficiently from sparse user feedback: we use offline pre-training to acquire a latent embedding space of useful, high-level robot behaviors, which, in turn, enables the system to focus on using online user feedback to learn a mapping from user inputs to desired high-level behaviors. The key insight is that access to a pre-trained policy enables the system to learn more from sparse rewards than a naïve RL algorithm: using the pre-trained policy, the system can make use of successful task executions to relabel, in hindsight, what the user actually meant to do during unsuccessful executions. We evaluate our method primarily through a user study with 12 participants who perform tasks in three simulated robotic manipulation domains using a webcam and their eye gaze: flipping light switches, opening a shelf door to reach objects inside, and rotating a valve. The results show that our method successfully learns to map 128-dimensional gaze features to 7-dimensional joint torques from sparse rewards in under 10 minutes of online training, and seamlessly helps users who employ different gaze strategies, while adapting to distributional shift in webcam inputs, tasks, and environments.

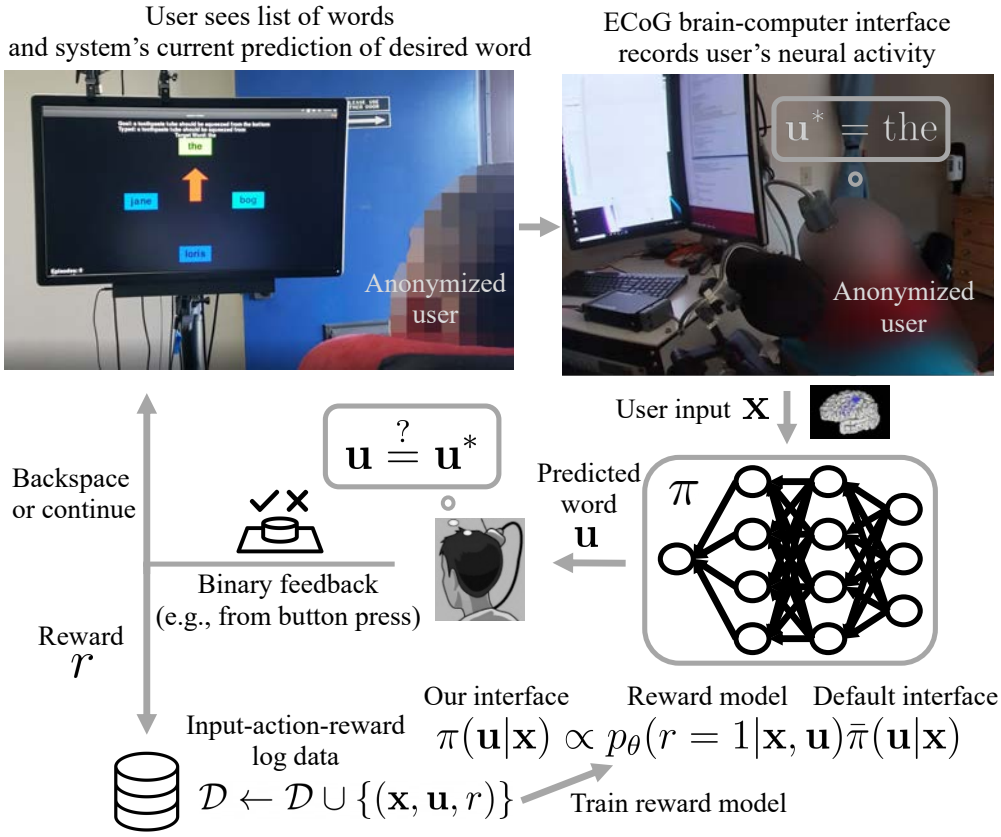


Figure 2.1: In this example, the user directs their gaze to control a wheelchair-mounted Jaco arm to push different light switches. During the autonomous pre-training phase, ASHA learns a task-conditioned policy $\pi^{\text{spec}}_{\psi,\phi}$ that can flip various switches. This policy is decomposed into a latent variable model with two components: a specification encoder f^{spec}_{ψ}, which maps a task specification τ^{spec} (e.g., goal state) to a latent embedding $\mathbf{z}$; and a latent-conditioned policy g_ϕ. In phase 1, we jointly pre-train f^{spec}_{ψ} and g_ϕ to flip switches. In phase 2, we use human-in-the-loop RL to train an interface $\pi^{\text{inpt}}_{\theta,\phi}$ that enables the user to control the arm using their eye gaze, and perform new tasks sampled from the same distribution as the pre-training tasks. To speed up learning, this interface is also represented as a latent variable model with two components: an input encoder f^{inpt}_{θ}, which maps the user's control input $\mathbf{x}$ (e.g., webcam image) to a high-level, latent action $\mathbf{z}$; and the pre-trained latent-conditioned policy g_ϕ.

2.1 Introduction

Shared-control teleoperation interfaces can help users control systems like robotic arms and wheelchairs more effectively [43, 65, 11, 4, 37]. For example, they can help users perform dexterous robotic manipulation tasks by automatically maximizing contact area with grasped objects [102], or enable control via complex user input streams like eye gaze [7, 5] and brain-computer interfaces [67]. 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 an arbitrary, high-dimensional, noisy control input (e.g., webcam image of eye gaze). This stands in contrast to

prior work on shared autonomy that assumes the user already has a viable interface for direct teleoperation and only aims to improve the user's performance by minimally intervening in the user's actions to avoid collisions [10, 83, 86, 24], preserving the reachability of potential goals [39], or inferring goals and acting to reach them [33, 38, 76, 46, 68]. Other prior methods do not require a direct teleop interface, and instead perform supervised calibration on paired examples of inputs and actions [29, 23, 22, 66, 93, 3, 42, 27]. However, 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.

In this paper, we consider a different problem setting than the aforementioned prior work: instead of requiring a direct teleop interface or limiting data collection to explicit calibration phases, we elicit user feedback on the system's online performance and train the interface through reinforcement learning (RL) [90]. 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 [82, 62, 77], the more competent and personalized the interface becomes. Note that, in contrast to prior work on human-in-the-loop RL like COACH [59, 6], TAMER [45, 95], and preference learning [85, 15], we aim to train an interface that enables the user to control the robot at test time and perform different tasks, instead of training the robot to autonomously perform a single task. The main challenge is that, due to the sparsity of rewards, it can require a large amount of training data, which may be impractical for an individual user operating a physical robot.

We propose a hierarchical solution to this challenge: use offline pre-training to learn to perform potentially-useful tasks, then use online user feedback to learn a mapping from user inputs to robot behaviors (see Figure 2.1). In the pre-training phase, we train a task-conditioned policy to perform a wide variety of tasks without the user in the loop (e.g., rotating a valve to various target angles), and automatically discover useful, high-level robot behaviors in the process (e.g., rotating the valve clockwise or counter-clockwise). Then, in the online learning phase, we bring the user into the loop, and use RL with sparse, user-provided rewards to learn how to interpret the user's inputs as desired high-level behaviors. We leverage the pre-trained policy to extract more information from the user's sparse online rewards 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 call this algorithm *ASsistive teleoperation via HumAn-in-the-loop reinforcement learning* (ASHA).

We primarily evaluate ASHA through a user study with 12 participants who use a webcam and their eye gaze to perform tasks in three simulated manipulation domains: flipping light switches, opening a shelf door to reach objects inside, and rotating a valve (see Figure 2.2 for

screenshots). The results show that our method successfully learns to map 128-dimensional gaze features to 7-dimensional joint torques from sparse rewards in under 10 minutes of online training, while adapting to distributional shift in the user's webcam input caused by changes in ambient lighting and head position (Section 2.3); changes in the user's set of desired tasks, like the addition of a new light switch (Section 2.3); and changes in environmental conditions, like whether a shelf door is initially open or closed (Section 2.3). In both domains, ASHA increased success rates for the majority of users, compared to a non-adaptive baseline interface. Even though 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 – ASHA was able to seamlessly adapt to the different communication styles by learning from individual user data.

2.2 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. We do not require prior knowledge of how to parse the user's input, and instead treat the user's input as a raw, undifferentiated bitstream. 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) [41]. The state consists of the state of the environment $\mathbf{s}_t$ (e.g., the position and orientation of the robot) and the user's desired task $\mathcal{T}$ (e.g., flipping a particular light switch). The observation consists of the state of the environment $\mathbf{s}_t$ and the user's control input $\mathbf{x}_t$ (e.g., an image of their eyes that captures gaze direction), but not the task $\mathcal{T}$. The user's control input $\mathbf{x}_t$ communicates their intent to the robot. We do not assume access to the user's desired task $\mathcal{T}$, since this can be difficult for the user to specify. 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 $\pi^{\text{inpt}}(\mathbf{a}_t|\mathbf{s}_{0:t}, \mathbf{x}_{0:t})$ 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 2.1. 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 [83]. 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 policy $g(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})$ that is parameterized by a high-level latent variable $\mathbf{z}$ and can perform potentially-useful tasks; and (2) learning a mapping $f^{\text{inpt}}(\mathbf{z}|\mathbf{s}_{0:t}, \mathbf{x}_{0:t})$ from the user's control inputs $\mathbf{x}$ to the user's desired high-level behavior $\mathbf{z}$.

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, and continuous input from the user will be used to infer the desired behavior in terms of this basis. This space can also be viewed as a reparameterization of the policy space: instead of searching over all possible policy parameters during the human-in-the-loop learning phase, the system will search over high-level behaviors in the latent space acquired during this autonomous pre-training phase.

During phase 1, we assume the ability to sample tasks $\mathcal{T}_i \sim p(\mathcal{T})$, a specification τ_i^{spec} of each task, and a reward function $R_i(\mathbf{s}_t, \mathbf{a}_t)$ for each task. These reward functions R_i are only used for pre-training, and are not required during phase 2 of human-in-the-loop learning, when the user will perform new, unknown tasks drawn from the same distribution $p(\mathcal{T})$. Also note that the specification τ_i^{spec} 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 τ_i^{spec} to be the 3D position of the target object or 1D target angle of the valve, and define a reward function R_i for each task. 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 [34] and represent the robot's policy as a latent variable model,

$$\pi_{\psi,\phi}^{\text{spec}}(\mathbf{a}_t|\mathbf{s}_t; \tau_i^{\text{spec}}) \triangleq \mathbb{E}_{\mathbf{z}\sim f_\psi^{\text{spec}}(\mathbf{z}|\tau_i^{\text{spec}})}[g_\phi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})], \tag{2.1}$$

where f_ψ^{spec} is the 'specification encoder', g_ϕ is the 'latent-conditioned policy', $\pi_{\psi,\phi}^{\text{spec}}$ is the composition of f_ψ^{spec} and g_ϕ, $\mathbf{z} \in \mathbb{R}^d$ is a latent variable that characterizes the task (we set $d = 3$ in our experiments), the prior distribution of $\mathbf{z}$ is the standard normal distribution $\mathcal{N}(\mathbf{0}, I_d)$, and the action $\mathbf{a}_t$ is conditionally independent of the specification τ_i^{spec} given the state $\mathbf{s}_t$ and latent embedding $\mathbf{z}$. At the beginning of each pre-training episode, we sample a task $\mathcal{T}_i \sim p(\mathcal{T})$. We then jointly pre-train the latent-conditioned policy g_ϕ and specification encoder f_ψ^{spec} to optimize the task rewards R_i using RL – in our implementation, we use the soft actor-critic algorithm (SAC) [32]. An important consequence of the latent variable model in Equation 2.1 is that, in addition to optimizing the task rewards R_i, we regularize the latent embedding $\mathbf{z}$ to its prior distribution $\mathcal{N}(\mathbf{0}, I_d)$ using a variational information bottleneck (VIB) [2, 1]. The VIB encourages the model to learn a smooth, compressed latent space that shares information across tasks, and encourages the specification encoder f_ψ^{spec} to discard task-irrelevant information about the specification τ_i^{spec} from the embedding $\mathbf{z}$ – 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 2.4).

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 2.1), we turn to the problem of learning an interface that maps user inputs to desired high-level behaviors (the right half of Figure 2.1). We represent the interface as a latent variable model that reuses the pre-trained latent-conditioned policy g_ϕ,

$$\pi^{\text{inpt}}_{\theta,\phi}(\mathbf{a}_t|\mathbf{s}_{0:t}, \mathbf{x}_{0:t}) \triangleq \mathbb{E}_{\mathbf{z}\sim f^{\text{inpt}}_\theta(\mathbf{z}|\mathbf{s}_{0:t},\mathbf{x}_{0:t})}[g_\phi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})], \tag{2.2}$$

where f^{inpt}_θ is the 'input encoder', $\pi^{\text{inpt}}_{\theta,\phi}$ is the composition of f^{inpt}_θ and g_ϕ, and the prior distribution of $\mathbf{z}$ is the standard normal distribution $\mathcal{N}(\mathbf{0}, I_d)$. Note that the input encoder f^{inpt}_θ differs from the specification encoder f^{spec}_ψ learned in the pre-training phase: f^{inpt}_θ reads in the user's control input $\mathbf{x}$ (e.g., gaze), while f^{spec}_ψ takes a specification τ^{spec} (e.g., goal state) as input instead. Since the user's inputs $\mathbf{x}$ are only partial observations of the state variable $\mathcal{T}$ that defines the task, the interface $\pi^{\text{inpt}}_{\theta,\phi}$ is generally conditioned on the full sequence of states $\mathbf{s}_{0:t}$ and inputs $\mathbf{x}_{0:t}$. However, in our switch and bottle experiments, we find that conditioning on only the most recent state $\mathbf{s}_t$ and input $\mathbf{x}_t$ works well empirically.

Given the latent variable model in Equation 2.2, we train f^{inpt}_θ through RL from user feedback. Recent work in this area [28] 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. Unfortunately, due to the sparsity of the rewards, this approach would typically require a prohibitive amount of human interaction (see **Q3** in Section 2.4). 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 $\mathcal{D}$ denote the set of successful trajectories. From each of these successes, we extract a specification τ^{spec} of the user's desired task at the time – e.g., in the switch and bottle domains, we set τ^{spec} to be the final 3D position of the object manipulated in the successful trajectory, analogous to the pre-training phase in Section 2.2. We then combine this task specification τ^{spec} with the pre-trained task-conditioned policy $\pi^{\text{spec}}_{\psi,\phi}$ to represent the optimal policy $\pi^{\text{spec}}_{\psi,\phi}(\mathbf{a}_t|\mathbf{s}_t, \tau^{\text{spec}})$ via Equation 2.1. The key idea is to

match the interface $\pi^{\text{inpt}}_{\theta,\phi}$ with the optimal policy $\pi^{\text{spec}}_{\psi,\phi}$, by optimizing the loss,

$$\mathcal{L}(\theta) = \sum_{\tau \in \mathcal{D}, t} D_{\text{KL}}(\pi^{\text{spec}}_{\psi,\phi}(\cdot|\mathbf{s}_t, \tau^{\text{spec}}) \parallel \pi^{\text{inpt}}_{\theta,\phi}(\cdot|\mathbf{s}_{0:t}, \mathbf{x}_{0:t})) + \beta D_{\text{KL}}(f^{\text{inpt}}_{\theta}(\cdot|\mathbf{s}_{0:t}, \mathbf{x}_{0:t}) \parallel \mathcal{N}(\mathbf{0}, I_d)), \tag{2.3}$$

where the second term is the VIB for the latent variable model in Equation 2.2, and β is a regularization constant. By minimizing the divergence between the policies induced by the input encoder f^{inpt}_{θ} and the pre-trained specification encoder f^{spec}_{ψ}, we force f^{inpt}_{θ} to infer a latent embedding that induces the same low-level action distribution as the embedding inferred by f^{spec}_{ψ}. This helps to reduce the amount of human interaction required to train the system (see **Q5** in Section 2.4). Note that optimizing Equation 2.3 does not necessarily force both encoders to produce the same embedding, since different embeddings can induce the same low-level action distribution. This keeps our method flexible, and enables the user to guide the robot to subgoals.

