Pre-print of accepted manuscript.
To appear at the 17th ACM/IEEE Int. Conf. on Human-Robot Interaction (HRI), 2022.

Correct Me If I'm Wrong: Using Non-Experts to Repair Reinforcement Learning Policies

Sanne van Waveren, Christian Pek, Jana Tumova, and Iolanda Leite
Division of Robotics, Perception and Learning
KTH Royal Institute of Technology
Stockholm, Sweden
{sannevw,pek2,tumova,iolanda}@kth.se

***Abstract*—Reinforcement learning has shown great potential for learning sequential decision-making tasks. Yet, it is difficult to anticipate all possible real-world scenarios during training, causing robots to inevitably fail in the long run. Many of these failures are due to variations in the robot's environment. Usually experts are called to correct the robot's behavior; however, some of these failures do not necessarily require an expert to solve them. In this work, we query non-experts online for help and explore 1) if/how non-experts can provide feedback to the robot after a failure and 2) how the robot can use this feedback to avoid such failures in the future by generating shields that restrict or correct its high-level actions. We demonstrate our approach on common daily scenarios of a simulated kitchen robot. The results indicate that non-experts can indeed understand and repair robot failures. Our generated shields accelerate learning and improve data-efficiency during retraining.**

***Index Terms*—robot failure; policy repair; non-experts; shielded reinforcement learning**

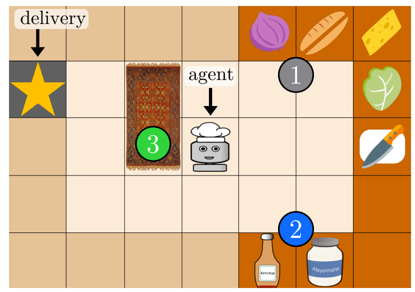


Fig. 1. The three failure corrections considered in this work: (1) *action refinement:* the onion is not yet cut and the robot needs to cut it; (2) *alternative item:* the ketchup is empty and the robot needs to find a replacement; (3) *forbidden action:* the robot needs to avoid getting stuck on the carpet.

I. Introduction

When we deploy learning-based robots in the real world, their policy will inevitably fail at some point, e.g., due to changes in the environment, such as additional or missing objects [1], [2]. Typically, such failures require costly expert intervention, substantial time or data to retrain the policy. Yet, certain failures, especially when they happen in everyday context, are also understandable by non-experts (NEs).

In this paper, we ask NEs to identify and repair the robot's policy by correcting its high-level actions. Specifically, we ask NE input for three types of failure corrections. Let us consider a cooking task in which the robot has to assemble and deliver a sandwich as an example (see Fig. 1):

(1) **Action refinement:** The robot learned to fetch an onion slice, which is no longer available during deployment. To correct this failure, the robot needs to *refine its action*, e.g., fetch and chop the whole onion.
(2) **Alternative item:** The robot learned to fetch the ketchup for the sandwich but fails during deployment because it ran out of ketchup. The robot needs to use an *alternative item*, e.g., it could use mayonnaise instead of ketchup.
(3) **Forbidden action:** The robot learned to move on linoleum floor tiles, but in the deployment environment, it gets stuck on a carpet. The robot should treat moving to the carpet at all times as a *forbidden action*.

While many situations can already be anticipated during development, it becomes tedious, or even impossible, to anticipate all possible failures [3]. For the failure correction types described above, we can update the robot's policy by incorporating simple rules or domain knowledge that even people who do not necessarily have programming experience (NEs) could provide. We focus on crowdsourcing to alleviate the need for a teacher to constantly monitor the robot.

This work proposes a policy repair approach in which a robot queries for NE input only *after* it detects a failure state. In an online study, we explore if/how NEs can provide such corrective input. We evaluate the efficiency of our approach through a series of experiments, in which we construct shields from ground-truth and incorrect NE feedback. We show for the first time that shielded RL can be used to repair robot failures.

II. Related Work

Robots need the ability to recover from failure states without expert intervention. We review work that leverages NE input for robot learning and approaches to enforce robot behaviors.

This work was financially supported by the Swedish Research Council (reg. number 2017-05189 and 2017-05102), the Swedish Foundation for Strategic Research (SSF FFL18-0199), the Vinnova Competence Center for Trustworthy Edge Computing Systems and Applications at KTH Royal Institute of Technology and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

A. Learning from NE Feedback

Human feedback can enable agents to solve sequential decision-making tasks for which the rewards are not easily defined [4]. Prior work used NE input to author [5], [6] or interactively shape robot behaviors [7], accelerate the learning, guide exploration, and prevent undesired actions [8]–[11].

Common approaches to learn from human teachers are Learning from Demonstration (LfD) [12], [13] and interactive RL (IRL) [8], [9], [14], [15]. While LfD is typically used to teach new skills through kinesthetic guidance, IRL can solve sequential decision-making tasks, encoding human feedback as numeric rewards, correction of actions, and preferences.

Human feedback, e.g., through guidance and positive rewards [8], can improve learning efficiency and make exploration more robust [9]. Such explicit rewards are used incorporate domain knowledge into RL [15], [16]. Alternatively, teacher intervention can serve as implicit negative rewards, indicating that actions are undesirable [17], [18].

These works highlight the potential of human feedback in RL, yet, often require teachers to constantly monitor the entire training process. We actively query people for input only when it is really needed, i.e., after a failure occurs. Our approach leverages NEs during the deployment phase, allowing robots to learn even after they complete training in a controlled environment, without constant human supervision.

B. Enforcing Desired Robot Behaviors

Constrained RL (CRL) enforces desired behaviors by restricting the state or action spaces, providing penalty rewards, or by interfering with the policy update, e.g., through Lagrangian methods [19]–[21]. However, CRL requires technical expertise and can often not formally ensure that undesired behaviors are always avoided.

Formal guarantees are crucial for robots' real-world deployment and trustworthiness [22]. Contrary to CRL, formal approaches verify whether a system adheres to a given specification at all times. Specifications define desired behaviors, e.g., avoiding obstacle collisions or never fetching incorrect items. Temporal Logics (TLs) are popular specification tools due to their resemblance to natural language, balancing the trade-off between rigorousness and simplicity for NEs. TL specifications can be used to verify system behaviors [23], [24]. Linear TL (LTL) has been used to verify human-robot interactions [5], or help agents learn in game environments [25], [26]. While TL-based approaches are effective, they often interfere with the RL problem, e.g., by pruning actions and paths in the RL model [20], potentially limiting exploration.

