Enabling End Users to Program Robots Using Reinforcement Learning

Tewodros W. Ayalew*
University of Chicago
tewodrosayalew@uchicago.edu

Jennifer Wang*
Brown University
jennifer_wang2@brown.edu

Michael L. Littman
Brown University
mlittman@cs.brown.edu

Blase Ur
University of Chicago
blase@uchicago.edu

Sarah Sebo
University of Chicago
sarahsebo@uchicago.edu

***Abstract*—Reinforcement learning (RL) is a powerful learning technique in robotics, where people can specify rewards that robots learn how to maximize through a process of trial-and-error. Despite the numerous advantages of RL to robot programming, no approaches to our knowledge have sought to enable *non-technical users* to specify RL programs for robots. In this work, we designed two novel RL-based robot programming paradigms for non-technical users: *Full MDP Programming* (*Full-MDP*) and *Goal-Only MDP Programming* (*Goal-MDP*). To evaluate the efficacy of these two approaches, we ran a between-subjects online user study ($N = 409$) where participants were asked to program a simulated robot to complete example household tasks (e.g., delivering coffee) using one of our RL programming paradigms or a commonly used baseline: *Sequential Programming* (*Seq*), or *Trigger-Action Programming* (*TAP*). While users neither performed well nor reported positive experiences with the *Full-MDP* interface, user performance and experience with *Goal-MDP* was similar to the baselines (*Seq* and *TAP*) with significantly shorter programs. These results demonstrate that RL-based paradigms like *Goal-MDP* are a viable alternative to more traditional approaches and provide a starting point for robot programming interfaces that allow end-users to leverage the myriad benefits of RL for programming robots.**

***Index Terms*—End-User Robot Programming, Reinforcement Learning, Human-Robot Interaction**

I. Introduction

Robots have the potential to improve the lives of a broad range of people, serving as tutors to elementary school children [1]–[4], providing care to older adults [5]–[8], delivering supplies [9]–[11], and helping with household tasks [12]–[14]. In these settings, it is essential that users can directly program robots to meet their needs, adhere to their preferences, and customize their behavior to the user's specific context.

Various existing methods let end users customize robot behavior [15]–[25]. However, to our knowledge no current methods allow for learning-based approaches where a robot discovers how to satisfy user preferences through either planning or an empirical process based on information specified by end users, such as **reinforcement learning (RL)**. In traditional uses of RL [26], an expert would specify a **Markov decision process (MDP)** representing an environment with an agent (e.g., robot) in it. The agent would use this specification algorithmically through exploration and/or optimization to learn a policy specifying what actions the robot should take.

In this work, we introduce the notion of enabling **non-technical end users** to specify the most important parts of an MDP via a visual programming approach suitable for novices, leveraging RL to learn the actions their robot should take based on these specifications. Such an approach could have several benefits. Rather than specifying a long list of actions for robots to execute in sequence, RL could enable users to focus solely on specifying desired outcomes. Additionally, the robot could use a reinforcement-learning approach to determine the most effective plan of action to accomplish the goal. This kind of specification may take less time for the user and enable the robot to determine a more effective solution.

We designed and implemented two novel end-user interfaces where users specify MDPs. The first is the ***Full MDP Programming* (*Full-MDP*)** user interface, where an end user specifies the main components of an MDP: the goal(s) (*rewards*) they want the robot to achieve, possible behaviors (*actions*) for the robot to try, and key attributes (*states*) of the environment. We also designed a second novel interface ***Goal-Only MDP Programming* (*Goal-MDP*)**, in which the end user only specifies the robot's goals, and all states and actions are automatically included to compute the robot's policy. We automatically translate the user's specification to a traditional MDP and use a reinforcement learning approach to determine the most effective policy for the robot to accomplish the goal.

To gauge whether our novel end-user RL approach was successful and to understand its specific pros and cons compared to traditional end-user programming techniques for robots, we sought to answer the following research questions:

- **RQ1:** *Using an RL end-user paradigm for programming a robot, (a) can users successfully author programs and (b) do they report a positive user experience?*
- **RQ2:** *How do RL end-user robot programming paradigms compare with traditional approaches in (a) user's ability to author correct programs, (b) the time required, (c) program length, and (d) user experience?*

* Tewodros W. Ayalew and Jennifer Wang contributed equally to this paper.

979-8-3503-7893-1/25/$31.00 ©2025 IEEE

- **RQ3:** *How do RL end-user robot programming paradigms compare with traditional approaches based on specific features of the programming task (e.g., handling repetitive actions, uncertain initial conditions)?*

We answered these research questions via a between-subjects online study in which 409 participants were asked to program a simulated robot to complete a variety of household tasks (e.g., delivering coffee) using a randomly assigned programming paradigm. In addition to our *Full-MDP* and *Goal-MDP* paradigms, we also implemented and tested two baseline paradigms: *Sequential Programming* (*Seq*), commonly used for visual end-user programming in robotics, and *Trigger-Action Programming* (*TAP*), a common approach in other domains (e.g., smart homes), though less common in robotics.

II. Related Work

A. End-User Robot Programming

To enable a broad range of end users to customize robots, researchers have developed a variety of end-user programming paradigms [15]. Visual programming, in which users manipulate their graphical representation, is common. These graphical representations include behavior trees [27]–[29], blocks [23], [30], [31], and icons [25], [32]. These interfaces often include elements like buttons, sliders, and input forms [33]–[36].

Researchers have explored several other approaches. Robot learning from demonstration (LfD), also known as imitation learning, can teach robots skills without the need for end users to explicitly program them [37]–[41] and can also be incorporated into explicit robot programs to simplify the specification of more complex behaviors [19], [23], [42]. Augmented reality (AR) and mixed reality (MR) interfaces enable users to minimize context switching by embedding into their environment an interface supporting gestures, touch, and direct interaction [16]–[18], [43]. Speech and natural language are sometimes incorporated into end-user robot programming, most often in combination with other modalities [18], [22], [44]–[46]. Finally, tangible programming involves the physical manipulation of real-world objects [24], [25], [32], [47], [48] and is most often designed for use by children [25], [32].

This work focuses on situations where a user can specify an entire program for tasks that robots complete in everyday environments (e.g., the home). Given its past successes in enabling users to author complete robot programs, we selected a block-based visual programming end-user interface.

B. Sequential Programming

We define *Sequential Programming* as a programming approach where users specify step-by-step instructions for execution (i.e., imperative programming). The most common form of sequential programming for non-technical end-users in both robotics and non-robotics contexts are block-based programming interfaces where users drag and drop blocks representing program instructions that are executed in sequence. These block-based sequential programming interfaces are the most widely recognized and successful interfaces for non-technical users [15], [31], [49], [50]. In robotics, end users have successfully used block-based sequential programming interfaces to author robot programs that pick and place objects in industrial settings [31], [51], deliver items to rooms in hotels [49], grasp items in everyday environments [23], and express emotions through sound and movement [30]. Block-based sequential interfaces are also commonly used to teach children how to program robots (e.g., Dash, Cozmo, Sphero).

