

Flexible Language-Guided Representation Editing for Adaptive Robot Behavior

Audrey Lee¹
audreylee@alum.mit.edu

Nathaniel Dennler¹
dennler@mit.edu

Ilia Sucholutsky²
isucholu@purdue.edu

Andreea Bobu¹
abobu@mit.edu

¹ MIT Computer Science & Artificial Intelligence Laboratory ² New York University

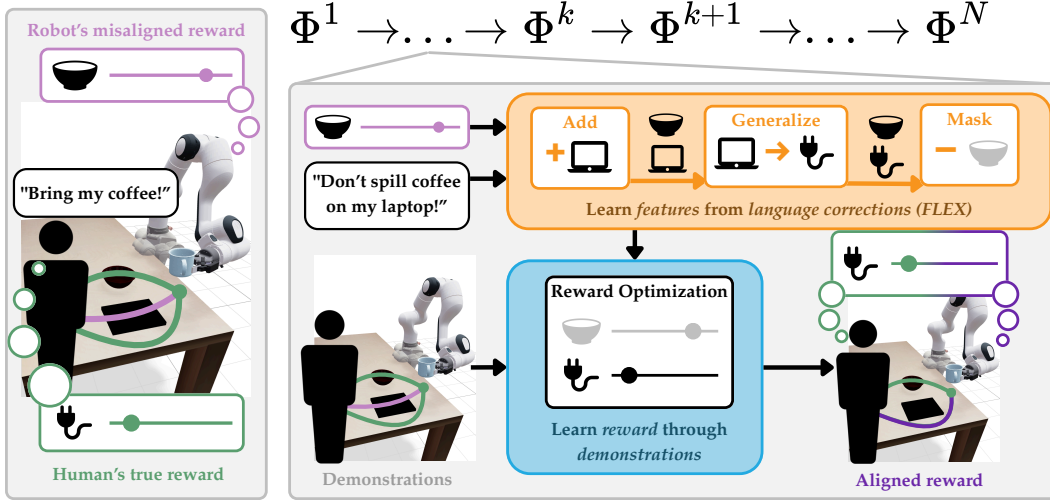


Figure 1: FLEX overview. A human’s task instruction (left) results in a misalignment between the human’s (green) and robot’s (lavender) internal task representations. To address this misalignment, FLEX receives natural language corrections and applies three operations (orange) to ADD missing features, GENERALIZE overly specific features, and MASK irrelevant features. The robot then plans under the corrected feature set to optimize a reward function through demonstrations (blue), yielding trajectories that are better aligned with the human’s true reward (purple, bottom-right).

Abstract: Trajectory demonstrations show robots how to behave but not why, leaving them to guess which *features* of the task actually matter. Incorrect guesses produce misaligned internal representations and lead to incorrect robot behaviors. Because task features are often semantically defined, we argue that natural language is better suited for specifying features rather than relying on demonstrations. Although existing representation editing methods have proposed using language to specify new features, only allowing additive edits produces feature sets that are often over-specified or too narrowly scoped, especially in sequential editing tasks. We propose the Flexible Feature Editing Pipeline (FLEX), which uses language models to iteratively refine a robot’s internal representation of task features from natural language corrections via three complementary operations: ADD missing features, GENERALIZE existing specific features, and MASK irrelevant features. We evaluate FLEX on 2D navigation scenarios, 7DoF Franka tabletop manipulation tasks, and a real-world study using naturalistic human language corrections and real-robot demonstrations. Comparisons against single-operation and unscaffolded baselines show that all three operations are necessary in interactive scenarios, and that FLEX achieves consistently higher normalized reward compared to other feature editing approaches while maintaining compact, transferable representations. Project available at <https://mit-clear-lab.github.io/FLEX/>.

Keywords: reward learning, representation alignment, language feedback

1 Introduction

Imagine a scenario like in Fig. 1, where a human asks their personal robot to bring them coffee. While the instruction seems simple, executing it well depends on the *features*, or measurable properties of the environment, that the robot internally uses to represent the task. For example, a user may prefer that the robot stays away from sensitive electronics or that it stays close to the table to avoid damage from spills. Unfortunately, human instructions rarely describe these features explicitly when specifying tasks, forcing the robot to guess which features matter. When the robot guesses incorrectly, the human and robot end up with *misaligned* representations, and the robot fails to complete the task in the way that the human desires.

A common approach to address the representation misalignment problem is to iterate between specifying features and optimizing reward, often through demonstrations [1, 2]. However, demonstrations carry little semantic information, so the robot can only *implicitly* recover which features actually matter to the user. In contrast, natural language contains rich semantic content. When a user says, “Don’t spill coffee on my laptop!” their language communicates both what matters about the task (avoiding the laptop) and why that matters (preventing damage). Language contains the semantic content that demonstrations lack. We therefore propose that feature specification is better addressed through language, while demonstrations are better-suited for reward optimization once the right features are known.

Even a simple language correction contains a wealth of information when interpreted in context. However, existing methods fail to leverage this fully, either failing to update the robot’s underlying representation in a generalizable way [3] or continually adding features until representations become redundant and inefficient [1]. None of these methods account for the full space of ways a feature representation can be wrong: it may be *incomplete* (missing relevant features), *over-specified* (tracking irrelevant features), or *incorrectly abstracted* (tracking a feature at the wrong level of abstraction).

We propose FLEX, a flexible feature editing pipeline that uses Language Models (LMs) to adapt a robot’s internal representation of task features in response to natural language corrections. As shown in Fig. 1, when the human says “Don’t spill coffee on my laptop!”, FLEX identifies the correction’s intent and applies three operations: ADD introduces the laptop as a relevant obstacle, GENERALIZE broadens the newly added feature to all electronics, and MASK removes the now-irrelevant bowl feature. This yields a feature set aligned with the human’s true intent. Unlike prior methods that support only a single operation, FLEX selectively applies whichever combination of edits the user’s language correction warrants.

2 Related Work

Reward Learning from Human Feedback. Learning from human feedback is necessary to adapt robot policies to new users and tasks [4–7]. Past work has proposed many feedback signals—such as demonstrations [8–11], corrections [12–14], rankings [15–17], and other user behaviors [18–21]—to adapt robot behavior to user preferences. However, many of the most common forms of feedback are underspecified; for example, similar demonstrations can be consistent with many possible reward functions [22–24]. Resolving this ambiguity requires large amounts of data that are impractical to collect from non-expert users [20, 25]. While past work has addressed the data efficiency problem by combining multiple types of feedback [18, 26–28], these approaches collect data and train rewards offline. We instead focus on robots that adapt at deployment time through iterative user feedback.

Specifying Trajectory Features. To make online reward learning tractable for the amount of data that can be collected at deployment time, the robot must restrict its set of reward hypotheses to a low-dimensional space, rather than attempting to learn directly from high-dimensional observations [29, 30]. Prior work defines this space by using expressive trajectory features that are either designed by experts [31–34], learned from data [2, 20, 35–38], or generated with LMs [1, 39]. The problem of *representation misalignment* [22, 40] arises when these feature sets do not capture what the user actually cares about, which results in the robot failing to complete the user’s desired task. This

problem motivates recent works that allow users to directly edit the feature set to align user and robot representations [1, 34]. However, these prior methods support only a single type of edit: adding features [1], removing features [41], or generalizing the scope of features [42]. Our work unifies these separate editing operations into a single framework, and demonstrates that all three edit types are necessary to address the full range of representational misalignment types.

Natural Language for Robot Learning. A growing body of work incorporates natural language into robot learning pipelines, using language to clarify task goals [3, 43–47], redirect ongoing behavior [26, 48, 49], or update internal representations such as safety constraints [50, 51]. However, these approaches treat language primarily as a signal for *how* to perform a task, rather than *which* features of the environment matter [27]. We identify that language carries unique semantic information that makes it particularly well-suited for feature editing. LMs are equipped to leverage this data because they are trained on human-generated data and have reasonable priors over shared preferences [42, 52, 53], which allows them to efficiently leverage sparse natural language corrections [28, 54]. The iterative nature of our approach provides advantages over approaches that use language solely to specify task goals from single instances of feedback.

3 Problem Definition

We focus on the problem of learning reward functions that reflect users’ underlying preferences, where our goal is to generate robot behavior from limited amounts of user feedback. In this work, the user provides feedback as demonstrations and language.

Preliminaries. We consider the robot to be operating in an MDP defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, with state space $\mathcal{S} \subseteq \mathbb{R}^{d_s}$, action space $\mathcal{A} \subseteq \mathbb{R}^{d_a}$, transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, and reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. A solution to this MDP is a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that when rolled out produces a trajectory, $\tau = (s^0, a^0, s^1, a^1, \dots, s^T)$. Following past work in inverse reinforcement learning [31, 55], we model the reward as a linear combination of state features, $\Phi(s)$, such that the reward over a full trajectory is expressed as:

$$R_\theta(\tau) = \sum_{i=0}^T \theta^\top \Phi(s^i) = \theta^\top \Phi(\tau) . \quad (1)$$

Here, θ represents the user’s preferences as a vector of coefficients that express a weighted tradeoff across the task features, Φ .