Learning efficiently from failures in hindsight. Instead of simply treating failed trajectories as examples of behavior that achieved zero reward, we take the final, successful trajectory at the end of a string of failed trajectories that attempted to perform the same task, extract a task specification τ^{spec} from this successful trajectory, compute the optimal policy $\pi^{\text{spec}}_{\psi,\phi}(\mathbf{a}_t|\mathbf{s}_t, \tau^{\text{spec}})$ for all the states in the success *and the failures*, and optimize the loss in Equation 2.3. The key idea is that successful episodes enable us to compute the optimal policy for the most recent failure episodes, because a string of failures and eventual success are all attempts to perform the same task. This helps to minimize the human interaction required to train the system (see **Q4** in Section 2.4).

Algorithm Summary

Our complete assistive teleoperation method is summarized in Algorithm 2. We initially pre-train the latent-conditioned policy g_ϕ and specification encoder f^{spec}_{ψ} with a set of task specifications and reward functions $\{\tau^{\text{spec}}_i, R_i\}_i$ using a standard RL algorithm with a VIB – our implementation constructs several goal-reaching tasks and pre-trains on them with SAC. We then begin training the input encoder f^{inpt}_{θ} with the user in the loop. First, the user decides on a task $\mathcal{T}$, which we assume is sampled from the same distribution $p(\mathcal{T})$ as the pre-training tasks. At each timestep t, the environment generates the next state $\mathbf{s}_t$, and the user provides the system with input $\mathbf{x}_t$. After seeing the input $\mathbf{x}_t$, the robot takes an action $\mathbf{a}_t$ sampled from the interface $\pi^{\text{inpt}}_{\theta,\phi}$ defined by the input encoder f^{inpt}_{θ} and the pre-trained latent-conditioned policy g_ϕ via Equation 2.2. 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 τ^{spec} from it, use the pre-trained specification encoder f^{spec}_{ψ} and latent-conditioned policy g_ϕ to define an optimal policy for the task via Equation 2.1, and train the input encoder f^{inpt}_{θ} to induce actions that match that optimal policy, by

Algorithm 2 Assistive Teleoperation via Human-in-the-Loop Reinforcement Learning (ASHA)

1: $g_\phi, f_\psi^{\text{spec}} \leftarrow \text{RL}(\{\tau_i^{\text{spec}}, R_i\}_i)$ ▷ pre-train the latent-conditioned policy and specification encoder autonomously
2: **while** true **do**
3: $\quad\mathcal{T} \sim p(\mathcal{T})$ ▷ user chooses a task
4: $\quad\mathcal{D} \leftarrow []$ ▷ initialize empty list of trajectories for current task
5: $\quad$**while** robot has not succeeded at task $\mathcal{T}$ yet **do**
6: $\quad\quad\tau \leftarrow []$ ▷ initialize empty trajectory
7: $\quad\quad\mathbf{s}_0 \sim p(\mathbf{s}_0)$ ▷ reset environment
8: $\quad\quad$**for** $t \in \{0, 1, 2, ..., T-1\}$ **do**
9: $\quad\quad\quad\mathbf{x}_t \leftarrow$ user's control input
10: $\quad\quad\quad\mathbf{a}_t \sim \pi_{\theta,\phi}^{\text{inpt}}(\mathbf{a}_t|\mathbf{s}_{0:t}, \mathbf{x}_{0:t})$ ▷ robot performs action
11: $\quad\quad\quad\tau$.append$(\mathbf{s}_t, \mathbf{x}_t)$
12: $\quad\quad\quad\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ environment evolves
13: $\quad\quad\mathcal{D}$.append(τ) ▷ store trajectory (even if a failure)
14: $\quad\tau^{\text{spec}} \leftarrow$ final, successful trajectory τ in $\mathcal{D}$
15: $\quad\theta \leftarrow \theta - \nabla_\theta \sum_{\tau \in \mathcal{D}, t} D_{\text{KL}}(\pi_{\psi,\phi}^{\text{spec}}(\cdot|\mathbf{s}_t, \tau^{\text{spec}}) \parallel \pi_{\theta,\phi}^{\text{inpt}}(\cdot|\mathbf{s}_{0:t}, \mathbf{x}_{0:t}))$
16: $\quad+\text{VIB}(\theta)$ ▷ update input encoder

optimizing the loss in Equation 2.3. We find that training f_θ^{inpt} to convergence in line 16 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_θ^{inpt}.

2.3 User Studies

In our experiments, we evaluate to what extent ASHA can adapt to the user's inputs (Section 2.3), to users that want to perform new tasks (Section 2.3), and to changes in the environment (Section 2.3). We conduct a user study with 12 participants who control a simulated 7-DoF Jaco robotic arm using gaze (see Figure 2.2). The interface receives 128-dimensional feature vectors that represent the user's webcam image inputs $\mathbf{x}_t$, and outputs 7-dimensional joint torques as actions $\mathbf{a}_t$. The users perform tasks in three simulated manipulation domains implemented with the PyBullet real-time physics simulator [18] using assets from Assistive Gym [25]: flipping light switches, opening a shelf to reach objects inside, and rotating a valve.

Adapting to Distributional Shift in Gaze Inputs

In this experiment, we aim to test ASHA's ability to improve over time by learning from user feedback. We compare to a non-adaptive baseline interface that is initially calibrated

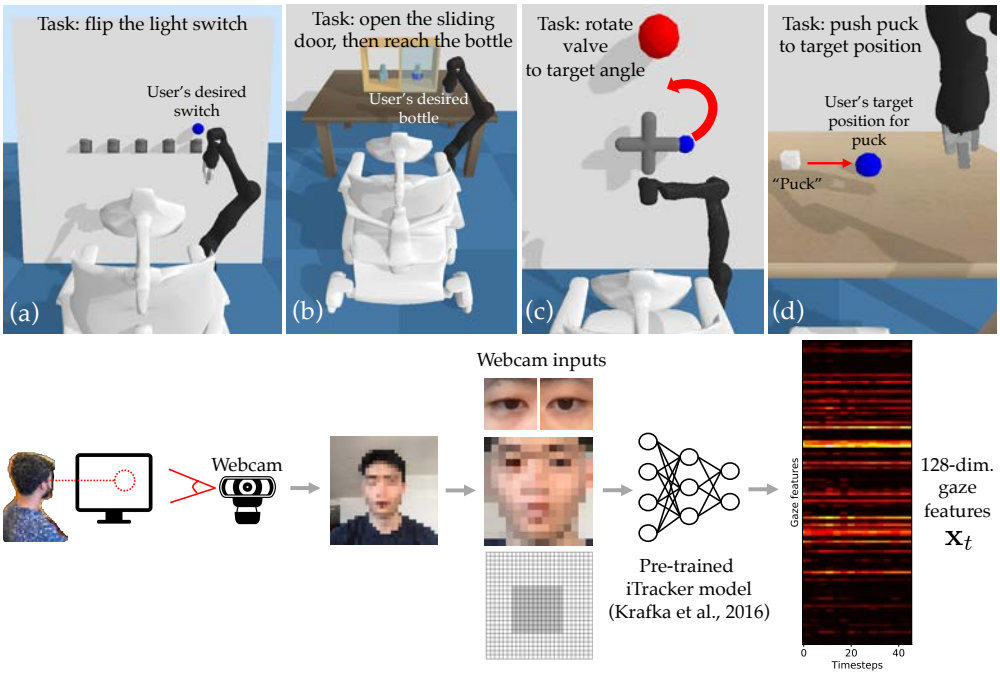


Figure 2.2: The user sees the simulated environment through the point of view of someone sitting in the wheelchair, and directs their gaze to communicate what they want the robot to do. In the switch domain (a), the user must push down on the blue lever. In the bottle domain (b), the user must open the sliding door if necessary, then reach for the blue bottle. In the valve domain (c), the user must rotate the valve so that the blue tip points at the red sphere. In the puck domain (d), the user must push the white puck to the blue target.

via supervised learning, but does not adapt during deployment (analogous to the prior work discussed in Section 2.1). 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 switch, 8 in bottle, and 8 in valve. We then train the baseline's input encoder $f_{\theta_0}^{\text{inpt}}$ on the objective in Equation 2.3, treating the paired data as a set of successful trajectories $\mathcal{D}$. We refer to this supervised learning procedure as 'calibration'. Note that this implicitly assumes that the user's passive inputs are equivalent to their active control inputs, which is often not the case in practice [20, 97]. To improve the initial performance and sample efficiency of our method, we initialize ASHA's input encoder f_{θ}^{inpt} with the calibrated baseline parameters θ_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). We calibrate and evaluate on the same distribution of tasks: in the switch domain, a

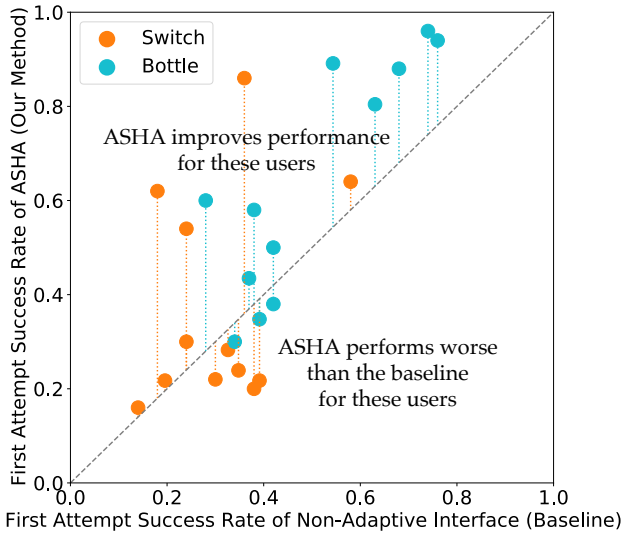


Figure 2.3: Each circle represents the success rate for one participant, averaged over 50 online episodes (11 minutes).

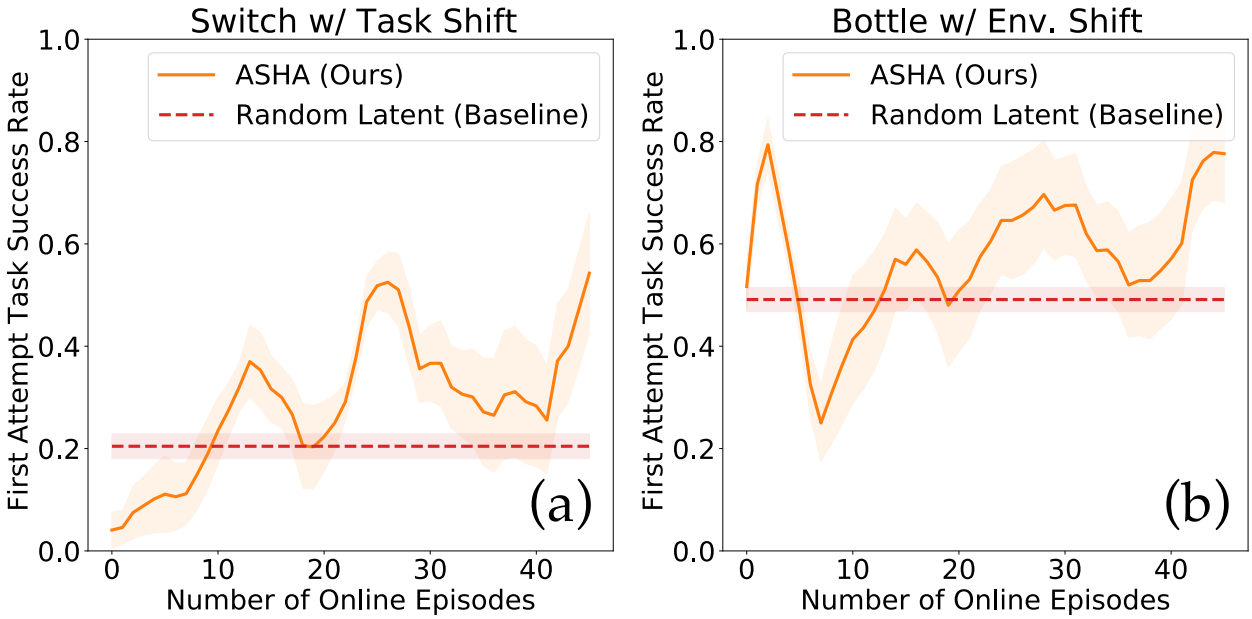
uniform distribution over flipping one of the three switches in the middle; and in the bottle domain, a uniform distribution over reaching one of the two bottles. To establish a lower bound on performance, we also compare to a baseline that randomly samples a latent $\mathbf{z}$ and executes the policy $g_\phi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z})$, without taking any user input.

The results in Figure 2.3 show that ASHA improves the success rates of the majority of users, relative to the non-adaptive baseline. 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. We ran a one-way repeated measures ANOVA on the success rates from the baseline and ASHA conditions, with the presence of ASHA as a factor, and found that $f(1, 11) = 8.26, p < .05$ in the switch domain, and $f(1, 11) = 7.28, p < .05$ in the bottle domain. Subjective evaluations corroborate these results: users reported feeling more in control of the robot with ASHA compared to the baseline.

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

Figure 2.4: Shift in Task Distribution or Environment



calibrate the input encoder on paired data generated from one distribution of tasks – a uniform distribution over the 2nd and 3rd switches from the left – then evaluate online on a different distribution of tasks – a uniform distribution over the 2nd, 3rd, and 4th switches from the left. This is challenging, since examples of the 4th 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 2.4a 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.

Adapting to a Changing Environment

Adaptation is useful, not only because the user's input might drift or their desired tasks might be novel relative to the training tasks, 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. Figure 2.4b shows that ASHA adapts to the new environmental conditions, increasing the success rate over time.

Table 2.1: Ablation Experiments

	Switch	Bottle
Random Latent (Baseline)	0.19 ± 0.02	0.44 ± 0.02
Non-Adaptive (Baseline)	0.50 ± 0.05	0.53 ± 0.02
ASHA (Ours)	$\mathbf{0.83 \pm 0.02}$	$\mathbf{0.79 \pm 0.03}$
ASHA w/ Det. Input Enc. (**Q1**)	0.70 ± 0.03	0.73 ± 0.02
ASHA w/ Det. Pre-train Enc. (**Q2**)	0.66 ± 0.06	0.46 ± 0.03
SAC from Scratch (**Q3**)	0.00 ± 0.00	0.00 ± 0.00
ASHA w/o Failure Relabeling (**Q4**)	0.54 ± 0.03	0.55 ± 0.02
ASHA w/ Latent Regression (**Q5**)	0.41 ± 0.04	0.57 ± 0.02

Success rates across 100 episodes and 10 random seeds

2.4 Simulation Experiments

Ablation Study

To run ablation experiments at a scale that would be impractical in a user study, we simulate user input $\mathbf{x} \in \mathbb{R}^3$ as the 3D position of the target switch or bottle with i.i.d. isotropic Gaussian noise added at each timestep. We seek to answer the following questions. **Q1**: Does sampling from a stochastic input encoder f_θ^{inpt} 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 latent-conditioned policy g_ϕ 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 2.3 perform better than regressing onto sampled latents that led to a success? The results in Table 2.1 show that all the ablated variants of ASHA perform worse than the full ASHA method, suggesting that sampling from a stochastic input encoder f_θ^{inpt} improves exploration (**Q1**), pre-training with a VIB and reusing the pre-trained latent-conditioned policy g_ϕ 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 latents (**Q5**).

Demonstration on Continuous Task Spaces

The switch and bottle environments tested in the previous experiments have discrete task spaces: the user either wants to flip one of five switches, or reach one of two bottles. However,

ASHA can also effectively assist users who have continuous task spaces. To demonstrate this capability, we ran experiments with simulated users and three expert human users who rotate a valve to a desired target angle $\theta \sim \text{Unif}(0, 2\pi)$ – i.e., a continuous, 1D task space (see Figure 2.2c). In addition to the expert human input, we tested various types of simulated user inputs, including static inputs that noisily encode the target angle, dynamic inputs that noisily encode subgoals on a path to the goal state, and directional inputs that noisily indicate whether to rotate clockwise, counter-clockwise, or remain. The results in Figure 2.5a show that ASHA learns to perform the desired task with an 80% success rate when an expert human provides input (pink), a 90% success rate when we simulate user input that is static and simple to decode (orange), and performs substantially better than a random-latent baseline policy (red) when we simulate user input that encodes nearby subgoals (blue) or desired direction of rotation (gray).

