Journal of Artificial Intelligence Research 74 (2022) 143-177 Submitted 12/2021; published 05/2022

Avoiding Negative Side Effects of Autonomous Systems in the Open World

Sandhya Saisubramanian SANDHYA.SAI@OREGONSTATE.EDU
School of Electrical Engineering and Computer Science
Oregon State University
110 SW Park Terrace, Corvallis, OR 97331 USA

Ece Kamar ECKAMAR@MICROSOFT.COM
Microsoft Research
14865 NE 36th St, Redmond, WA 98052 USA

Shlomo Zilberstein SHLOMO@CS.UMASS.EDU
College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive, Amherst, MA 01003 USA

Abstract

Autonomous systems that operate in the open world often use incomplete models of their environment. Model incompleteness is inevitable due to the practical limitations in precise model specification and data collection about open-world environments. Due to the limited fidelity of the model, agent actions may produce *negative side effects* (NSEs) when deployed. Negative side effects are undesirable, unmodeled effects of agent actions on the environment. NSEs are inherently challenging to identify at design time and may affect the reliability, usability and safety of the system. We present two complementary approaches to mitigate the NSE via: (1) learning from feedback, and (2) environment shaping. The solution approaches target settings with different assumptions and agent responsibilities. In learning from feedback, the agent learns a penalty function associated with a NSE. We investigate the efficiency of different feedback mechanisms, including human feedback and autonomous exploration. The problem is formulated as a multi-objective Markov decision process such that optimizing the agent's assigned task is prioritized over mitigating NSE. A slack parameter denotes the maximum allowed deviation from the optimal expected reward for the agent's task in order to mitigate NSE. In environment shaping, we examine how a human can assist an agent, beyond providing feedback, and utilize their broader scope of knowledge to mitigate the impacts of NSE. We formulate the problem as a human-agent collaboration with decoupled objectives. The agent optimizes its assigned task and may produce NSE during its operation. The human assists the agent by performing modest reconfigurations of the environment so as to mitigate the impacts of NSE, without affecting the agent's ability to complete its assigned task. We present an algorithm for shaping and analyze its properties. Empirical evaluations demonstrate the trade-offs in the performance of different approaches in mitigating NSE in different settings.

1. Introduction

Deploying artificial intelligence (AI) systems requires complex design choices to support safe operation in the open world. During the design and initial testing, the system designer typically ensures that the agent's model includes all the necessary details relevant to its

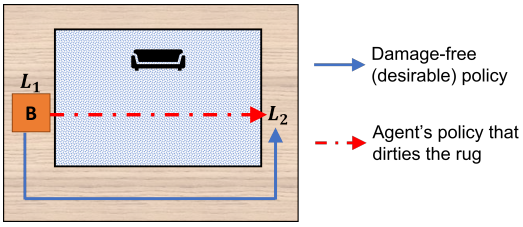


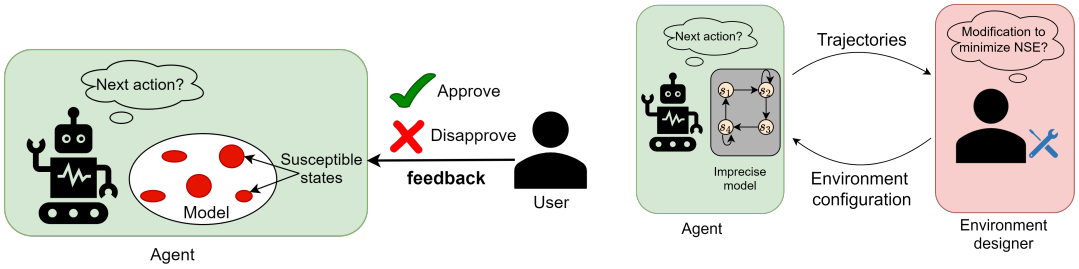
Figure 1: Illustration of a negative side effect in the boxpushing domain: A policy based on the agent's incomplete model, which overlooks the impact of pushing the box over the rug, dirties the rug (shaded area).

assigned task. Inevitably, some details in the environment that are unrelated to the agent's primary objective may be ignored. The incompleteness of any given model—handcrafted or machine acquired—is inevitable due to practical limitations in model specification (the ramification and qualification problems) (Dietterich, 2017) and limited information that may be available during the design phase (Saisubramanian, Wray, Pineda, & Zilberstein, 2019), including unanticipated domain characteristics and cultural differences among the target user and development team (Saisubramanian, Zilberstein, & Kamar, 2022). Due to the limited fidelity of its reasoning model, an agent's actions may create *negative side effects* (NSEs) during policy execution in the deployed environment (Amodei, Olah, Steinhardt, Christiano, Schulman, & Mané, 2016; Hibbard, 2012; Saisubramanian et al., 2022; Saisubramanian, Kamar, & Zilberstein, 2020b; Saisubramanian & Zilberstein, 2021).

Negative side effects are undesired, unmodeled effects of an agent's actions that occur in addition to the agent's intended effects when operating in the open world. Different types of model incompleteness lead to different types of negative side effects, whose severity ranges from mild events that are tolerable to catastrophic events. Mitigating NSEs is critical to improve the safety and reliability of deployed AI systems. However, it is practically impossible to identify all the NSEs at design time since agents are deployed in varied settings. Deployed agents often do not have any *prior* knowledge about NSEs, in which case this information is gathered after deployment.

In this paper, we focus on NSEs that are undesirable, but not catastrophic or prohibitive. That is, we consider NSEs that do not affect the agent's ability to complete its assigned task. For example, consider an agent that is required to push a box B as quickly as possible from location L_1 to location L_2 (Figure 1). The agent's model includes all the details essential to optimize the time taken to push the box, including the reward for pushing the box and the associated transition dynamics. However, details such as the impact of pushing the box over a rug may not be included in the model if the issue is overlooked during system design. Consequently, when operating based on this model, the agent may push the box over the rug, dirtying the rug as a side effect. *How can we mitigate the negative side effects, when the agents are unaware of the side effects and the associated penalties?*

Interest in minimizing negative side effects has been growing among AI researchers. Existing approaches to mitigate NSEs accomplish that by redesigning the reward function for the agent's primary objective (Hadfield-Menell, Milli, Abbeel, Russell, & Dragan, 2017),



(a) Mitigating NSE by learning from user feedback

(b) Mitigating NSE via environment shaping

Figure 2: An illustration of our proposed complementary solution approaches with varying nature of autonomy to mitigate NSE.

constraining the agent's actions (Zhang, Durfee, & Singh, 2018, 2020), minimizing deviations from a baseline state (Krakovna, Orseau, Martic, & Legg, 2019), or maximizing the attainable utility of auxiliary objectives and completion of future tasks (Turner, Hadfield-Menell, & Tadepalli, 2020b; Turner, Ratzlaff, & Tadepalli, 2020a; Krakovna, Orseau, Ngo, Martic, & Legg, 2020). All these approaches target settings with avoidable NSEs and assume that the agent has some prior knowledge about the cause and occurrence of NSEs. However, these assumptions are often violated when deploying AI systems in the open world. Furthermore, in situations where the primary objective is prioritized, the user may be willing to trade off some NSE for solution quality. For example, if the agent takes ten times longer to push the box so as to avoid the rug area, the user may be willing to tolerate some dirt on the rug instead. Prior works, however, do not offer a principled approach to balance this trade-off and do not provide bounded performance guarantees.

To mitigate NSEs in settings where the agent does not have prior knowledge about the side effects of its actions, we propose two complementary approaches that have different properties and underlying assumptions: (1) learning from feedback, and (2) environment shaping. Figure 2 illustrates these two approaches.

Learning from feedback We first propose a multi-objective approach that exploits the reliability of the existing model with respect to the primary objective, while allowing a deployed agent to learn to avoid NSE as much as possible. The agent's model is augmented with a secondary reward function that represents the penalty for NSE. The problem is formulated as a multi-objective Markov decision process with lexicographic reward preferences (LMDP) and slack (Wray, Zilberstein, & Mouaddib, 2015). The agent's primary objective is to complete its assigned task, while the secondary objective is to minimize NSE. The slack denotes the acceptable deviation from the optimal expected reward of its primary objective so as to minimize NSE. A lexicographic approach is adopted because (1) we target settings in which optimizing the reward associated with agent's assigned task is prioritized over mitigating NSE, (2) the reward for the primary objective and the penalty for NSE may have different units, such as time taken to push a box and the cost of cleaning a rug, and (3) alternative scalarization methods require non-trivial parameter tuning and may

not offer performance guarantees with respect to the primary objective (Roijers, Vamplew, Whiteson, & Dazeley, 2013).

Our solution framework utilizes a three-step approach to detect and mitigate NSE: (1) the agent collects data about NSE penalty through a feedback mechanism, (2) the agent learns a predictive model of the penalty for NSE that generalizes the available data, and (3) the agent replans using a model augmented with the learned NSE penalty. We investigate the efficiency of different feedback approaches to learn about the NSE, including oracle feedback and agent exploration. The oracle feedback typically represents some type of human feedback such as penalty signals, action approval, correcting agent's behavior, or demonstrations of correct ways to perform the task. In learning by exploration, the agent collects data about the NSE by autonomously exploring the environment.

Environment shaping Our second approach is *environment shaping* for deployed agents: a process of applying simple modifications to the current environment to make it more agent-friendly and minimize the occurrence of NSE. This solution approach is motivated by the observation that agents often operate in environments that are configurable. Real world examples show that modest modifications can substantially improve agents' performance. For example, Amazon improved the performance of its warehouse robots by optimizing the warehouse layouts, covering skylights to reduce glare, and repositioning the air conditioning units to blow out to the side (Simon, 2019). Recent studies show the need for infrastructure modifications such as traffic signs with barcodes and redesigned roads to improve the reliability of autonomous vehicles in the real world (Agashe & Chapman, 2019; Hendrickson, Biehler, & Mashayekh, 2014; McFarland, 2020). Simple modifications to the environment have also proved to be beneficial in agent learning (Randløv, 2000) and goal recognition (Keren, Gal, & Karpas, 2019). These examples show that environment shaping can improve an agent's performance with respect to its assigned task. We target settings in which environment shaping can reduce NSE, without significantly degrading the performance of the agent in completing its assigned task. Studies with human subjects have shown that users are generally willing to perform minor environment modifications to mitigate the impacts of NSEs (Saisubramanian, Roberts, & Zilberstein, 2021).

Our formulation consists of an acting agent and an environment designer. The acting agent computes a policy that optimizes the completion of its assigned task and has no knowledge about the NSE. The environment designer agent is a helper agent that shapes the environment through minor modifications so as to mitigate the NSE of acting agent's actions, without affecting the acting agent's ability to complete its assigned task. While this helper agent can be any autonomous agent, the formulation is inspired by having a human in the role. The environment is described using a configuration file, such as a map, which is shared between the actor and the designer. Given an environment configuration, the actor computes and shares sample trajectories of its optimal policy with the designer. The actor shares trajectories, instead of the entire policy, since compact representation of the entire policy is a practical challenge in large problems (Guestrin, Koller, & Parr, 2001). The designer estimates the actor's policy based on the trajectories, measures the associated NSE, and modifies the environment by updating the configuration file, if necessary. Environment shaping is more desirable than providing feedback or demonstrating a safe way of performing the task, when the agent's learning capabilities and its model fidelity are

Assumptions/ Requirements	Learning from Feedback	Environment Shaping
Agent state representation	sufficiently rich to learn about NSE	–
Model update requirements	augment with NSE penalty function	no model update
Type of human assistance	provide immediate feedback	reconfigure environment
Factors affecting performance	sampling biases in feedback collection	designer's ability to accurately predict actor policy from trajectories
Environment configuration	–	user-configurable & described by a file
Unavoidable NSE	cannot be eliminated	can be eliminated

Table 1: Assumptions and requirements of the two proposed solution approaches. "–" indicates that the approach is indifferent to the corresponding property.

unknown. Environment shaping can eliminate both avoidable and unavoidable NSE with respect to the *original*, unmodified environment. Besides the environment configuration file, the actor's objective and policy, no other information is shared between the actor and the designer. Mitigating NSE via environment shaping is more suitable for settings in which the human has more control over the environment, agent's learning capabilities are unknown, and the environment is configurable.

Summary of contributions We present two complementary solution approaches to mitigate agent's NSE. The agent and the human have different responsibilities in the two approaches. In learning from feedback, the agent is directly involved in mitigating the NSE as it learns about NSE and updates its model, while the human facilitates this by providing feedback. In environment shaping, the human has more control over the environment, by definition, and the agent has an indirect involvement in mitigating NSE since it only reacts to the reconfiguration by revising its policy. Table 1 summarizes the assumptions and requirements of the two approaches.