Reward Optimization. In practice, the robot doesn’t know the reward function R_θ and must learn it from human feedback. This problem is well-studied in domains where the correct Φ is known *a priori* [6, 13, 31, 32]. One standard approach is to estimate the preference weights, θ , by collecting samples of the user’s preferred behavior as a dataset of demonstrations, $\mathcal{D} = \{\tau_i\}_{i=0}^N$.

While our framework is agnostic to the specific reward learning algorithm used, we adopt the maximum entropy principle [32, 55, 56]. This principle models the human as a *noisily rational* agent that intends to provide optimal trajectories but does not always execute them perfectly, as captured by the Boltzmann distribution over trajectories:

$$p(\tau \mid \theta) \propto \exp(\theta^\top \Phi(\tau)) . \quad (2)$$

The Boltzmann distribution assigns higher probability to trajectories with higher reward, while still allowing for suboptimal behavior. The likelihood of the full demonstration dataset is then $p(\mathcal{D} \mid \theta) = \prod_{\tau \in \mathcal{D}} p(\tau \mid \theta)$, and the robot can estimate θ via Bayes’ Rule:

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta)p(\theta) , \quad (3)$$

where $p(\theta)$ is a uniform prior over θ . Once the robot infers θ , it can solve $\tau^* = \arg \max_\tau \theta^\top \Phi(\tau)$ to recover the user’s preferred trajectory.

Feature Specification. While this approach is effective when the right features are available, it assumes that the relevant features are specified before deployment. This assumption is often unrealistic in unstructured and evolving human-centered environments. In this work, we consider the more

general problem of jointly estimating the representation Φ alongside the reward weights θ [1]:

$$\theta^*, \Phi^* = \arg \max_{\theta, \Phi} p(\Phi, \theta \mid \mathcal{D}) . \quad (4)$$

Prior works have addressed this problem by decomposing the problem into two iterative steps: *feature specification*, which refines the feature set Φ^{k+1} under a fixed θ^k , and *reward optimization*, which estimates θ^{k+1} using a fixed Φ^{k+1} at each iteration [1, 2].

The goal of feature specification is to use data collected from the user to estimate a refined feature set at each iteration, given the current feature set Φ^k and reward weights θ^k :

$$\Phi^{k+1} = \arg \max_{\Phi} p(\Phi \mid \Phi^k, \mathcal{D}; \theta^k) . \quad (5)$$

Past work uses demonstrations for both steps of the process [1, 2], but we argue this is suboptimal: a single demonstrated trajectory may be consistent with many possible representations. While one solution is to collect more demonstrations, this is labor-intensive and places a heavy burden on a signal that does not efficiently carry representational information.

Language corrections, by contrast, can express representational content directly, making them a more natural and efficient signal for feature specification. Our key insight is that feature specification can better leverage *semantic* information such as language corrections, whereas demonstrations are better for communicating *preferential* information for reward optimization. We thus extend the feature specification problem to leverage iterative language corrections, ℓ^k :

$$\Phi^{k+1} = \arg \max_{\Phi} p(\Phi \mid \Phi^k, \ell^k; \theta^k) . \quad (6)$$

We next describe our algorithm for performing feature specification with language feedback.

4 Flexible Feature Editing Pipeline

The Flexible Feature Editing Pipeline (FLEX) uses natural language to specify features by interpreting semantic feedback through a sequence of complementary operations.

At each iteration k , the robot executes its current best trajectory $\tau^k = \arg \max_{\tau} \theta^{k\top} \Phi^k(\tau)$, which reflects both its current representation Φ^k and weights θ^k . The human’s desired behavior is an imagined trajectory $\tau^* = \arg \max_{\tau} \theta^{*\top} \Phi^*(\tau)$ that is optimal under Φ^* and reward weights θ^* . We assume that the human observes τ^k , identifies the misalignment between τ^k and τ^* , and provides a language correction ℓ^k to communicate this misalignment: $\ell^k \sim P(\ell \mid \tau^*, \tau^k)$.

Since τ^* and τ^k are deterministic functions of (Φ^*, θ^*) and (Φ^k, θ^k) respectively, we can write $\ell^k \sim P(\ell \mid \Phi^*, \theta^*, \Phi^k, \theta^k)$. In this paper, we use ℓ^k to infer which features matter (Φ^*), but not to infer how much they matter (θ^*). Under this assumption, ℓ is conditionally independent of θ^* and θ^k given Φ^* and Φ^k , simplifying to $\ell^k \sim P(\ell \mid \Phi^*, \Phi^k)$.

This gives us a probabilistic model of human language corrections. Since we are interested in estimating the human’s true representation Φ^* , we apply Bayes’ rule to obtain:

$$P(\Phi^* \mid \Phi^k, \ell^k) \propto P(\ell^k \mid \Phi^*, \Phi^k) \cdot P(\Phi^* \mid \Phi^k) , \quad (7)$$

where we assume a uniform prior $P(\Phi^* \mid \Phi^k)$, reflecting the assumption that the human’s true representation Φ^* is independent of what the robot currently knows Φ^k . This exactly corresponds to Eq. 6 with θ^k dropped under our representational feedback assumption. Rather than computing the posterior in Eq. 7 explicitly, we approximate $\Phi^{k+1} = \arg \max_{\Phi} P(\Phi \mid \Phi^k, \ell^k)$ directly using a language model, which is well suited to proposing feature sets that explain the observed correction ℓ^k given the current feature set Φ^k : $\Phi^{k+1} \approx f_{\text{LM}}(\Phi^k, \ell^k)$.

We break down $f_{\text{LM}}(\Phi^k, \ell^k)$ into a set of three distinct editing operations in the following order: ADD, GENERALIZE, and MASK. While prior work has proposed individual operations for feature editing, we show that a single language correction can simultaneously motivate several distinct types of edits, and that all three are necessary for robust reward learning. Each operation takes as input a

textual description of the task and environment with the user’s natural language correction and maps the current feature set to a refined one, $\Phi^k \rightarrow \Phi^{k+1}$.

The ADD Operation. The ADD operation takes the user’s correction ℓ and the current feature set Φ as input to expand the feature set to consider new objectives: $\text{ADD}(\ell, \Phi) = \Phi'$, where $\Phi' \supseteq \Phi$.

ADD queries an LM twice: once for the name of the new feature and once to generate code that implements the function that computes this new feature. First, the LM takes ℓ^k and Φ^k as input to jointly (1) estimate a boolean *sufficiency score*, $b \in \{0, 1\}$, that determines if the existing feature set Φ^k captures the content of the language correction, and (2) compute a *proposed feature* ϕ' that would capture the content of the language correction. The LM returns the proposed feature if Φ^k did not explain the user’s correction or the null set if Φ^k was sufficient:

$$f_{\text{LM}}(\Phi^k, \ell^k) = \begin{cases} \phi' & \text{if } b = 0 \\ \emptyset & \text{otherwise} \end{cases} \quad (8)$$

If a new feature ϕ' is returned, we append it to the existing set: $\Phi^{k+1} = \Phi^k \cup \phi'$.

The GENERALIZE Operation. The GENERALIZE operation takes the user’s correction ℓ , and the current feature set Φ as input, and returns an augmented feature set: $\text{GENERALIZE}(\ell, \Phi) = \Phi'$, where $|\Phi'| = |\Phi|$.

The augmented feature set contains features that are more general than the input feature set. For example, if a user tells a robot carrying water to stay away from a laptop, the higher-level intent may be to “stay away from electronic devices”. The resulting feature generalization produces a new feature that not only indicates distance to laptops, but also distance to other electronic devices, such as cell phones or calculators. We leverage the common-sense reasoning capabilities of LMs [42] to explicitly extract a higher-level intent ω behind the correction through zero-shot prompting:

$$f_{\text{LM}}(\Phi^k, \ell^k) = \omega \quad (9)$$

For each feature $\phi_i \in \Phi^k$, we determine if that feature can be generalized to new objects that match the high-level intent, ω . This process produces a new feature based on the high-level intent:

$$f_{\text{LM}}(\ell^k, \phi_i, \omega) = \phi_i^{\text{gen}} \quad (10)$$

In practice, we create these generalized features by assuming objects have a set of text-based descriptive properties to generalize across. We implement the generalized feature with code generated by the LM, which replaces its original implementation, $\Phi^{k+1} = (\Phi^k \setminus \{\phi_i\}) \cup \{\phi_i^{\text{gen}}\}$.

The MASK Operation. The MASK operation takes the user’s correction ℓ and the current feature set Φ as input to reduce the current feature set to remove extraneous or irrelevant features: $\text{MASK}(\ell, \Phi) = \Phi'$, where $\Phi' \subseteq \Phi$.