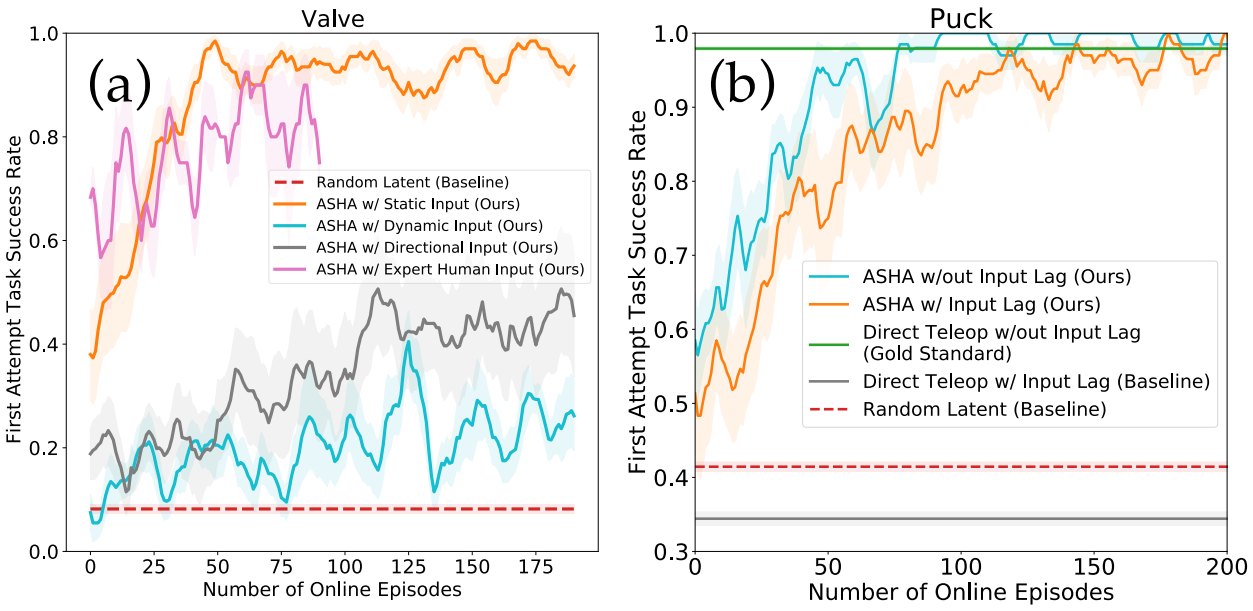
Demonstration on Structured User Inputs

This paper focuses on the problem of interpreting raw user inputs like webcam images as commands. However, ASHA can also be used to assist users who already have access to a direct teleoperation interface for translating raw user inputs into robot actions, but still require help to perform their desired tasks. To illustrate this capability, we ran experiments with simulated users who push a puck on a table to a desired target position, which is sampled uniformly at random from a continuous, 2D task space (see Figure 2.2d). We simulated user inputs by training an oracle policy to perform the pushing task, then adding lag to the oracle actions, which models real-world conditions like network latency – as a result, the user inputs to ASHA are (suboptimal) 7-dimensional joint torques. The results in Figure 2.5b show that, when the user's input is laggy, ASHA (orange) achieves substantially higher success rates than the direct teleop interface alone (gray), eventually reaching the performance of a hypothetical direct teleop interface that receives user input without lag (green).

2.5 Discussion

We presented a system that efficiently trains an adaptive interface through RL from sparse user feedback. Our user studies in three 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 the ability to sample pre-training tasks and accompanying reward functions (see Section 2.2). Future work could use a self-supervised RL algorithm to discover a latent skill space without a pre-determined distribution of tasks [26, 72, 58]. 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 [31, 8].

Figure 2.5: Continuous Task Spaces and Structured User Inputs



2.6 Acknowledgements

This work was supported by NVIDIA Graduate Fellowship, Berkeley Existential Risk Initiative, AFOSR FA9550-17-1-0308, Weill Neurohub, NSF CAREER, CIFAR, Office of Naval Research, Army Research Office, ARL DCIST CRA W911NF-17-2-0181, NSF IIS-1651843, and computing resources from NVIDIA. This work was originally published as the conference paper "ASHA: Assistive Teleoperation via Human-in-the-Loop Reinforcement Learning" at the 2022 International Conference on Robotics and Automation (ICRA) [14].

Chapter 3

Conclusions and Future Work

In this technical report, we presented two different approaches that enable assistive systems to adapt to human input over time in order to infer a user's intent, and assist them in performing their desired objectives. While our experimental studies demonstrated the effectiveness of these approaches compared to static interfaces that do not adapt to human input, the scope of these evaluations were limited to controlled, simulated settings that are not necessarily reflective of potential effectiveness in the real world. For deep learning systems to be effective in such settings, it is possible they will need to leverage large amounts of diverse data in order to generalize to the complexities of the real world [9]. This need is further exacerbated in human-interactive systems by the complexity and unpredictable nature of human behavior, and it would also be difficult to scale up online human interaction data to the levels needed for this.

The methods outlined in this paper make preliminary steps towards leveraging offline and autonomously collected data for human-interactive systems, which reduces the reliance on online interaction data for learning. However, there remain limitations that future work will need to address to obtain real-world effectiveness. In X2T, we first pre-train our model on data collected from a user operating a static default interface, which we show significantly improves both initial and overall performance. However, our simple supervised learning-based approach is limited to bandit settings, while more complex real-world tasks will require reasoning about sequential decision making. In ASHA, we pre-train our model autonomously to perform a variety of tasks in its environment, using deep RL and optionally human demonstrations. However, we make the assumption that we can specify all possible tasks the user would like to perform apriori, which limits the scalability of this approach. Furthermore, training multi-task deep RL algorithms in the real world remains a difficult open problem, due to challenges such as reward specification, perception, and safety.

One promising approach that can potentially better leverage offline data is offline RL [53]. In offline RL, we assume access to a static dataset of environment interaction data, and seek to train a policy that can perform tasks effectively using only this data and no online interaction. We would also like for such policies to be able to further improve once deployed by fine-tuning using online interaction data. This offline RL with fine-tuning framework

is particularly appealing in human-interactive settings, due to the cost and difficulty of collecting online human-in-the-loop data, and the potential abundance of offline human-interaction data with prior human-computer interfaces. Offline RL in general is still an unsolved problem, but with better offline RL algorithms and clever ways of leveraging them in human-interactive settings, we can hope to scale assistive systems, similar to those presented in this work, using large amounts of diverse data to be effective and robust in the real world.

Bibliography

[1] Alessandro Achille and Stefano Soatto. "Information dropout: Learning optimal representations through noisy computation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[2] Alexander A Alemi et al. "Deep variational information bottleneck". In: *arXiv preprint arXiv:1612.00410* (2016).

[3] Gopala K Anumanchipalli, Josh Chartier, and Edward F Chang. "Speech synthesis from neural decoding of spoken sentences". In: *Nature* (2019).

[4] Brenna D Argall. "Modular and adaptive wheelchair automation". In: *Experimental Robotics*. 2016.

[5] Reuben M Aronson et al. "Eye-hand behavior in human-robot shared manipulation". In: *IEEE Conference on Human-Robot Interaction*. 2018.

[6] Dilip Arumugam et al. "Deep reinforcement learning from policy-dependent human feedback". In: *arXiv preprint arXiv:1902.04257* (2019).

[7] Zeungnam Bien et al. "Integration of a rehabilitation robotic system (KARES II) with human-friendly man-machine interaction units". In: *Autonomous Robots* (2004).

[8] Florent Bocquelet et al. "Real-time control of an articulatory-based speech synthesizer for brain computer interfaces". In: *PLoS Computational Biology* (2016).

[9] Rishi Bommasani et al. "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258* (2021).

[10] Alexander Broad, Todd David Murphey, and Brenna Dee Argall. "Learning models for shared control of human-machine systems with unknown dynamics". In: *Robotics: Science and Systems*. 2017.

[11] Tom Carlson and Yiannis Demiris. "Collaborative control for a robotic wheelchair: evaluation of performance, attention, and workload". In: *IEEE Transactions on Systems, Man, and Cybernetics* (2012).

[12] Jose M Carmena. "Advances in neuroprosthetic learning and control". In: *PLoS Biology* (2013).

[13] Ciprian Chelba et al. "One billion word benchmark for measuring progress in statistical language modeling". In: *arXiv preprint arXiv:1312.3005* (2013).

[14] Sean Chen et al. "ASHA: Assistive Teleoperation via Human-in-the-Loop Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation* (2022).

[15] Paul F Christiano et al. "Deep reinforcement learning from human preferences". In: *Neural Information Processing Systems*. 2017.

[16] Gregory Cohen et al. "EMNIST: Extending MNIST to handwritten letters". In: *International Joint Conference on Neural Networks*. 2017.

[17] Mark Collier and Hector Urdiales Llorens. "Deep contextual multi-armed bandits". In: *arXiv preprint arXiv:1807.09809* (2018).

[18] Erwin Coumans and Yunfei Bai. "PyBullet, a Python module for physics simulation for games, robotics and machine learning". In: (2016).

[19] Yuchen Cui et al. "The EMPATHIC Framework for Task Learning from Implicit Human Feedback". In: *arXiv preprint arXiv:2009.13649* (2020).

[20] John P Cunningham et al. "A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces". In: *Journal of Neurophysiology* (2011).

[21] Zihang Dai et al. "Transformer-xl: Attentive language models beyond a fixed-length context". In: *arXiv preprint arXiv:1901.02860* (2019).

[22] Siddharth Dangi et al. "Continuous closed-loop decoder adaptation with a recursive maximum likelihood algorithm allows for rapid performance acquisition in brain-machine interfaces". In: *Neural computation* (2014).

[23] Siddharth Dangi et al. "Design and analysis of closed-loop decoder adaptation algorithms for brain-machine interfaces". In: *Neural computation* (2013).

[24] Yuqing Du et al. "AvE: Assistance via Empowerment". In: *arXiv preprint arXiv:2006.14796* (2020).

[25] Zackory Erickson et al. "Assistive gym: A physics simulation framework for assistive robotics". In: *IEEE International Conference on Robotics and Automation*. 2020.

[26] Benjamin Eysenbach et al. "Diversity is all you need: Learning skills without a reward function". In: *arXiv preprint arXiv:1802.06070* (2018).

[27] David Gaddy and Dan Klein. "Digital Voicing of Silent Speech". In: *arXiv preprint arXiv:2010.02960* (2020).

[28] Jensen Gao et al. "X2T: Training an X-to-Text Typing Interface with Online Learning from User Feedback". In: *International Conference on Learning Representations*. 2021.

[29] Vikash Gilja et al. "A high-performance neural prosthesis enabled by control algorithm design". In: *Nature neuroscience* (2012).

[30] Ethan K Gordon et al. "Adaptive Robot-Assisted Feeding: An Online Learning Framework for Acquiring Previously-Unseen Food Items". In: *arXiv preprint arXiv:1908.07088* (2019).

[31] Frank H Guenther et al. "A wireless brain-machine interface for real-time speech synthesis". In: *PloS one* (2009).

[32] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International Conference on Machine Learning*. 2018.

[33] Kris Hauser. "Recognition, prediction, and planning for assisted teleoperation of freeform tasks". In: *Autonomous Robots* (2013).

[34] Karol Hausman et al. "Learning an embedding space for transferable robot skills". In: *International Conference on Learning Representations*. 2018.

[35] Robert D Hawkins, Michael C Frank, and Noah D Goodman. "Characterizing the dynamics of learning in repeated reference games". In: *Cognitive Science* (2020).

[36] Natasha Jaques et al. "Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control". In: *International Conference on Machine Learning*. 2017.

[37] Shervin Javdani. "Acting under Uncertainty for Information Gathering and Shared Autonomy". PhD thesis. Carnegie Mellon University, 2017.

[38] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. "Shared autonomy via hindsight optimization". In: *arXiv preprint arXiv:1503.07619* (2015).

[39] Hong Jun Jeon, Dylan P Losey, and Dorsa Sadigh. "Shared Autonomy with Learned Latent Actions". In: *arXiv preprint arXiv:2005.03210* (2020).

[40] Eleanor Jones et al. "GesText: accelerometer-based gestural text-entry systems". In: *SIGCHI Conference on Human Factors in Computing Systems*. 2010.

[41] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* (1998).

[42] Siddharth Karamcheti, Dorsa Sadigh, and Percy Liang. "Learning Adaptive Language Interfaces through Decomposition". In: *arXiv preprint arXiv:2010.05190* (2020).

[43] Hyun K Kim et al. "Continuous shared control for stabilizing reaching and grasping with brain-machine interfaces". In: *IEEE Transactions on Biomedical Engineering* (2006).

[44] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[45] W Bradley Knox and Peter Stone. "Interactively shaping agents via human reinforcement: The TAMER framework". In: *International Conference on Knowledge Capture*. 2009.

[46] Hema S Koppula and Ashutosh Saxena. "Anticipating human activities using object affordances for reactive robotic response". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2016).

[47] Kyle Krafka et al. "Eye tracking for everyone". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[48] Andrew S Lan and Richard G Baraniuk. "A Contextual Bandits Framework for Personalized Learning Action Selection." In: *Educational Data Mining*. 2016.

[49] John Langford and Tong Zhang. "The epoch-greedy algorithm for multi-armed bandits with side information". In: *Neural Information Processing Systems*. 2008.

[50] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[51] Eric C Leuthardt et al. "A brain–computer interface using electrocorticographic signals in humans". In: *Journal of Neural Engineering* (2004).

[52] Sergey Levine. "Reinforcement learning and control as probabilistic inference: Tutorial and review". In: *arXiv preprint arXiv:1805.00909* (2018).

[53] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[54] Andrew Levy, Robert Platt, and Kate Saenko. "Hierarchical actor-critic". In: *arXiv preprint arXiv:1712.00948* (2017).

[55] Alexander Li et al. "Sub-policy Adaptation for Hierarchical Reinforcement Learning". In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=ByeWogStDS`.

[56] Lihong Li et al. "A contextual-bandit approach to personalized news article recommendation". In: *International Conference on World Wide Web*. 2010.

[57] David Llorens et al. "The UJIpenchars Database: a Pen-Based Database of Isolated Handwritten Characters." In: *International Conference on Language Resources and Evaluation*. 2008.

[58] Corey Lynch et al. "Learning latent plans from play". In: *Conference on Robot Learning*. 2020.

[59] James MacGlashan et al. "Interactive learning from policy-dependent human feedback". In: *arXiv preprint arXiv:1701.06049* (2017).

[60] I Scott MacKenzie and Kumiko Tanaka-Ishii. *Text entry systems: Mobility, accessibility, universality*. 2010.

[61] Joseph G Makin, David A Moses, and Edward F Chang. *Machine translation of cortical activity to text with an encoder–decoder framework*. Tech. rep. 2020.

[62] Marcus Mast et al. "User-centered design of a dynamic-autonomy remote interaction concept for manipulation-capable robots to assist elderly people in the home". In: *Journal of Human-Robot Interaction* (2012).

[63] Daniel McDuff and Ashish Kapoor. "Visceral Machines: Reinforcement Learning with Intrinsic Physiological Rewards". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SyNvti09KQ.

[64] Amy McGovern et al. "Hierarchical optimal control of MDPs". In: 1998.

[65] David P McMullen 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 prosthetic". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* (2013).

[66] Josh Merel et al. "Neuroprosthetic decoder training as imitation learning". In: *arXiv preprint arXiv:1511.04156* (2015).

[67] Katharina Muelling et al. "Autonomy infused teleoperation with application to BCI manipulation". In: *arXiv preprint arXiv:1503.05451* (2015).

[68] Katharina Muelling et al. "Autonomy infused teleoperation with application to brain computer interface controlled manipulation". In: *Autonomous Robots* (2017).

[69] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Neural Information Processing Systems*. 2018.

[70] Ofir Nachum et al. "Near-optimal representation learning for hierarchical reinforcement learning". In: *arXiv preprint arXiv:1810.01257* (2018).