Rather than directly changing the policy, shielded RL (SRL) restricts the agent's actions [27]–[29]. This way, it minimally interferes with the RL model while still enforcing desired behaviors. Shields can be constructed from TL specifications to define a set of constraints, e.g., *never move out of reach of the Wi-Fi router*, and to prevent actions that will lead to specification violations. To date, SRL is primarily used to enforce safety properties, but not to repair policies in case of failures. SRL often requires experts to construct specifications or to turn them into shields, and an automaton with all possible states for safety checks. We apply shields only to the states the agent actually visits, reducing computational complexity.

III. Correcting Failures of Policies

Fig. 2 illustrates the main steps of our policy repair approach. Initially, an RL agent learns a decision-making task in a training kitchen environment where it can find all the required ingredients to make sandwiches. Being deployed in a target environment, the agent's policy of high-level actions may fail, due to changes not seen during training, e.g., it needs a chopped onion but there is none.

If the agent detects a failure state, it sends the sequence of performed actions so far (a failure trace) and information about the environment to NEs for feedback. For instance, the NEs may suggest that the agent should fetch an onion and chop it. We use this feedback to generate shields that repair the policy by either 1) refining actions, 2) choosing alternative items, or 3) forbidding certain actions. We retrain the agent with the generated shield to correct the failure.

A. Training the Reinforcement Learning System

We consider the problem of correcting failures of RL agents in sequential decision-making tasks. In these tasks, the agent learns a policy $\pi = (a_1, a_2, \ldots, a_n)$ of available high-level actions $a_i \in \mathcal{A}$ that leads the agent from its current state $s_I \in \mathcal{S}$ to a predefined goal state $s_G \in \mathcal{S}$. We assume that if a high-level action is feasible the agent can successfully execute it. In Fig. 1, the states $\mathcal{S}$ correspond to the agent's position on the floor tiles and the available set of actions to moving around the kitchen, i.e., $\mathcal{A} = \{\text{left}, \text{right}, \text{up}, \text{down}\}$. The transition function $\delta(s_i, a_i) = s_{i+1}$ returns the subsequent state s_{i+1} when applying action a_i in state s_i. Some actions may require a parameter $p \in \mathbb{P}$, e.g., to encode that an action a is performed on an object p, which we denote with $a^{\langle p \rangle}$. We assume that the agent is aware of all objects in its environment.

The agent learns an optimal policy π, which maximizes the cumulative reward, by interacting with a training environment, and collecting rewards $r \in \mathbb{R}$ over time, given by the function $R : (s, a) \mapsto r$. After training, the optimal policy is deployed on the agent so that it can perform the task in the deployment environment, e.g., the agent owner's kitchen.

B. Query NEs to Correct Failures after Deployment

We assume that the agent can detect a failure state $s_F = \delta(s_i, a_i)$, e.g., through a failure detection module. The agent prepares a query $Q = \big(s_F, (a_1, \ldots, a_F), T\big)$ to send to NEs, which consists of the failure state s_F, the executed action sequence $(a_1, \ldots, a_F)$ until the failed action a_F, and a function T that maps the failure state and action sequence into a human interpretable format. For instance, if the agent fails to fetch a chopped onion because there are no more chopped onions (see failure (1) in Fig. 1), the function T maps the failure state into a visual representation of the environment (see Fig. 3a), the action sequence into a textual representation (see Fig. 3b), and creates a failure query (see Fig. 3c). The

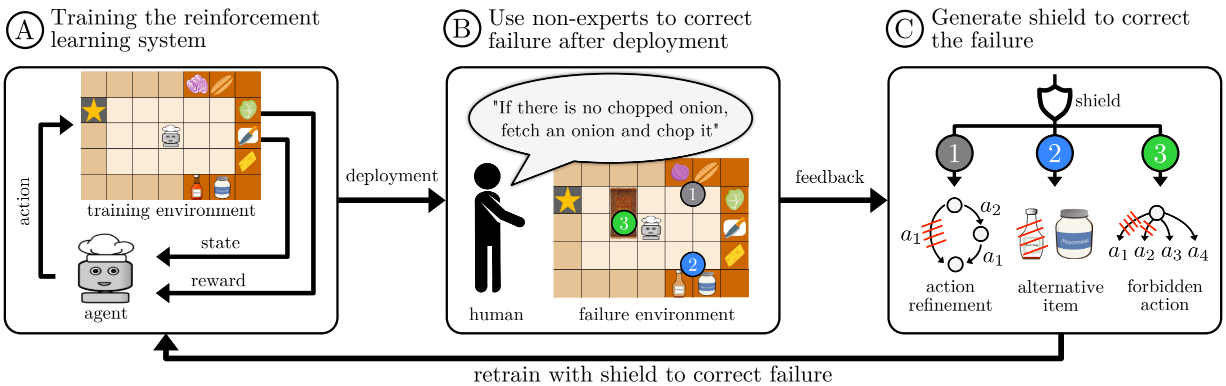
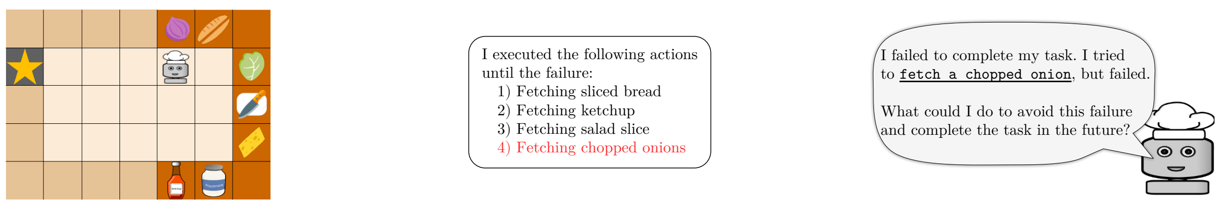


Fig. 2. Overview of our framework to correct policy failures after deployment using NE feedback. (A) The robot learns to solve its task in a training environment and is then deployed in a target environment. (B) In the target environment, the robot may encounter failures, e.g., it cannot find chopped onions (see number 1). We show the sequence of already executed actions and the failure environment to NEs and ask them how the robot could avoid this failure in the future. (C) From the human feedback we generate a shield that either 1) refines actions (chop an onion if there is no chopped onion), 2) picks alternative items (use mayonnaise if there is no ketchup any more), or 3) forbids certain actions (never move to a carpeted tile).



(a) Failure environment (b) Failure trace (c) Failure query message

Fig. 3. Example on what a failure query to a NE can look like. Fig. 3a shows a 2D view of the failure scenario. Fig. 3b summarizes the actions that the agent has executed until the failed action. Fig. 3c shows the failure query message that the agent sends to the NE.

design of T depends on the application and may be obtained empirically in user studies, which is not this work's focus.