C. Trigger-Action Programming (TAP)

In recent years, *Trigger-Action Programming* (*TAP*) has become a common form of end-user programming in academic literature and commercial services like IFTTT and Zapier [52], [53]. In TAP, users create a set of event-driven rules, typically following an if-then format, using a GUI. TAP has seen particular adoption for Internet of Things (IoT) devices and online services [52], [54], but TAP-like systems have also been used for building automation [55], email processing [56], and recently for robotics [57]–[59]. In the IoT context, prior work has reported that end users can successfully write TAP rules regardless of programming experience [60], [61]. To help users overcome common bugs [62], [63] and ensure program correctness, researchers have also developed debugging tools [64]–[66], paired TAP systems with formal models [67]–[70], generated recommendations [71]–[73], and synthesized TAP programs [74]–[77].

In the robotics context, Leonardi et al. [57] designed a TAP system for the Pepper humanoid robot, where most participants successfully wrote TAP rules on the first try. Senft et al. [58] successfully extended TAP to support situated live programming of collaborative robots. Most recently, Ikeda and Szafir [59] integrated TAP with AR, finding that doing so improves the experience of programming a collaborative robot.

D. Markov Decision Process (MDP) Programming

A Markov decision process [78], or MDP, consists of a set of states, a set of actions, a transition function dictating the probability of moving between any pair of states under the influence of an action, a discount factor, and a reward function assigning a numerical value for being in each state. An optimal policy in an MDP consists of a mapping from states to actions that maximizes the cumulative expected discounted reward.

The problem of reinforcement learning [26], or RL, is often formalized as an agent learning to behave optimally in an MDP given knowledge of the states, actions, and rewards, but only indirect access to the transition function via observing the transitions made in its environment.

Prior work in robotics and human-robot interaction (HRI) have utilized RL approaches to both choosing reward functions and providing reward-like feedback to robots to produce desired robot behavior. Several researchers have studied optimizing reward functions to solve specific tasks [79], [80] and the expressiveness of reward functions [81], as well as ways of making reward functions more expressive to capture a broader class of tasks [82]–[84]. Additionally, others have demonstrated success in allowing end users to provide reward-like feedback to RL agents for bringing about desirable

behaviors [85], [86]. Porfirio et al. [87] introduced Polaris, an end-user programming (EUP) system that stands out by using goal predicates as the core building blocks for programming. Polaris allows users to specify both high-level robot objectives and detailed checkpoints, with an integrated task planner filling in the remaining details. Nonetheless, we are not aware of any prior attempts to let end users directly specify an MDP.

III. Enabling End Users to Specify RL Problems

Currently, RL problems are specified by experts constructing an MDP using a traditional programming language. While that approach makes sense for highly technical tasks, our goal was instead to empower non-technical end users to leverage RL's benefits for customization and personalization. For instance, we imagine that an end user with a domestic robot like the TIAGo [88] or Stretch 3 [89] mobile manipulator robots would need to specify exactly *how* their robot, which already has basic navigation and manipulation capabilities, should support them in their particular environment.

A. Key Challenges and Design Decisions

Enabling end users for the first time to specify RL problems required us to overcome four key challenges. First, as non-technical end users do not know traditional programming languages, we needed to create an appropriate user interface for specifying MDPs. We did so through an iterative design process based on pilot testing and used as inspiration the existing block-based languages (e.g., Scratch [50]) often used for novice programmers. The blocks we designed allowed users to specify the three key components of an MDP: the *states* describing the environment and RL agent, the *actions* an agent can take in exploration, and the *rewards* for reaching particular states. Our choice of the primitives for these blocks built on the widespread trigger-action programming paradigm [60].

The second key challenge is reward specification. Typically, an expert user specifying an MDP would assign numerical values to specific state transitions. However, as even experts struggle to assign appropriate rewards [90], we worried that end users would be overwhelmed and struggle to assign numbers. In response, we reframed rewards as goals and had users assign each goal to one of three priority levels—high, medium, and low. This approach reduces the complexity for users while sufficing for domestic customization, where a user typically has a small number of primary goals and perhaps some secondary goals.

The third key challenge is *comprehensive* enumeration of task-relevant states and actions. If any are missed, the RL process might not find an optimal or feasible solution. Thus, in addition to *Full MDP Programming*, we also designed *Goal-Only MDP Programming* in which end users would *not* specify states or actions, but only goals. The RL algorithm considers *all possible states* and tries *all possible actions*. While this increases the computational complexity for search, for domestic robots the number of possible states and actions is often fairly small. Therefore, the benefits of lower complexity in specification would outweigh the added computational costs.

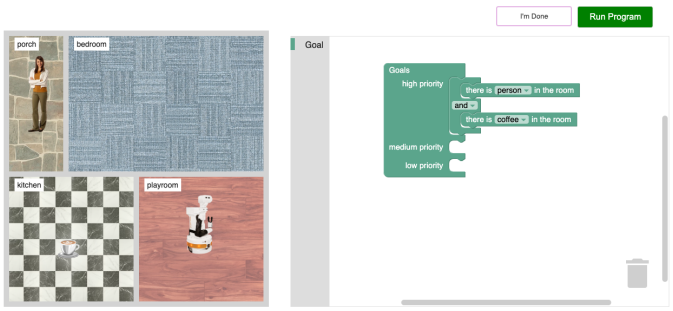


Fig. 1: We designed a graphical interface based on Blockly in which end users program a robot to complete various household tasks. Here, the user is programming the robot to deliver coffee to them (Task 4) using the *Goal-MDP* paradigm.

The fourth challenge is non-technical end user understanding of the learned policy. While a technical specification of a policy might suffice for an expert, it would fall short for end users. To increase user understanding, we displayed visual simulations (on the left side of Figure 1) of the robot following the learned policy from randomized starting conditions.[1]

B. Choice of RL Algorithm and Performance

After eliciting the necessary parts of the MDP from the user, we automatically construct a more traditional MDP. For both *Full-MDP* and *Goal-MDP*, all states within the state space meeting the user-specified goal condition are assigned the reward value (1, 3, or 5) corresponding to the user-specified priority level. For *Goal-MDP*, we automatically include all possible states and actions in our representation of the MDP.

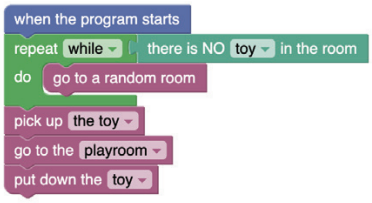
We benchmarked policy generation on a commodity laptop by running value iteration for 20 episodes with a discount factor $\gamma = 0.92$ on all final specifications (correct and incorrect) submitted by participants for all tasks in our user study. For *Full-MDP*, the median time to generate policies was 0.008 seconds (Q_1: 0.003 seconds, Q_3: 0.070 seconds). For *Goal-MDP*, the median time was 0.047 seconds (Q_1: 0.005 seconds, Q_3: 0.649 seconds). Notably, even with *Goal-MDP*'s increases to the state and action spaces, policy generation was fast enough to minimally impact the user experience. In environments with more states/actions, faster methods—like using deep networks [91]—could be considered.