To compute the MASK operation, we use an LM to generate a binary mask over the feature set, Φ . For each feature, the LM estimates a boolean relevancy score, $m_i \in \{0, 1\}$. We query the LM once per feature to determine whether each $\phi_i \in \Phi^k$ remains relevant given the current context:

$$m_i = f_{\text{LM}}(\Phi^k, \ell^k) = \begin{cases} 1 & \text{if relevant}(\phi_i, \ell^k) \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

Operation Order. FLEX applies operations in the order $\text{ADD} \rightarrow \text{GENERALIZE} \rightarrow \text{MASK}$, reflecting dependencies between the operations. ADD runs first to ensure newly added features are fully processed. GENERALIZE follows, and MASK runs last so that masking decisions are made on the features’ final forms. Other orderings require re-iteration to reach a consistent representation.

5 Experiments

We evaluate our algorithm in simulated and real environments through three key research questions that progressively relax assumptions about robot and human behavior: (1) Are all three operations

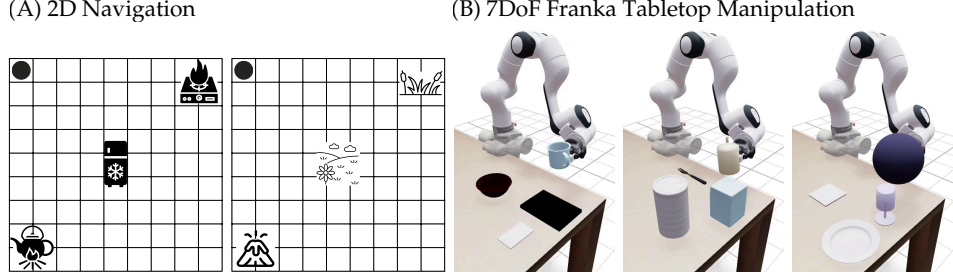


Figure 2: Simulation domains. (A) A 9x9 2D navigation environment, and (B) 7DoF Franka, a tabletop manipulation environment. Each domain includes object variants to test generalization.

necessary for aligning human and robot representations? (2) Does FLEX perform well in continuous control settings with sequential corrections? (3) Does FLEX perform well with the natural and idiosyncratic feedback that real humans provide?

Domains. Our experiments span two domains: *2D Navigation* and *7DoF Franka Tabletop Manipulation*, shown in Fig. 2. In the *2D Navigation* domain, each scenario occurs in a 9-by-9 2D grid where a robot moves from a start position to a goal position while avoiding obstacles. In the *7DoF Franka Tabletop Manipulation* domain, a robot arm manipulates various objects in a tabletop setting by moving from a start configuration to a goal configuration while avoiding objects.

Baselines. We evaluate FLEX against two single-operation baselines and one unscaffolded LM baseline. (1) **FLEX** uses all operations described in Section 4: ADD, GENERALIZE, and MASK. (2) **ALGAE** [1] is a baseline that only adds new features based on the human’s language correction and demonstrations. (3) **Mask-From-Exhaustive (MFE)** starts with an exhaustive list of features and queries an LM with the human’s language correction to mask out all irrelevant features. (4) **Unscaffolded (USF)** queries the LM once to directly generate the full updated Φ^{k+1} from the correction, demonstration, and current Φ^k without decomposing the edit into our three operations. All experiments use GPT-4o as the LM.

Metrics. All robot scenarios involve a robot moving from a starting configuration to a goal configuration with various objects in the environment. In all scenarios, we randomize start, goal, and object positions across five seeds for evaluation. Our main evaluation metric is *normalized reward*, which is computed as $\hat{R} = (R - R_{\text{worst}}) / (R_{\text{best}} - R_{\text{worst}})$. R_{best} and R_{worst} are the rewards of the ground-truth optimal and worst-case trajectories, respectively. A score of $\hat{R} = 1$ indicates optimal behavior and a score of $\hat{R} = 0$ indicates worst-case behavior.

5.1 Isolation of FLEX’s Components (RQ1)

Our first research question evaluates whether all three operations of FLEX are necessary.

Experimental Details. We evaluate RQ1 using two variants of the 2D Navigation domain to demonstrate FLEX’s usefulness across different semantic contexts. In *KitchenWorld* (Fig. 2A, left), the robot must avoid hot appliances (a stove and a kettle) with a fridge as a *distractor object*—an object that is present in the environment but does not affect the human’s preference for how the robot completes the task. In *LavaLand* (Fig. 2A, right), the robot must avoid dangerous terrain (a swamp and lava) with a meadow as a distractor terrain.

Across these two variants, we define seven distinct scenarios that each test a combination of FLEX’s three operations. Each algorithm we test takes a misaligned feature set and a natural language correction as input, and returns an updated feature set. We use the resulting updated feature set and the simulated expert demonstrations to learn a reward. To evaluate how well each of these algorithms performs, we compare our estimated reward to a ground-truth reward that is pre-defined for each scenario using the normalized reward metric described above. We provide additional details about the specific configuration of each scenario in the Appendix.

Method	Add Only	Gen. Only	Mask Only	Add+Gen.	Add+Mask	Mask+Gen.	All Three
ALGAE	1.00 ± .00	.59 ± .02	.95 ± .03	.77 ± .02	.59 ± .02	.63 ± .02	.69 ± .03
MFE	.95 ± .01	.70 ± .02	1.00 ± .00	.75 ± .02	.43 ± .01	.67 ± .02	.83 ± .02
USF	.64 ± .06	.72 ± .03	.90 ± .02	.80 ± .01	.99 ± .01	.78 ± .04	.78 ± .03
FLEX (Ours)	1.00 ± .00	.91 ± .03	1.00 ± .00	1.00 ± .00	1.00 ± .00	1.00 ± .01	.99 ± .01

Table 1: Average normalized reward $\bar{R}$ (mean $\pm$ SE) across seven 2D Navigation scenarios. **Bold** indicates the best-performing method. FLEX maintains strong performance across all scenarios.

Results. We show the results of the average normalized reward for each scenario in Table 1. **FLEX** achieves the highest normalized reward across all seven misalignment scenarios. The single-operation baselines perform well on scenarios that use the edit types they are best-suited for: **ALGAE** performs competitively for the ADD-only scenario and **MFE** performs competitively for the MASK-only scenario. However, their performance substantially degrades when the scenarios require other operations, particularly GENERALIZE. **USF** is highly inconsistent, displaying a high range of mean rewards. Without operation-level scaffolding, its success on a given operation is difficult to anticipate, achieving only 0.64 reward in the simplest ADD ONLY scenario yet 0.99 reward when combining ADD+MASK. Together, these results show that all three operations are necessary to address a range of feature misalignments, and that scaffolding them explicitly makes FLEX more robust at learning users’ preferences than either restricted or unstructured editing.

5.2 Evaluation in Simulated Continuous Control Domain (RQ2)

Our second research question examines sequential feature edits in a continuous control task.

Domain Details. RQ2 evaluates FLEX in a task where a person performs a series of edits, reflective of the intended use of FLEX. RQ2 considers three variants of the 7DoF Franka domain: CoffeeRobot, CandleRobot, and BallRobot. In CoffeeRobot (Fig. 2B, left), the robot must carry a full cup of coffee while avoiding electronic devices (a laptop and a phone) with a wooden bowl as a distractor object. In CandleRobot (Fig. 2B, middle), the robot must carry a lit candle while avoiding flammable objects (a box of tissues and a paper towel roll) with a fork as a distractor object. In BallRobot (Fig. 2B, right), the robot must carry a bowling ball while avoiding fragile objects (a wineglass and a ceramic plate) with a napkin as a distractor object. Rather than evaluating each correction in isolation, we construct five-stage correction sequences (Fig. 3 shows the CoffeeRobot sequence) in which edits stack, with each stage starting from the representation Φ^k produced by the previous stage. Between stages, we change the objects in the environment to expose new misalignments and to show how all three FLEX operations can become necessary over long-term deployment.

Results. Each panel in Fig. 4, “Evaluation on Stage j ,” shows the average normalized reward earned on stage j by the representation Φ^k learned after training on stage k . A good representation should perform well on the stage it was trained on, on earlier stages (no forgetting), and on later stages (generalization). We find that FLEX learns the user’s ground-truth reward function after one or two corrections and maintains high performance across all evaluation stages as corrections accumulate. Baseline methods instead require additional corrections to learn the user’s reward, and their performance degrades as corrections accumulate because they gather spurious features, fail to re-abstract earlier features, and overfit to the stage they were trained on.

5.3 Evaluation with Real Corrections and Demonstrations (RQ3)

Our third research question explores realistic human language and demonstrations.

User Study Setup. We conducted an in-person, within-subjects study with $N = 12$ participants recruited from a university pool (6 women, 6 men). Five participants were novice robot users and self-reported little to no prior familiarity with robots on a 7-point scale. Each participant interacted with three methods (**FLEX**, **ALGAE**, **MFE**). Each method was randomly paired with one of the three Franka task scenarios from Fig. 2B. We randomized and counterbalanced the order of method and task presentation to control for ordering effects. This study was IRB-approved, and we compensated participants \$25 for a one-hour session.