[71] Ashvin Nair et al. "Accelerating online reinforcement learning with offline datasets". In: *arXiv preprint arXiv:2006.09359* (2020).

[72] Ashvin Nair et al. "Contextual imagined goals for self-supervised robotic learning". In: *Conference on Robot Learning*. 2020.

[73] Stefanos Nikolaidis et al. "Human-robot mutual adaptation in shared autonomy". In: *ACM/IEEE International Conference on Human-Robot Interaction*. 2017.

[74] João Oliveira et al. "BrailleType: unleashing braille over touch screen mobile phones". In: *IFIP Conference on Human-Computer Interaction*. 2011.

[75] Ronald Parr and Stuart J Russell. "Reinforcement learning with hierarchies of machines". In: *Neural Information Processing Systems*. 1998.

[76] Claudia Pérez-D'Arpino and Julie A Shah. "Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification". In: *IEEE International Conference on Robotics and Automation*. 2015.

[77] Laura Petrich et al. "Assistive arm and hand manipulation: How does current research intersect with actual healthcare needs?" In: *arXiv preprint arXiv:2101.02750* (2021).

[78] Patrick M Pilarski et al. "Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning". In: *IEEE International Conference on Rehabilitation Robotics*. 2011.

[79] Krsto Proroković, Michael Wand, and Jürgen Schmidhuber. "Meta-Learning for Recalibration of EMG-Based Upper Limb Prostheses". In: (2020).

[80] Filip Radlinski and Thorsten Joachims. "Evaluating the robustness of learning from implicit feedback". In: *arXiv preprint cs/0605036* (2006).

[81] Kate Rakelly et al. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *International Conference on Machine Learning*. 2019.

[82] Céline Ray, Francesco Mondada, and Roland Siegwart. "What do people expect from robots?" In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008.

[83] Siddharth Reddy, Anca D Dragan, and Sergey Levine. "Shared autonomy via deep reinforcement learning". In: *arXiv preprint arXiv:1802.01744* (2018).

[84] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *International Conference on Artificial Intelligence and Statistics*. 2011.

[85] Dorsa Sadigh et al. "Active preference-based learning of reward functions". In: *Robotics: Science and Systems*. 2017.

[86] Charles Schaff and Matthew R Walter. "Residual Policy Learning for Shared Autonomy". In: *arXiv preprint arXiv:2004.05097* (2020).

[87] Jonathon W Sensinger, Blair A Lock, and Todd A Kuiken. "Adaptive pattern recognition of myoelectric signals: exploration of conceptual framework and practical algorithms". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* (2009).

[88] Krishna V Shenoy and Jose M Carmena. "Combining decoder design and neural adaptation in brain-machine interfaces". In: *Neuron* (2014).

[89] Daniel B Silversmith et al. "Plug-and-play control of a brain–computer interface through neural map stabilization". In: *Nature Biotechnology* (2020).

[90] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[91] Dawn M Taylor, Stephen I Helms Tillery, and Andrew B Schwartz. "Direct cortical control of 3D neuroprosthetic devices". In: *Science* (2002).

[92] Cheng-Yao Wang et al. "PalmType: Using palms as keyboards for smart glasses". In: *International Conference on Human-Computer Interaction with Mobile Devices and Services*. 2015.

[93] Sida I Wang, Percy Liang, and Christopher D Manning. "Learning language games through interaction". In: *arXiv preprint arXiv:1606.02447* (2016).

[94] David J Ward, Alan F Blackwell, and David JC MacKay. "Dasher—a data entry interface using continuous gestures and language models". In: *ACM Symposium on User Interface Software and Technology*. 2000.

[95] Garrett Warnell et al. "Deep tamer: Interactive agent shaping in high-dimensional state spaces". In: *arXiv preprint arXiv:1709.10163* (2017).

[96] Francis R Willett et al. "High-performance brain-to-text communication via imagined handwriting". In: *bioRxiv* (2020).

[97] Francis R Willett et al. "Principled BCI decoder design and parameter selection using a feedback control model". In: *Scientific Reports* (2019).

[98] Alan Wrench. *The MOCHA-TIMIT Articulatory Database*. 1999.

[99] Duo Xu et al. "Accelerating Reinforcement Learning Agent with EEG-based Implicit Human Feedback". In: *arXiv preprint arXiv:2006.16498* (2020).

[100] Yisong Yue and Thorsten Joachims. "Interactively optimizing information retrieval systems as a dueling bandits problem". In: *International Conference on Machine Learning*. 2009.

[101] Dongruo Zhou, Lihong Li, and Quanquan Gu. "Neural Contextual Bandits with Upper Confidence Bound-Based Exploration". In: *arXiv preprint arXiv:1911.04462* (2019).

[102] Katie Z Zhuang et al. "Shared human–robot proportional control of a dexterous myoelectric prosthesis". In: *Nature Machine Intelligence* (2019).

Chapter 4

Appendices

4.1 X2T: Training an X-to-Text Typing Interface with Online Learning from User Feedback

Implementation Details

Stochastic gradient descent. We use Adam [44] to optimize the binary cross-entropy loss described in Section 1.2,

$$\ell(\theta) = -\sum_{(\mathbf{x},\mathbf{u},r)\in\mathcal{D}} r \log\left(p_\theta(r=1|\mathbf{x},\mathbf{u})\right) + (1-r)\log\left(1-p_\theta(r=1|\mathbf{x},\mathbf{u})\right). \tag{4.1}$$

We set gradient norm clipping to 10 in all experiments.
Offline pretraining. To pretrain X2T on offline data, we first train the reward model p_θ to convergence on the offline data, and store the learned network weights θ_{init}. Once a minimum of four online input-action-reward triples have been collected, after every interaction, we update the reward model p_θ by taking one gradient step. We use the values θ_{init} to initialize the network weights (e.g., instead of a random initialization).

Online User Study: Typing with Eye Gaze

Eye image features. Instead of operating directly on 224x224 images of the user's eyes, we use the activations of the last fully-connected layer in the iTracker model [47] as the input $\mathbf{x}$ to our reward model $p_\theta(r|\mathbf{x},\mathbf{u})$. When training the reward model p_θ, we freeze the iTracker network weights used to generate this 128-dimensional input $\mathbf{x}$.
Reward model architecture and learning. We set the learning rate for Adam to 10^{-3}, and batch size to 128. We do not perform online updates to the reward model parameters θ while $|\mathcal{D}|< 4$. In our experiments, for every input $\mathbf{x}$, there is exactly one desired action $\mathbf{u}^*$ that will result in a positive reward $r = 1$, while all other actions $\mathbf{u} \neq \mathbf{u}^*$ result in a zero reward $r = 0$. In other words, $p(r = 1|\mathbf{x},\mathbf{u}) = p(\mathbf{u} = \mathbf{u}^*|\mathbf{x})$. Hence, we structure the reward

model for X2T as $p_\theta(r = 1|\mathbf{x}, \mathbf{u}) = f_\theta(\mathbf{u}|\mathbf{x})$, where f_θ is an action classifier. Furthermore, since we expect the reward model to learn to implicitly estimate the user's gaze position in order to predict actions, we directly incorporate this inductive bias into the model: we structure the action classifier as $f_\theta(\mathbf{u}|\mathbf{x}) \propto \exp\left(-\|g_\theta(\mathbf{x}) - \text{pos}(\mathbf{u})\|_2\right)$, where $g_\theta(\mathbf{x})$ outputs a 2D position of the user's estimated gaze, and $\text{pos}(\mathbf{u})$ is the known 2D position of the button for action $\mathbf{u}$. Note that even though $g_\theta(\mathbf{x})$ outputs a 2D position, the parameters θ are still trained on the reward prediction objective in Equation 4.1 (e.g., instead of a 2D gaze position prediction objective). We represent g_θ using a feedforward neural network with one hidden layer containing 64 units, a dropout layer with a dropout rate of 0.3 between the hidden layer and the output layer, and ReLU activations. At each timestep t, we record 10 eye images $\{\mathbf{x}_t^i\}_{i=1}^{10}$ at a sampling rate of 10 Hz, and average our predictions over these 10 inputs. Specifically, for X2T, we set $g_\theta(\mathbf{x}_t) = \frac{1}{10}\sum_{i=1}^{10} g_\theta(\mathbf{x}_t^i)$. For the default interface, we average the 2D gaze position estimates across the 10 samples before predicting the action whose button position is nearest to the average gaze position estimate. We initialize θ_{init} with the offline pretraining scheme described in Section 1.2, using 250 input-action-reward triples collected with the default interface.

Experiment design. We recruited 11 male and 1 female participants, with an average age of 21. Each participant was provided with the rules of the task and a short practice period of 20 interactions to familiarize themselves with the system. Each interaction – which consisted of providing an input, observing the interface's action, and deciding whether or not to backspace – took an average of 4 seconds. Each participant completed three phases of experiments: A, B, and C. In phase A, they operate the default interface for 250 steps, generating an offline dataset of input-action-reward triples that we use to initialize X2T. In phase B, they operate X2T for 250 steps. In phase C, they operate the default interface for 250 steps. To avoid the confounding effects of user learning or fatigue over time, we counterbalance the order of phase B and C: six randomly-selected participants completed phase B before C, and the other six participants completed phase C before B. Phase A is used solely to generate offline data to initialize X2T in phase B. One participant's room lighting changed substantially during phase B. Since their performance during phase A was substantially better than during phase B, we use their phase A data (instead of phase B data) to measure the default interface's performance on this one participant. We sample goal sentences from the MOCHA-TIMIT database [98], following prior work on speech interfaces [61]. We set the same goal sentences for each user in each condition.

Deterministic policy. In our experiments, we find that sampling actions $\mathbf{u} \sim \pi(\mathbf{u}|\mathbf{x})$ from the stochastic policy π does not substantially improve exploration, and can in fact degrade performance by not always choosing the optimal action. This is most likely due to the use of the default interface $\bar{\pi}$ in Equation 1.1, which already provides an effective exploration mechanism. Hence, instead of randomly sampling actions, we deterministically select the highest-likelihood action: $\mathbf{u} \leftarrow \arg\max_{\mathbf{u}} \pi(\mathbf{u}|\mathbf{x})$.

Observational Study: Typing by Drawing Characters

Perturbing user inputs. We perturb the character drawings in the UJI Pen Characters Database by decomposing each drawing into a sequence of pen tip velocity vectors, adding Brownian noise to each velocity, then integrating over the perturbed velocities to yield a complete, perturbed drawing (see Figure 4.2 in the appendix). We compute the Brownian noise by sampling an independent Gaussian noise vector with zero mean and variance of $2 \cdot 10^{-4}$ at each timestep, and summing over these noise vectors from time 0 to time t to compute the Brownian noise vector for time t. The same Brownian noise vector is applied to all the velocity segments in a given drawing. Each user input $\mathbf{x}$ is a 28x28 image of the user's complete, perturbed drawing of a given character. There are 27 possible characters in the action space $\mathcal{U}$: 26 lower-case letters, and space (which we represent using the digit 7).
Reward model architecture and learning. We set the learning rate for Adam to $5 \cdot 10^{-4}$, batch size to 128, pretrain on the offline data for 20 epochs, and sample actions from the stochastic policy described in Equation 1.1 (instead of the deterministic policy described in Appendix 4.1). We also limit the size of the replay buffer $\mathcal{D}$ in Algorithm 1 to the latest 500 input-action-reward triples. We do not perform online updates to the reward model parameters θ while $|\mathcal{D}|< 100$. We represent the reward model p_θ as a neural network with the following architecture: 28x28 input layer, 32x5x5 convolutional layer, 2x2 max pool layer, dropout layer with dropout rate of 0.5, 64x5x5 convolutional layer, 2x2 max pool layer, dropout layer with dropout rate of 0.3, and a fully-connected output layer. We structure the reward model as an action classifier, as in Section 4.1. We initialize θ_{init} with the offline pretraining scheme described in Section 1.2, using 1000 input-action-reward triples collected with the default interface.
Experiment design. The UJI Pen Characters Database [57] contains handwriting samples from 60 users. For each user, it includes two repetitions of lowercase letters, uppercase letters, and 10 digits, for a total of 1364 samples per user. In our observational study, we randomly sample target sentences, and replay user inputs that attempt to type the characters in those target sentences. In the replays, we automatically backspace incorrect actions; a realistic modeling choice, given that in the online user study in Section 1.4, the user did indeed backspace mistakes, and did not backspace correct actions, in 98.6% of their interactions. As in the online user study, we run each user's data through two conditions: default and X2T. Since this is an observational study, we do not need to counterbalance the order of the two conditions. For the personalization experiment in Table 1.1, we assign a randomly-selected, constant value to the random seed of the Brownian noise for each user in each condition. This ensures that comparisons between entry (i, i) and entries $(i, j \neq i)$ in the 4x4 table are not confounded by differences in the input noise, and are only influenced by systematic differences in the users' individual handwriting styles. As in the online user study, we sample goal sentences from the MOCHA-TIMIT database [98], and set the same goal sentences for each user in each condition.
Language model. We use the Transformer-XL language model [21] – specifically, the word-level version that is pretrained on the One Billion Word dataset [13] – to compute

the prior likelihood $p_{\text{LM}}(\mathbf{u}_t|\mathbf{u}_{0:t-1})$. To compute character-level likelihoods, we marginalize over the 60000 words in the language model's vocabulary that occur the most frequently in the One Billion Word training corpus, not including words with punctuation and converting upper-case to lower-case characters. We feed the previously-typed characters in the current sentence (i.e., not including the previous sentences) as context to the language model.

Pilot Study with a Brain-Computer Interface

Due to constraints on the duration and number of sessions we were able to conduct with the participant, we made two simplifications to the interface: we simplified the display to use 4 buttons instead of 8 buttons, and we automated backspaces.

Reward model architecture and learning. We set the learning rate for Adam to 10^{-4}, batch size to 128, and maximum buffer size $|\mathcal{D}|$ to 1000. Unlike the previous experiments, we did not use dropout. We used an L2 regularization constant of 0.001 for X2T, and 0.01 for the default interface. For both the default interface and reward model, we used a feedforward network architecture with 3 layers of 256 hidden units each. We deterministically selected actions with the maximum conditional likelihood under the policy, instead of sampling from the stochastic policy in Equation 1.1. When pretraining the reward model on offline data, we initialized the reward model network weights with the default interface network weights. We set the prior policy $\bar{\pi}$ in Equation 1.1 to be a uniform prior, instead of using the default interface. To accommodate noise in the user inputs, we require 4 consecutive bins of input to be mapped to the same action by the policy before executing that action; after 10 bins with no string of 4 consecutive, equal actions, the majority action is executed. In the counterfactual experiment with the default interface, if we reach the end of the bins for a given step without a string of 4 consecutive, equal actions, the majority action is executed. After an action is executed and reward collected, we add an input-action-reward tuple for the input at each bin to the buffer $\mathcal{D}$.

Experiment design. We conducted three experimental sessions: the first on 2/26/21, in which we collected 997 input-action-reward samples; the second on 3/5/21, which yielded 1916 samples; and the third on 3/13/21, which yielded 1984 samples. The default interface was trained on 3686 samples (left button intended: 959, down: 916, right: 897, up: 914) recorded prior to 2/26/21. There were 3 seconds of no input before each word selection. The interface was paused intermittently during each session to accommodate user fatigue.

Feature processing. The ECoG signals were binned at a frequency of 2 Hz. Each user input has 128 dimensions, which consist of high-band frequencies of raw ECoG signal recorded during each binning period. The remaining implementation details are described in [89].

	p-value	Default Interface	X2T
The system selected the words I wanted	$<$ **.05**	4.50	**5.42**
The system improved over time	$<$ **.05**	3.17	**4.75**
I improved at using the system over time	$>$ .05	4.08	4.42
The system did not select the words I wanted	$<$ **.05**	4.08	**2.83**
The system got worse over time	$>$ .05	3.25	2.42
I got worse at using the system over time	$>$ .05	3.25	2.67
I backspaced when there was a mistake	$>$ .05	5.42	5.83
I ignored mistakes (did not backspace them)	$>$ .05	2.33	2.17

Table 4.1: Subjective evaluations from the 12 participants in the online user study. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of X2T as a factor influencing responses.