IV. User Interface Design

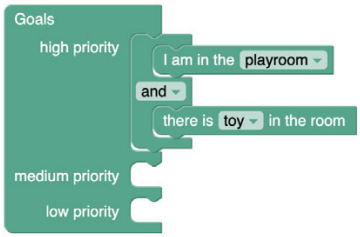
We designed and implemented an interface for end users to program a simulated TIAGo mobile manipulator robot to complete tasks in a home. The simulated home has four rooms: kitchen, bedroom, playroom, and porch (see Figure 1). Many tasks also include a person and everyday objects, such as toys or mail. We have open-sourced our code [92].

Our interface builds on Blockly [93]. Users can drag and drop blocks representing robot actions, environment attributes,

[1]To avoid confounds, in our user study we showed analogous visual simulations for all paradigms.



(a) *Sequential Programming*



(d) *Goal-Only MDP Programming*

(b) *Trigger-Action Programming*

(c) *Full MDP Programming*

Fig. 2: Example solutions in each end-user programming paradigm for *Task 2: Toy in Random Room*, where the user programs the robot to pick up a toy (that can start in any room) and move it to the playroom.

and programming structure (e.g., loops). Block-based interfaces, which are commonly adopted in trigger-action programming, are powerful because they let end users customize their robots according to their preferences. The four programming paradigms we designed share two common high-level categories of blocks. First, **state attributes** represent the current condition of the robot and the environment (e.g., "I am in the kitchen"). We assumed the robot only perceives states it can sense within the same room. Moreover, the robot does not retain the state of the environment from previous time steps. Second, **actions** are specific behaviors that the robot can perform (e.g., "pick up the coffee").

A. Sequential Programming (Seq)

In the *Sequential Programming* (*Seq*) paradigm, users specify a sequence of actions that are executed one after the other, beginning once the program starts. As shown in Figure 2a, action blocks are thus the focus. Note that action blocks may have parameters (e.g., "playroom" in "go to the [playroom]" set via drop-down menus. It also includes *control structures*, specifically loops ("repeat while/until") and conditionals.

B. Trigger-Action Programming (TAP)

In *Trigger-Action Programming* (*TAP*), users create an unordered set of event-driven, if-then rules. When a rule's triggering event occurs, that action is taken. Rules can take either of the following forms: "if <event> then <action>" or "if <event> and <state(s)> then <action>." In the latter, when the event occurs, the system checks whether the specified state (or states, connected with a Boolean "and") is true. Triggering events often represent changes in the robot's environment.

C. Full MDP Programming (Full-MDP)

In *Full MDP Programming* (*Full-MDP*), users specify sets of blocks for each MDP component: "actions," "states," and "goals." These components are translated as inputs to the reinforcement learning algorithm, as described in Section III-B. In essence, the learning process aims to achieve the specified goals (automatically transformed into the reward function typical of RL) by taking actions that transition between the specified states. In our simulation, the state space includes rooms, robot/environment attributes, and the position of objects, people, and the robot. States and actions with general determiners (e.g., "pick up *any*") map to multiple states and actions (e.g., both "pick up mail" and "pick up toy").

D. Goal-Only MDP Programming (Goal-MDP)

Goal-Only MDP Programming (*Goal-MDP*) is a simplified version of *Full-MDP*. It pre-specifies all available actions and state attributes (see Section III-A), only requiring the user to specify the goal(s). This design transforms the cognitive burden of configuring the state and action space into a computational cost, introducing a trade-off between the usability of MDP specification and the tractability of the learning problem. In all *Goal-MDP* implementations, the robot's action set and state space remain fixed across tasks to represent all possible states and actions in the domestic robotic system. Once the user provides a goal, the transition dynamics are computed based on these predefined actions and states.

V. User Study Methodology

We conducted a between-subjects user study ($N = 409$) to assess the usability of *Full-MDP* and *Goal-MDP* in comparison to the *Seq* and *TAP* baselines. Each participant was assigned to one of these four programming paradigms and used it to complete programming tasks. Tasks varied in the programming features present in each. We evaluated program correctness, completion time, program length, user experience, and error types. This study was approved by the University of Chicago's Institutional Review Board (IRB23-0093)

TABLE I: Description and features of each programming task.

Task	Features
0. **Baseline**: Move from the bedroom to the kitchen.	–
1. **Person Avoidance**: Avoid a person by exiting any room the person enters. The person moves from room to room constantly.	LP
2. **Toy in Random Room**: Move a toy from some unknown room to the playroom. The toy's initial location varies.	MSA, IU, UNP
3. **Toys in Kitchen**: Move all toys (variable number) from the kitchen.	MSA, IU, UG
4. **Coffee Delivery**: Deliver coffee from the kitchen to a person in an unknown room. The person's location varies.	MSA, EU, UNP
5. **Coffee-or-Mail (Separate Rooms)**: Move mail or coffee to different destinations. The initial item varies between mail and coffee.	MSA, IU, EU
6. **Mail on Porch**: Move three pieces of mail off the porch.	MSA, UG
7. **Person Avoidance While in Kitchen**: Stay in the kitchen while avoiding a person. Exit/re-enter the the room based on the person.	LP, PL
8. **Coffee-or-Mail (Same Room)** Move mail or coffee from the porch to the kitchen. The initial item varies.	MSA, IU
9. **Coffee to Kitchen**: Move a cup of coffee from the bedroom to the kitchen. The robot starts in the playroom.	MSA

A. Programming Tasks and Task Features

The primary use case we envision for having end users program robots using RL is to customize robots to meet specific, subjective goals. In this context, we designed 10 tasks that capture a range of end-user customizations of a TIAGo mobile manipulator robot in domestic settings. We believe these tasks reflect the functionality that users have desired in customizing domestic robots [54], [60]. Table I describes the ten tasks, while information in the supplementary materials includes the exact instructions given to participants.

We designed tasks based on seven key programming features identified in a pilot study. To gauge the influence of each, we compared participant performance on two tasks that differ in that feature. For instance, *Task 0* has no features, while *Task 1* has one (Loop Required). If participants perform worse on *Task 1* than *Task 0*, it suggests difficulty with Loop Required. The seven features and their task pairs follow:

1) **Loop Required (LR)** tasks lack terminal states and thus require using a loop. We compare *Task 0* to *Task 1*.
2) **Priority Levels (PL)** tasks require conveying a preference order for actions. We compare *Task 1* to *Task 7*.
3) **Multiple States and Actions (MSA)** tasks require 3+ state attributes and actions. We compare *Task 0* to *Task 9*.
4) **Initial Uncertainty (IU)** tasks involve limited information at the program's start (e.g., uncertainty about a toy's initial location). We compare *Task 3* to *Task 6*.
5) **End Uncertainty (EU)** tasks involve limited information about the final state. We compare *Task 5* to *Task 8*.
6) **Unwieldy Number of Possibilities (UNP)** tasks involve three or more possible arrangements (e.g., three rooms to check). We compare *Task 2* to *Task 8*.
7) **Unintuitive Goals (UG)** tasks involve a goal state that must be observed upon task completion that may not be straightforward. We compare *Task 3* to *Task 8*.