Our primary contributions in this paper are: (1) formally defining NSE of agent actions; (2) formalizing the problem of mitigating NSE as a multi-objective MDP with slack; (3) presenting a solution approach to update the agent's policy by learning about NSE as a secondary reward function and estimating the minimum slack required to avoid NSE; (4) studying various types of feedback mechanisms to learn the penalty associated with NSE; (5) evaluating the performance and analyzing the bias associated with each feedback mechanism; (6) presenting environment shaping as a method to mitigate NSE and an algorithm for shaping; and (7) empirical evaluation of environment shaping on two proof-of-concept domains.

2. Related Work

In this section, we review the emerging approaches to mitigate the impacts of negative side effects. Table 2 summarizes the characteristics of the approaches we discuss.

A natural approach to overcome NSE caused by reward misspecification is to update the agent's model by learning the intended reward function. By treating the specified proxy reward as a set of demonstrations and using approximate solutions for inference, the agent

Approach	Multi-objective formulation	Bounded-guarantees wrt. agent's task	Generalize learned NSE across states	Distinguish and handle different magnitudes of NSE
(Hadfield-Menell et al., 2017)	✗	✗	✓	✗
(Zhang et al., 2018)	✗	✗	✗	✗
(Shah et al., 2019)	✓	✗	✗	✗
(Krakovna et al., 2019)	✓	✗	✗	✓
(Zhang et al., 2020)	✗	✗	✗	✗
(Turner et al., 2020a)	✗	✗	✗	✗
(Turner et al., 2020b)	✗	✗	✗	✗
Our approach	✓	✓	✓	✓

Table 2: An overview of the characteristics of different approaches in mitigating NSE.

can learn the intended reward function (Hadfield-Menell et al., 2017). As acknowledged by the authors, this algorithm does not scale to large, complex settings. Further, updating the reward function of a deployed system requires extensive evaluation of the system to ensure that no new risks are introduced by the update. This approach is therefore better suited for handling safety-critical NSE that justifies the suspension of the deployed system—something that we try to avoid in our work.

Another form of NSE arises when an agent alters features in the environment that the user does not expect or desire to be changed. This can be addressed by constraining the features that can be altered by the agent during its operation. Zhang et al. (2018) consider a setting in which the uncertainty over the desirability of altering a feature is included in the agent's model. The agent first computes a policy assuming all the uncertain features are "locked" for alteration. If a policy exists, then the agent executes it. If no policy exists, the agent queries the human to determine which features can be altered and recomputes a policy. A regret minimization approach is used to select the top-k features for querying. Recently, the authors extended this approach to identify if NSEs are unavoidable by casting it as a set-cover problem (Zhang et al., 2020). If the side effects are unavoidable, the agent ceases operation. Therefore, these approaches are not suitable for settings where the agent is expected to alleviate (unavoidable) NSEs to the extent possible, while completing its assigned task.

Another class of solution methods defines a penalty function for the NSE as a measure of deviation from a baseline state, based on the features altered. Krakovna et al. (2019) present a multi-objective formulation with scalarization, with the deviation from baseline state measured using reachability-based metrics. The agent's sensitivity to NSE can be adjusted by tuning the scalarization parameters. This approach may penalize all side effects, including positive side effects, since it does not account for human preferences. To address this drawback, Shah et al. (2019) present an approach to infer human preferences from the initial state. They assume that an environment is typically optimized for human preferences and the agent can mitigate NSEs by inferring human preferences before it starts acting. Attainable utility (Turner et al., 2020a, 2020b) measures the impact of side effects as the shift in the agent's ability to optimize auxiliary objectives, generalizing the rela-

tive reachability measure presented earlier (Krakovna et al., 2019). All these approaches employ impact regularizers to mitigate NSEs. Designing the right impact regularizer is challenging since the agent's behavior is sensitive to the choice of baseline state, the metric used to calculate the deviation, and requires knowledge about the dynamics of the environment (Lindner, Matoba, & Meulemans, 2021). Certain NSEs may also impact the agent's ability to complete tasks in the future. Krakovna et al. (2020) present an approach that provides the agent with an auxiliary reward for preserving its ability to perform future tasks in the environment. We target settings where the agent has no prior knowledge about the NSE of its actions and the side effects are experienced immediately after action execution.

There have also been numerous recent studies focused on the broad challenge of building safe and reliable AI systems (Amodei et al., 2016; Russell, Dewey, & Tegmark, 2015; Saria & Subbaswamy, 2019; Thomas, da Silva, Barto, Giguere, Brun, & Brunskill, 2019), on the safe deployment of systems trained in simulation or a structured environment (Ramakrishnan, Kamar, Dey, Horvitz, & Shah, 2020; Ramakrishnan, Kamar, Nushi, Dey, Shah, & Horvitz, 2019; Lakkaraju, Kamar, Caruana, & Horvitz, 2017), and on the problem of configuring environments to enable efficient and interpretable agent operation (Keren, Pineda, Gal, Karpas, & Zilberstein, 2017; Kulkarni, Sreedharan, Keren, Chakraborti, Smith, & Kambhampati, 2020). In this work, we focus specifically on negative side effects that are discovered when the system is deployed.

3. Problem Definition

Consider an environment E in which an agent reasons using its acquired model—a Markov decision process (MDP) $\tilde{M} = \langle \tilde{S}, \tilde{A}, \tilde{T}, \tilde{R} \rangle$. The agent's state space is denoted by $\tilde{S}$, its action space by $\tilde{A}$, transition function by $\tilde{T}$, and its reward function by $\tilde{R}$. The agent's model $\tilde{M}$ includes a single objective, which is the primary task of the agent, and all the components necessary to complete the primary task. A factored state representation is considered. A *primary policy* is an optimal policy for $\tilde{M}$, optimizing the agent's primary objective defined by $\tilde{R}$. Executing a primary policy may produce negative side effect in some states since it only optimizes the primary task. We assume there exists a function that determines the probability of NSE occurrence associated with an agent trajectory—a sequence of state-action pairs representing a possible trace of the agent in the environment—but this function may be unknown to the agent.

Definition 1. *Let $\zeta = \{(s_1, a_1), \ldots, (s_n, a_n)\}$, $n \geq 1$, be a finite trajectory of an agent operating in an environment $E \in \mathcal{E}$, with $\mathcal{E}$ denoting a finite set of environments. Let Z denote the set of agent trajectories. A **negative side effect** is an event that occurs when the agent executes a certain trajectory in an environment, with probability determined by a function $N : Z \times \mathcal{E} \to [0, 1]$ where $N(\zeta, E)$ denotes the probability of negative side effect occurring when the agent follows a trajectory $\zeta \in Z$ in environment $E \in \mathcal{E}$.*

Multiple types of NSEs may occur during the system operation and each type of side effect i may be associated with a function N_i, which determines the probability of a specific type of negative side effect occurring when executing a trajectory.

Definition 2. *An **Markovian** negative side effect is one that is observed immediately after an action execution in a state. When $\zeta = \{(s_1, a_1)\}$, $N(\zeta, E)$ denotes the probability of a Markovian negative side effect.*

States in which an agent's actions have Markovian NSE are called *susceptible states*. The severity of a side effect may range from tolerable to severe impact, depending on the agent's primary task and the environment in which it is situated. While it is challenging to determine N during system design, in general, the user can estimate N by observing the agent behavior and its consequences in the deployed environment. Tolerance to NSE and the associated penalty are user-specific. Let $\theta \in \Theta$ denote user tolerance parameter for NSE in a given environment, where Θ denotes the discrete set of values θ can take.

Definition 3. *The **penalty** for a negative side effect is defined by $R_{N,\theta} : Z \times \mathcal{E} \rightarrow \mathbb{R}$, with θ denoting user tolerance parameter and $R_{N,\theta}(\zeta, E)$ denoting the penalty associated with trajectory $\zeta \in Z$ in environment $E \in \mathcal{E}$.*

Since NSE occurrence depends on the environment and the penalty depends on user tolerance, it is challenging to accurately specify details about NSE during system design. Further, the designer may inadvertently omit details that are unrelated to the primary task. As a result, the agent often does not have prior knowledge about the side effects of its actions and therefore does not have the ability to minimize NSE. The side effects may be avoidable or unavoidable by the agent, when it is performing its assigned task.

Definition 4. *A negative side effect is **avoidable** in an environment E if there exists a policy $\tilde{\pi}$ to complete the agent's assigned task such that $N(\zeta, E) = 0, \forall \zeta \sim \tilde{\pi}$, and unavoidable otherwise.*

In this work, we target Markovian negative side effects that are (1) undesirable, but not safety-critical; (2) deterministic—always occur when the agent executes certain actions in some states; and (3) not preventing the agent from completing its assigned task. The learning from feedback approach focuses on the Markovian NSE. The challenges in extending it to NSE associated with agent trajectories (general case defined in Definition 1) are discussed in Section 5. The environment shaping approach can handle both Markovian NSE and those associated with agent trajectories. The solution approaches we present naturally extend to settings in which a state in $\tilde{S}$ maps to multiple states in the real world with different NSE severities, by considering the expected penalty that accounts for the likelihood of occurrence.

4. Mitigating NSE using Feedback

An agent that is unaware of the negative side effects of its actions can gather this information through feedback signals from an oracle or through autonomous exploration. We formulate the problem of mitigating NSE as a multi-objective planning problem with lexicographic ordering over the reward functions. The agent learns a secondary reward function corresponding to NSE penalty. There are two main benefits of a multi-objective formulation for this setting. First, using multi-objective models makes it easier for users to understand and control the trade-off between the agent's ability to mitigate the NSE and optimize its

assigned task. Second, by separating the reward functions of the agent's primary objective and NSE, it is ensured that the model aspects corresponding to agent's assigned task are not corrupted during the update.

4.1 Background on Lexicographic MDP

We provide a brief background on lexicographic Markov decision process (LMDP) (Wray et al., 2015), which is a multi-objective MDP with lexicographic preferences over reward functions. LMDPs are particularly useful and convenient to model multi-objective MDPs with competing objectives and with an inherent lexicographic ordering over them.

An LMDP is defined by the tuple $M = \langle S, A, T, \mathbf{R}, \mathbf{\Delta}, \mathcal{S}, o \rangle$ (Wray et al., 2015) with S denoting the finite set of states; A denoting the finite set of actions; $T : S \times A \times S \rightarrow [0, 1]$ is the transition function indicating the probability of reaching state $s' \in S$ when executing action $a \in A$ in state $s \in S$; $\mathbf{R} = [R_1, ..., R_k]^T$ is a vector of reward functions, with $R_i : S \times A \rightarrow \mathbb{R}$ denoting the reward associated with executing action $a \in A$ in state $s \in S$ corresponding to i^{th} objective; $\mathbf{\Delta} = [\delta_1, ..., \delta_{k-1}]^T$ is a vector of *slack* variables with $\delta_i \geq 0$; $\mathcal{S} = \{S_1, \ldots, S_l\}$ is an l-partition over the state space; and $o = \langle o_1, \ldots, o_l \rangle$ denotes the strict preference ordering with $L = \{1, \ldots, l\}$, $\forall j \in L, o_j$ is a k-tuple ordering elements of k.

We consider $l = 1$, denoting one preference ordering over the objectives. That is, $o = \langle o_1 \rangle$ and $o_1 = \langle 1, \ldots, k \rangle$ denoting the strict preference ordering over the objectives. In the rest of the paper, we denote the first objective as o_1, second objective in the ordering as o_2, and so on, in the interest of clarity and to improve the readability. The strict preference ordering over them is denoted as $o_1 \succ o_2 \succ \ldots o_{k-1} \succ o_k$, where the lexicographic preference operator "$\succ$" denotes that the left term has a higher preference ordering and is always optimized prior to the right term.

The slack δ_i is an additive value denoting the acceptable deviation from the optimal expected reward for objective o_i so as to improve the lower priority objectives. The set of value functions is denoted by $\boldsymbol{V} = [V_1, ..., V_k]^T$, with V_i denoting the value function corresponding to o_i, and calculated as

$$\boldsymbol{V}^{\pi}(s) = \mathbf{R}(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \boldsymbol{V}^{\pi}(s'), \forall s \in S.$$

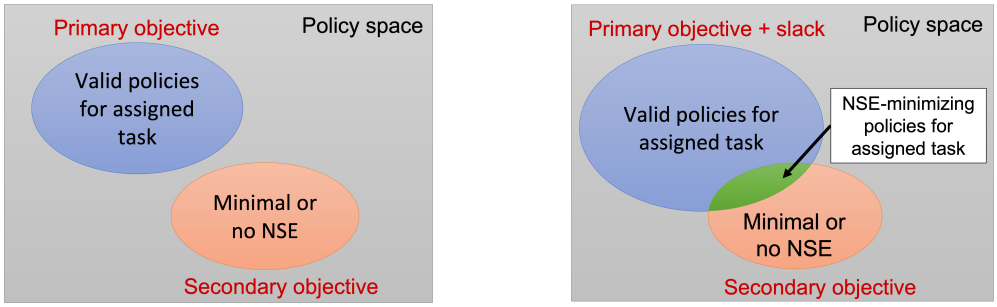
LMDP sequentially processes each objective in the lexicographic order and therefore the policy of the current objective and the slack determine the actions available for optimizing the next objective. The set of restricted actions for o_{i+1} is:

$$A_{i+1}(s) = \{a \in A | \max_{a' \in A_i} Q_i(s, a') - Q_i(s, a) \leq \eta_i\}$$

where $\eta_i = (1 - \gamma)\delta_i$, with a discount factor $\gamma \in [0, 1)$. We refer to (Wray et al., 2015) for a detailed background on LMDP.