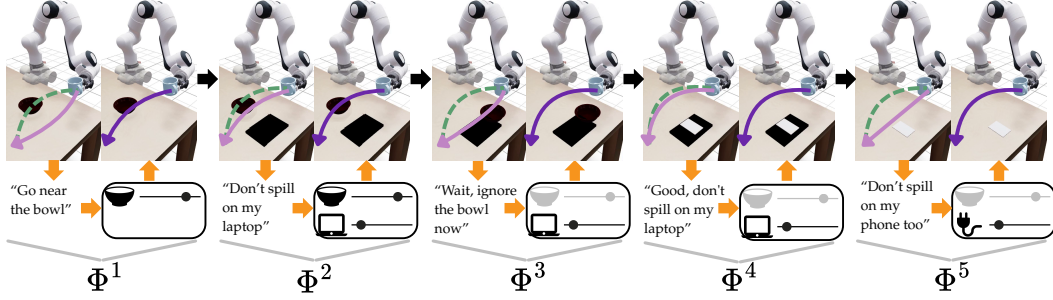


Figure 3: Five-stage Franka correction sequence. At each stage, the user observes a trajectory (lavender) that is misaligned with their ground truth trajectory (green). They then provide a language correction, prompting the robot to update its internal representation Φ^k . Although corrections may immediately realign behavior (purple), environment changes in the subsequent stages can expose new misalignments. Stages are designed so that all three FLEX operations become necessary at some point during the sequence, reflecting realistic long-term deployment conditions.

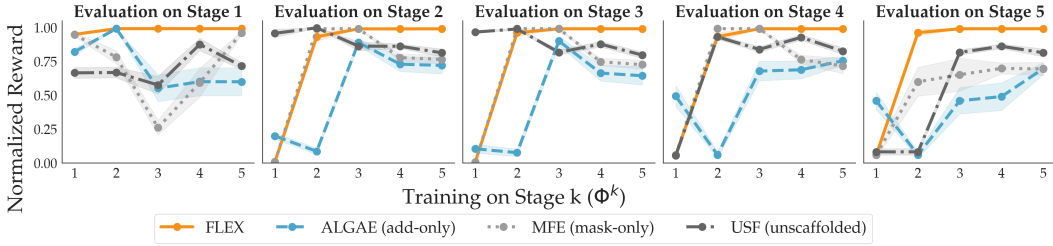


Figure 4: Average normalized reward $\hat{R}$ across five stages of the Franka correction sequences, averaged across task variants. The shaded regions indicate ± 1 SE. The x-axis denotes the iteration of Φ^k , the feature set learned after training on Stage k . FLEX sustains high performance across all stages, while all baseline performances degrade as training stages accumulate.

Before the experiment, participants filled out a consent form and demographics questionnaire. Participants then spent ~ 10 minutes performing a tutorial that familiarized them with the interface’s button and mouse controls (shown in App. 7.5). In the tutorial, each participant completed two practice corrections on a task that was different from the study scenarios.

During the experiment, the user progressed through three experimental blocks. Each block consisted of five sequential correction stages (example in Fig. 3). In each stage, participants observed the robot’s current trajectory alongside a goal trajectory optimized for the ground-truth reward. To keep conditions consistent across participants, each task had a pre-defined ground-truth preference (e.g., avoiding electronics when carrying coffee), and participants gave typed natural language corrections to correct the robot’s behavior so that it matched the pre-defined preference. The robot’s learned reward updated in real-time after each correction, so the “current” trajectory shown in subsequent rounds reflected the participant’s full correction history. After each five-stage block, participants completed a Likert questionnaire to evaluate how well the algorithm performed (items in App. 7.5).

Results. Participants viewed robot behavior in simulation rather than on the physical robot for safety reasons, since some baselines produced erratic arm motions. To evaluate whether corrections collected this way transfer to real-world settings, we separately ran offline learning using participants’ real corrections with a database of expert-collected demonstrations on the physical Franka, shown in Fig. 5. We found that FLEX successfully learns the user’s reward by the end of the sequence. This demonstrates the capability of FLEX to adapt to realistic and imprecise user feedback.

Fig. 6 shows participants’ Likert responses. Using pairwise Wilcoxon signed-rank tests with Holm-Bonferroni correction (all reported p -values are Holm-adjusted), we found that participants rated FLEX significantly higher than ALGAE on task performance ($p = .035$), intent understanding ($p = .006$), and robustness to unintended consequences ($p = .047$). FLEX significantly outperformed MFE on intent understanding ($p = .031$) and marginally outperformed MFE on robustness

to unintended consequences ($p = .074$). FLEX also received the highest ratings for ease of use and collaborativity, though the differences were not significant.

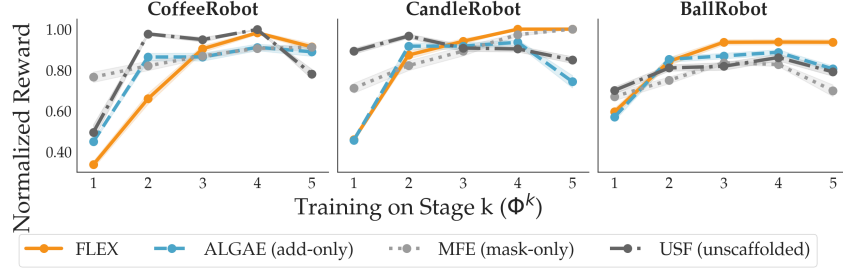


Figure 5: Average normalized reward $\hat{R}$ of representations learned from real human language corrections and demonstrations across the three variants of 7DoF Franka correction sequences. The shaded regions indicate ± 1 SE. The x-axis denotes the iteration of Φ^k , the feature set learned after training on Stage k . FLEX sustains its high performance after learning the user’s reward, while all baseline performances degrade as their feature sets accumulate spurious features.

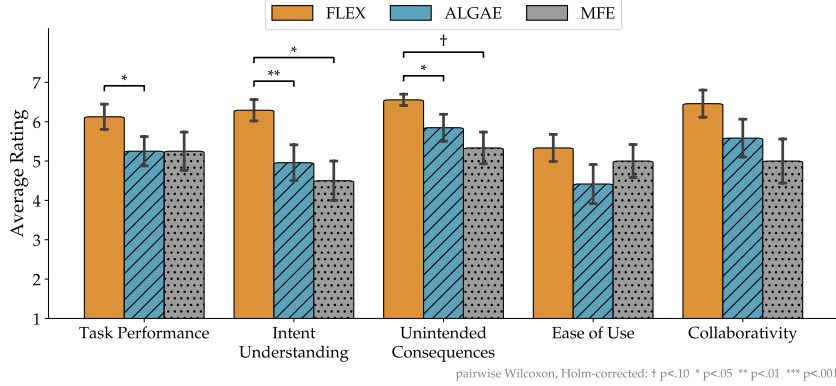


Figure 6: Participants’ Likert ratings ($N = 12$) for **FLEX**, **ALGAE**, and **MFE** across five dimensions: task performance, intent understanding, robustness to unintended consequences, ease of use, and collaborativity. Bars show the average rating (1–7 scale) with error bars denoting ± 1 SE. Participants rated FLEX highest on all five dimensions. Markers denote significant differences between FLEX and each baseline (pairwise Wilcoxon signed-rank tests, Holm-corrected).

6 Conclusion

In this work, we introduced FLEX, a framework that uses language corrections to perform *feature edits* through three distinct operations: ADD new features, GENERALIZE existing features, and MASK irrelevant features. We evaluate FLEX across 2D navigation and 7DoF manipulation domains using simulated and real-world user feedback. Across our experiments, we find that FLEX accurately learns desired robot behaviors under several types of misspecified feature sets. Furthermore, we demonstrate that FLEX transfers effectively from simulated domains to realistic human feedback.

Limitations. FLEX focuses on quasi-static manipulation of rigid objects, where task-relevant features are naturally expressible in language; dynamic tasks or deformable objects may require additional modalities beyond language to specify relevant features. FLEX also relies on LM common-sense reasoning, which easily handles widely held preferences but breaks down for personal or counterintuitive ones. For corrections that jointly encode Φ and θ (e.g., “Stay *even further* away from the laptop”), FLEX ignores the contribution to θ , as we assume ℓ_k is conditionally independent of θ^* , θ_k . Finally, we present a closed-loop user study conducted in simulation, supported by hardware feasibility checks, rather than closed-loop on a physical robot. Future work could address these limitations by grounding corrections in multimodal observations, developing personalized user models to better support unique users, relaxing the conditional independence assumption to handle language that encodes both Φ and θ , and conducting a full closed-loop user study on physical hardware.