C. Generating Shields to Correct Failures

We generate shields to 1) refine actions, 2) suggest alternative items to use, or 3) forbid actions that lead to undesired behaviors in our kitchen example in Fig. 1.

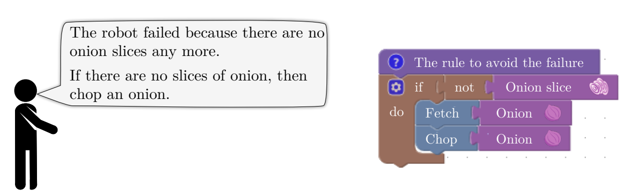
a) Action refinement: If the agent's failed action a_F can be corrected by proposing alternative actions, we apply action refinement. In the chopped onion example (section III-B) NEs may suggest to chop an onion first, e.g., in natural language or through a block program (See Fig. 4a and Fig. 4b, respectively). This feedback indicates that the action $\text{fetch}^{\text{onionSlice}}$ will result in a failure state when there are no more chopped onions. We denote a set of desired states that the agent should stay in as $\mathcal{S}_{\text{desired}} := \mathcal{S} \setminus \mathcal{S}_F$, which does not include the set of failure states $\mathcal{S}_F$, e.g., states in which the agent ends up on the carpeted floor tiles in the coffee example. If the agent were to leave $\mathcal{S}_{\text{desired}}$ in the next time step, we would correct the chosen action a with the action(s) suggested by the NE. The function $c(s,a)$ returns these corrective action(s), e.g., $\text{chop}^{\text{onion}}$ instead of $\text{fetch}^{\text{onionSlice}}$. We use the refine shield to retrain the agent and correct action a to avoid the failure:

$$\text{refine}_s(a) = \begin{cases} a, & \text{if } \delta(s,a) \in \mathcal{S}_{\text{desired}} \\ c(s,a), & \text{if } \delta(s,a) \notin \mathcal{S}_{\text{desired}} \end{cases} \tag{1}$$

b) Alternative item: The agent gets an alternative item p_{a} in its action $a^{\langle p_{\text{o}} \rangle}$, if it fails because it can no longer use the item p_{o}. For instance, the agent may not be able to use the ketchup (p_{o}) if it is empty (see failure (2) in Fig. 2). NEs may suggest to use an alternative item p_{a}, e.g., mayonnaise, instead. To avoid the failure, our shield selects an alternative item p_{a} if the original item p_{o} is not in the environment:

$$\text{alt}(a^{\langle p_{\text{o}} \rangle}) = \begin{cases} a^{\langle p_{\text{o}} \rangle}, & \text{if } p_{\text{o}} \text{ in environment} \\ a^{\langle p_{\text{a}} \rangle}, & \text{if } p_{\text{o}} \text{ not in environment} \end{cases} \tag{2}$$

c) Forbidden actions: We remove actions from the set of actions $\mathcal{A}$ when they lead to an undesired state of the agent, i.e., $\delta(s,a) \notin \mathcal{S}_{\text{desired}}$. For instance, the human may tell the agent it should avoid the carpet since it gets stuck on it (see failure (3) in Fig. 1). We denote a set of allowed actions that the agent can choose from in state s as $\mathcal{A}_{\text{allowed}}(s)$, which



(a) Natural language feedback (b) Block-based feedback

Fig. 4. Examples of open answer and block-based NE feedback.

does not include actions that, if executed, result in undesired states (e.g., ending up on a carpeted tile):

$$\mathcal{A}_{\text{allowed}}(s) = \{a \in \mathcal{A} \mid \delta(s,a) \in \mathcal{S}_{\text{desired}}\} \tag{3}$$

Finally, we retrain the agent and restrict its actions to $\mathcal{A}_{\text{allowed}}$.

D. Algorithm

Alg. 1 summarizes the main steps of our approach. We first prepare the query Q and ask NEs for feedback. Afterwards, we add the new shields to our set of shields, which might already contain shields obtained in previous failures. The variables ∇_{ref}, ∇_{alt}, and ∇_{all} denote the sets of action refinement, alternative item, and forbidden action shields, respectively. With the updated set of shields, we retrain the policy to ensure its optimality even after applying corrections. We obtain a desired action from the shielded set of actions in Line 8 and modify it with other shields if necessary in Line 9. This action is executed and the agent retrieves its next state and reward.

IV. Evaluation of Policy Correction

We evaluate the efficiency of our policy repair approach in three experiments. We describe our general RL setup, analysis plan, implementation, and results of these three experiments.

A. Reinforcement Learning Setup

Similar to [8], [30], we use the popular tabular Q-learning, allowing us to analyze how the policy considers our shields. We showcase our results in an adaptation of the 2D *Overcooked-AI* environment [31]. The videos of our experiments can be found in the supplementary material, our code is available at https://github.com/Sannevw/correctmehri2022.

In the action refinement and alternative item experiments, the objects are placed in a ring in the 3x3 grid environment and the agent is located in the center (see Fig. 5a and Fig. 6a). The agent can reach each location/item by rotating around its axis, starting in orientation 2 and each rotation in-/decreases it by one. In the salad experiment, sliced objects and the plate are placed at orientation 3 on the counter south of the agent, while non-sliced ingredients are placed at 2. In the cake example, all ingredients are placed at orientation 2. The agent can see and reach all objects in the kitchen. For the forbidden action experiment, we use a 4x5 grid world in which the agent can move freely (see Fig. 7a). For all tasks, we provide a step reward of -0.04, to favor shorter sequences. When action refinement is applied, we give one step cost even if the replacement consists of a sequence of actions.

B. Analysis Plan

We evaluate how well our shielded RL can recover from failures as compared to traditional RL when we allow little random exploration during retraining ($\epsilon = 0.1$) with the aim to repair policies without extensive retraining. We expect that by retraining with a shield from a correct NE feedback (ground-truth) obtained in an online study as described in Sec. V, the agent can learn a corrected policy more efficiently, which is beneficial in tasks with expensive data collection using a real robot or in computationally expensive simulations. The focus of this work is a proof-of-concept; refining the RL problem and (hyper)-parameters is left to future work. We also create shields from incorrect open answers (OAs) to gain insights into their effect on the failure recovery. For each experiment, we obtained only one incorrect OA in our study that contained actions available to the robot (e.g., fetch) and hence, could be implemented as a shield. We report the OA we used for the incorrect shield in each experiment in the following sections and in the supplementary material. OAs success and error rates are reported in Sec. V-D and shown in Fig. 9.