4.2 Problem Formulation

In this section, we formulate the problem of mitigating NSE using the LMDP framework. We consider Markovian NSE that is observed immediately after action execution in a state.



(a) Disjoint policy sets

(b) Slack-induced availability of NSE-minimizing policies for assigned task

Figure 3: An illustration of the benefits of slack.

The agent's model is augmented with a secondary reward function that represents the penalty for NSE. We investigate the efficiency of different forms of feedback to learn about NSE, including oracle feedback and agent exploration. The agent may not be able to observe the NSE except for the penalty, which is proportional to the severity of the NSE, provided by the feedback mechanism. Given $\tilde{M}$ and feedback data regarding NSE, the agent is expected to compute a policy that optimizes its assigned task, while avoiding NSE as much as possible, subject to a slack.

Definition 5. *The **augmented MDP** of a given model $\tilde{M}$ is a lexicographic MDP, denoted by $M{=}\langle S, A, T, \boldsymbol{R}, o, \delta\rangle$ such that:*

- $S = \tilde{S}$ *denotes the state space;*
- $A = \tilde{A}$ *denotes the set of actions;*
- $T = \tilde{T}$ *denotes the transition function;*
- $\boldsymbol{R}{=}[\mathrm{R}_1, \mathrm{R}_2]$ *with* $\mathrm{R}_1 = \tilde{R}$ *denotes the reward associated with primary objective of the agent and* $\mathrm{R}_2 = R_N$ *denotes the NSE penalty;*
- $o{=}[o_1, o_2]$ *denotes the objectives where* o_1 *is the primary objective denoting the agent's assigned task and* o_2 *is minimizing NSE, with* $o_1 \succ o_2$*; and*
- $\delta \geq 0$ *is the slack denoting the maximum deviation from optimal expected reward for* o_1 *in order to minimize NSE.*

There may be limited opportunity for avoiding NSE when optimizing the agent's assigned task, denoted by o_1. In many problems, the set of policies that are optimal (or valid) for o_1 and the set of policies that avoid NSE may be distinct (Figure 3(a)). Allowing for a slack on the primary objective indicates willingness to tolerate a certain degree of sub-optimal behavior, with respect to o_1. This expands the set of policies that are "valid" for its primary objective, thereby enabling the agent to compute a NSE-minimizing policy for the assigned task (Figure 3(b)). Hence, we consider a slack value δ on o_1, which denotes the maximum allowed deviation of a policy from the optimal expected reward for o_1 in order to minimize NSE.

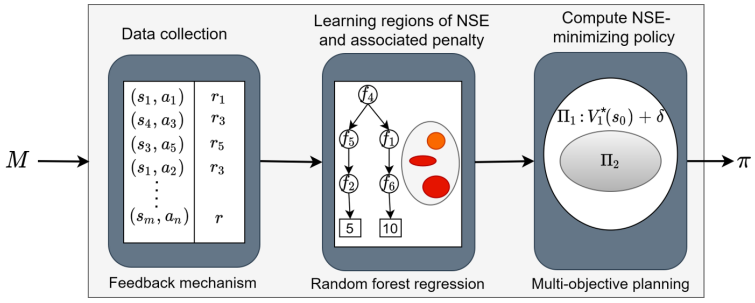


Figure 4: Overview of our framework for mitigating NSE using feedback.

Since the agent cannot predict the NSE a priori, it must learn R_N using feedback mechanisms (discussed in Section 4.4). Our overall framework for minimizing the NSE involves the following three steps (Figure 4):

1. The agent collects data about NSE through various types of oracle feedback or by exploring the environment;
2. A predictive model of NSE is trained using the gathered data to generalize the agent's observations to unseen situations, represented as a reward function R_N;
3. The agent computes a policy π by solving the augmented MDP optimally with the given δ and learned R_N.

4.3 Slack Estimation

Slack denotes the maximum allowed loss in the expected reward of the agent's primary objective in order to minimize the NSE. A smaller slack value limits the scope for minimizing NSE. A very high slack can allow the agent to not fulfill o_1 in an attempt to minimize the NSE. Therefore, the slack determines the overall performance of the agent with respect to both its objectives. Typically the slack value is based on user preferences and their general tolerance towards NSE. We present an approach to determine the minimum slack required to avoid NSE altogether, when feasible (once knowledge about the NSE is obtained).

Algorithm 1 determines the slack as the difference between the expected reward of the model before and after disabling all the actions that produce NSE. The expected reward for completing the task from the single start state s_0 and using $\tilde{M}$ is denoted by $\tilde{V}_1^*(s_0)$. The expected reward after disabling all the actions with NSE, denoted by $\hat{V}_1^*(s_0)$, is the maximum reward that can be achieved without causing any NSE, when possible. Thus the difference in values indicates the minimum slack required to avoid NSE, when feasible, and ensures that the slack is in the same unit as the primary objective. We calculate the absolute difference to allow for negative rewards. A solution may no longer exist after the actions are disabled (Lines 3-4), in which case $\delta = \infty$ is returned, indicating that it is impossible to completely avoid NSE and still accomplish the task.

Proposition 1. *Given the function N that determines the probability of NSE occurrence, Algorithm 1 calculates the minimum slack required to avoid NSE, while still accomplishing the agent's primary objective, when the NSE is avoidable.*

Algorithm 1 `Slack Estimation` $(\tilde{M}, N, E)$

1: $\delta \leftarrow \infty$
2: $\tilde{V}_1^*(s_o) \leftarrow$ Solve $\tilde{M}$ optimally with respect to o_1
3: Compute NSE-free transition ($\hat{T}$) by disabling all actions with NSE, $\forall(\tilde{s}, \tilde{a}, \tilde{s}')$:
4: $\hat{T}(\tilde{s}, \tilde{a}, \tilde{s}') \leftarrow \begin{cases} \tilde{T}(\tilde{s}, \tilde{a}, \tilde{s}'), & \text{if } N((\tilde{s}, \tilde{a}), E) = 0 \\ 0, & \text{otherwise} \end{cases}$
5: **if** solution exists for $\langle \tilde{S}, \tilde{A}, \hat{T}, \tilde{R} \rangle$ with respect to o_1 **then**
6: $\quad \hat{V}_1^*(s_o) \leftarrow$ Solve $\langle \tilde{S}, \tilde{A}, \hat{T}, \tilde{R} \rangle$ optimally for o_1
7: $\quad \delta \leftarrow |\tilde{V}_1^*(s_o) - \hat{V}_1^*(s_o)|$
8: **return** δ

Proof. We prove this by contradiction. Let δ denote the slack returned by Algorithm 1 and δ^* denote the minimum slack required to avoid NSE in a setting with avoidable NSE. Assume $\delta^* > \delta$. This indicates that Algorithm 1 produced a lower value of slack than required, resulting in NSE during policy execution. This is possible when the model is not solved optimally or if all the actions that lead to NSE have not been disabled. By Lines 2-6 in Algorithm 1, all unacceptable actions are disabled based on N and the model is solved optimally. Therefore, $\delta^* \leq \delta$. Let $\delta^* < \delta$. By Lines 2-6 in Algorithm 1, all actions with NSE are disabled and the model is solved optimally. Hence, $\delta^* \not< \delta$. Therefore, $\delta^* = \delta$. □

Algorithm 1 breaks ties in favor of the primary objective. That is, if there are multiple policies, within the slack, that result in the same expected penalty for NSE, the algorithm will output a policy that optimizes the primary objective. Note that Algorithm 1 will fully utilize the slack value to minimize NSE, even if the improvement in NSE is minimal. For example, consider an agent operating with $\delta = 0.5\tilde{V}_1^*(s_0)$ to avoid NSE. However, if the agent operates with $\delta = 0.25\tilde{V}_1^*(s_0)$, the expected penalty for NSE is three. For a 25% increase in the slack value, the resulting improvement in NSE is three units. In many problems, such a trade-off may be undesirable. Algorithm 1 calculates the minimum slack required to avoid the NSE, and does not optimize this trade-off in the expected rewards.

The slack is specified by the user when NSE are unavoidable or when δ estimated using Algorithm 1 is beyond the user tolerance. If the models are solved approximately, without solution guarantees, Algorithm 1 is not guaranteed to return the minimum slack but our approach still produces a policy that minimizes NSE, given the slack.

4.4 Learning from Feedback

To learn about *Markovian NSE*, we consider two forms of feedback that correlate with features in $\tilde{S}$: feedback acquired from an oracle and feedback the agent acquires by exploring the environment. The oracle feedback typically represents human feedback and provides signals about undesirable actions, with respect to NSE. Alternatively, the agent may explore the environment to gather reward signals with respect to NSE.

4.4.1 Learning from Human Feedback

We discuss four different forms of human feedback, each providing information about NSE directly or indirectly, along with their limitations. In this section, "budget" refers to the maximum number of queries by the agent or the number of feedback signals received from the human, and "without replacement" means that the agent queries only unique state-action pairs.

Random Queries This feedback type represents an ideal setting in which the agent randomly selects an (s, a) pair for querying an oracle, given a querying budget and without replacement, and receives the exact, corresponding penalty for the NSE.

Despite the benefits offered by this approach, it is often unrealistic to expect exact penalty specification for randomly selected state-action pairs. Hence we also consider other feedback mechanisms where the human *approves* the agent's actions, *corrects* the agent's policy, or *demonstrates* an acceptable trajectory. Since such feedback types do not provide the exact penalty, feedback indicating acceptable actions are mapped to a zero penalty and others are mapped to a fixed, non-zero penalty denoted by k, which in turn contributes to R_N. In our experiments, k is set to the maximum penalty incurred for NSE in the problem.

Approval (HA) The agent randomly selects (s, a) pairs, without replacement, to query the human, who in turn either approves or disapproves the action in that state. The agent learns by mapping the approved actions to zero penalty, $R_N(s, a) = 0$, and the disapproved actions are mapped to a non-zero penalty $R_N(s, a) = k$. We consider two types of human approval: *strict* (HA-S) and *lenient* (HA-L). Strict feedback disapproves all actions that result in NSE. Lenient approval only disapproves actions with severe NSE. The severity threshold for HA-L is a tunable parameter that is problem-specific. Thus with HA-L, the agent will not learn about NSE with low severity and with HA-S, the agent cannot distinguish between actions with different severities of NSE. We consider two types of approval for simplicity but the framework supports more complex forms of human approval that enable the agent to learn NSEs with different levels of severity with a piecewise R_N. Despite the simplicity of this approach, it may be difficult for the human to approve or disapprove actions in isolation, since they do not convey the agent's overall behavior.

Corrections (C) In this form of feedback, the agent performs a trajectory of its primary policy, with the human monitoring. If the human observes an unacceptable action at any state, they stop the agent and specify an acceptable action to execute in that state. If all actions in a state lead to NSE, then the user specifies an action with the least NSE. The agent proceeds until the goal is reached or until interrupted again. When interrupted, the agent assumes that all actions, except the correction, are unacceptable in that state. If not interrupted, the action is considered to be acceptable. Acceptable actions are mapped to zero penalty, $R_N(s, a) = 0$, and unacceptable actions are mapped to a non-zero penalty, $R_N(s, a) = k$. This approach requires constant human oversight, which may be practically infeasible in many settings.

Demo-action mismatch (D-AM) In demo-action mismatch, the human provides limited number of demonstrations. Each demonstration is a complete trajectory of performing the task, from start to the goal. The agent collects these trajectories and compares them

with its primary policy. For all states in which there is an action mismatch, the agent assumes its policy leads to NSE and assigns $R_N(s,a)=k$, otherwise assigns $R_N(s,a)=0$. The agent does not mimic human behavior. Instead, it only uses the demonstrations to gather knowledge about NSE. A limitation of this approach is that the agent does not receive any information about NSE in states that are not included in the demonstrations.

4.4.2 Learning from Exploration

When human feedback is expensive to collect or unavailable, it may be easier to allow the agent to identify susceptible states through exploration. Learning from exploration uses ϵ-greedy action selection—the agent exploits the action prescribed by its primary policy or explores a random action to learn about NSE. The agent executes an action and receives the corresponding NSE penalty, $R_N(s,a)$. Note that the agent explores only to learn R_N and the final policy is computed by solving the augmented model. "Budget" for exploration strategies refers to the number of episodes.