B. User Study Protocol

We recruited participants through Prolific, requiring that they be 18+ years old, live in the USA, have a 95%+ approval rating on Prolific, and complete the study on a computer (not a phone). Participants were randomly assigned to one of the four programming paradigms and three of the ten programming tasks. They first answered questions about their programming backgrounds, then completed a text-based tutorial on their assigned paradigm, which included a quiz with attention checks. This was followed by an interactive tutorial with three sample tasks. Then, participants completed the programming tasks, answering questions about their experience and approach after each. Finally, they provided their demographics and feedback on their overall experience. Participants were compensated $12 USD and took an average of 70 minutes to complete the study.

C. Measures

To compare programming paradigms, we assessed participants' programs, timing, and reported experiences as follows:

1) **Program Correctness**: We simulated each solution to compare its end state with the expected results. For tasks without end conditions, we ran the program for 1200 time steps and documented the robot and person's locations at each step. For tasks with randomization, we simulated the program five times, each with a randomized start state, to ensure that the solution holds under different conditions. We considered a solution correct only if it produced the expected end state in every simulation trial.
2) **Program Characteristics**: Our interface recorded the time elapsed, number of test runs, number of blocks used, and the state of the robot environment at submission.
3) **System Usability**: Participants completed the 10-item System Usability Scale [94] using 5-point Likert scales.
4) **Subjective Experience**: After each task, participants responded to statements using 5-point Likert scales about the ease of the task, their confidence that they completed the task correctly, and their belief in being able to program the robot. We included open-ended questions about the participant's process, any situations where their program did not work, and how they handled such issues.
5) **Prior Experience**: Participants reported their experience with programming, trigger-action programming, machine learning, and RL (class/tutorial/work/hobby).
6) **Demographics**: Participants gave their age and gender.

D. Statistical Analysis

To analyze quantitative data, we used mixed-effects logistic regression for (binary) program correctness, mixed-effects linear regression for completion time and program length, and ordinal mixed-effects logistic regression for Likert-scale data. All models included the paradigm (*Seq*, *TAP*, *Full-MDP*, *Goal-MDP*), task number, age, gender, and programming experience as fixed effects, as well as the participant ID as a random effect. For significant factors, we report the fixed effects coefficient (c), the standard error (SE), and the p-value. To analyze responses to the System Usability

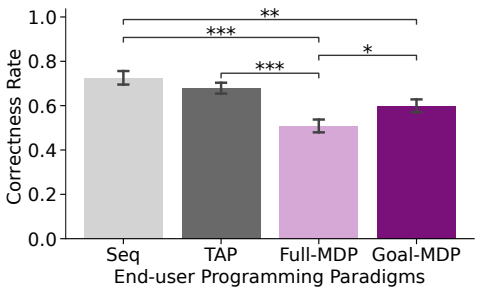


Fig. 3: The correctness of participants' programs by paradigm. ** $p < 0.01$, *** $p < 0.001$. Error bars show standard error.

TABLE II: Differences in program correctness across paradigms by feature. * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

Feature	Presence	Absence
Initial Uncertainty		*Seq* > *Goal-MDP**** *TAP* > *Goal-MDP****
End Uncertainty	*Seq* > *TAP***	*Seq* > *Goal-MDP*** *TAP* > *Goal-MDP**
Unwieldy # of Possibilities	*Goal-MDP* > *TAP***	*Seq* > *Goal-MDP*** *TAP* > *Goal-MDP**
Unintuitive Goals	*Seq* > *Full-MDP**** *TAP* > *Full-MDP****	*Seq* > *Goal-MDP*** *TAP* > *Goal-MDP**

Scale [94], we used a one-way analysis of variance (ANOVA) test. We included the paradigm, age, gender, and programming experience as fixed effects. We report the effect size as η^2. For post-hoc pairwise comparisons, we used Tukey's HSD test.

VI. Results

A. Participants

474 participants completed the study. We excluded 53 with empty submissions on the programming interface and 12 who failed the attention check, leaving 409 participants for analysis. Among participants, 42.8% identified as female, 53.0% as male, and 3.2% as non-binary, while 0.5% preferred self-description and 0.5% opted not to disclose. 5.6% of participants were age 18–24, 35.2% age 25–34, 29.8% age 35–44, 3.7% age 45–55, and 5.6% age 55+. 34.0% indicated some level of experience with programming (taken a course/tutorial), 20.8% some familiarity with *TAP*, and 7.6% some level of experience with reinforcement learning. There were no significant differences in age, gender, or programming experience between conditions.

B. Program Correctness

We found significant differences across the end-user programming paradigms in program correctness. As shown in Figure 3, participants who used *Full-MDP* had a significantly lower average correctness (50.68%) compared with *Seq* ($72.35\%, c = 1.29, SE = 0.29, p < 0.001$), *TAP* ($67.89\%, c = 1.01, SE = 0.26, p < 0.001$), and *Goal-MDP* ($59.93\%, c = 0.55, SE = 0.26, p = 0.035$). Additionally, *Seq* had a higher correctness rate than *Goal-MDP* ($c = -0.73, SE = 0.28, p = 0.010$). No other comparisons yielded significant differences. We also examined how participants in each paradigm differed in program correctness per task, the details of which are included in the supplementary materials.

C. Performance on Task Features

To determine how each paradigm performed on the task features, we examined the differences in correctness in the presence and absence of each feature in the two programming tasks that differed by that one feature (Table II).

We noticed a significant interaction between the paradigm and the presence of the **Initial Uncertainty** feature ($c = 0.66, SE = 0.002, p = 0.025$). While no significant differences exist in the presence of Initial Uncertainty, in its absence, *Seq* ($\chi^2 = 14.14, p < 0.0001$) and *TAP* ($\chi^2 = 17.83, p = 0.0001$) had a higher correctness rate compared to *Full-MDP*. Thus, in the presence of Initial Uncertainty the performance of *Goal-MDP* suffers no more than *Seq* and *TAP*.

We also observed a significant interaction between the paradigm and the **End Uncertainty** feature ($c = -3.12, SE = 1.03, p = 0.003$). In the presence of End Uncertainty, participants in *Seq* performed significantly better than those in *TAP* ($\chi^2 = 0.95, p = 0.0037$). Without End Uncertainty, both *Seq* ($p = 0.001, \chi^2 = 14.07$) and *TAP* ($p = 0.03, \chi^2 = 7.44$) had higher average correctness compared to *Goal-MDP*. These results suggest that *Seq* better handles End Uncertainty than *TAP* and that the performance of *Goal-MDP* suffers less than *Seq* and *TAP* in the presence of End Uncertainty.