C. Make a Salad - Action Refinement

1) Problem Representation: The agent's goal is to assemble and deliver a salad, which consists of 1 tomato slice and 1 lettuce slice, served on 1 plate (see Fig. 5a). The actions are $\mathcal{A} = \{\mathrm{turn}^{\mathrm{cw}}, \mathrm{turn}^{\mathrm{ccw}}, \mathrm{fetch}^{<p>}, \mathrm{chop}^{<p>}, \mathrm{deliver}^{<p>}\}$, where cw and ccw denote turning clockwise and counterclockwise, respectively, and p is an object. The actions $\mathrm{fetch}^{<p>}, \mathrm{chop}^{<p>}, \mathrm{deliver}^{<p>}$ involve multiple substeps, but are counted as single timestep actions, e.g., $\mathrm{chop}^{<\mathrm{tomato}>}$ takes the tomato from the counter, chops it, and places it on the counter again. The state includes the *agent's orientation*, the *location of each item* (Shelf, Counter, or Delivery), and the *ingredients' state* (sliced or not). Correct deliveries are rewarded with $r = +1$ and incorrect deliveries with $r = -1$.

2) Failure: The tomato slice is no longer available in the deployment environment (see Fig. 5b) and the agent fails when executing $\mathrm{fetch}^{<\mathrm{tomatoSlice}>}$. To resolve the failure, the agent needs to fetch a whole tomato and chop it.

3) Results: Fig. 5c shows the rewards when retraining the policy with our shield, which refines the $\mathrm{fetch}^{<\mathrm{tomatoSlice}>}$ action by chopping a tomato when there is no slice, against the rewards gathered without a shield, and an incorrect shield. Our approach converges 30% faster on average and obtains less negative rewards (i.e., undesired outcomes) than without

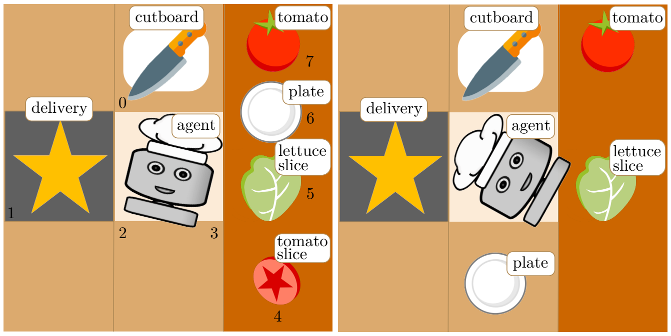
---

Algorithm 1 Correcting failures using NEs

Input: policy π, failure state s_F, trace $(a_1, \ldots, a_F)$, mapping function T, shields $(\nabla_{\text{ref}}, \nabla_{\text{alt}}, \nabla_{\text{all}})$

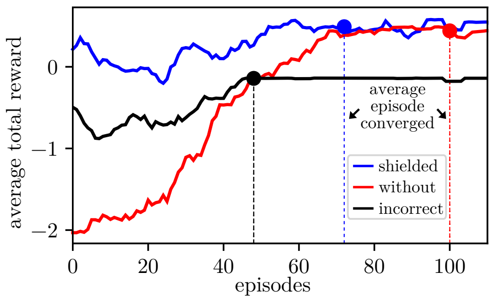
1: `// obtain feedback to correct failure`
2: $Q \leftarrow (s_F, (a_1, \ldots, a_F), T)$
3: $\mathcal{F} \leftarrow$ queryHumanFeedback(Q)
4: $(\nabla_{\text{ref}}, \nabla_{\text{alt}}, \nabla_{\text{all}}) \leftarrow$ updateShields($\mathcal{F}$)
5: `// retrain policy with shields`
6: $i \leftarrow 1$
7: **while** not converged **do**
8: $\quad a_i \leftarrow$ getAction(s_i, ∇_{all})
9: $\quad a_i \leftarrow$ shieldAction($s_i, a_i, \nabla_{\text{ref}}, \nabla_{\text{alt}}$)
10: $\quad s_{i+1}, r_i \leftarrow$ env.Step(a_i)
11: $\quad \pi_c \leftarrow$ trainPolicy(s_i, a_i, r_i, s_{i+1})
12: $\quad$ **if** Done **then**
13: $\quad\quad s_i \leftarrow$env.Reset()
14: $\quad i \leftarrow i + 1$

Return: corrected policy π_c



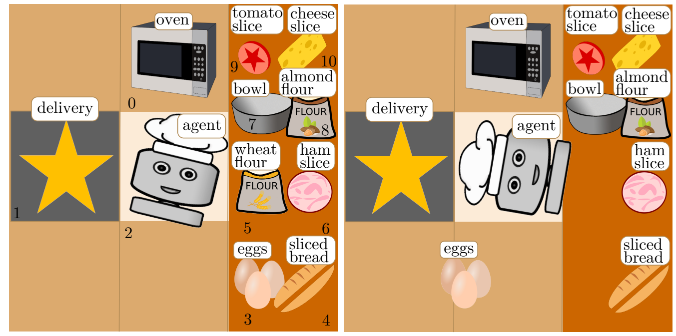
(a) Training environment. Numbers indicate the agent's orientations.

(b) Deployment environment.



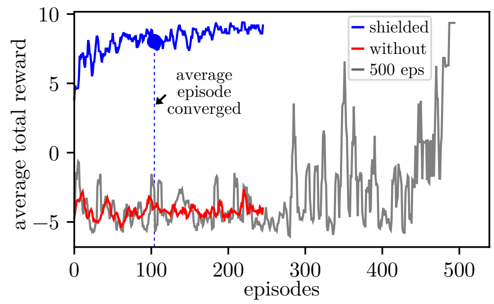
(c) Rewards with and without shielding.

Fig. 5. Action refinement: The agent needs to make a salad by fetching a tomato slice, a lettuce slice, and a plate from the shelf (dark brown), and deliver the dish from the counter (light brown) to the delivery. Fig. 5a and Fig. 5b depict the training and deployment environment, respectively. During deployment, the tomato slice is no longer readily available and the agent needs to refine its action of fetching a tomato slice to fetching a fresh tomato and chopping it into a slice. Fig. 5c shows the rewards of retraining the policy with correct, with incorrect, and without a shield to correct the failure.



(a) Training environment. Numbers indicate the agent's orientations.

(b) Deployment environment.



(c) Rewards with and without shielding.