We consider three exploration strategies: *conservative* – where the agent explores an action with probability 0.1 or follows its primary policy, *moderate* – where the agent either explores an action with probability 0.5 or follows its primary policy, and *radical* – where the agent predominantly explores with probability 0.9, allowing the agent to possibly gather more information about NSE than the other exploration strategies. Learning by exploring the real-world may be risky in some contexts, such as autonomous vehicles. In such settings, exploration may be performed in a simulator that is designed to train the agent to avoid the negative side effects, or the agent can learn from human feedback.

4.4.3 Model Learning

The agent's observations are generalized to unseen situations by training a random forest regression (RF) model to predict the penalty R_N. We use the RF model to handle the continuous nature of the penalty and any regression technique may be used in practice. The hyperparameters for the training are determined by a randomized search in the space of RF parameters. For each hyperparameter setting, a three-fold cross validation is performed and the mean squared error is calculated. Parameters with the least mean squared error are selected for training, which is then used to predict the penalty R_N. The augmented MDP is then updated with this learned R_N and the agent computes a NSE-minimizing policy for execution by solving the augmented MDP.

4.5 Experimental Setup

We perform extensive evaluation of the different feedback mechanisms for mitigating avoidable and unavoidable NSE.

Baselines The performance of our approach is compared with three baselines. First is the *Oracle* agent that avoids the NSE while satisfying the primary objective. The *Oracle* has a perfect model of NSE and its policy is computed by solving the model after avoiding all the actions with NSE. In problems with unavoidable NSE, it selects the action with the least NSE since that is the best possible performance that can be achieved while satisfying the primary objective. The performance of the *Oracle* provides a lower bound on the

penalty for NSE incurred by the agent. The second baseline is the *No queries* case in which the agent does not query or learn about NSE. Instead, it naively executes its primary policy. This provides an upper bound on the penalty for NSE incurred by the agent. Third is the scalarization approach (Krakovna et al., 2019) (RR) in which the agent optimizes $r(s_t, a_t) - \beta d(s_t, b_t)$, where $r(s_t, a_t)$ is the reward corresponding to o_1 and $d(s_t, b_t)$ is the measure of deviation from the baseline state b_t, denoting the NSE. A direct comparison with this approach is not feasible since it is based on assumptions that do not hold in our setting. Therefore, we modified the RR approach to make it applicable in our setting—by calculating the deviation based on a model of NSE learned with Random Query approach, as it does not introduce any bias. We compute the deviation from inaction baseline, which measures the NSE of the agent's action with respect to no action execution in that state (Krakovna et al., 2019). The side effects we consider are irreversible by an agent, once occurred, making the baseline state unreachable. We tested with $\beta \in [0.1, 0.9]$ since o_1 is prioritized in our formulation and report results with $\beta = 0.8$ as it achieved the best trade-off in training.

Side Effects In the interest of clarity, we consider two types of NSE severity: mild and severe. Each action can either result in a mild negative side effect, severe negative side effect, or no negative side effect. The strict human approval (HA-S) feedback disapproves all actions that result in NSE. The lenient human approval (HA-L) only disapproves actions with severe NSE. For learning NSE by exploration, we consider agent exploration in a simulator where the reward signals indicate NSE penalty.

In our experiments, we optimize costs, which are negations of the rewards. Random forest regression from `sklearn` Python package is used for model learning. The augmented MDP is solved using a lexicographic variant of LAO* (Hansen & Zilberstein, 2001). The slack is computed using Algorithm 1 and $\gamma = 0.95$. Values averaged over 100 trials of planning and execution, along with their standard errors, are reported for the following domains.

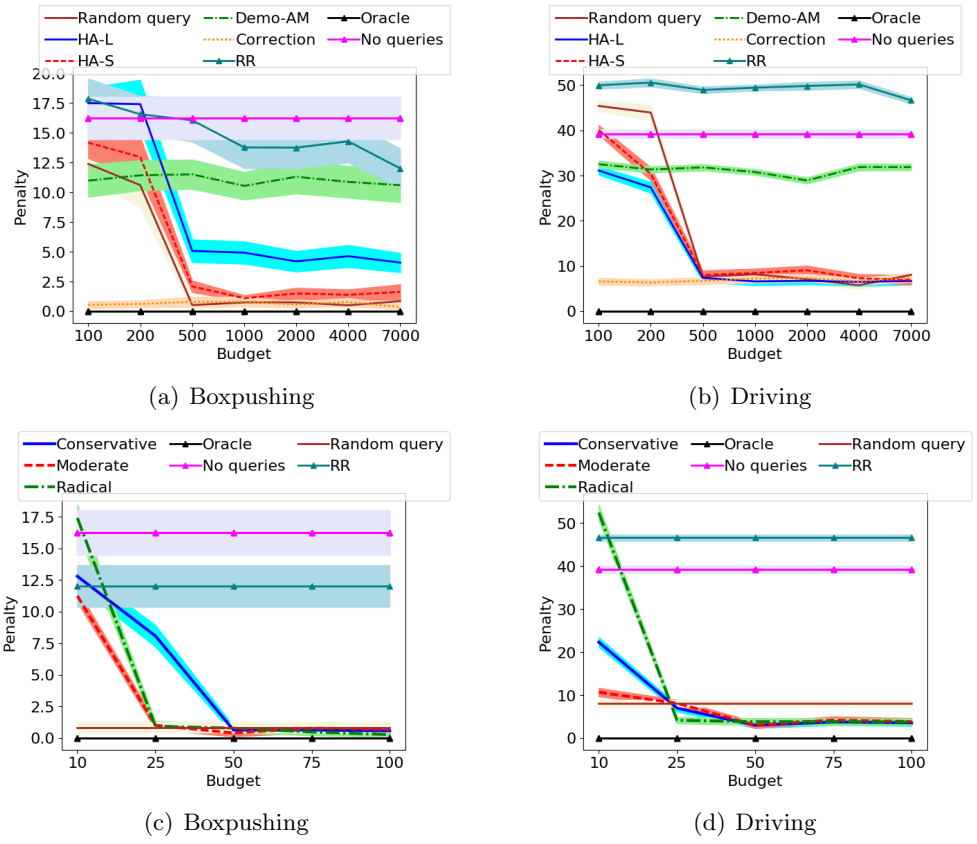
Boxpushing We consider a modified boxpushing domain (Seuken & Zilberstein, 2007) in which the agent is expected to minimize the expected time taken to push a box to the goal (Figure 1). Each action takes +1 unit of time. Each state is represented as $\langle x, y, b, c \rangle$ where x, y denote the agent's location, b indicates if the agent is pushing the box, c indicates the current cell's surface type. Pushing the box on a surface type $c = 1$ results in severe NSE with a penalty of 10, pushing the box on a surface $c = 2$ results in mild NSE and a penalty of 5, and no NSE otherwise. The state features used for training are $\langle b, c \rangle$. We test on five instances with grid size 15×15 and with varying initial location of the box and NSE regions.

Driving Our second domain is based on simulated autonomous driving (Saisubramanian, Kamar, & Zilberstein, 2020a; Wray et al., 2015) in which the agent's primary objective is to minimize the expected cost of navigation from start to a goal, during which it may encounter some puddles. The agent can move in all four directions at low and high speeds. The cost of navigating at a low speed is two and that of high speed is one. The agent spatters water when navigating over a puddle at high speed, which is undesirable. Some puddles may have pedestrians in the vicinity and splashing water on them results in severe NSE with a penalty of 10 and a mild NSE otherwise with a penalty of 5. Each state is represented by $\langle x, y, l, p, h \rangle$ where x, y denote the agent's location, l is the speed, p, h indicate the presence of a puddle and a pedestrian in the vicinity. Features used for training are $\langle l, p, h \rangle$. Five

test instances are generated with grid size 15×15 and by varying the start, goal, puddle, and pedestrian locations.

4.6 Results and Discussion

Effectiveness of learning from feedback We first discuss the effectiveness of various feedback approaches in learning about NSE in settings with avoidable NSE and then discuss results with unavoidable NSE.

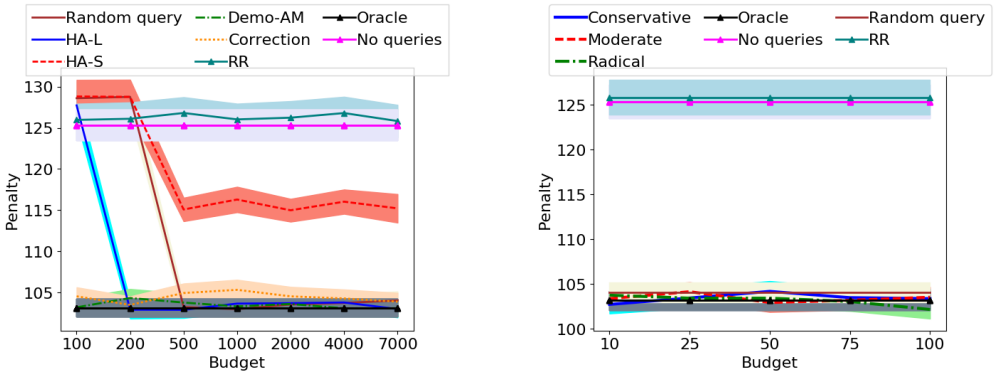


(a) Boxpushing (b) Driving

(c) Boxpushing (d) Driving

Figure 5: Effect of learning from human feedback methods (a-b) and with exploration strategies (c-d) on problems with avoidable NSE.

Figure 5 shows the average penalty incurred for NSE in settings with avoidable NSE, as the feedback budget is increased. The Corrections feedback is efficient in terms of samples required since the human corrects the agent by prescribing an acceptable action in each state. Random querying and HA-S, which rely on random samples of states, achieve significant reduction in NSE with 500 samples. Although HA-L also relies on random sampling of states, it does not provide information about mild NSE, which affects its performance. Training with Demo-AM feedback does not always minimize NSE even as the budget is increased, since the agent does not receive enough negative samples and has no information about the NSE in states unvisited in the demonstrations. Additionally, it is unable to



(a) Human feedback

(b) Exploration

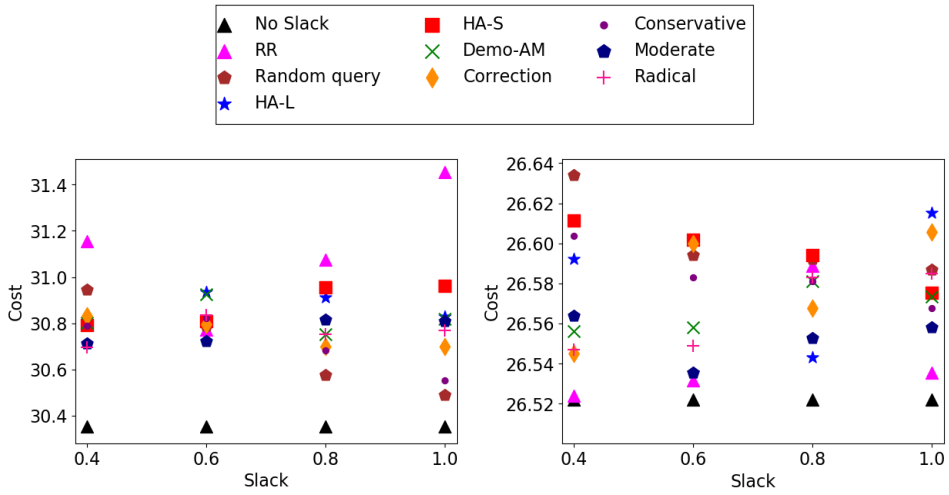
Figure 6: Performance on Boxpushing problems with unavoidable NSE.

distinguish between the different levels of severity of the NSE of its actions. The performance with Demo-AM is also affected when the acceptable actions demonstrated by the oracle violate the slack constraints for the agent and therefore, the agent may not be able to minimize NSE. However, Demo-AM is still better than *No queries*. *RR* with the model of NSE learned using Random Query and with the given budget, performs poorly irrespective of the budget. Apart from the reported results, we also tested *RR* with a perfect model of NSE and its performance was significantly better and sometimes comparable to the *Oracle*'s performance. However, obtaining a perfect model of NSE is non-trivial in practice. In Figure 5 (c-d), Random querying with the maximum budget (7000) and *RR* with this learned model are plotted in to compare the performance of exploration strategies and to understand how the correlated samples affect the performance. Irrespective of the exploration type, the agent gathers knowledge about NSE with a reasonable number of episodes.