For **Unwieldy Number of Possibilities**, we found a significant interaction between the end-user programming paradigm and the presence of the feature ($c = -4.07, SE = 1.10, p < 0.001$). In the presence of an Unwieldy Number of Possibilities, *Goal-MDP* had a significantly higher correctness rate compared to *TAP* ($\chi^2 = 7.44, p = 0.015$). In the absence of the Unwieldy Number of Possibilities feature, participants in *Goal-MDP* performed worse compared to *Seq* ($\chi^2 = 14.07, p = 0.001$) and *TAP* ($\chi^2 = 7.44, p = 0.032$). These results suggest that *Goal-MDP* is more suitable for solving tasks with a cumbersome number of possible conditions.

Lastly, we found a significant interaction between the paradigm and the presence of the **Unintuitive Goals** feature ($c = 2.32, SE = 0.92, p = 0.012$). In the presence of Unintuitive Goals, we found that participants in *Full-MDP* had lower correctness compared to *Seq* ($\chi^2 = 6.02, p < 0.001$) and *TAP* ($\chi^2 = 1.84, p < 0.001$). This indicates that when a task has Unintuitive Goals, users fare worse with *Full-MDP* than *Seq* or *TAP*. In the feature's absence, participants assigned to *Goal-MDP* had lower average correctness compared to *Seq* ($\chi^2 = 14.08, p = 0.001$) and *TAP* ($\chi^2 = 7.45, p = 0.032$).

D. Time to Complete the Program

We found no significant differences across paradigms in program completion time: *Seq* ($M = 4.75$ min, $SD = 7.47$), *TAP* ($M = 4.33$ min, $SD = 6.39$), *Full-MDP* ($M = 5.34$ min, $SD = 8.51$), *Goal-MDP* ($M = 5.28$ min, $SD = 9.65$).

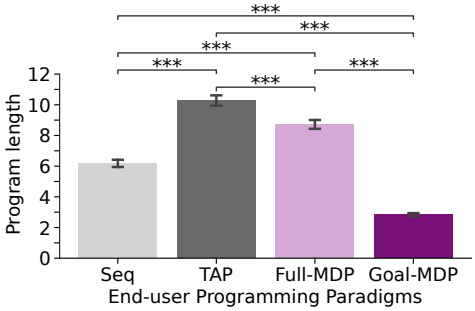


Fig. 4: Participants' *TAP* programs were the longest (# blocks), while *Goal-MDP* programs were the shortest. *** $p < 0.001$.

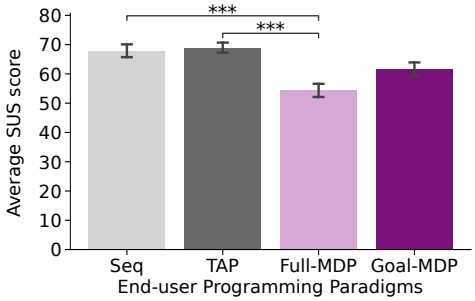


Fig. 5: System Usability Scale (SUS) scores by paradigm. *Sequential Programming* and *Trigger-Action Programming* were rated higher than *Full MDP Programming*. *** $p < 0.001$.

E. Program Length

We found that the paradigm had a significant influence on participants' program length (number of total blocks in the program), see Figure 4. Participants in *Goal-MDP* authored the shortest programs ($M = 2.84, SD = 1.96$), with significantly fewer blocks compared to *Seq* ($M = 6.17, SD = 3.82, c = 2.71, SE = 0.61, p < 0.001$), *TAP* ($M = 10.24, SD = 6.04, c = 7.09, SE = 0.54, p < 0.001$), and *Full-MDP* ($M = 8.65, SD = 4.78, c = 5.17, SE = 0.50, p < 0.001$). Participants in *Seq* authored the second shortest programs, using significantly fewer blocks compared to *TAP* ($c = 4.38, SE = 0.62, p < 0.001$) and *Full-MDP* ($c = 2.47, SE = 0.59, p < 0.001$). Finally, *Full-MDP* had fewer blocks compared to *TAP* ($c = 1.92, SE = 0.52, p < 0.001$).

F. User Experiences of Programming the Tasks

Perceptions of how easy the tasks were differed significantly across paradigms. Participants assigned to *Seq* ($M = 3.60, SD = 1.39$) found the tasks significantly **easier** compared to *Full-MDP* ($c = -0.81, SE = 0.16, M = 3.08, p < 0.001$) and *TAP* ($c = -0.31, SE = 0.149, M = 3.42, p = 0.035$). Further, participants in *Goal-MDP* ($M = 3.45, SD = 1.01$) found the tasks to be significantly easier than those in *Full-MDP* ($c = -0.66, SE = 0.15, M = 3.08, p < 0.001$).

Additionally, we found significant differences in participants' **confidence in accomplishing the tasks** across the end-user programming paradigms. Participants felt more confident using the *TAP* ($M = 4.31$) interface compared to both *Full-MDP* ($c = 1.00, SE = 0.15, M = 3.63, p < 0.001$) and *Goal-MDP* ($c = 0.32, SE = 0.15, M = 4.05, p = 0.033$). Similarly, *Seq* ($c = 0.81, SE = 0.17, M = 4.12, p < 0.001$) and *Goal-MDP* ($c = 0.69, SE = 0.15, M = 4.05, p < 0.001$) had significantly higher confidence compared to *Full-MDP*.

We found a significant difference across paradigms in participants' **perceived ability to program the task correctly**. Participants in the *Seq* ($M = 4.44$) condition had a significantly higher perceived ability to complete the task than *Goal-MDP* ($c = -0.43, SE = 0.17, M = 4.20, p = 0.011$) and *Full-MDP* ($c = -1.20, SE = 0.17, M = 3.76, p < 0.001$). Similarly, participants in the *TAP* condition had a significantly higher perceived ability to complete the task than *Goal-MDP* ($c = -0.35, SE = 0.15, M = 4.20, p = 0.020$) and *Full-MDP* ($c = -1.12, SE = 0.15, M = 3.76, p < 0.001$).

G. System Usability

We also examined the usability of paradigms based on their System Usability Scale (SUS) scores (see Figure 5), which differed significantly across paradigms ($F = 9.67, \eta^2 = 0.07, p < 0.001$). Our post-hoc pairwise comparisons revealed that participants assigned significantly higher usability scores for *Seq* ($M = 67.89, SD = 20.12, p < 0.001$) and *TAP* ($M = 68.94, SD = 19.17, p < 0.001$) compared to *Full-MDP* ($M = 54.47, SD = 23.49$). The SUS score for *Goal-MDP* fell in the middle ($M = 61.56, SD = 24.14$) and did not differ significantly from other paradigms.

H. Types of Errors in Programming Paradigms

In this section, we describe the main errors made by participants in each paradigm. The supplementary material further details all error types we observed and their frequency.

1) Common Errors in Sequential Programming: Most errors in *Seq* solutions can be ascribed to loop misuse. In Loop Required tasks (i.e., *Task 1* and *Task 7*), 37.0% ($N = 10$) of participants who authored incorrect programs in *Seq* did not use any loops, and 29.6% ($N = 8$) had incorrect conditions in their loops, resulting in the program's premature termination or the failure to execute the code within the loop.