Acknowledgments

This work has taken place in the Collaborative Learning and Autonomy Research Lab (CLEAR Lab) at MIT and was partly supported by the MIT Generative AI Impact Consortium and the MIT Schwarzman College of Computing’s Social and Ethical Responsibilities of Computing program. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

- [1] A. Peng, B. Z. Li, I. Sucholutsky, N. Kumar, J. A. Shah, J. Andreas, and A. Bobu. Adaptive language-guided abstraction from contrastive explanations. *arXiv preprint arXiv:2409.08212*, 2024.
- [2] S. Levine, Z. Popovic, and V. Koltun. Feature construction for inverse reinforcement learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/a8f15eda80c50adb0e71943adc8015cf-Paper.pdf.
- [3] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, Z. Xu, D. Sadigh, A. Zeng, and A. Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Conference on Robot Learning*. PMLR, 2023.
- [4] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.
- [5] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
- [6] D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.
- [7] K. Baraka, I. Idrees, T. Kessler Faulkner, E. Biyik, S. Booth, M. Chetouani, D. H. Grollman, A. Saran, E. Senft, S. Tulli, et al. Human-interactive robot learning: Definition, challenges, and recommendations. *ACM Transactions on Human-Robot Interaction*, 15(2):1–31, 2026.
- [8] H. Chen, C. Zhu, Y. Li, and K. Driggs-Campbell. Tool-as-interface: Learning robot policies from human tool usage through imitation learning. In *Conference on Robot Learning*. PMLR, 2025.
- [9] M. Xu, H. Zhang, Y. Hou, Z. Xu, L. Fan, M. Veloso, and S. Song. Dexumi: Using human hand as the universal manipulation interface for dexterous manipulation. *arXiv preprint arXiv:2505.21864*, 2025.
- [10] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.
- [11] C. Tang, A. Xiao, Y. Deng, T. Hu, W. Dong, H. Zhang, D. Hsu, and H. Zhang. Mimicfunc: Imitating tool manipulation from a single human video via functional correspondence. *arXiv preprint arXiv:2508.13534*, 2025.
- [12] M. Li, A. Canberk, D. P. Losey, and D. Sadigh. Learning human objectives from sequences of physical corrections. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2877–2883. IEEE, 2021.
- [13] A. Bajcsy, D. P. Losey, M. K. O’malley, and A. D. Dragan. Learning robot objectives from physical human interaction. In *Conference on robot learning*, pages 217–226. PMLR, 2017.

- [14] A. Bobu, A. Bajcsy, J. F. Fisac, and A. D. Dragan. Learning under misspecified objective spaces. In *Conference on Robot Learning*, pages 796–805. PMLR, 2018.
- [15] N. Dennler, Z. Shi, Y. Tao, A. Bobu, S. Nikolaidis, and M. Matarić. Improving through interaction: Searching behavioral representation spaces with cma-es-ig. *arXiv preprint arXiv:2603.09011*, 2026.
- [16] V. Myers, E. Biyik, N. Anari, and D. Sadigh. Learning multimodal rewards from rankings. In *Conference on robot learning*, pages 342–352. PMLR, 2022.
- [17] S. Lu, M. Zheng, M. C. Fontaine, S. Nikolaidis, and H. Culbertson. Preference-driven texture modeling through interactive generation and search. *IEEE transactions on haptics*, 15(3):508–520, 2022.
- [18] A. Liang, J. Thomason, and E. Biyik. Visarl: Visual reinforcement learning guided by human saliency. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2907–2912. IEEE, 2024.
- [19] H. J. Jeon, S. Milli, and A. D. Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. In *Advances in Neural Information Processing Systems*, 2020.
- [20] N. Dennler, S. Nikolaidis, and M. Matarić. Contrastive learning from exploratory actions: Leveraging natural interactions for preference elicitation. In *2025 20th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 778–788. IEEE, 2025.
- [21] I. Sheidlower, A. Moore, and E. Short. Keeping humans in the loop: Teaching via feedback in continuous action space environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 863–870. IEEE, 2022.
- [22] A. Bobu, A. Peng, P. Agrawal, J. Shah, and A. D. Dragan. Aligning human and robot representations. In *International Conference on Human-Robot Interaction*, pages 42–54. ACM, 2024.
- [23] S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, T. Wang, S. Marks, C.-R. Segerie, M. Carroll, A. Peng, P. Christoffersen, M. Damani, S. Slocum, U. Anwar, A. Siththaranjan, M. Nadeau, E. J. Michaud, J. Pfau, D. Krasheninnikov, X. Chen, L. Langosco, P. Hase, E. Biyik, A. Dragan, D. Krueger, D. Sadigh, and D. Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.
- [24] A. Bobu, A. Bajcsy, J. F. Fisac, S. Deglurkar, and A. D. Dragan. Quantifying hypothesis space misspecification in learning from human-robot demonstrations and physical corrections. *IEEE Transactions on Robotics*, 36(3):835–854, 2020.
- [25] E. Hirota, Z. Yang, A. Hiranaka, M. Jun, J. Tien, S. J. Russell, A. Dragan, and E. Biyik. Active reward learning and iterative trajectory improvement from comparative language feedback. *The International Journal of Robotics Research*, page 02783649251394742, 2025.
- [26] Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh. No, to the right: Online language corrections for robotic manipulation via shared autonomy. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 93–101, 2023.
- [27] M. Hwang, A. Forsey-Smerek, N. Dennler, and A. Bobu. Masked irl: Llm-guided reward disambiguation from demonstrations and language. In *International Conference on Robotics and Automation*, 2026.
- [28] J. A. Nader, D. Lee, N. Dennler, and A. Bobu. Quicklap: Quick language-action preference learning for autonomous driving agents. *Robotics: Science and Systems*, 2025.

- [29] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of machine learning research*, 3(Nov):463–482, 2002.
- [30] A. Bobu, M. Wiggert, C. Tomlin, and A. D. Dragan. Inducing structure in reward learning by learning features. *The International Journal of Robotics Research*, 41(5):497–518, 2022.
- [31] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.
- [32] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.
- [33] M. Tucker, E. Novoseller, C. Kann, Y. Sui, Y. Yue, J. W. Burdick, and A. D. Ames. Preference-based learning for exoskeleton gait optimization. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 2351–2357. IEEE, 2020.
- [34] Y. Koyama, I. Sato, and M. Goto. Sequential gallery for interactive visual design optimization. *ACM Transactions on Graphics (TOG)*, 39(4):88–1, 2020.
- [35] A. Bobu, Y. Liu, R. Shah, D. S. Brown, and A. D. Dragan. Sirl: Similarity-based implicit representation learning. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, pages 565–574, 2023.
- [36] U. H. Lee, V. S. Shetty, P. W. Franks, J. Tan, G. Evangelopoulos, S. Ha, and E. J. Rouse. User preference optimization for control of ankle exoskeletons using sample efficient active learning. *Science Robotics*, 8(83):eadg3705, 2023. doi:10.1126/scirobotics.adg3705. URL <https://www.science.org/doi/abs/10.1126/scirobotics.adg3705>.
- [37] A. Bobu, M. Wiggert, C. J. Tomlin, and A. D. Dragan. Feature expansive reward learning: Rethinking human input. In C. L. Bethel, A. Paiva, E. Broadbent, D. Feil-Seifer, and D. Szafir, editors, *ACM/IEEE International Conference on Human-Robot Interaction, HRI 2021, Boulder, CO, USA, March 8-11, 2021*, pages 216–224. ACM, 2021. doi:10.1145/3434073.3444667. URL <https://doi.org/10.1145/3434073.3444667>.
- [38] A. Forsey-Smerek, J. Shah, and A. Bobu. Learning contextually-adaptive rewards via calibrated features. In L. Baillie, W. D. Smart, M. de Graaf, M. Gombolay, and I. Torre, editors, *Proceedings of the 21st ACM/IEEE International Conference on Human-Robot Interaction, HRI 2026, Edinburgh, Scotland, UK, March 16-19, 2026*, pages 697–705. ACM, 2026. doi:10.1145/3757279.3788669. URL <https://doi.org/10.1145/3757279.3788669>.
- [39] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, J. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 26516–26560, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/70c26937fbf3d4600b69a129031b66ec-Paper-Conference.pdf.
- [40] I. Sucholutsky, L. Muttenthaler, A. Weller, A. Peng, A. Bobu, B. Kim, B. C. Love, C. J. Cueva, E. Grant, I. I. A. Groen, J. Achterberg, J. B. Tenenbaum, K. M. Collins, K. L. Hermann, K. Oktar, K. Greff, M. N. Hebart, N. Cloos, N. Kriegeskorte, N. Jacoby, Q. Zhang, R. Marjeh, R. Geirhos, S. Chen, S. Kornblith, S. Rane, T. Konkle, T. P. O’Connell, T. Unterthiner, A. K. Lampinen, K. Müller, M. Toneva, and T. L. Griffiths. Getting aligned on representational alignment. *Trans. Mach. Learn. Res.*, 2025, 2025. URL <https://openreview.net/forum?id=Hiq71Uh4Yn>.
- [41] A. Peng, I. Sucholutsky, B. Z. Li, T. R. Sumers, T. L. Griffiths, J. Andreas, and J. A. Shah. Learning with language-guided state abstractions. In *International Conference on Learning Representations*, 2024.