When NSE are unavoidable, Algorithm 1 does not produce a finite slack. Therefore, we experiment with a slack value that is 15% of the expected cost of o_1, based on the observation that slack returned by the algorithm for problems with avoidable NSE are typically within 15% of the expected cost for o_1. Figure 6 plots the results for boxpushing problems with unavoidable NSE. Problem instances with unavoidable NSE case were generated by making sure there is no path to the goal over surface $c = 3$. We do not report results on the driving domain because there are no unavoidable NSE in the problem setting we consider—the agent can avoid NSE by navigating at a low speed.

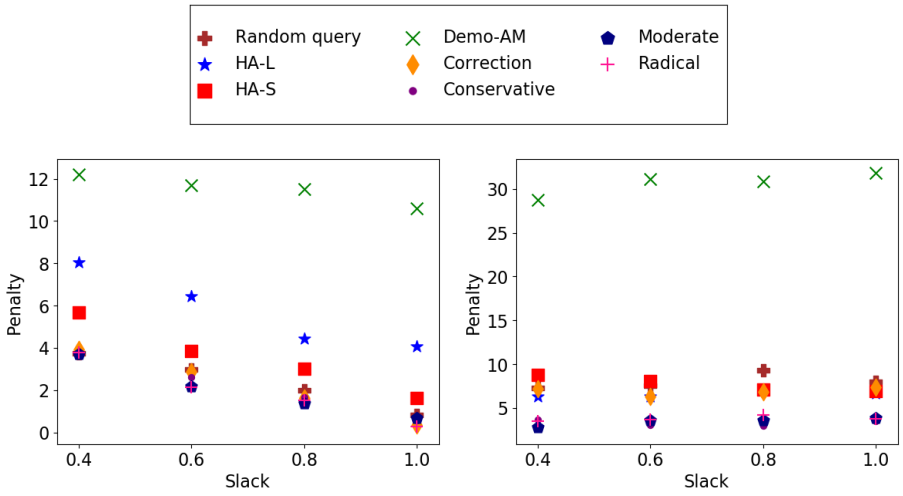
The performance of human feedback techniques are similar to that of avoidable NSE setting, except for HA-S and Demo-AM. Since HA-S cannot distinguish between different severities, its performance is affected when NSE are unavoidable. Demo-AM matches the performance of other techniques when NSE is unavoidable. Similar to Figure 5(c-d), learning by exploration in settings with unavoidable NSE matches the performance of Random query. Overall, the results indicate that our framework can effectively learn and mitigate the impacts of both avoidable and unavoidable NSE.

Effect of slack We study the effect of slack on the expected cost of o_1 and on NSE (o_2), by varying the slack from 40% to 100% of the value returned by Algorithm 1 on problems



(a) Boxpushing (b) Driving

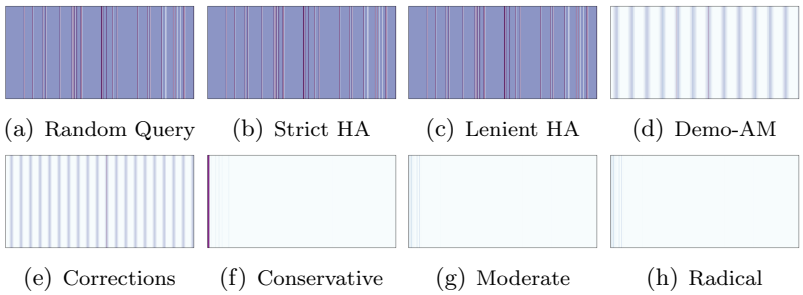
Figure 7: Effect of slack on o_1.



(a) Boxpushing (b) Driving

Figure 8: Effect of slack on NSE.

with avoidable NSE. Figures 7 and 8 shows the results with 7000 queries for human feedback and 100 exploration episodes. The values show the average cost in 100 episodes of executing the updated policy with each slack value. We do not compare with the baselines, which are unaffected by the variation in slack. Results with No Slack bound the performance of other techniques. As the slack is increased, the average penalty incurred for NSE tends to decrease. The minimal differences in the average costs for o_1 shows that the slack values returned by Algorithm 1 are reasonable and affect o_1 only by a small margin.



(a) Random Query (b) Strict HA (c) Lenient HA (d) Demo-AM

(e) Corrections (f) Conservative (g) Moderate (h) Radical

Figure 9: Frequency of querying across the state space with different feedback approaches.

5. Limitations of Learning from Feedback

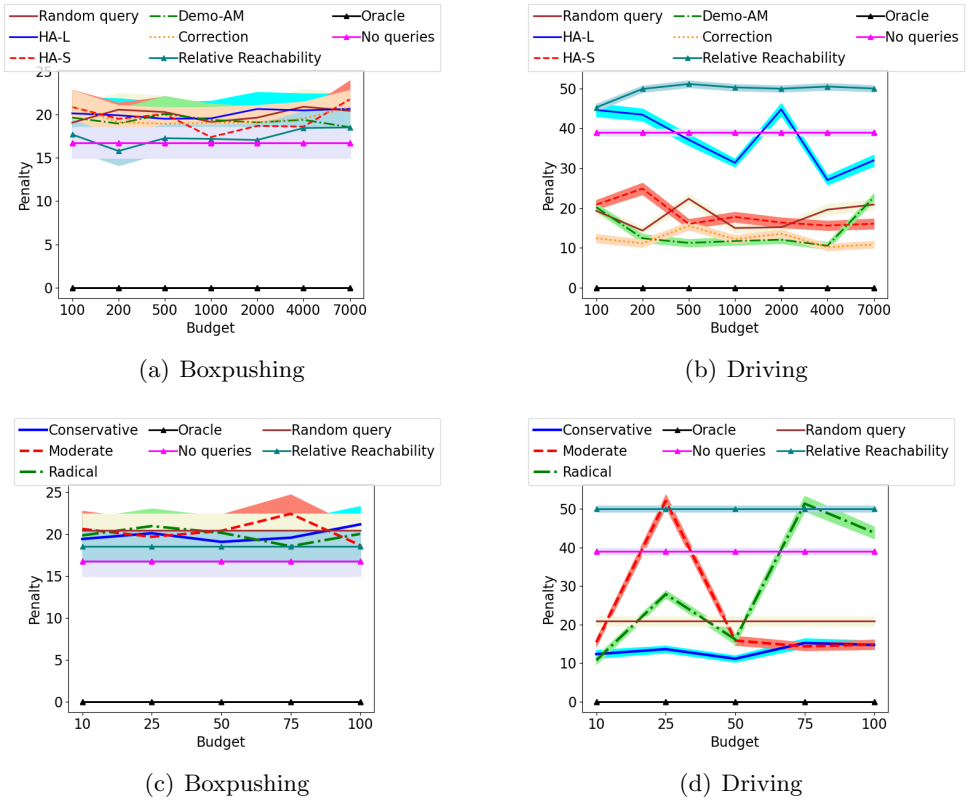
While our results show that learning from feedback successfully mitigates NSE, four key factors affect agent learning and its ability to mitigate NSE: (1) challenges in collecting accurate feedback; (2) sampling biases in feedback; (3) limited fidelity of agent state representation; and (4) unavoidability of certain NSEs.

Challenges in accurate feedback collection Our empirical results show that learning from feedback can successfully mitigate NSE. We now discuss the challenges and drawbacks of collecting feedback using the approaches discussed in Section 4.4.1.

In random query feedback, the human is expected to provide exact penalty for NSE, which is non-trivial and can result in misspecification in practice. While the approval approach overcomes this drawback, the agent cannot differentiate between NSE with different severities and it is often infeasible for the human to provide feedback on randomly selected state-action pairs. The corrections approach overcomes these limitations, but it requires constant human oversight, similar to action approval or random query approach. The Demo-AM approach alleviates the need for constant human supervision and it is relatively easier for the human to provide demonstrations than correcting or supervising the agent. However, the agent does not receive information about NSE in states that are not included in the demonstrations, and cannot differentiate between NSE with different severities. The autonomous exploration approach avoids the need for human supervision altogether, but it still requires the designer to specify the penalty for agent training in the simulator and may lead to misspecification. Further, learning by exploration may be unsafe in certain settings.

These approaches, except random query and approval, also introduce sampling bias since the samples are not *i.i.d.*, as the visited states are correlated. Furthermore, all actions other than the corrections or those demonstrated are assumed to be unacceptable. Since there may be multiple acceptable actions in each state, this introduces additional bias. We now discuss how the biases affect agent learning.

Bias in various feedback mechanisms The sampling biases in a feedback approach affects agent performance. Different agent behaviors emerge when learning from different forms of feedback, as shown in Figure 5. Figure 9 plots the frequency of querying in different regions of the state space with different feedback mechanisms. The x-axis denotes the state space and darker shades indicate regions that were frequently queried. The results are



(a) Boxpushing (b) Driving

(c) Boxpushing (d) Driving

Figure 10: Effect of learning from human feedback (a-b) and with autonomous exploration (c-d), when the agent state representation has low fidelity.

plotted for driving domain with avoidable NSE and using 7000 queries for human feedback and 100 trials for exploration. Feedback techniques based on random sampling of states have a higher coverage of the state space, contributing to a better performance. The exploration techniques are the most restricted, due to which their performances are largely similar. Since an ϵ-greedy approach is followed for exploration, these techniques likely cover only the region surrounding that of the primary policy. As states in this region are often critical for satisfying the agent's primary objective, learning about NSE in this restricted region is often sufficient to effectively mitigate the penalty for NSE. These results show that different feedback types focus on different regions of the state space, which in turn affects the NSE model learning.

Fidelity of state representation Learning from feedback assumes that the agent state representation has all the necessary features to learn about NSEs. However, in many scenarios, the agent's state representation may include only the features related to the agent's task. Limited fidelity of the state representation may affect the agent's learning and adaptation, since the NSE could potentially be non-Markovian and the feedback signals received for a state-action pair may be conflicting. Figure 10 plots the effect of learning from various feedback approaches when the agent's state representation has limited fidelity. For

the boxpushing domain, the agent's state representation does not include the surface type, since it is unrelated to its task. The state feature used for training in this case is $\langle b \rangle$, where b indicates if the agent is currently pushing the box. However, the occurrence of NSE correlates with surface type. Hence, the agent's learning and its ability to mitigate NSE are affected. Similarly for the driving domain, we do not include the presence of puddle and nearby pedestrians in the state representation, since they are not relevant to o_1. The state feature used for training in this case is $\langle l \rangle$, which denotes the current speed. Since the presence of puddles and nearby pedestrians are important features for learning a model of NSE, the agent's performance is affected when the state representation does not include these features.

Unavoidable NSE and non-Markovian NSE Updating the agent's policy may not considerably mitigate the impacts of NSE when they are unavoidable, with respect to the agent's task, as shown in Figure 6. For example, in the boxpushing domain, NSE are unavoidable if the entire floor is covered with a rug. Alternate approaches that do not rely on agent learning and adaptation are required to mitigate NSE in such settings.

Negative side effects are considered to be non-Markovian when they are associated with a trajectory (Definition 1). While any form of feedback can be used to learn about NSE penalties associated with a sequence of actions, the proposed learning from feedback framework requires the penalty to be associated with each each state-action pair (by definition of R_N). Hence, the proposed approach cannot handle scenarios in which the NSE penalty for a trajectory is not decomposable into additive penalties associated with state-action pairs.

In the following section, we present an approach that can avoid NSE in such settings, when the human can assist the system, beyond providing feedback.

6. Mitigating NSE via Environment Shaping

This section presents another approach for mitigating NSEs when agents operate in environments that are configurable. In many settings, the human operating the system has a broader scope of knowledge and more control over the environment in which the agent is situated. We leverage these insights to mitigate the impacts of NSE when the agent has no prior knowledge about the side effects of its actions. The problem is formulated as a human-agent team with decoupled objectives. The agent optimizes its assigned task, during which its actions may produce NSE. The human shapes the environment through minor reconfiguration actions so as to mitigate the impacts of agent's side effects, without affecting the agent's ability to complete its assigned task. This decoupled approach does not make any assumptions regarding the agent's model and its learning capabilities. Further, by controlling the space of available modifications, the user can prioritize which side effects should be avoided.

6.1 Problem Formulation

Our formulation consists of an actor agent and an environment designer agent. The actor agent operates based on an MDP $\tilde{M}$ in an environment that is configurable and described by a configuration file E, such as a map of the environment. Describing the environment using configuration files is a common practice in robotics. A factored state representation is

assumed. The actor optimizes the reward for its assigned task, which is its primary objective o_P, and its model includes the necessary details relevant to o_P. Executing the policy π computed using $\tilde{M}$ may produce NSE, unknown to the actor. The environment designer measures the impact of NSE associated with the actor's π and shapes the environment, if necessary. While the environment designer can be any autonomous agent, the formulation is inspired by having a human in the role. The actor and the environment designer share the configuration file of the environment, which is updated by the environment designer to reflect the modifications. Optimizing o_P is prioritized over avoiding NSE. Hence, shaping is performed *in response* to the actor's policy. In the rest of the paper, we refer to the environment designer simply as designer—not to be confused with the designer of the agent. We also use the terms 'actor' and 'agent' interchangeably to denote the actor agent.