2) Common Errors in Trigger-Action Programming: Of the *TAP* participants who made a mistake, 32.5% ($N = 13$) composed conflicting rules, which led to inconsistencies during program execution. These can be particularly difficult to debug given the robot's nondeterministic behavior. Another 30.0% ($N = 12$) failed to include rules essential for task completion. Regarding specific components of the rules in *TAP*, 22.5% ($N = 9$) of participants who submitted incorrect *TAP* programs chose a wrong event, and 10.0% ($N = 4$) chose a wrong state; participants often seemed to confuse the two. These errors highlight the difficulty in formulating rules with the necessary event and state combinations.

3) Common Errors in Full MDP Programming: Of the *Full-MDP* participants who authored incorrect solutions, 49.0% ($N = 26$) omitted necessary states and actions for

policy generation. Furthermore, 37.7% ($N = 20$) of *Full-MDP* participants who failed to complete their tasks detailed step-by-step goals rather than indicating the desired end state. Notably, 22.6% ($N = 12$) of them had a counterintuitive goal, in which the stated goal contradicts the expected outcome from the robot's perspective. Additionally, in the presence of Priority Levels, 22.6% ($N = 12$) of participants failed to establish a clear order for the specified goals. These errors illustrate the difficulty of balancing the tractability of the learning problem against the user's cognitive burden.

4) Common Errors in Goal-Only MDP Programming: Participants in *Goal-MDP* encountered similar types of errors as in *Full-MDP*. Notably, 34.0% ($N = 17$) delineated the goals at every step of completing the task rather than stating the final desired outcome. In the task featuring Priority Levels (i.e., *Task 7*), 30.0% ($N = 15$) of participants who submitted an incorrect solution failed to establish a clear order for task completion, and 12.0% ($N = 6$) of them used 'and' incorrectly. Participants struggled to distinguish the 'and' operator from the priority structure built into the 'Goal' block. Finally, 20.0% ($N = 10$) of incorrect *Goal-MDP* programs had a counterintuitive goal, underscoring challenges in aligning the user's mental model with the robot's state representation.

VII. Discussion & Conclusion

We investigated whether end users could effectively program a robot using an RL-based approach (RQ1). We developed two RL-based interfaces with different levels of abstraction for specifying an MDP — *Full-MDP* and *Goal-MDP*. We found *Goal-MDP*, but not *Full-MDP*, to be a viable alternative to traditional approaches for users programming robots (RQ2). Compared to *Seq* and *TAP*, *Goal-MDP* was more robust to higher levels of task uncertainty and tasks requiring the robot to check multiple states or conditions (RQ3).

A. Takeaways from RL End-User Programming

Participants using *Goal-MDP* achieved a 59.93% correctness rate, compared with the more widely used *Seq* (72.32%) and *TAP* (67.89%) paradigms. In eight out of the ten tasks, they specified goals that led to the correct policies, demonstrating performance comparable to the traditional baselines.

In particular, *Goal-MDP* outperformed *Seq* and *TAP* in tasks with an unwieldy number of possibilities (e.g., moving multiple items to specific rooms), and a higher level of uncertainty (e.g., locating and retrieving objects). *Goal-MDP* excelled in these tasks because specifying goals was easier than outlining a step-by-step execution plan. *Goal-MDP* users also wrote significantly shorter programs with fewer blocks. While shorter programs may not necessarily translate to greater ease of use, as users become more used to programming a robot over time, we hypothesize that shorter programs might enable a better user experience (e.g., by reducing programming time or simplifying debugging.) Finally, *Goal-MDP* achieved a high system usability score of 61.56, slightly lower than but still comparable to *Seq* (67.89) and *TAP* (68.94) with no statistically significant difference between them.

While many *Full-MDP* users defined accurate goals, they struggled to specify the complete set of states, actions, and objectives necessary to derive optimal policies. *Full-MDP* consistently performed worse than all other paradigms. Its limitations suggest that *Goal-MDP* is a more feasible option.

B. Future Improvements to End-User RL Programming

By examining common errors made across *Full-MDP* and *Goal-MDP*, we identified a few areas for improvement. The first is the discrepancy between the user's perception and the robot's sensing of its environment. Because MDP components are defined from the robot's perspective, the proper goal specification may contradict users' intuition. One approach is to provide the robot with global knowledge and implement a mechanism for the robot to memorize and update information about the environment. This would reduce the need for counterintuitive goals that rely on direct observation.

The absence of a debugging mechanism for RL-based paradigms also presented ostensible hurdles. Participants' mistakes often led to static or counter-intuitive robot behavior. Future work should focus on developing a system to visually debug RL-programming paradigms and observe the change in the robot's behavior according to reward values. One possibility is for users to specify impromptu positive and negative rewards and observe the robot's plan adapt in real time. This would help users understand how their goal specifications correspond to the execution of a robot's policy.

In the future, we hope to explore the unique advantages of RL-based paradigms through more complex tasks, such as those featuring an unwieldy number of possibilities. Examining long-term learning effects and user adaptation to *Goal-MDP* would offer valuable insights into its practicality as well.

C. Limitations

As an initial exploration of end-user programming with RL, we made several simplifying assumptions in designing the robot's environment, to prevent overly complex state and action spaces and to shorten the RL training time. Expanding the range of skills (actions) is computationally costly. Both the RL and robotics communities are actively investigating methods to scale RL for real-world tasks. We hope to leverage this line of work to enable end users to harness these capabilities and specify tasks that are relevant to them. Future work could develop end-user programming interfaces capable of integrating state-of-the-art reinforcement learning techniques.

D. Generalizability of End-User RL Programming

Our study features high-level tasks and transferable skills for robots in various domains. We focused on fundamental operations, including navigation and object manipulation, which are common in commercial and healthcare settings. This ensures our tasks are specific enough for precise execution, yet general enough for applications across platforms and environments. Given that our study maintains the core set of programmable capabilities essential for any robot, we believe our findings can be generalized to robotic tasks outside the home.

REFERENCES

[1] T. Belpaeme, J. Kennedy, A. Ramachandran, B. Scassellati, and F. Tanaka, "Social robots for education: A review," *Science Robotics*, vol. 3, no. 21, pp. 1–9, 2018.

[2] J. E. Michaelis and B. Mutlu, "Reading socially: Transforming the in-home reading experience with a learning-companion robot," *Science Robotics*, vol. 3, no. 21, pp. 1–11, 2018.

[3] H. W. Park, I. Grover, S. Spaulding, L. Gomez, and C. Breazeal, "A model-free affective reinforcement learning approach to personalization of an autonomous social robot companion for early literacy education," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 687–694.

[4] A. Ramachandran, S. S. Sebo, and B. Scassellati, "Personalized robot tutoring using the assistive tutor POMDP (AT-POMDP)," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 8050–8057.

[5] H.-M. Gross, S. Mueller, C. Schroeter, M. Volkhardt, A. Scheidig, K. Debes, K. Richter, and N. Doering, "Robot companion for domestic health assistance: Implementation, test and case study under everyday conditions in private apartments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5992–5999.