Fig. 6. Alternative item example: The agent needs to bake a cake by fetching eggs, wheat flour, the bowl, baking the dough and delivering it from the counter (light brown) to the delivery. Fig. 6a illustrates the training environment and Fig. 6b illustrates the deployment environment in which there is no wheat flour available and the agent needs to replace it with almond flour as an alternative. Fig. 6c shows the rewards of retraining the policy with and without the shield to correct the failure.

shield. The incorrect shield, corresponding to open OA 13 in Fig. 2 in the supplementary material, refines the action by fetching a plate and then fetching a tomato slice. When the agent tries to fetch a tomato slice, it fails again, resulting in an infinite loop of corrections. The agent learned that activating the shield, which counts as one time step and ends the episode, is less costly than exploration over more time steps. Expert knowledge in the form of a large negative reward for the correction, prevented convergence to this sub-optimal solution.

D. *Bake a Cake - Alternative Item*

1) Problem Representation: The agent's goal is to make and deliver a cake, which consists of eggs and flour (see Fig. 6a). The actions are $\mathcal{A} = \{\text{turn}^{\text{cw}}, \text{turn}^{\text{ccw}}, \text{fetch}^{<p>}, \text{bake}^{<p>}, \text{deliver}^{<p>}\}$. The $\text{bake}^{<p>}$ action takes the bowl with its contents, bakes it in the oven, and places it on the counter again. Items can only be baked when they are in the bowl. The state is the same as in the salad experiment (see Section IV-C), but the ingredients' state denotes baked or not. Correct deliveries are rewarded with $r = +10$, and incorrect deliveries with $r = -1$. Baking the right ingredients is rewarded $r = +1$.

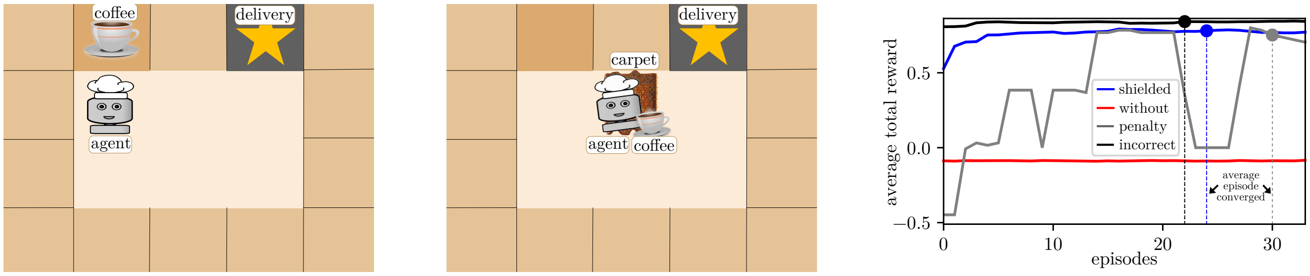
2) Failure: The wheat flour is no longer available in the deployment environment (see Fig. 6b), and the agent needs to learn to use almond flour as an alternative.

3) Results: Fig. 6c shows the rewards when retraining the policy with our generated shield, which replaces $\text{fetch}^{<\text{wheatFlour}>}$ with $\text{fetch}^{<\text{almondFlour}>}$, and without a shield. Our policy repair approach learns the task successfully and converges after 104 episodes on average. Without a shield, the policy is unable to robustly learn the task or obtain positive rewards, even when trained for 500 episodes instead of 250. Our policy repair approach avoids the failure while being more data-efficient. The incorrect shield, corresponding to OA 13 in Fig. 4 in the supplementary material, refines the action by fetching a bowl, fetching fresh eggs, and fetching wheat flour. Similar to the salad experiment, this correction results in an infinite corrective loop, as the shield calls the failed action.

E. *Deliver Coffee - Forbidden Actions*

1) Problem Representation: The agent has to pick up and deliver coffee (see Fig. 7a). The actions are $\mathcal{A} = \{\text{move}^{\text{left,right,up,down}}, \text{pickup}^{<p>}, \text{putDown}^{<p>}\text{deliver}^{<p>}\}$. Picking and placing items on counters is done by moving to the corresponding tile, e.g., in Fig. 7a, the agent picks up the coffee by moving up. The state includes the location of the *agent* and *coffee*. Correct delivery is rewarded with $r = +1$.

2) Failure: There is a carpeted floor tile in the deployment environment (see Fig. 7b), and the agent fails because it gets stuck on it. The agent needs to learn to avoid carpeted tiles.



(a) Training environment.

(b) Deployment environment.

(c) Rewards with, without shielding and penalty.

Fig. 7. Forbidden actions example: The agent needs to pickup the coffee from the counter and deliver it at the delivery. Fig. 7a illustrates the training environment and Fig. 7b illustrates the deployment environment which contains carpeted floor tiles, on which the agent gets stuck, and it needs to avoid actions that lead it to the carpet. Fig. 7c shows the rewards of retraining the policy with and without the shield to correct the failure, as well as the reward of a penalty version that required manual intervention and the use of an incorrect shield.

3) Results: Fig. 7c shows the rewards when retraining the policy with our shield, which removes actions that move the agent to carpeted floor tiles, without a shield, and with an incorrect shield. Our policy repair approach learns the task successfully and converges after 24 episodes on average. Without a shield, the policy does not learn the task and only obtains negative rewards. The agent repeatedly ends up in the carpeted floor tile, since it does not get any additional penalty reward but only the negative step cost. We modified the reward function by providing an additional penalty of $r = -1$ for moving to carpeted tiles. This change helped the policy to converge, yet it still requires 25% more episodes compared to our shielded learning. In addition, it has a less stable learning curve, which indicates more undesirable actions during training. The incorrect shield, corresponding to open OA 7 in Fig. 6 in the supplementary material, corrects the action by moving the agent one tile down, one right, one up. The corrective action sequence gets triggered twice consecutively, as the first correction made the agent move into the carpet again. The agent learns that with activating the shield, which counts as one time step, it ends up at the delivery tile, which is less costly than exploration over multiple steps.

V. Collect Non-expert Input to Repair Policies

Our goal is to see 1) if NEs can understand what caused a failure and 2) if they can provide rules to the robot to correct its policy. We use natural language as an intuitive way for inexperienced users to instruct the robot [32], it allows us to analyze whether people understand the task, it reduces the risk of having the interface design as a confounding factor on task performance, and it can be translated into code [33]. There is a large body of work that focuses on NE robot programming, e.g., [34]–[38]. While that is not our focus, we did ask people to code their solution using the Google Blockly interface [39], to get an idea how this compares to natural language.