- [42] F. Amin, N. Dennler, and A. Bobu. Gift: Generalizing intent for flexible test-time rewards. *2026 IEEE International Conference on Robotics and Automation (ICRA)*, June 2026.
- [43] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, Q. Vuong, V. Vanhoucke, H. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sankei, G. Salazar, M. S. Ryoo, K. Reymann, K. Rao, K. Pertsch, I. Mordatch, H. Michalewski, Y. Lu, S. Levine, L. Lee, T.-W. E. Lee, I. Leal, Y. Kuang, D. Kalashnikov, R. Julian, N. J. Joshi, A. Irpan, B. Ichter, J. Hsu, A. Herzog, K. Hausman, K. Gopalakrishnan, C. Fu, P. Florence, C. Finn, K. A. Dubey, D. Driess, T. Ding, K. M. Choromanski, X. Chen, Y. Chebotar, J. Carbajal, N. Brown, A. Brohan, M. G. Arenas, and K. Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 06–09 Nov 2023. URL <https://proceedings.mlr.press/v229/zitkovich23a.html>.
- [44] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.
- [45] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. R. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025. URL <https://openreview.net/forum?id=vlhoswksB0>.
- [46] H. Yin, H. Wei, X. Xu, W. Guo, J. Zhou, and J. Lu. Gc-vln: Instruction as graph constraints for training-free vision-and-language navigation. In *Conference on Robot Learning*. PMLR, 2025.
- [47] Y. Wu, R. Tian, G. Swamy, and A. Bajcsy. From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment. *arXiv preprint arXiv:2502.01828*, 2025.
- [48] L. X. Shi, Z. Hu, T. Z. Zhao, A. Sharma, K. Pertsch, J. Luo, S. Levine, and C. Finn. Yell at your robot: Improving on-the-fly from language corrections. In *Robotics: Science and Systems*, 2024.
- [49] H. Liu, A. Chen, Y. Zhu, A. Swaminathan, A. Kolobov, and C.-A. Cheng. Interactive robot learning from verbal correction. *arXiv preprint arXiv:2310.17555*, 2023.
- [50] M. Abugurain and S. Park. Integrating disambiguation and user preferences into large language models for robot motion planning. *arXiv preprint arXiv:2404.14547*, 2024.
- [51] L. Santos, Z. Li, L. Peters, S. Bansal, and A. Bajcsy. Updating robot safety representations online from natural language feedback. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7778–7785. IEEE, 2025.
- [52] B. Zhang and H. Soh. Large language models as zero-shot human models for human-robot interaction. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7961–7968, 2023. doi:10.1109/IROS55552.2023.10341488.
- [53] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.
- [54] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, and D. Sadigh. Distilling and retrieving generalizable knowledge for robot manipulation via language corrections. In

2024 IEEE international conference on robotics and automation (ICRA), pages 15172–15179. IEEE, 2024.

- [55] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.
- [56] E. T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

7 Appendix

7.1 LM Prompts

Add: Generate Feature

This is an environment config: `{environment.description}`

The following is an instruction for a task that the agent needs to complete in this environment. `instruction = {instruction}`

The following is a history of corrections that the agent has been provided. `history = {corrections history}`

The following is the most recent addendum to the original instruction after the agent asked for more clarification about the task. `addendum = {correction}`

This is the current list of features that we already know: `{ $\Phi^k$ }`.

The feature list contains computable numeric values (e.g. "distance from agent to object X") derived from scene state. These features will be used by a downstream model that has NO access to the instruction, corrections, or addendum text -- only to the feature values.

Consider adding a new feature to this list to explain all parts of the instruction and addendum. Carefully consider all parts of the addendum, especially if multiple objects are mentioned.

1. If adding a new feature would not provide any new information that helps explain the instruction or addendum -- for example, if the existing features already capture all relevant objects, properties, or relationships -- do not propose any additional feature and return "NO NEW FEATURE". Do NOT propose a feature merely because an object exists in the environment or because it could be generally relevant in other contexts -- only if the instruction or addendum clearly implies that object or property matters for the task. If the existing features already cover all relevant entities or relationships mentioned or implied by the instruction and addendum, respond exactly with: NO NEW FEATURE

Add at most one sentence of explanation with specific references to the addendum

2. Only propose a new feature if it clearly adds missing information needed to explain the instruction or addendum. You may provide the name of the specific feature in this environment that explains the instruction and addendum. If there are multiple of the same type of object, please give the average distance to all objects of that type.

Do not mix multiple types or objects in the feature. Please specify a feature (only one feature e.g. if there are multiple possible explanations choose only one most likely object) that we can write code to compute from the position of the agent to other objects in the scene. Please give a different feature in your answer (e.g. if distance to goal feature is already in the list, do not give distance to goal again). Ground your answer in visible objects in the scene, and give an answer (only 1 specific answer (e.g. distance to heavy objects)) that we can compute relative to the position of the agent.

Follow your answer with one newline and at most one sentence of explanation with specific references to the addendum.

Generalize: Extract Intent

This is an environment config: {environment.description}

The following is an instruction for a task that the agent needs to complete in this environment. instruction = {instruction}

The following is a history of corrections that the agent has been provided. history = {corrections history}

The following is the most recent addendum to the original instruction after the agent asked for clarification about the task. addendum = {correction}

Extract the intent behind the human's addendum in at most 2 sentences, explaining the purpose or reasoning behind the addendum. Focus on what the agent needs to achieve and why the human may have said this particular addendum, including any important context or conditions that affect the task. Do not base the explanation on specific locations, as this may change in later scenarios.

Generalize: Generalize Feature Names

A {"robot"/"robot arm"} is in a {"2D gridworld"/"3D tabletop"} environment with the following objects: {environment.description}

The following is an instruction for a task that the agent needs to complete in this environment. instruction = {instruction}

The following is a history of corrections that the agent has been provided. history = {corrections history}

The following is an addendum to the original instruction after the agent asked for clarification about the task. addendum = {correction}

{intent (from "Generalize: Extract Intent")}

The robot has the following set of features: $\{\Phi^k\}$

For each feature, please determine whether the intent is semantically relevant for that feature.

If yes (and only if yes), generalize the feature name using properties of objects in the environment that directly reflect the semantic intent of the instruction or addendum, without creating spurious correlations.

Do not generalize based on properties that are not directly connected to the instruction intent. Do not generalize to multiple properties. Feature names may remain unchanged if they already sufficiently capture the semantic intent. Feature names may remain unchanged if they are already sufficiently general. Do not simply rename a feature that points to the same property. Features that share the same semantic relevance should be generalized to the same name. Only generalize using object properties explicitly listed in the environment.

Return a newline-delimited list of the original feature names, followed by "->", followed by either NO CHANGE or a generalized feature name. For example

```
feature_a -> feature_letter
feature_b -> feature_letter
feature_letter -> feature_letter
feature_1 -> NO CHANGE
```

The following prompt applies once per feature ϕ_i in the current representation Φ^k , as this yielded more consistent results than a prompt asking for the full mask at once.

Mask: Compute Mask Element m_i

This is an environment config: `{environment_description}`

The following is an instruction for a task that the agent needs to complete in this environment. `instruction = {instruction}`

The following is a history of corrections that the agent has been provided. `history = {corrections history}`

The following is an addendum to the original instruction after the agent asked for clarification about the task. `addendum = {correction}`

The current available features are: `{ $\Phi^k$ }`

We are evaluating whether `{ $\phi_i$ }` is part of the minimal set of features needed for the agent to compute its reward given that it has NO access to the instruction, corrections, or addendum text -- only to the feature values. Please return either a 1 or 0 for the following feature: `feature = { $\phi_i$ }`, where 1 means that the agent uses this feature to calculate its reward and 0 means that the agent does not need this feature when calculating its reward.

Please reason step by step. In particular, consider whether the feature provides precise and relevant information for the task. Also consider whether the feature might lead the agent to treat multiple objects the same way, even if only one of them is relevant to the instruction.

For example, for the instruction go to a book and the features ["book_dist", "table_dist", "chair_dist"], the minimal set would include only the features that refer specifically to the object mentioned in the instruction.

Provide your answer as ONLY a single integer. Do not include any surrounding explanation or text.

Example Prompt Inputs

`environment_description`: See "2D Navigation Example Environment Description" and "Franka 7DoF Example Environment Description" below

`instruction`: "Move the full coffee cup"

`corrections history`: ["Maybe try to swerve near the bowl?", "Don't swing the coffee so much.", "Don't go near the laptop."]