[6] H. R. Lee and L. D. Riek, "Reframing assistive robots to promote successful aging," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 7, no. 1, pp. 1–23, 2018.

[7] H. Robinson, B. MacDonald, and E. Broadbent, "The role of healthcare robots for older people at home: A review," *International Journal of Social Robotics*, vol. 6, no. 4, pp. 575–591, 2014.

[8] C.-A. Smarr, A. Prakash, J. M. Beer, T. L. Mitzner, C. C. Kemp, and W. A. Rogers, "Older adults' preferences for and acceptance of robot assistance for everyday living tasks," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 56, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2012, pp. 153–157.

[9] J. E. Martinez, D. VanLeeuwen, B. B. Stringam, and M. R. Fraune, "Hey?! what did you think about that robot?: Groups polarize users' acceptance and trust of food delivery robots," in *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, 2023, pp. 417–427.

[10] B. Mutlu and J. Forlizzi, "Robots in organizations: the role of workflow, social, and environmental factors in human-robot interaction," in *Proceedings of the 3rd ACM/IEEE International Conference on Human-Robot Interaction*, 2008, pp. 287–294.

[11] D. Weinberg, H. Dwyer, S. E. Fox, and N. Martelaro, "Sharing the sidewalk: Observing delivery robot interactions with pedestrians during a pilot in Pittsburgh, PA," *Multimodal Technologies and Interaction*, vol. 7, no. 5, p. 53, 2023.

[12] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: A study of the Romba vacuum in the home," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, 2006, pp. 258–265.

[13] X. Lin, Y. Wang, Z. Huang, and D. Held, "Learning visible connectivity dynamics for cloth smoothing," in *Conference on Robot Learning*. PMLR, 2022, pp. 256–266.

[14] B. Tang, M. Corsaro, G. Konidaris, S. Nikolaidis, and S. Tellex, "Learning collaborative pushing and grasping policies in dense clutter," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6177–6184.

[15] G. Ajaykumar, M. Steele, and C.-M. Huang, "A survey on end-user robot programming," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–36, 2021.

[16] D. Bambuŝek, Z. Materna, M. Kapinus, V. Beran, and P. Smrž, "Combining interactive spatial augmented reality with head-mounted display for end-user collaborative robot programming," in *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2019, pp. 1–8.

[17] S. Y. Gadre, E. Rosen, G. Chien, E. Phillips, S. Tellex, and G. Konidaris, "End-user robot programming using mixed reality," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2707–2713.

[18] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. M. Van der Loos, and E. Croft, "Robot programming through augmented trajectories in augmented reality," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1838–1844.

[19] S. Alexandrova, Z. Tatlock, and M. Cakmak, "Roboflow: A flow-based visual programming language for mobile manipulation tasks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5537–5544.

[20] Y. S. Liang, D. Pellier, H. Fiorino, S. Pesty, and M. Cakmak, "Simultaneous end-user programming of goals and actions for robotic shelf organization," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6566–6573.

[21] J. E. Young, T. Igarashi, E. Sharlin, D. Sakamoto, and J. Allen, "Design and evaluation techniques for authoring interactive and stylistic behaviors," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 3, no. 4, pp. 1–36, 2014.

[22] M. Cakmak and L. Takayama, "Teaching people how to teach robots: The effect of instructional materials and dialog design," in *Proceedings of the 2014 ACM/IEEE International Conference on Human-Robot Interaction*, 2014, pp. 431–438.

[23] J. Huang and M. Cakmak, "Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 2017, pp. 453–462.

[24] A. Kubota, E. I. Peterson, V. Rajendren, H. Kress-Gazit, and L. D. Riek, "Jessie: Synthesizing social robot behaviors for personalized neurorehabilitation and beyond," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 121–130.

[25] T. Sapounidis and S. Demetriadis, "Tangible versus graphical user interfaces for robot programming: Exploring cross-age children's preferences," *Personal and Ubiquitous Computing*, vol. 17, pp. 1775–1786, 2013.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[27] A. Barišić, J. Cambeiro, V. Amaral, M. Goulão, and T. Mota, "Leveraging teenagers feedback in the development of a domain-specific language: The case of programming low-cost robots," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1221–1229.

[28] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, and G. D. Hager, "Evaluating methods for end-user creation of robot task plans," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6086–6092.

[29] D. Porfirio, E. Fisher, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Bodystorming human-robot interactions," in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 2019, pp. 479–491.

[30] S. Moros, L. Wood, B. Robins, K. Dautenhahn, and Á. Castro-González, "Programming a humanoid robot with the Scratch language," in *Robotics in Education: Current Research and Innovations 10*. Springer, 2020, pp. 222–233.

[31] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blockly goes to work: Block-based programming for industrial robots," in *2017 IEEE Blocks and Beyond Workshop (B&B)*. IEEE, 2017, pp. 29–36.

[32] K. Ryokai, M. J. Lee, and J. M. Breitbart, "Children's storytelling and programming with robotic characters," in *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, 2009, pp. 19–28.

[33] Y. S. Liang, D. Pellier, H. Fiorino, and S. Pesty, "End-user programming of low-and high-level actions for robotic task planning," in *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2019, pp. 1–8.

[34] M. Racca, V. Kyrki, and M. Cakmak, "Interactive tuning of robot program parameters via expected divergence maximization," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 2020, pp. 629–638.

[35] C. Schou, R. S. Andersen, D. Chrysostomou, S. Bøgh, and O. Madsen, "Skill-based instruction of collaborative robots in industrial settings," *Robotics and Computer-Integrated Manufacturing*, vol. 53, pp. 72–80, 2018.

[36] M. Hagenow, E. Senft, R. Radwin, M. Gleicher, M. Zinn, and B. Mutlu, "A system for human-robot teaming through end-user programming and shared autonomy," *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, vol. 54, p. 231–239, Mar 2024.

[37] 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.

[38] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Survey: Robot programming by demonstration," Springer, Tech. Rep., 2008.

[39] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 481–487, 2002.

[40] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[41] Y. Liu, Z. Li, H. Liu, and Z. Kan, "Skill transfer learning for autonomous robots and human–robot cooperation: A survey," *Robotics and Autonomous Systems*, vol. 128, p. 103515, 2020.

[42] 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.

[43] Y. Gao and C.-M. Huang, "PATI: A projection-based augmented table-top interface for robot programming," in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, 2019, pp. 345–355.

[44] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations." in *Robotics: Science and Systems*, 2014, pp. 1–9.

[45] J. F. Gorostiza and M. A. Salichs, "End-user programming of a social robot by dialog," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1102–1114, 2011.

[46] U. B. Karli, J.-T. Chen, V. N. Antony, and C.-M. Huang, "Alchemist: LLM-aided end-user development of robot applications: Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction," Mar 2024. [Online]. Available: https://doi.org/10.1145/3610977.3634969

[47] D. J. Porfirio, L. Stegner, M. Cakmak, A. Sauppé, A. Albarghouthi, and B. Mutlu, "Figaro: A tabletop authoring environment for human-robot interaction," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.