Each modification is a sequence of design actions. An example of a modification is $\{move(table, l_1, l_2), remove(rug)\}$, which moves the table from location l_1 to l_2 and removes the rug in the current setting, resulting in a new environment configuration. We consider the set of modifications to be finite since the environment is generally optimized for the human and the agent's primary task (Shah et al., 2019), and the human may not be willing to drastically modify it. Additionally, the set of modifications for an environment is included in the problem specification since it is typically controlled by the user and rooted in the NSE they want to mitigate.

We make the following assumptions about the nature of modifications: (1) the start and goal conditions of the actor are fixed and cannot be altered, so that the modifications do not alter the agent's task or its capabilities; and (2) modifications are applied tentatively for evaluation purposes and the environment is reset if the reconfiguration affects the actor's ability to complete its task or the actor's policy in the modified setting does not minimize the side effects.

Definition 6. *An actor-designer framework to mitigate negative side effect (***AD-NSE***) is defined by* $\langle E_0, \mathcal{E}, \tilde{M}, M_d, \delta_A, \delta_D \rangle$ *with:*

- E_0 *denoting the initial environment configuration of the actor;*
- $\mathcal{E}$ *denoting a finite set of possible reconfigurations of* E_0*;*
- $\tilde{M}$ *is the actor's MDP;*
- $M_d = \langle \Omega, \Psi, C, R_N \rangle$ *is the model of the designer with*
  - Ω *denoting a finite set of valid modifications that are available for* E_0*, including* $\emptyset$ *to indicate that no changes are made;*
  - $\Psi : \mathcal{E} \times \Omega \to \mathcal{E}$ *determines the resulting environment configuration after applying a modification* $\omega \in \Omega$ *to the current configuration, and is denoted by* $\Psi(E, \omega)$*;*
  - $C : \mathcal{E} \times \Omega \to \mathbb{R}$ *is a cost function that specifies the cost of applying a modification to an environment, denoted by* $C(E, \omega)$*, with* $C(E, \emptyset) = 0, \forall E \in \mathcal{E}$*; and*
  - R_N *is a model specifying the penalty for negative side effect, with* $R_N(\pi, E)$ *denoting the penalty associated with the actor's policy* π *in environment* $E \in \mathcal{E}$*.*
- $\delta_A \geq 0$ *is the actor's slack, indicating the maximum allowed deviation from the initial optimal policy for* o_P *when recomputing its policy in a modified environment; and*

- $\delta_D \geq 0$ *indicates the designer's tolerance threshold for NSE.*

Decoupled Objectives The actor agent's objective is to compute a policy π that maximizes its expected reward from start state $\tilde{s}_0$ in the current environment configuration E, with respect to o_P:

$$\max_{\pi \in \Pi} V_P^\pi(\tilde{s}_0|E),$$
$$V_P^\pi(\tilde{s}) = R(\tilde{s}, \pi(\tilde{s})) + \sum_{\tilde{s}' \in \tilde{S}} T(\tilde{s}, \pi(\tilde{s}), \tilde{s}') V_P^\pi(\tilde{s}'), \forall \tilde{s} \in \tilde{S}.$$

When the environment is modified, the actor agent recomputes its policy and may end up with a longer path to its goal. The slack δ_A denotes the maximum allowed deviation from the optimal expected reward in E_0, $V_P^*(\tilde{s}_0|E_0)$, to facilitate minimizing NSE via shaping. A policy π' in a reconfigured environment, described by E', satisfies the slack δ_A if

$$V_P^*(\tilde{s}_0|E_0) - V_P^{\pi'}(\tilde{s}_0|E') \leq \delta_A.$$

The designer first estimates the penalty for NSE associated with agent behavior, denoted by $R_N(\pi, E)$. If the NSE exceeds the designer's tolerance threshold, δ_D, then the environment is reconfigured, assuming agent's policy π is fixed. Given π and a set of valid modifications Ω, the designer selects a modification that maximizes its utility:

$$\max_{\omega \in \Omega} U_\pi(\omega)$$
$$U_\pi(\omega) = \underbrace{\big(R_N(\pi, E) - R_N(\pi, \Psi(E, \omega))\big)}_{\text{reduction in NSE penalty}} - C(E, \omega). \tag{1}$$

Typically the designer is the user of the system and their preferences and tolerance for NSE is captured by the utility of a modification. The designer's objective is to balance the trade-off between minimizing the NSE and the cost of applying the modification. It is assumed that the cost of the modification is measured in the same units as the NSE penalty. The cost of a modification may be amortized over episodes of the actor performing the task in the environment. The following properties are used to guarantee bounded-performance of the actor when shaping the environment to minimize NSE.

Definition 7. *Shaping is* ***admissible*** *if it results in an environment configuration where (1) the actor can complete its assigned task, given δ_A and (2) the NSE does not increase, relative to E_0.*

Definition 8. *Shaping is* ***proper*** *if (1) it is admissible and (2) reduces the actor's NSE to be within δ_D.*

Definition 9. *Shaping is* ***robust*** *if it results in an environment configuration E where* all *valid policies of the actor for o_P, given δ_A, produce NSE within δ_D. That is, $R_N(\pi, E) \leq \delta_D$, $\forall \pi : V_P^*(\tilde{s}_0|E_0) - V_P^\pi(\tilde{s}_0|E) \leq \delta_A$.*

Shared Knowledge The actor and the designer do not have details about each other's model. Since the objectives are decoupled, knowledge about the exact model parameters of the other agent is not required. Instead, the two key details that are necessary to achieve collaboration are shared: configuration file describing the environment and the actor's policy. The shared configuration file, such as the map of the environment, allows the designer to effectively communicate the modifications to the actor. The actor's policy is required for the designer to shape the environment. However, compact representation of the actor's policy is a practical challenge in large problems (Guestrin et al., 2001). Therefore, instead of sharing the complete policy, the actor provides a finite number of demonstrations $\mathcal{D} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of its optimal policy for o_P in the current environment configuration. Each demonstration τ is a trajectory from start to goal, following π. We consider deterministic policies for the actor. Using $\mathcal{D}$, the designer can extract the actor's policy and measure its NSE. The designer is aware of the general capabilities of the agent and its objectives as they are assigning the task, which makes it easier for them to construe the agent's trajectories. Naturally, increasing the number and diversity of sample trajectories that cover the actor's reachable states helps the designer improve the accuracy of estimating the actor's NSE and select an appropriate modification. If $\mathcal{D}$ does not starve any reachable state, following π, then the designer can extract the actor's complete policy.

Algorithm 2 Environment shaping to mitigate NSE

Require: $\langle E_0, \mathcal{E}, \tilde{M}, M_d, \delta_A, \delta_D \rangle$: AD-NSE
Require: d: Number of sample trajectories
Require: b: Budget for evaluating modifications
$E^* \leftarrow E_0$ ▷ Initialize best configuration
$E \leftarrow E_0$
$n \leftarrow \infty$
$\mathcal{D} \leftarrow$ Solve $\tilde{M}$ for E_0 and sample d trajectories
$\bar{\Omega} \leftarrow$ Diverse_modifications(b, Ω, M_d, E_0)
while $|\bar{\Omega}| > 0$ **do**
  $\hat{\pi} \leftarrow$ Extract policy from $\mathcal{D}$
  if $R_N(\hat{\pi}, E) < n$ **then**
    $n \leftarrow R_N(\hat{\pi}, E)$
    $E^* \leftarrow E$
  if $R_N(\hat{\pi}, E) \leq \delta_D$ **then** break
  $\omega^* \leftarrow \arg\max_{\omega \in \bar{\Omega}} U_{\hat{\pi}}(\omega)$
  if $\omega^* = \emptyset$ **then** break
  $\bar{\Omega} \leftarrow \bar{\Omega} \setminus \omega^*$
  $\mathcal{D}' \leftarrow$ Solve $\tilde{M}$ for $\Psi(E_0, \omega^*)$ and sample d trajectories
  if $\mathcal{D}' \neq \{\}$ **then**
    $\mathcal{D} \leftarrow \mathcal{D}'$
    $E \leftarrow \Psi(E_0, \omega^*)$
return E^*

Shaping phase

6.2 Algorithm for Environment Shaping

Our solution approach for solving AD-NSE, described in Algorithm 2, proceeds in two phases: a planning phase and a shaping phase. In the planning phase (Line 4), the actor computes its policy π for o_P in the current environment configuration and generates a finite number of sample trajectories $\mathcal{D}$, following π. The planning phase ends with disclosing $\mathcal{D}$ to the designer. In the shaping phase (Lines 7-14), the designer first associates states with actions observed in $\mathcal{D}$ to extract a policy $\hat{\pi}$. The designer then estimates the corresponding NSE penalty, denoted by $R_N(\hat{\pi}, E)$. If the estimated NSE in the environment described by E is lower than the current best estimate of NSE, denoted by n, then the best environment configuration E^* and n are updated with E and $R_N(\hat{\pi}, E)$ respectively (Lines 8-10). If the NSE exceeds the designer's tolerance threshold, $R_N(\hat{\pi}, E) > \delta_D$, then a utility-maximizing modification is applied and the configuration file is updated to reflect the modifications (Lines 12-15). The designer may be willing to evaluate only a few diverse modifications when the space of modifications is large (Line 5), which is described in detail in Algorithm 3.

The actor returns $\mathcal{D} = \{\}$ when the modification affects its ability to reach the goal, given δ_A (Line 15). Modifications are applied tentatively for evaluation and the environment is reset if the actor returns $\mathcal{D} = \{\}$ or if the reconfiguration does not minimize the NSE. Therefore, all modifications are applied to E_0 and it suffices to test each ω without replacement, as the actor always calculates the corresponding optimal policy. When $\tilde{M}$ is solved approximately, without bounded-guarantees, it is non-trivial to verify if δ_A is violated but the designer selects a utility-maximizing ω corresponding to this policy.

The planning and shaping phases alternate until the algorithm terminates. Algorithm 2 terminates when at least one of the following conditions is satisfied: (1) the NSE is within δ_D (Line 8); (2) all $\omega \in \Omega$ have been tested (Line 6); or (3) the utility-maximizing option does not make any modification at all, $\omega^* = \emptyset$ (Line 13). The first condition is straightforward. The second criteria describes the worst case scenario when no modification reduces the NSE to be within δ_D. In this case, our algorithm identifies an ω that results in an environment configuration with the least NSE possible. When $\omega^* = \emptyset$, it indicates that cost of modification exceeds the reduction in NSE or no modification can reduce the NSE further.

Though the designer calculates the utility for all modifications, there is an implicit pruning in the shaping phase since only utility-maximizing modifications are evaluated (Line 15). However, when multiple modifications have the same cost and produce similar environment configurations, the algorithm will alternate between planning and shaping multiple times to evaluate all these modifications, which is $|\Omega|$ in the worst case. To minimize the number of evaluations in settings with large Ω, consisting of multiple similar modifications, we present a greedy approach to identify and evaluate diverse modifications.

Selecting diverse modifications Let $b > 0$ denote the maximum number of modifications the designer is willing to evaluate. When the budget $b < |\Omega|$, it is beneficial to evaluate b diverse modifications. Algorithm 3 presents a greedy approach to select b diverse modifications. If two modifications result in similar environment configurations, the algorithm prunes the modification with the higher cost. The similarity threshold is controlled by ϵ. This process is repeated until b modifications are identified. Measures such as the Jaccard distance or embeddings may be used to estimate the similarity between two configurations.

Algorithm 3 Diverse_modifications(b, Ω, M_d, E_0)

```
1: Ω̄ ← Ω
2: if b ≥ |Ω| then return Ω
3: for each ω1, ω2 ∈ Ω̄ do
4:     if similarity(ω1, ω2) ≤ ϵ then
5:         Ω̄ = Ω̄ \ arg max_{ω1,ω2}(C(E0, ω1), C(E0, ω2))
6:         if |Ω̄| = b then return Ω̄
```

Remark. *Shaping with Algorithm 2 is guaranteed to be admissible but may not be proper.*

Algorithm 2 ensures that a modification does not negatively impact the actor's ability to complete its task, given δ_A (Lines 16-19), and stores the configuration with the least NSE (Line 10). Therefore, Algorithm 2 is guaranteed to return an E^* with NSE equal to or lower than that of the initial configuration E_0, thereby guaranteeing admissible shaping.

Shaping with Algorithm 2 is not guaranteed to be proper because (1) no modifications in Ω may be able to reduce the NSE penalty below δ_D; or (2) the cost of such a modification may exceed the corresponding reduction in NSE.

6.2.1 Policy-Preserving Modification

With each reconfiguration, the actor is required to recompute its policy. We now show that there exists a class of problems for which the actor's policy is unaffected by shaping.