A. Study Design

We iteratively designed our online study using data from 18 participants (5 female, 13 male) between 20–63 years of age (M = 31.8, SD = 8.86), recruited from Amazon Mechanical Turk (AMT). All participants had little to no prior experience with programming (M = 1.94, SD = 0.73; 1 = none, 2 = beginner, 3 = intermediate, and 4 = advanced) and with robots (M = 2.33, SD = 0.49; 1 = never seen a robot in media or real life, to 4 = interact with robots on a regular basis). 15 participants were American, 1 German, 1 Irish, 1 Brazilian. They were compensated $7USD.

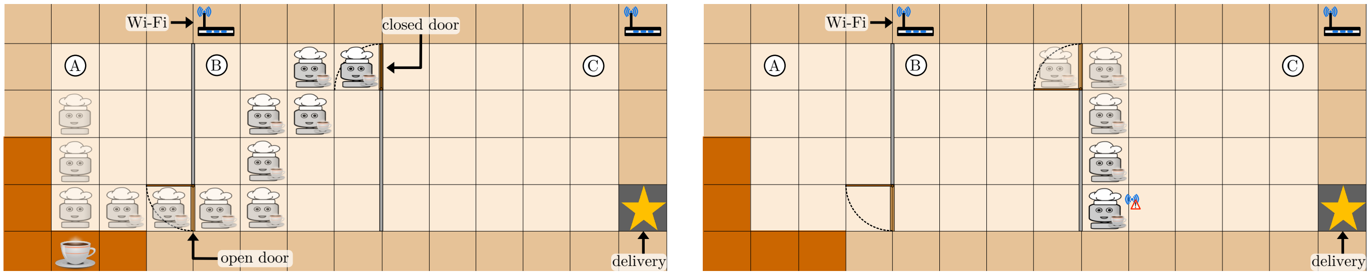
1) Results and Final Design: The final online study uses a within-subjects (task: salad, cake, coffee) design. For each task, we asked participants to help the robot using natural language in two open questions, and then using visual programming blocks. There are six combinations in total, and we counterbalanced with which order participants started.

We refined the task instructions, we added an animation of the robot getting stuck on the carpet in the coffee task, because the failure was hard to understand from a static 2D image. Lastly, we added an introduction page with example rules: *If a door is closed, open the door*, and *never move out of the WiFi signal's range* (see Fig. 8).

B. Participants and Procedure

Exclusion of one non-serious submission resulted in a total of 28 participants recruited through AMT (10 female, 18 male) between 27–57 years of age (M = 38.68, SD = 7.68). They took 28 minutes on average (SD = 10 minutes) and were compensated $7USD. They had little to no programming experience (M = 1.89, SD = 0.92; 1 = none, 2 = beginner, 3 = intermediate, and 4 = advanced). Nine participants had interacted with a robot before (e.g., vacuum cleaning robots or office/home robots) and one participant used to work with assembly line robots. 23 participants were American, 4 Indian, 1 German. Six participants completed vocational qualification, 8 participants completed secondary education, 11 participants completed bachelor level education, and three completed master level education.

Participants first completed an introduction phase with the examples as shown in Fig. 8. Then, for each task, they first answered the following two open questions: 1. *"Why did the robot fail to fetch the tomato slice in this kitchen?"* and 2. *"In this kitchen, what rule can the robot use to avoid this type of failure and successfully complete this task on future occasions?"*. We asked participants to specify a general 'rule'



(a) Door Failure

(b) Wi-Fi Failure

Fig. 8. Two examples of robot failures. In Fig. 8a, the robot fails to pass through a closed door while delivering coffee, since it was trained in an environment with open doors only. In Fig. 8b, the robot fails because it left the Wi-Fi signal's range, since it was trained in an environment with different router positions.

that defines what the robot could do to avoid the failure. Then, we explained to the participants that robots do not understand natural language really well yet, and we asked them to create a program that helps the robot to complete the task in the future using a Blockly interface. Finally, we collected their demographics and thanked them for their participation.

C. Can Non-Experts Understand Failures?

Two of the authors coded the open answers (OAs) as (in)correct, with inter-rater agreement of 94% for the failures understood, and 98% for the rules. The OAs can be found in the supplementary material. Most participants understood the failures correctly (see Fig. 9). 17 people (61%) understood all three failures correctly, 5 people (18%) understood two failures correctly, four people (14%) understood one failure correctly, and two people (7%) did not understand any failure.

For the salad task, 22 participants understood the failure correctly (79%) and six participants (21%) did not. An example of a correctly identified failure is *"You failed to fetch the tomato slice; because there are only whole tomatoes on the shelf"*. For the cake task, 20 participants (71%) correctly understood the failure, e.g., *"It knew to fetch wheat flour but didn't understand that almond flour is an acceptable substitute"*. Eight participants (29%) did not understand the failure correctly. For the coffee task, 23 participants (82%) identified the failure correctly, e.g., *"You failed to move right because there was a shag carpet on the floor and you became stuck"*, and 5 participants (18%) did not.

D. Can Correct and Incorrect Feedback Be Differentiated?

All three failures could successfully be corrected by the majority of participants using natural language: 71% of the participants succeeded in the salad experiment, 61% in the cake experiment, and 61% in the coffee experiment (see Fig. 9). However, even participants whose OAs were correct, struggled to code rules using the visual interface, suggesting that natural language is more intuitive for NEs.

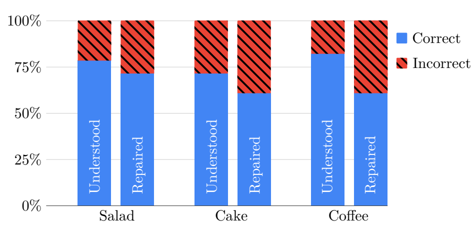


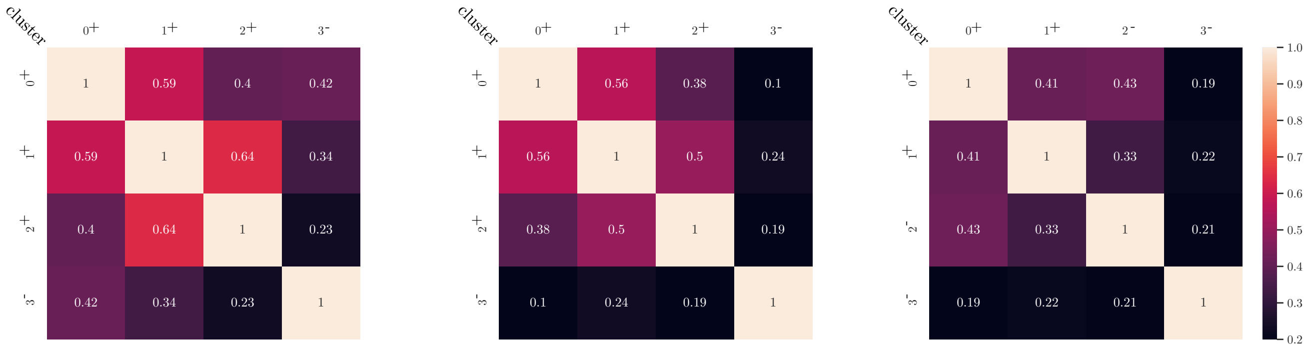
Fig. 9. Percentage correctly identified and repaired failures per experiment.