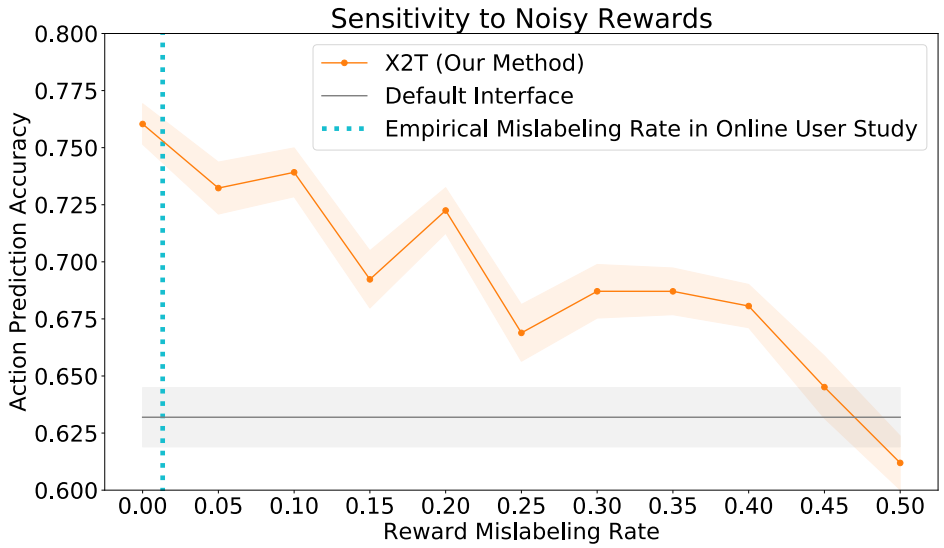


Figure 4.1: To measure X2T's sensitivity to noise in the reward signal, we conduct a counterfactual experiment with the eye gaze user study data, similar to Section 1.4. In this experiment, we flip the reward r to $1 - r$ with probability p, which we call the reward mislabeling rate. We then train our reward model on these noisy rewards. X2T outperforms the default interface for a wide range of mislabeling rates, and only performs worse than the default interface when the rewards are completely random (i.e., the mislabeling rate is 50%). In practice, we find that users' backspaces tend to follow the assumptions in Section 1.2, which leads to a relatively low empirical mislabeling rate of 1.4%.

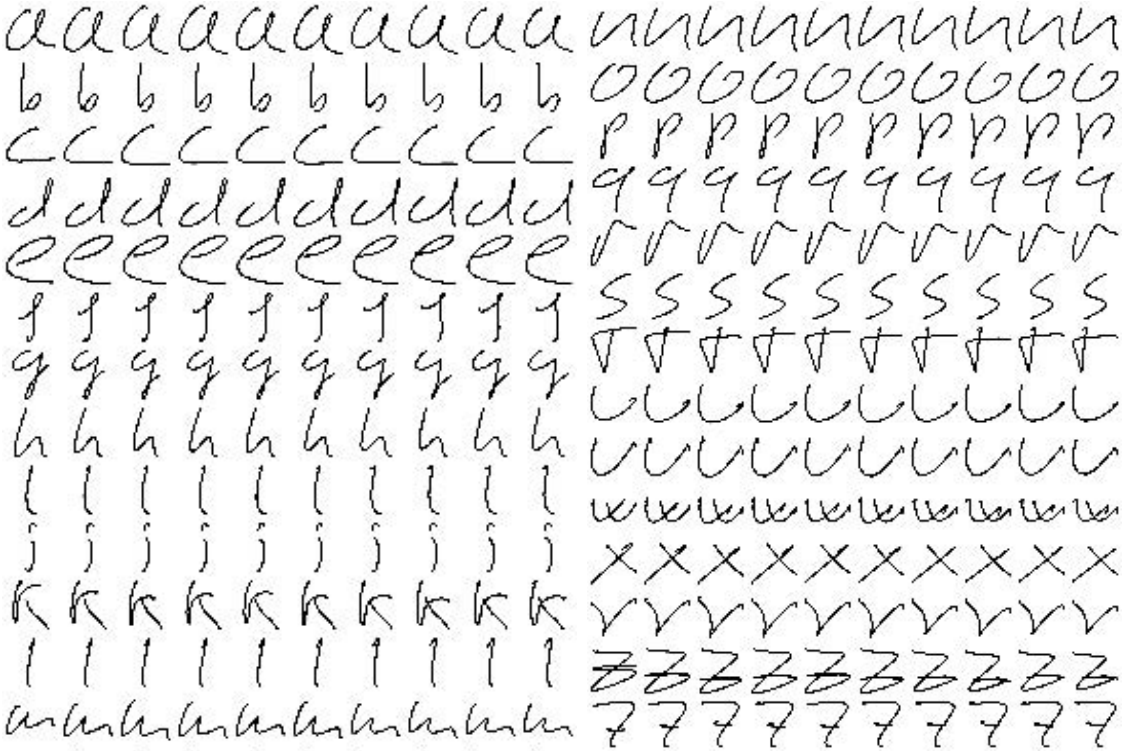


Figure 4.2: Handwriting samples from the UJI Pen Characters Database that have been perturbed by adding Brownian noise to pen tip velocities. The left-most column shows the true input, and each successive column shows the perturbed input after 100-timestep intervals. We used the character '7' in place of the space character.

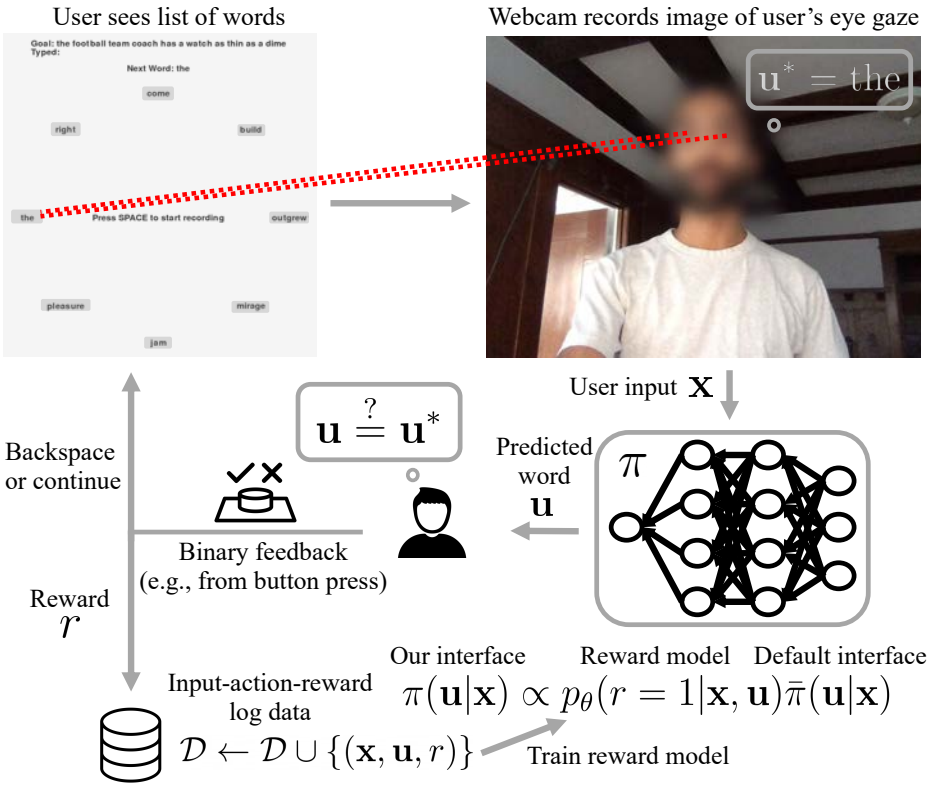


Figure 4.3: An illustration of the eye gaze experiments in Section 1.4

4.2 ASHA: Assistive Teleoperation via Human-in-the-Loop Reinforcement Learning

Additional Related Work

ASHA's use of a pre-training phase to learn a low-level policy g_ϕ that accelerates downstream learning of a high-level user interface f_θ^{inpt} resembles hierarchical RL methods [64, 75], which generally aim to improve exploration and credit assignment by dividing the original Markov decision process (MDP) into simpler high- and low-level MDPs. Recent work in this area focuses on solving the divided MDPs concurrently [54, 69, 55] and combining the results [70]. Our problem setting and assumptions are markedly different, in that the engineered reward function used to pre-train the low-level policy in phase 1 of ASHA may differ substantially from the user-provided reward that is used to train the interface online in phase 2, while standard hierarchical methods assume that these two reward functions are equivalent.

Prior work on myoelectric interfaces for prosthetic limb control explores unsupervised learning [87] and RL [78] methods for online adaptation to the user's EMG signals, but only evaluates on simple motion-tracking tasks with small, discrete state spaces, whereas our work focuses on more complex manipulation tasks with large, continuous state spaces (see Figure 2.2 and Appendix 4.2).

ASHA's approach to relabeling trajectories with an optimal policy (see line 16 in Algorithm 2) is analogous to the DAgger imitation learning algorithm, which relabels on-policy trajectories of an imitation policy with expert action labels [84].

Implementation Details

Timeouts

To ensure that the user studies do not get stalled on a single task, if the user does not succeed at a given task after 5 episodes of attempts, we 'timeout' and ask them to move on to a new task. We do not learn from these 5 episodes in our method, since we do not have a success that we can use to compute the optimal policy in hindsight (see Section 2.2). Across all methods tested and all experimental conditions in the user studies, timeouts occurred in 6% of tasks in the bottle domain, and 20% of tasks in the switch domain.

Simulator Setup for Switch and Bottle Domains

We setup our switch and bottle domains using assets from Assistive Gym [25]. The sizes of the simulated Jaco arm and wheelchair are proportional to those of their real-world counterparts. In the switch domain, the switches are placed 0.22 units of distance apart horizontally. The horizontal position of the switches vary uniformly at random within an interval of 0.3 units, but all switches share the same positional noise (i.e., all switches move together). The distance between the user and the wall varies uniformly at random within an interval of

0.2 units during the pre-training and calibration phases – during online evaluation, it only varies within half that range. The initial arm end-effector position varies within a uniform box of size (1, 0.2, 0.2). In the bottle domain, the bottles are placed 0.3 units of distance apart horizontally. The horizontal position of the table varies uniformly at random within an interval of 0.4 units. The bottles always maintain the same relative position with respect to the table. The initial arm end-effector position varies uniformly at random within a box of size (0.8, 0.2, 0.2). Due to the noise added to the positions of switches and bottles at the beginning of each episode, the positions for different switches and bottles can overlap across episodes (e.g., switch 1 may be located very close to the previous position of switch 2 in a past episode). While the arm is reset to its initial position at the start of each episode, regardless of the success or failure of the previous episode, the other conditions of the environment (e.g., the positions of the switches, table, or wall) are only reset after a successful episode or a timeout. The arm is reset by first sampling a 3D position, then solving for the joint positions of the arm using an inverse kinematics function.

The environments use a frame skip of 5 steps, and a maximum episode length of 200 steps (approximately 13 seconds of wall-clock time).

Representation of States, Actions, Task Specifications, and Latent Embeddings

In all domains, the low-level action $\mathbf{a} \in \mathbb{R}^7$ consists of 7 joint forces for the Jaco arm. Each action dimension is clipped to (-0.25, 0.25) before being executed in the simulator. In the switch domain, the state $\mathbf{s} \in \mathbb{R}^{48}$ consists of the 7 joint positions of the arm, the 3D position and 4D orientation of the end effector, and the 3D position and 1D angle of each of the 5 switches. In the bottle domain, the state $\mathbf{s} \in \mathbb{R}^{37}$ consists of the 7 joint positions of the arm, the 3D position and 4D orientation of the end effector, the 3D position of each of the 2 bottles, and the 3D position of door handle. We set each specification $\tau^{\text{spec}} \in \mathbb{R}^3$ to be the 3D position of the target switch or bottle. In both domains, we set the dimensionality of the latent embedding space to $d = 3$.

Recording the User's Eye Gaze

With a standard webcam, we record a 224x224x3 segmented image of the user's face, 224x224x3 image of each of the user's eyes, and a 25x25 binary grid that characterizes the overall position of the face in the webcam image, then feed these as input to a pre-trained iTracker model [47] (see Figure 2.2c). We treat the 128-dimensional activations of the last linear layer in the pre-trained iTracker neural network as the user's control input $\mathbf{x} \in \mathbb{R}^{128}$. Gaze is recorded by having an asynchronous thread that takes the webcam image, segments it, feeds it into the iTracker network, then extracts the features, sets it equal to the most recent features, and terminates. On every step of the environment simulator, we restart the gaze update thread if it has terminated, then pull the most recent gaze features and treat them as the user input $\mathbf{x}_t$ for that timestep.

Network Architecture and Optimization

In all phases (i.e., pre-training, calibration, and online learning), we use the Adam optimizer [44] with a batch size of 256 to train our models. We use the same network architecture to represent all encoders f_{ψ}^{spec}, $f_{\theta_0}^{\text{inpt}}$, and f_{θ}^{inpt}: a feedforward network with ReLU activations and a single hidden layer of 64 units.

In all phases, we set the regularization constant β for the VIB (e.g., see Equation 2.3) to 0.01.

In the pre-training phase, we run SAC with default hyperparameters: a 2-layer, 256-unit feedforward network to represent both the policy and Q-function, a learning rate of $3 \cdot 10^{-4}$, a reward scale of 1, automatic entropy tuning with a heuristic of setting the target entropy to the negative dimensionality of the action space, a target Q-function update period of 1, Polyack update τ set to $5 \cdot 10^{-3}$, epochs of 1000 environment steps followed by 1000 training steps, 1000 steps before training starts, and a replay buffer size of $5 \cdot 10^5$ in the switch domain and $2 \cdot 10^7$ in the bottle domain. We pre-train for 1000 epochs in the switch domain, and for 3150 epochs in the bottle domain. In the bottle domain, we initialize the pre-training replay buffer with 5000 demonstrations obtained from a scripted agent. When we execute the pre-training policy $\pi_{\psi,\phi}^{\text{spec}}$, we feed the expected value of the specification encoder output $\mathbf{z}$ to the latent-conditioned policy g_{ϕ}, instead of sampling from the posterior of the encoder.

During the online learning phase, we set the learning rate to $5 \cdot 10^{-4}$, perform 1000 gradient updates on the calibration data (which is typically sufficient for convergence), keep the calibration data in the replay buffer during online learning, perform 100 gradient updates after each successful episode (which is typically sufficient for convergence), and do not limit the size of the replay buffer (i.e., never discard old data).

We use the same random seeds for each user, and use a different random seed for each method and experimental condition within a given user. We use 10 different random seeds for each ablation experiment, and use the same 10 seeds across the ablations.

We set the optimal policy $\pi_{\psi,\phi}^{\text{spec}}$ used in Equation 2.3 during the online learning phase to be deterministic. Implicitly assuming that the interface $\pi_{\theta,\phi}^{\text{inpt}}$ has some fixed, diagonal covariance $I\sigma^2$, this enables us to simplify the KL divergence loss between the two policies $\pi_{\psi,\phi}^{\text{spec}}$ and $\pi_{\theta,\phi}^{\text{inpt}}$ in Equation 2.3 to the mean-squared error loss between the mean actions outputted by both policies.

We do not use a recurrent input encoder $f_{\theta}^{\text{inpt}}(\mathbf{z}|\mathbf{s}_{0:t}, \mathbf{x}_{0:t})$, and instead use a feedforward encoder $f_{\theta}^{\text{inpt}}(\mathbf{z}|\mathbf{s}_t, \mathbf{x}_t)$ that operates on only the most recent state $\mathbf{s}_t$ and user input $\mathbf{x}_t$.