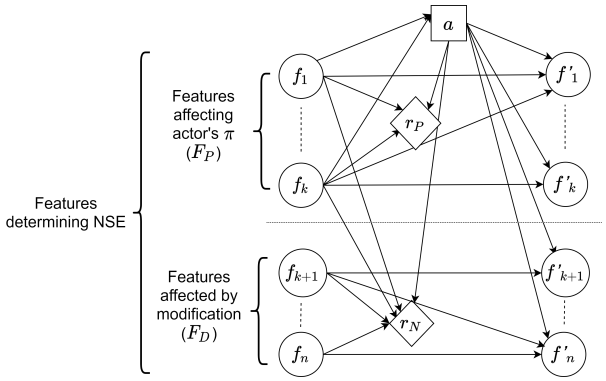


Figure 11: A dynamic Bayesian network description of a policy-preserving modification, where r_P denotes the reward associated with o_P and r_N denotes the NSE penalty.

Definition 10. *A modification is* ***policy-preserving*** *if the actor's initial policy is unaltered by environment shaping.*

This property induces policy invariance before and after shaping and hence the policy is backward-compatible. Additionally, the actor is guaranteed to complete its task with $\delta_A = 0$. Figure 11 illustrates the dynamic Bayesian network for this class of problems.

Let F denote the set of features in the environment, with $F_P \subset F$ affecting the actor's policy and $F_D \subset F$ altered by the designer's modifications. Let $\vec{f}$ and $\vec{f'}$ be the feature

values before and after environment shaping. Given the actor's π, a policy-preserving ω follows $F_D = F \setminus F_P$, ensuring $\vec{f_P} = \vec{f'_P}$. When $\vec{f} = \vec{f_P} \cup \vec{f_D}$, a policy-preserving ω mitigates NSE by enforcing $Pr(F_D = \vec{f_D} | \vec{f_P}, \pi) = 0, \forall \vec{f}$. For example in the boxpushing domain, regions in the agent's shortest path to its goal cannot be covered by a rug.

Proposition 2. *Given an environment configuration E and actor's policy π, a modification $\omega \in \Omega$ is guaranteed to be* ***policy-preserving*** *if $F_D = F \setminus F_P$.*

Proof. We prove by contradiction. Let $\omega \in \Omega$ be a modification consistent with $F_D \cap F_P = \emptyset$ and $F = F_P \cup F_D$. Let π and π' denote the actor's policy before and after shaping the environment with ω such that $\pi \neq \pi'$. Since the actor always computes an optimal policy for o_P (assuming fixed strategy for tie-breaking), a difference in the policy indicates that at least one feature in F_P is affected by ω, $\vec{f_P} \neq \vec{f'_P}$. This is a contradiction since $F_D \cap F_P = \emptyset$. Therefore $\vec{f_P} = \vec{f'_P}$ and $\pi = \pi'$. Thus, $\omega \in \Omega$ is policy-preserving when $F_D = F \setminus F_P$. □

Corollary 1. *A policy-preserving modification identified by Algorithm 2 guarantees* admissible *shaping.*

Furthermore, a policy-preserving modification results in robust shaping when $Pr(F_D = \vec{f_D}) = 0$, for all $\vec{f} = \vec{f_P} \cup \vec{f_D}$ that lead to NSE greater than δ_D. The policy-preserving property is sensitive to the actor's state representation. However, many real-world problems exhibit this feature independence. In the boxpushing example (Figure 1), the actor's state representation may not include details about the rug since it is not relevant to the task. Moving the rug to a different part of the room that is not on the actor's path to the goal is an example of a policy-preserving and admissible shaping. Modifications such as removing the rug or covering it with a protective sheet are policy-preserving and robust shaping since there is no direct contact between the box and the rug, for all policies of the actor.

6.2.2 Relation to Game Theory

Our solution approach with decoupled objectives can be viewed as a game between the actor and the designer. The action profile or the set of strategies for the actor is the policy space Π, with respect to o_P and $\mathcal{E}$, with payoffs defined by its reward function R. The action profiles for the designer is the set of all modifications Ω with payoffs defined by its utility function U. In each round of the game, the designer selects a modification that is a best response to the actor's policy π:

$$b_D(\pi) = \{\omega \in \Omega | \forall \omega' \in \Omega, U_\pi(\omega) \geq U_\pi(\omega')\}. \tag{2}$$

The actor's best response is its optimal policy for o_P in the current environment configuration, given δ_A:

$$b_A(E) = \{\pi \in \Pi | \forall \pi' \in \Pi, V_P^\pi(\tilde{s}_0) \geq V_P^{\pi'}(\tilde{s}_0) \ \wedge V_P^{\pi_0}(\tilde{s}_0) - V_P^\pi(\tilde{s}_0) \leq \delta_A\}, \tag{3}$$

where π_0 denotes the optimal policy in E_0 before initiating environment shaping. These best responses are pure strategies since there exists a deterministic optimal policy for a discrete MDP and a modification is selected deterministically. Proposition 3 shows that AD-NSE is an extensive-form game, which is useful to understand the link between two often distinct fields of decision theory and game theory. This also opens the possibility for future work on game-theoretic approaches for mitigating NSE.

Proposition 3. *An AD-NSE with policy constraints induced by Equations 2 and 3 induces an equivalent extensive form game with incomplete information, denoted by $\mathcal{N}$.*

Proof Sketch. Our solution approach alternates between the planning phase and the shaping phase. This induces an extensive form game $\mathcal{N}$ with strategy profiles Ω for the designer and Π for the actor, given start state $\tilde{s}_0$. However, each agent selects a best response based on the information available to it and its payoff matrix, and is unaware of the strategies and payoffs of the other agent. The designer is unaware of how its modifications may impact the actor's policy until the actor recomputes its policy and the actor is unaware of the NSE of its actions. Hence this is an extensive form game with incomplete information. □

6.3 Experimental Setup

The effectiveness of shaping is studied in settings with avoidable and unavoidable NSE.

Baselines The performance of shaping with budget is compared with the following baselines. First is the *Initial* approach in which the actor's policy is naively executed and does not involve shaping or any form of learning to mitigate NSE. This is similar to the *No queries* approach considered in Section 4.5. Second is the *shaping* with exhaustive search to select a modification, $b = |\Omega|$. Third is the *Feedback* approach in which the agent performs a trajectory of its optimal policy for o_P and the human approves or disapproves the observed actions based on the NSE occurrence. The agent then disables all the disapproved actions and recomputes a policy for execution. If the updated policy violates the slack, the actor ignores the feedback and executes its initial policy since o_P is prioritized. Using the human feedback as training data, the agent learns a predictive model of NSE.

We use Jaccard distance to measure the similarity between environment configurations. A random forest classifier from the `sklearn` Python package is used for learning a predictive model. The actor's MDP is solved using value iteration. The algorithms are implemented in Python and tested on a computer with 16GB of RAM. We optimize action costs, which are negation of rewards. Values averaged over 100 trials of planning and execution, along with the standard errors, are reported for the following domains.

Boxpushing We consider a modified boxpushing domain in which the actor is required to minimize the expected time taken to push a box to the goal location. Each action takes +1 unit of time. The actions succeed with probability 0.9 and may slide right with probability 0.1. Each state is represented as $\langle x, y, b\rangle$ where x, y denote the agent's location and b is a boolean variable indicating if the agent is pushing the box. Pushing the box over the rug or knocking over a vase on its way results in NSE, incurring a penalty of 5. The designers are the system users. We experiment with grid size 15×15. We consider 24 modifications for the boxpushing domain, such as adding a protective sheet over the rug, moving the vase to corners of the room, removing the rug, block access to the rug and vase area, among others. Removing the rug costs 0.4/unit area covered by the rug, moving the vase costs 1, and all other modifications cost 0.2/unit. Except for blocking access to the rug and vase area, all the other modifications are policy-preserving for the representation we consider.

Driving Our second domain is based on simulated autonomous driving, in which the actor's objective is to minimize the expected cost of navigation from start to a goal location,

during which it may encounter some potholes. Each state is the agent's location represented by $\langle x, y \rangle$. We consider a grid of size 25×15. The actor can move in all four directions at low and high speeds, with costs 2 and 1 respectively. Driving fast through shallow potholes results in a bumpy ride for the user, which is a mild NSE with a penalty of 2. Driving fast through a deep pothole may damage the car, in addition to the unpleasant experience for the rider, and therefore it is a severe NSE with a penalty of 5. The cost of modifications are amortized over multiple episodes of agent operation in the environment. The modifications considered are: disabling 'move fast' action for the actor in zone i, $1 \leq i \leq 4$; disabling 'move fast' action always; disabling 'move fast' action in zones with shallow potholes; fill all potholes; fill deep potholes; and fill deep potholes in zone i, $1 \leq i \leq 4$. Disabling 'move fast' is achieved by reducing the speed limit, and costs 0.5 units per pothole in that zone. Filling a pothole costs two units, and it is a policy-preserving modification.

6.4 Results and Discussion

Effect of slack We vary δ_A and δ_D, and plot the resulting NSE penalty in Figure 12. We report results on the driving domain for shaping based on 100 actor trajectories. We vary δ_A between 0-25% of $V^*_P(\tilde{s}_0|E_0)$ and δ_D between 0-25% of the NSE penalty of the actor's policy in E_0. Our results show that increasing the slack helps reduce the NSE, as expected. In particular, when $\delta_D \geq 15\%$, the NSE penalty is considerably reduced with $\delta_A = 15\%$. Overall, increasing δ_A is most effective in reducing the NSE. We also tested the effect of slack on the cost for o_P. The results showed that the cost was predominantly affected by δ_A. We observed similar performance for all values of δ_D for a fixed δ_A and therefore do not include that plot. This is an expected behavior since o_P is prioritized over minimizing NSE in our setting.

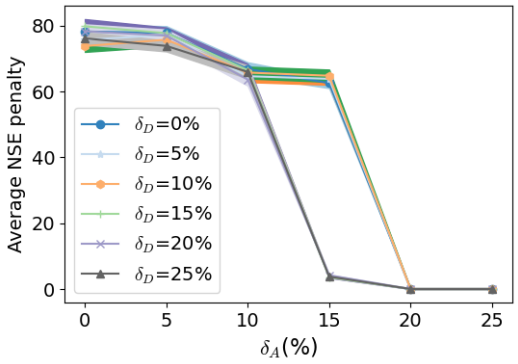
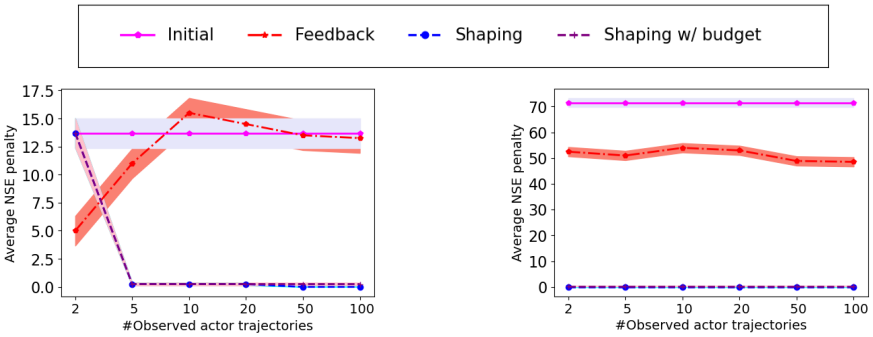


Figure 12: Effect of δ_A and δ_D on the average NSE penalty.

Effectiveness of shaping The effectiveness of shaping is evaluated in terms of the average NSE penalty incurred and the expected value of o_P after shaping. Figure 13 plots the results for the boxpushing domain with $\delta_A = 0$, $\delta_D = 0$, and $b = 3$, as the number of observed actor trajectories is increased. We do not report results for o_P since we consider $\delta_A = 0$ and therefore the agent always optimizes o_P. Feedback budget is 500. With at most five trajectories, the designer is able to select a *policy-preserving* modification that avoids



(a) Avoidable NSE

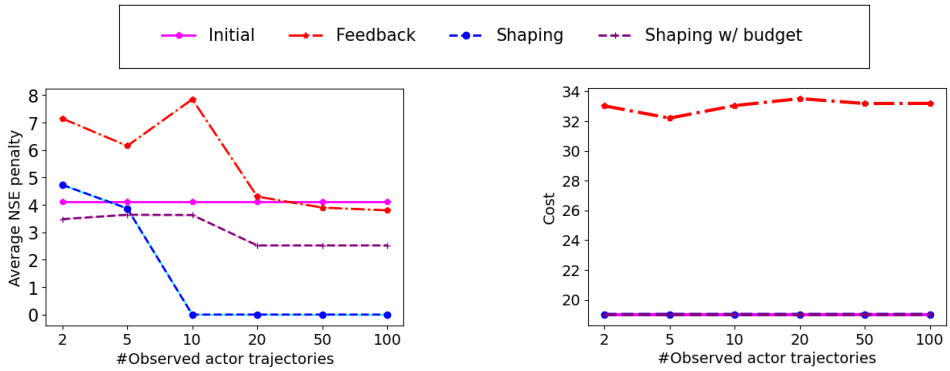
(b) Unavoidable NSE

#Trajectories oberved before reconfiguring	Shaping	Shaping w/ budget
2	1	1
5	2	1
10	2	1
20	2	1
50	6	3
100	6	3

(c) Number of modifications evaluated when NSE is avoidable.