We analyzed the similarities between OAs to shed light on the potential to automatically process NE feedback into shields. We study 1) whether we can distinguish (in)correct answers, and 2) the similarity between correct OAs.

We pre-processed the OAs by correcting spelling mistakes, removing upper-case and punctuation, and converted each OA into a feature vector of token counts considering 1-gram words using scikit-learn [40]. For each OA, we computed cosine similarity with the other OAs (a score of 0 indicates no similarity, and a score of 1 indicates complete similarity), and grouped similar OAs using agglomerative clustering.

Fig. 10 shows the pair-wise similarity scores between the computed clusters in each experiment. Generally, we observe that correct answers clusters have higher similarity to each other, and the lowest similarity with the incorrect answers. The average pair-wise similarity score between correct and incorrect answers was 43%, 27%, and 38% in the salad, cake, and coffee experiment, respectively, which suggests that we can differentiate between correct and incorrect OAs. 83% of OAs were clustered correctly: 52 out of 55 correct answers and 18 out of 29 incorrect answers. There were one and two correct answers clustered as incorrect for the salad and coffee experiment, respectively. No correct answers were clustered as incorrect for the cake experiment. Three incorrect answers were clustered as correct for the salad and cake experiment, and five for the coffee experiment. We found a high average similarity between correct answers, 65%, 60%, and 60%, for the salad, cake, and coffee experiment, respectively. The incorrect OAs had lower average similarity, 30%, 25%, and 39% for the salad, cake, and coffee experiment, respectively. This suggests that correct answers tend to be consistent across people and incorrect answers can be identified based on their dissimilarity with correct answers.

Within the correct answers clusters, dissimilarities mainly stem from the way participants describe actions, e.g., *"slice the tomato"* versus *"if the tomato is not cut, cut a slice"* or e.g., *"fetch almond flour"* versus *"use an alternative flour"*. The two incorrect clusters for the coffee experiment stem from people who copied parts of the instructions as their answer.



(a) Salad experiment (b) Cake experiment (c) Coffee experiment

Fig. 10. Similarity scores of the obtained open-answer clusters in our three experiments. Similarity values 0 and 1 denote no and full similarity, respectively. Cluster indices with a + correspond to answers clustered as correct and indices with a − correspond to answers clustered as incorrect.

VI. Discussion and Conclusions

This work is a first step towards a full NE policy repair pipeline and validates for the first time that shields can be used to repair high-level policies after failures, so that robots can continue their task without constant supervision, even if the environment changes. The experiments show that simple shields can drastically improve the retraining by converging faster and thus requiring less training data. Shielded exploration learns more robustly, avoiding undesired outcomes. While expert intervention (e.g., assigning explicit rewards) helped to retrain policies, our shielded approach was successful without the need for such intervention. While these results are promising, we acknowledge important avenues for future work.

A. Dealing with Inaccurate Feedback

The majority of NEs could successfully understand the failure (77% on average) and suggest policy repairs (67% on average), indicating that methods such as majority voting can be used to select correct feedback. 83% of OAs were correctly clustered, with correct answers generally being highly similar to each other but not to incorrect answers. While these results highlight the potential to automate shield generation, similarity clustering only suggests which answers are likely to be (in)correct. More work is needed to automatically filter out inaccurate feedback [41]. For example, multiple NEs can be asked to judge whether corrections are sensical before sending them to the robot [42], or answers could be filtered by whether they contain actions not currently available to the robot, which we observed in our results, to see if these actions are nonsensical or if they should be added to the robot's action set.

Robots need to be able to automatically detect inaccurate feedback to query additional help. In the salad and cake experiments, the incorrect shields resulted in an infinite loop of corrections. This can happen for learning-based robots that interact with the environment through sparse rewards while favoring shorter policies. To detect such behaviors of incorrect shields, one can incorporate performance measures to evaluate how effective the corrections are. For instance, one may keep track of the number of consecutive shield corrections, as this indicates that the agent repeatedly visited undesired states.

In the coffee experiment, the incorrect shield's policy converged the fastest with a higher reward, but it only holds for this specific layout, e.g., a larger environment might result in the agent moving further away from its goal. The correct shield generally avoids carpets in any environment layout. NEs who see just one failure instance, might provide instance-specific corrections. One solution could be to run a benchmark suite of scenarios as a sanity check for how well the shield generalizes.

B. Broader Applicability of our Approach

Our approach is applicable to sequential decision-making robots that perform common tasks, e.g., a robot in a table-setting task might wrongly place objects. NEs can specify rules to define desired relations between the objects over time (e.g., always place the fork left of the plate) to ensure the correct target task configuration. Robots can use on-the-fly NE feedback to synthesize controllers to adhere to such norms.

When testing the approach with real robots, we can leverage sensor fusion to provide richer representation of the environment and failure in the robot's queries and test what information representations help NEs repair the robot's policies effectively. Our approach works for lower-level skills, if they can be decomposed into higher-level actions. For example, it is difficult for NEs to suggest corrections for a control input trajectory to fetch a slice of bread. However, decomposing this trajectory into actions, such as 'move gripper to bread' and 'close gripper', can be semantically interpreted by NEs. Future work needs to investigate the trade-off between correcting lower-level skills and simplicity for NEs.

We implemented a proof-of-concept using tabular Q-learning, but we want to test our approach with more complex learning algorithms and policies outside RL. Finally, our approach can complement work on failure explainability [43], e.g., to use OAs for explaining failures to users.

In this work, we showed that shields can be used to repair failures of learning-based robots and NEs are able to provide corrections from which such shields could be generated. Overall, this paper showcases the potential for multidisciplinary research bringing together approaches from human-robot interaction, learning, and formal methods [44].

REFERENCES

[1] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The Int. Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[2] E. Guizzo and E. Ackerman, "The hard lessons of DARPA's robotics challenge [news]," *IEEE Spectrum*, vol. 52, no. 8, pp. 11–13, 2015.

[3] D. Hadfield-Menell, S. Milli, P. Abbeel, S. Russell, and A. Dragan, "Inverse reward design," *arXiv preprint arXiv:1711.02827*, 2017.