To simplify the optimization of Equation 2.3, we do not integrate over the posterior distribution of latents $\mathbf{z}$ when computing the optimal policy $\pi_{\psi,\phi}^{\text{spec}}$ and interface $\pi_{\theta,\phi}^{\text{inpt}}$, and instead only feed the expected value of $\mathbf{z}$ to the latent-conditioned policy g_{ϕ} – i.e., we represent the optimal policy as $g_{\phi}(\mathbf{a}_t|\mathbf{s}_t, \mathbb{E}_{\mathbf{z}\sim f_{\psi}^{\text{spec}}(\mathbf{z}|\tau^{\text{spec}})}[\mathbf{z}])$ (instead of Equation 2.1), and the interface as $g_{\phi}(\mathbf{a}_t|\mathbf{s}_t, \mathbb{E}_{\mathbf{z}\sim f_{\theta}^{\text{inpt}}(\mathbf{z}|\mathbf{s}_{0:t},\mathbf{x}_{0:t})}[\mathbf{z}])$ (instead of Equation 2.2).

Pre-Training Tasks

In the switch domain, we have 5 pre-training tasks (one for each switch on the wall in Figure 2.2). In the bottle domain, we have 2 pre-training tasks (one for each bottle inside the shelf in Figure 2.2). The pre-training reward in the switch domain is 0 upon success, and $\exp(-\|\text{end effector pos.} - \text{target switch pos.}\| - 0.2) - 1$ otherwise. The pre-training reward in the bottle domain is $-1 + 0.5 \cdot \mathbb{1}[\text{door opened}] + 0.5 \cdot \mathbb{1}[\text{bottle reached}]$.

Episode Termination and User Feedback

During the pre-training phase, episodes only end if the task is successfully completed or the environment times out – we set the terminal flag to true only when the task is successfully completed. During the online learning phase, episodes also end if the wrong task is performed (e.g., the wrong switch is flipped).

If the episode ends in a success or in completing the wrong task, the user's binary feedback (provided through a button press) is treated as the reward signal. However, when the episode ends due to a timeout, we automatically generate a negative feedback signal. The user's button presses matched the automated feedback in 98% of episodes in the user study.

Calibration Videos

To generate the 6 videos in the switch domain and 8 videos in the bottle domain that are used to calibrate the non-adaptive baseline interface and our method (see Section 2.3), we execute the pre-trained robot policy $\pi^{\text{spec}}_{\psi,\phi}$. In the switch domain, we generate successful videos for 3 different switches, with 2 episodes per switch. In the bottle domain, we generate successful videos for 2 different bottles and 2 different settings of the door (the door can cover either one of the 2 compartments), with 2 episodes in each of the 4 configurations.

Simulated User Model Parameters

For the simulated user input in Section 2.4, we set the standard deviation of the Gaussian user input noise to 0.1 in the switch domain, and 0.15 in the bottle domain.

Valve Rotation Experiment Details

We use a similar pre-training procedure as the one described earlier in Appendix 4.2 for the switch and bottle domains, except that we train for 8000 epochs, train on 50 human demonstrations, initialize the encoder and critic networks by running AWAC [71] on the human demonstrations for 25000 iterations, set the pre-training reward function to $\exp(-5 \cdot |\text{diff. between current and target angle in radians}|) - 1$, and only terminate an episode upon reaching 200 timesteps. Each observation includes the 3D end effector position, 4D end effector orientation, 3D end effector velocity, 3D valve position, 2D representation of the current valve orientation (sin and cos), 1D velocity of the valve joint, 7D arm joint

positions, and 7D arm joint velocities. To speed up human-in-the-loop learning, our encoders only operate on a subset of the observation features: the 3D end effector position, 2D valve orientation, and 3D valve position. The task specification τ^{spec} is the 2D valve orientation. Each action consists of 7D joint torques, clipped to $[-0.25, 0.25]$, as in the switch and bottle domains. During the pre-training phase, the initial valve angle is sampled uniformly at random from $[0, 2\pi)$, and the target angle is sampled uniformly at random from the same interval but excluding points within $\frac{\pi}{32}$ radians of the initial angle. During the calibration phase, the initial angle is always 0, and 7 videos are shown to the user, where the target angle is sampled uniformly at random from the discrete set $\{\frac{\pi}{4} \cdot k\}_{k=1}^{7}$. During the online learning phase, the initial angle is always 0, and the target angle is sampled uniformly at random from $[0, 2\pi)$ but excluding points within $\frac{\pi}{16}$ radians of the initial angle. The user is only allowed to end an episode when the valve angle has been within $\frac{\pi}{16}$ radians of the target angle for 20 consecutive steps. The episode automatically times out after 200 steps, but the user can still indicate a successful task completion after a timeout. Note that the task specification for each successful trajectory is extracted from the final state that was actually reached – since the user only needs to be within $\frac{\pi}{16}$ radians of the target to succeed, we may extract a specification that is near, but not necessarily identical to, the ground-truth specification. During the online learning phase, we use a batch size of 256, where each batch consists of 128 examples from the calibration dataset and 128 examples from the online dataset. The simulated static user input consists of a 2D target position on a circle centered at the valve. The simulated dynamic user input consists of a similar 2D target position, but dynamically adjusted so that it is always at most $\frac{\pi}{8}$ radians from the current valve angle, and also on the shortest arc from the current state to the target state. The simulated directional user input are 3D one-hot encodings for the actions {clockwise, counter-clockwise, remain} – the remain input is generated when the current valve angle is within $\frac{\pi}{16}$ radians of the target angle. All the simulated user inputs have i.i.d. isotropic Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.2I)$ added to them at each timestep. Following prior work on deep set encoders [81], we generate latent embeddings using an encoder that first processes individual $(\mathbf{s}_t, \mathbf{x}_t)$ pairs, and outputs the mean and variance of an isotropic Gaussian for each pair. The Gaussian factors for the most recent 10 timesteps are multiplied, and the latent embedding for the current timestep is sampled from this new Gaussian. During training, we sample the latent embedding from this Gaussian. During online episodes, we set the latent embedding to be the mean of the Gaussian. We set the dimensionality of the latent embedding space to $d = 2$. After a maximum number of 3 failed attempts to complete the current task, we automatically timeout and sample a new task. At the start of each episode, we add uniform random noise of magnitude 0.1 to the horizontal position of the valve, initialize the arm position with the same noisy procedure described in Appendix 4.2, and do not add noise to any other state variables (e.g., the distance to the wall).

Puck Pushing Experiment Details

We pre-train for 10000 epochs without any human demonstrations, and set the pre-training reward function to $1 - \beta\sigma(\alpha($change in distance of block position from goal+ $.5 \cdot$ change in distance of tool position from block), where σ is the sigmoid function. The puck and goal positions are initialized uniformly at random within a square of half extent (0.25, 0.15), which is the area that the end effector can reliably reach. The positions are constrained to be at least 0.1 apart. The puck is a cube with a half extent of 0.05, and is kept on the table by a gravity of 10, with mass of 0.1 and friction of 0.5. Each episode terminates automatically when the robot pushes the puck within 0.05 distance of the goal, or 200 timesteps expires. Each observation consists of the 3D tool position, 4D tool orientation, 3D block position, and 7D arm joint angles. The task specification τ^{spec} is the 3D position of the goal. As in the valve domain, note that the task specification for each successful trajectory is extracted from the final state that was actually reached – since the user only needs to be within 0.05 distance of the target to succeed, we may extract a specification that is near, but not necessarily identical to, the ground-truth specification. During the online learning phase, we use a batch size of 256, where each batch consists of 128 examples from the calibration dataset and 128 examples from the online dataset. The simulated user input is a 7D vector of arm joint torques, which is generated using a pre-trained oracle policy. The simulated user inputs are smoothed using an exponential moving average with smoothing factor $\alpha = 0.99$ – i.e., we set $\mathbf{x}_t \leftarrow \frac{1}{1-\alpha}(\alpha\mathbf{x}_{t-1} + (1-\alpha)\mathbf{x}_t)$. We use the same deep set encoder architecture as in the valve domain to represent the input encoder, and operate on the most recent 20 timesteps of observations and user inputs. We set the dimensionality of the latent embedding space to $d = 4$. As in the switch and bottle domains, after a maximum number of 5 failed attempts to complete the current task, we automatically timeout and sample a new task.

Details of User Study

Experiment Design

We recruited 10 male and 2 female participants, with an average age of 21. Each participant was provided with the rules of each domain (see Figure 4.4) and a short practice period of 10 episodes to familiarize themselves with the simulation. Each episode took an average of 13 seconds. Each participant completed three phases of experiments – A, B, and C – in each of the two domains. Before each phase in the switch domain, each participant generated 6 episodes of calibration data; in the bottle domain, 8 episodes. In phase A, they use the non-adaptive baseline interface to complete 50 episodes. In phase B, they use our method to complete 50 episodes. In phase C in the switch domain, they use our method to complete 50 episodes in which the task distribution is intentionally mismatched with the calibration data (see Section 2.3). In phase C in the bottle domain, they use our method to complete 50 episodes in which the environment conditions are intentionally mismatched with the calibration data (see Section 2.3). To control for the confounding effects of the user

learning or getting fatigued over the course of the full study, we counterbalanced the order of the three phases (i.e., 2 participants followed the order ABC, another 2 participants followed BAC, etc.).

Subjective Evaluations and Additional Quantitative Results

When prompted to "please describe your input strategy", participants responded as follows.

User 1:
Switch Domain:
After Phase A:
gaze at target switch 100% of the time, hold same gaze throughout
After Phase B:
same as phase A
After Phase C:
same as phase A
Bottle Domain:
After Phase A:
If door needs to be opened, then stare at door and then slide gaze to open the door once hand reaches door. After door is opened, stare at vase. If no door needs to be openend, just stare at vase.
After Phase B:
same as phase A
After Phase C:
same as phase A

User 2:
Switch Domain:
After Phase A:
I looked as far in the left/right direction as possible until the system reached the target
After Phase B:
same as phase A
After Phase C:
same as phase A
Bottle Domain:
After Phase A:
I looked as far in the left/right direction as possible until the system reached the target
After Phase B:
same as phase A
After Phase C:
I looked as far in the left/right direction as possible until the system reached the target. **I stopped looking at the door to "pull" it once the system had grasped the handle.**

User 3:
Switch Domain:
After Phase A:
Look towards a direction I wanted the arm to move in, then stare down the location when I wanted it to flip.
After Phase B:
exaggerate movement when it was wrong
After Phase C:
same as A
Bottle Domain:
After Phase A:
continuous movement
After Phase B:
try to gaze relative to the arm instead of at item
After Phase C:
left blank

User 4:
Switch Domain:
After Phase A:
I looked at the target and if the arm was going in the wrong place I compensated by looking further in the necessary direction
After Phase B:
Same strategy as before, **more compensating because missed more often**
After Phase C:
Same as before Bottle Domain:
After Phase A:
Just looked in the direction of the target, sometimes a little bit off to side of the shelf it was on.
After Phase B:
Same as before, but more often on target
After Phase C:
Same as before

User 5:
Switch Domain:
After Phase A:
look in direction to move
After Phase B:
exaggerated looking in direction
After Phase C:

same
Bottle Domain:
After Phase A:
lok left oor right
After Phase B:
same
After Phase C:
same, sometimes looko strraright

User 6:
Switch Domain:
After Phase A:
Exaggerate look in desired direction and look at target if robot was performing correct task
After Phase B:
Same as phase A
After Phase C:
Look at final target
Bottle Domain:
After Phase A:
Exaggerate for left side, look directly at target for right side
After Phase B:
Same as phase A
After Phase C:
Same as phase A

User 7:
Switch Domain:
After Phase A:
When the robot is close to the goal, gaze at some middle point between robot and goal; when robot is faraway from goal, gaze at some point that goes beyond the goal in the goal direction.
After Phase B:
Almost always look to the right of the goal; the farther the goal is, the farther the gaze will be from the goal as well.
After Phase C:
Look at a point slightly beyond the goal in the goal direction and adjust the gaze location according to robot behavior.
Bottle Domain:
After Phase A:
Look at a point beyond the goal in the goal direction, as far as possible, and holding the same gaze; it seems to be working most of the time unless the bottles are far away

from the middle.
After Phase B:
Same as phase A above.
After Phase C:
Same as phase A above.

User 8:
Switch Domain:
After Phase A:
Looked at target and exaggerated/altered gaze if arm moved too far in one direction
After Phase B:
Same as above
After Phase C:
Same as above
Bottle Domain:
After Phase A:
Same as above
After Phase B:
Same as above
After Phase C:
left blank

User 9:
Switch Domain:
After Phase A:
I started out looking at the final target, but also tried sweeping my gaze over the intended trajectory and also exaggeratedly looking in the direction I wanted the arm to go
After Phase B:
I did much the same, but when the arm seemed to be way off the intended trajectory, I'd test out looking in different directions to see how it would affect the trajectory. I wasn't sure how it was actually affecting the trajectory
After Phase C:
I would try looking directly at the target, then exxagerate my gaze a bit when the arm wasn't doing exactly what I wanted
Bottle Domain:
After Phase A:
I would look at points on the shelf that I wanted the arm to go to, even if not the final bottle. If the bottle was blocked, I'd look at the outer top left corner, then middle of outer edge, then middle divider to move the glass door. If the arm wasn't moving exactly in the direction I wanted, I'd exaggerate my gaze. If it wasn't blocked, I would look directly at the bottle.

After Phase B:
In this phase, I mainly exaggerated my gaze in the direction I wanted the arm to go, while relying on peripheral vision for tellling where the robot is and relying on the color change for telling if the episode was over
After Phase C:
I basically had the same strategy as last time, except I exaggerated my gaze even more. I also did the exaggeration in the calibration phase, when I hadn't before.

User 10:
Switch Domain:
After Phase A:
left blank
After Phase B:
left blank
After Phase C:
left blank
Bottle Domain:
After Phase A:
left blank
After Phase B:
left blank
After Phase C:
left blank

User 11:
Switch Domain:
After Phase A:
Gazing in the far extreme direction that I wanted the arm to move towards
After Phase B:
Gazing at the neighbor of the correct switch in the direction I wanted the arm to move in. (I.e., if I wanted the arm to move farther left, I gazed at the neighbor on the left).
After Phase C:
Same as above
Bottle Domain:
After Phase A:
Gazing in the extreme direction I wanted the arm to move in
After Phase B:
I fixed my gaze directly at the bottle I wanted the system to grab.
After Phase C:
Same as above

User 12:

Switch Domain:
After Phase A:
I stared at the blue sphere for most of the time and would try to correct the robot if it went to the wrong one by exaggerating my look in a certain direction.
After Phase B:
I started with the same strategy as in phase 1 where I would stare at the blue sphere above the switch until the arm went there. However, I noticed that this only worked for the switches on the right side, and it wouldn't ever go to the first two switches on the left side–this made me try to compensate for the arm by looking more to the left but it still did not work so I tried different corners of the screen.
After Phase C:
I started again by trying to stare at the blue sphere and track it with my eyes. However, it seemed like the robot had the opposite problem as phase 2. Instead of always going for the right, it would only go for the second switch on the left. I also tried compensating by looking as far right as I could, but it did not seem to affect the arm.
Bottle Domain:
After Phase A:
If the bottle was behind the glass door, I would stare at the edge of the door and then pretend to slide it with my eyes. Then I would stare at the blue part of the bottle. If the bottle was not behind a door, I looked at the bottle directly.
After Phase B:
I used a similar strategy to phase 1, where if the bottle was behind the glass door I looked at the edge of the glass door. Once the arm reached the handle of the door, I panned my eyes over to the opposite edge of the box. Once the door was fully open, I would look at the blue part of the bottle.
After Phase C:
I used my strategy from phase 2 with some slight modifications. I noticed that the arm would get caught on the wrong side of the handle sometimes after opening the door and it would close the door again, so I would sometimes try moving my eyes up and down to get the arm out of the way. There were also times where the arm would open the door half way and then it would move away from the handle, so I tried looking at the opposite corner of the box to keep the arm on the handle.

These responses show that users employed a variety of different communication styles, including looking directly at the target (users 1, 4, and 8), looking at distant parts of the screen to indicate different targets (users 2 and 3), exaggerating their gaze to correct the robot (users 3-6 and 8), and dynamically guiding the robot to subgoals (users 1, 2, and 7).