Figure 13: Results on boxpushing domain.

NSE. Shaping w/ budget $b = 3$ performs similar to shaping by evaluating all modifications, and reduces the number of shaping evaluations by 50%. The feedback approach is not able to mitigate NSE since it violates the actor's slack and the agent state representation has limited fidelity, similar to the results in Section 5.



(a) Average NSE penalty.

(b) Expected cost of o_P.

Figure 14: Results on driving domain with avoidable NSE.

Figure 14 plots the results on the driving domain with avoidable NSE, $\delta_A = 25\%$, $\delta_D = 0$, and $b = 4$. We choose $\delta_A = 25\%$ since the agent cannot avoid NSE unless it slows down or chooses an alternate longer route, which requires a non-zero slack, and Figure 12 shows that

NSE are avoided in the driving domain when $\delta_A = 25\%$, irrespective of δ_D. The driving domain does not have unavoidable NSE since the agent can always avoid NSE by slowing down over potholes. Shaping with budget reduces NSE as more trajectories are observed but is unable to avoid NSE. However, shaping by evaluating all modifications avoids NSE by observing ten actor trajectories. The feedback approach is unable to mitigate NSE due to the limited state representation of the agent. In both domains, we observe that shaping avoids NSE, after observing few actor trajectories, and shaping w/ budget mitigates NSE considerably.

7. Limitations of Environment Shaping

While our results show that environment shaping can successfully mitigate both avoidable and unavoidable NSE, the following factors may affect the performance.

Challenges in performing shaping Environment shaping requires the designer, typically the user, to supervise the agent and have control over the environment to perform shaping. In general, certain reconfigurations may require specific skills. Our approach, however, does not model the designer's competency and instead assumes that the set of available modifications is optimized for the designer's skills. Access to a model of the designer's competency is required to automatically identify useful modifications that the designer can implement. Our formulation assumes that the designer can predict the actor's policy from trajectories to perform shaping effectively. In practice, the designer requires insights about agent behavior to be able to predict agent policy with high confidence.

While our algorithm identifies an appropriate modification, the designer may be required to evaluate multiple reconfigurations to effectively avoid NSE when there is a large set of diverse modifications, or when each reconfiguration involves a long sequence of design actions. This can be time-consuming for the user. Though shaping w/ budget helps alleviate this concern, it introduces a trade-off between number of modifications to evaluate and the final NSE. A lower budget reduces the number of modifications to evaluate, but it is not guaranteed to find an optimal, utility-maximizing reconfiguration. Another approach to quickly identify right modifications is to generalize the observed performance across modifications. However, it is non-trivial to generalize the effect of shaping on agent performance and the resulting NSE, since we do not assume knowledge about the agent's model.

Our framework requires the agent to recompute its policy every time a modification is being evaluated, which can be computationally expensive if the agent operates based on a large, complex model. Though policy-preserving modifications overcome this limitation, it is challenging to determine whether a given modification is policy-preserving a priori since it depends on the agent's model.

Interference with future tasks and other agents Reconfigurations can be temporary or permanent. Permanent modifications may affect the agent's ability to perform future tasks in the environment or even introduce new NSE, especially when the agent performs different types of tasks. In this work, we consider settings where the agent repeatedly performs the same task, and therefore this issue does not arise. It is not straightforward to extend our algorithm to handle situations where an agent performs different types of task, since this requires knowledge about the agent's behavior and estimating slack for all

the tasks performed by the agent. Further, there may be multiple agents, with different tasks and capabilities, operating simultaneously in an environment. In such settings, even temporary modifications may affect the performance of other agents. However, our approach currently does not account for these types of interference when identifying an appropriate reconfiguration to mitigate NSE.

8. Summary and Future Work

Autonomous agents may create negative side effects when executing a policy computed using an incomplete model of the environment. It is generally infeasible to model all negative side effects a priori, since the occurrence of side effects depends on the deployed environment and the tolerance to a side effect is user-specific. In situations where the agent has no prior knowledge about the side effects of its actions, the user can assist the system in mitigating the impacts of NSE after deployment. The user assistance can be in the form of feedback or environment reconfiguration. The agent may also be able to learn about NSE by exploring the environment to gather signals corresponding to NSE penalty.

We formalize the problem of mitigating NSE and present two solution approaches that are complementary in their assumptions and agent responsibilities. First, we investigate the effectiveness of various forms of feedback in learning a model of NSE. The problem is formulated as a multi-objective problem with slack, and we propose an algorithm to determine the minimum slack required to avoid these side effects. Second, we leverage human assistance in the form of environment shaping. The problem is formulated as a human-agent collaboration with decoupled objectives and present an algorithm for shaping. Empirical evaluations show that our approaches are effective in mitigating avoidable and unavoidable NSE. We also discuss the limitations of each approach.

There are many interesting directions for future research. In this work, we target side effects that are undesirable but not safety-critical. The proposed lexicographic ordering for learning from feedback may have to be modified when the side effects are safety-critical because the agent will be expected to prioritize minimizing NSE to maximize safety, over optimizing the performance of its task. In addition, it is assumed that the user can over ride the slack value, if the slack computed using the proposed approach violated user tolerance. In practice, choosing the "right" slack value for a problem can be challenging since it requires a deep understanding about the trade-offs between the objectives. One approach to guide the user in picking the slack value is to modify our algorithm such that it presents solutions with different slack values within a range, instead of outputting a single slack value. This would allow the user to compare the effect of different slack values and select a value that best matches their tolerance. Extending learning from feedback approach to NSE associated with trajectories that are not decomposable into additive penalties of state-action pairs is another interesting direction for the future.

Our solution approaches rely on immediate human assistance, either in the form of feedback or shaping the environment. However, humans are bounded-rational, are slow to respond, and are not guaranteed to provide accurate feedback or identify optimal reconfigurations. In the future, we aim to make our approaches robust to errors in human assistance. Different feedback approaches have different drawbacks, which we describe in Section 5. In the future, we aim to develop techniques to learn using a combination of dif-

ferent feedback approaches. In our current formulation of environment shaping, the space of available modifications for an environment is assumed to be given. In the future, we aim to develop methods to automatically identify useful modifications for a given problem. We also aim to develop techniques to automatically switch between solution methods, depending on various factors such as agent capabilities, type of NSE, and the human's preferred form of assistance.

In this work, we target side effects that affect the safety and reliability of agent operation. In the future, we plan to expand the scope to address other forms of side effects such as amplifying underlying biases or increasing vulnerability to attacks, which may occur when the system optimizes incompletely specified objectives.

Acknowledgments

We thank the reviewers for their feedback. This work performed while Sandhya was a student at the University of Massachusetts Amherst. Support for this work was provided in part by the Semiconductor Research Corporation under grant #2906.001.

References

Agashe, N., & Chapman, S. (2019). Traffic signs in the evolving world of autonomous vehicles. Tech. rep., Avery Dennison.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. *CoRR*, *abs/1606.06565*.

Dietterich, T. G. (2017). Steps toward robust artificial intelligence. *AI Magazine*, *38*(3), 3–24.

Guestrin, C., Koller, D., & Parr, R. (2001). Max-norm projections for factored MDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pp. 673–682.

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., & Dragan, A. (2017). Inverse reward design. In *Advances in Neural Information Processing Systems*, pp. 6765–6774.

Hansen, E. A., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*, 35–62.

Hendrickson, C., Biehler, A., & Mashayekh, Y. (2014). Connected and autonomous vehicles 2040 vision. Tech. rep., Pennsylvania Department of Transportation.

Hibbard, B. (2012). Avoiding unintended AI behaviors. In *International Conference on Artificial General Intelligence*, pp. 107–116. Springer.

Keren, S., Gal, A., & Karpas, E. (2019). Goal recognition design in deterministic environments. *Journal of Artificial Intelligence Research*, *65*, 209–269.

Keren, S., Pineda, L., Gal, A., Karpas, E., & Zilberstein, S. (2017). Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 4353–4360.

Krakovna, V., Orseau, L., Martic, M., & Legg, S. (2019). Penalizing side effects using stepwise relative reachability. In *AI Safety Workshop, IJCAI*.

Krakovna, V., Orseau, L., Ngo, R., Martic, M., & Legg, S. (2020). Avoiding side effects by considering future tasks. In *Advances in Neural Information Processing Systems*.

Kulkarni, A., Sreedharan, S., Keren, S., Chakraborti, T., Smith, D. E., & Kambhampati, S. (2020). Designing environments conducive to interpretable robot behavior. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 10982–10989.

Lakkaraju, H., Kamar, E., Caruana, R., & Horvitz, E. (2017). Identifying unknown unknowns in the open world: Representations and policies for guided exploration. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 2124–2132.

Lindner, D., Matoba, K., & Meulemans, A. (2021). Challenges for using impact regularizers to avoid negative side effects. *CoRR*, *abs/2101.12509*.

McFarland, M. (2020). Michigan plans to redesign a stretch of road for self-driving cars. `https://www.cnn.com/2020/08/13/cars/michigan-self-driving-road/index.html`.

Ramakrishnan, R., Kamar, E., Dey, D., Horvitz, E., & Shah, J. (2020). Blind spot detection for safe sim-to-real transfer. *Journal of Artificial Intelligence Research*, *67*, 191–234.

Ramakrishnan, R., Kamar, E., Nushi, B., Dey, D., Shah, J., & Horvitz, E. (2019). Overcoming blind spots in the real world: Leveraging complementary abilities for joint execution. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 6137–6145.

Randløv, J. (2000). Shaping in reinforcement learning by changing the physics of the problem. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 767–774.

Roijers, D. M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, *48*, 67–113.

Russell, S., Dewey, D., & Tegmark, M. (2015). Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, *36*(4), 105–114.

Saisubramanian, S., Kamar, E., & Zilberstein, S. (2020a). Mitigating the negative side effects of reasoning with imperfect models: A multi-objective approach. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1984–1986.

Saisubramanian, S., Kamar, E., & Zilberstein, S. (2020b). A multi-objective approach to mitigate negative side effects. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pp. 354–361.

Saisubramanian, S., Roberts, S. C., & Zilberstein, S. (2021). Understanding user attitudes towards negative side effects of AI systems. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pp. 368:1–368:6.

Saisubramanian, S., Wray, K., Pineda, L., & Zilberstein, S. (2019). Planning in stochastic environments with goal uncertainty. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1649–1654.

Saisubramanian, S., & Zilberstein, S. (2021). Mitigating negative side effects via environment shaping. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1640–1642.

Saisubramanian, S., Zilberstein, S., & Kamar, E. (2022). Avoiding negative side effects due to incomplete knowledge of AI systems. *AI Magazine*, *42*(4), 62–71.

Saria, S., & Subbaswamy, A. (2019). Tutorial: Safe and reliable machine learning. *CoRR*, *abs/1904.07204*.

Seuken, S., & Zilberstein, S. (2007). Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pp. 344–351.

Shah, R., Krasheninnikov, D., Alexander, J., Abbeel, P., & Dragan, A. (2019). Preferences implicit in the state of the world. In *Proceedings of the 7th International Conference on Learning Representations*.

Simon, M. (2019). Inside the amazon warehouse where humans and machines become one. `https://www.wired.com/story/amazon-warehouse-robots/`.

Thomas, P. S., da Silva, B. C., Barto, A. G., Giguere, S., Brun, Y., & Brunskill, E. (2019). Preventing undesirable behavior of intelligent machines. *Science*, *366*(6468), 999–1004.

Turner, A., Ratzlaff, N., & Tadepalli, P. (2020a). Avoiding side effects in complex environments. In *Advances in Neural Information Processing Systems*.

Turner, A. M., Hadfield-Menell, D., & Tadepalli, P. (2020b). Conservative agency via attainable utility preservation. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 385–391.

Wray, K. H., Zilberstein, S., & Mouaddib, A.-I. (2015). Multi-objective MDPs with conditional lexicographic reward preferences. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, pp. 3418–3424.

Zhang, S., Durfee, E. H., & Singh, S. (2020). Querying to find a safe policy under uncertain safety constraints in Markov decision processes. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pp. 2552–2559.

Zhang, S., Durfee, E. H., & Singh, S. P. (2018). Minimax-regret on side effects for safe optimality in factored Markov decision processes. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4867–4873.