`correction`: "Ignore what I said about the bowl, just avoid the laptop"

`$\Phi^k$` : [distance_to_bowl, coffee_orientation, laptop_avoidance]

Add and Generalize: Generate Code

Based on the feature identified above, write a Python method that takes as input: `{function inputs}`

In your method, you may use `get_objects_with_property(property: str, environment: dict, robot_env)`, which takes a property name, the environment dict, and the robot environment object, and returns a list of `[x, y, z]` positions for all objects in the environment that have that property, type, or name. You must import this function from `flexible_features.utils.env_utils`.

`{example distance feature}`

`{example orientation feature}`

Only include one feature (e.g. `distance_to_book`) and not a general feature (e.g. `distance_to_object`) in your answer. Do not use objects or properties that do not exist in this environment. Please write code that is generalizable to new environments (for example, the new environment may have objects in different positions than the current environment)

Output your answer in form:

Feature name: `"feature_name"`

Feature code:

```
'''python
import ...
def feature_name(environment, traj, robot_env):
    feat_val = np.zeros((traj.shape[0], 1))
    for pos in ...:
        ...
    feat_val += ... # (K, 1)
    return feat_val
'''
```

Please write code for all the features you have identified, and ensure the code is executable. Make sure your code is generalizable to new environments. Don't assume you are given any other information about the environment other than what is explicitly specified in the environment config. Do not include any surrounding explanation or text.

2D Navigation System Prompt

An agent is in a 2D gridworld environment of size 9x9. (0,0) is the top left corner and (8, 8) is the bottom right corner (note this is not a typical coordinate grid).

The environment config has the following attributes:

`X (int)`: width of the environment

`Y (int)`: height of the environment

`starts (List[Tuple[int]])`: starting coordinate of the agent

`goals_pos (List[Tuple[int]])`: coordinate position of the goal

`goals_color (str)`: color of the goal

`goals_type (str)`: object type of the goal

`objects{1-3}_pos (List[Tuple[int]])`: position(s) of other objects in the environment. This will be empty if there is no object in that slot.

`objects{1-3}_color (str)`: color of the corresponding object at the position `objects{1-3}`. This will be empty if there is no object in that slot.

`objects{1-3}_type (str)`: object type of the corresponding object at the position `objects{1-3}`. This will be empty if there is no object in that slot.

Franka 7DoF System Prompt

A 7DoF Franka robot arm is carrying a full coffee cup / a lit candle / a heavy bowling ball on a tabletop environment. The arm is controlled via a PyBullet simulator.

The environment config has the following attributes:

```
objects:
  object1:
    urdf: object1.urdf
    position: [x, y, z]
    properties: [object properties]
  ...
tabletop: z # z coordinate plane of the tabletop
```

2D Navigation Example Environment Description

```
env:
type: gridrobot
X: 9
Y: 9
starts: [[0,0]]
goals_pos: [[8,8]]
goals_color: 'green'
goals_type: 'cross'
objects1_pos: [[0,8]]
objects1_color: 'blue'
objects1_type: 'fridge'
objects1_property: ['fridge', 'light_source', 'icy', 'stationary', 'appliance']
objects2_pos: [[8,0]]
objects2_color: 'red'
objects2_type: 'stove'
objects2_property: ['stove', 'heat_source', 'metal', 'stationary', 'appliance']
objects3_pos: [[4,4]]
objects3_color: 'orange'
objects3_type: 'kettle'
objects3_property: ['kettle', 'portable', 'heat_source', 'appliance']
```

Franka 7DoF Example Environment Description

```
env:
objects:
  laptop:
    urdf: laptop.urdf
    position: [-0.6, -0.5, 0.64]
    properties: ["laptop", "rectangular", "metal", "electronic"]
  bowl:
    urdf: bowl.urdf
    position: [-0.6, -0.8, 0.9]
    properties: ["bowl", "wooden", "lightweight", "round", "dishware"]
  phone:
    urdf: phone.urdf
    position: [-0.6, -0.5, 0.64]
    properties: ["phone", "lightweight", "metal", "electronic"]
```

2D Navigation: Feature Function Inputs

Based on the feature description above, write a Python method that takes as input:

- * the ground truth environment features specified in the config (dict)
- * a trajectory (List[float]) representing the agent's position over time as a list of x,y coordinates (e.g. [0.0 0.0 0.0 1.0] means the agent

started at (0,0) and moved to (0,1). The trajectory should be of length 34 (representing 17 timesteps where each timestep has a x,y coordinate for the agent's position)

and outputs:

- * the cumulative value for the feature over the course of the entire trajectory (averaged over multiple objects if present) (array with dtype=np.float64).

7DoF Franka: Feature Function Inputs

Based on the feature identified above, write a Python method that takes as input:

- * the ground truth environment features specified in the config (dict)
- * a batch of trajectories (dim KxTxN)
- * the robot environment object (robot_env)

and outputs:

- * the cumulative value for the feature over the course of all of the trajectories, shape (K, 1)

Each trajectory batch is a KxTxN tensor where:

K is the number of trajectories in the batch

T is the number of timesteps (21)

N is the state dimension

Each state in the timestep is an N-dimensional vector laid out as follows:

waypt[:7] -- joint angles

orientations.flatten() -- robot link orientations

coords.flatten() -- robot link xyz coordinates

The end-effector xy position is at indices 88:90 of each state vector.

2D Navigation: Example Distance Feature

For example, for "agent distance to book objects" feature, you may write code

```
'''from flexible_features.utils.env_utils import get_objects_with_property

def agent_distance_to_book_objects(environment, trajectory, r=3.0):
# environment is the ground truth environment

    # returns the position(s) of the object(s) with property "book"
    book_objects_pos = get_objects_with_property("book", environment)

    # initialize the feature value
    book_objects_feat = np.zeros(1)

    # iterate through the trajectory
    for i in range(0, len(trajectory) - 1, 2):

        # iterate through the book objects
        for book_object in book_objects_pos:

            # calculate the dist between the agent and the book object (x)
            dx = abs(trajectory[i] - book_object[0])

            # calc the distance between the agent and the book object (y)
            dy = abs(trajectory[i + 1] - book_object[1])
            dist = np.linalg.norm([abs(dx), abs(dy)])

            # Cap distance at r (saturate)
```

```

        bump_dist = min(dist, r)
        book_objects_feat += bump_dist

# return the average distance to the book object(s) for each trajectory
feature_value = book_objects_feat / len(book_objects_pos)
return feature_value # array with dtype=np.float64'''

```

7DoF Franka: Example Distance Feature

For example, for "distance to book" feature, you may write code:

```

'''import numpy as np
from flexible_features.utils.env_utils import get_objects_with_property

def distance_to_book(environment, trajs, robot_env, margin=0.4):
    """
    Computes cumulative dist reward from end-effector to book in xy coords.
    Params:
        environment -- dict with 'objects' mapping object names to their data
        trajs -- np.ndarray of shape (K, T, N), batch of K trajectories
        robot_env -- robot environment object for live object position lookup
    """

    book_positions = get_objects_with_property("book", environment, robot_env)
    feat_val = np.zeros((trajs.shape[0], 1))
    for pos in book_positions:

        book_xy = np.asarray(pos[:2])
        ee_xy = trajs[:, :, 88:90]
        dists = np.linalg.norm(ee_xy - book_xy, axis=2) - margin
        feat_val += np.sum(-((dists < 0) * dists), axis=1, keepdims=True)

    return feat_val / max(1, len(book_positions)) '''

feat_val is a (K, 1) array representing the average over matched objects of
the summed per-timestep distance reward across the trajectory batch.

```

7DoF Franka: Example Orientation Feature

Another example, for "orientation of coffee" feature, you may write code:

```

'''import numpy as np

def coffee_orientation_feature(environment, trajs, robot_env):

    """Computes the coffee orientation feature by checking if the end
    effector is oriented vertically."""

    # Rotation matrix of link6 is stored as a flattened 3x3 at indices 61:70
    link6_rot_flat = trajs[:, :, 61:70]
    K, T, _ = link6_rot_flat.shape
    link6_rot = link6_rot_flat.reshape(K, T, 3, 3)
    # The z-component of the EE x-axis indicates how much it points downward
    # Pulled directly from rotation matrix (no additional transform needed)
    ee_x_axis_z_component = link6_rot[:, :, 2, 0]
    return np.sum(1 - ee_x_axis_z_component, axis=1, keepdims=True)
'''

```

7.2 Baseline Implementations

MFE: Get Exhaustive Feature List

This is an environment config: `{environment.description}`

Use this to give specific feature(s) in that environment that explains what are relevant properties in the environment. If there are multiple of the same object, please give the average distance to all objects of that type.

Please specify feature(s) that we can write code to compute from the position of the agent to other objects in the scene. This is the current list of features that we already know: `{ $\Phi^k$ }`. Please give different features in your answer (e.g. if distance to goal feature is already in the list, do not give distance to goal again).

Please give your confidence for each feature between 0 and 1. You may ground your answer in visible objects in the scene, and give answers that we can compute relative to the position of the agent.