Table 4.2: User Study - Subjective Evaluation

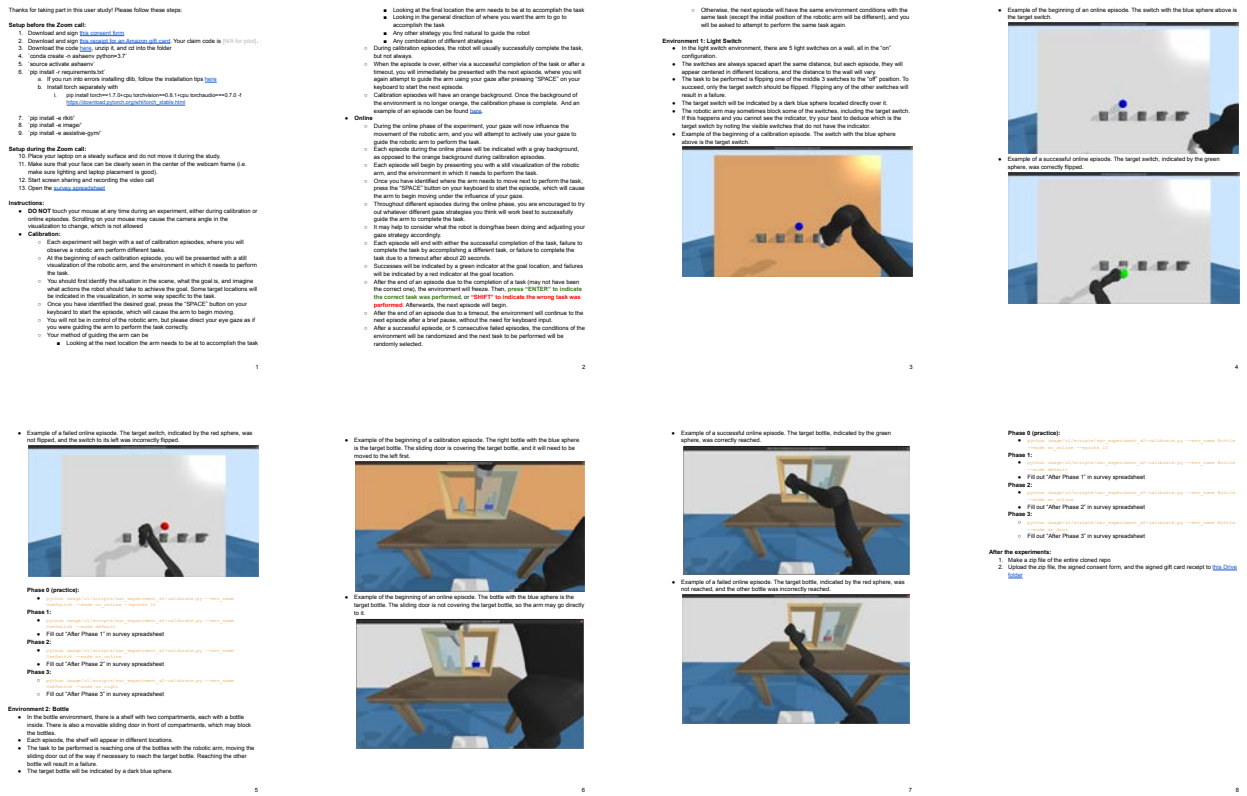
	Bottle			Switch		
	ASHA	Baseline	p	ASHA	Baseline	p
The system performed the task I wanted	4.8	3.9	$> .1$	**4.2**	**3.2**	$< .1$
I felt in control	4.0	3.0	$> .1$	3.6	3.1	$> .1$
The system responded to my input...						
...in the way that I expected	4.5	3.4	$> .1$	3.5	3.2	$> .1$
The system was competent at performing tasks...						
...even if they weren't the tasks I wanted	5.2	4.9	$> .1$	5.1	4.7	$> .1$
The system improved over time	**4.9**	**3.5**	$< .05$	3.9	3.0	$> .1$
I improved at using the system over time	4.0	3.2	$> .1$	3.9	3.7	$> .1$
I always looked directly at my final target...						
...holding the same gaze throughout an episode	4.2	3.5	$> .1$	3.5	2.9	$> .1$
I compensated for flaws in the system...						
...by changing my gaze over time	4.7	4.0	$> .1$	5.4	5.5	$> .1$

Subjective evaluations from the 12 participants in the user study. 'Baseline' refers to the non-adaptive baseline interface. Means reported below for responses on a 7-point Likert scale, where 1 = Strongly Disagree, 4 = Neither Disagree nor Agree, and 7 = Strongly Agree. p-values from a one-way repeated measures ANOVA with the presence of ASHA as a factor influencing responses. While none of the differences shown here are statistically significant, ASHA does outperform the baseline method in terms of the objective metrics analyzed in Section 2.3.

Table 4.3: User Study - Quantitative Evaluation

	Switch		Bottle	
	Success Rate	Failed Attempts	Success Rate	Failed Attempts
Random Latent (Baseline)	0.20 ± 0.02	2.7 ± 0.1	0.49 ± 0.02	1.0 ± 0.1
Non-Adaptive (Baseline)	0.41 ± 0.04	1.8 ± 0.2	0.65 ± 0.04	1.8 ± 0.2
ASHA (Ours)	$\mathbf{0.52 \pm 0.04}$	$\mathbf{1.6 \pm 0.2}$	$\mathbf{0.74 \pm 0.04}$	0.8 ± 0.2
ASHA with Task/Env. Shift (Ours)	0.43 ± 0.08	2.1 ± 0.4	0.74 ± 0.06	$\mathbf{0.6 \pm 0.2}$

Means measured across 50 episodes, and standard errors measured across the 12 participants. 'Failed Attempts' refers to the number of failed attempts per task, for which the maximum value is 5 due to timeouts (see Appendix 4.2). In the switch domain, 'ASHA with Task/Env. Shift' refers to task distribution shift (see Section 2.3). In the bottle domain, 'ASHA with Task/Env. Shift' refers to environment shift (see Section 2.3). These results show that ASHA outperforms the baselines, not just in terms of learning speed or final performance, but also in terms of cumulative regret throughout the experiment.



Thanks for taking part in this user study! Please follow these steps:

Setup before the Zoom call:

1. Download and sign this consent form
2. Download and sign this receipt for an Amazon gift card. Your claim code is [N/A for pilot].
3. Download the code here, unzip it, and cd into the folder
4. `conda create -n ashaenv python=3.7`
5. `source activate ashaenv/`
6. `pip install -r requirements.txt`
   a. If you run into errors installing dlib, follow the installation tips here
   b. Install torch separately with
      i. `pip install torch==1.7.0+cpu torchvision==0.8.1+cpu torchaudio===0.7.0 -f https://download.pytorch.org/whl/torch_stable.html`
7. `pip install -e rlkit/`
8. `pip install -e image/`
9. `pip install -e assistive-gym/`

Setup during the Zoom call:

10. Place your laptop on a steady surface and do not move it during the study.
11. Make sure that your face can be clearly seen in the center of the webcam frame (i.e. make sure lighting and laptop placement is good).
12. Start screen sharing and recording the video call
13. Open the survey spreadsheet

Instructions:

- **DO NOT** touch your mouse at any time during an experiment, either during calibration or online episodes. Scrolling on your mouse may cause the camera angle in the visualization to change, which is not allowed
- **Calibration:**
  - Each experiment will begin with a set of calibration episodes, where you will observe a robotic arm perform different tasks.
  - At the beginning of each calibration episode, you will be presented with a still visualization of the robotic arm, and the environment in which it needs to perform the task.
  - You should first identify the situation in the scene, what the goal is, and imagine what actions the robot should take to achieve the goal. Some target locations will be indicated in the visualization, in some way specific to the task.
  - Once you have identified the desired goal, press the "SPACE" button on your keyboard to start the episode, which will cause the arm to begin moving.
  - You will not be in control of the robotic arm, but please direct your eye gaze as if you were guiding the arm to perform the task correctly.
  - Your method of guiding the arm can be
    - Looking at the next location the arm needs to be at to accomplish the task
    - Looking at the final location the arm needs to be at to accomplish the task
    - Looking in the general direction of where you want the arm to go to accomplish the task
    - Any other strategy you find natural to guide the robot
    - Any combination of different strategies
  - During calibration episodes, the robot will usually successfully complete the task, but not always.
  - When the episode is over, either via a successful completion of the task or after a timeout, you will immediately be presented with the next episode, where you will again attempt to guide the arm using your gaze after pressing "SPACE" on your keyboard to start the next episode.
  - Calibration episodes will have an orange background. Once the background of the environment is no longer orange, the calibration phase is complete. And an example of an episode can be found here.
- **Online**
  - During the online phase of the experiment, your gaze will now influence the movement of the robotic arm, and you will attempt to actively use your gaze to guide the robotic arm to perform the task.
  - Each episode during the online phase will be indicated with a gray background, as opposed to the orange background during calibration episodes.
  - Each episode will begin by presenting you with a still visualization of the robotic arm, and the environment in which it needs to perform the task.
  - Once you have identified where the arm needs to move next to perform the task, press the "SPACE" button on your keyboard to start the episode, which will cause the arm to begin moving under the influence of your gaze.
  - Throughout different episodes during the online phase, you are encouraged to try out whatever different gaze strategies you think will work best to successfully guide the arm to complete the task.
  - It may help to consider what the robot is doinghas been doing and adjusting your gaze strategy accordingly.
  - Each episode will end with either the successful completion of the task, failure to complete the task by accomplishing a different task, or failure to complete the task due to a timeout after about 20 seconds.
  - Successes will be indicated by a green indicator at the goal location, and failures will be indicated by a red indicator at the goal location.
  - After the end of an episode due to the completion of a task (may not have been the correct one), the environment will freeze. Then, press "ENTER" to indicate the correct task was performed, or "SHIFT" to indicate the wrong task was performed. Afterwards, the next episode will begin.
  - After the end of an episode due to a timeout, the environment will continue to the next episode after a brief pause, without the need for keyboard input.
  - After a successful episode, or 5 consecutive failed episodes, the conditions of the environment will be randomized and the next task to be performed will be randomly selected.
  - Otherwise, the next episode will have the same environment conditions with the same task (except the initial position of the robotic arm will be different), and you will be asked to attempt to perform the same task again.

Environment 1: Light Switch

- In the light switch environment, there are 5 light switches on a wall, all in the "on" configuration.
- The switches are always spaced apart the same distance, but each episode, they will appear centered in different locations, and the distance to the wall will vary.
- The task to be performed is flipping one of the middle 3 switches to the "off" position. To succeed, only the target switch should be flipped. Flipping any of the other switches will result in a failure.
- The target switch will be indicated by a dark blue sphere located directly over it.
- The robotic arm may sometimes block some of the switches, including the target switch. If this happens and you cannot see the indicator, try your best to deduce which is the target switch by noting the visible switches that do not have the indicator.
- Example of the beginning of a calibration episode. The switch with the blue sphere above is the target switch.

- Example of the beginning of an online episode. The switch with the blue sphere above is the target switch.
- Example of a successful online episode. The target switch, indicated by the green sphere, was correctly flipped.

- Example of a failed online episode. The target switch, indicated by the red sphere, was not flipped, and the switch to its left was incorrectly flipped.

Phase 0 (practice):

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name SwitchLift --mode no_online --nepochs 10`

Phase 1:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name SwitchLift --mode default`
- Fill out "After Phase 1" in survey spreadsheet

Phase 2:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name SwitchLift --mode no_online`
- Fill out "After Phase 2" in survey spreadsheet

Phase 3:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name SwitchLift --mode no_above`
- Fill out "After Phase 3" in survey spreadsheet

Environment 2: Bottle

- In the bottle environment, there is a shelf with two compartments, each with a bottle inside. There is also a movable sliding door in front of compartments, which may block the bottles.
- Each episode, the shelf will appear in different locations.
- The task to be performed is reaching one of the bottles with the robotic arm, moving the sliding door out of the way if necessary to reach the target bottle. Reaching the other bottle will result in a failure.
- The target bottle will be indicated by a dark blue sphere.

- Example of a beginning of a calibration episode. The right bottle with the blue sphere is the target bottle. The sliding door is covering the target bottle, and it will need to be moved to the left first.
- Example of the beginning of an online episode. The bottle with the blue sphere is the target bottle. The sliding door is not covering the target bottle, so the arm may go directly to it.

- Example of a successful online episode. The target bottle, indicated by the green sphere, was correctly reached.
- Example of a failed online episode. The target bottle, indicated by the red sphere, was not reached, and the other bottle was incorrectly reached.

Phase 0 (practice):

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name BottleReach --mode no_online --nepochs 10`

Phase 1:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name BottleReach --mode default`
- Fill out "After Phase 1" in survey spreadsheet

Phase 2:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name BottleReach --mode no_online`
- Fill out "After Phase 2" in survey spreadsheet

Phase 3:

- `python image/rl/scripts/aas_experiment_s2-calibrate.py --env_name BottleReach --mode no_above`
- Fill out "After Phase 3" in survey spreadsheet

After the experiments:

1. Make a zip file of the entire cloned repo
2. Upload the zip file, the signed consent form, and the signed gift card receipt to this Drive folder

Figure 4.4: Instructional document for participants in the user study

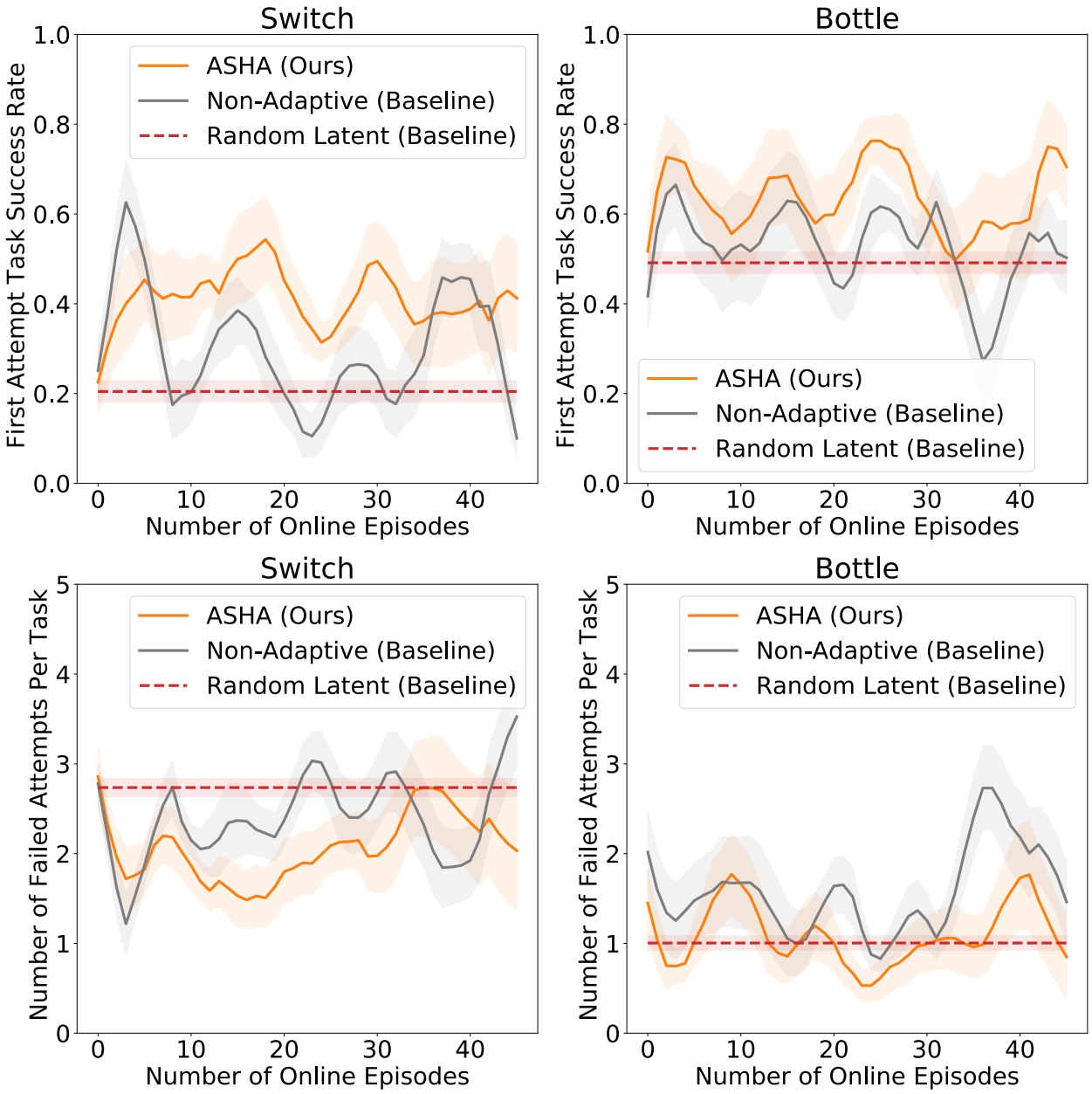


Figure 4.5: Error bars show standard error across the 12 participants. The maximum number of attempts per task is 5 (see Appendix 4.2). Both performance metrics – success rate on the first attempt for each task, and number of failed attempts per task – generally illustrate similar gaps between ASHA and the baseline methods. However, in the bottle domain, while ASHA achieves a higher success rate than the random-latent baseline, it does not achieve a lower number of failed attempts. This can be attributed to selection effects for difficult tasks in subsequent attempts – see Figure 4.7 for details.