[48] Y. S. Sefidgar, P. Agarwal, and M. Cakmak, "Situated tangible robot programming," in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, 2017, pp. 473–482.

[49] J. Huang, T. Lau, and M. Cakmak, "Design and evaluation of a rapid programming system for service robots," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2016, pp. 295–302.

[50] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[51] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd, and D. Franklin, "Evaluating coblox: A comparative study of robotics programming environments for adult novices," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[52] A. Rahmati, E. Fernandes, J. Jung, and A. Prakash, "IFTTT vs. Zapier: A comparative study of trigger-action programming frameworks," *arXiv preprint arXiv:1709.02788*, 2017.

[53] X. Mi, F. Qian, Y. Zhang, and X. Wang, "An empirical characterization of IFTTT: Ecosystem, usage, and performance," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 398–404.

[54] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman, "Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 3227–3231.

[55] A. A. Nacci, V. Rana, B. Balaji, P. Spoletini, R. Gupta, D. Sciuto, and Y. Agarwal, "Buildingrules: A trigger-action–based system to manage complex commercial buildings," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 2, pp. 1–22, 2018.

[56] Microsoft Dynamics 365 Team, "Announcing RPA, enhanced security, no-code virtual agents, and more for Microsoft Power Platform," Blog Post, 2019. [Online]. Available: https://cloudblogs.microsoft.com/dynamics365/bdm/2019/11/04/announcing-rpa-enhanced-security-no-code-virtual-agents-and-more-for-microsoft-power-platform/

[57] N. Leonardi, M. Manca, F. Paternò, and C. Santoro, "Trigger-action programming for personalising humanoid robot behaviour," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[58] E. Senft, M. Hagenow, R. Radwin, M. Zinn, M. Gleicher, and B. Mutlu, "Situated live programming for human-robot collaboration," in *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021, pp. 613–625.

[59] B. Ikeda and D. Szafir, "Programar: Augmented reality end-user robot programming," *ACM Transactions on Human-Robot Interaction*, 2024.

[60] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 803–812.

[61] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, "Personalization of context-dependent applications through trigger-action rules," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 24, no. 2, pp. 1–33, 2017.

[62] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, M. L. Littman, and B. Ur, "How users interpret bugs in trigger-action programming," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–12.

[63] J. Huang and M. Cakmak, "Supporting mental model accuracy in trigger-action programming," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 215–225.

[64] L. Zhang, C. Zhou, M. L. Littman, B. Ur, and S. Lu, "Helping users debug trigger-action programs," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 4, pp. 1–32, 2023.

[65] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[66] M. Manca, F. Paternò, C. Santoro, and L. Corcella, "Supporting end-user debugging of trigger-action rules for IoT applications," *International Journal of Human-Computer Studies*, vol. 123, pp. 56–69, 2019.

[67] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "AutoTap: Synthesizing and repairing trigger-action programs using LTL properties," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 281–291.

[68] L. Bu, W. Xiong, C.-J. M. Liang, S. Han, D. Zhang, S. Lin, and X. Li, "Systematically ensuring the confidence of real-time home automation IoT systems," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 3, pp. 1–23, 2018.

[69] C.-J. M. Liang, L. Bu, Z. Li, J. Zhang, S. Han, B. F. Karlsson, D. Zhang, and F. Zhao, "Systematically debugging IoT control system correctness for building automation," in *Proceedings of the 3rd ACM International Conference on Systems for Energy-efficient Built Environments*, 2016, pp. 133–142.

[70] V. Zhao, L. Zhang, B. Wang, M. L. Littman, S. Lu, and B. Ur, "Understanding trigger-action programs through novel visualizations of program differences," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–17.

[71] F. Corno, L. De Russis, and A. Monge Roffarello, "TAPrec: Supporting the composition of trigger-action rules through dynamic recommendations," in *Proceedings of the 25th International Conference on Intelligent User Interfaces*, 2020, pp. 579–588.

[72] I. N. B. Yusuf, D. B. A. Jamal, L. Jiang, and D. Lo, "RecipeGen++: an automated trigger action programs generator," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1672–1676.

[73] A. Mattioli and F. Paternò, "Recommendations for creating trigger-action rules in a block-based environment," *Behaviour & Information Technology*, vol. 40, no. 10, pp. 1024–1034, 2021.

[74] L. Zhang, W. He, O. Morkved, V. Zhao, M. L. Littman, S. Lu, and B. Ur, "Trace2TAP: Synthesizing trigger-action programs from traces of behavior," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 3, pp. 1–26, 2020.

[75] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, pp. 97–102.

[76] F. Corno, L. De Russis, and A. Monge Roffarello, "From users' intentions to if-then rules in the internet of things," *ACM Transactions on Information Systems (TOIS)*, vol. 39, no. 4, pp. 1–33, 2021.

[77] T.-H. K. Huang, A. Azaria, and J. P. Bigham, "Instructablecrowd: Creating if-then rules via conversations with the crowd," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 1555–1562.

[78] M. L. Puterman, *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc., 1994.

[79] D. H. Ackley and M. L. Littman, "Interactions between learning and evolution," in *Artificial Life II: Santa Fe Institute Studies in the Sciences of Complexity*, C. Langton, C. Taylor, J. D. Farmer, and S. Ramussen, Eds. Redwood City, CA: Addison-Wesley, 1991, vol. 10, pp. 487–509.
[80] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.
[81] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. L. Littman, D. Precup, and S. Singh, "On the expressivity of Markov reward," in *Neural Information Processing Systems*, 2021.
[82] F. Bacchus, C. Boutilier, and A. Grove, "Rewarding behaviors," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. AAAI Press/The MIT Press, 1996, pp. 1160–1167.
[83] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan, "Environment-independent task specifications via GLTL," 2017, arXiv preprint arXiv:1704.04341.
[84] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.
[85] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The TAMER framework," in *Proceedings of the Fifth International Conference on Knowledge Capture*, 2009, pp. 9–16.
[86] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," in *Proceedings of the Thirty-Fourth International Conference on Machine Learning*, 2017.
[87] D. Porfirio, M. Roberts, and L. M. Hiatt, "Goal-oriented end-user programming of robots," in *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 2024, pp. 582–591.
[88] PAL Robotics, "Tiago mobile manipulator robot," 2024. [Online]. Available: https://pal-robotics.com/robots/tiago/
[89] Hello Robot, "Stretch 3 a fully integrated mobile manipulator," 2024. [Online]. Available: https://hello-robot.com/stretch-3-product
[90] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette, "Learning to understand goal specifications by modelling reward," *arXiv:1806.01946*, 2018.
[91] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015, dqn.
[92] [Online]. Available: https://github.com/jennjwang/eup-blockly.git
[93] Google Developer Program, "Blockly," 2024. [Online]. Available: https://developers.google.com/blockly/
[94] J. Brooke, "SUS: A quick and dirty usability scale," *Usability Evaluation in Industry*, vol. 189, no. 3, pp. 189–194, 1996.