USF: Get Updated Feature List

This is an environment config: `{environment.description}`

The following is an instruction for a task that the agent needs to complete in this environment. `instruction = {instruction}`

The following is a history of corrections that the agent has been provided. `history = {corrections history}`

The following is the most recent addendum to the original instruction after the agent asked for clarification about the task. `addendum = {correction}`

Based on this, create a feature list that contains computable numeric values (e.g. "distance from agent to object X") derived from scene state. These features will be used by a downstream model that has NO access to the instruction, corrections, or addendum text -- only to the feature values.

If there are multiple of the same object, please give the average distance to all objects of that type. Please specify feature(s) that we can write code to compute from the position of the agent to other objects in the scene. Every feature in your answer must be unique (e.g. if one feature is distance to goal, do not create another feature that is distance to goal again). Please give your confidence for each feature between 0 and 1. Ground your answer in visible objects in the scene, and give answers that we can compute relative to the position of the agent.

MFE uses an equivalent masking prompt as above. MFE and USF use an equivalent code generation prompt as above. Our implementation and prompts for **ALGAE** exactly follow [1].

7.3 RQ1 Additional Details

We evaluate the contribution of each operation by constructing seven scenarios that represent all combinations of ADD, GENERALIZE, and MASK. Table 2 lists the initial features and expected outputs for each scenario in the KitchenWorld domain with the correction "Don't burn yourself on the stove." For example, in the MASK+GENERALIZE scenario, the initial features measure distance to the goal, the stove, and the fridge. The correction indicates that the robot should not care about the fridge at all, so the fridge feature should get MASKed away. Since a stove-related feature already exists, the ADD operation is not necessary. However, the existing feature should be GENERALIZED to apply to all dangerous heat sources.

Operations	Initial Rep	Expected Actions
Add	goal	add stove
Generalize	goal, stove	generalize stove → heat source
Mask	goal, stove, fridge	mask fridge
Add + Generalize	goal, fridge	add stove, generalize stove → heat source
Add + Mask	goal, fridge	add stove, mask fridge
Mask + Generalize	goal, stove, fridge	mask fridge, generalize stove → heat source
All Three	goal, fridge	add stove, mask fridge, generalize stove → heat source

Table 2: 2D Navigation misalignment scenarios. Each scenario requires a distinct subset of FLEX’s three operations. The initial features determine which operations are needed to recover the true reward, and the expected actions describe the correct response to the language correction “Don’t burn yourself on the stove.” All features are Euclidean distances to the named object or location.

To quantify per-operation reliability, we ran ADD, GENERALIZE, and MASK in isolation, outside the full FLEX pipeline, on all seven 2D Navigation scenarios. We used 15 additional seeds, separate from the five evaluation seeds, yielding 105 trials. Each operation received the scenario’s initial misaligned feature set and natural language correction as input. A trial counts as an error if the operation’s output differs from the expected edit, either because it (i) applied an edit when none was warranted (or omitted one that was), or (ii) applied the edit but produced the wrong feature(s). Error rates were 8.6% for ADD (9/105), 3.8% for MASK (4/105), and 5.7% for GENERALIZE (6/105).

7.4 RQ2 Additional Details

To validate FLEX performance in more realistic settings, we design a sequence of five object configurations in the Franka 7DoF tabletop manipulation domain through which each operation of FLEX becomes necessary. Fig. 3 lists an example sequence of corrections in the CoffeeRobot task variant. Each stage of the sequence depicts the object configuration, an incorrect robot trajectory (lavender), the user’s ground-truth trajectory (green), an example language correction, a potential representation update, and the resulting aligned trajectory for that specific configuration (purple). Throughout the example CoffeeRobot sequence, the user’s ground-truth preference is to avoid any damage from spilling coffee on their laptop, phone, calculator, or other electronic device.

Stage 1: The user only sees a bowl and, mistakenly believing it to be relevant, provides the correction “Go near the bowl.” In response, the robot ADDs a new feature related to bowl proximity, establishing a use case where an incorrect or irrelevant feature enters the robot’s representation.

Stage 2: The user sees a laptop in addition to the bowl and wants the robot to keep the coffee far away from the laptop with the correction “Don’t spill on my laptop.” To accommodate the user’s stated preference, the robot must ADD a new feature related to laptop avoidance.

Stage 3: The user sees the bowl overlap with the laptop. However, the robot’s previously learned bowl proximity feature becomes a spurious correlation, distorting the trajectory by simultaneously pulling it away from the laptop and towards the bowl. Since the user truly doesn’t care about the bowl, they give the correction “Wait, ignore the bowl now,” causing the robot to MASK away this now-irrelevant feature.

Stage 4: The user sees a phone overlap with the laptop, and desires to keep the coffee far away from both devices. Even with only a laptop avoidance feature, the robot maintains aligned behavior, because a trajectory that veers away from the laptop also veers away from the overlapping phone. Visually, the user believes that little to no correction is necessary: “Good, don’t spill on my laptop.” Although the robot’s representation does not change, this stage demonstrates that behavioral alignment can mask representational misalignment. The robot *appears* to be avoiding both the laptop and the phone, but its internal representation reveals that the robot does not consider the phone at all. This hidden misalignment will only become apparent when the laptop is removed.

Stage 5: Now the laptop is removed, exposing the misalignment from Stage 4. The robot’s laptop avoidance feature is ineffective with only a phone present. The user provides a correction “Don’t

spill on my phone too,” and the robot must GENERALIZE its learned laptop avoidance feature to all electronics devices. Only then can the robot maintain aligned behavior in environments with only a laptop, a phone, or any other electronic device.

Together, these five stages show that behavioral alignment is necessary but not sufficient to guarantee robustness to new configurations. Despite a correction producing aligned behavior on every stage, a new configuration quickly reveals further misalignments that must be combated through a combination of ADD, GENERALIZE, and MASK operations.

7.5 RQ3 Additional Details

User Study Interface. Participants viewed the robot trajectories through a simulated interface consisting of a main screen showing the scene and robot, and a side panel containing the available controls. They could replay the goal and current trajectories as many times as they wished, and typed their natural language corrections for each stage. To inspect the scene from different angles, they could switch between three preset camera views (“Top Down,” “Side,” and “Front”), and the tutorial additionally taught them to rotate, pan, and zoom the scene with the computer mouse. Although the drawn-out trajectories were not visible to participants, we visualize them here to illustrate how the goal and current trajectories can differ.

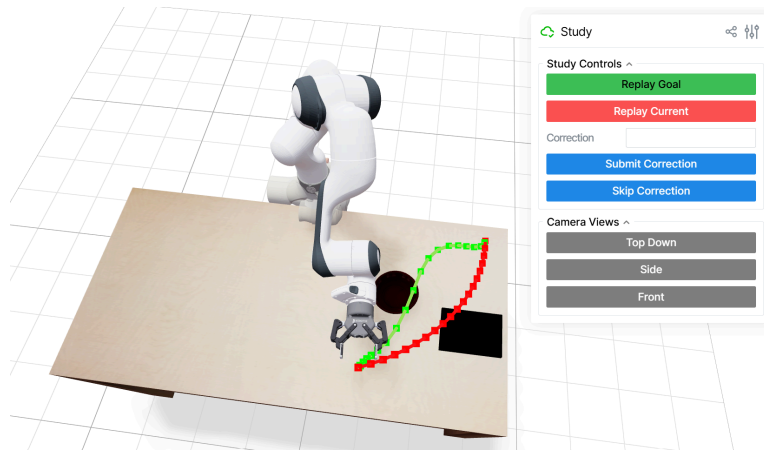


Figure 7: User study interface. The side panel includes buttons to replay the ground truth (green) and current (red) trajectories, as well as pre-set camera views labeled “Top Down,” “Side,” and “Front.” Participants enter their natural language corrections in the text field on the side panel. Participants were also instructed on how to use the computer mouse to rotate, pan, and zoom the scene.

Likert Items. Please rate your agreement with each statement about the robot learning via Algorithm 1/2/3.

Strongly disagree (1) to Strongly agree (7)

1. Collaborativity Scale

- The robot did not collaborate with me to complete the task.
- The robot worked with me to complete the task.

2. Ease of Use Scale

- The robot required minimal corrections.
- I had to keep correcting the robot.

3. Task Performance Scale

- The robot accomplished the task in the way I wanted.
- The robot was NOT able to complete the task correctly.

4. Understanding of User Intent Scale

- The robot understood how I wanted the task to be done.
- The robot learned from my corrections.

5. Robustness to Unintended Consequences Scale

- The robot wrongly updated its understanding about aspects of the task I did NOT want to change.
- After I adjusted the robot, it continued to perform the other parts of the task correctly.
- After I adjusted the robot, it incorrectly updated parts of the task that were already correct.

Please write any notes or observations you have about Algorithm 1/2/3.

What preferences were you trying to get the robot to learn in Task 1/2/3?

Please write any observations you have about the differences between Algorithms 1, 2, and 3.