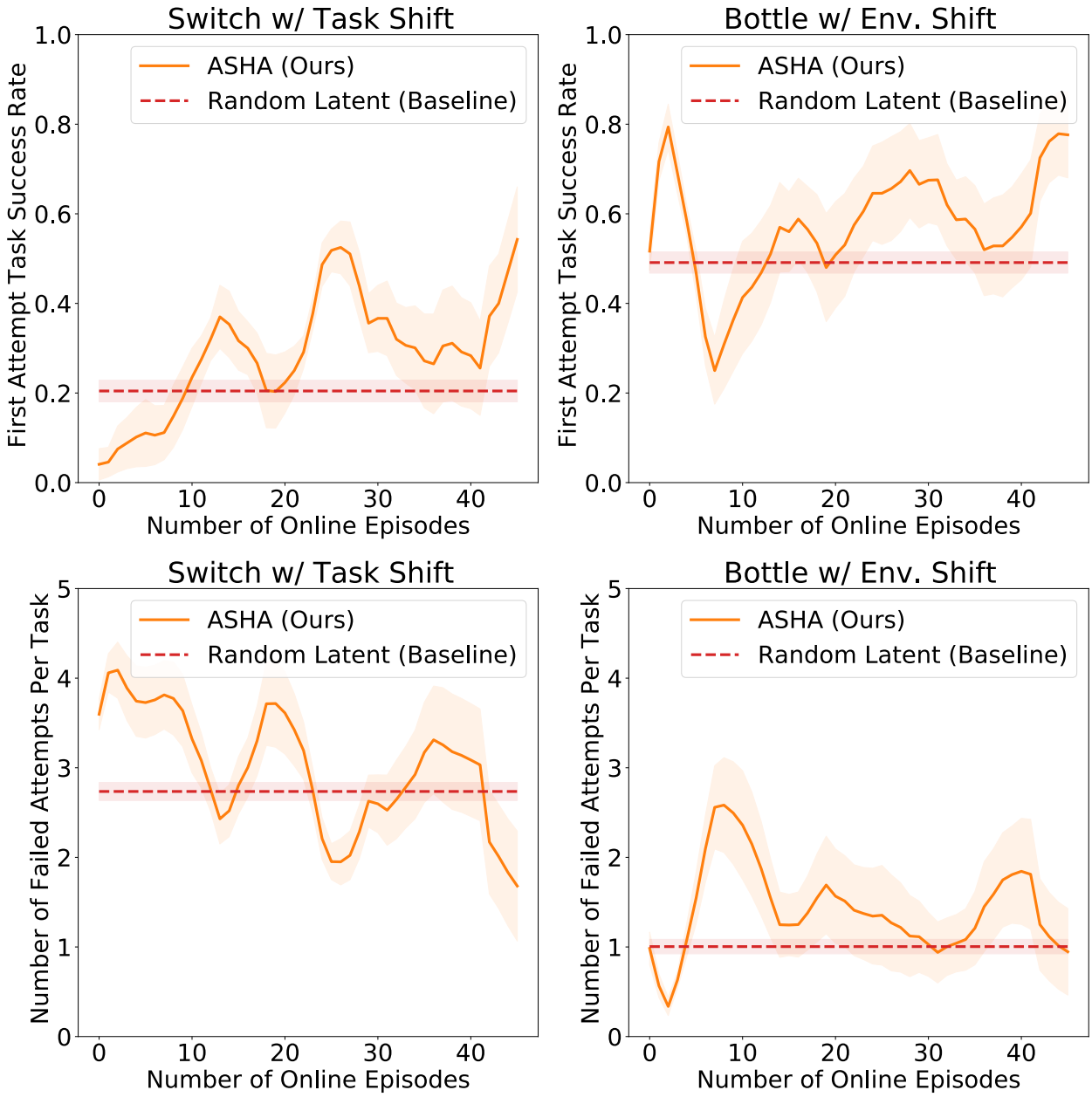


Figure 4.6: Error bars show standard error across the 12 participants. The maximum number of attempts per task is 5 (see Appendix 4.2). Both performance metrics – success rate on the first attempt for each task, and number of failed attempts per task – generally illustrate similar gaps between ASHA and the random-latent baseline method. However, in the bottle domain, while ASHA achieves a higher success rate than the baseline, it does not achieve a lower number of failed attempts. This can be attributed to selection effects for difficult tasks in subsequent attempts – see Figure 4.7 for details.

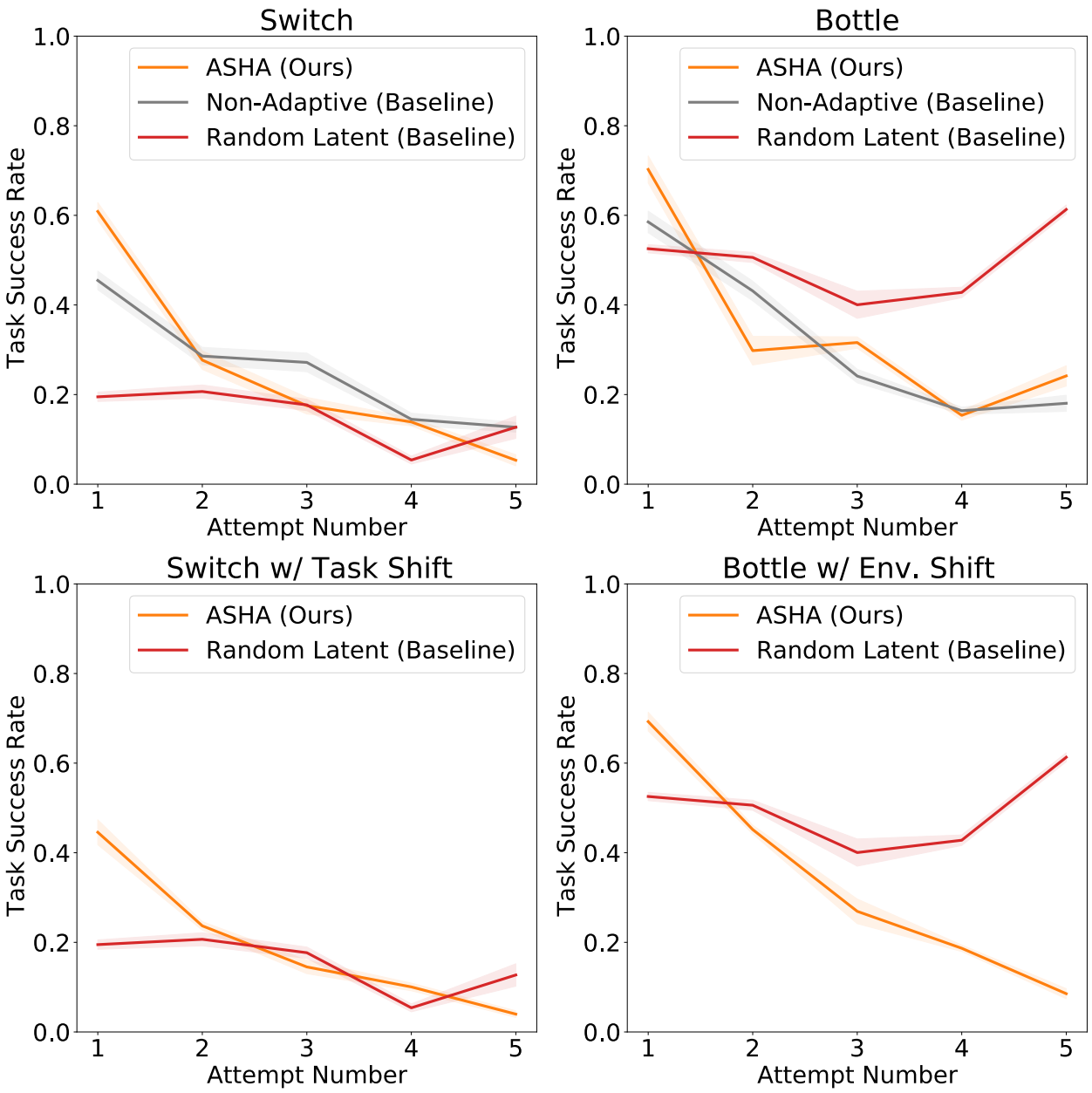


Figure 4.7: Error bars show standard error across the 12 participants. Performance of ASHA and the non-adaptive baseline tends to decrease on later attempts due to selection effects: tasks in which the user's inputs are easy to interpret for ASHA and the non-adaptive baseline are completed within a small number of attempts, while tasks for which user inputs are difficult to interpret for these two methods tend to require more attempts. Performance of the random-latent baseline is relatively constant across attempts, since it does not take user input, and hence difficult episodes are not selected for in later attempts. On the first attempt, where selection effects do not exist for any of the three methods, ASHA outperforms both the non-adaptive and random-latent baselines.

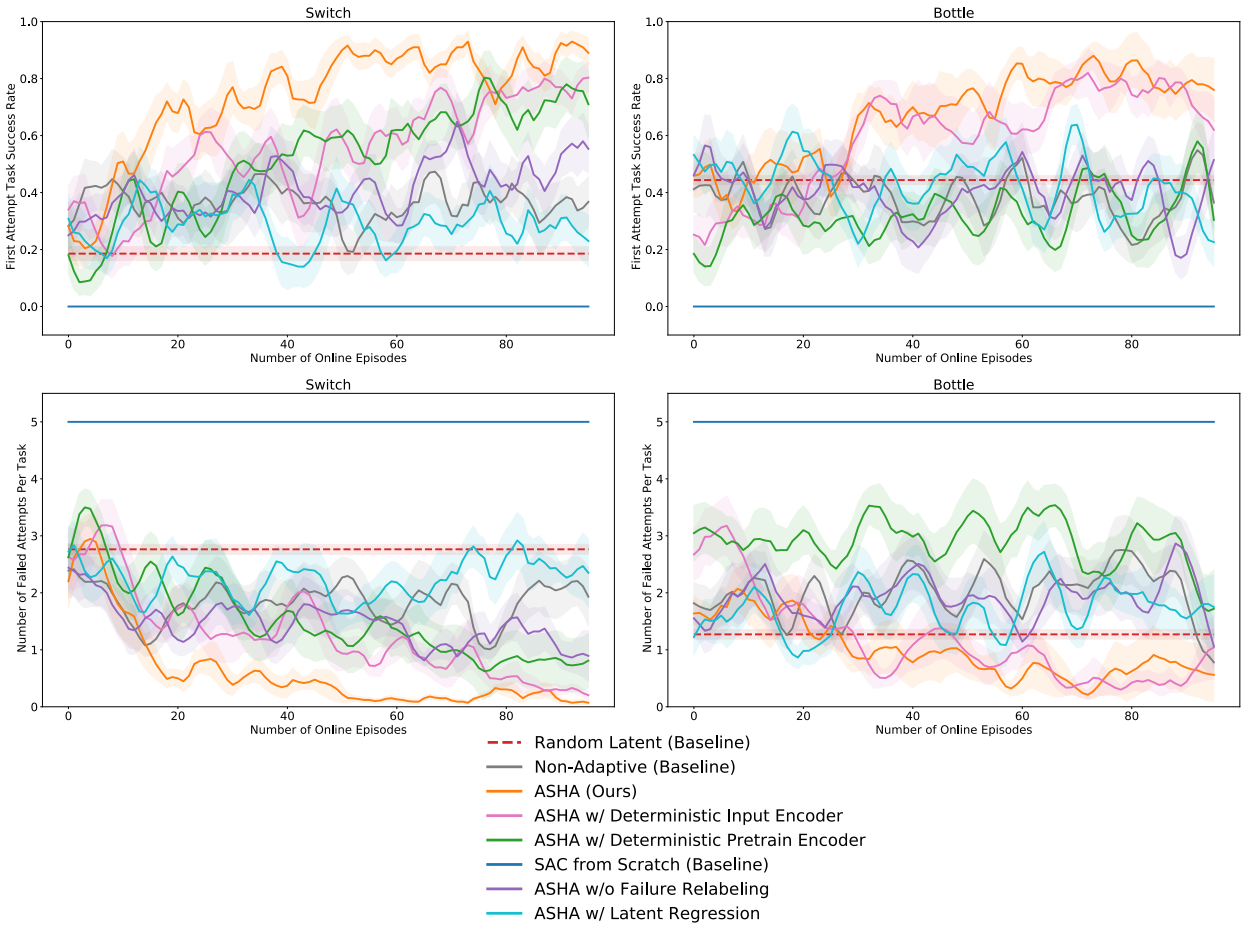


Figure 4.8: Results from the ablation experiments in Section 2.4. Error bars show standard error across 10 random seeds. The maximum number of attempts per task is 5 (see Appendix 4.2). As in Table 2.1 in Section 2.4, the results show that all the ablated variants of ASHA perform worse than the full ASHA method, suggesting that sampling from a stochastic input encoder f_θ^{inpt} improves exploration (**Q1**), pre-training with a VIB and reusing the pre-trained latent-conditioned policy g_ϕ 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 latents (**Q5**).

Table 4.4: Ablation Experiments

	Switch		Bottle	
	Success Rate	Failed Attempts	Success Rate	Failed Attempts
Random Latent (Baseline)	0.19 ± 0.02	2.8 ± 0.1	0.44 ± 0.02	1.3 ± 0.1
Non-Adaptive (Baseline)	0.50 ± 0.05	1.3 ± 0.2	0.53 ± 0.02	1.3 ± 0.1
ASHA (Ours)	$\mathbf{0.83 \pm 0.02}$	$\mathbf{0.3 \pm 0.0}$	$\mathbf{0.79 \pm 0.03}$	$\mathbf{0.6 \pm 0.2}$
ASHA w/ Det. Input Enc. (**Q1**)	0.70 ± 0.03	0.7 ± 0.1	0.73 ± 0.02	0.6 ± 0.1
ASHA w/ Det. Pre-train Enc. (**Q2**)	0.66 ± 0.06	1.0 ± 0.3	0.46 ± 0.03	2.1 ± 0.2
SAC from Scratch (**Q3**)	0.00 ± 0.00	5.0 ± 0.0	0.00 ± 0.00	5.0 ± 0.0
ASHA w/o Failure Relabeling (**Q4**)	0.54 ± 0.03	1.0 ± 0.1	0.55 ± 0.02	1.3 ± 0.1
ASHA w/ Latent Regression (**Q5**)	0.41 ± 0.04	1.7 ± 0.2	0.57 ± 0.02	1.1 ± 0.1

Means and standard errors measured across 100 episodes and 10 random seeds. See Figure 4.8 in the appendix for more detailed plots. As in Table 2.1 in Section 2.4, the results show that all the ablated variants of ASHA perform worse than the full ASHA method, suggesting that sampling from a stochastic input encoder f_θ^{inpt} improves exploration (**Q1**), pre-training with a VIB and reusing the pre-trained latent-conditioned policy g_ϕ 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 latents (**Q5**).