[4] R. Shah, C. Wild, S. H. Wang, N. Alex, B. Houghton, W. Guss, S. Mohanty, A. Kanervisto, S. Milani, N. Topin *et al.*, "The MineRL BASALT competition on learning from human feedback," *arXiv preprint arXiv:2107.01969*, 2021.

[5] D. Porfirio, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Authoring and verifying human-robot interactions," in *Proc. of the ACM Symposium on User Interface Software and Technology*, 2018, pp. 75–86.

[6] E. Senft, M. Hagenow, K. Welsh, R. Radwin, M. Zinn, M. Gleicher, and B. Mutlu, "Task-level authoring for remote robot teleoperation," *arXiv preprint arXiv:2109.02301*, 2021.

[7] N. Wilde, A. Blidaru, S. L. Smith, and D. Kulić, "Improving user specifications for robot behavior through active preference learning: Framework and evaluation," *The Int. Journal of Robotics Research*, vol. 39, no. 6, pp. 651–667, 2020.

[8] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.

[9] ——, "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *Aaai*, vol. 6, 2006, pp. 1000–1005.

[10] T. Cederborg, I. Grover, C. L. Isbell, and A. L. Thomaz, "Policy shaping with human teachers," in *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 2015.

[11] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *Advances in neural information processing systems*, 2013, pp. 2625–2633.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[13] S. Chernova and A. L. Thomaz, "Robot learning from human teachers," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[14] A. L. Thomaz and C. Breazeal, "Asymmetric interpretations of positive and negative human feedback for a social learning agent," in *Proc. of the IEEE Int. Symposium on Robot and Human Interactive Communication*. IEEE, 2007, pp. 720–725.

[15] W. B. Knox, P. Stone, and C. Breazeal, "Training a robot via human feedback: A case study," in *Int. Conf. on Social Robotics*. Springer, 2013, pp. 460–470.

[16] H. B. Suay and S. Chernova, "Effect of human guidance and state space size on interactive reinforcement learning," in *Proc. of the IEEE Int. Symp. on Robot and Human Interactive Communication*, 2011, pp. 1–6.

[17] E. Senft, P. Baxter, J. Kennedy, S. Lemaignan, and T. Belpaeme, "Supervised autonomy for online learning in human-robot interaction," *Pattern Recognition Letters*, vol. 99, pp. 77–86, 2017.

[18] D. Koert, M. Kircher, V. Salikutluk, C. D'Eramo, and J. Peters, "Multi-channel interactive reinforcement learning for sequential tasks," *Frontiers in Robotics and AI*, vol. 7, 2020.

[19] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Int. Conf. on Machine Learning*, 2017, pp. 22–31.

[20] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[21] A. Wachi and Y. Sui, "Safe reinforcement learning in constrained Markov decision processes," in *Int. Conf. on Machine Learning*, 2020, pp. 9797–9806.

[22] H. Kress-Gazit, K. Eder, G. Hoffman, H. Admoni, B. Argall, R. Ehlers, C. Heckman, N. Jansen, R. Knepper, and J. Křetínskỳ, "Formalizing and guaranteeing human-robot interaction," *arXiv preprint arXiv:2006.16732*, 2020.

[23] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[24] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[25] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained reinforcement learning," *arXiv preprint arXiv:1801.08099*, 2018.

[26] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proc. of the Int. Conf. on Autonomous Agents and MultiAgent Systems*, 2018, pp. 452–461.

[27] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Int. Conf. on Artificial Intelligence*, 2018.

[28] N. Jansen, B. Könighofer, S. Junges, and R. Bloem, "Shielded decision-making in MDPs," *arXiv preprint arXiv:1807.06096*, 2018.

[29] B. Könighofer, J. Rudolf, A. Palmisano, M. Tappler, and R. Bloem, "Online shielding for stochastic systems," in *NASA Formal Methods Symposium*, 2021, pp. 231–248.

[30] S. A. Raza and M.-A. Williams, "Human feedback as action assignment in interactive reinforcement learning," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 14, no. 4, pp. 1–24, 2020.

[31] R. E. Wang, S. A. Wu, J. A. Evans, J. B. Tenenbaum, D. C. Parkes, and M. Kleiman-Weiner, "Too many cooks: Coordinating multi-agent collaboration through inverse planning," *arXiv preprint arXiv:2003.11778*, 2020.

[32] J. Y. Chai, M. Cakmak, C. Sidner, and J. Lupp, "Teaching robots new tasks through natural interaction," in *Interactive Task Learning: Agents, Robots, and Humans Acquiring New Tasks through Natural Interactions, Strüngmann Forum Reports, J. Lupp, series editor*, vol. 26, 2017.

[33] W. Zaremba, G. Brockman, and OpenAI. (2021) OpenAI codex. [Online]. Available: https://openai.com/blog/openai-codex/

[34] E. M. Orendt, M. Fichtner, and D. Henrich, "Robot programming by non-experts: Intuitiveness and robustness of one-shot robot programming," in *Proc. of the IEEE Int. Symposium on Robot and Human Interactive Communication*, 2016, pp. 192–199.

[35] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 5537–5544.

[36] S. Elliott, R. Toris, and M. Cakmak, "Efficient programming of manipulation tasks by demonstration and adaptation," in *Proc. of the IEEE Int. Symposium on Robot and Human Interactive Communication*. IEEE, 2017, pp. 1146–1153.

[37] M. Stenmark, M. Haage, and E. A. Topp, "Simplified programming of re-usable skills on a safe industrial robot: Prototype and evaluation," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 2017, pp. 463–472.

[38] S. van Waveren, E. J. Carter, O. Örnberg, and I. Leite, "Exploring non-expert robot programming through crowdsourcing," *Frontiers in Robotics and AI*, p. 242, 2021.

[39] Google. (2021) Blockly. [Online]. Available: https://developers.google.com/blockly

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[41] T. A. K. Faulkner, E. S. Short, and A. L. Thomaz, "Interactive reinforcement learning with inaccurate feedback," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2020, pp. 7498–7504.

[42] I. Leite, A. Pereira, A. Funkhouser, B. Li, and J. F. Lehman, "Semi-situated learning of verbal and nonverbal content for repeated human-robot interaction," in *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, 2016, pp. 13–20.

[43] D. Das, S. Banerjee, and S. Chernova, "Explainable ai for robot failures: Generating explanations that improve user assistance in fault recovery," *arXiv preprint arXiv:2101.01625*, 2021.

[44] D. Kragic and Y. Sandamirskaya, "Effective and natural human-robot interaction requires multidisciplinary research," *Science Robotics*, vol. 6, no. 58, p. eabl7022, 2021.